Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение   
высшего образования «Национальный исследовательский университет   
«Московский институт электронной техники»

Кафедра «Информационная безопасность»

ОТЧЕТ

по проведению лабораторной работы

ЛАБОРАТОРНАЯ РАБОТА №ТЕСТ

«Устновка и настройка bWAPP»

|  |  |
| --- | --- |
| Выполнил студент учебной группы ИБ- | \_\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_\_\_\_\_/  подпись (фамилия, инициалы) |
|  |
| Проверил | \_\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_\_\_\_\_/  подпись (фамилия, инициалы) |
|  |

Москва, 2024 г.

СОДЕРЖАНИЕ

[ЦЕЛИ РАБОТЫ 3](#_Toc182315616)

[ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 3](#_Toc182315617)

[1 Определение SQLi 3](#_Toc182315618)

[2 Обнаружение и эксплуатация уязвимостей 4](#_Toc182315619)

[2.1 Извлечение скрытых данных 5](#_Toc182315620)

[2.2 Извлечение данных из других таблиц базы данных 6](#_Toc182315621)

[2.3 «Слепые» уязвимости SQL-инъекций 6](#_Toc182315622)

[2.4 SQL-инъекции второго порядка 7](#_Toc182315623)

[2.5 Изучение базы данных 8](#_Toc182315624)

[2.6 Справочник по SQL-инъекциям 9](#_Toc182315625)

[2.7 SQL-инъекция с использованием UNION 11](#_Toc182315626)

[ПРАКТИЧЕСКАЯ ЧАСТЬ 13](#_Toc182315627)

[1 Установка Docker на Kali Linux 14](#_Toc182315628)

[2 Установка и запуск bWAPP 17](#_Toc182315629)

[3 Решение задач 20](#_Toc182315630)

[3.1 SQL Injection (Login Form/Hero) 20](#_Toc182315631)

[3.2 SQL Injection (GET/Select) 23](#_Toc182315632)

# ЦЕЛИ РАБОТЫ

1. Проверить эффективность и результативность самостоятельной работы студентов над учебным материалом.

**Время на выполнение:** 4 учебных часа (180 минут).

# ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

## 1 Определение SQLi

SQLi (SQL injection) – уязвимость веб приложений, которая позволяет злоумышленнику воздействовать на базу данных сервиса, используя инъекцию в SQL скрипты. Атакующий может получить доступ к данным, которые не может получить в обычных условиях. Среди них могут быть данные о других пользователях или другая информация о приложении. Также злоумышленник получает доступ к изменению и удалению записей в таблицах базы данных.

SQL-инъекция может быть использована злоумышленниками для эскалации атак и компрометации серверов или другой бэкэнд-инфраструктуры, а также может позволить им осуществлять атаки типа «отказ в обслуживании» (DoS).

Успешная SQL-инъекция может привести к несанкционированному доступу к чувствительным данным, таким как:

Пароли.

Данные кредитных карт.

Личная информация пользователей.

SQL-инъекции сыграли значительную роль во многих крупных утечках данных, что привело к репутационным потерям и штрафам за нарушение норм регулирования. В некоторых случаях злоумышленники могут получить долгосрочный доступ к системам, который может оставаться незамеченным в течение продолжительного времени.

## 2 Обнаружение и эксплуатация уязвимостей

Для ручного обнаружения SQL-инъекций необходимо систематически тестировать каждую точку ввода данных в приложении, отправляя:

Одинарную кавычку ' и анализируя ошибки или аномалии в откликах.

SQL-синтаксис, который возвращает исходное значение точки ввода, а затем иное значение, для выявления различий в ответах приложения.

Булевы условия, например, OR 1=1 и OR 1=2, и анализируя различия в откликах приложения.

Нагрузки, вызывающие временные задержки при исполнении в SQL-запросе, и измеряя время отклика.

OAST-нагрузки для инициирования сетевого взаимодействия вне полосы (out-of-band), при исполнении в SQL-запросе, и наблюдая за такими взаимодействиями.

Также для быстрого и надежного обнаружения большинства уязвимостей SQL-инъекций можно использовать Burp Scanner.

Большинство уязвимостей SQL-инъекций возникает в предложении WHERE запроса SELECT. Однако такие уязвимости могут проявляться в любом месте запроса и в различных типах запросов. Некоторые распространенные случаи включают:

UPDATE-запросы с уязвимостями в обновляемых значениях или предложении WHERE.

INSERT-запросы с уязвимостями в вставляемых значениях.

SELECT-запросы с уязвимостями в именах таблиц или столбцов.

SELECT-запросы с уязвимостями в предложении ORDER BY.

Примеры SQL-инъекций

Существует множество уязвимостей и методов атак с использованием SQL-инъекций. Основные примеры включают:

Извлечение скрытых данных путем изменения запроса для возврата дополнительных результатов.

Нарушение логики приложения, когда запрос меняется для вмешательства в логику приложения.

Атаки UNION, позволяющие извлекать данные из других таблиц базы данных.

«Слепые» SQL-инъекции, где результаты контролируемого запроса не отображаются в откликах приложения.

### ****2.1 Извлечение скрытых данных****

Предположим, что в интернет-магазине отображаются товары по категориям. При выборе категории «Подарки» браузер пользователя отправляет запрос:

<https://insecure-website.com/products?category=Gifts>

Это вызывает SQL-запрос:

SELECT \* FROM products WHERE category = 'Gifts' AND released = 1

Этот запрос возвращает все товары из таблицы products, где category = 'Gifts' и released = 1. Ограничение released = 1 скрывает товары, которые еще не выпущены. Однако отсутствие защиты от SQL-инъекций позволяет злоумышленнику выполнить атаку:

<https://insecure-website.com/products?category=Gifts'-->

Этот запрос преобразуется в:

SELECT \* FROM products WHERE category = 'Gifts'--' AND released = 1

Здесь -- указывает, что остальная часть запроса рассматривается как комментарий. Таким образом, условие AND released = 1 игнорируется, и отображаются все товары, включая еще не выпущенные.

