Intense Introduction to Modern Web Application Hacking

Lab Guide

Omar Ωr Santos (@santosomar)

Introduction

This course starts with an introduction to modern web applications and immediately starts diving directly into the mapping and discovery phase of testing. In this course, you will learn new methodologies used and adopted by many penetration testers and ethical hackers. This is a hands-on training where 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

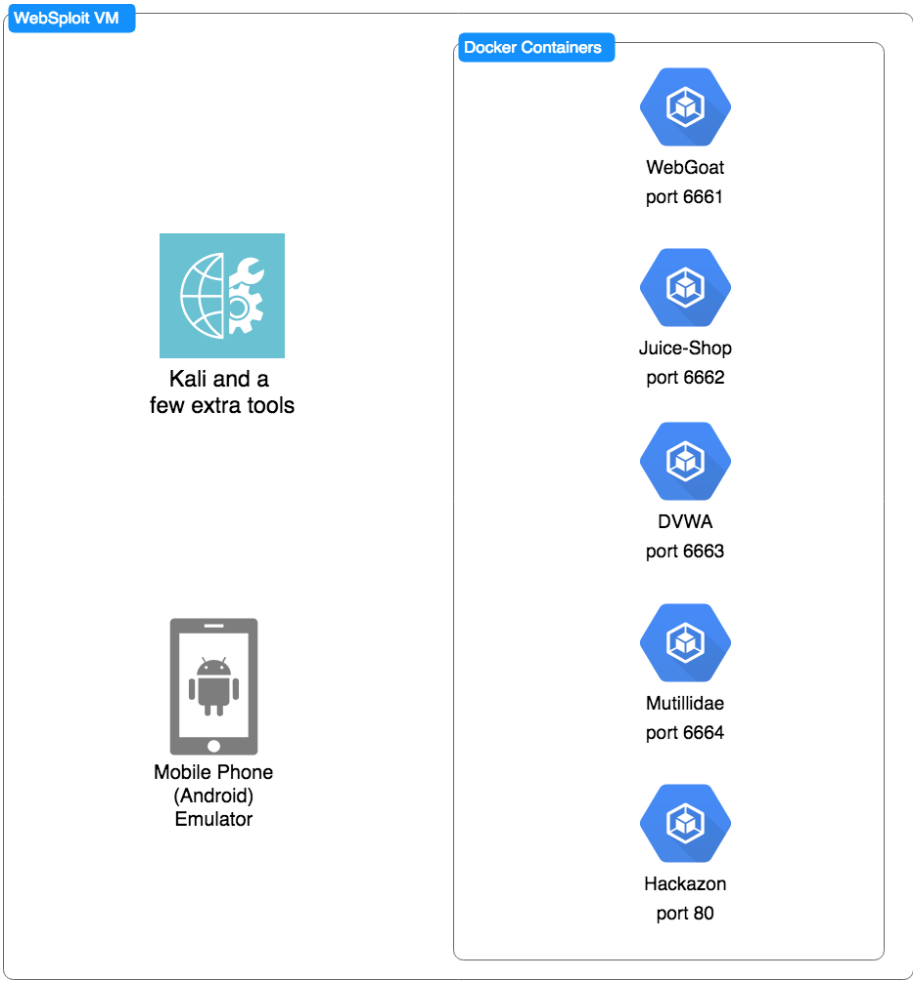
---

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

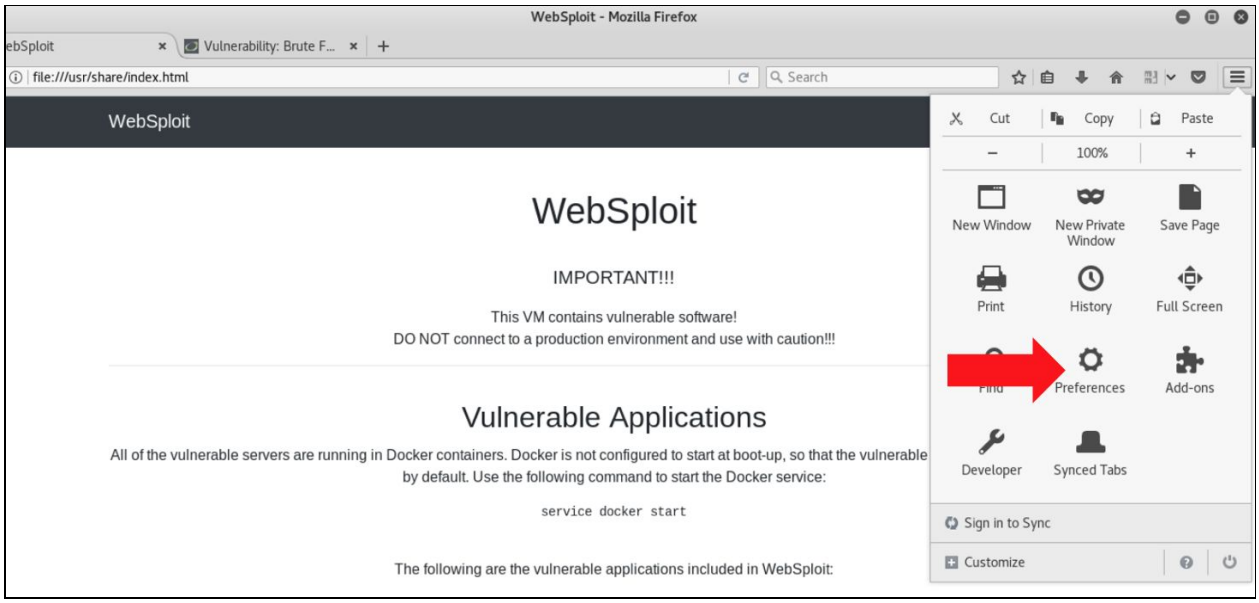
An attacker can bypass authentication in vulnerable systems via several methods. The following are the most common ways that you can take advantage of authentication-based vulnerabilities in an affected system:

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

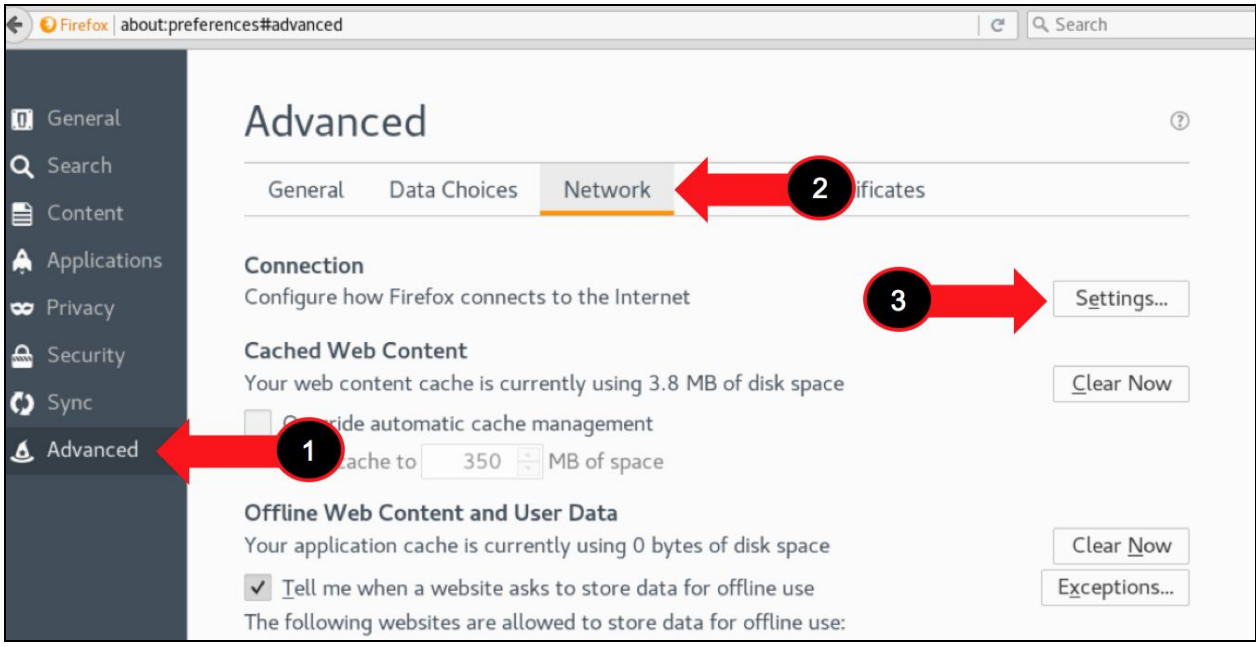
A large number of web applications keep track of information about each user for the duration of the web transactions. Several web applications have the ability to establish variables like access rights and localization settings and many others. These variables apply to each and every interaction a user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

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

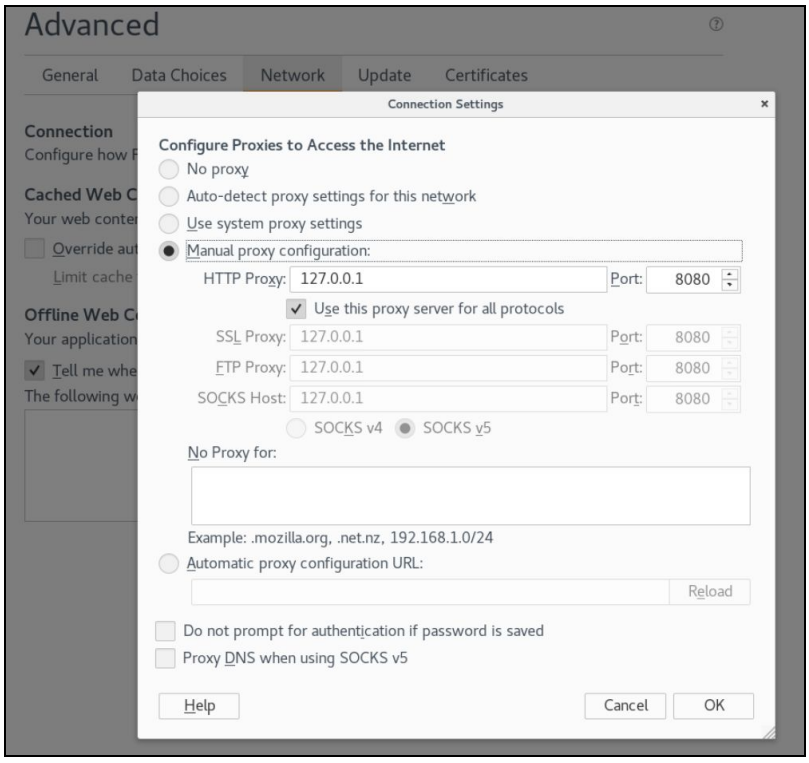
1. In this exercise you will try to determine what type of programming language and backend infrastructure is used by looking at **sessions IDs**. However, first you need to configure your browser to send traffic to the proxy (you can use Burp Suite or OWASP ZAP). Navigate to **Preferences**:



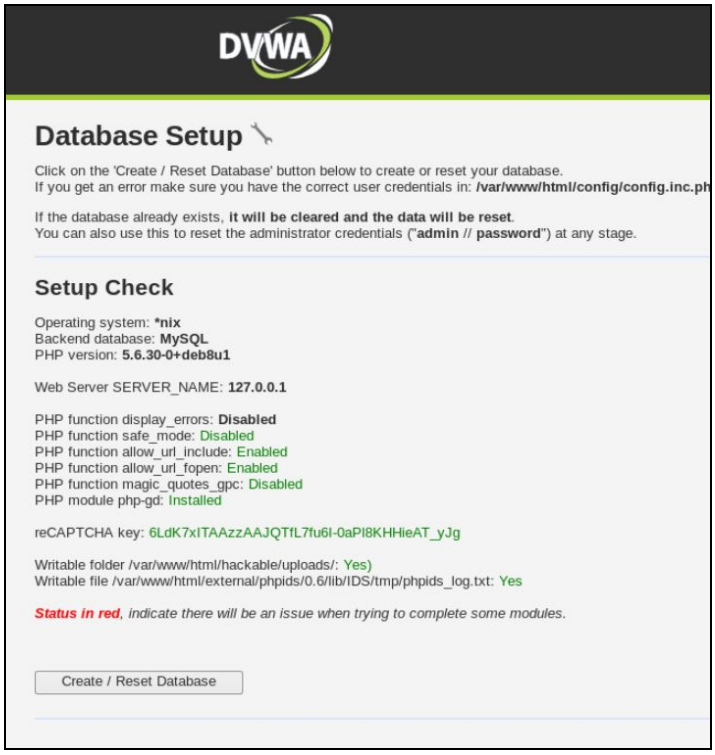
2. Then navigate to **Advanced > Network > Settings**.



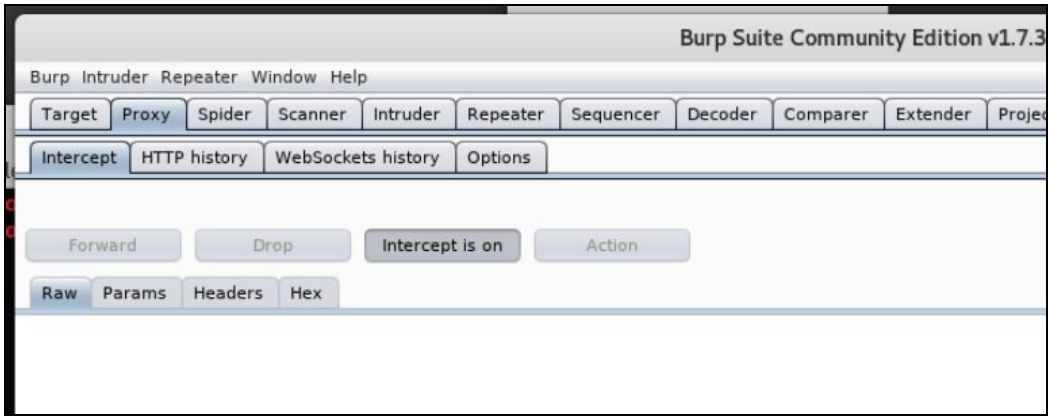
3. Configure the proxy as shown below. Make sure that the “No proxy for” box does not have any entry on it.



4. 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.



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



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

   Answer: _____

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

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

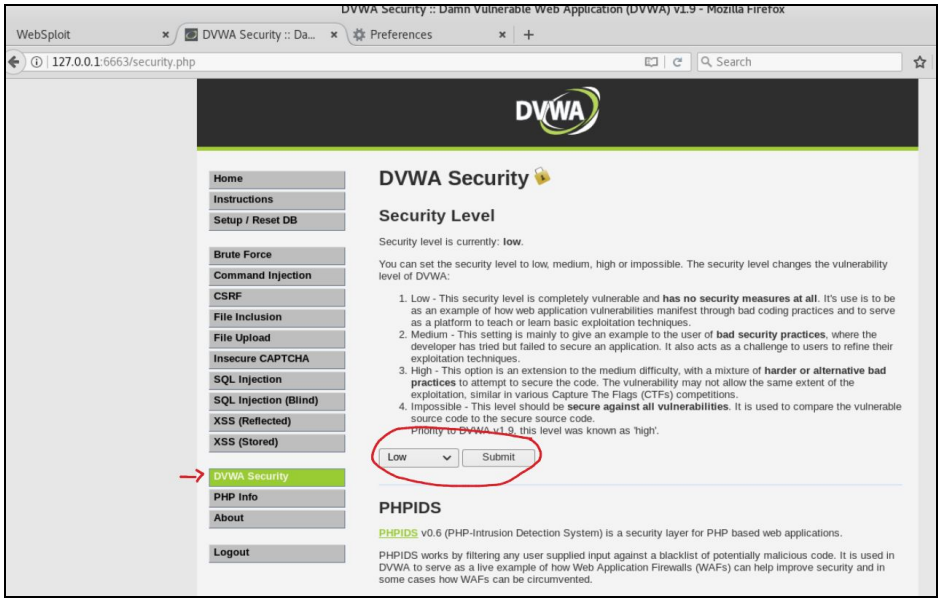
10. You can go to the Proxy History tab. This contains a table of all HTTP messages that have passed through the Proxy. Select an item in the table, and look at the HTTP messages in the request and response tabs. If you select the item that you modified, you will see separate tabs for the original and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

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

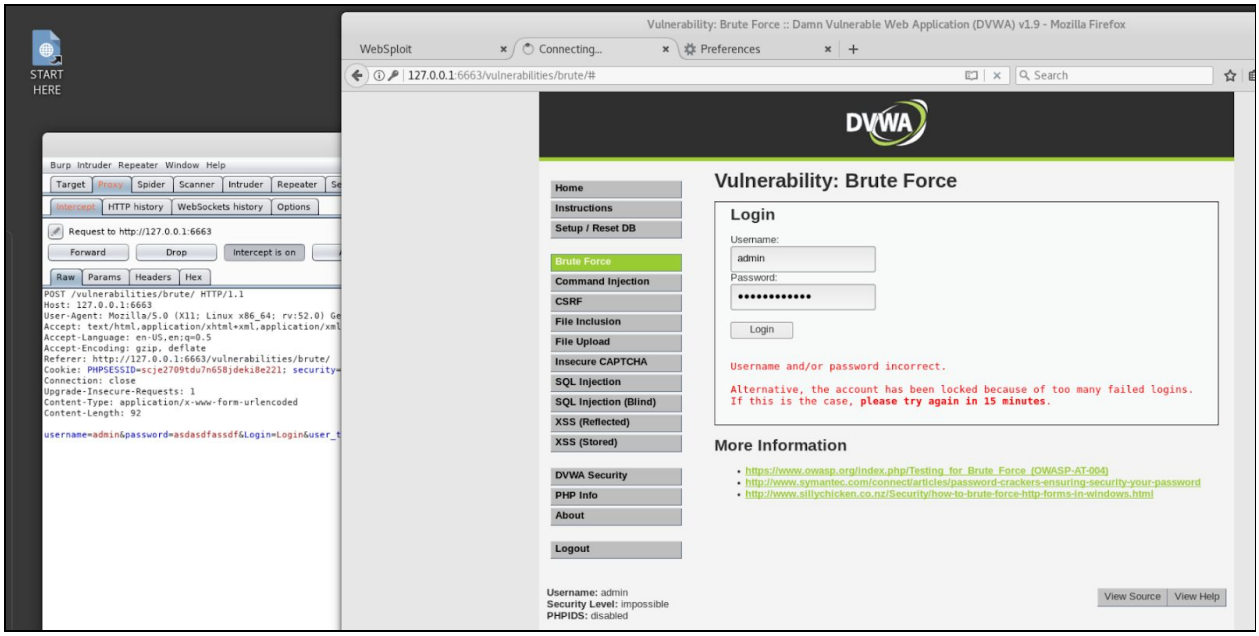
12. Within the history table, click on a cell in the leftmost column, and choose a color from the drop-down menu. This will highlight that row in the selected color. In another row, double-click within the Comment column and type a comment. You can use highlights and comments to annotate the history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

