


# Beginner's Guide to sqlmap





This ebook is a collection of lessons from our course, [Beginner's Guide to sqlmap](#). We know some people may prefer the format of an ebook over a series of lessons online, or of written versus video lessons. In any case, we hope you enjoy it!



---

## Table of Contents

1. [About the course](#)
  - a. About the course
  - b. About sqlmap
  - c. Pre-requisites
2. [Getting started](#)
  - a. Creating a home lab environment
  - b. Downloading and installing sqlmap
  - c. Using sqlmap for the first time
3. [Finding and exploiting SQL injections](#)
  - a. Your first SQL injection with sqlmap
  - b. Extracting passwords from a database
4. [Next steps](#)
  - a. Continue building sqlmap skills



---

# About the course

## About the course

One of the most devastating and common web attacks today is SQL injections.

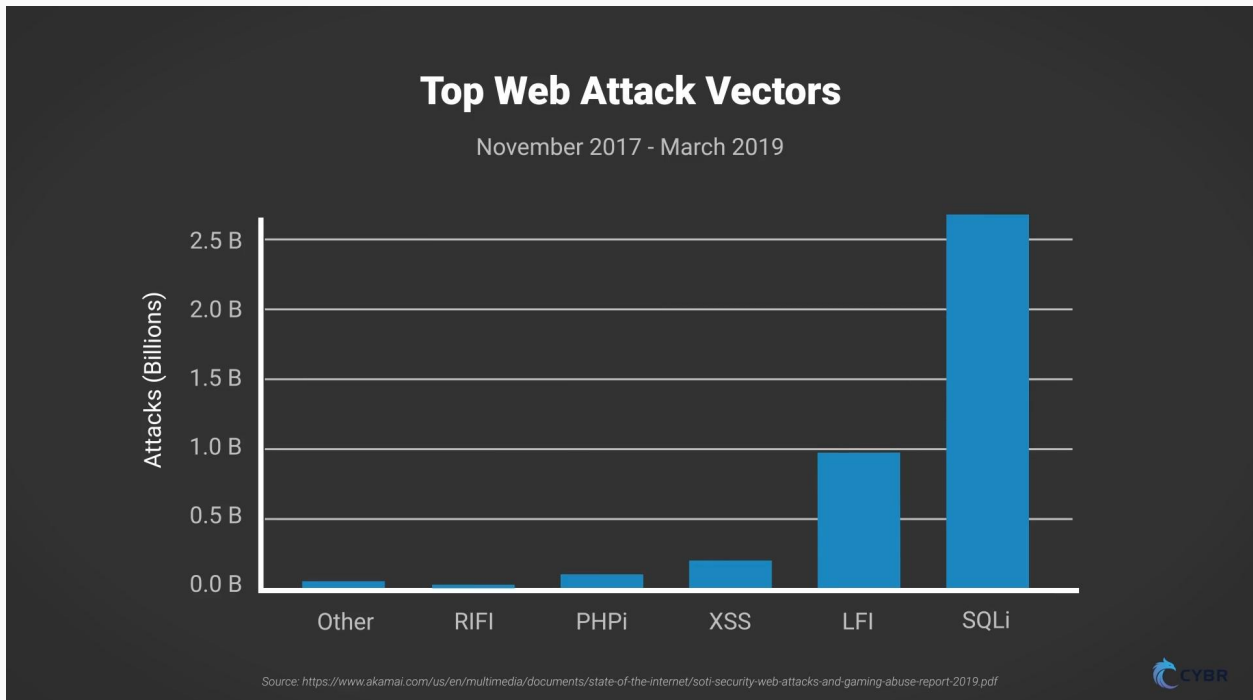
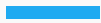
They have been around for years and they've been the result of some of the most serious data breaches in history.

Why? Because databases are at the heart of web applications, and they are filled with valuable data.

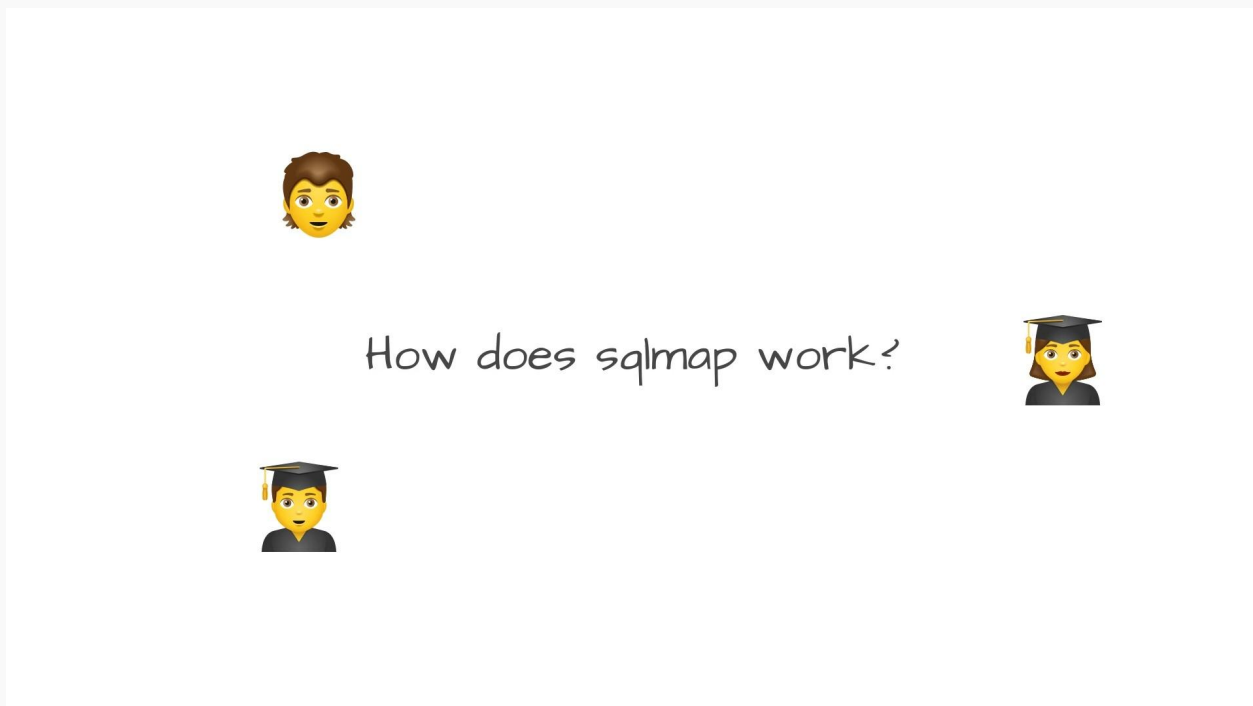
So you might think that all organizations and all web applications would make defending against these attacks a big priority, but that's simply not the case.