Похожий запрос может быть использован для отображения всех товаров в любой категории:

<https://insecure-website.com/products?category=Gifts'+OR+1=1-->

Этот запрос преобразуется в:

SELECT \* FROM products WHERE category = 'Gifts' OR 1=1--' AND released = 1

Так как 1=1 всегда истинно, запрос возвращает все товары.

### 2.2 Извлечение данных из других таблиц базы данных

Когда приложение возвращает результаты SQL-запроса, злоумышленник может использовать уязвимость SQL-инъекции для извлечения данных из других таблиц базы данных. Использование ключевого слова UNION позволяет выполнить дополнительный SELECT-запрос и добавить его результаты к исходному запросу.

Например, если приложение выполняет следующий запрос с пользовательским вводом «Gifts»:

SELECT name, description FROM products WHERE category = 'Gifts'

Злоумышленник может ввести:

' UNION SELECT username, password FROM users—

Это приведет к тому, что приложение вернет все имена пользователей и пароли наряду с названиями и описаниями продуктов.

### 2.3 «Слепые» уязвимости SQL-инъекций

Многие случаи SQL-инъекций являются «слепыми» уязвимостями. Это означает, что приложение не возвращает результаты SQL-запроса или подробности об ошибках базы данных в своих откликах. «Слепые» уязвимости по-прежнему можно использовать для доступа к несанкционированным данным, но такие методы сложнее и требуют более детального подхода.

Техники, используемые для эксплуатации «слепых» уязвимостей, зависят от их характера и базы данных:

Логика запроса может быть изменена так, чтобы вызвала различие в отклике приложения в зависимости от истинности условия. Это может включать инъекцию нового условия в логическое выражение или условное вызывание ошибки, например деление на ноль.

Можно использовать условные временные задержки при обработке запроса. Это позволяет сделать вывод о правдивости условия, исходя из времени отклика приложения.

Возможен вызов сетевого взаимодействия вне полосы (out-of-band) с использованием техник OAST. Это мощный метод, который подходит, когда другие способы не работают. Злоумышленник может напрямую извлечь данные через данный канал, например, вставляя данные в запросы DNS для контролируемого домена.

### 2.4 SQL-инъекции второго порядка

SQL-инъекция первого порядка происходит, когда приложение небезопасно обрабатывает пользовательский ввод из HTTP-запроса, включая его в SQL-запрос. SQL-инъекция второго порядка происходит, когда приложение принимает пользовательский ввод из HTTP-запроса и сохраняет его для будущего использования, например, помещая в базу данных. Уязвимость проявляется не на этапе сохранения данных, а позже, когда приложение небезопасно обрабатывает сохраненные данные в новом SQL-запросе. Поэтому SQL-инъекции второго порядка также известны как хранимые SQL-инъекции.

Зачастую такие уязвимости появляются, когда разработчики знают о SQL-инъекциях и безопасно обрабатывают первоначальное сохранение данных в базе. Однако при последующей обработке сохраненные данные рассматриваются как безопасные и могут быть использованы в небезопасной манере.

### 2.5 Изучение базы данных

Некоторые базовые функции языка SQL реализованы одинаково на популярных платформах баз данных, что позволяет применять одни и те же методы обнаружения и эксплуатации SQL-инъекций для различных типов баз данных.

Однако есть и различия между типами баз данных, которые влияют на методы работы с SQL-инъекциями. К таким особенностям относятся:

Синтаксис объединения строк.

Комментарии.

Пакетные (или стековые) запросы.

Специфичные для платформы API.

Сообщения об ошибках.

После идентификации уязвимости SQL-инъекции полезно собрать информацию о базе данных. Это может помочь в дальнейшем использовании уязвимости.

Вы можете запросить версию базы данных. Различные методы работают для различных типов баз данных. Это позволяет определить тип базы данных, если найденный метод работает. Например, для Oracle можно выполнить запрос:

SELECT \* FROM v$version

Вы также можете определить существующие таблицы в базе данных и столбцы в этих таблицах. Например, на большинстве баз данных можно выполнить запрос для отображения таблиц:

SELECT \* FROM information\_schema.tables

### 2.6 Справочник по SQL-инъекциям

Конкатенация строк.

* **Oracle**: 'foo'||'bar'
* **Microsoft**: 'foo'+'bar'
* **PostgreSQL**: 'foo'||'bar'
* **MySQL**: 'foo' 'bar' или CONCAT('foo','bar')

Извлечение подстроки.

Позволяет извлечь часть строки с указанного смещения и длины (смещение 1-based).

* **Oracle**: SUBSTR('foobar', 4, 2)
* **Microsoft**: SUBSTRING('foobar', 4, 2)
* **PostgreSQL**: SUBSTRING('foobar', 4, 2)
* **MySQL**: SUBSTRING('foobar', 4, 2)

Комментарии.

* **Oracle**: --comment
* **Microsoft**: --comment или /\*comment\*/
* **PostgreSQL**: --comment или /\*comment\*/
* **MySQL**: #comment, -- comment (с пробелом после двойного дефиса) или /\*comment\*/

Определение версии базы данных.