Exercise 1b: Brute Forcing the Application

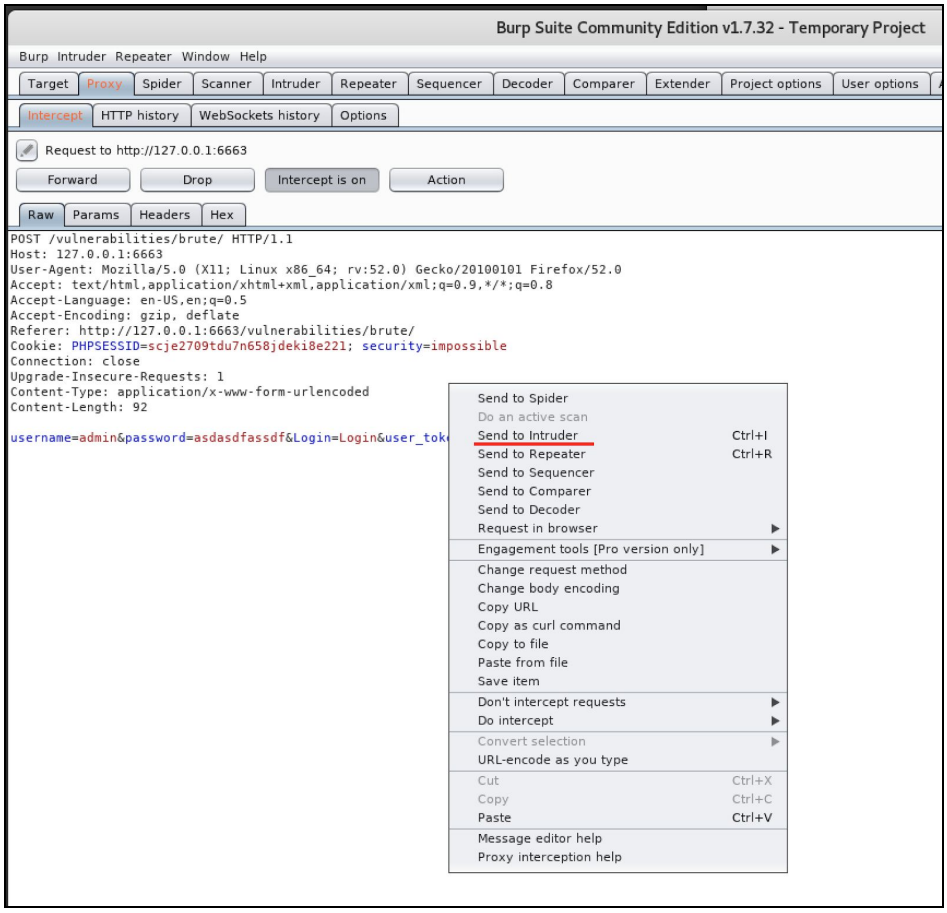
1. In this exercise you will try to bruteforce the admin password. This is a very simple example and should not take you more than 2-3 minutes. Set the DVWA Security Level to low, as shown below:



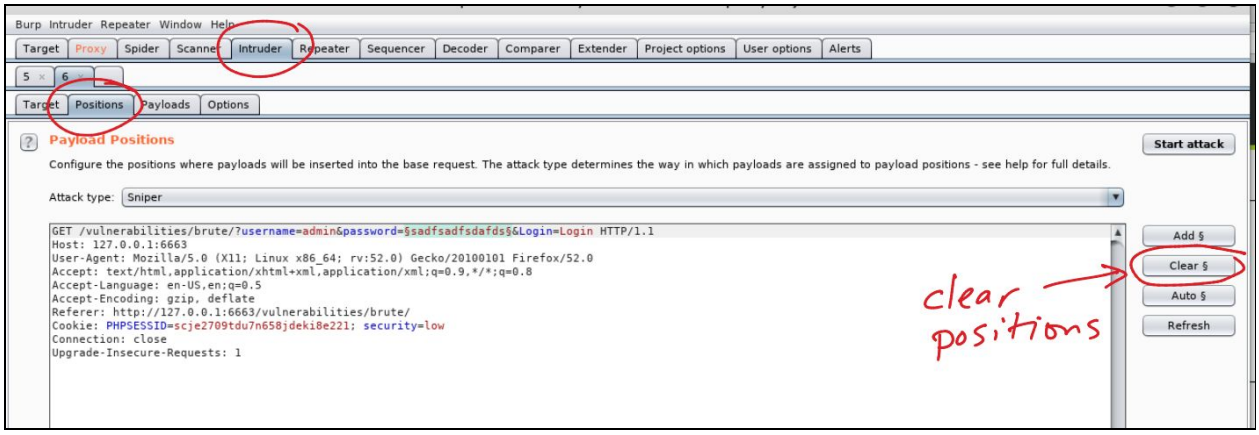
2. Navigate to DVWA and **Brute Force** again and type admin and any password.



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

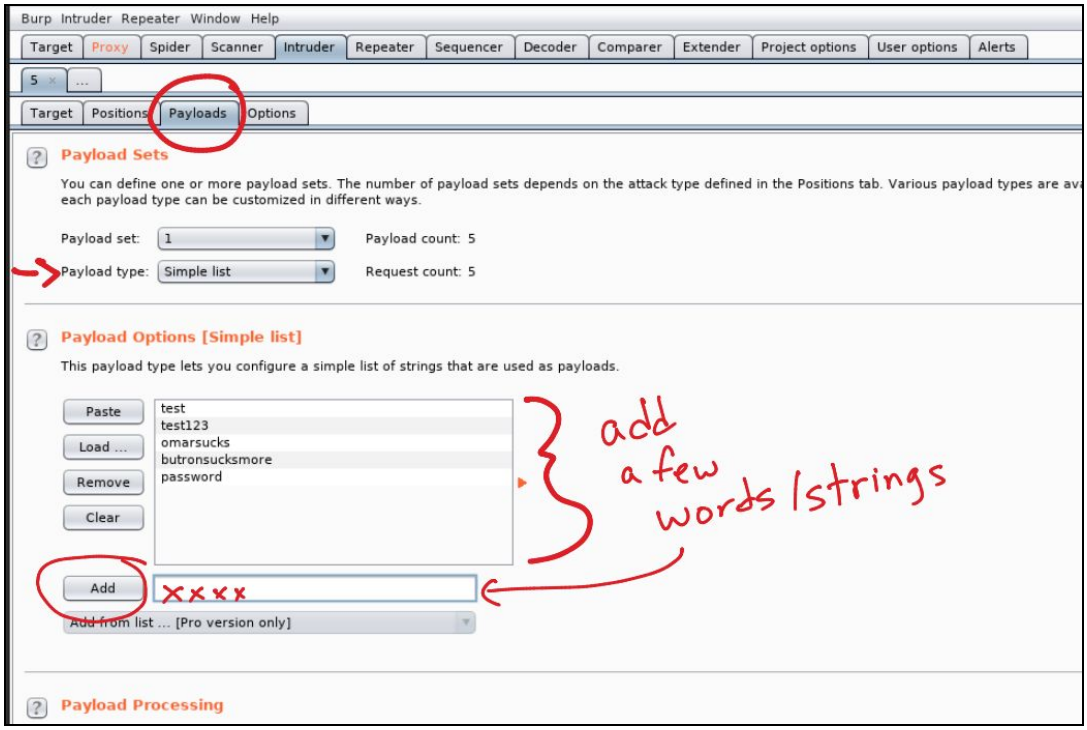


4. Navigate to **Intruder > Positions** and click on the **Clear** button.



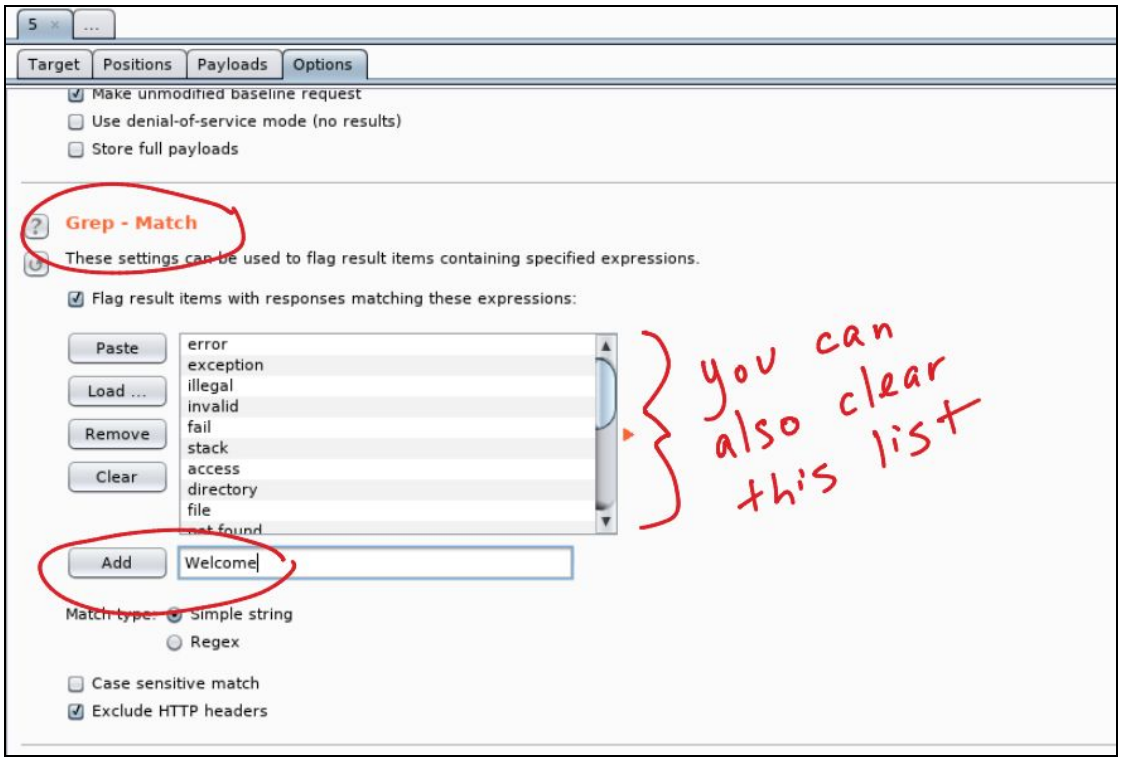
5. We can brute force any elements, but for this simple example we will just brute force the password.

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

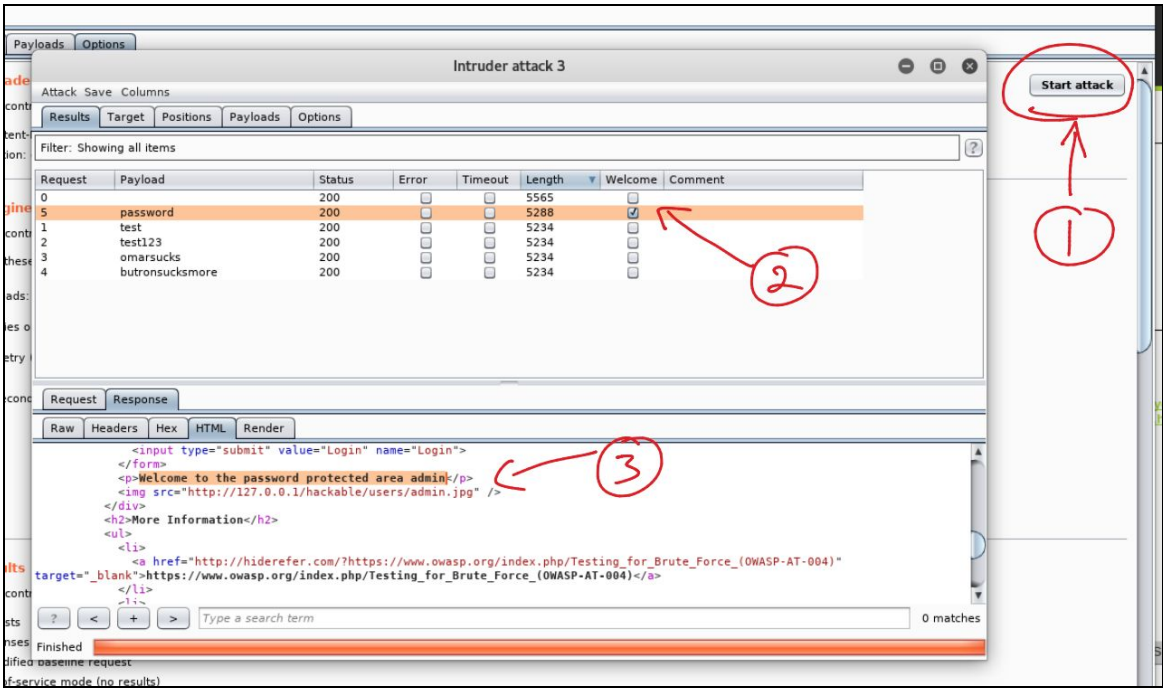


Note: You can only use wordlists in the Pro version of Burp; however, you can use the OWASP Zed Attack Proxy (ZAP) to also perform this task. As described by OWASP, the OWASP Zed Attack Proxy (ZAP) "is one of the world's most popular free security tools and is actively maintained by hundreds of international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

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



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



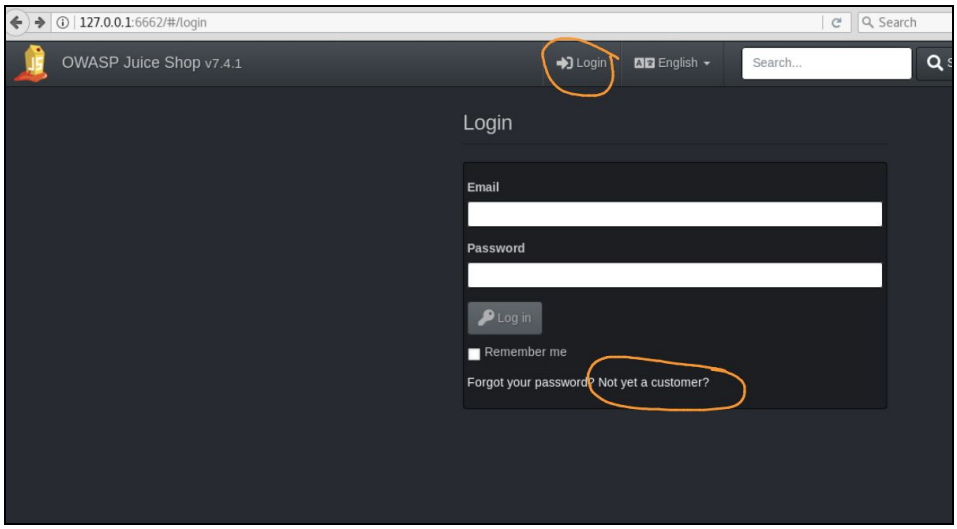
Exercise 1c: Bypassing Authorization

In this exercise we will use the OWASP Juice Shop (http://127.0.0.1:6662) 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.

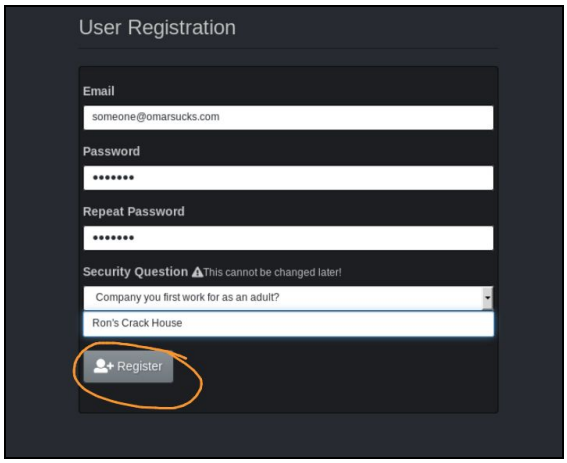
1. BONUS POINT (in under 60 seconds): The OWASP Juice Shop is a "capture-the-flag-like" application. Navigate to the OWASP Juice Shop (http://127.0.0.1:6662) and try to find the hidden scoring board for the "CTF". You only need your browser.

   Answer: _____

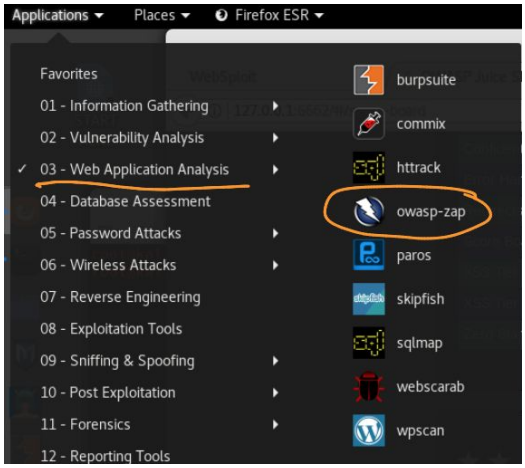
2. In the OWASP Juice Shop, navigate to Login and create a user.



