FIRST STEPS IN SECURING YOUR SCRIPTS

Tim Curwick

MadWithPowerShell.com

Automation Consultant

RBA Consulting

Jeff Scripter

Application Systems Engineer

Wells Fargo

1E

adaptiva

DELL

Kollective

Microsoft

vmware

▸Tim Curwick		▸Jeff Scripter
▸@MadWPowerShell		▸@JeffTheScripter
▸PowerShell MVP since 2016		▸ MMS Speaker
▸Computer geek since 1982		▸10 years
▸I like cows		▸Brewing Beer and Coffee

MMS

OVERVIEW

- PowerShell 'Security' Settings
  - Execution Policy
  - Constrained Language mode
  - Just Enough Admin (JEA)
  - Over-The-Shoulder Logging
- Access Control Security and PowerShell
  - ACLs and Permissions
  - Service Accounts and Managed Service Accounts
  - Peer Review
- SQL Injection
  - Parameterization
  - Stored Procedures
- Encryption
  - Obfuscation
  - Symmetric Encryption
  - Data Protection API
  - Asymmetric Encryption
- Conclusion
- Questions?

POWERSHELL SECURITY SETTINGS

Lets talk about Execution Policy, Constrained Language Mode and Just Enough Admin.

MMS

EXECUTION POLICY

- What is it:
  - It was never a solution to prevent users from running scripts.
  - Prevents accidental execution of scripts.

- How does it work:
  - It is a tool to determine which scripts can run by DEFAULT
    - Bypass – Everything can run
    - Unrestricted – Everything can run but you might be prompted for downloaded
    - Remote Signed – Scripts and modules with remote bit have to be signed by a trusted publisher
    - All Signed – Everything needs to be signed by a trusted publisher
    - Restricted – nothing can run

CONSTRAINED LANGUAGE MODE

- What is it:
  - A restriction in PowerShell that limits PowerShell
    - Full language: everything is available
    - Constrained Language: Disables com objects, many .Net objects, custom types, methods, dot sourcing, and a lot more.
    - No language: No PowerShell
- How does it work:
  - It follows predefined rules.
  - The intent is to use this as part of a larger security stance including app locker.

Quick Demo

- Testing with Constrained Mode:

$ExecutionContext.SessionState.LanguageMode = <0,1,2>

```
Cannot start "powershell". Interactive console applications are not supported.
To run the application, use the Start-Process cmdlet or use "Start PowerShell.exe" from the File menu.
To view/modify the list of blocked console applications, use $psUnsupportedConsoleApplications, or consult online help.
At line:0 char:0
```

```
At line:1 char:1
+ [System.Environment]::OSVersion
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Property references are not allowed in restricted language mode or a Data section.
At line:1 char:1
+ [System.Environment]::OSVersion
+ ~~~~~~~~~~~~~~~~~~~~
The type System.Environment is not allowed in restricted language mode or a Data section.
    + CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
    + FullyQualifiedErrorId : PropertyReferenceNotSupportedInDataSection
```

MMS

JUST ENOUGH ADMIN (JEA)

- What is it:
  - Customizable cmdlet whitelist which runs under the users account, a service account, or a virtual account.

- How does it work:
  - This is defined by a trusted admin and installed on a server.
  - The intention is to not give admins normal permissions but rather give define narrow commands that an admin needs to do their job.
    - Capabilities file – What the users can do?
    - Session Configuration file - Who can do it? And as Whom?

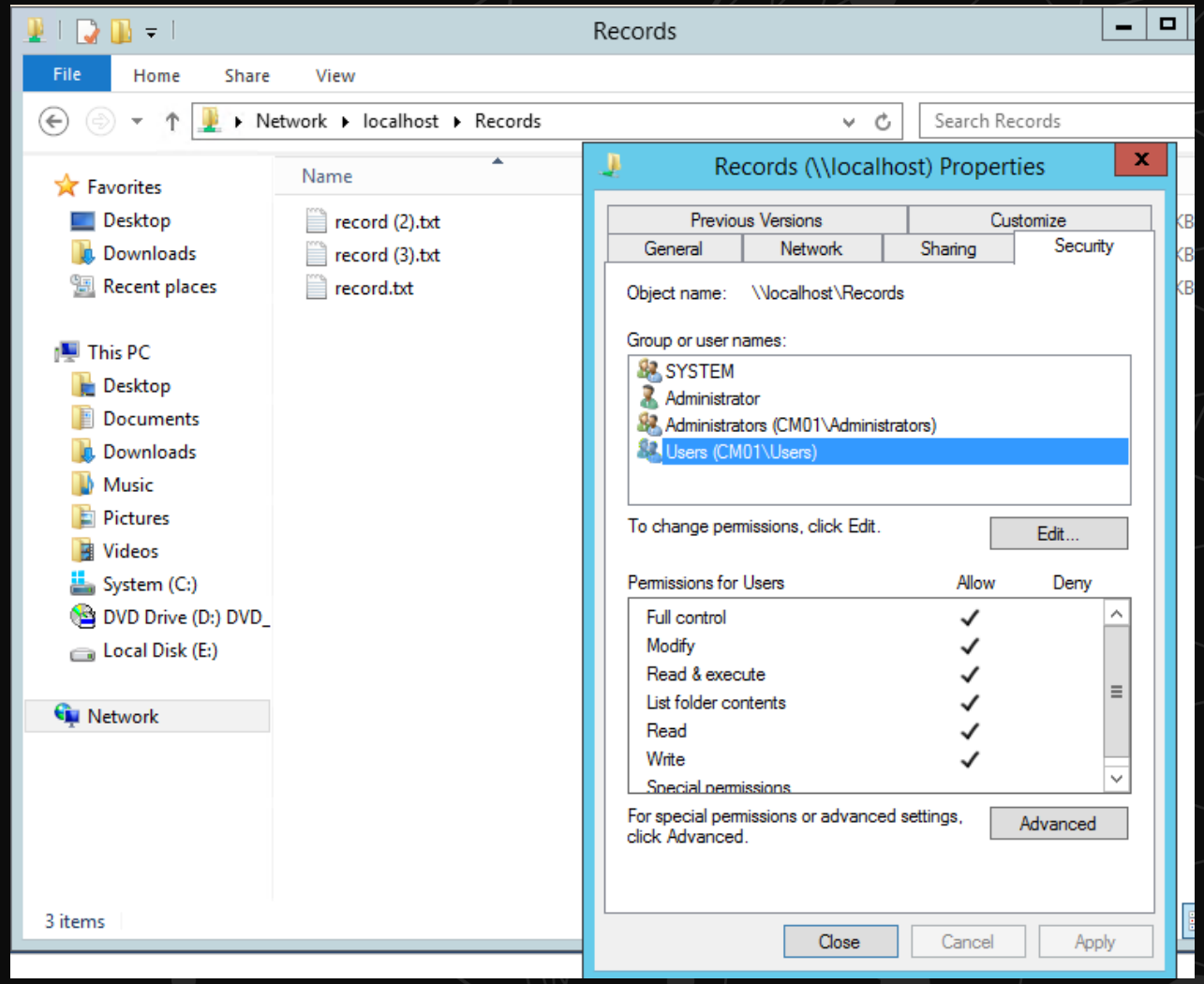
* Requires PowerShell 5

ACCESS CONTROL SECURITY

How much information is exposed to users?

ACLS AND PERMISSIONS

- When does this matter:
  - Scheduled tasks
  - File\User input
- What does this protect against:
  - Changes in your script
  - Inappropriate inputs for your script
  - Any sensitive data in your scripts:
    - Usernames and Passwords
    - Server Names



SERVICE ACCOUNTS VS. MANAGED SERVICE ACCOUNTS

- Service Accounts:
  - These are user accounts that are denied interactive logon and used to preform tasks.
  - Require password changes
  - Password is usually random

- Managed Service Accounts:
  - These are user accounts that are denied interactive logon and used to preform tasks.
  - Require password changes, but this is handled for you
  - Password are random and complex
  - Passwords are stored in AD

MMS

MANAGED SERVICE ACCOUNTS SETUP (RSAT REQUIRED)

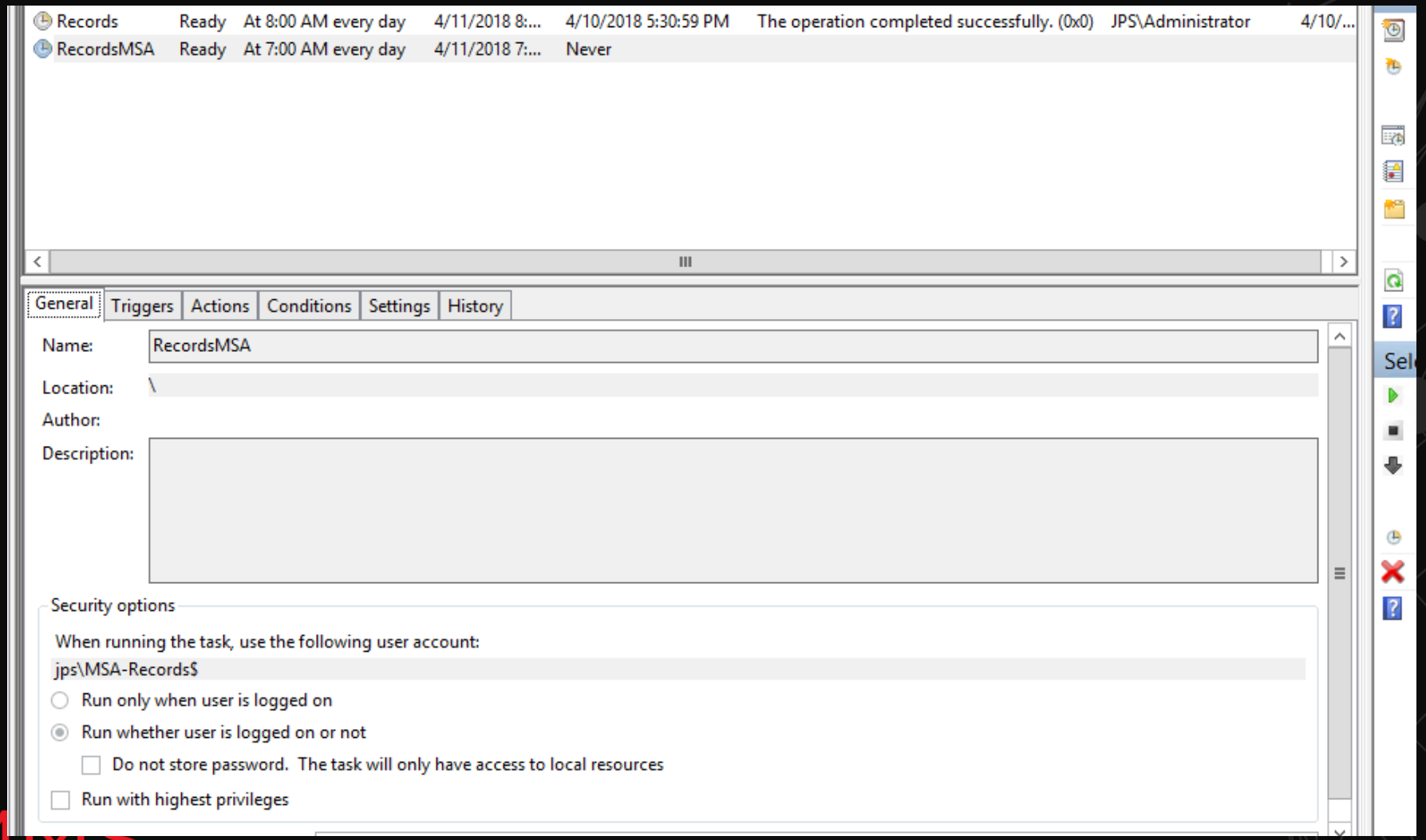
```
PS C:\windows\system32> New-ADServiceAccount -name MSA-Records -DNSHostName MSA-Records.jps.com -PrincipalsAllowedToRetr
ieveManagedPassword 'CN=CM01,OU=Servers,OU=JPS,DC=corp,DC=JPS,DC=com'
PS C:\windows\system32> Install-ADServiceAccount MGS-Records
```