* **Oracle**: SELECT banner FROM v$version или SELECT version FROM v$instance
* **Microsoft**: SELECT @@version
* **PostgreSQL**: SELECT version()
* **MySQL**: SELECT @@version

Содержимое базы данных. Отобразить таблицы и столбцы базы данных.

**Oracle**:

SELECT \* FROM all\_tables;

SELECT \* FROM all\_tab\_columns WHERE table\_name = 'TABLE-NAME-HERE';

**Microsoft**, **PostgreSQL**, **MySQL**:

SELECT \* FROM information\_schema.tables;

SELECT \* FROM information\_schema.columns WHERE table\_name = 'TABLE-NAME-HERE';

Условные ошибки. Проверить условие и вызвать ошибку базы данных, если оно истинно.

**Oracle**:

SELECT CASE WHEN (YOUR-CONDITION-HERE) THEN TO\_CHAR(1/0) ELSE NULL END FROM dual;

**Microsoft**:

SELECT CASE WHEN (YOUR-CONDITION-HERE) THEN 1/0 ELSE NULL END;

**PostgreSQL**:

1 = (SELECT CASE WHEN (YOUR-CONDITION-HERE) THEN 1/(SELECT 0) ELSE NULL END);

**MySQL**:

SELECT IF(YOUR-CONDITION-HERE, (SELECT table\_name FROM information\_schema.tables), 'a');

Извлечение данных через видимые сообщения об ошибках. Некоторые запросы могут вызывать сообщения об ошибках, которые утекут чувствительные данные.

**Microsoft**:

SELECT 'foo' WHERE 1 = (SELECT 'secret');

**PostgreSQL**:

SELECT CAST((SELECT password FROM users LIMIT 1) AS int);

**MySQL**:

SELECT 'foo' WHERE 1=1 AND EXTRACTVALUE(1, CONCAT(0x5c, (SELECT 'secret')));

Пакетные (или стековые) запросы. Возможность выполнять несколько запросов последовательно.

**Microsoft, PostgreSQL, MySQL**:

QUERY-1-HERE; QUERY-2-HERE;

Задержка выполнения. Вы можете вызвать задержку обработки запроса, чтобы проверить уязвимость.

**Oracle**: dbms\_pipe.receive\_message(('a'),10)

**Microsoft**: WAITFOR DELAY '0:0:10'

**PostgreSQL**: SELECT pg\_sleep(10)

**MySQL**: SELECT SLEEP(10)

### 2.7 SQL-инъекция с использованием UNION

Когда приложение уязвимо к SQL-инъекциям, и результаты запроса отображаются в ответах приложения, злоумышленники могут использовать ключевое слово **UNION** для получения данных из других таблиц базы данных. Этот вид атаки известен как SQL-инъекция с использованием UNION.

Ключевое слово **UNION** позволяет выполнять один или несколько дополнительных **SELECT**-запросов и объединять их результаты с исходным запросом. Например:

SELECT a, b FROM table1 UNION SELECT c, d FROM table2

Этот запрос объединяет два набора результатов, выводя значения из колонок a и b таблицы table1 и из колонок c и d таблицы table2.

Условия успешного выполнения запроса с UNION:

Все объединяемые запросы должны возвращать одинаковое количество столбцов.

Типы данных в столбцах должны быть совместимы.

Чтобы выполнить атаку SQL-инъекции с использованием UNION, необходимо убедиться, что эти условия выполнены. Обычно это включает:

Определение количества столбцов, возвращаемых исходным запросом.

Поиск столбцов с подходящими типами данных, которые могут содержать результаты внедренного запроса.

Определение количества столбцов

Для атаки можно использовать два эффективных метода:

**Метод с использованием ORDER BY**. Внедряйте последовательность операторов **ORDER BY**, увеличивая индекс столбца до возникновения ошибки. Например, если точка инъекции находится в строке внутри условия **WHERE** исходного запроса, вводите:

' ORDER BY 1-- '

ORDER BY 2-- '

ORDER BY 3—

и так далее.

Это изменяет исходный запрос, упорядочивая результаты по разным столбцам. Когда указанный индекс столбца превышает фактическое количество столбцов в результате, база данных возвращает ошибку, например позиционный номер 3 в ORDER BY вне диапазона количества элементов в списке SELECT.

Ответ приложения может содержать ошибку базы данных, общий ответ об ошибке или просто отсутствие результатов. Любое различие в ответе позволяет определить количество столбцов.

**Метод с использованием UNION SELECT**. Внедряйте последовательные запросы **UNION SELECT**, указывая разное количество значений **NULL**:

' UNION SELECT NULL—

' UNION SELECT NULL, NULL—

' UNION SELECT NULL, NULL, NULL—

и так далее.

Если количество **NULL** не совпадает с количеством столбцов, база данных возвращает ошибку, например все запросы, объединенные оператором UNION, INTERSECT или EXCEPT, должны иметь одинаковое количество выражений в целевых списках.

Мы используем **NULL** в качестве возвращаемых значений, так как этот тип совместим с большинством типов данных, что увеличивает вероятность успешной инъекции при правильном количестве столбцов. Как и в случае с **ORDER BY**, реакция приложения может варьироваться — от отображения ошибки до появления дополнительных данных.

# ПРАКТИЧЕСКАЯ ЧАСТЬ

Предполагается, что у студентов установлен образ Kali Linux на виртуальной машине.

## 1 Установка Docker на Kali Linux

Docker — это платформа с открытым исходным кодом, предназначенная для разработки, развертывания и запуска приложений с использованием контейнеризации. Контейнеры Docker позволяют упаковывать приложение вместе с его зависимостями в единый переносимый образ, который можно запускать на любой системе, где установлен Docker. Это помогает устранить проблемы совместимости между различными средами разработки и эксплуатации.