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. Launch **ZAP** in Kali by navigating to **Applications > Web Application Analysis > OWASP ZAP**, as shown below:

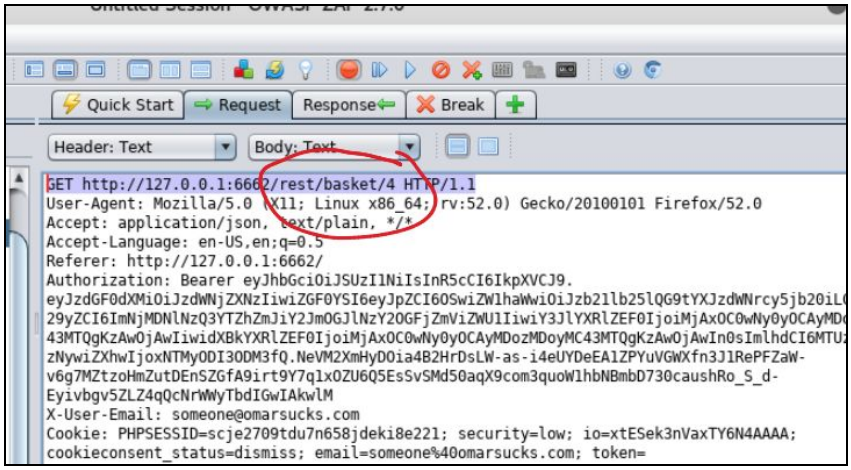


6. Add any item to your cart in the Juice Shop.

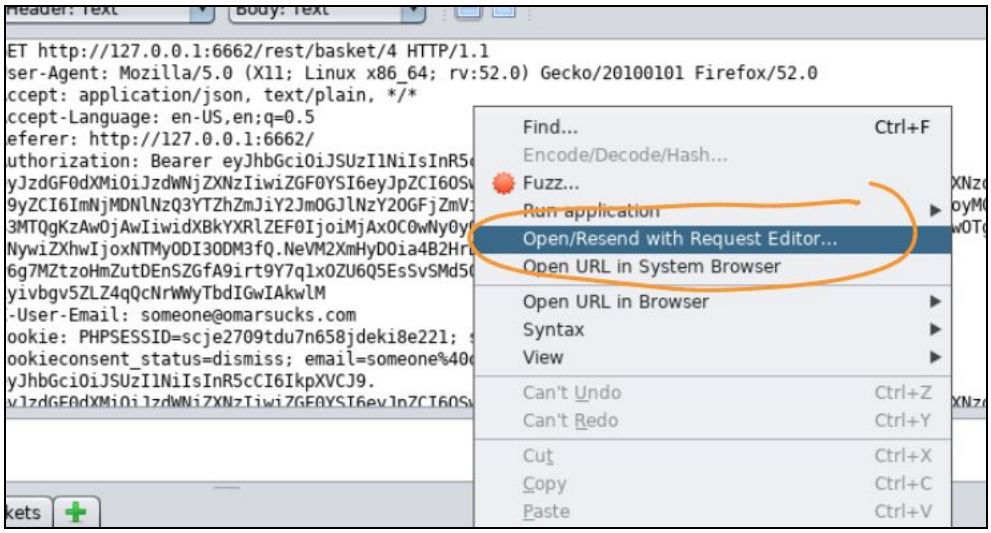
7. Make sure that your browser's proxy settings are configured correctly.

8. In the OWASP ZAP click on the Set Break for all requests and responses icon, as shown below.

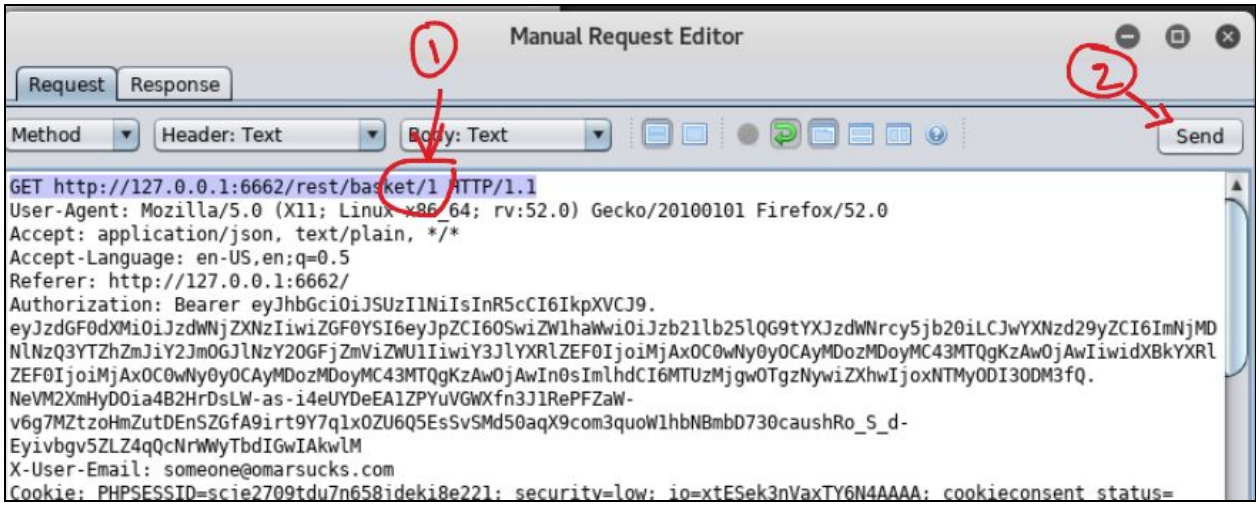
9. Navigate to your cart in the Juice Shop and capture the HTTP Request. You will observe a flaw where the request includes the **basket ID** in the **URL** (looks like a REST API request).



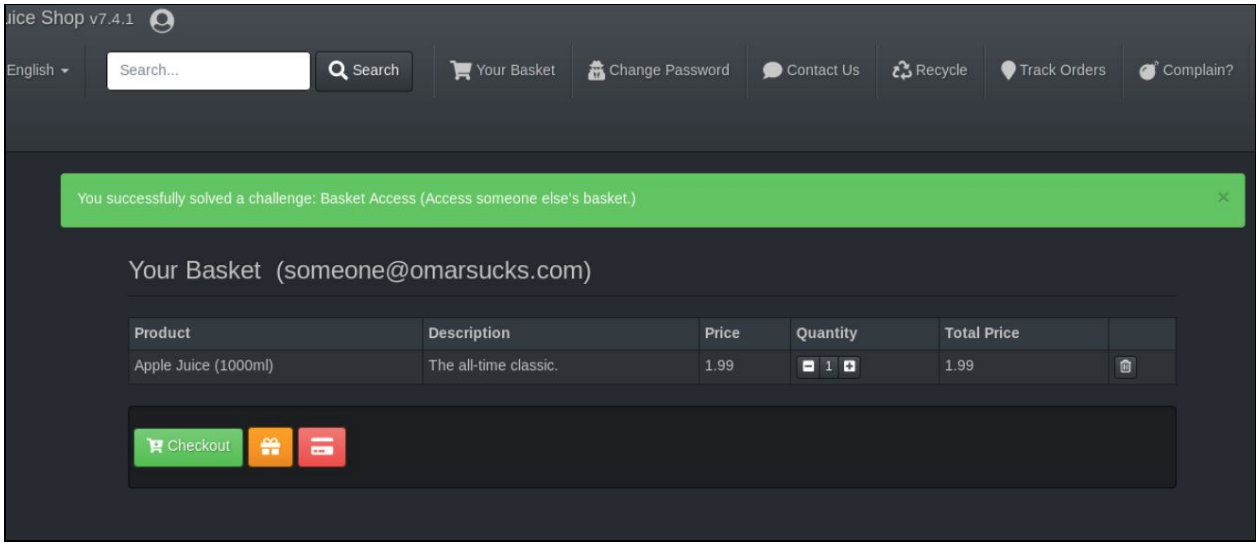
10. Right click on the **Request** window and select **Open/Resend with Request Editor...**



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).



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

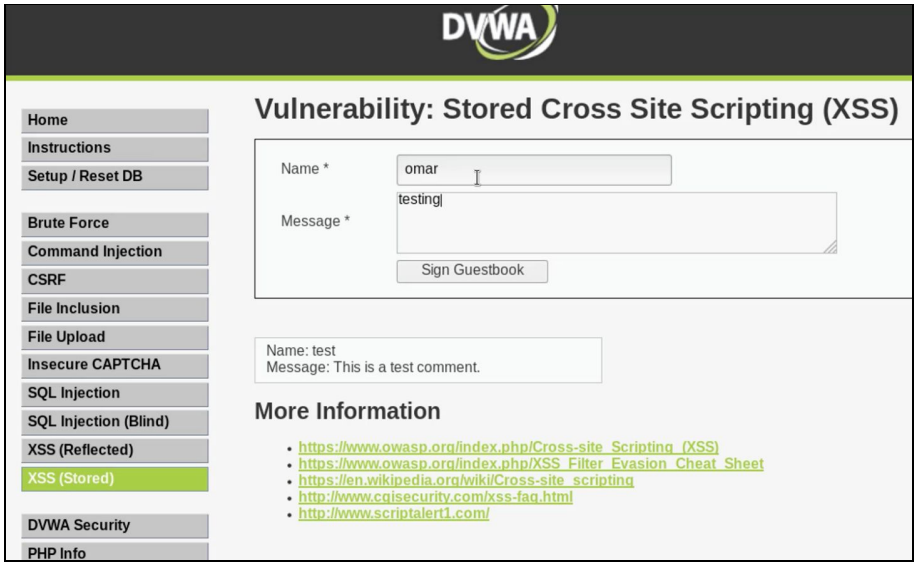
1. Find a DOM-based XSS in the Juice Shop application/site. You only need your browser for this attack. Find out how the Juice Shop is susceptible to DOM-based XSS.

   You can use the following string:

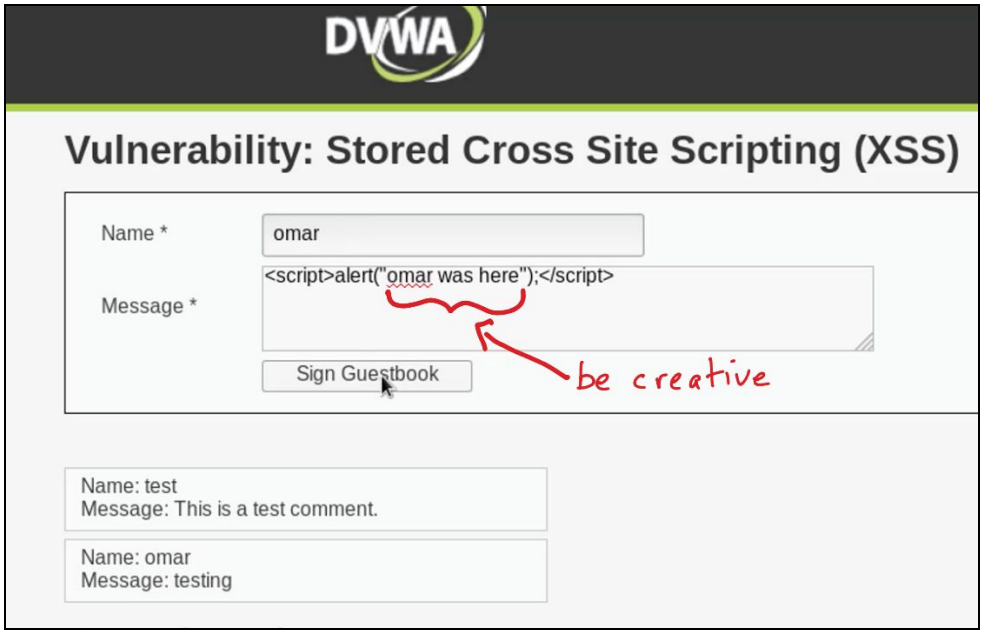
```
<script>alert("XSS")</script>
```

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.



3. Test for XSS, as shown below:



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

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

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

Exercise 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.

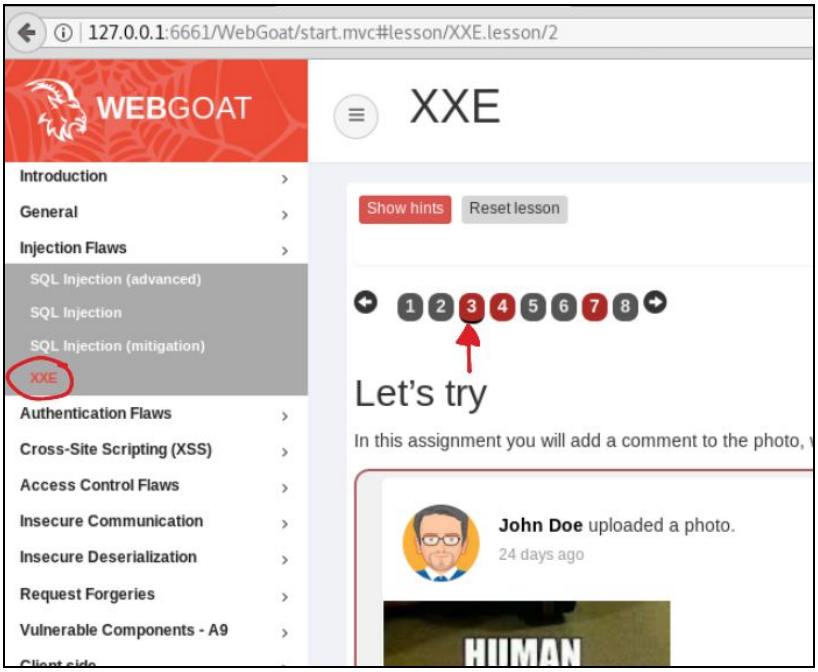
1. Access WebGoat using your browser (http://127.0.0.1:6661/WebGoat).

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

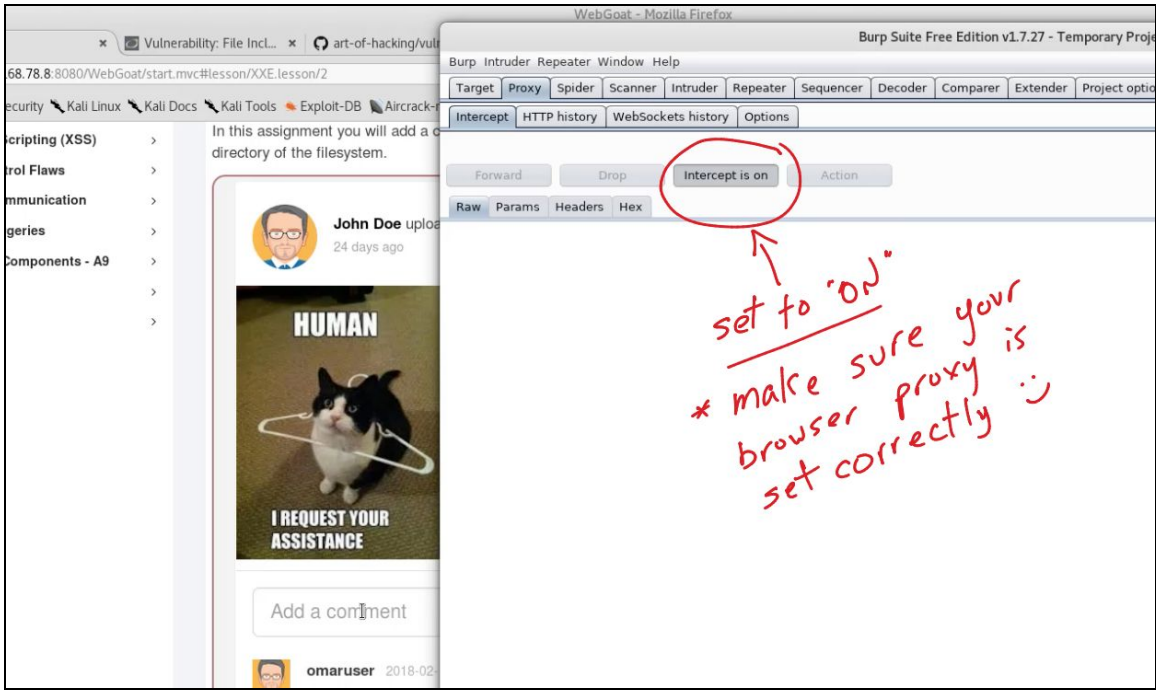
3. Navigate to **Injection Flaws > XXE**.

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

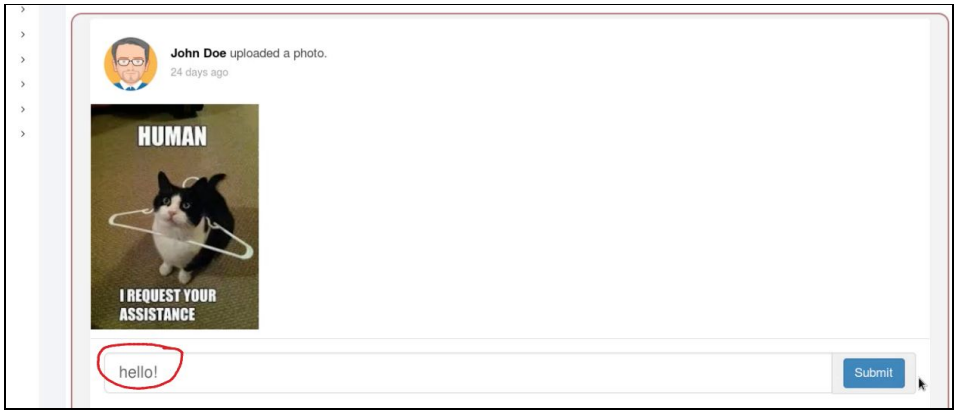
5. Then navigate to the WebGoat **Step 3**, as shown in the following figure.