SQL injections are still prevalent. In fact, an Akamai report tracking the top web attack vectors from November 2017 to March 2019 found that SQL injections represent nearly two-thirds of all web application attacks.





So, a few months ago I launched an SQL Injection Attacks course that quickly gathered over 17,000 students. Then, I started getting questions about sqlmap, so I decided to make this course.



---

In this course, we start out by learning what sqlmap is, then we move on to creating a home lab environment so that we can use sqlmap against vulnerable environments and find our first SQL injections. We'll also take a look at some of the common and most useful features sqlmap has to offer for beginners.

Because I show you exactly how to set up your own home lab, you'll be able to follow along with everything that I do. You will have the exact same environment and the exact same tools to work with that I do, and we do all of that for free.

By the way, once you complete this introductory-level course, we have a more advanced sqlmap Deep Dive course that you can check out and use to take your skills to the next level. (Might still be in development depending on when you're watching this! But be sure to check it out)

So let's get started and I'll see you in the course!

## About sqlmap

Before we jump in and learn how to use sqlmap, we need to have a good understanding of what it is and, just as importantly, what it's not.

sqlmap is an automatic SQL injection and database takeover tool. It's an open source tool, meaning that you can view its entire codebase and this codebase is maintained by a group of contributors including the original authors. You can contribute to the project, and you could even create your own version of it if you wanted to. That's the beauty of open source.



**sqlmap®**  
Automatic SQL injection and database takeover tool

View project on GitHub

### Introduction()

sqlmap is an open source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over of database servers. It comes with a powerful detection engine, many niche features for the ultimate penetration tester and a broad range of switches lasting from database fingerprinting, over data fetching from the database, to accessing the underlying file system and executing commands on the operating system via out-of-band connections.

```
python sqlmap.py -u "http://debiandev/sqlmap/mysql/get_int.php?id=1" --batch
```

Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[\*] starting @ 10:44:53 /2019-04-30/

[10:44:54] [INFO] testing connection to the target URL  
[10:44:54] [INFO] heuristics detected web page charset 'ascii'  
[10:44:54] [INFO] checking if the target is protected by some kind of WAF/IPS  
[10:44:54] [INFO] testing if the target URL content is stable  
[10:44:55] [INFO] target URL content is stable  
[10:44:55] [INFO] testing if GET parameter 'id' is dynamic  
[10:44:55] [INFO] GET parameter 'id' appears to be dynamic

Download zip file

Download tar.gz file

Proudly sponsored by SpyderSec

Tweets by @sqlmap

CYBR

## sqlmap Features

sqlmap is designed as a penetration testing tool that automates the process of detecting and exploiting SQL injection flaws. Once SQL injection flaws are found, this tool can help you exploit the vulnerabilities to their maximum extent by using a large number of different features. In some cases, sqlmap can help you completely take over database servers.

In terms of SQL injections, this really is a powerful tool. It includes features like:

- **Database fingerprinting** - gather information about what database engine, version, etc, an application is using, which is really useful for reconnaissance
- **Full support for major database engines** - like MySQL, Oracle, Postgres, SQL Server, SQLite, MariaDB, Redshift, and a bunch of others
- **Full support for 6 SQL injection techniques** - boolean-based blind, time-based blind, error-based, UNION query-based, stacked queries, and out-of-band
  - If you're not familiar with what these techniques are, how they work, or what they do, then be sure to check out my free Injection Attacks course where we cover all of these concepts in great detail
- **Support to enumerate users, password hashes, privileges, roles, databases, tables and columns** - enumeration gathers additional information from the database in order to then



exploit it and hopefully (at least if you're on the red team), reveal all of that information I just mentioned

- **Support to download & upload files, and execute arbitrary commands on the database server**
- **Support to establish out-of-band connections between an attacker machine and the database server** - which is a more advanced technique that makes connections from the database server to your own remote server in order to retrieve results or execute commands

There are other features that we didn't mention, but those are some of the big ones.

## What sqlmap *isn't*

While it sounds like a magical tool, and to some degree it is pretty magical, it's not a silver bullet. For one, while it automates *a lot*, it still requires human interaction and human skill to point it in the right direction and to properly configure it.

Just like any other tool, it also has its limitations. So while it's a great tool to have in your arsenal and any web pentester should absolutely learn how to use it, it's not a silver bullet, meaning that you can't just run this tool, find nothing, and then call it a day. That won't guarantee that there aren't any SQL injections, it will just help speed up the process and potentially find vulnerabilities that you wouldn't have manually found.

It can also be a tool that you add to your automated workflow when deploying new code, which again, can help augment your team and help as part of your DevOps pipeline.