Основные понятия Docker:

Контейнеры - это изолированные среды, в которых работает приложение со всеми необходимыми библиотеками, зависимостями и конфигурациями. Контейнеры легко запускать, останавливать и перемещать между разными системами.

Образы (Images) - это шаблоны, из которых создаются контейнеры. Образы содержат все необходимое для работы приложения. Обычно образы создаются на основе Dockerfile (описание сборки образа).

Dockerfile - текстовый файл, содержащий инструкции для создания образа. Он описывает шаги установки программ, конфигурации и настройки системы для запуска приложения.

Docker Hub - это облачный реестр, где пользователи могут загружать, искать и скачивать образы Docker, созданные сообществом или официальными поставщиками.

Преимущества Docker:

Портативность: контейнеры можно запускать на любом сервере или устройстве с установленным Docker, независимо от окружения.

Изоляция: контейнеры изолированы друг от друга, что позволяет минимизировать влияние одного приложения на другое.

Быстрая разработка и развертывание: использование контейнеров упрощает перенос приложений между стадиями разработки и эксплуатации.

Эффективность ресурсов: контейнеры потребляют меньше ресурсов по сравнению с традиционными виртуальными машинами, так как они используют общее ядро ОС.

Основные команды Docker:

Запуск нового контейнера:

docker run

Отображение работающих контейнеров:

docker ps

Создание образа из Dockerfile:

docker build

Загрузка образа из Docker Hub:

docker pull

Загрузка образа в Docker Hub

docker push

Docker широко используется для создания современных приложений, микросервисов, CI/CD процессов, тестирования и разработки.

При установке Docker на Kali Linux необходимо учитывать, что в системе уже существует пакет с именем «docker». Поэтому для установки контейнерной версии Docker используется другой пакет под названием docker.io. Если установить стандартный пакет «docker», это не приведет к установке контейнерной версии Docker.

Мы будем устанавливать пакет docker.io. Все команды, связанные с Docker, остаются прежними, поэтому для работы в командной строке нужно использовать команду docker.

Процесс установки выглядит следующим образом:

1. Обновление списка пакетов:

kali@kali:~$ sudo apt update

1. Установка Docker:

kali@kali:~$ sudo apt install -y docker.io

1. Активация и автоматический запуск службы Docker:

kali@kali:~$ sudo systemctl enable docker --now

1. Проверка установки и работы Docker:

kali@kali:~$ docker

В конце должно высветиться сообщение (рис. 1.1)

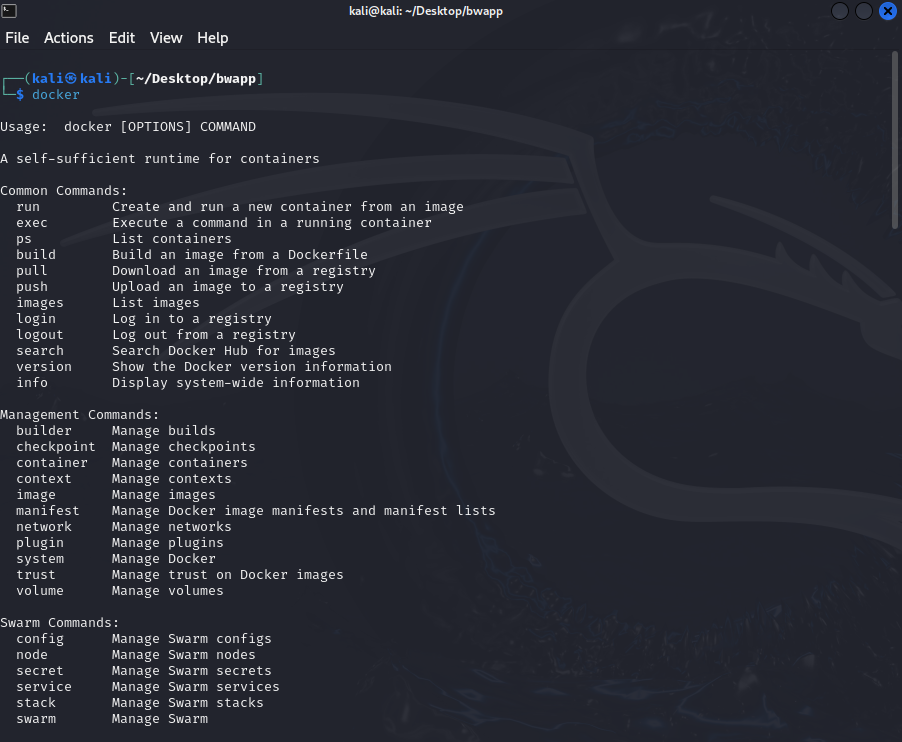


Рисунок 1.1. Проверка на наличии утилиты Docker в системе.

## 2 Установка и запуск bWAPP

bWAPP (Buggy Web Application) — это бесплатное и открытое программное обеспечение, представляющее собой намеренно уязвимое веб-приложение. Оно создано для обучения и практики в области безопасности и служит полезным инструментом для специалистов по кибербезопасности, разработчиков и студентов.

bWAPP позволяет исследовать и предотвращать различные веб-уязвимости, помогая пользователям осваивать навыки, необходимые для успешного проведения тестов на проникновение и проектов в сфере этичного хакинга.

Официальный сайт приложения с уязвимостями можно найти по адресу <http://www.itsecgames.com/index.htm.> Но для работы достаточно установить и запустить Docker контейнер.