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

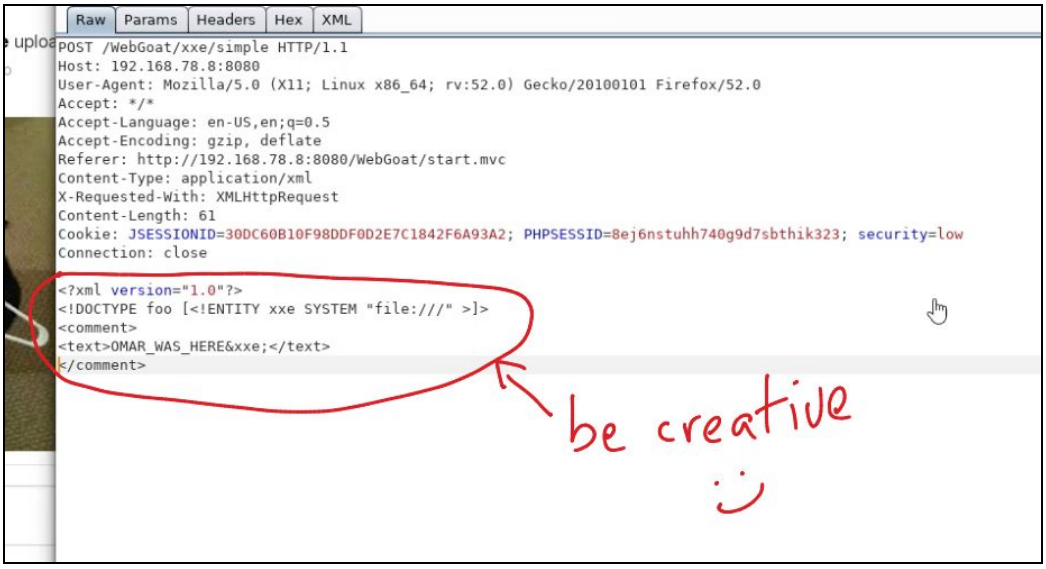


7. Go back to **WebGoat** and enter a comment in the web form (any text) and click **Submit**.

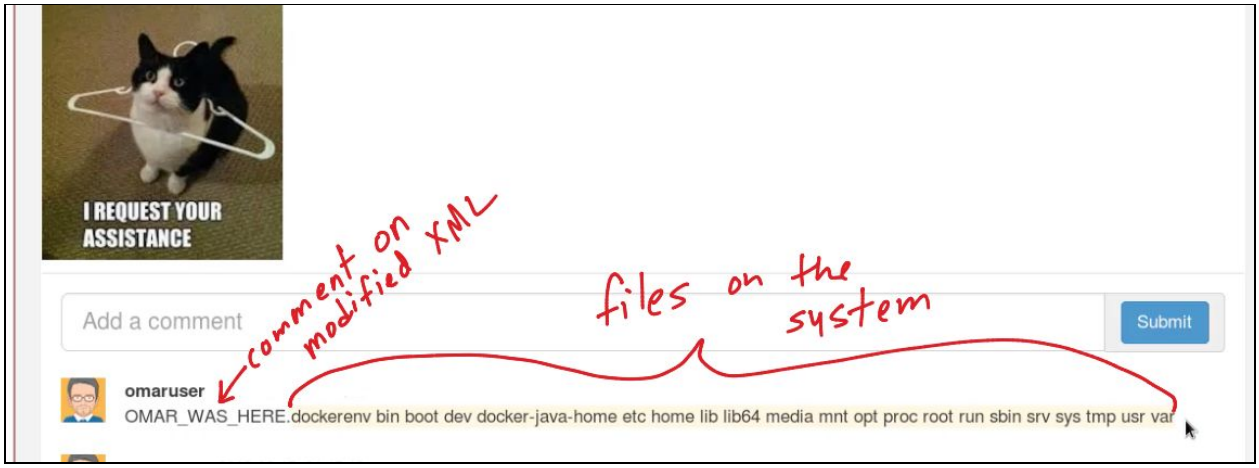


8. Go back to **Burp** and you will see the **HTTP POST message** shown below:

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



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



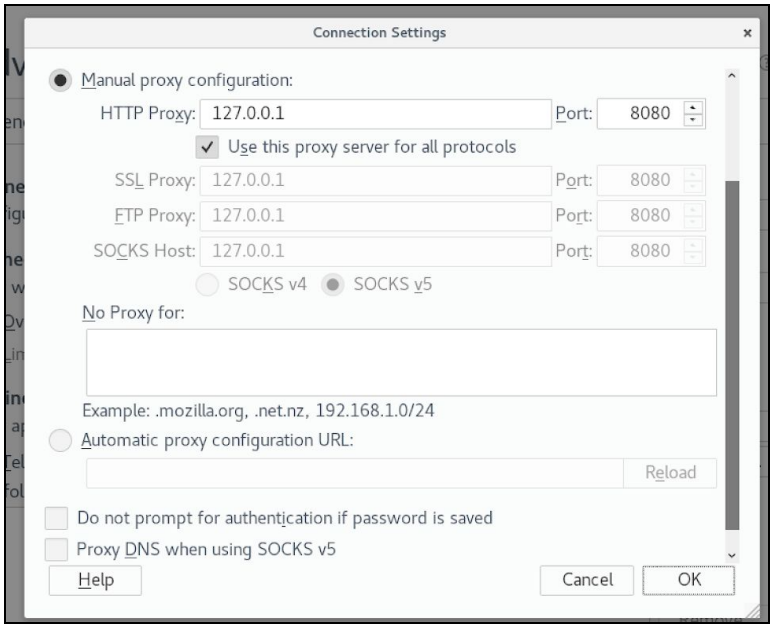
11. Now, in your own, try to list the contents of the `/etc/passwd` file using a similar approach.

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

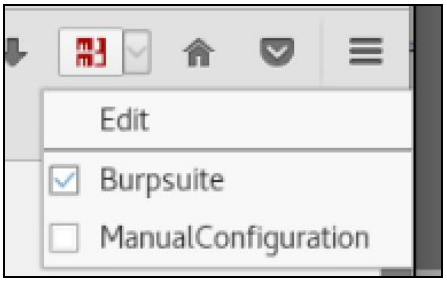
Exercise 5: SQL Injection using SQLmap

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.

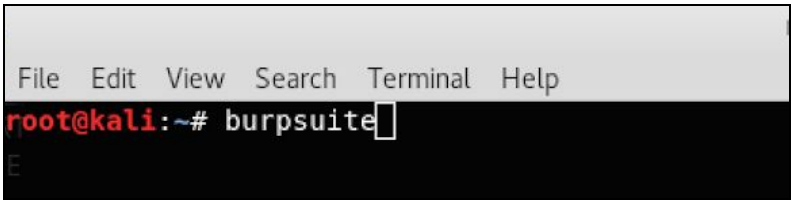
1. We will start by configuring the browser in Kali to send our web traffic through Burp Suite. If this is already done you can move on to the next step. The manual way to do this is in the browser preferences seen below.



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.



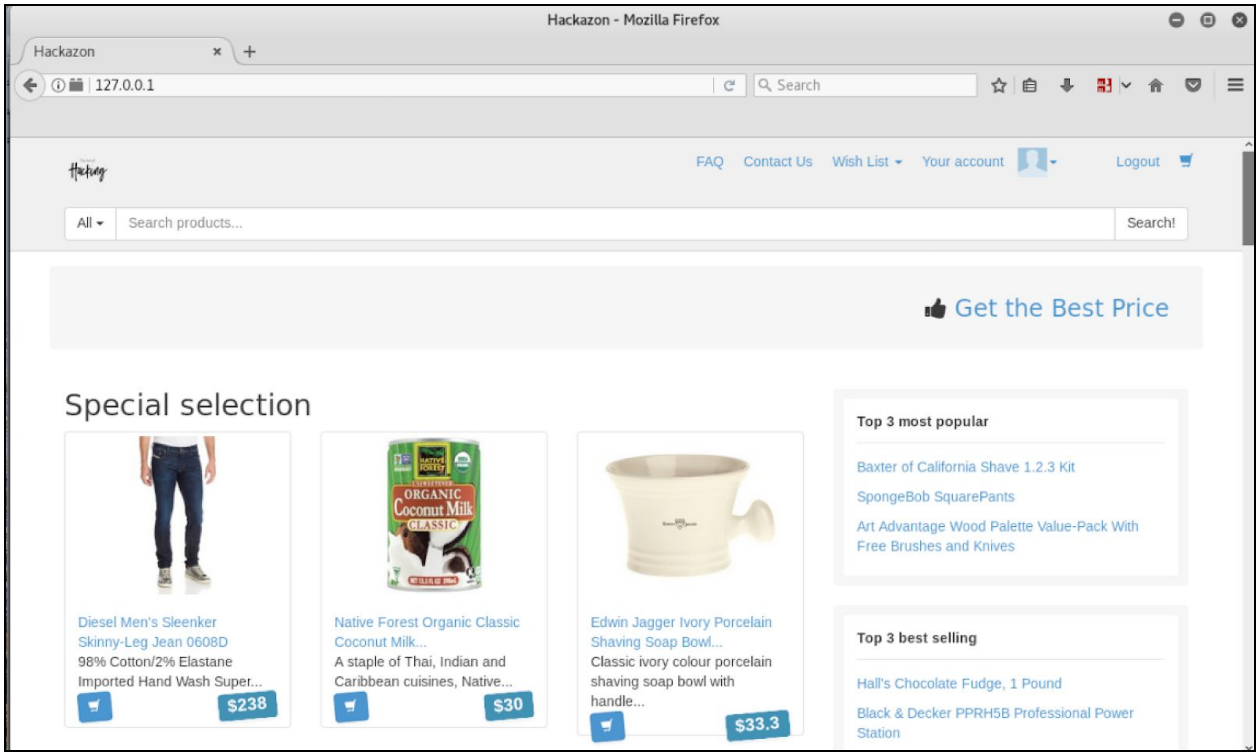
2. Let's first start Burp Suite by opening a terminal window and typing `burpsuite` at the command line.



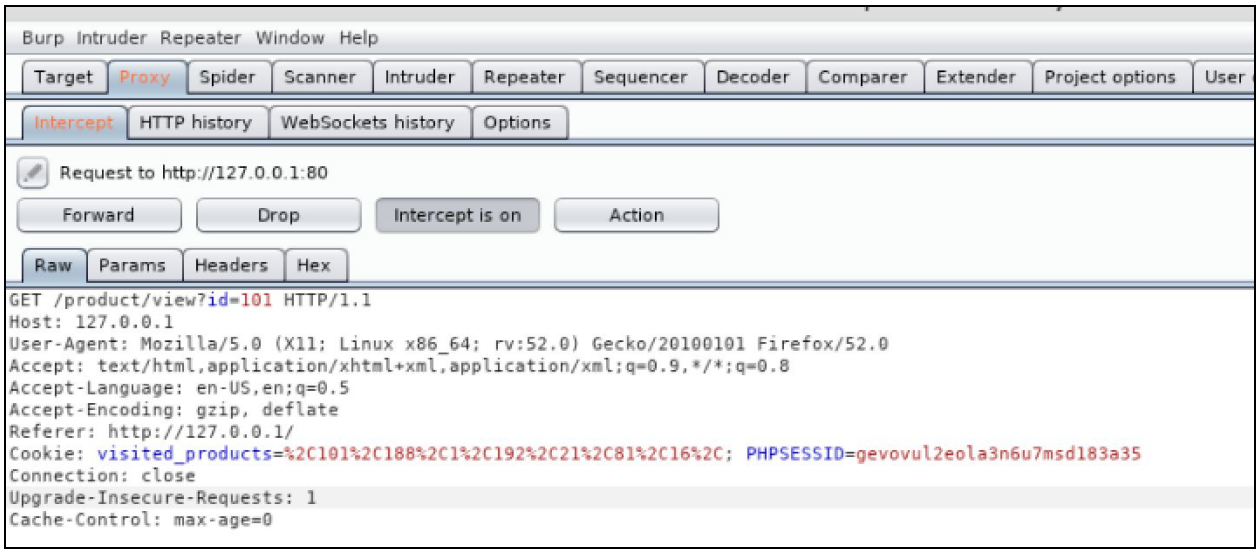
3. You will see the **Burp** splash screen.

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..

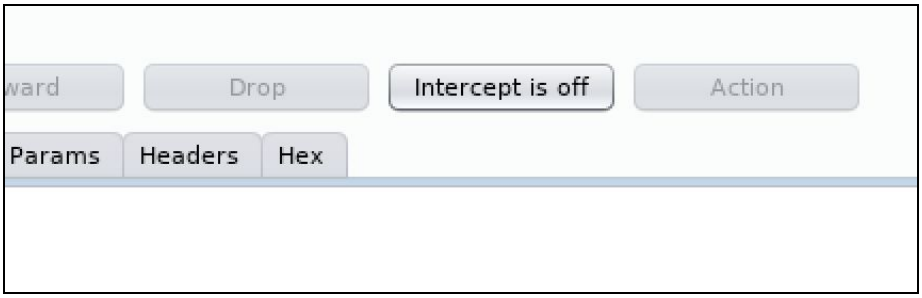


6. Go to Burp Suite and find your requests in the **Proxy->Intercept** tab.

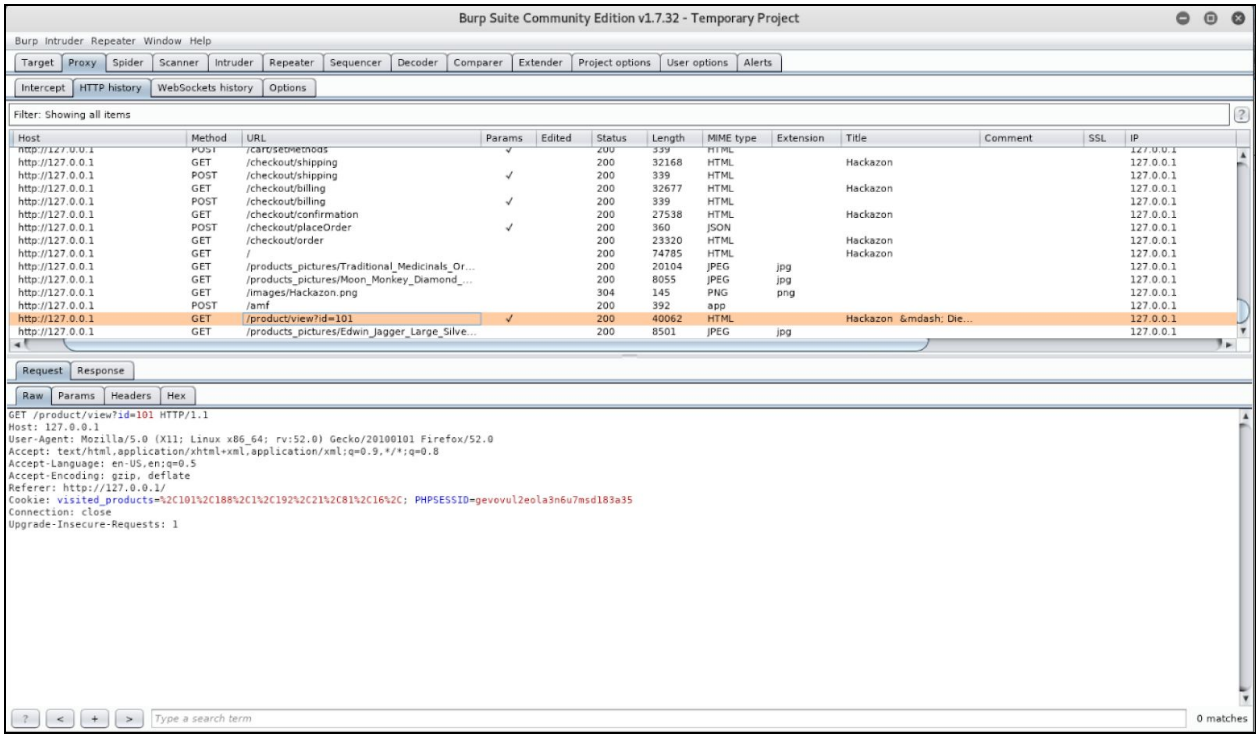


8. The request should start with something like `GET / HTTP/1.1` with several headers. You can modify any of these if you want, and then click Forward, this will forward your request to the server. The response will be sent to your browser. Alternate between Firefox and Burp Suite, forwarding requests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

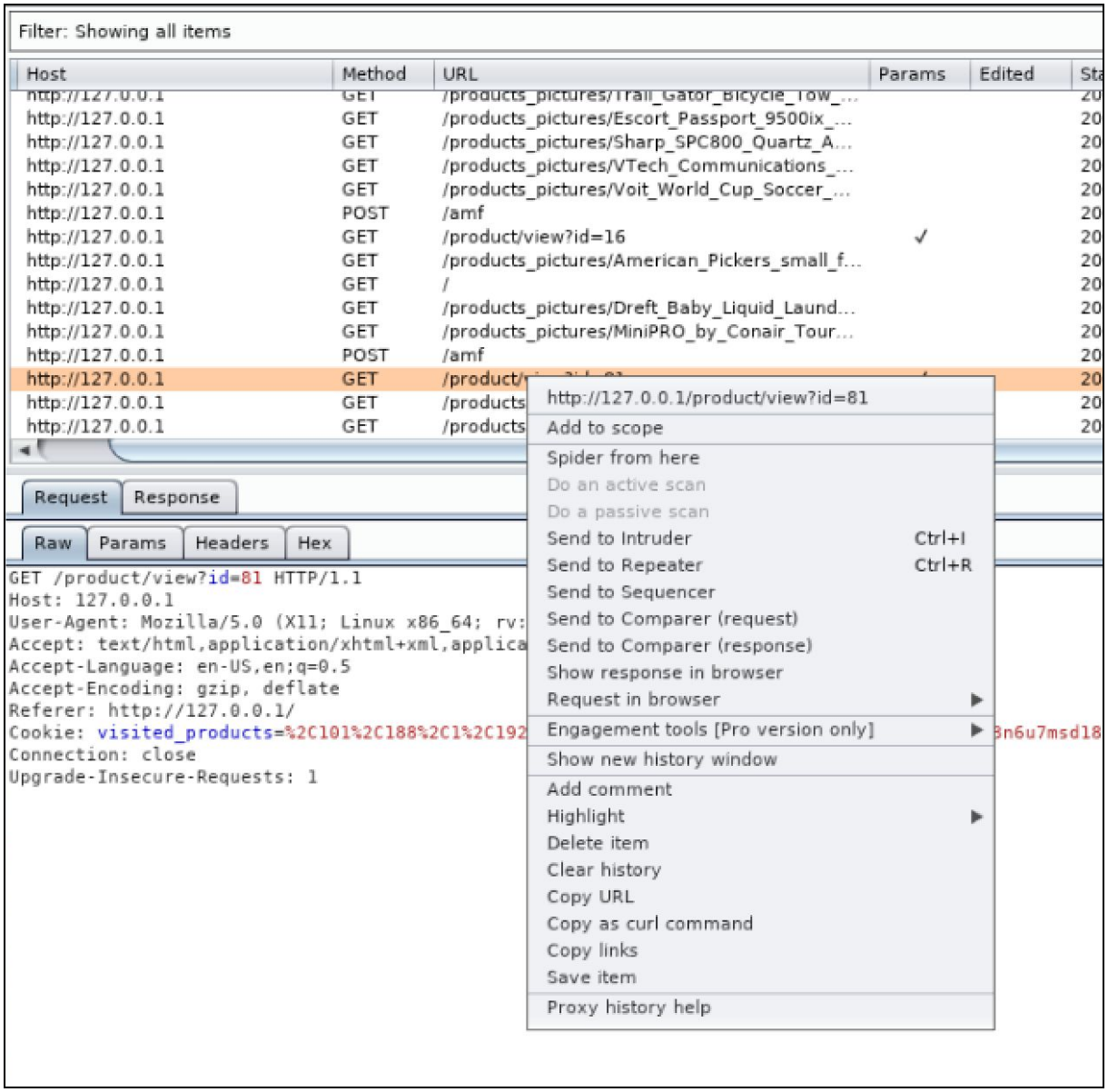


Ok, our target is hackazon.net, which is running on http://127.0.0.1.