```
PS C:\windows\system32> Install-ADServiceAccount Msa-Records
PS C:\windows\system32> $env:computername
CM01
PS C:\windows\system32> Test-ADServiceAccount msa-records
True
```

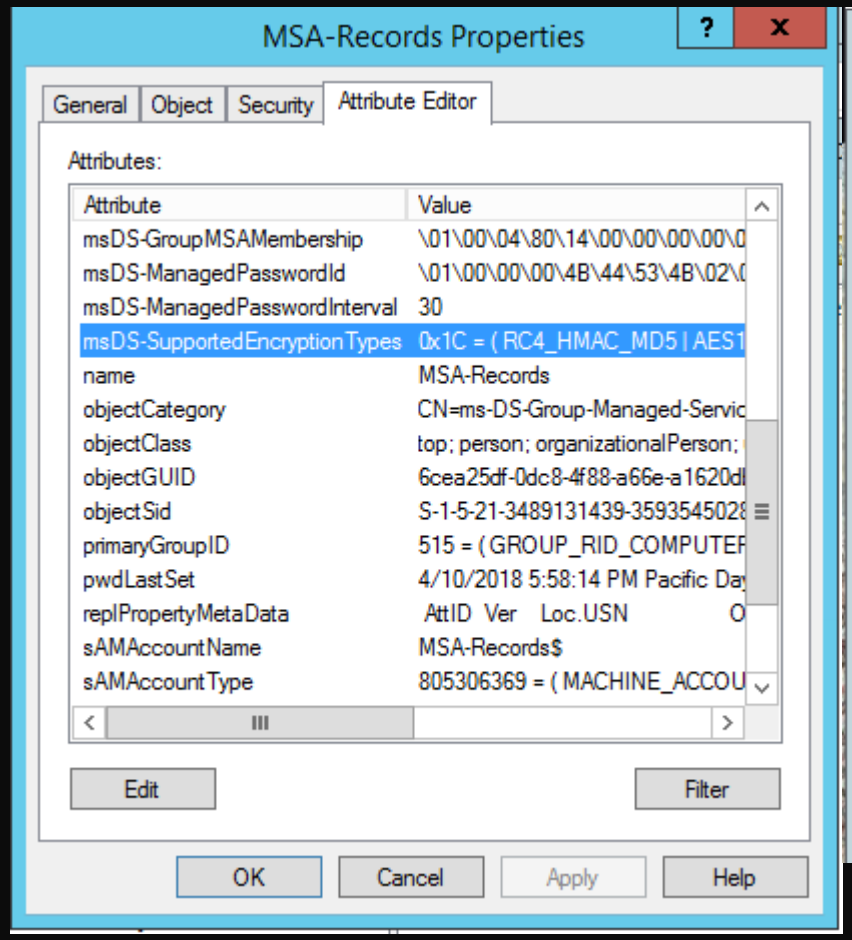
```
PS C:\windows\system32> $action =New-ScheduledTaskAction -Execute powershell -Argument '-file c:\scripts\Register.ps1'
PS C:\windows\system32> $schedule = New-ScheduledTaskTrigger -At 7:00 -Daily
PS C:\windows\system32> $prin = New-ScheduledTaskPrincipal -UserId jps\MSA-Records$ -LogonType Password
PS C:\windows\system32> register-ScheduledTask -Principal $prin -Action $action -Trigger $schedule

cmdlet Register-ScheduledTask at command pipeline position 1
Supply values for the following parameters:
TaskName: RecordsMSA

TaskPath                                 TaskName                       State
--------                                 --------                       -----
\                                        RecordsMSA                     Ready


PS C:\windows\system32> _
```

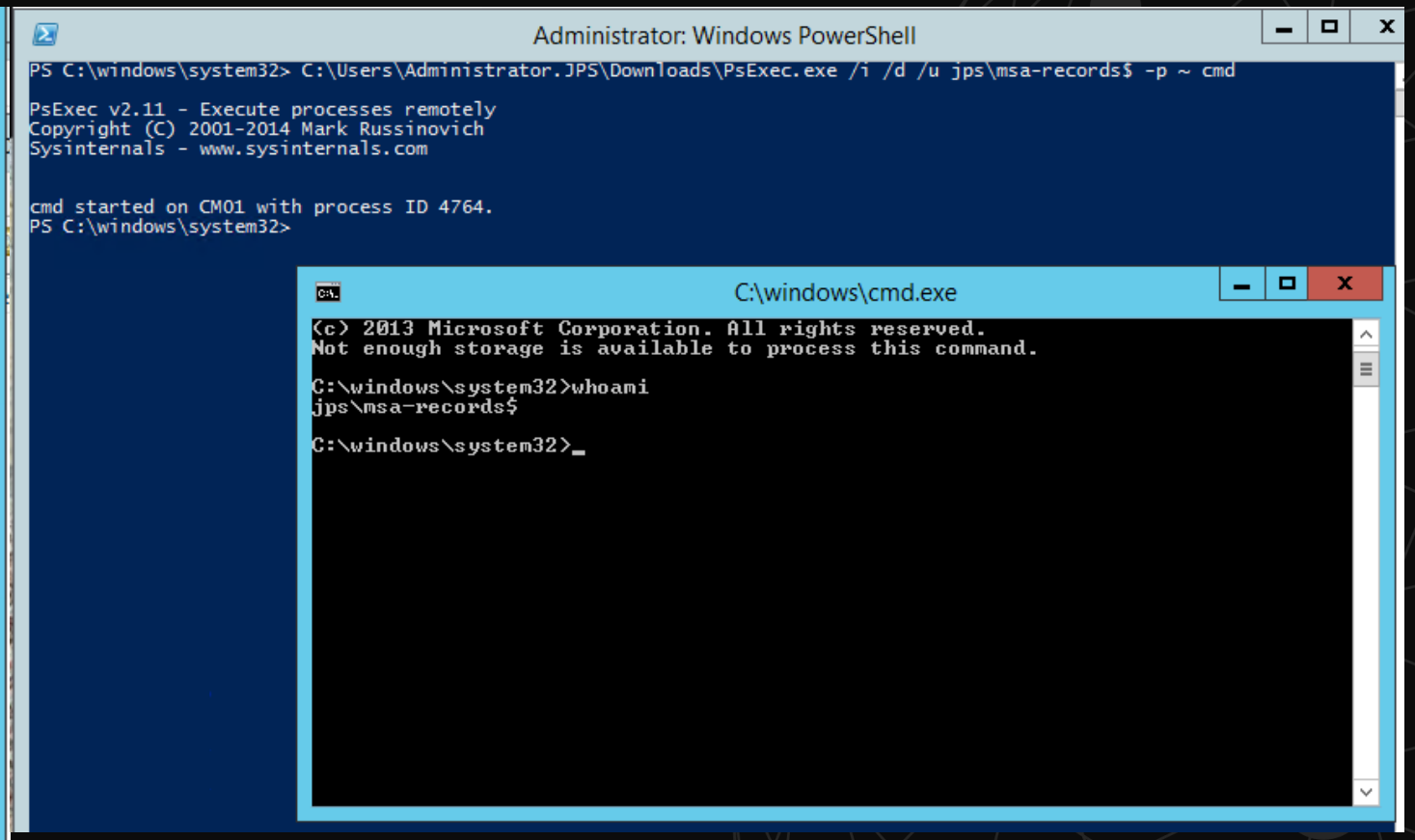
MMS

MANAGED SERVICE ACCOUNTS SETUP



MANAGED SERVICE ACCOUNTS SECURITY





MMS

MANAGED SERVICE ACCOUNTS SECURITY

Retrieving MSA Passwords

- Only with the system account that has permissions
- 256 bit random password
- Not stored locally*

```
PS C:\windows\system32> Get-ADServiceAccount msa-records -Properties 'msDS-ManagedPassword'


DistinguishedName : CN=MSA-Records,CN=Managed Service Accounts,DC=corp,DC=JPS,DC=com
Enabled           : True
Name              : MSA-Records
ObjectClass       : msDS-GroupManagedServiceAccount
ObjectGUID        : 6cea25df-0dc8-4f88-a66e-a1620dbb8385
SamAccountName    : MSA-Records$
SID               : S-1-5-21-3489131439-3593545028-3395961476-1127
UserPrincipalName :



PS C:\windows\system32>
```

Run with highest privileges

```
Windows PowerShell
Copyright (C) 2013 Microsoft Corporation. All rights reserved.