Установка происходит следующим образом:

1. Скачать образ:

sudo docker pull raesene/bwapp

1. Запустить контейнер:

sudo docker run -d -p 80:80 raesene/bwapp

Теперь достаточно перейти на страницу <http://localhost/install.php> и кликнуть «here» в надписи «Click here to install bWAPP.»

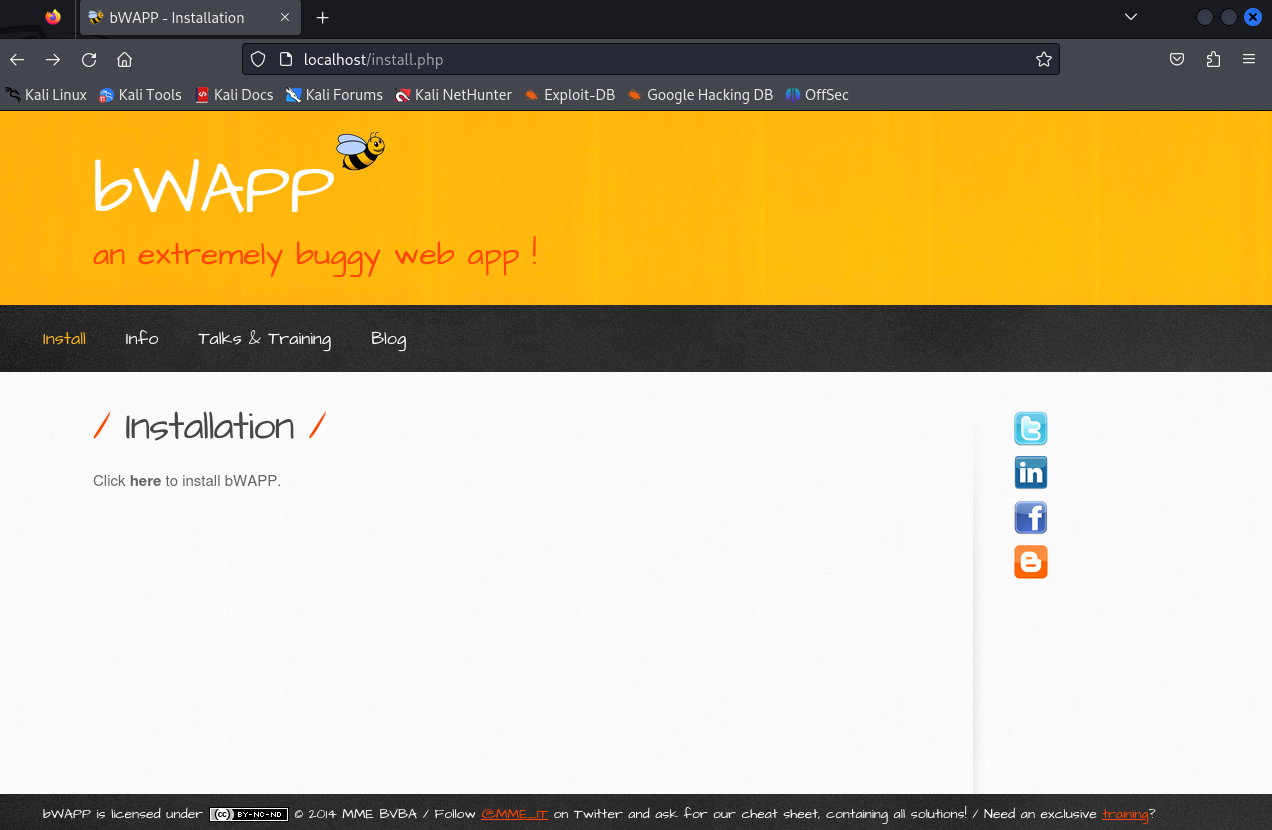


Рисунок 2.1. Окно приветствия.

Если все получилось, откроется основная страничка (рис. 2.2).