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**"



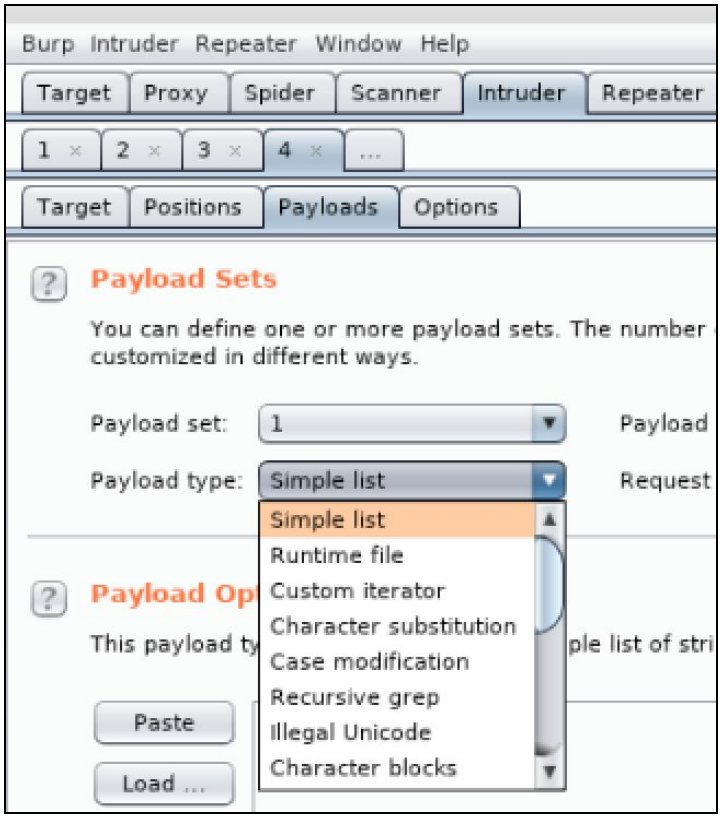
13. Move over to the "**Intruder**" tab and select the "**Positions**" tab.

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.

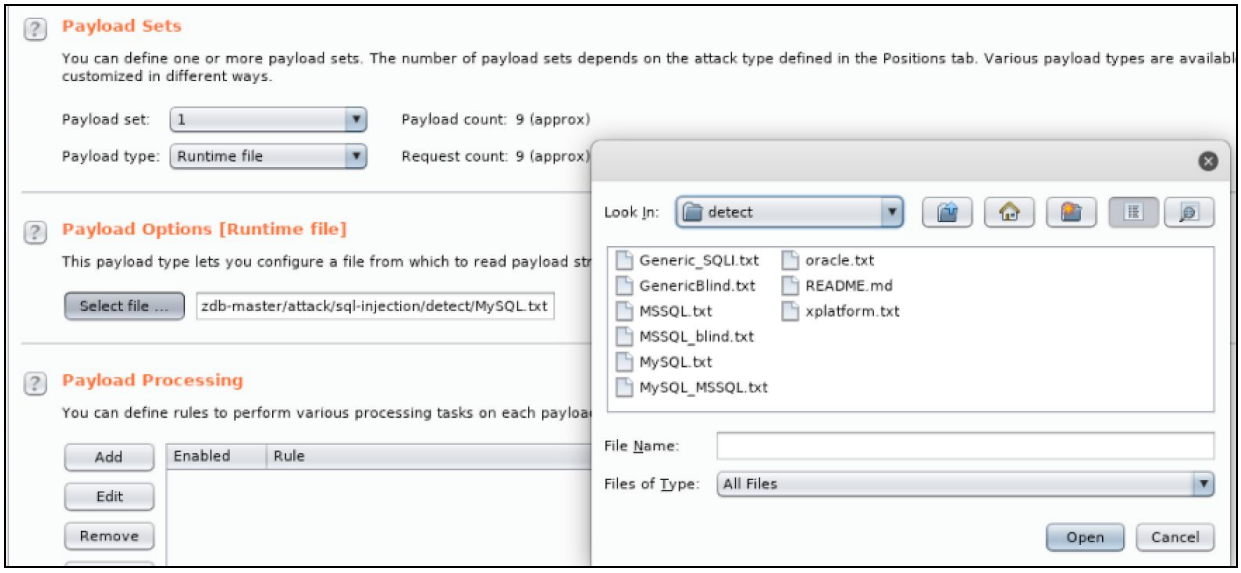
16. Click on the **Payloads** tab and navigate to the "**Payload type**" pull down menu.

17. Select "**Runtime file**" from the pull down menu.



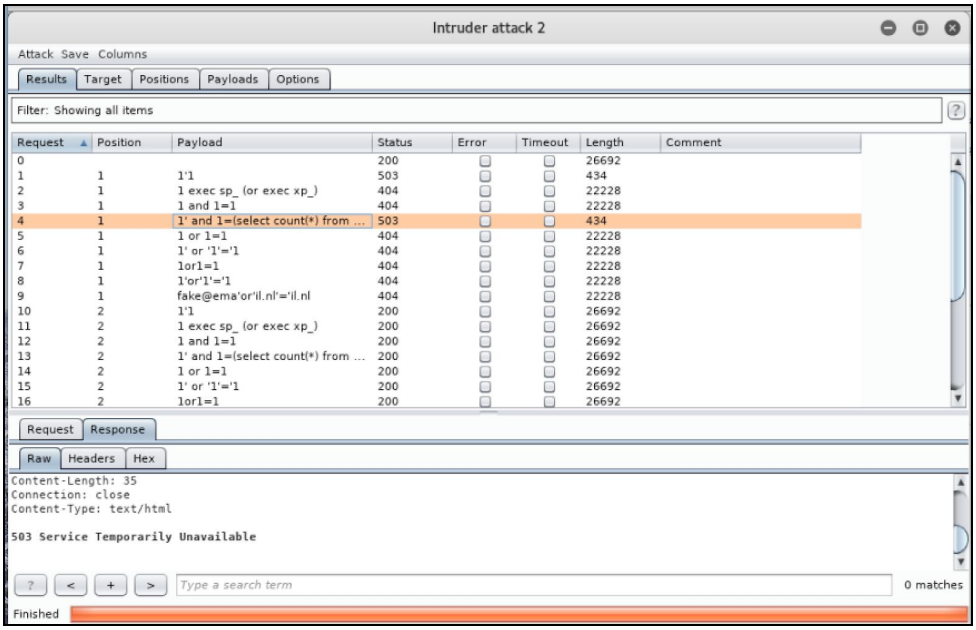
18. Click the "Select file" button, as demonstrated in the following figure. Select the file "*~/Downloads/fuzzdb-master/attack/sql-injection/detect/MySQL.txt*"

    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.



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.



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

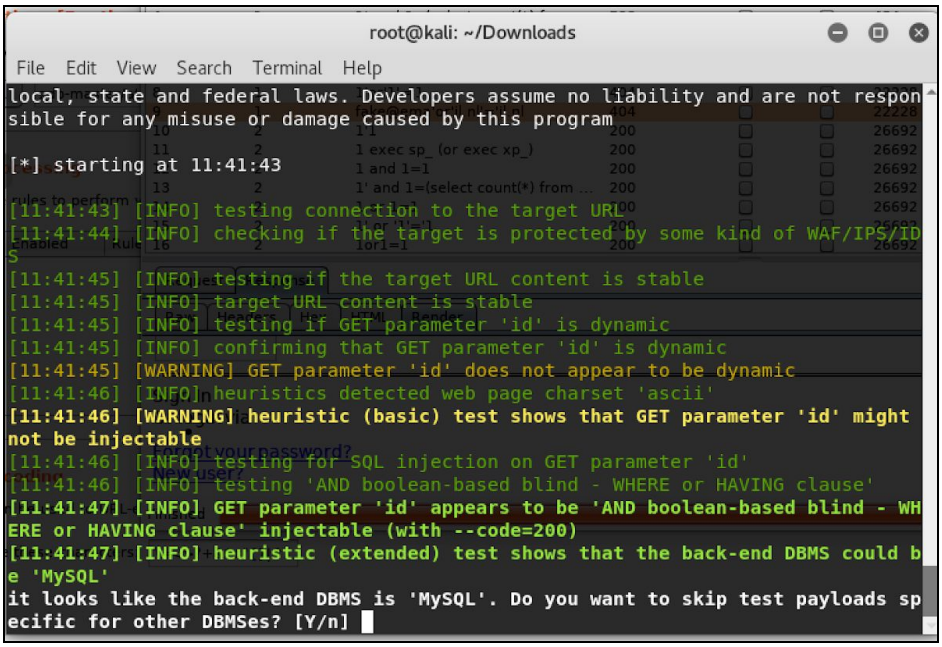
Tip: Now you know that the vulnerable application/site is vulnerable to SQL injection flaw in the URL *"127.0.0.1/category/view?id=2".*

19. Run the following command from the CLI:

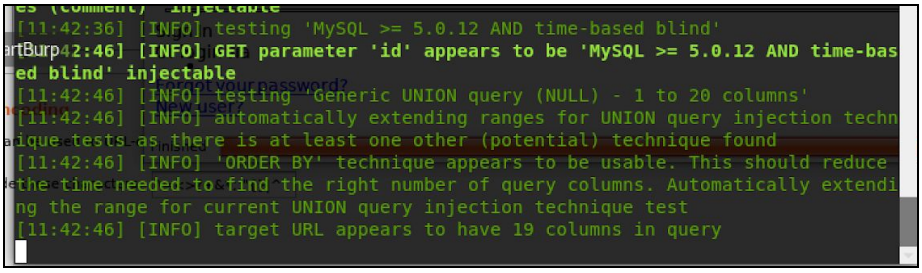
```
sqlmap -u http://127.0.0.1/category/view?id=2
```

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

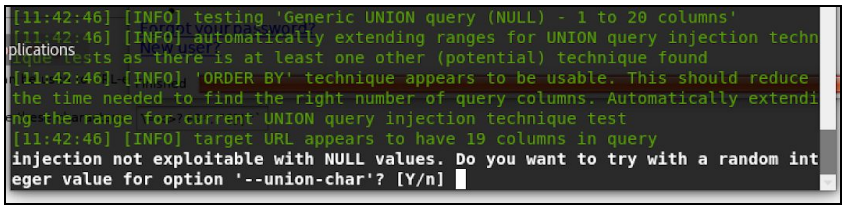
20. The tool will detect that the back-end database is a MySQL database. Select "**Y**" to skip payloads for other databases.



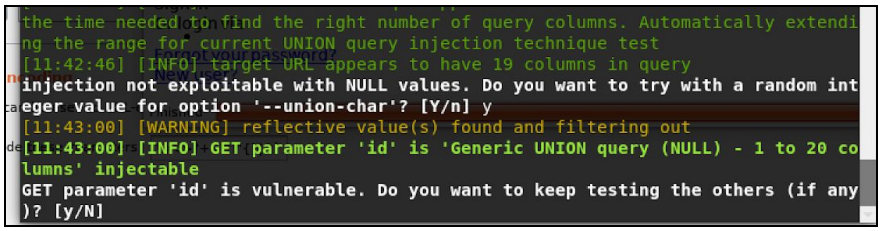
21. You should see the message shown below stating that the application is susceptible to blind SQL injection.



22. Answer **"Y"** to the question about trying injection with random integer values.



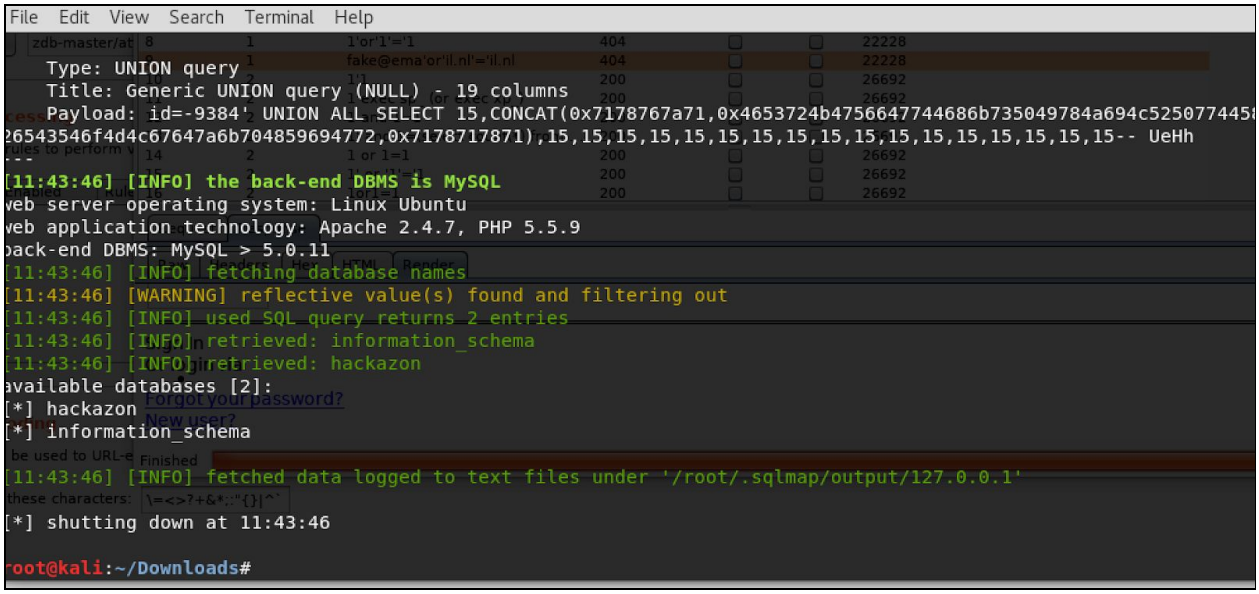
23. The next message confirms that the "**id**" parameter is indeed vulnerable to SQL injection and asks if you would like to test all other parameters. Optionally, you can continue with further testing or if you are running out of time you can answer "**N**" to stop testing here.



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
```

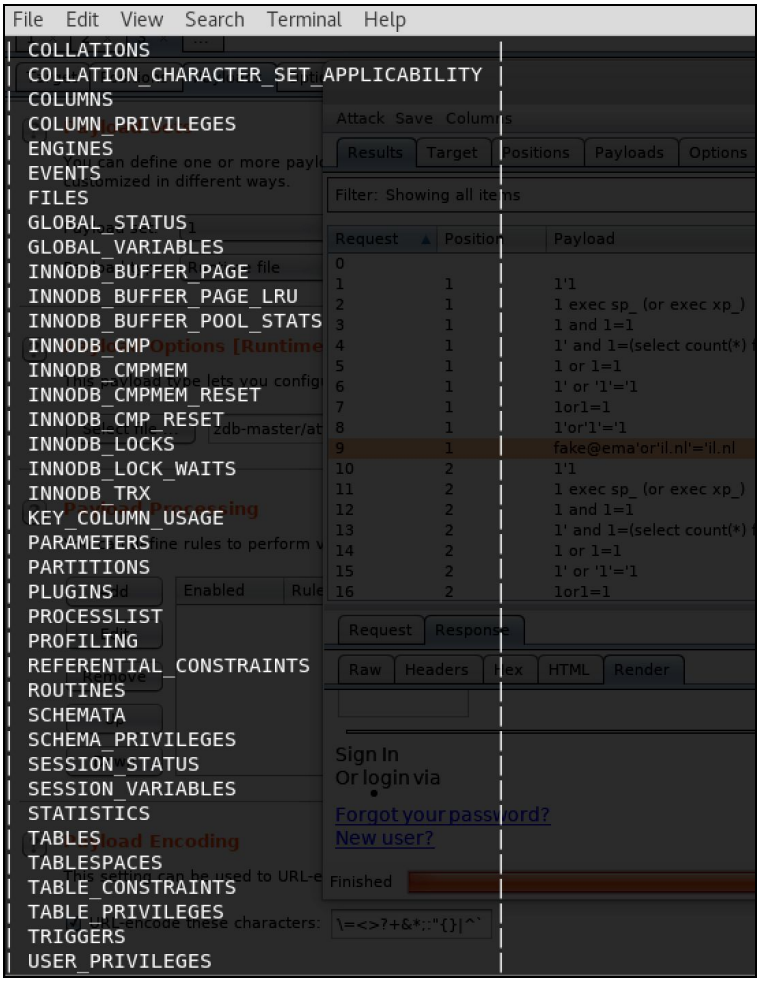
This command should tell you what databases are currently available on the server.



25. Run the following command:

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

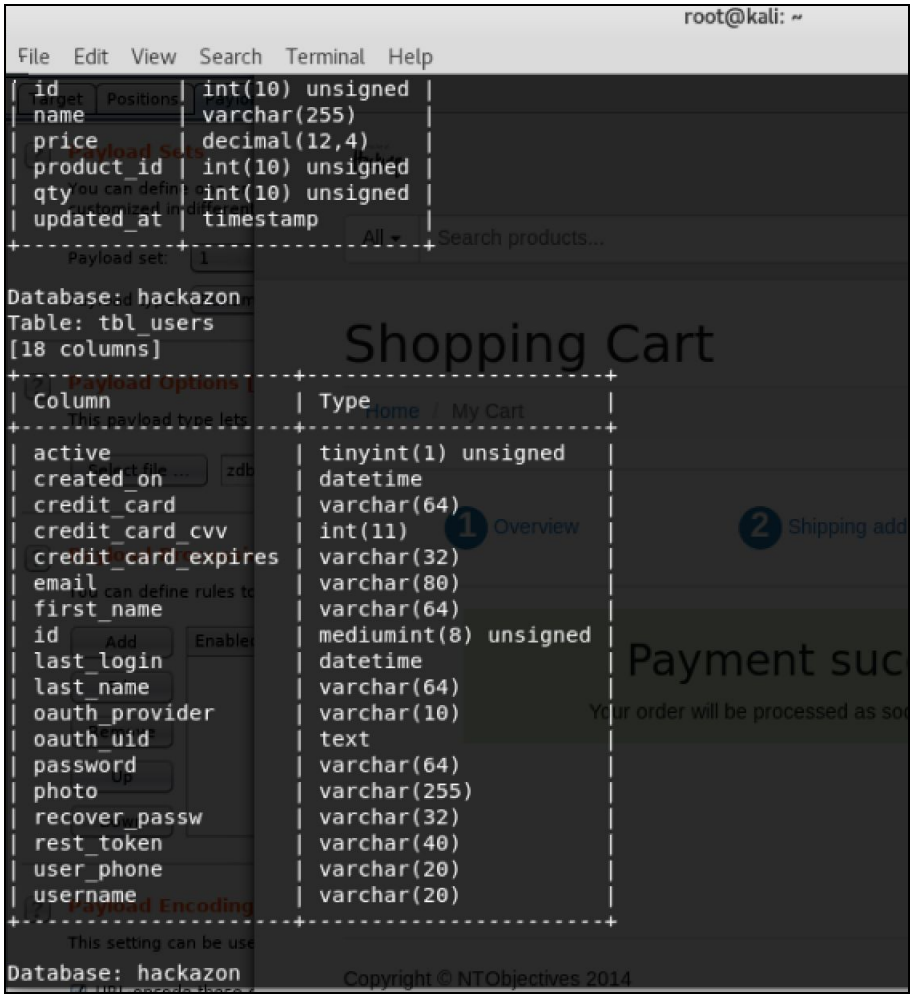
This command will dump the list of *tables* available in the *hackazon* database, as shown in the next figure.



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
```