## sqlmap FAQ

I saw these questions when doing a quick Google search for sqlmap, so I figured we could kick off the course by answering them since you likely also have these questions:





### People also ask :

Is SQLmap illegal?



What is the purpose of SQLmap?



Can SQLmap be traced?



What is SQLmap in Termux?



Feedback

### Is sqlmap illegal?

Short answer is no - sqlmap is perfectly legal. The long answer is 'it depends.' Just like buying, owning, and using a knife in and of itself is perfectly legal, going and stabbing someone is not legal.

So, the moral of the story is that how we're going to use sqlmap in this course is perfectly legal. However, if you start to try and use sqlmap against environments that you do not have explicit written permission for, then you enter illegal territory and you should not do that. Always have permission.

### What is the purpose of sqlmap?

We already answered this one so I'll skip it.

### Can sqlmap be traced?

This answer also depends on what you mean by traced and also what configurations you use for sqlmap, but if we go with the typical meaning of traced in cybersecurity – meaning that someone could find out you used sqlmap against their platform – then with default sqlmap configurations, yes you would be traceable. However, there are other options you can use to mask your traces, such as using a `--tor` setting that tunnels your traffic through ToR which means you are just as untraceable as any other use of ToR. Privacy and anonymization is a big and complex topic outside the scope of this course, so we won't get into that here, but that's my answer for now.



## What is sqlmap in Termux?

Termux is an Android terminal emulator and Linux environment application, so it sounds like people are wondering what sqlmap is when they're using that...and it's not different than what we've already discussed!

## Conclusion

OK, so that's it for this lesson on what sqlmap is. Let's complete this lesson and move on to the next!

## Pre-requisites

Really quickly before we move on, I did want to highlight some pre-requisites that you should have before taking this course.

In order to fully utilize SQLMap, it's important that you have a solid foundation in these areas:

- **SQL** - You should know SQL pretty well. You don't have to be a Database Administrator, but if you don't know what I mean by SQL, then I would stop here and I would go find a course dedicated to SQL. Then, I would recommend that you check out my Injection Attacks course, because you will also need to learn about SQL injections before you can really understand this tool. Otherwise, it's like buying a car when you've never driven one before. Yes, technically you can try to drive it, but you're probably going to crash.
- **Databases**
  - What they are, how they work
  - Different database engines and their differences (ie: MySQL vs SQLite)
  - You don't have to be an expert in this, of course, but if you don't understand what I mean by database engines, that would be a red flag. Stop here and go find a database course!
- **Web or Software Development**
  - At least understand how applications are built, structured, and how they use databases, because otherwise finding SQL injections is going to be very difficult, and you need to be able to find potential areas of attack to configure & use sqlmap properly



---

So I would say that those are the main 3 areas that you should be at least familiar with before taking this course. If any thing I said in this lesson is foreign to you, and you don't understand it, or you're rusty because it's been a while since you've touched SQL (for example), then I would just try and brush up on that first, and then come right back and keep going!

That's it, go ahead and complete this lesson, and let's get started!

# Getting Started

## Creating a home lab environment

In this lesson, we walk through setting up our environment in order to follow along with the hands-on demonstrations throughout the course. This is an important lesson to complete if you want to apply what you're learning hands-on, so if you get stuck at any point in time, please reach out and we'll help you resolve the issue so that you can move on.

The first thing we need to configure is Kali Linux, which is a free Linux distribution that's often used for digital forensics and penetration testing. The reason we want to use Kali is because it comes pre-installed with many of the tools we'll be using throughout the course, which will help us get going and avoid issues that can come from running different operating systems.

If you already have a lab environment set up, feel free to skip ahead to the section called **"Installing Docker in Kali"** in this lesson, and pick up from there.





## Creating a Kali Virtual Machine with VirtualBox

Don't worry, this step is not difficult and it doesn't take too much time. And again, this is all free.

If you don't already have VirtualBox or VMWare, go ahead and download whichever one you prefer, but I'll be using VirtualBox.

All you have to do is go to [virtualbox.org](https://www.virtualbox.org) and download the latest version for your current operating system. I'm on a mac, so I'll download the OS X version, but if you're on Windows you would download that version.

Then, follow the steps to install VirtualBox. At this point, if you have any issues during the installation and you can't figure out a solution, please reach out in our forums and we'll be glad to help.

Once you have VirtualBox installed and running, it's time to set up Kali Linux.

I'll use an OVA version. This is a very simple way of getting Kali up and running without having to configure a lot of settings, and it will work just fine for this course.

First, we'll want to download Kali at this URL: <https://www.kali.org/downloads/>



Since we're using the OVA version and VirtualBox, we'll need to click on this link:

<https://www.offensive-security.com/kali-linux-vm-vmware-virtualbox-image-download/>

And we'll download the 64-Bit version. This can take a few minutes to a couple of hours depending on your internet connection.

While that's downloading, a quick note for those on Windows: if you have WSL (Windows Subsystem for Linux) on Windows 10 installed, some students have reported issues with downloading and updating packages inside of Kali. The following subsection addresses that issue. If you don't have WSL installed, you can skip this section. If you're not sure what it is and you are running Windows 10, follow the steps below just in case.

---

### **Optional: WSL fix**

Open up a Windows PowerShell with admin privileges and type in this command:

```
bcdedit /set hypervisorlaunchtype off
```

You may want to reboot just for good measure.

You can always re-enable it whenever you need, but that should prevent issues with labs in this course.