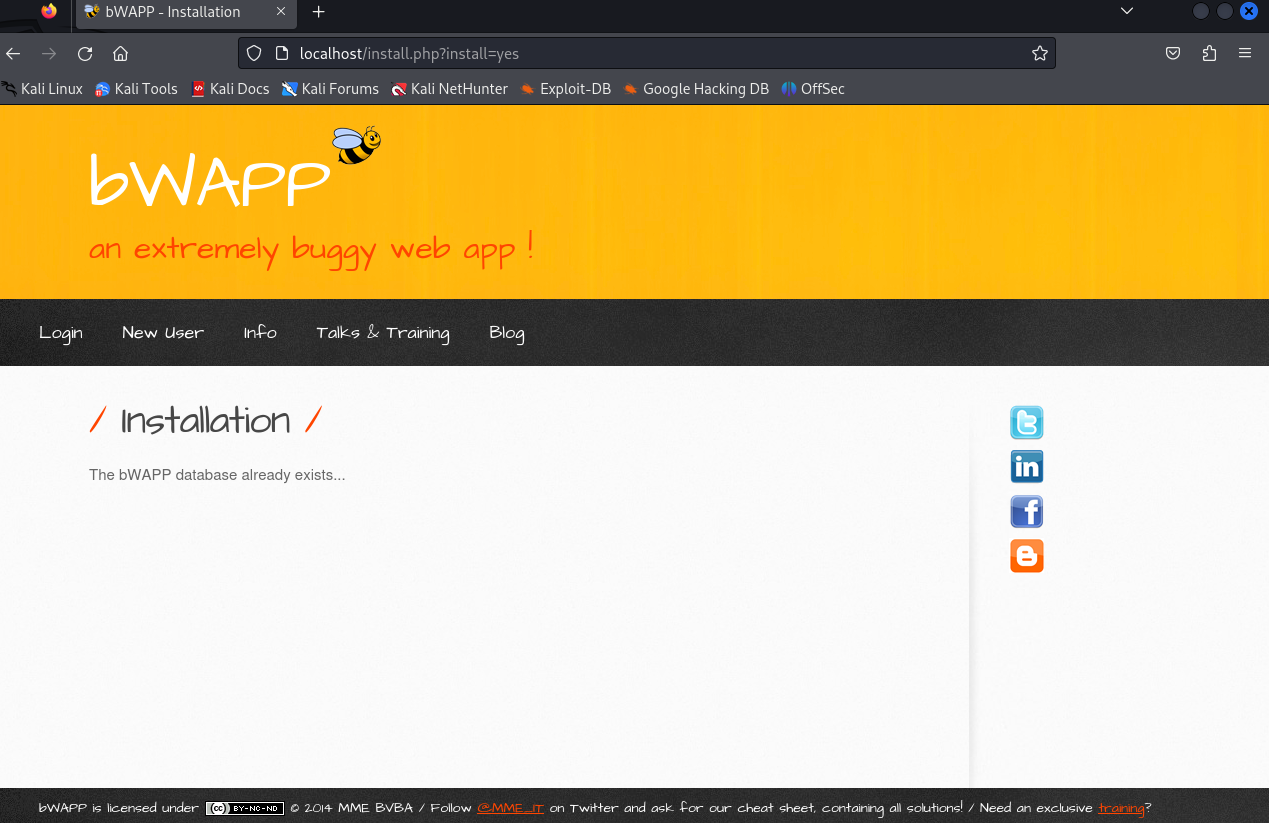


Рисунок 2.2. Стартовая станица.

На стартовой странице можно наблюдать меню с несколькими кнопками:

* «Login» – перенаправляет на страницу авторизации;
* «New User» – страница для создания нового пользователя (это делать необязательно, существуют стандартные логин и пароль: bee/bug);
* «Info» – рассказывает о проекте bWAPP
* «Tasks & Tranning» – смежные проекты bWAPP
* «Blog» – переход на блог авторов

Для дальнейшей работы необходимо перейти во вкладку Login и ввести логин: bee и пароль: bug. Уровень сложности оставить low. После прохождения авторизации открывается портал с выбором заданий (рис. 2.3)



Рисунок 2.3. Портал с заданиями.

## 3 Решение задач

### 3.1 SQL Injection (Login Form/Hero)

Необходимо открыть Bugs > SQL Injection (Login Form/Hero). Откроется станица с заданием. Здесь можно наблюдать форму логина и предложение ввести данные для авторизации (рис. 3.1.1)

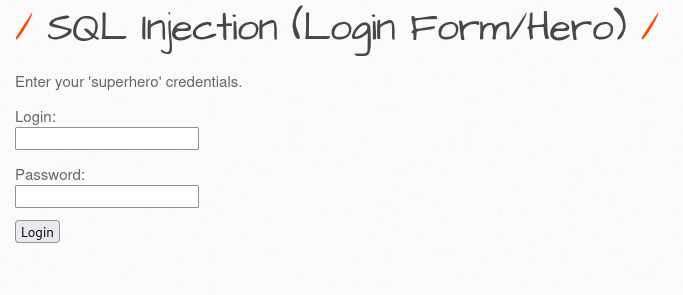


Рисунок 3.1.1. Задание SQL Injection (Login Form/Hero).

Становится нажав F12 и исследовав код, становится понятно, что форма принимает 2 параметра ligin и password (рис. 3.1.2)

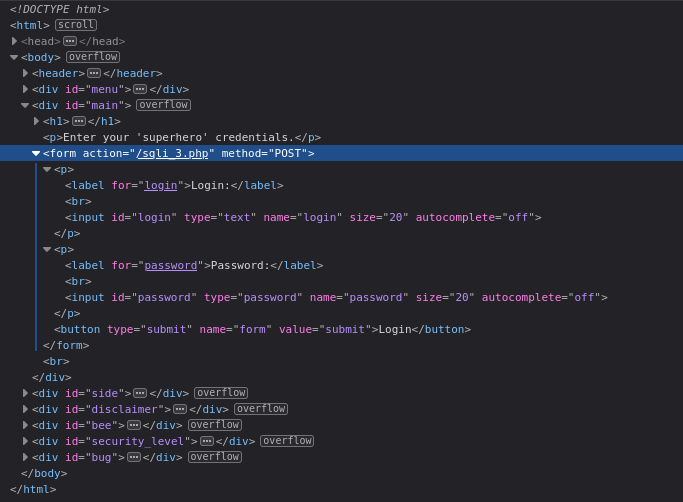


Рисунок 3.1.2. Форма запроса

Введя случайные символы получаем сообщение об ошибке (рис. 3.1.3)

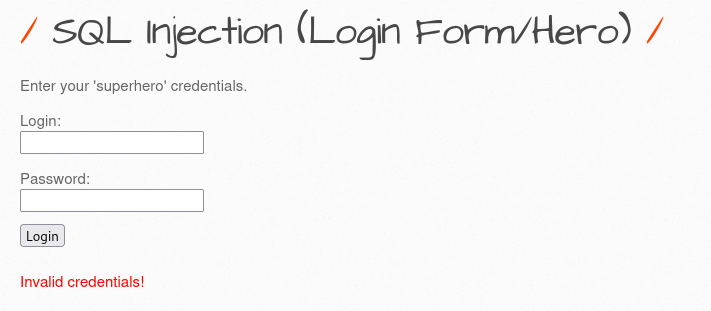


Рисунок 3.1.3. Сообщение об ошибке.