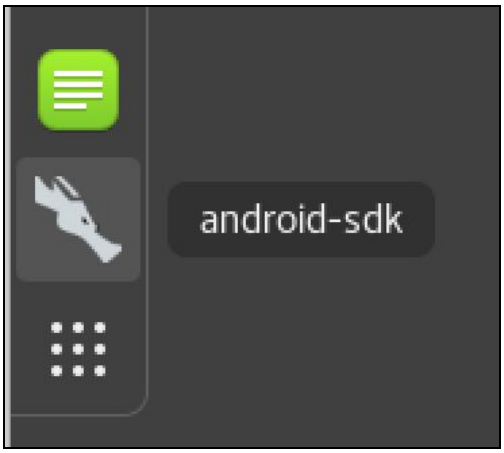
28. Answer **No (N)** to the question "*do you want to store hashes to a temporary file for eventual further processing with other tools*"
You can also perform a dictionary-based attack to crack the password hashes found in the database. If you are running out of time or if you have a slow system, answer **No (N).**

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

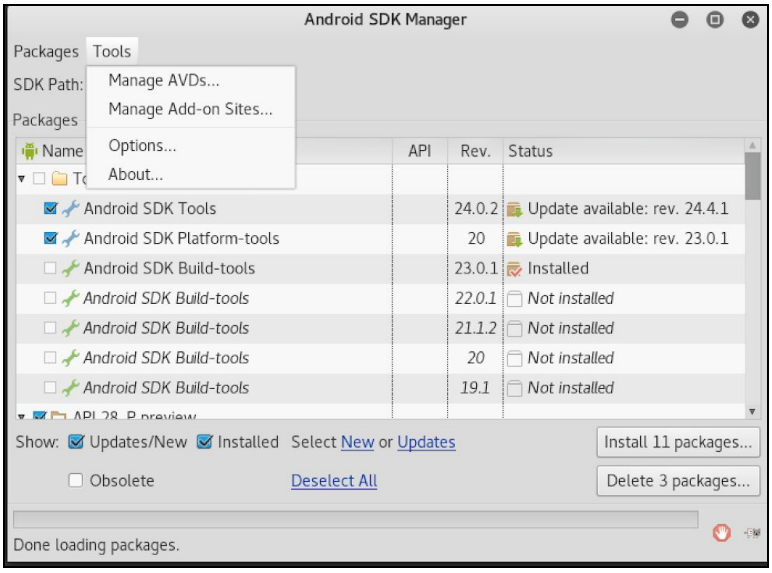
1. The Hackazon web application should be running, as you just completed the SQL injection exercise. However, verify that is running by browsing to http://127.0.0.1. If it is not, please follow the procedure outlined in the beginning of this document.

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)**.

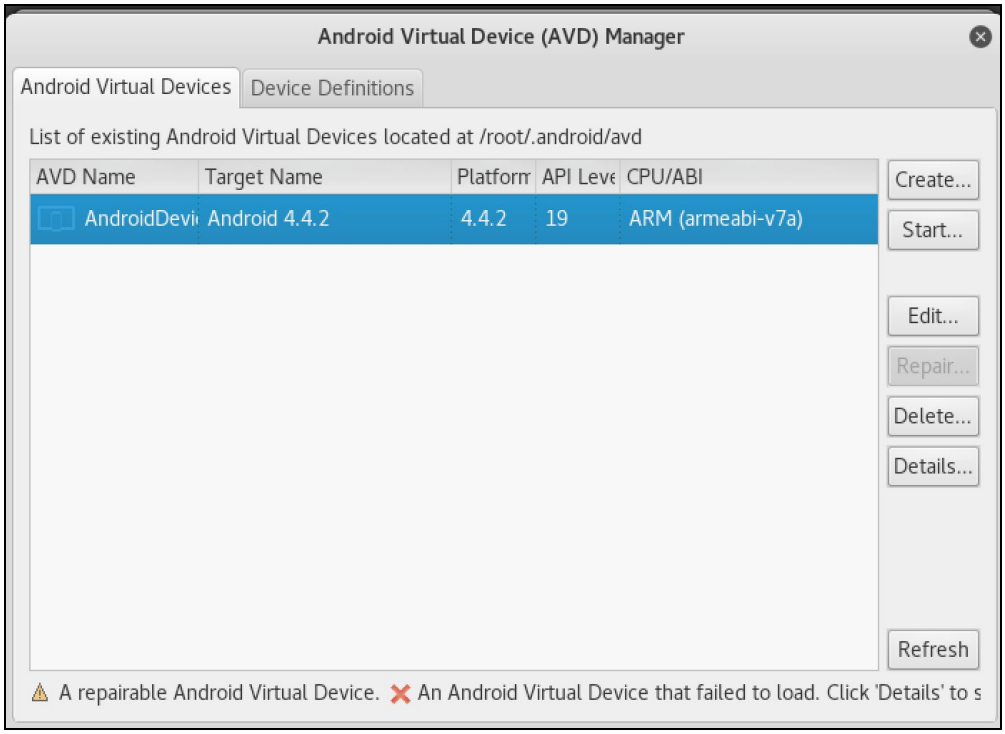


3. Once the Android SDK Manager is open, click on **Tools**.

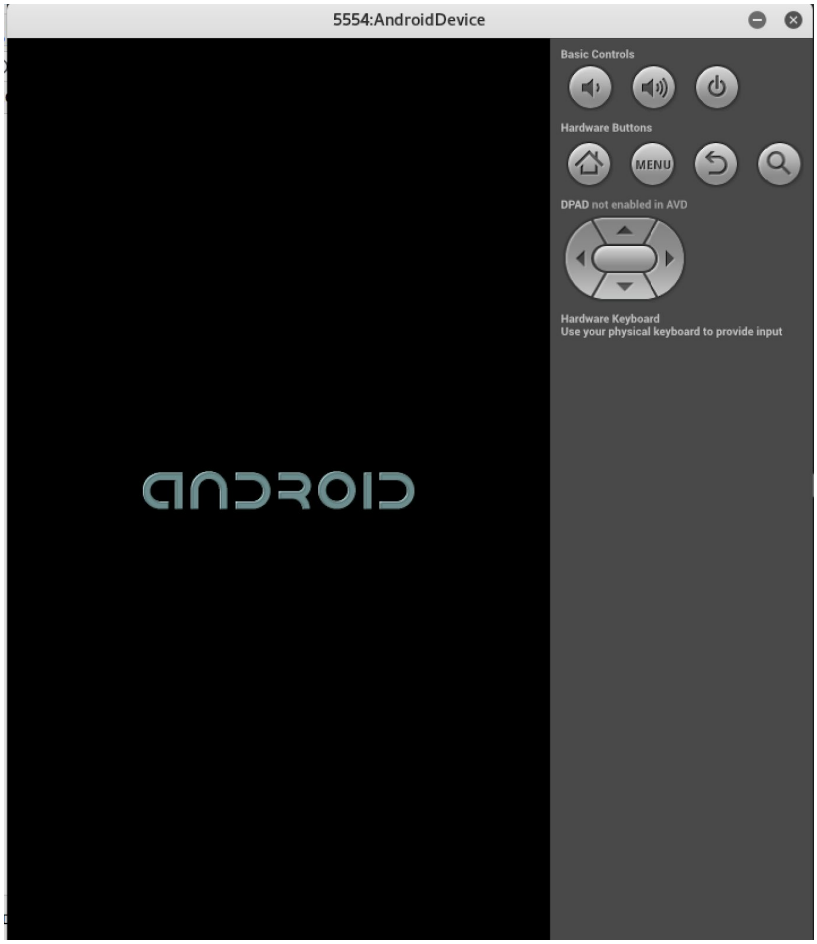
4. Select **Manage AVDs.** This will open up the **Android Virtual Device (AVD) Manager**.



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.

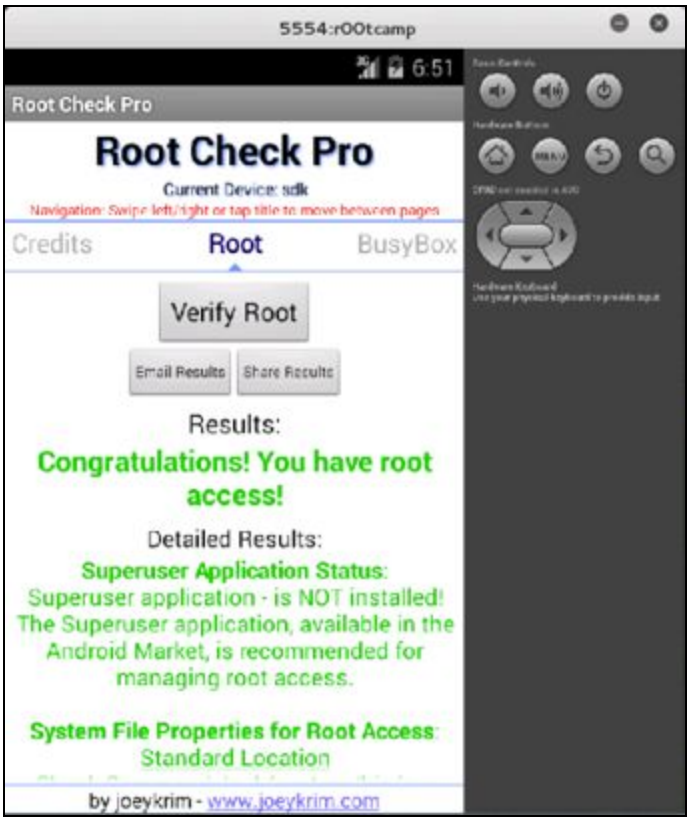


When the AVD has finished booting you will be at the home screen.

- 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.



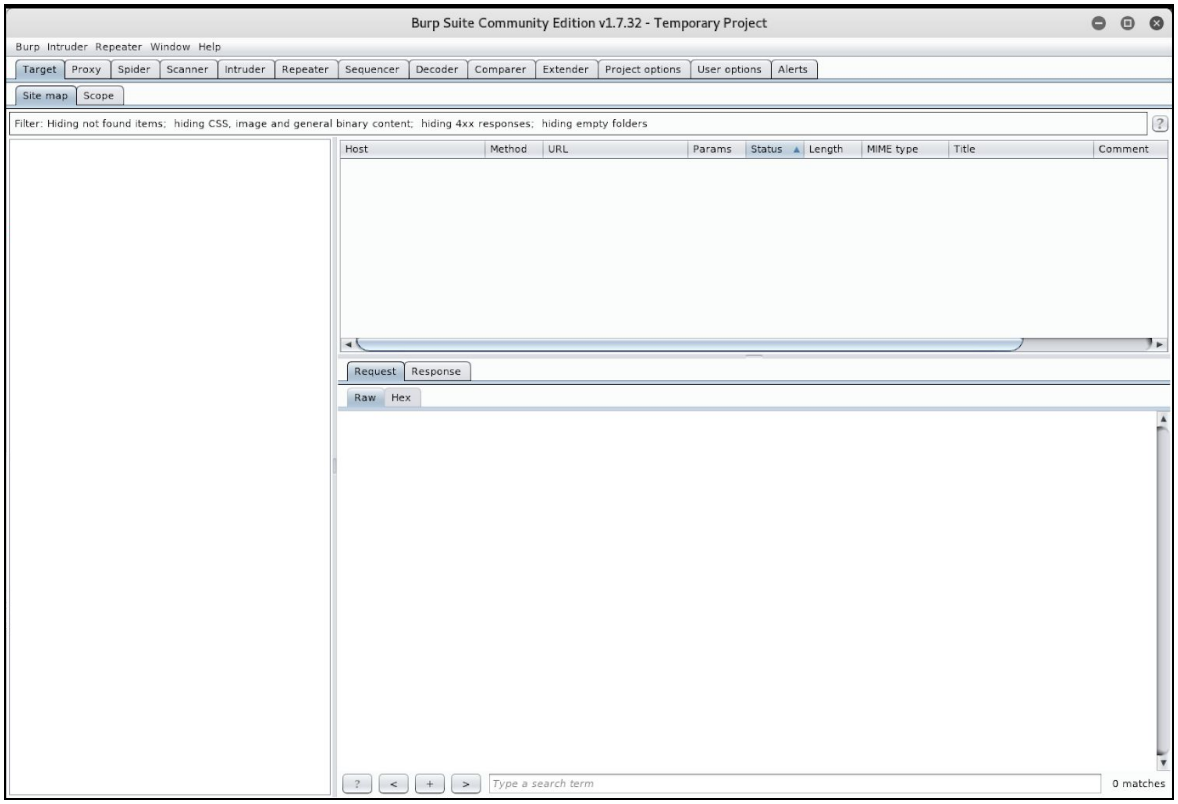
6. From here we want to fire up burp as we did in the previous lab by running the command "**`burpsuite`**" from the terminal window.

```
root@kali# burpsuite
```

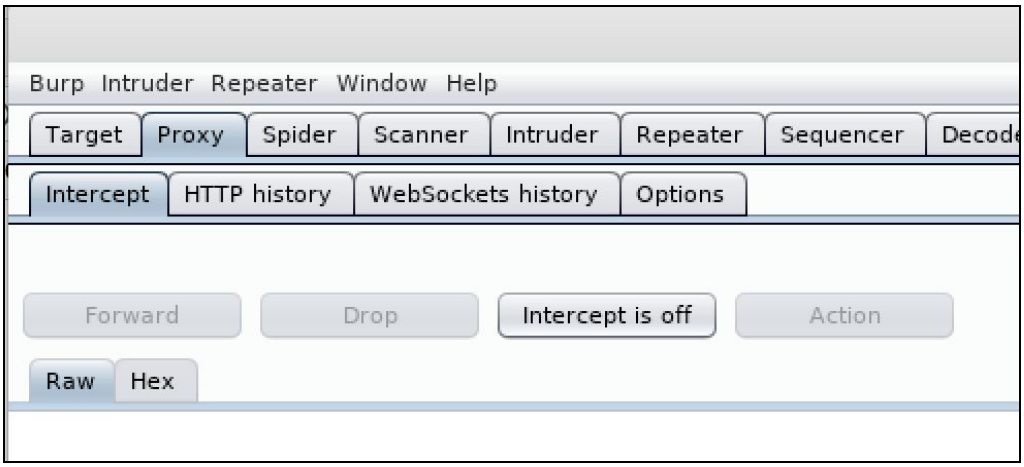
7. Click **Ok** to the message about the JRE version. Also, click **Close** if asked to update burp suite.

8. Click **Next** to open a temporary project. Then click the "**Start Burp**" button.

You should now be at the main BurpSuite screen.