PS C:\windows\system32> Get-ADServiceAccount msa-records -Properties 'msDS-Manag
edPassword'


DistinguishedName     : CN=MSA-Records,CN=Managed Service
                        Accounts,DC=corp,DC=JPS,DC=com
Enabled               : True
msDS-ManagedPassword  : {1, 0, 0, 0...}
Name                  : MSA-Records
ObjectClass           : msDS-GroupManagedServiceAccount
ObjectGUID            : 6cea25df-0dc8-4f88-a66e-a1620dbb8385
SamAccountName        : MSA-Records$
SID                   : S-1-5-21-3489131439-3593545028-3395961476-1127
UserPrincipalName     :
```

MMS

OVER-THE-SHOULDER LOGGING

- Event Logging:
  - Some executions of PowerShell are logged in the event viewer.
- Task manager:
  - The command line can be exposed in the task manager.

PEER REVIEW

- There is a lot to consider!
- Learn from your teammates!

SQL INJECTION

What is it and how do you prevent it?

MMS

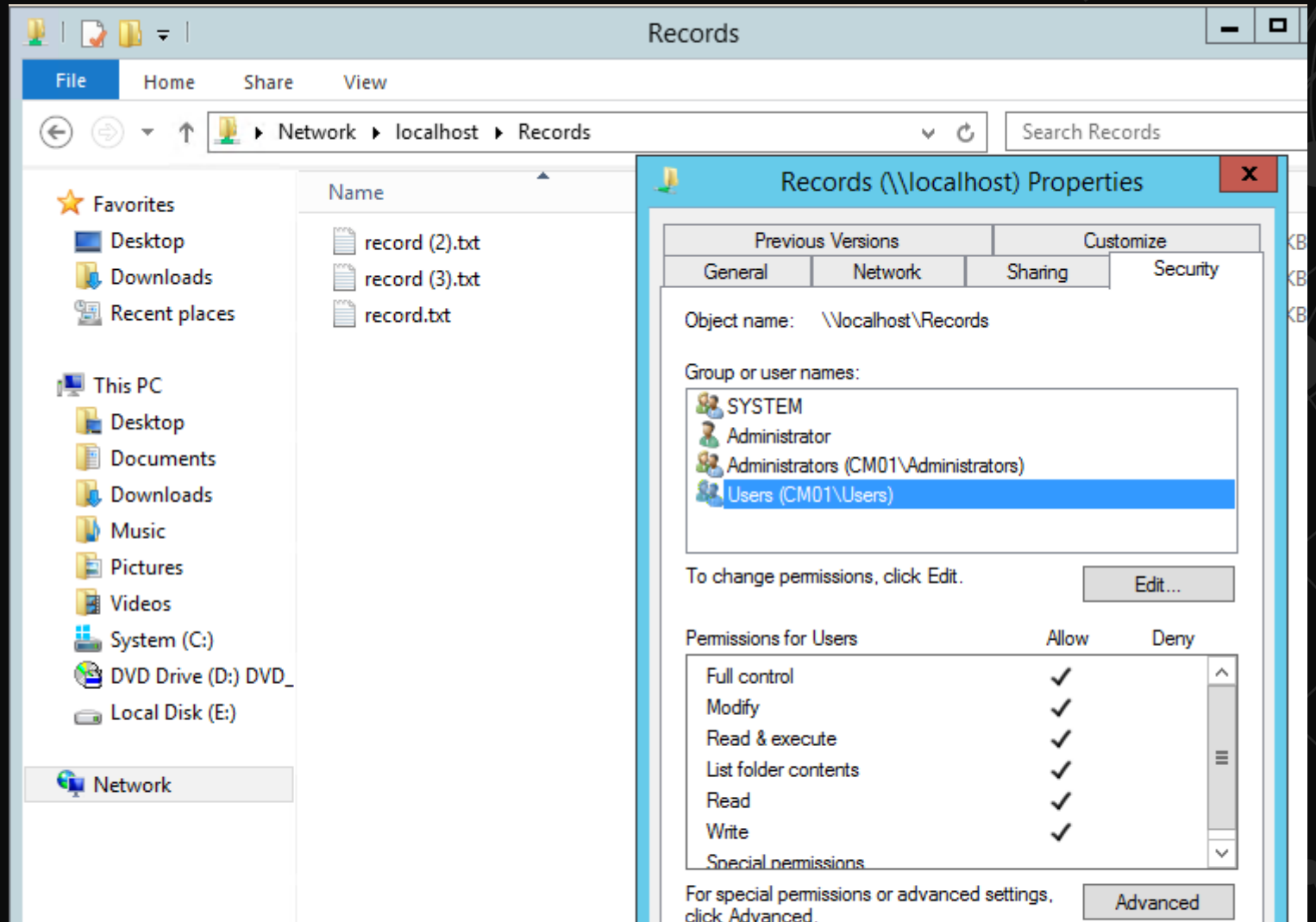
SQL INJECTION

- When are you at risk:
  - When Users input data that is intended for a SQL query.

- How do you prevent this:
  - Parameterization – This reformats the sql query to automatically sanitize inputs
  - Stored Procedures – Using stored procedures doesn't prevent sql injection but it allows you to restrict permissions

MMS

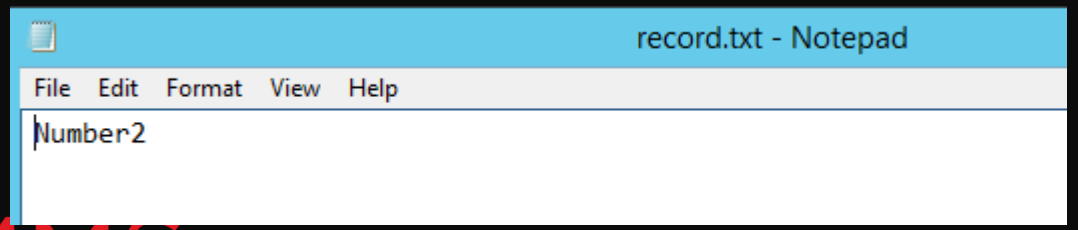
EXAMPLE


Records
File
Home
Share
View
Network ▸ localhost ▸ Records
Search Records
Favorites
Desktop
Downloads
Recent places
This PC
Desktop
Documents
Downloads
Music
Pictures
Videos
System (C:)
DVD Drive (D:) DVD_
Local Disk (E:)
Network
Name
record (2).txt
record (3).txt
record.txt
Records (\\localhost) Properties
Previous Versions
Customize
General
Network
Sharing
Security
Object name: \\localhost\Records
Group or user names:
SYSTEM
Administrator
Administrators (CM01\Administrators)
Users (CM01\Users)
To change permissions, click Edit.
Edit...
Permissions for Users
Allow
Deny
Full control
Modify
Read & execute
List folder contents
Read
Write
Special permissions
For special permissions or advanced settings, click Advanced.
Advanced

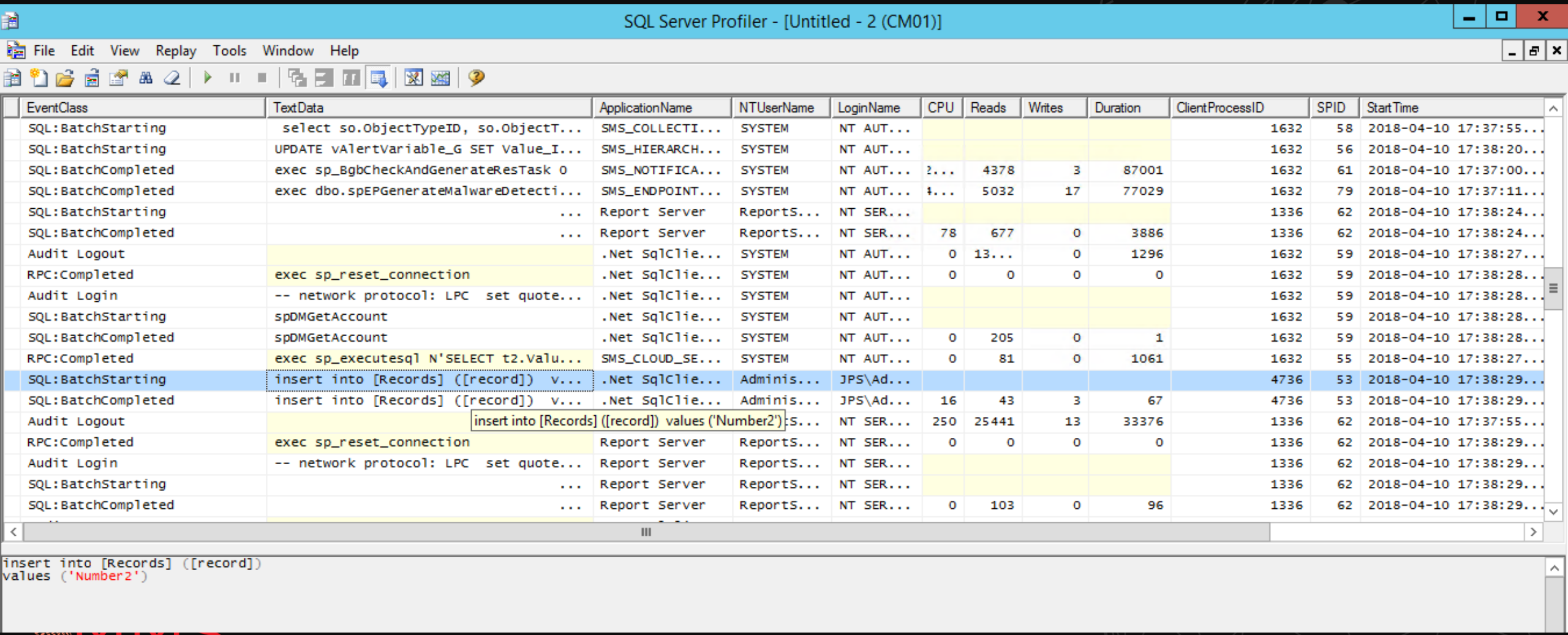
EXAMPLE

Register-Params.ps1 | Register.ps1

```powershell
$PWD = ConvertTo-SecureString -String 'P@sswOrd' -AsPlainText -Force
$cred = new-object pscredential ('jps\administrator',$pwd)
New-PSDrive -Name R -PSProvider FileSystem -Root \\dc01\c$ -Credential $cred