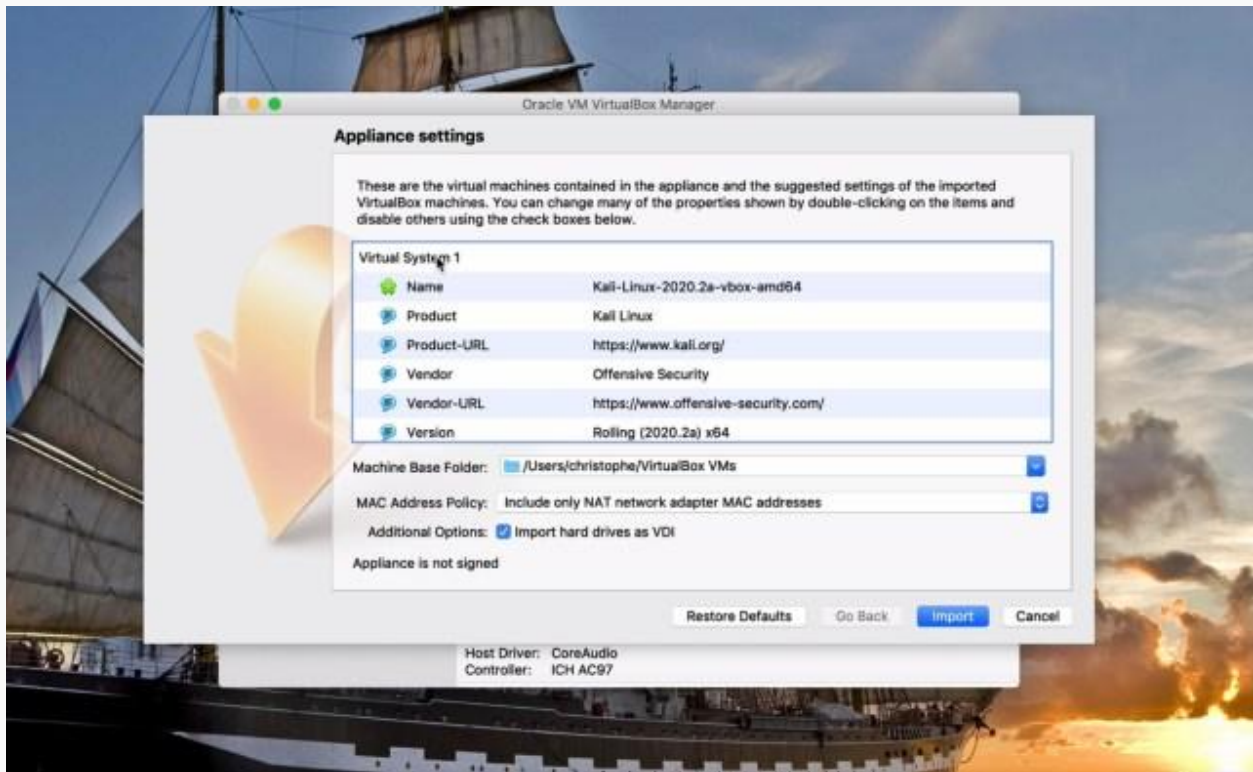
### **End of Optional: WSL fix**

---

Once you've downloaded the OVA, go to VirtualBox and Import the Appliance (File -> Import Appliance), or simply open the OVA file.

Then, start the import process. This can take a few minutes.





### *Importing the Kali OVA into VirtualBox*

After importing the appliance we can check the machine's settings and tweak anything that we'd like. This is where you can add more resources to the virtual machine, for example, but I'm personally going to leave it to defaults. We're now ready to start the machine.

Log in using kali/kali as username/password (we will change this in a moment).

Then, you'll probably need to resize the window since it's usually very small when you first start it. You can do that from the View menu, or by dragging the corner of the window.

Now that we're logged in, let's change the default password.

### Changing the default password

`passwd`

Make sure you read the instructions because people oftentimes blow through those steps and wonder why it doesn't work :-). The system will ask you to put in your *current password* first, then your new password twice.



Now that we've got a new password, let's install Docker.

## Installing Docker in Kali

```
sudo apt update
sudo apt install -y docker.io
```

At this point, the docker service is started but not enabled. If you want to enable docker to start automatically after a reboot, which won't be the case by default, you can type:

```
sudo systemctl enable docker --now
```

The last step is to add our non-root user to the docker group so that we can use Docker:

```
sudo usermod -aG docker $USER
```

We now need to reload settings so that this permissions change applies. **The best way to reload permissions is to log out and back in.**

**If you don't want to do that, a quick workaround that will only apply to the current terminal window is:**

```
newgrp docker
```

If that doesn't work, try to reboot the system. Otherwise, you may find that other terminal windows haven't reloaded settings and you may get "permission denied" errors. But, if you'd rather not log out or reboot at this time, you can use the above command.

## Running our target environment with Docker

With docker installed, we can now pull in different environments as we need them, without having to install any other software for those environments.

## The Damn Vulnerable Web Application (DVWA)

For example, if we want to run the Damn Vulnerable Web Application, we can do that with this simple command:

```
docker run --rm -it -p 80:80 vulnerables/web-dvwa
```