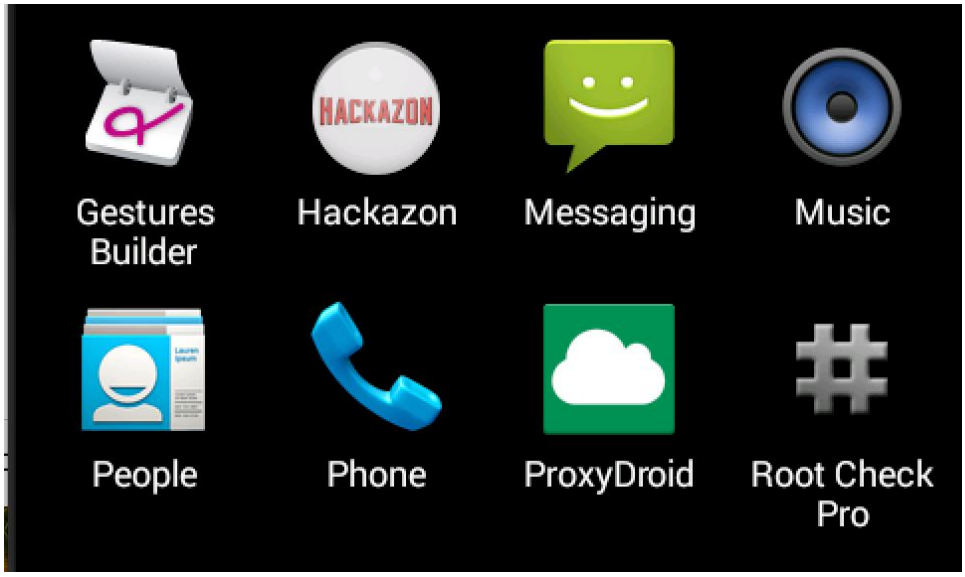


9. Double check that BurpSuite Proxy **Intercept is off**.

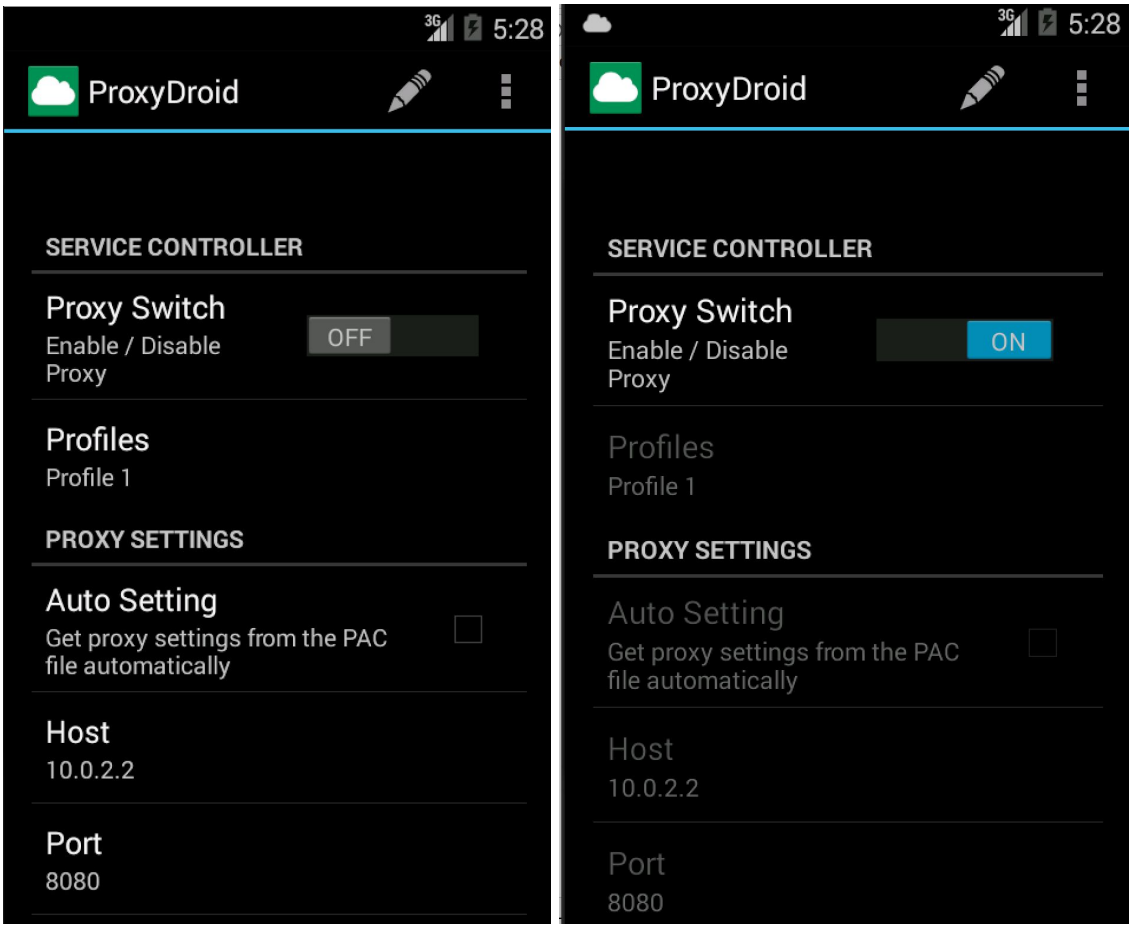


10. Before we start our application, we need to tell the AVD to send all traffic through the Burp Suite proxy.

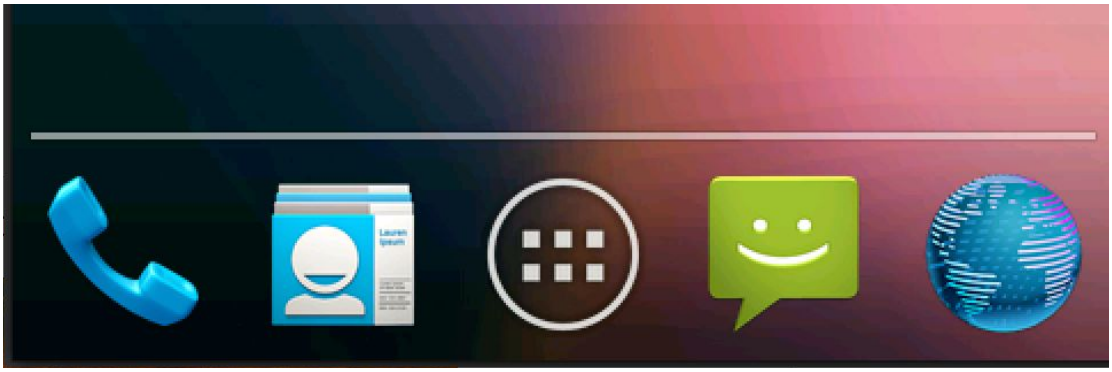
11. Move back to the **Android ADV** and click on the Applications icon at the bottom. Click on the "**ProxyDroid**" application.



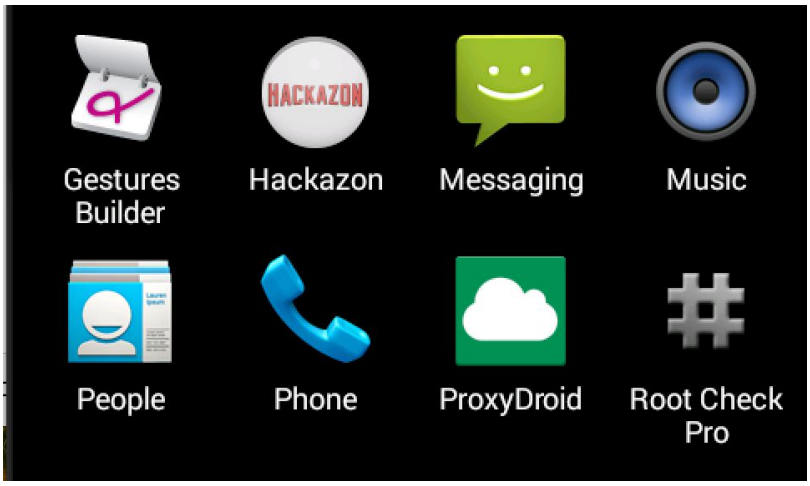
12. From there, click on the toggle to switch it to on.



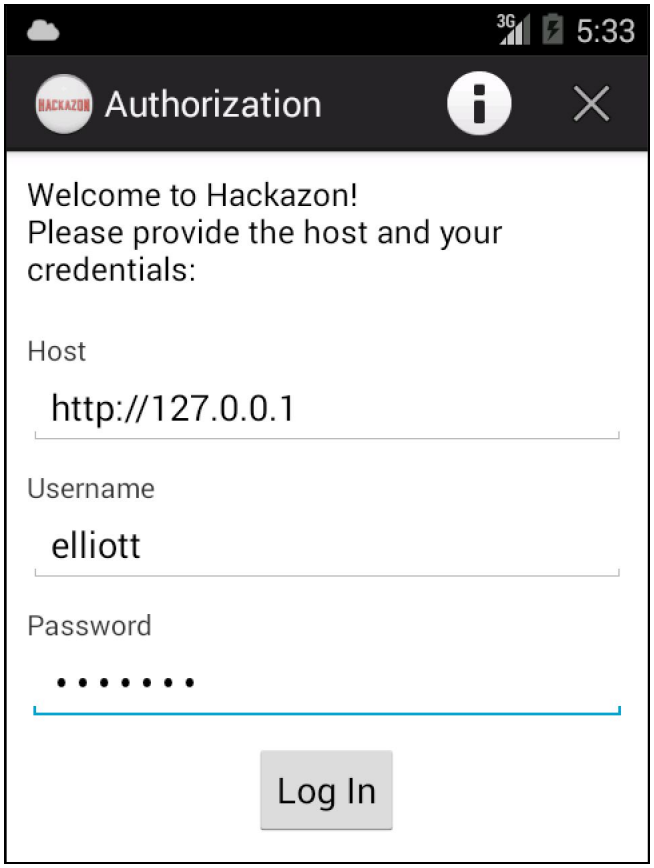
12. Start the Hackzon app from the Android device desktop. Move back to the Android ADV desktop by clicking the home button on the right control panel. Then click on the **applications icon** at the bottom.



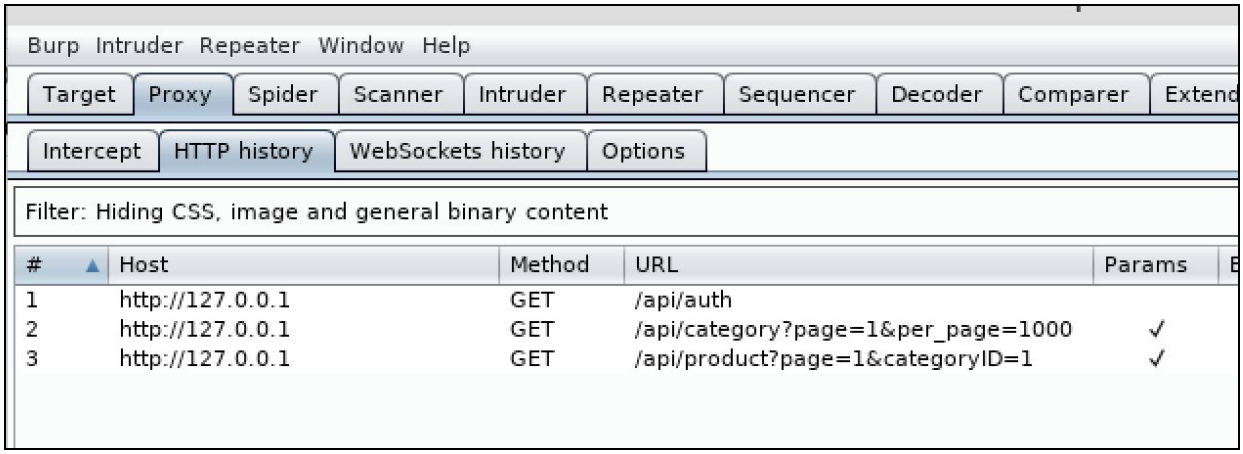
13. Click on the **Hackazon Application**.



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



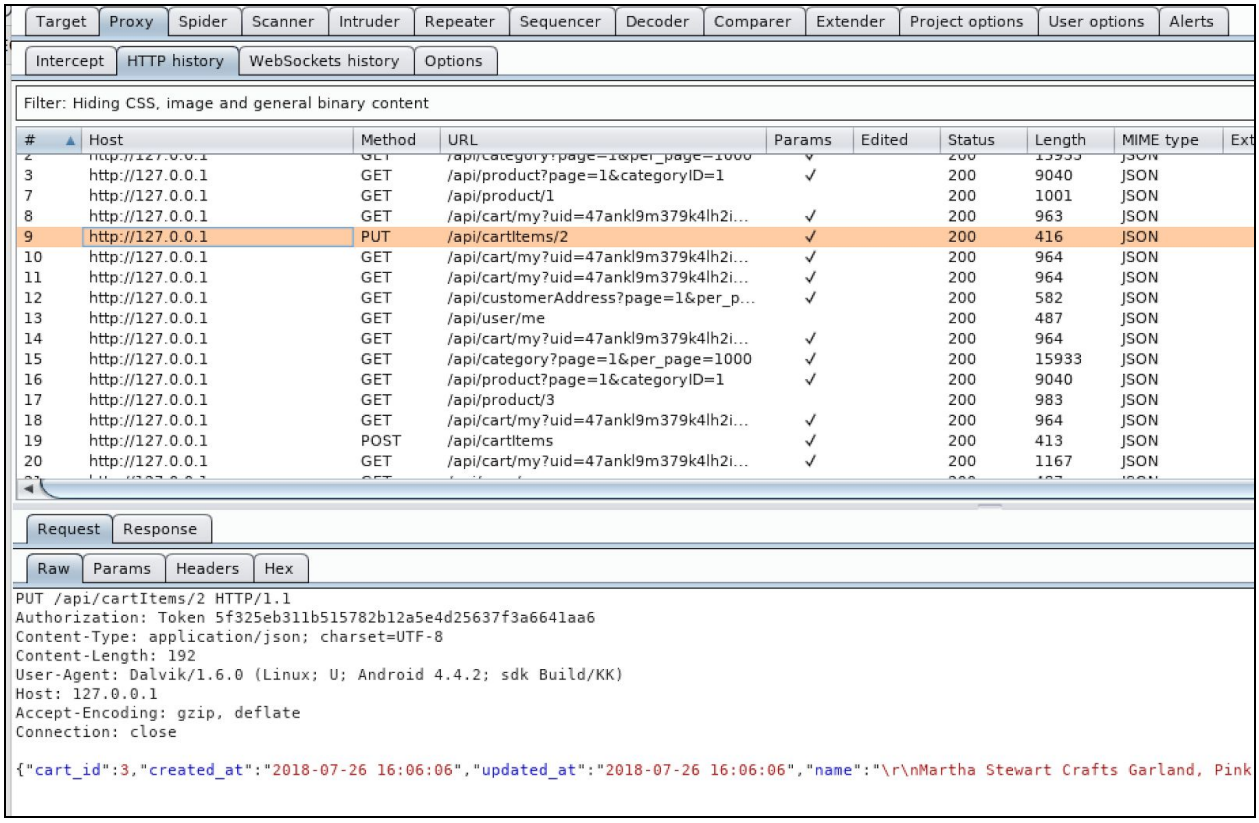
14. To verify that the traffic is being sent to the BurpSuite Proxy, by navigating to the **Proxy** tab, and then selecting the **HTTP history** tab. You should see some traffic from the mobile application to the Hackazon server.



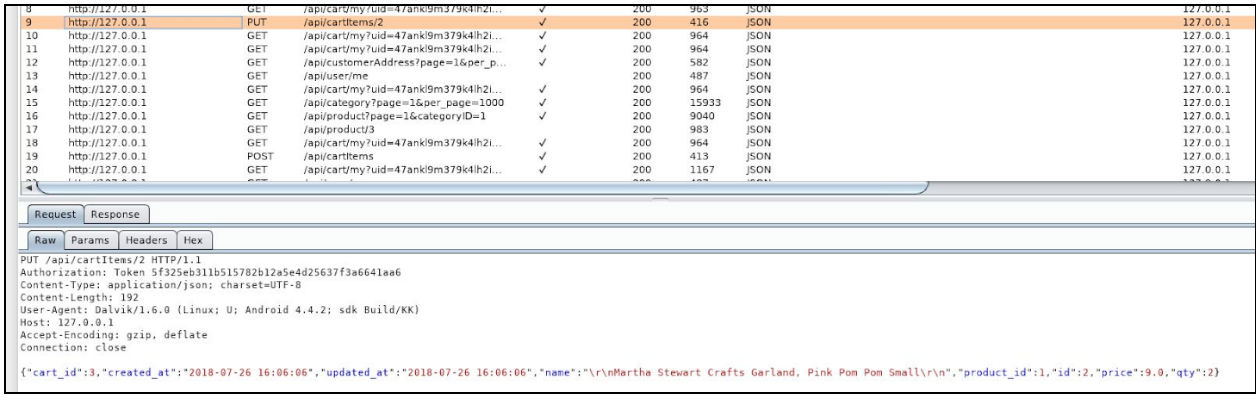
15. To expose the API to BurpSuite, run through some functions in the application (for example, add things to the cart, view the cart, submit, etc.). Go back to the AVD and explore the Hackazon application for a few minutes.

16. Be sure to add something to your cart.

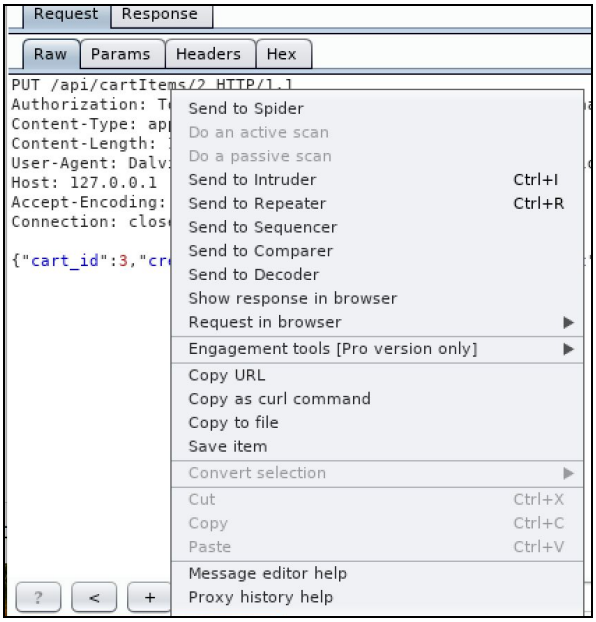
17. Go back to the **Proxy > HTTP history** tab in Burp and look for interesting HTTP requests and responses.



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.

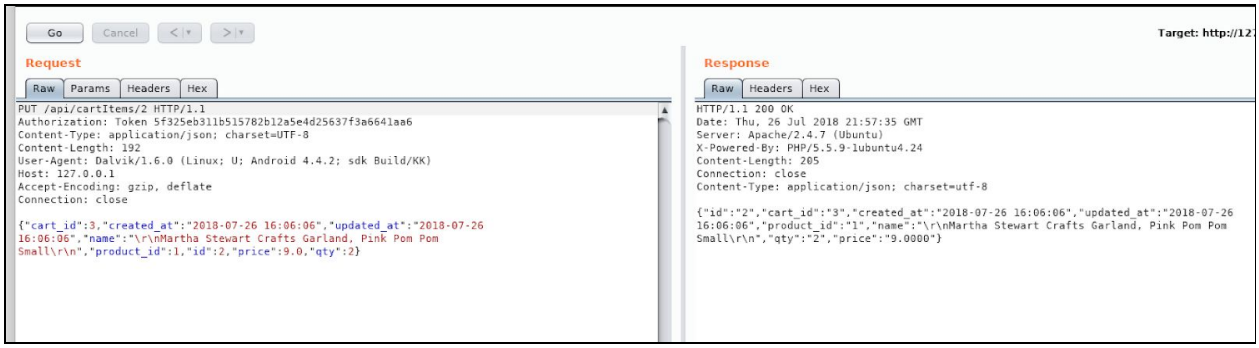


17. The price is being sent in the POST request. Send that request to **Repeater** by **right clicking** and selecting **Send to repeater**.