$record = Get-Content -path c:\records\Record.txt

$query = "insert into [Records] ([record])
values ('$record')"

$SQLConnection = New-object System.Data.SqlClient.SqlConnection('Data Source=cm01;Integrated security = true; Initial catalog = re
$SQLConnection.open()

$SQLInsert = new-object System.Data.SqlClient.SqlCommand($query,$SQLConnection)
$Null = $SQLInsert.ExecuteNonQuery()

copy-item C:\Records\record.txt -Destination \\dc01\c$\

Remove-PSDrive -Name R
```



EXAMPLE


SQL Server Profiler - [Untitled - 2 (CM01)]
File Edit View Replay Tools Window Help
EventClass | TextData | ApplicationName | NTUserName | LoginName | CPU | Reads | Writes | Duration | ClientProcessID | SPID | StartTime
SQL:BatchStarting | select so.ObjectTypeID, so.ObjectT... | SMS_COLLECTI... | SYSTEM | NT AUT... | | | | | 1632 | 58 | 2018-04-10 17:37:55...
SQL:BatchStarting | UPDATE vAlertVariable_G SET Value_I... | SMS_HIERARCH... | SYSTEM | NT AUT... | | | | | 1632 | 56 | 2018-04-10 17:38:20...
SQL:BatchCompleted | exec sp_BgbCheckAndGenerateResTask 0 | SMS_NOTIFICA... | SYSTEM | NT AUT... | ?... | 4378 | 3 | 87001 | 1632 | 61 | 2018-04-10 17:37:00...
SQL:BatchCompleted | exec dbo.spEPGenerateMalwareDetecti... | SMS_ENDPOINT... | SYSTEM | NT AUT... | 4... | 5032 | 17 | 77029 | 1632 | 79 | 2018-04-10 17:37:11...
SQL:BatchStarting | ... | Report Server | ReportS... | NT SER... | | | | | 1336 | 62 | 2018-04-10 17:38:24...
SQL:BatchCompleted | ... | Report Server | ReportS... | NT SER... | 78 | 677 | 0 | 3886 | 1336 | 62 | 2018-04-10 17:38:24...
Audit Logout | | .Net SqlClie... | SYSTEM | NT AUT... | 0 | 13... | 0 | 1296 | 1632 | 59 | 2018-04-10 17:38:27...
RPC:Completed | exec sp_reset_connection | .Net SqlClie... | SYSTEM | NT AUT... | 0 | 0 | 0 | 0 | 1632 | 59 | 2018-04-10 17:38:28...
Audit Login | -- network protocol: LPC set quote... | .Net SqlClie... | SYSTEM | NT AUT... | | | | | 1632 | 59 | 2018-04-10 17:38:28...
SQL:BatchStarting | spDMGetAccount | .Net SqlClie... | SYSTEM | NT AUT... | | | | | 1632 | 59 | 2018-04-10 17:38:28...
SQL:BatchCompleted | spDMGetAccount | .Net SqlClie... | SYSTEM | NT AUT... | 0 | 205 | 0 | 1 | 1632 | 59 | 2018-04-10 17:38:28...
RPC:Completed | exec sp_executesql N'SELECT t2.Valu... | SMS_CLOUD_SE... | SYSTEM | NT AUT... | 0 | 81 | 0 | 1061 | 1632 | 55 | 2018-04-10 17:38:27...
SQL:BatchStarting | insert into [Records] ([record]) v... | .Net SqlClie... | Adminis... | JPS\Ad... | | | | | 4736 | 53 | 2018-04-10 17:38:29...
SQL:BatchCompleted | insert into [Records] ([record]) v... | .Net SqlClie... | Adminis... | JPS\Ad... | 16 | 43 | 3 | 67 | 4736 | 53 | 2018-04-10 17:38:29...
Audit Logout | | insert into [Records] ([record]) values ('Number2') | NT SER... | 250 | 25441 | 13 | 33376 | 1336 | 62 | 2018-04-10 17:37:55...
RPC:Completed | exec sp_reset_connection | Report Server | ReportS... | NT SER... | 0 | 0 | 0 | 0 | 1336 | 62 | 2018-04-10 17:38:29...
Audit Login | -- network protocol: LPC set quote... | Report Server | ReportS... | NT SER... | | | | | 1336 | 62 | 2018-04-10 17:38:29...
SQL:BatchStarting | ... | Report Server | ReportS... | NT SER... | | | | | 1336 | 62 | 2018-04-10 17:38:29...
SQL:BatchCompleted | ... | Report Server | ReportS... | NT SER... | 0 | 103 | 0 | 96 | 1336 | 62 | 2018-04-10 17:38:29...
insert into [Records] ([record])
values ('Number2')

EXAMPLE

record (2).txt - Notepad

File Edit Format View Help

```
Number3'); Delete from records --
```

SQL Server Profiler - [Untitled - 2 (CM01)]

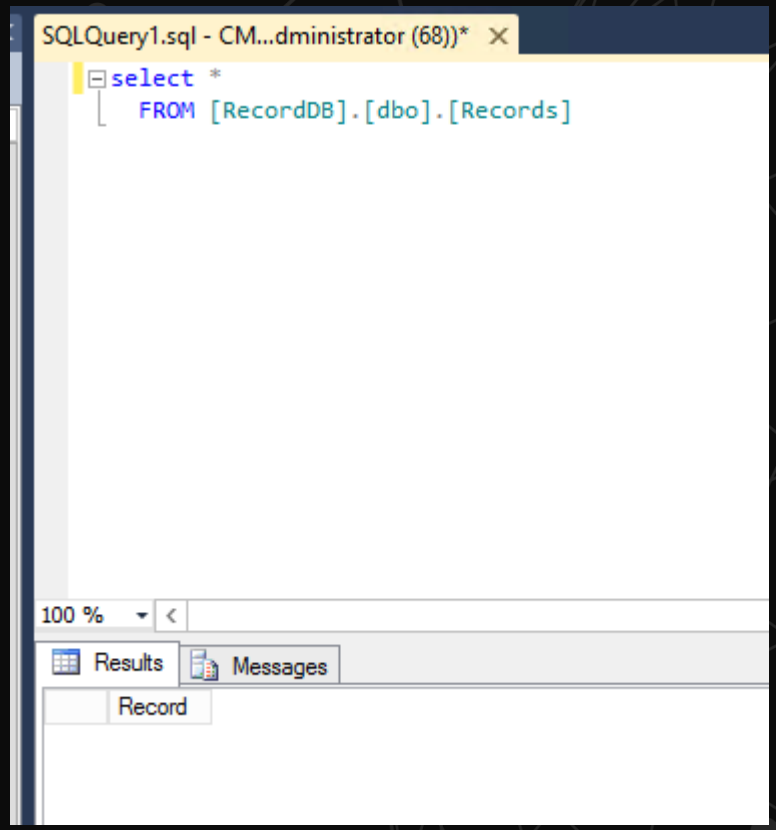
File Edit View Replay Tools Window Help

EventClass	TextData	ApplicationName	NTUserName	LoginName	CPU	Reads	Writes	Duration	ClientProcessID	SPID	StartTime
SQL:BatchStarting	select DateLastModified from SiteCo...	SMS_OBJECT_R...	SYSTEM	NT AUT...					1632	78	2018-04-10 17:
SQL:BatchCompleted	select DateLastModified from SiteCo...	SMS_OBJECT_R...	SYSTEM	NT AUT...	0	117	0	15	1632	78	2018-04-10 17:
SQL:BatchCompleted	exec sp_BgbCheckAndGenerateResTask 0	SMS_NOTIFICA...	SYSTEM	NT AUT...	63	2808	0	386	1632	67	2018-04-10 17:
Audit Logout		.Net SqlClie...	SYSTEM	NT AUT...	0	15069	0	1293	1632	60	2018-04-10 17:
RPC:Completed	exec sp_reset_connection	.Net SqlClie...	SYSTEM	NT AUT...	0	0	0	0	1632	60	2018-04-10 17:
Audit Login	-- network protocol: LPC set quote...	.Net SqlClie...	SYSTEM	NT AUT...					1632	60	2018-04-10 17:
SQL:BatchStarting	spDMGetAccount	.Net SqlClie...	SYSTEM	NT AUT...					1632	60	2018-04-10 17:
SQL:BatchCompleted	spDMGetAccount	.Net SqlClie...	SYSTEM	NT AUT...	0	205	0	1	1632	60	2018-04-10 17:
Audit Logout		.Net SqlClie...	SYSTEM	NT AUT...	0	15274	0	1030	1632	60	2018-04-10 17:
RPC:Completed	exec sp_reset_connection	.Net SqlClie...	SYSTEM	NT AUT...	0	0	0	0	1632	60	2018-04-10 17:
Audit Login	-- network protocol: LPC set quote...	.Net SqlClie...	SYSTEM	NT AUT...					1632	60	2018-04-10 17:
SQL:BatchStarting	spDMGetAccount	.Net SqlClie...	SYSTEM	NT AUT...					1632	60	2018-04-10 17:
SQL:BatchCompleted	spDMGetAccount	.Net SqlClie...	SYSTEM	NT AUT...	0	205	0	1	1632	60	2018-04-10 17:
Audit Logout		.Net SqlClie...	Adminis...	JPS\Ad...	16	143	3	229296	4736	53	2018-04-10 17:
RPC:Completed	exec sp_reset_connection	.Net SqlClie...	Adminis...	JPS\Ad...	0	0	0	0	4736	53	2018-04-10 17:
Audit Login	-- network protocol: LPC set quote...	.Net SqlClie...	Adminis...	JPS\Ad...					4736	53	2018-04-10 17:
SQL:BatchStarting	insert into [Records] ([record]) v...	.Net SqlClie...	Adminis...	JPS\Ad...					4736	53	2018-04-10 17:
SQL:BatchCompleted	insert into [Records] ([record]) v...	.Net SqlClie...	Adminis...	JPS\Ad...	0	46	2	14	4736	53	2018-04-10 17:
Audit Login	-- network protocol: LPC set quote...	SMS_STATE_SY...	SYSTEM	NT AUT...					1632	74	2018-04-10 17:

```
insert into [Records] ([record])
values ('Number3'); Delete from records --')
```

EXAMPLE

Other Attacks:
- Drop database
- Add permissions for user
- Change records
- Create Backups



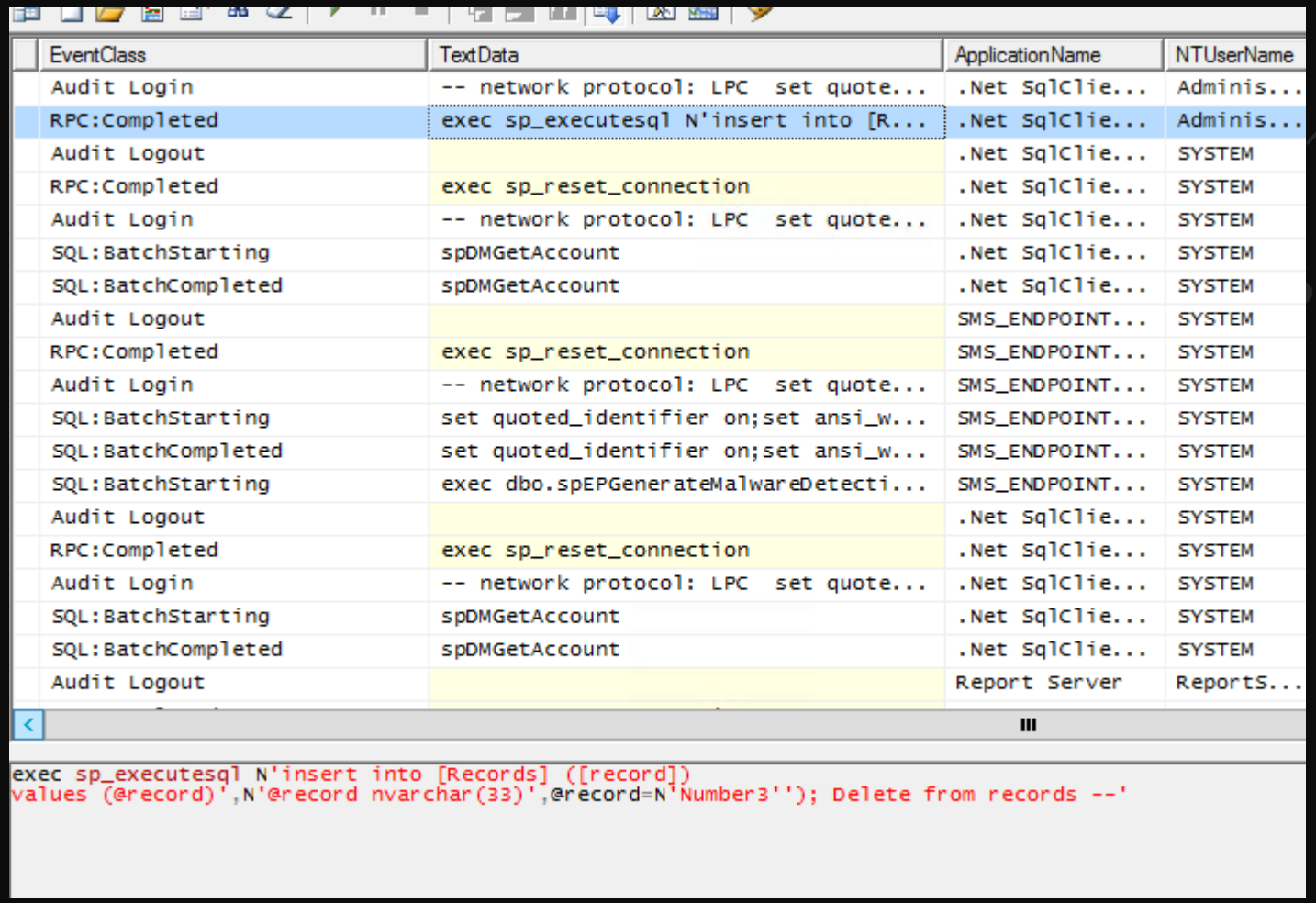
PARAMETERIZATION

Register-Params.ps1 | Register.ps1

```powershell
$PWD = ConvertTo-SecureString -String 'P@sswOrd' -AsPlainText -Force
$cred = new-object pscredential ('jps\administrator',$pwd)
New-PSDrive -Name R -PSProvider FileSystem -Root \\dc01\c$ -Credential $cred

$record = Get-Content -path c:\records\Record.txt

$query = "insert into [Records] ([record])
values ('$record')"

$SQLConnection = New-object System.Data.SqlClient.SqlConnection('Data Source=cm01;Integrated security = true; Initial catalog = re
$SQLConnection.open()

$SQLInsert = new-object System.Data.SqlClient.SqlCommand($query,$SQLConnection)
$Null = $SQLInsert.ExecuteNonQuery()

copy-item C:\Records\record.txt -Destination \\dc01\c$\

Remove-PSDrive -Name R
```

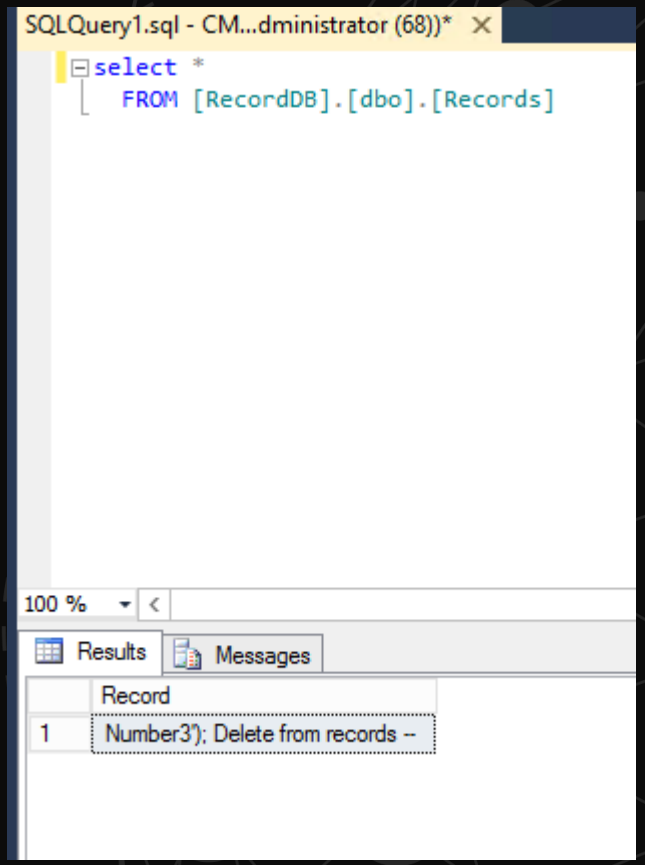
```powershell
$PWD = ConvertTo-SecureString -String 'P@sswOrd' -AsPlainText -Force
$cred = new-object pscredential ('jps\administrator',$pwd)
New-PSDrive -Name R -PSProvider FileSystem -Root \\dc01\c$ -Credential $cred

$record = Get-Content -path c:\records\Record.txt

$query = "insert into [Records] ([record])
values (@record)"

$SQLConnection = New-object System.Data.SqlClient.SqlConnection('Data Source=cm01;Integrated security = true; Initial catalog = re
$SQLConnection.open()

$SQLInsert = new-object System.Data.SqlClient.SqlCommand($query,$SQLConnection)
$SQLInsert.Parameters.Add('@record',$record)
$SQLInsert.ExecuteNonQuery()

copy-item C:\Records\record.txt -Destination \\dc01\c$\

Remove-PSDrive -Name R
```

PARAMETERIZATION



EventClass	TextData	ApplicationName	NTUserName
Audit Login	-- network protocol: LPC set quote...	.Net SqlClie...	Adminis...
RPC:Completed	exec sp_executesql N'insert into [R...	.Net SqlClie...	Adminis...
Audit Logout		.Net SqlClie...	SYSTEM
RPC:Completed	exec sp_reset_connection	.Net SqlClie...	SYSTEM
Audit Login	-- network protocol: LPC set quote...	.Net SqlClie...	SYSTEM
SQL:BatchStarting	spDMGetAccount	.Net SqlClie...	SYSTEM
SQL:BatchCompleted	spDMGetAccount	.Net SqlClie...	SYSTEM
Audit Logout		SMS_ENDPOINT...	SYSTEM
RPC:Completed	exec sp_reset_connection	SMS_ENDPOINT...	SYSTEM
Audit Login	-- network protocol: LPC set quote...	SMS_ENDPOINT...	SYSTEM
SQL:BatchStarting	set quoted_identifier on;set ansi_w...	SMS_ENDPOINT...	SYSTEM
SQL:BatchCompleted	set quoted_identifier on;set ansi_w...	SMS_ENDPOINT...	SYSTEM
SQL:BatchStarting	exec dbo.spEPGenerateMalwareDetecti...	SMS_ENDPOINT...	SYSTEM
Audit Logout		.Net SqlClie...	SYSTEM
RPC:Completed	exec sp_reset_connection	.Net SqlClie...	SYSTEM
Audit Login	-- network protocol: LPC set quote...	.Net SqlClie...	SYSTEM
SQL:BatchStarting	spDMGetAccount	.Net SqlClie...	SYSTEM
SQL:BatchCompleted	spDMGetAccount	.Net SqlClie...	SYSTEM
Audit Logout		Report Server	ReportS...