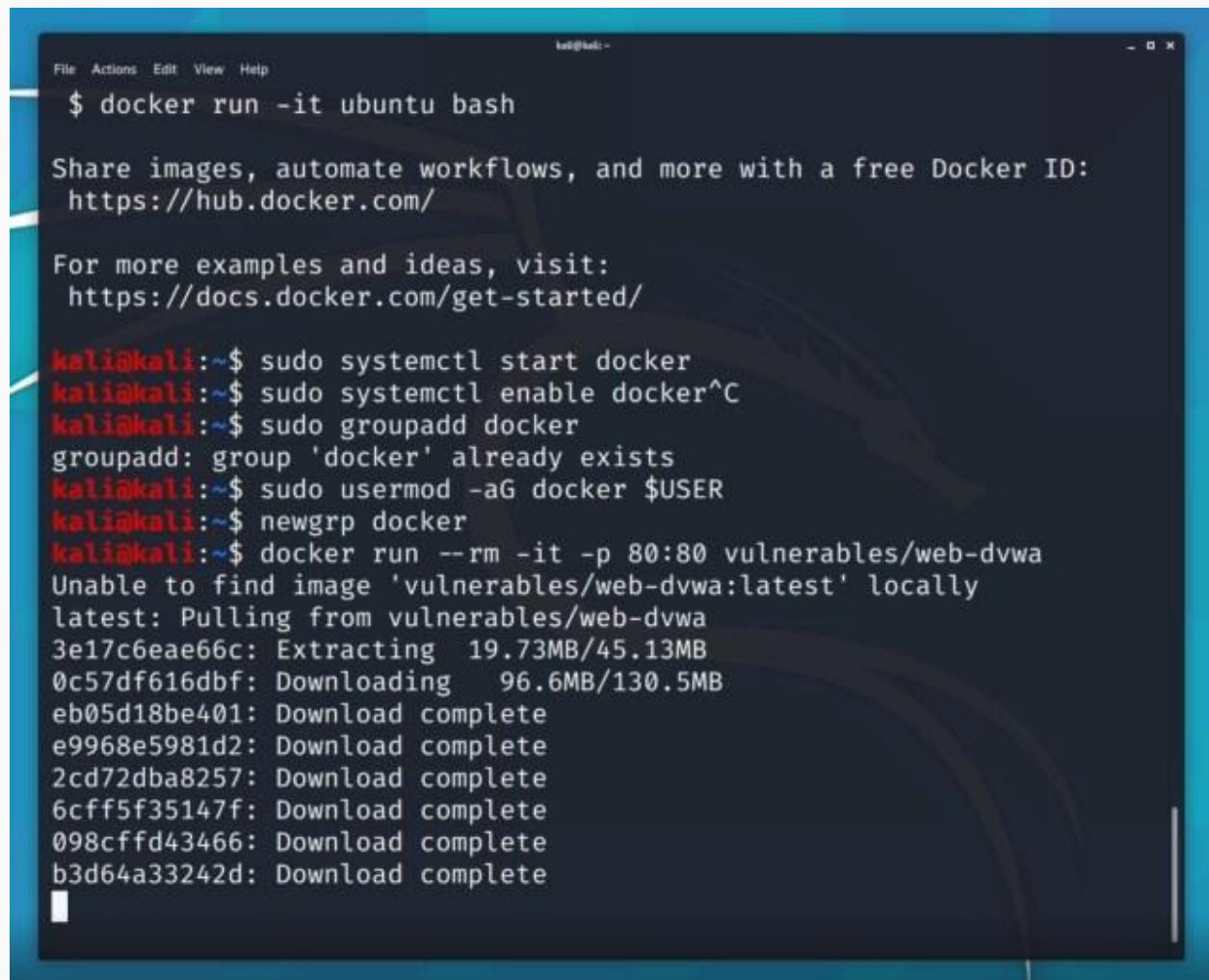
If that doesn't work, try running this command first:

```
docker pull vulnerables/web-dvwa
```



and then re-run the `docker run` command above.

You'll have to wait until it downloads the needed images and starts the container. After that, it will show you the apache access logs so you can see requests going through the webserver.

A terminal window with a dark background and light blue text. The window title is 'kali@kali: ~'. The menu bar shows 'File', 'Actions', 'Edit', 'View', and 'Help'. The terminal content shows the execution of 'docker run -it ubuntu bash'. Below this, there are informational messages from Docker about sharing images and getting started. Then, several systemctl and groupadd commands are run to start and enable Docker. Finally, the 'docker run --rm -it -p 80:80 vulnerables/web-dvwa' command is executed, which triggers the download of the 'vulnerables/web-dvwa:latest' image. The progress of the download is shown for several layers, including '3e17c6eae66c' and '0c57df616dbf', with their respective sizes and download status. The terminal ends with a cursor on a new line.

```
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/

kali@kali:~$ sudo systemctl start docker
kali@kali:~$ sudo systemctl enable docker^C
kali@kali:~$ sudo groupadd docker
groupadd: group 'docker' already exists
kali@kali:~$ sudo usermod -aG docker $USER
kali@kali:~$ newgrp docker
kali@kali:~$ docker run --rm -it -p 80:80 vulnerables/web-dvwa
Unable to find image 'vulnerables/web-dvwa:latest' locally
latest: Pulling from vulnerables/web-dvwa
3e17c6eae66c: Extracting 19.73MB/45.13MB
0c57df616dbf: Downloading 96.6MB/130.5MB
eb05d18be401: Download complete
e9968e5981d2: Download complete
2cd72dba8257: Download complete
6cff5f35147f: Download complete
098cffd43466: Download complete
b3d64a33242d: Download complete
```

You can navigate to [127.0.0.1](http://127.0.0.1) in your browser in order to access the web application.

It will ask you to login, and you can use the username *admin* and password *password*. Initially, you will be redirected to [localhost/setup.php](http://localhost/setup.php) where you can check configurations and then create the database. It should automatically redirect you to log in again, but if it doesn't, scroll down and click on login to re login.





---

Please note that you'll have to do these quick steps each time you take down your environment and bring it back up. So if you take a break from the course and come back later after shutting down the environment, you'll have to use the docker run command again.

Do manually shut down the environment, go back to the terminal window where we started the container, and use `Ctrl + C` or `Cmd + C` to get our terminal back and terminate the docker environment.

Now that we've got our lab environment up and running, it's time to download sqlmap!