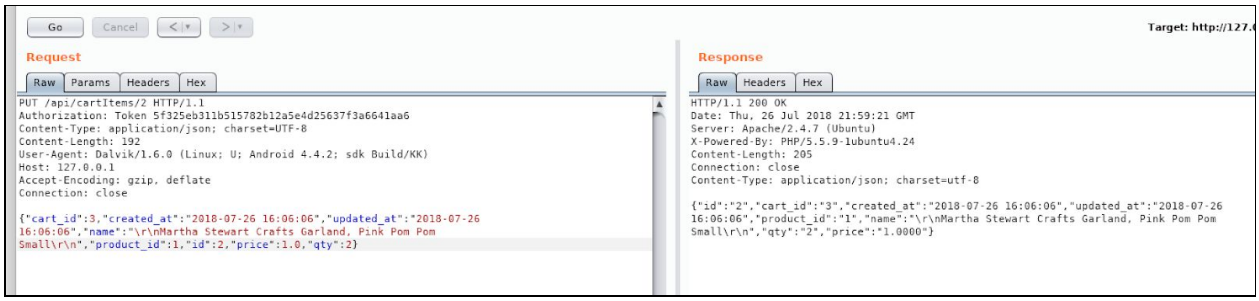


18. Navigate to to the **Repeater** by clicking the **Repeater** tab at the top of the screen.

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

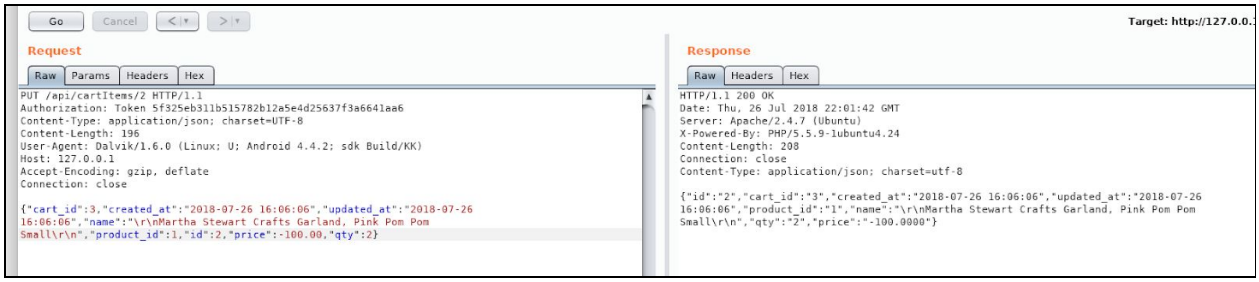


19. Try to modify some information and see if you can get it to pass through. Modify the price from **9.0** to **1.0**. Then click **Go** again.



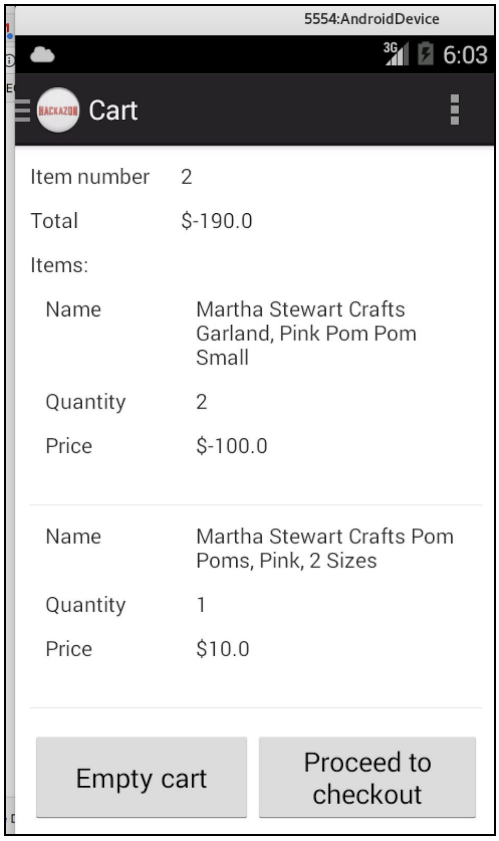
20. You should see a **200 OK** message in the response. This indicates that you are able to modify the price and that the request was ok. Definitely a severe vulnerability in the application.

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



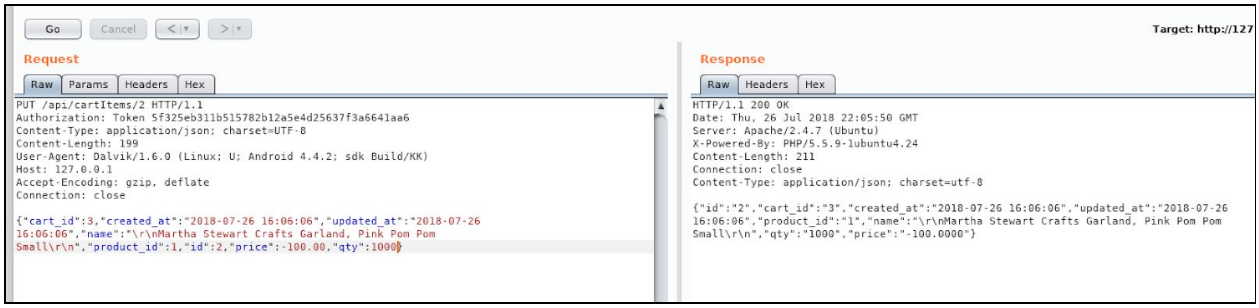
21. You should see a **200 OK** response.

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

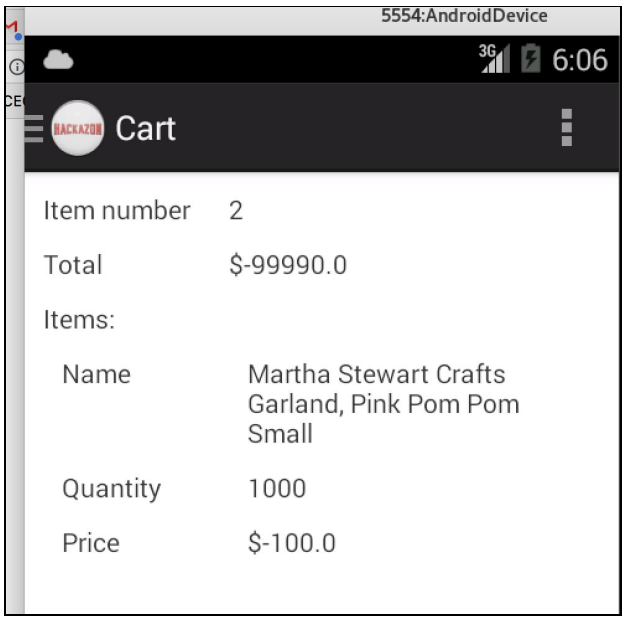


The cart shows you that you are now getting 2 of these items for -**$100**. This is a pretty good deal ;-)

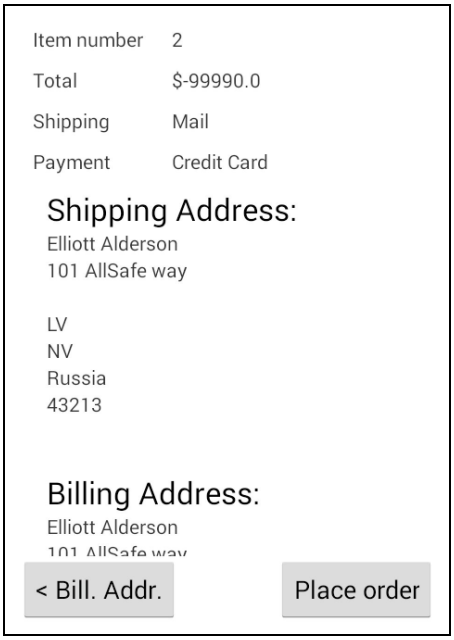
22. Go back to **BurpSuite's Repeater** and try modifying something else (like the item quantity, etc). This time change the quantity to **1000** and click **Go**.



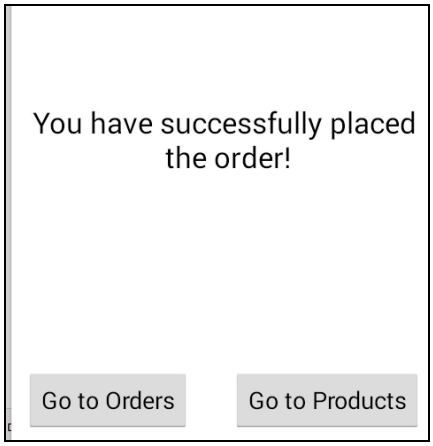
23. If you back to the Hackazon app and refresh the cart, you should now see that you are getting a real good deal! ;-)



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.



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 ;-)

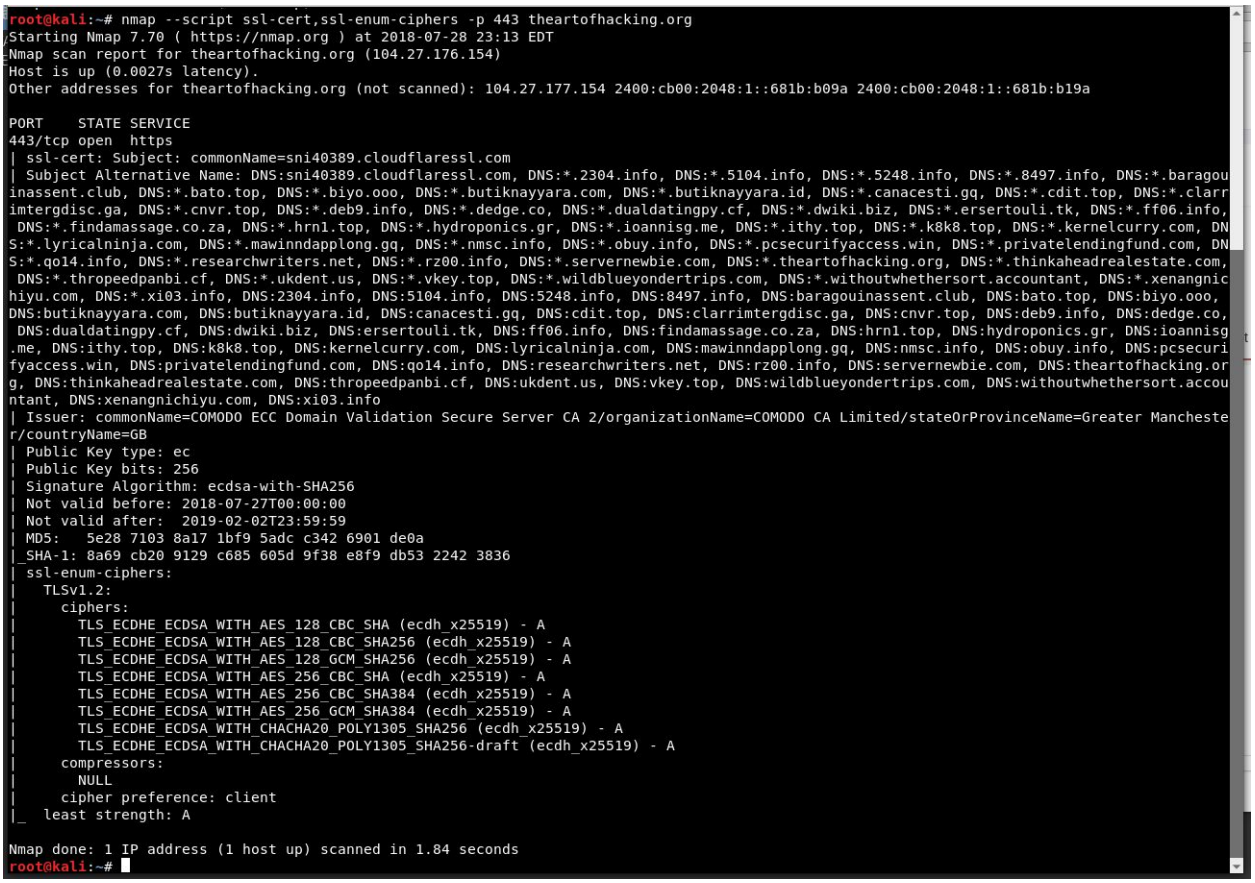


Exercise 7: Exploiting Weak Cryptographic Implementations

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
```



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)
```

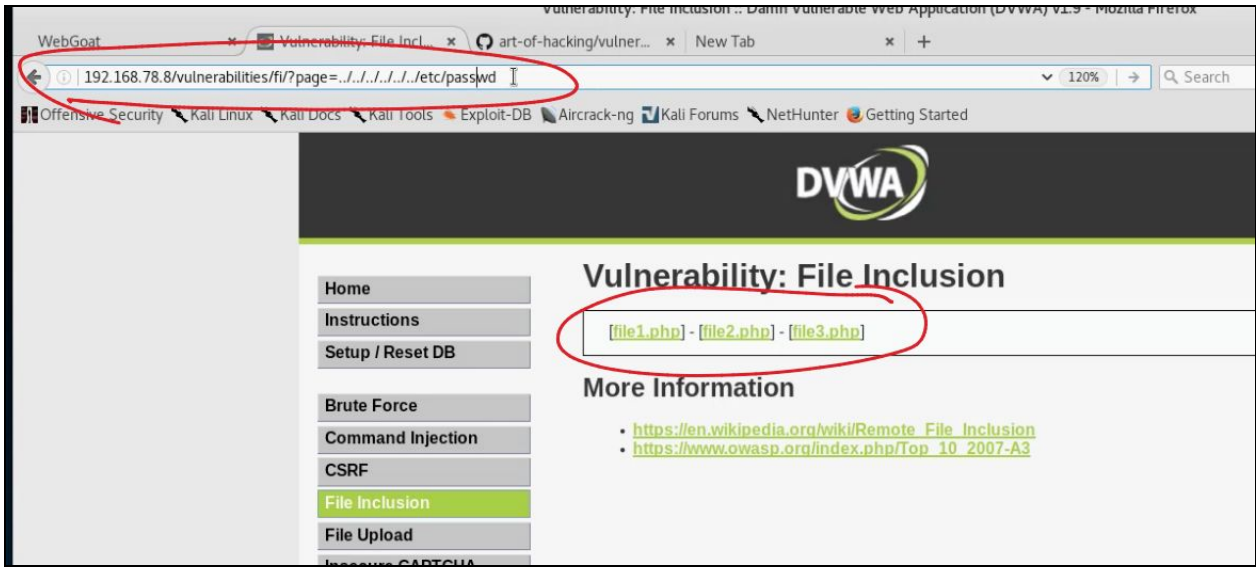
```
 Weak 128 Bit ciphers (SEED, IDEA, RC[2,4])    not offered (OK)
 Triple DES Ciphers (Medium)                   not offered (OK)
 High encryption (AES+Camellia, no AEAD)       offered (OK)
 Strong encryption (AEAD ciphers)              offered (OK)

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

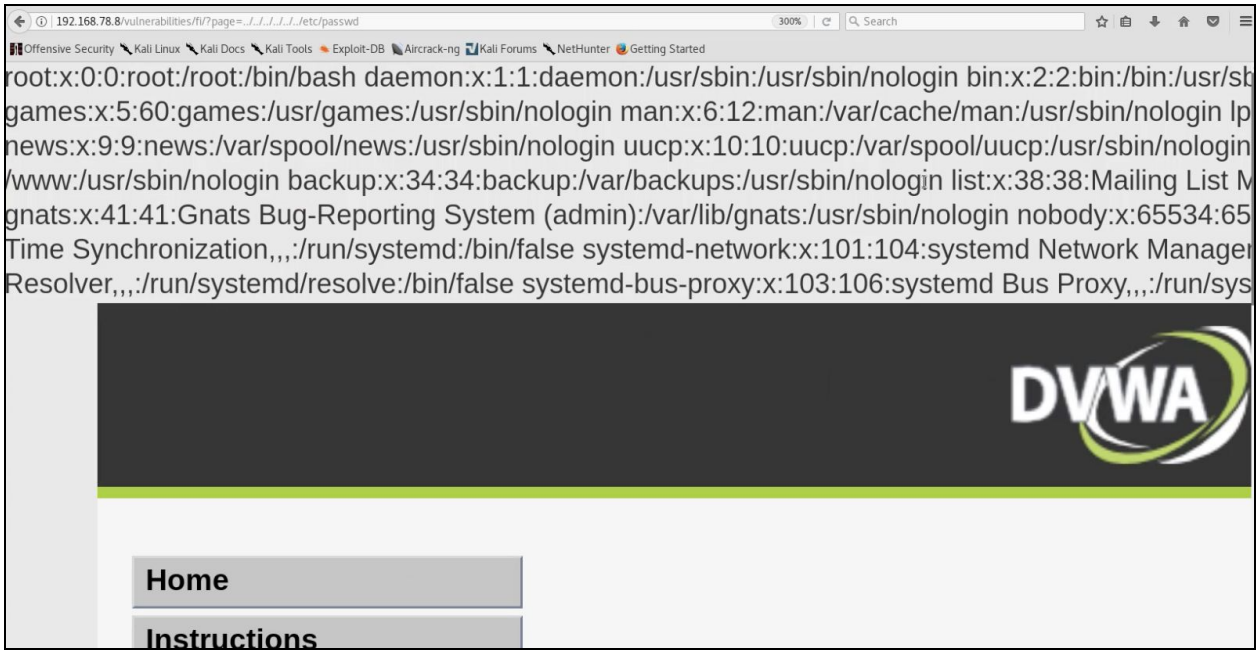
1. Go to the Damn Vulnerable Web Application (DVWA) in WebSploit and navigate to **File Inclusion**.

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: 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 o♪ and om having fun   and omar
sucık
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...