```
exec sp_executesql N'insert into [Records] ([record])
values (@record)',N'@record nvarchar(33)',@record=N'Number3''); Delete from records --'
```



MMS

STORED PROCEDURES

What does it not do:

- Prevent Injection attacks

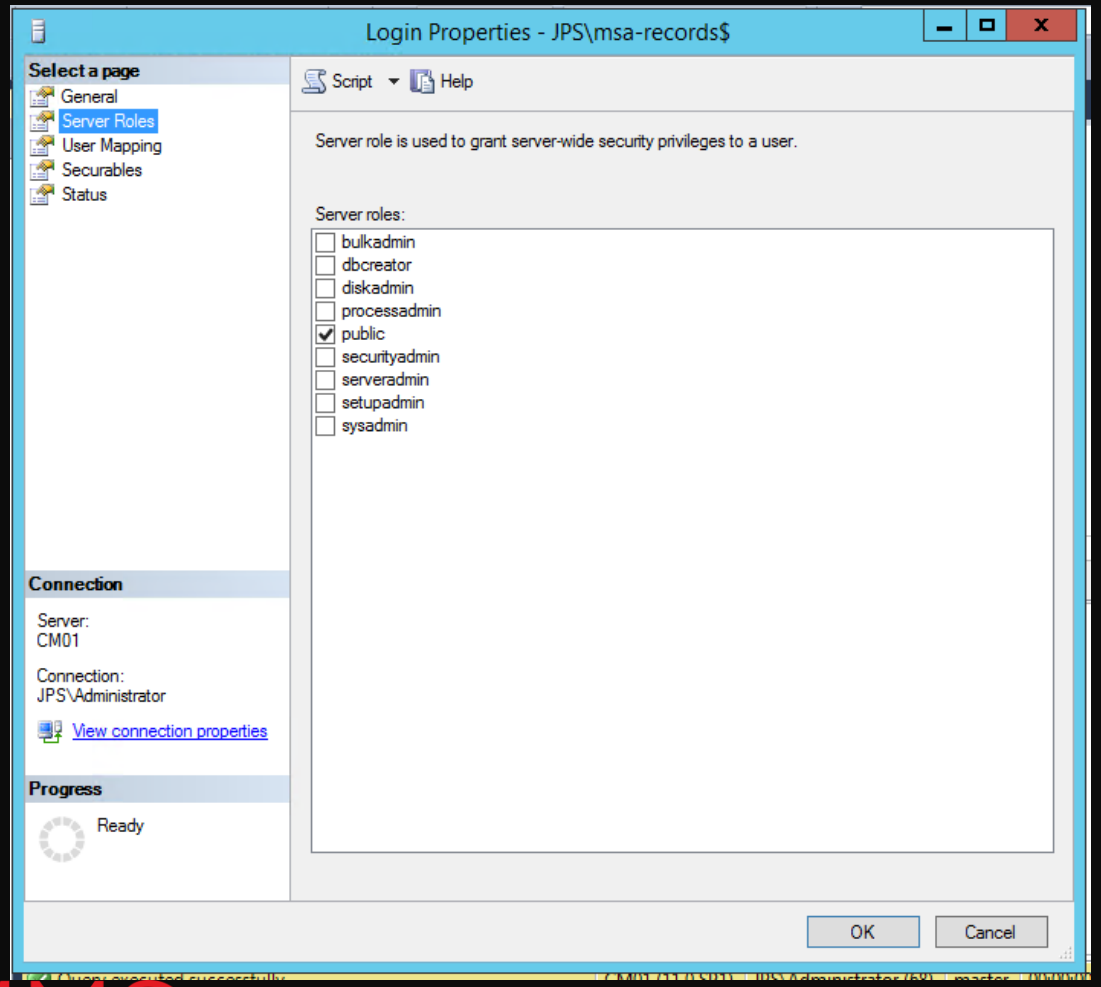
What does it do:

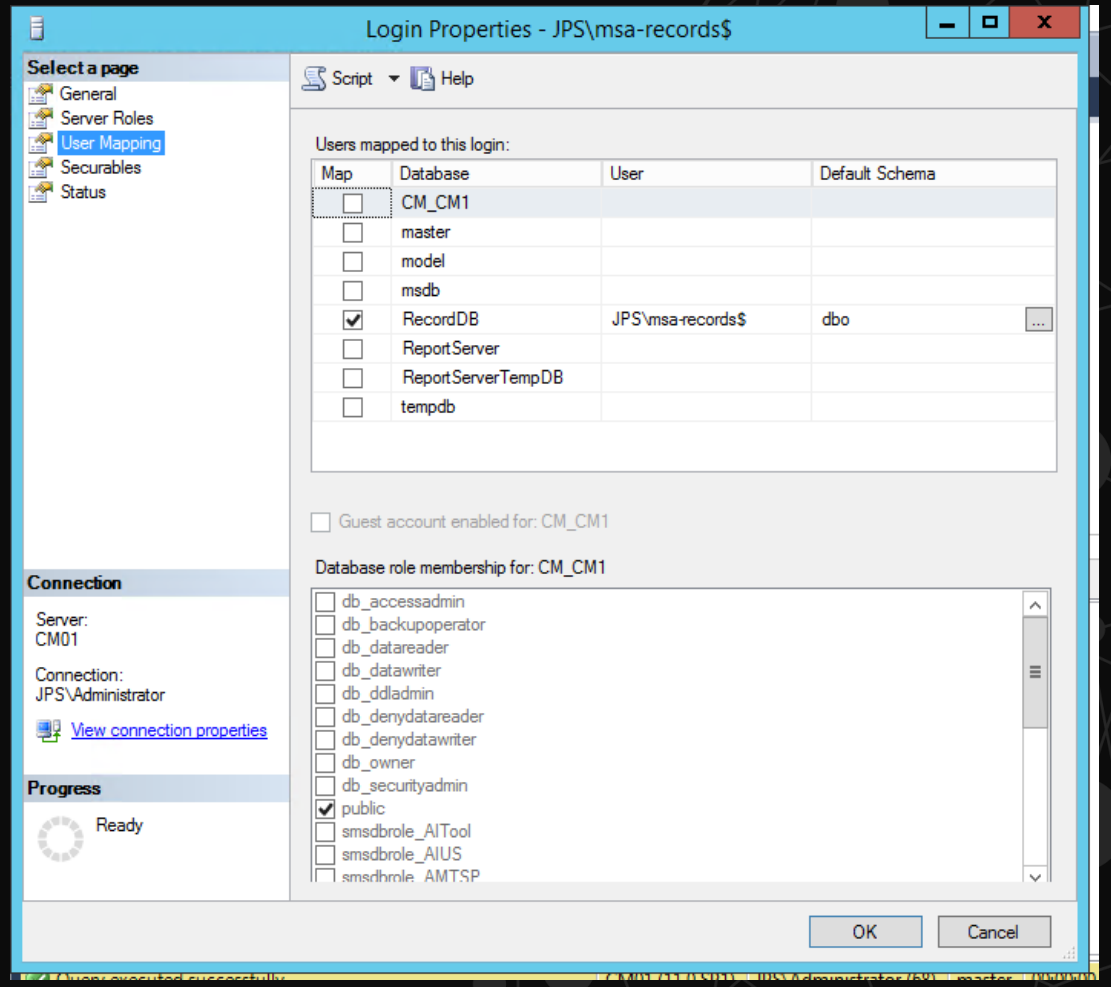
- Give you granular permissions for specific SQL tasks

```
USE [RecordDB]
GO
/****** Object:  StoredProcedure [dbo].[InsertRecor
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:      <Author,,Name>
-- Create date: <Create Date,,>
-- Description: <Description,,>
-- =============================================
ALTER PROCEDURE [dbo].[InsertRecord] (
@Record varchar(50)
)
AS
BEGIN
insert into [Records] ([record])
values (@record)
END
```

MMS

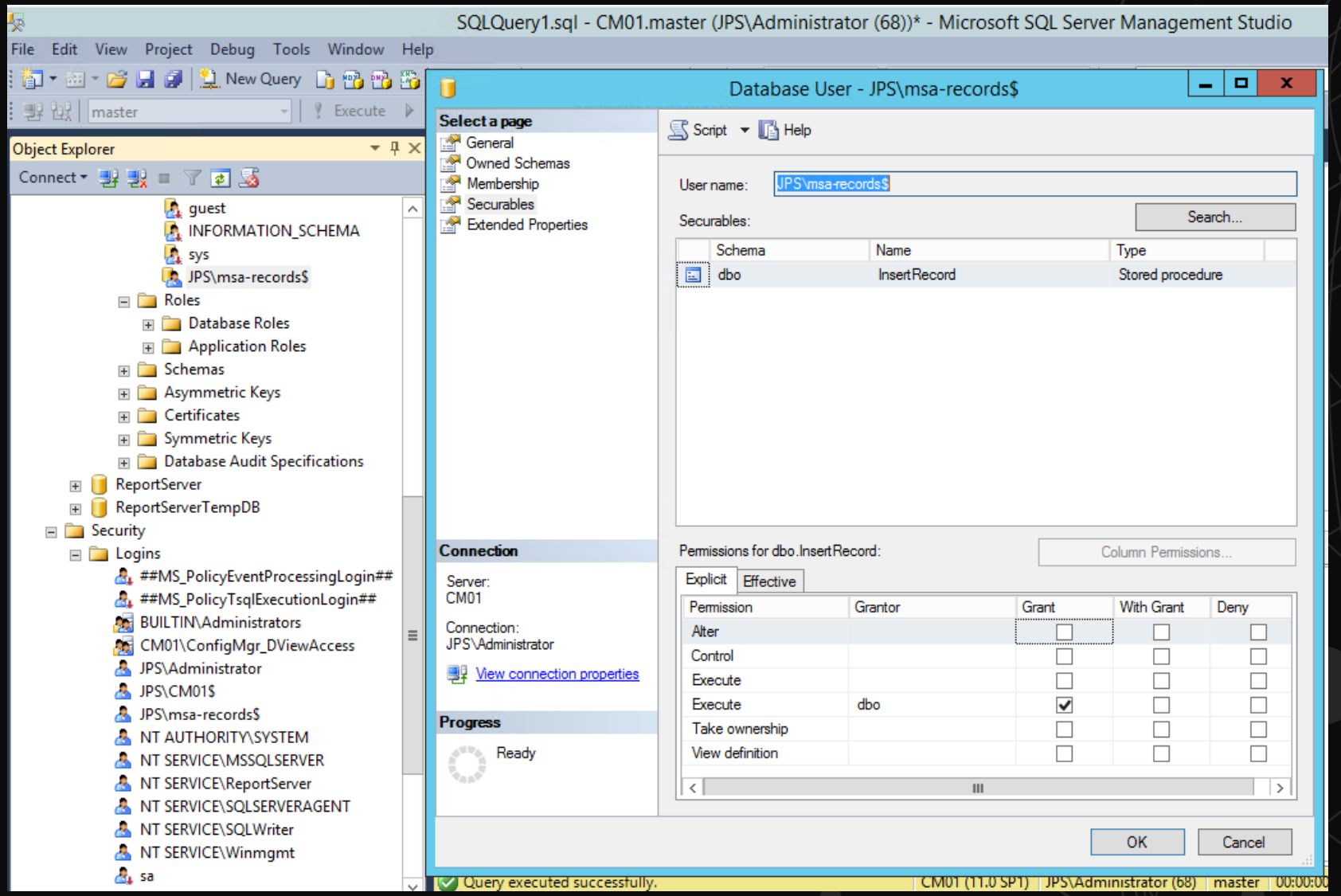
STORED PROCEDURES





MMS

STORED PROCEDURES



STORED PROCEDURES

