Finding Sensitive Data on Domain SQL Servers using PowerUpSQL

In this blog I'll show how PowerUpSQL can be used to rapidly target and sample sensitive data stored in SQL Server databases associated with Active Directory domains. We've used the techniques below to discover millions of PCI, HIPAA, and other sensitive records living inside and outside of protected network zones. Hopefully PowerUpSQL can help you do the same.

Finding Domain SQL Servers to Log Into

I touched on how to do this in another blog so I've only provided a summary of the PowerUpSQL commands below. For more information on how to discover accessible SQL Servers check out https://blog.netspi.com/blindly-discover-sql-server-instances-powerupsql/.

1. Download PowerUpSQL.

   https://github.com/NetSPI/PowerUpSQL

2. Import the Module

   ```
   PS C:\> Import-Module PowerUpSQL.psd1
   ```

3. Get a list of accessible SQL Servers on the domain.

   ```
   PS C:\> $Servers = Get-SQLInstanceDomain –Verbose | Get-
   SQLConnectionTestThreaded –Verbose -Threads 10
   ```

4. View accessible servers

   ```
   PS C:\> $Accessible = $Servers | Where-Object {$_.Status –eq "Accessible"}
   PS C:\> $Accessible

   ComputerName   Instance                       Status
   ------------   --------                       ------
   SQLServer1     SQLServer1\SQLEXPRESS          Accessible
   SQLServer1     SQLServer1\STANDARDDEV2014     Accessible
   SQLServer1     SQLServer1                     Accessible
   ```

Pro tip: Once you've obtained Domain Admin privileges, add yourself to the DBA groups and run through the process again. More access = more data.

Finding Sensitive Data on Domain SQL Servers

If you followed the instructions in the last section you should have a variable named "$Accessible" that contains a list of all accessible SQL Server instances. The command below uses that variable to perform a broad search across all accessible SQL Servers for database table column names that contain

provided keywords. I've created an example showing one server, but in real environments there are often hundreds.

```
PS C:\> $Accessible | Get-SQLColumnSampleDataThreaded –Verbose –Threads 10
–Keyword “card, password” –SampleSize 2 –ValidateCC -NoDefaults | ft -AutoSize

...[SNIP]...

VERBOSE: SQLServer1\STANDARDDEV2014 : START SEARCH DATA BY COLUMN
VERBOSE: SQLServer1\STANDARDDEV2014 : CONNECTION SUCCESS
VERBOSE: SQLServer1\STANDARDDEV2014 : - Searching for column names that match
criteria...
VERBOSE: SQLServer1\STANDARDDEV2014 : - Column match:
[testdb].[dbo].[tracking].[card]
VERBOSE: SQLServer1\STANDARDDEV2014 : - Selecting 2 rows of data sample from
column [testdb].[dbo].[tracking].[card].
VERBOSE: SQLServer1\STANDARDDEV2014 : COMPLETED SEARCH DATA BY COLUMN

...[SNIP]...

ComputerName   Instance                    Database Schema Table    Column
Sample           RowCount IsCC
------------   --------                    -------- ------ -----    ------ -----
-           -------- ----
SQLServer1     SQLServer1\STANDARDDEV2014 testdb   dbo    tracking card
4111111111111111 2        True
SQLServer1     SQLServer1\STANDARDDEV2014 testdb   dbo    tracking card
41111111111ASDFD 2        False

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

Below is a breakdown of what the command does:

- It runs 10 concurrent host threads at a time
- It searches accessible domain SQL Servers for database table columns containing the keywords “card” or “password”
- It grabs a two sample records from each matching column
- It checks if the sample data contains a credit card number using the Luhn formula
- It filters out all default databases

If you want to target a single server you can also use the command below.

```
Get-SQLColumnSampleData –Verbose –Keyword “card, password” –SampleSize 2
–ValidateCC -NoDefaults  –Instance “Server1\Instance1”
```

Targeting Potentially Sensitive Databases

To save time in larger environments you may want to be a little more picky about what servers you're targeting during data searches. Especially if you're searching for multiple keywords. Dumping a list of databases and their properties can give you the information you need to make better server targeting decisions.

Some key pieces of information include:

- **Database Name**
  This is the most intuitive. Databases are often named after the associated application or the type of data they contain.
- **is_encrypted Flag**
  This tells us if transparent encryption is used. People tend to encrypt things they want to protect so these databases make good targets. Transparent encryption is intended to protect data at rest, but if we login as a sysadmin, SQL Server will do the work of decrypting it for us. A big thanks goes out to James Houston for sharing that trend with us.
- **Database File Size**
  The database file size can help you determine if the database is actually being used. The bigger the database, the more data to sample.

To dump a list of all accessible SQL Server databases you can use the command below. Once again, we'll use the "$Accessible" variable we created earlier. Storing the accessible servers in a variable allows us to quickly execute different PowerUpSQL functions against those servers without having to run the discovery commands again.

Note: The example only shows a sample of the output for one record, but in most environments you would have a lot more.

```
PS C:\> $Databases = $Accessible | Get-SQLDatabaseThreaded –Verbose –Threads 10
-NoDefaults
PS C:\> $Databases

...[SNIP]...

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        : True
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]...
```

Once the results are stored in the "$Databases" variable there a ton of ways to view the data. Below are some of the more common options. In the examples, the results are sorted by the database name alphabetically.

```
# Output results to display
$Databases | Sort-Object DatabaseName

# Output results to display in table format
$Databases | Sort-Object DatabaseName | Format-Table -AutoSize

# Output results to pop grid with search functionality
$Databases | Sort-Object DatabaseName | Out-GridView

# Output results to a csv file
$Databases | Sort-Object DatabaseName | Export-Csv -NoTypeInformation
C:\temp\databases.csv
```

If you're only interested in encrypted databases you can use the command below.

```
$Databases | Where-Object {$_.is_encrypted –eq "TRUE"}
```

The "$Databases" output can also be piped directly into the Get-SQLColumnSampleDataThreaded command as shown below.

```
$Databases | Where-Object {$_.is_encrypted –eq "TRUE"} |Get-
SQLColumnSampleDataThreaded –Verbose –Threads 10 –Keyword "card, password"
–SampleSize 2 –ValidateCC -NoDefaults
```

Of course, some people are not fans of multi step commands…

Bringing it All Together

If you prefer to fully automate your data sampling experience everything can be executed as a single command. Below is an example:

```
Get-SQLInstanceDomain -Verbose | Get-SQLColumnSampleDataThreaded –Verbose
–Threads 10 –Keyword “credit,ssn,password” –SampleSize 2 –ValidateCC
–NoDefaults |
Export-CSV –NoTypeInformation c:\temp\datasample.csv
```

Wrap Up

In this blog I showed how sensitive data could be targeted and quickly sampled from domain SQL Servers using PowerUpSQL. I also noted that databases that use transparent encryption tend to make good targets for review. Hopefully the scripts will save you as much time as they’ve saved us. Either way, good luck and hack responsibly!