Подобные формы отправляют данные на сервер, где происходит обработка с использованием базы данных. Можем предположить, что существует запрос, который проверяет наличие пользователя и правильность пароля:

SELECT \* FROM users WHERE login = '<login \_input>' AND password = '<password\_input>';

Предположим, что введенные символы не экранируются и проведем атаку, введя в поля формы код:

a' or 1=1--’

Получаем сообщение об успешном входе (рис. 3.1.4)

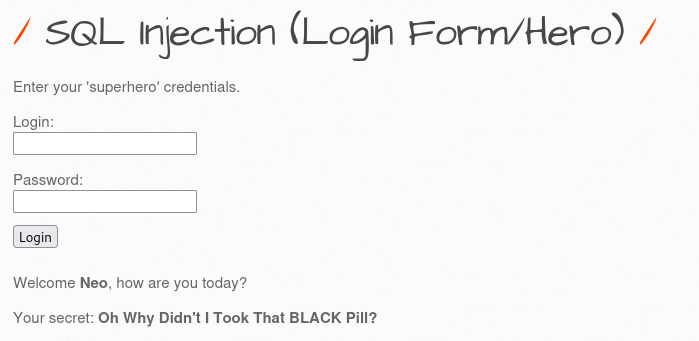


Рисунок 3.1.4. Успешная эксплуатация уязвимости.

### 3.2 SQL Injection (GET/Select)

Необходимо открыть Bugs > Injection (GET/Select). Откроется станица с заданием. Здесь расположены выпадающий список и таблица под ним (рис. 3.2.1)



Рисунок 3.2.1. Задание Injection (GET/Select).

Выбрав фильм из выпадающего списка, можно увидеть, параметры запроса передаются в URL адресе, что говорит о том, что форма использует GET запрос (рис. 3.2.2)

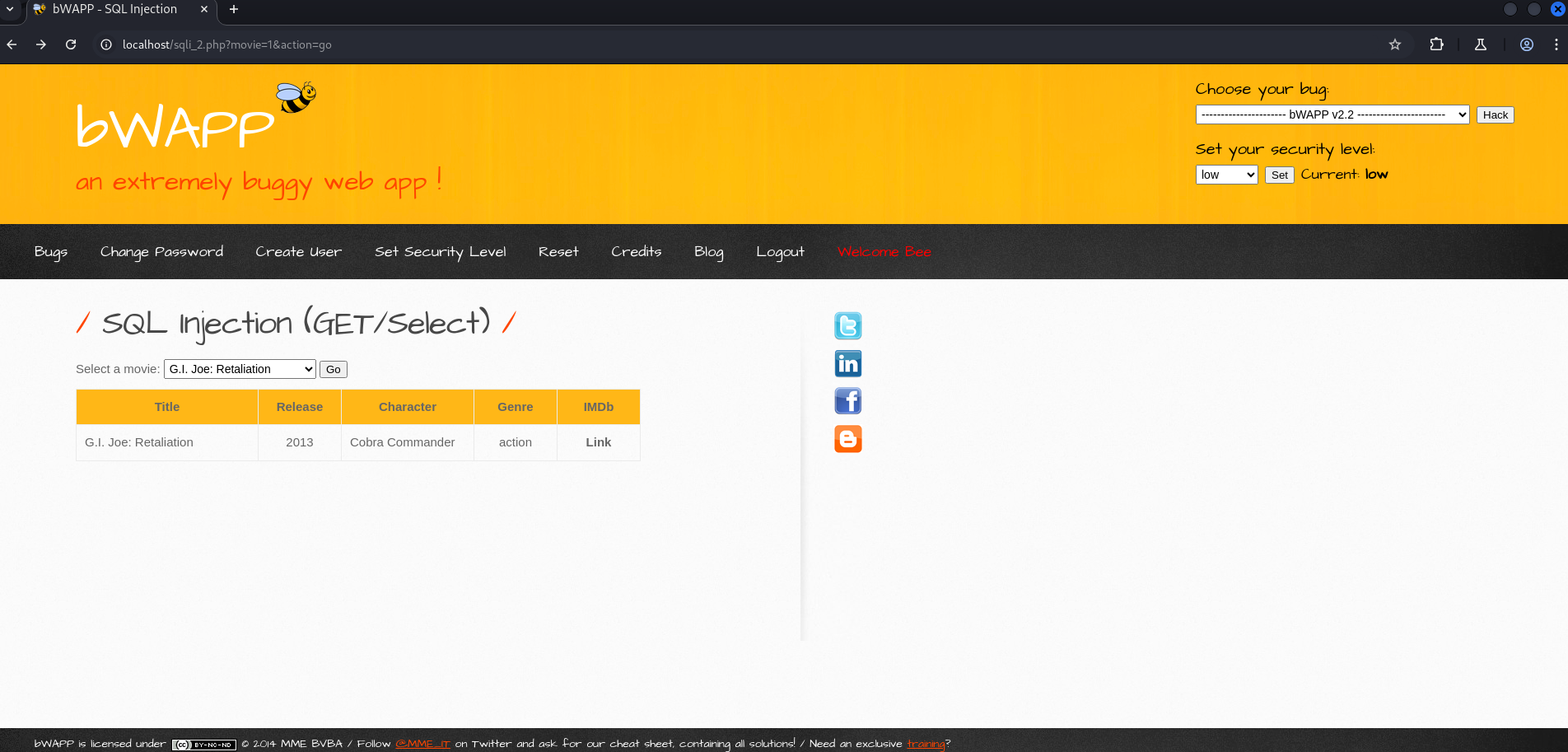


Рисунок 3.2.2. Выбор фильма.