```
$record = Get-Content -path c:\records\Record.txt

$query = "[dbo].[InsertRecord] '$record'"

$SQLConnection = New-object System.Data.SqlClient.SqlConnection('Data Source=cm01;Integrated security = true; Ini
$SQLConnection.open()

$SQLInsert = new-object System.Data.SqlClient.SqlCommand($query,$SQLConnection)
$SQLInsert.ExecuteNonQuery()

copy-item C:\Records\record.txt -Destination \\dc01\c$\
```

```
\windows\system32> $SQLConnection = New-object System.Data.SqlClient.SqlConnection('Data Source=cm01;Integrated se

\windows\system32> $SQLConnection.open()

\windows\system32> $SQLInsert = new-object System.Data.SqlClient.SqlCommand($query,$SQLConnection)

\windows\system32> $SQLInsert.ExecuteNonQuery()
tion calling "ExecuteNonQuery" with "0" argument(s): "The DELETE permission was denied on the object 'Records',
ase 'RecordDB', schema 'dbo'."
ne:1 char:1
LInsert.ExecuteNonQuery()
~~~~~~~~~~~~~~~~~~~~~~~~~
 CategoryInfo          : NotSpecified: (:) [], MethodInvocationException
 FullyQualifiedErrorId : SqlException


\windows\system32>
```

MMS

ENCRYPTION

How to protect shared secrets and sensitive data.

MMS

ENCRYPTION

- It is always best to not store data that you do not need anymore.
  - Use the running context as much as possible for permissions.
- Eventually storing sensitive data is unavoidable.
  - Encryption is no good if your keys and certificates are not secure.
- There are already great tools built into PowerShell and .NET but there are more advanced options if you need.

OBFUSCATION

- THIS IS NOT ENCRYPTION!
- Base64 encoding is the most common in PowerShell.
  - This is very useful for encoding data for storage
- This is fast!
  - Encoding 10,000 times ~0.2 seconds
  - Decoding 10,000 times ~0.18 seconds

OBFUSCATION

- **Encoding**:

```
$pwd = 'Password123'
$bytes =  [System.Text.Encoding]::UTF8.GetBytes($pwd)
$Base64String = [Convert]::ToBase64String($bytes)
$Base64String|clip
$Base64String
```

- **Decoding**:

```
$ConvertedBytes = [Convert]::FromBase64String('UGFzc3dvcmQxMjM=')
[System.Text.Encoding]::UTF8.Getstring($ConvertedBytes)
```

MMS

SYMMETRIC ENCRYPTION

- This is a common and well respected encryption solution.
  - Disk encryption
  - File encryption
  - SQL encryption
  - Data Protection API

- Speeds

(Convertfrom-securestring)Encoding 10,000 times ~4.5 seconds

(Convertfrom-securestring)Encoding Decoding 10,000 times ~3.7 seconds

(.Net)Encoding 10,000 times ~1 seconds

(.Net)Encoding Decoding 10,000 times ~1.06 seconds

SYMMETRIC ENCRYPTION

- **Encoding**:

```
$Userinput = 'someParam2Secure'
$keybytes = [System.Text.Encoding]::UTF8.GetBytes($Userinput)
[Byte[]] $Key = [System.Security.Cryptography.HashAlgorithm]::Create('SHA256').ComputeHash($keybytes)
$secure = ConvertTo-SecureString -String 'Password123' -AsPlainText -Force
ConvertFrom-SecureString -SecureString $secure -Key $key
```

- **Decoding**:

```
$securestring2 = ConvertTo-SecureString -Key $key -String
'76492d1116743f0423413b16050a5345MgB8AGMASwBCAE0ASABWAEEASABCAGkAZgBLAGgAUABrAFcA
MwBGAEEAcgB5AEEAPQA9AHwAMAAwADcAMQBlADkAMwBhAGUAZABlAGEAMgBjADAANwBiADkAYQA
zADkANQBkADIANgBiADIANgAzAGQAZgBlAA=='
[System.Runtime.InteropServices.marshal]::PtrToStringAuto([System.Runtime.InteropServices.marshal]::Secu
reStringToBSTR($securestring))
```

MMS

DPAPI ENCRYPTION

- This is the mechanism that is used by secure data in the windows OS.

Local Machine - Local WIFI Passwords, windows services passwords, Certificate Private keys

Current User - IE saved passwords, Credential Manager, Certificate Private keys, App Passwords (Chrome, Skype Dropbox, Icloud)

- This uses user password hash and system information to generate 256 bit keys.

- Speeds

(Convertfrom-securestring)Encoding 10,000 times ~10 seconds

(Convertfrom-securestring)Encoding Decoding 10,000 times ~8.3 seconds

(.Net)Encoding 10,000 times ~8.6 seconds

(.Net)Encoding Decoding 10,000 times ~8.6 seconds

MMS

DPAPI ENCRYPTION

- **Encoding**:

```
$securestring = ConvertTo-SecureString -String 'Password123' -AsPlainText -Force
ConvertFrom-SecureString -SecureString $securestring
```

- **Decoding**:

```
ConvertTo-SecureString
'01000000d08c9ddf0115d1118c7a00c04fc297eb01000000010a0f4cb8ab5a42a48998a4aa3754
d900000000020000000000106600000001000020000000ff6cfd739a8c52f3f06ff44c2089b356af9
52fc12ab9daec787377a90dfc9f51000000000e8000000020000200000034a67856119722d27a
3d0663d439373358e13cd2b2200802c16c8ad7084caa8f200000005e1053fa55aa396212a0dbbb
5e29cc947cf293a18a9310a0bcf6369ccd81d286400000003c1c7887c32c25c6e240761bedad3be
a05d68b4edb2b4157f3a3cda2e3062cdcb1fdd9b41b17db1f917706c27abe9cde1e4ece48800eb
85a867b3e263b916422'

[System.Runtime.InteropServices.marshal]::PtrToStringAuto([System.Runtime.InteropServices.ma
rshal]::SecureStringToBSTR($securestring))
```

ASYMMETRIC ENCRYPTION

- This is mainly useful when encryption and decryption need to be controlled separately.
  - SSL
  - Signing
- Try to use a trusted CA
- This is less common in PowerShell
- Speeds

(.Net)Encoding 10,000 times ~1.2 seconds

(.Net)Encoding Decoding 10,000 times ~17.9 seconds

ASYMMETRIC ENCRYPTION

- **Encoding**:

```
$cert = New-SelfSignedCertificate -Subject "Encrypt" -KeyUsage KeyEncipherment, DataEncipherment -Provider "Microsoft Enhanced RSA and AES Cryptographic Provider" -CertStoreLocation Cert:\CurrentUser\my

$pwd = 'Password123'
#$cert = get-item Cert:\CurrentUser\my\ABE7B616CF9858235072CE715A0DCC5F5436107A
$bytes =  [System.Text.Encoding]::UTF8.GetBytes($pwd)
$encryptedblob = $cert.PublicKey.Key.Encrypt($bytes,$true)
$EncryptedBase64String = [Convert]::ToBase64String($encryptedblob)
```

- **Decoding**:

```
$encryptedBytes = [System.Convert]::FromBase64String($EncryptedBase64String)
$pwdbytes = $cert.PrivateKey.Decrypt($encryptedBytes, $true)
[System.Text.Encoding]::UTF8.GetString($pwdbytes )
```

ENCRYPTION

- Which Encryption method is the best:
  - It depends

- I still have code that can expose passwords
  - Encryption separates the pieces needed to retrieve sensitive data.
  - Find an operation method for passing the key and certificates into the script. (Parameters, Retrieve from a management tool, etc)

- Can Multiple encryption solutions be used:
  - Yes, but try not to add complexity unnecessarily

Review

- Operation Security
  - Peer Reviews are important
- SQL Injection
  - This is a common attack vector.
  - There are easy steps to prevent it.
- Encryption
  - Encryption will not make lazy coding secure.
  - This is an important piece of a secure process.