## Downloading sqlmap

By default, sqlmap comes pre-installed with Kali Linux, so as long as you followed the steps in the prior lesson and downloaded the latest Kali image, then you should have a very up-to-date sqlmap installation.

With that said, it might not be the *latest* version of sqlmap. As you can see here, our version of sqlmap is `1.4.11#stable`. The term stable can mean a few different things when it comes to software and is up to the developers to define its meaning, but it typically means that the version has ironed out most of the major bugs and issues, and so it is the intended version to be used by most users. So this is a good version to use.

But again, we can check and see if there are newer versions by going to the project's GitHub repository and checking releases.

We can see that there *is* a newer release with a version bump to `1.5`.

Let's go ahead and download this version to our Kali installation!

We can use a few different methods to download the latest version, including by downloading archives, or by cloning the Git repository.

```
git clone --depth 1 https://github.com/sqlmapproject/sqlmap.git sqlmap-dev
```

A benefit of cloning the repository is that you can easily update your version by pulling changes through git at a later time, versus if you download an archive, that archive is "stuck in time" so to speak. You'd have to regularly go back out and download the latest archive to update.

So it's up to you, but we have git installed on this machine so I'll go ahead and use that method.



As you can see though by the ending of our command, this should be treaded as a `-dev` (development) version of sqlmap, meaning that it's the cutting edge and there may be some unknown or known bugs. But it also means that you get the latest features.

So it's a tradeoff to be aware of, and the version that you choose to use is up to you!

Going into our new directory:

```
cd sqlmap-dev
```

We can run this version by typing `python3 sqlmap.py` and we will see version `1.5.3.22#dev`

For this course, we will stick to the default installation of sqlmap to minimize the risk of bugs and issues along the way, but feel free to try using this latest version instead - just be aware that there may be some differences.

So now that we've verified that we have a working installation of sqlmap, and we've learned how to download the latest version, let's complete this lesson and move on to the next where we will use sqlmap for the first time!

## Using sqlmap for the first time

Whenever I download new command-line tools, there are a few things I like to do:

1. [Open up the tool's documentation](#) - which is usually on GitHub, at least for open source tools
2. Use `-h` in my terminal

`-h` typically gives a good overview of some of the most popular and useful commands or options that we can use with this particular tool, and it's usually in an easily digestible format which makes it easier to start with.

Combining that with the tool's documentation, which can sometimes be hit or miss, and we can be off to a good start within a matter of minutes.



## sqlmap documentation

Luckily, sqlmap's documentation is actually really good for this type of open source project, so props to the authors Bernardo & Miroslav!



Let's start by looking over the Introduction section of the tool's documentation on GitHub.

On this page, they give us a very simple example of what a target URL might look like for SQL injection:

```
http://192.168.136.131/sqlmap/mysql/get_int.php?id=1
```

We have a target IP, a path to an endpoint, and a query parameter of `id=1`.

To check for SQL injection, we might alter the parameter value from being `1` to being something different, like `1+AND+1=1`, or `1+AND+1=2`, and watching for the result on the webpage or via the HTTP request. This is something that we can do manually and that we can use a tool such as ZAP or Burp to help with.

But, this is also something that sqlmap can help with!

By passing in that same target to sqlmap:

```
http://192.168.136.131/sqlmap/mysql/get_int.php?id=1
```

The tool will:

- Identify the parameters to test (in this case `id`)
- Identify which SQL injection techniques can be used to try and exploit that parameter
- Fingerprint the back-end database management system (to gather information about what technologies we're dealing with)
- And, depending on what it finds, attempt to exploit vulnerabilities

So that's great, and that sounds awesome, but how do we go beyond concepts and actually put this into practice?



Well, we can head over to the **Usage** section of this documentation, which includes a massive comprehensive list of all sqlmap options and how to use them, but that can be overwhelming to beginners, so instead, let's go back over to our Kali installation and let's use the `-h` option to get us started.

## `-h` options

Scrolling to the top, we right away can see a list of options for `Target`. As we saw with the prior example, we want to be able to tell sqlmap which target to go after, and we can use the very first option of `-u` or `--url` in order to specify that. This is where we will enter our URL to go after, which I will show in just a moment.

Then, we see a section for `Request` which lets us specify how we want sqlmap to connect to our target URL.

This is important to understand because different requests and targets require different approaches. If it's a `GET` request, that's different than if it's a `POST` request. If it's a `POST` request, we'll need to pass in `--data=DATA` where the all-caps `DATA` is a string to be sent through the POST request.

We may also need to pass in `--cookie=COOKIE` where the all-caps `COOKIE` is an HTTP Cookie header value, because if we're going after an application that requires authentication and the authentication uses cookies, then our sqlmap requests won't go through and will get rejected by the application *unless* we provide that cookie information.

## Enumeration

Another important set of options to know about is under the `Enumeration` category. These options basically tell sqlmap what to do and what to return if it finds a vulnerability that it can exploit, or if it finds information that is not properly concealed.

For example, you can ask it to retrieve the current Database Management System user with `--current-user`, you can ask for a list of the database's tables with `--tables`, the entire database schema with `--schema` and more.

This is a really powerful set of options that we will definitely use in this course and that we use in the more advanced version of this course as well!

```
--wizard
```



Finally, before completing this lesson and moving on to the next where we will issue our first command, let's take a quick look at the `--wizard` option that's provided by sqlmap.

This option is really only useful when you're first getting started with sqlmap, because once you have more experience, you can work faster and issue more powerful commands without using the wizard. But, this option basically spoon-feeds you what it needs to properly attack a target.