Теперь можно составить инъекцию, которая будет выводить информацию о таблицах базы данных:

movie=1 UNION ALL SELECT table\_schema, table\_name, null, null, null, null, null FROM information\_schema.tables LIMIT 1 OFFSET 1;--

Объяснение запроса:

* movie=1 — это исходная часть запроса, которая передает параметр в приложение. В случае SQL-инъекции злоумышленник может изменить его, чтобы добавить свой вредоносный SQL-код.
* UNION ALL SELECT — используется для объединения результатов из исходного запроса с результатами, полученными из дополнительного SELECT-запроса, который извлекает информацию о таблицах.
* table\_schema, table\_name — из схемы information\_schema выбираются столбцы table\_schema (схема базы данных) и table\_name (имя таблицы). Эти столбцы содержат информацию о всех таблицах в базе данных.
* null, null, null, null, null, null — здесь добавляются дополнительные NULL значения, чтобы привести количество столбцов в запросе к тому же числу, что и в исходном запросе, который выполняется в приложении. Это необходимо для того, чтобы запрос не вызвал ошибку из-за несоответствия количества столбцов.
* LIMIT 1 OFFSET 1 — ограничивает выборку одним результатом и пропускает первую строку, что позволяет обойти возврат первой таблицы. Это полезно, если нужно получить конкретную таблицу или строку из списка.
* -- — это комментарий, который игнорирует оставшуюся часть исходного запроса, предотвращая ошибки и обеспечивая успешное выполнение инъекции.

Результат:

Этот запрос позволяет получить имена таблиц из базы данных, возвращая только одну строку с данными о таблице, пропуская первую строку (рис. 3.2.3)

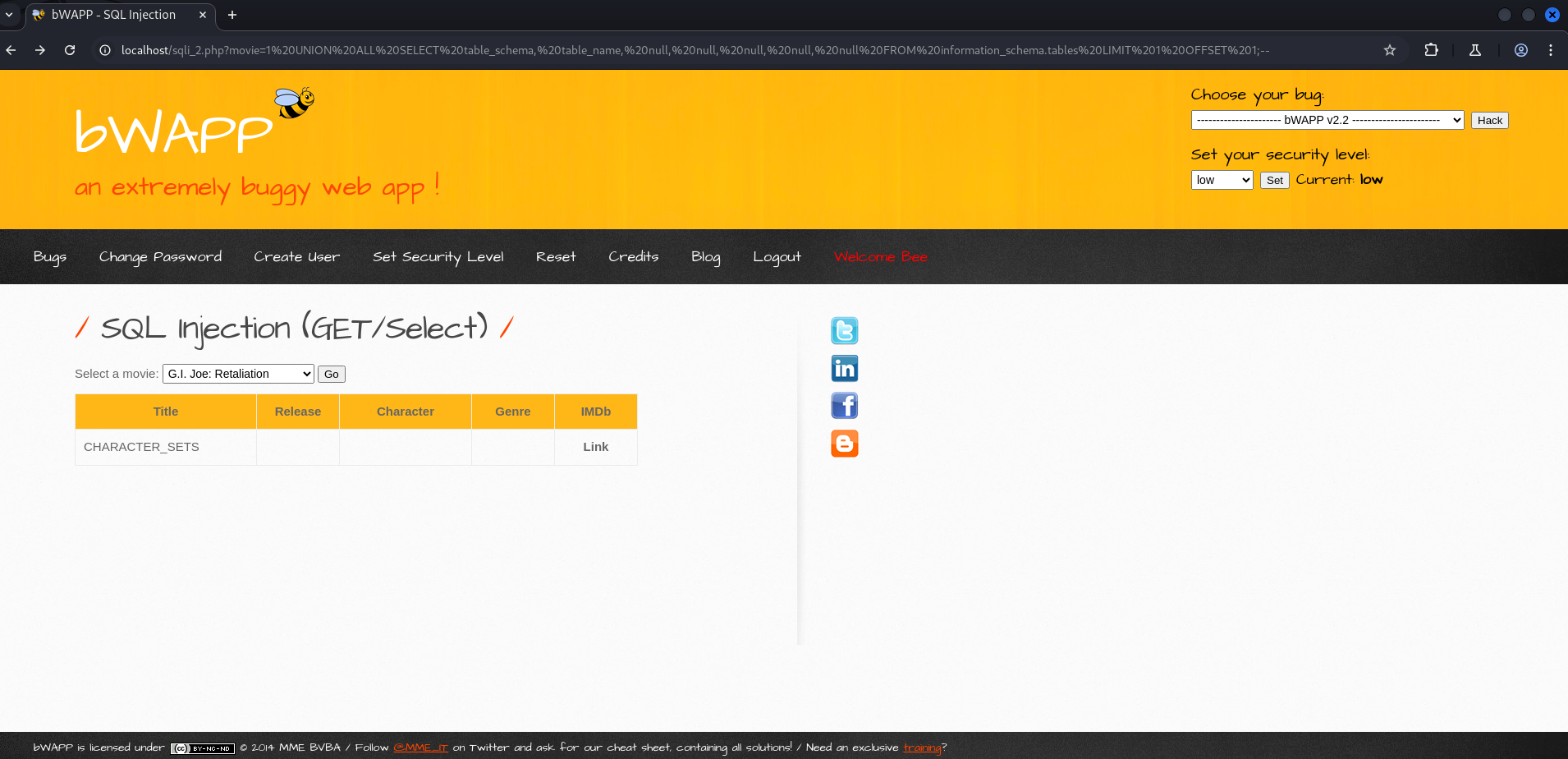


Рисунок 3.2.3. Успешная эксплуатация уязвимости.