Extended Q&A


1E


adaptiva


DELL


Kollective


Microsoft


vmware


CTGlobal


nowmicro


cireson
your system center experts


2Pint
Software


FLEXERA


Apajove


Lenovo

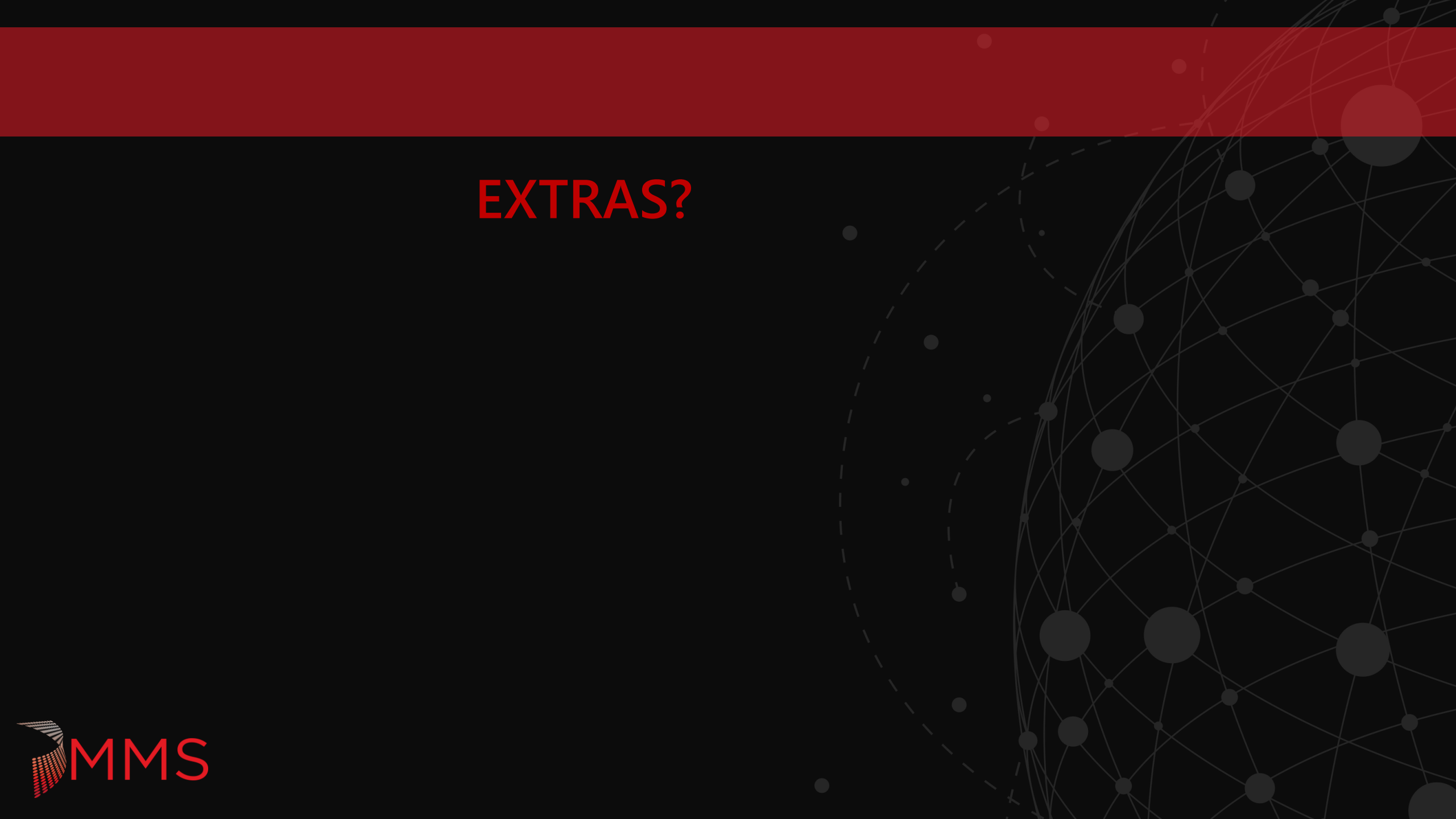

PolicyPak
SECURING YOUR STANDARDS


Parallels


Patch My PC
PATCH MANAGEMENT MADE EASY

MMS

EXTRAS?



DPAPI ENCRYPTION – CREDENTIAL MANAGER

- **Is the Credential Manager a good place for Passwords:**
  - **It depends**
  - **The API for Credential manager is standard and easy to use and an obvious spot for hackers**
  - **There is no way to add entropy**

https://github.com/davotronic5000/PowerShell_Credential_Manager

MEMORY PROTECTION

- This is used by application with extremely sensitive data
- Requires Specific 16 byte chunks
- This is less common in PowerShell
- Encrypt the data in memory. The result is stored in the same array as the original data.

MEMORY PROTECTION ENCRYPTION

- **Encoding**:

```
$pwd = 'Password'
[byte[]]$Bytes = [System.Text.Encoding]::Unicode.GetBytes($pwd)
$Bytes
[System.Security.Cryptography.ProtectedMemory]::Protect($Bytes,
[System.Security.Cryptography.MemoryProtectionScope]::SameProcess)
$Bytes
```

- **Decoding**:

```
[System.Security.Cryptography.ProtectedMemory]::unProtect($Bytes,
[System.Security.Cryptography.MemoryProtectionScope]::SameProcess)
$Bytes
```

DPAPI ENCRYPTION -.NET

- **Encoding**:

```
$pwd = 'Password123'

$Bytes = [System.Text.Encoding]::Unicode.GetBytes($pwd)

$LMProtectedBytes = [System.Security.Cryptography.ProtectedData]::Protect($Bytes, $null, [System.Security.Cryptography.DataProtectionScope]::LocalMachine)

$CUProtectedBytes = [System.Security.Cryptography.ProtectedData]::Protect($Bytes, $null, [System.Security.Cryptography.DataProtectionScope]::CurrentUser)

$LMEncryptedString = [Convert]::ToBase64String($lmProtectedBytes)

$LMEncryptedString |clip

$LMEncryptedString

$CUEncrypted4String = [Convert]::ToBase64String($CUProtectedBytes)

$CUEncrypted4String|clip

$CUEncrypted4String
```

- **Decoding**:

```
'01000000d08c9ddf0115d1118c7a00c04fc297eb01000000010a0f4cb8ab5a42a48998a4aa3754d900000000020000000000106600000001000020000000ff6cfd739a8c52f3f06ff44c2089b356af952fc12ab9daec787377
a90dfc9f51000000$LMEncryptedBytes=
[Convert]::FromBase64String('AQAAANCMnd8BFdERjHoAwE/Cl+sBAAAACRDlWpWBPkWMPcyvU/E8FAQAAAACAAAAAAAQZgAAAAEAACAAAAD5y0XdgLmhV7PlNeq/qx8DMbfpTzIqIdMZaU5Enu65LAAAAAAOgAAAAAIAACAAAABZczDuLNNvGX
gzkU3zvDqM+M762/wwa6XSCbeU3sYxEiAAAAAnWf+zlZVdJVb46QmbCo9/VIbnS7OQjIMySx1OWJbxOkAAAABIsXtH7nlfGqGJY4Dj1RNmdcVd1blQKq15t9UwWYcPXAcB+tB3hpKvcl/MsTWJjr5ha8QRFdWCcAxH4aDc7r/S')

$LMDecryptedBytes = [System.Security.Cryptography.ProtectedData]::Unprotect($LMEncryptedBytes, $null, [System.Security.Cryptography.DataProtectionScope]::LocalMachine)

[System.Text.Encoding]::Unicode.GetString($LMDecryptedBytes)


$CUEncryptedBytes=
[Convert]::FromBase64String('AQAAANCMnd8BFdERjHoAwE/Cl+sBAAAAAQoPTLirWkKkiZikqjdU2QAAAAACAAAAAAAQZgAAAAEAACAAAAC3YgkN7d85y3q3O3Mn6YXyOh1T5EfU4GMg8ohN6RZECAAAAAAOgAAAAAIAACAAAAA1MaVMjrlfYG8J5
lDCtTY39Hmp7hc8QflbSUiRhKbsFCAAAADallqxWy9HeUw2FoIF5A3LoZ9mMrl8nqikyIm8em9+GkAAAAC8/cq8QTFod93Mbbl0ksNI40Uw5EcH0aegk8AxNgWT7penxgYfSZPIfZ2xbUTDr//i1adtyLDTt/P4LOx+R76+')

$CUDecryptedBytes = [System.Security.Cryptography.ProtectedData]::Unprotect($CUEncryptedBytes, $null, [System.Security.Cryptography.DataProtectionScope]::CurrentUser)

[System.Text.Encoding]::Unicode.GetString($CUDecryptedBytes)
```

MMS

DPAPI ENCRYPTION - SUBSYSTEM

- Local system

%WINDIR%/System32/Microsoft/Protect

- Current User – Semi Portable

%appdata%\Microsoft\Protect

SYMMETRIC ENCRYPTION - .NET

- **Encoding**:

```
$pwd = 'Password123'
$bytes =  [System.Text.Encoding]::UTF8.GetBytes($pwd)

$Userinput = 'this is not the password but something else!'
$keybytes = [System.Text.Encoding]::UTF8.GetBytes($Userinput)
[Byte[]] $Key = [System.Security.Cryptography.HashAlgorithm]::Create('SHA256').ComputeHash($keybytes)
[Byte[]] $iv = 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

#or
#$Key = (new-Object Security.Cryptography.PasswordDeriveBytes $Userinput, $Null, "SHA1", 5).GetBytes(32)
#$iv = (new-Object Security.Cryptography.SHA1Managed).ComputeHash( [Text.Encoding]::UTF8.GetBytes('0') )[0..15]

# default is 256 bit
$aes = [System.Security.Cryptography.Aes]::Create()
$encryptor = $aes.CreateEncryptor($Key,$iv)
$Decryptor = $aes.CreateDecryptor($Key,$iv)

$stream = [System.IO.MemoryStream]::new()
$encryptostream = [System.Security.Cryptography.CryptoStream]::new($stream,$encryptor, 'write')
$EncryptedstreamWriter = [System.IO.StreamWriter]::new($encryptostream)
$EncryptedstreamWriter.Write($pwd)
$EncryptedstreamWriter.close()
$encryptostream.close()
[byte[]]$encryptedBytes = $stream.ToArray()
$encrypted = [Convert]::ToBase64String($encryptedBytes)
```

MMS

SYMMETRIC ENCRYPTION - .NET

- **Encoding**:

```
$pwd = 'Password123'
$bytes =  [System.Text.Encoding]::UTF8.GetBytes($pwd)

$Userinput = 'this is not the password but something else!'
$keybytes = [System.Text.Encoding]::UTF8.GetBytes($Userinput)
[Byte[]] $Key = [System.Security.Cryptography.HashAlgorithm]::Create('SHA256').ComputeHash($keybytes)
[Byte[]] $iv = $Key[0..15]
$aes = [System.Security.Cryptography.Aes]::Create()
$encryptor = $aes.CreateEncryptor($Key,$iv)
$Decryptor = $aes.CreateDecryptor($Key,$iv)

$stream = [System.IO.MemoryStream]::new()
$encryptostream = [System.Security.Cryptography.CryptoStream]::new($stream,$encryptor, 'write')
$EncryptedstreamWriter = [System.IO.StreamWriter]::new($encryptostream)
$EncryptedstreamWriter.Write($pwd)
$EncryptedstreamWriter.close()
$encryptostream.close()
[byte[]]$encryptedBytes = $stream.ToArray()
$encrypted = [Convert]::ToBase64String($encryptedBytes)
```

MMS

SYMMETRIC ENCRYPTION - .NET

- **Encoding**:

```
$encryptedBytes = [convert]::FromBase64String('UvYe1wRg0QVoxY8ItywJbw==')
$decryptstream = [System.IO.MemoryStream]::new($encryptedBytes)
$decryptostream = [System.Security.Cryptography.CryptoStream]::new($decryptstream,
$Decryptor, 'read')
$DeStreamReader = [io.streamreader]::new($decryptostream)
$DeStreamReader.ReadToEnd()
$DeStreamReader.close()
$cryptostream.close()
$decryptstream.close()
```

MMS