Let's try it, even though we don't have an environment running yet, just to see what it does:

```
Please enter full target URL (-u): http://localhost/

POST data (--data) [Enter for None]: Enter

Injection difficulty (--level/--risk). Please choose:
[1] Normal (default)
[2] Medium
[3] Hard
> 1

Enumeration (--banner/--current-user/etc). Please choose:
[1] Basic (default)
[2] Intermediate
[3] All
> 1

sqlmap is running, please wait..
```

It fails because we have no environment running on our localhost, but we get to see how it asks us questions to determine what it is that we need. Again, this is just a simpler way of using the same options and commands that we would issue via the terminal instead.

## Conclusion

Now that we've taken a look at the main available options to get us started in using sqlmap, let's complete this lesson and move on to the next where we will apply what we just learned in order to issue our first sqlmap command and find as well as exploit our first SQL injection with sqlmap.



---

# Finding and exploiting SQL injections

## Your first SQL injection with sqlmap

From what we talked about in the prior lesson, we actually have enough to issue our first command against a vulnerable target environment, so let's see what we can do.

First things first, turn on your target environment (if it's not already running) just like we showed in a prior lesson on creating our lab environment:

```
docker run --rm -it -p 80:80 vulnerables/web-dvwa
```

Log in to the app with `admin / password`, create the database, and re login.

Now that our environment is on, let's navigate to our target endpoint by going to `SQL Injection`.

At this endpoint, you can see that we have a user input field asking for a `User ID`. If we type in a random number and click on `Submit`, you'll notice that the URL changes, and this gives us a hint that it's probably a GET request which we can confirm by opening up DevTools (F12 on your keyboard) and going to the `Network` tab, then re-submitting a similar request.

We can see that it is indeed a GET request, and when we click on it, we'll get additional information about this request.

Let's leave this open for now and let's open up another terminal to use sqlmap from.



As we know, we need to pass in a target URL with the `-u` option. So let's copy this entire URL from our browser into the terminal.

Let's paste the URL, including the parameter, and let's paste it in our terminal in between quotes:

```
sqlmap -u "http://127.0.0.1/vulnerabilities/sqli/?id=123&Submit=Submit#"
```

In some cases, you may need to specify a port in addition to the target URL, especially if it's not running on the default http or https ports of 80/443 respectively. Because otherwise, sqlmap will try to use those default ports and won't be able to establish a connection. In this case we are using port 80 and we specified `HTTP` so there is no need to specify the port.

Let's submit this request even though it won't work. We see that it says:

```
[15:39:47] [INFO] testing connection to the target URL
got a 302 redirect to 'http://127.0.0.1:80/login.php'. Do you want to
follow? [Y/n]
```

Press `Ctrl + C` to get out of this prompt.

If you remember, we had to login to the application in order to access this endpoint. So because we are behind a gated page, we need to give sqlmap the options it needs to properly authenticate its calls.

In the case of this application, it's really easy because it just uses 2 simple cookies. Going back to our browser and DevTools, we can see exactly what those cookies are by clicking on the `Cookies` tab. There, we'll find:

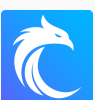
```
PHPSESSID: "ap6jvs0dj7mr4786v6uhn1hj11"
security: "low"
```

Your `PHPSESSID` will look different than mine, so don't copy mine, or it won't work.

Since we need to pass in cookie information to sqlmap, let's add that to our command:

```
sqlmap -u "http://127.0.0.1/vulnerabilities/sqli/?id=123&Submit=Submit#"
--cookie="PHPSESSID=ap6jvs0dj7mr4786v6uhn1hj11; security=low"
```

This should work now, but let's add one more option to this command: `--tables`. This `Enumeration` option will give sqlmap a specific task to do. In this case, `--tables` will make sqlmap enumerate DBMS database tables — meaning, it will crawl the database structure and information to find and return the names of all of the tables contained in this database. Of



course, that's assuming it finds a SQL injection vulnerability first, since that's going to be the entry point that allows us to do that.

So our final command is this:

```
sqlmap -u "http://127.0.0.1/vulnerabilities/sqli/?id=123&Submit=Submit#"
--cookie="PHPSESSID=ap6jvs0dj7mr4786v6uhn1hj11; security=low" --tables
```

Let's press <Enter> and let it do its thing.

With how we entered this command, sqlmap will periodically stop and prompt us to make a decision. Here it's saying that the back-end DBMS is MySQL, so it's asking if it wants us to skip test payloads specific to other DBMSes, which makes sense. If we know it's MySQL, there's no point in using unrelated payloads. So we'll say **Y** - and by the way, you'll notice the default because it's capitalized.

Again, it asks us another question: do we want to include all tests for MySQL? Let's say yes.

As sqlmap is running, let's take a look at our terminal that's running the container. You'll notice requests coming through from sqlmap — these are the requests being sent by sqlmap to the server (or in this case local container). So you can see the exact payloads being submitted.

But, once again, it asks us a question. This can get really annoying especially when you don't have any reason to doubt the defaults, so I'll show you how to skip this next time you run a sqlmap command, but now let's answer **N** since it says that the parameter 'id' is vulnerable, so there's no need for us to keep testing other parameters.

## sqlmap result

Scrolling back up to where we answered No, we can see additional information about injection points, successful payloads, and then information returned from sqlmap, like:

- The DBMS (MySQL)
- The database names (dvwa, information\_schema)
- The tables in those databases (dvwa: guestbook & users)

This information is returned since we specifically asked for database tables to be enumerated, so sqlmap did exactly as we asked! It attacked our target endpoint, found a SQL injection vulnerability via the `id` parameter, exploited that vulnerability, and then enumerated the databases' tables.





---

Before we complete this lesson, let me show you a way to bypass the prompts that have defaults with just a simple option. This time, let's also enumerate the database schema instead of tables just to switch it up:

```
sqlmap -u "http://127.0.0.1/vulnerabilities/sqli/?id=123&Submit=Submit#"
--cookie="PHPSESSID=ap6jvs0dj7mr4786v6uhn1hj11; security=low" --schema
--batch
```

Adding the `--batch` is what's going to automatically use default behavior instead of asking for user input.

We can watch it do its thing, and then return the results back containing the database schema!

## Conclusion

This gives us a ton of information that will help us write a report back to our client about the vulnerability, the information disclosure, and other potential exploitations.

If you've got experience with SQL injections by hand, it's also quite impressive how it managed to do all of this automatically. All it needed from us were a few options and configurations, and it handled the rest! Getting this much information and getting this far would have taken far longer if we were to do it by hand.

So this is a great start to using sqlmap! But, there's still more we can do, so go ahead and complete this lesson, and I'll see you in the next!

## Extracting passwords from a database

In the previous lesson, we successfully executed our first sqlmap command that found a SQL injection vulnerability and exploited it in order to give us all kinds of information about the application's database, including table names.

One of the tables we saw was *users*, which gives us a strong indication that there are probably user's passwords stored in there, so as a Red Teamer, one of our goals might be to see whether we can get that password information and crack those passwords.



## Enumerate the users table

First things first, we need to enumerate that table in order to see if we can grab those passwords, which will very likely be stored in an encrypted format, which we will have to try and crack.

We can do this by narrowing down sqlmap's focus to enumerate columns (`--columns`) in just this table with the `-T TBL` option, where TBL represents the database table or tables to enumerate.

```
sqlmap -u "<URL>" --cookie="PHPSESSID=iq3q8f9s2rtfuddrtiak2fabr0;  
security=low" --columns -T users --batch
```

Press enter, and we'll be able to confirm that there is a `passwords` column in this table.

So now, we're going to use the `--dump` option in order to dump the data contained in the `users` table.

```
sqlmap -u "http://127.0.0.1/vulnerabilities/sqli/?id=1&Submit=Submit#"  
--cookie="PHPSESSID=iq3q8f9s2rtfuddrtiak2fabr0; security=low" --dump -T  
users --batch
```

While this is working, we can see that it asked us if we wanted to store hashes to a temporary file for further processing with other tools. While this is a practical feature, we're not going to need that here. So the default was `N` which means it didn't save them.

Next, it asked if we wanted to crack the hashes via a dictionary-based attack, and the default was `Y`, which means sqlmap then launches a dictionary-based attack against the hashes found in the database.


We can see here that by default it uses the `/usr/share/sqlmap/data/txt/wordlist.tx_`, but we could have specified a custom dictionary file, or even a file with a list of dictionary files.

The final option that it asks is whether we want to use common password suffixes which is slower, so by default it says `N`.

Then, we can see that it started the attack and, shortly after, it was able to crack multiple password hashes.

So now we end up with a neatly formatted table that contains the `user_id`, `user`, `avatar`, `password` (includes the hash + cracked password), and other columns in this table.





sqlmap not only enumerated the table and found passwords, but it also has a built in cracker that was able to crack these password hashes and give us the plain-text version of the password.

Cool, right?

```
--passwords
```

One more option we can use that does something similar is the `--passwords` option. Except this time, it instructs sqlmap to list and crack the database management system (DBMS) users' password hashes. So not the application's users, but the users in the DBMS — ie: the users that manage the database.

When the user that we are assuming by running these exploits against the database has read access to the system table that contains the DBMS users' passwords, sqlmap is able to enumerate password hashes for each of those DBMS users. Again, sqlmap will ask us if we want it to try and crack the hashes, as we just saw earlier.

Unfortunately, in this environment, we do not have the proper permissions, so we can't demonstrate it this time. But this is a good reminder that not everything will work every time, even if you're able to find a vulnerability. That's why, as a bug bounty hunter or pentester, you have to continue pushing and trying to find different angles before giving up. What might work in one environment or in some circumstances may not work in others.

But, the fact remains that these are built-in features in sqlmap which makes it very practical, because it avoids the need to download or use other tools unless we're dealing with a more complex situation. We're able to do quite a bit from this tool alone.

## Conclusion

Now that we've completely compromised this application's users' passwords, let's mark this lesson as complete and move on to the next!



---

# Next Steps

## Continue building sqlmap skills

I hope you enjoyed this short course on learning how to use sqlmap for beginners. This course was meant as an easy-to-follow introductory course that taught you the basics of how to use sqlmap in order to find SQL injection vulnerabilities. But, as we talked about, sqlmap is a tool packed with amazing features, and there is still much left to learn.

Luckily, you have all of the tools and information you need to continue learning how to use sqlmap for free with just the [GitHub documentation](#) and the `-h` option. But, if you want a guided course similar to this one but that goes in a lot more detail and showcases more advanced features and functionality to find more SQL injections, then check out my sqlmap Deep Dive course on Cybr.com and enroll today!

Last, but certainly not least, I'd love to hear your feedback about what you thought of the course and if there are any areas you really enjoyed, or if you think I could improve it, please reach out - my email is [christophe@cybr.com](mailto:christophe@cybr.com).

I hope to see you in my next course, but even if you're not interested in checking out that course, be sure to check out my other free courses on [Cybr.com](#).

Be sure to follow Cybr's social media accounts for more content like this:

- [LinkedIn](#)
- [Facebook](#)
- [YouTube](#)

We also have a [Discord community](#) where you can chat in real-time with me, other course authors, and other Cybr members as well as mentors. [We have forums](#) if you'd prefer asking questions there and/or finding additional resources.

I'd also [love to connect on LinkedIn](#), where I post regularly.



---

Thanks again, and hope to see you soon!

- Christophe

