ATTACK DEFENSE

by PentesterAcademy

Name	WMI: Invoke-WMIMethod
URL	https://attackdefense.com/challengedetails?cid=2077
Type	Services Exploitation: WMI

Important Note: This document illustrates all the important steps required to complete this lab. This is by no means a comprehensive step-by-step solution for this exercise. This is only provided as a reference to various commands needed to complete this exercise and for your further research on this topic. Also, note that the IP addresses and domain names might be different in your lab.

Note: By default, if you are using Windows Server then, the WMI service is already up and running. You need to configure the service in order to access it remotely. In this manual, we are demonstrating how to configure WMI service and making necessary changes for learning purposes.

Step 1: Run powershell.exe to check for wmi service status, if it's running or not.

Command: Get-Service Winmgmt

```
PS C:\Users\Administrator> Get-Service Winmgmt

Status   Name               DisplayName
------   ----               -----------
Running  Winmgmt            Windows Management Instrumentation


PS C:\Users\Administrator>
```

The Windows Management Instrumentation i.e WMI service is running.

We will be using the "**Invoke-WMIMethod**" cmdlet to call WMI methods.

Invoke-WMIMethod:

"The Invoke-WmiMethod cmdlet calls the methods of Windows Management Instrumentation (WMI) objects."

Step 2: Check the help of the "**Invoke-WMIMethod**" cmdlet

Command: help Invoke-WMIMethod

```
NAME
    Invoke-WmiMethod

SYNTAX
    Invoke-WmiMethod [-Class] <string> [-Name] <string> [[-ArgumentList] <Object[]>] [-AsJob] [-Impersonation {Default | Anonymous | Identify |
    Impersonate | Delegate}] [-Authentication {Default | None | Connect | Call | Packet | PacketIntegrity | PacketPrivacy | Unchanged}]
    [-Locale <string>] [-EnableAllPrivileges] [-Authority <string>] [-Credential <pscredential>] [-ThrottleLimit <int>] [-ComputerName
    <string[]>] [-Namespace <string>] [-WhatIf] [-Confirm]  [<CommonParameters>]

    Invoke-WmiMethod [-Name] <string> -InputObject <wmi> [-ArgumentList <Object[]>] [-AsJob] [-ThrottleLimit <int>] [-WhatIf] [-Confirm]
    [<CommonParameters>]

    Invoke-WmiMethod [-Name] <string> -Path <string> [-ArgumentList <Object[]>] [-AsJob] [-Impersonation {Default | Anonymous | Identify |
    Impersonate | Delegate}] [-Authentication {Default | None | Connect | Call | Packet | PacketIntegrity | PacketPrivacy | Unchanged}]
    [-Locale <string>] [-EnableAllPrivileges] [-Authority <string>] [-Credential <pscredential>] [-ThrottleLimit <int>] [-ComputerName
    <string[]>] [-Namespace <string>] [-WhatIf] [-Confirm]  [<CommonParameters>]

    Invoke-WmiMethod [-Name] <string> [-AsJob] [-Impersonation {Default | Anonymous | Identify | Impersonate | Delegate}] [-Authentication
    {Default | None | Connect | Call | Packet | PacketIntegrity | PacketPrivacy | Unchanged}] [-Locale <string>] [-EnableAllPrivileges]
    [-Authority <string>] [-Credential <pscredential>] [-ThrottleLimit <int>] [-ComputerName <string[]>] [-Namespace <string>] [-WhatIf]
    [-Confirm]  [<CommonParameters>]

    Invoke-WmiMethod [-Name] <string> [-AsJob] [-Impersonation {Default | Anonymous | Identify | Impersonate | Delegate}] [-Authentication
    {Default | None | Connect | Call | Packet | PacketIntegrity | PacketPrivacy | Unchanged}] [-Locale <string>] [-EnableAllPrivileges]
    [-Authority <string>] [-Credential <pscredential>] [-ThrottleLimit <int>] [-ComputerName <string[]>] [-Namespace <string>] [-WhatIf]
    [-Confirm]  [<CommonParameters>]

    Invoke-WmiMethod [-Name] <string> [-AsJob] [-Impersonation {Default | Anonymous | Identify | Impersonate | Delegate}] [-Authentication
    {Default | None | Connect | Call | Packet | PacketIntegrity | PacketPrivacy | Unchanged}] [-Locale <string>] [-EnableAllPrivileges]
    [-Authority <string>] [-Credential <pscredential>] [-ThrottleLimit <int>] [-ComputerName <string[]>] [-Namespace <string>] [-WhatIf]
-- More  --
```

```
    Invoke-WmiMethod [-Name] <string> [-AsJob] [-Impersonation {Default | Anonymous | Identify | Impersonate | Delegate}] [-Authentication
    {Default | None | Connect | Call | Packet | PacketIntegrity | PacketPrivacy | Unchanged}] [-Locale <string>] [-EnableAllPrivileges]
    [-Authority <string>] [-Credential <pscredential>] [-ThrottleLimit <int>] [-ComputerName <string[]>] [-Namespace <string>] [-WhatIf]
    [-Confirm]  [<CommonParameters>]


ALIASES
    iwmi


REMARKS
    Get-Help cannot find the Help files for this cmdlet on this computer. It is displaying only partial help.
        -- To download and install Help files for the module that includes this cmdlet, use Update-Help.
        -- To view the Help topic for this cmdlet online, type: "Get-Help Invoke-WmiMethod -Online" or
           go to https://go.microsoft.com/fwlink/?LinkID=113346.
```

We have received all the syntax, aliases, and remarks related information. Also, we are already familiar with namespaces, classes, queries, and methods.

The Invoke-WMIMethod is mostly used for calling WMI methods for the operation. We will invoke the WMI methods without using the Invoke-WMIMethod cmdlet to understand how manually we can call WMI methods.

Step 1: Access the **Win32_Share** class and check all the available methods.

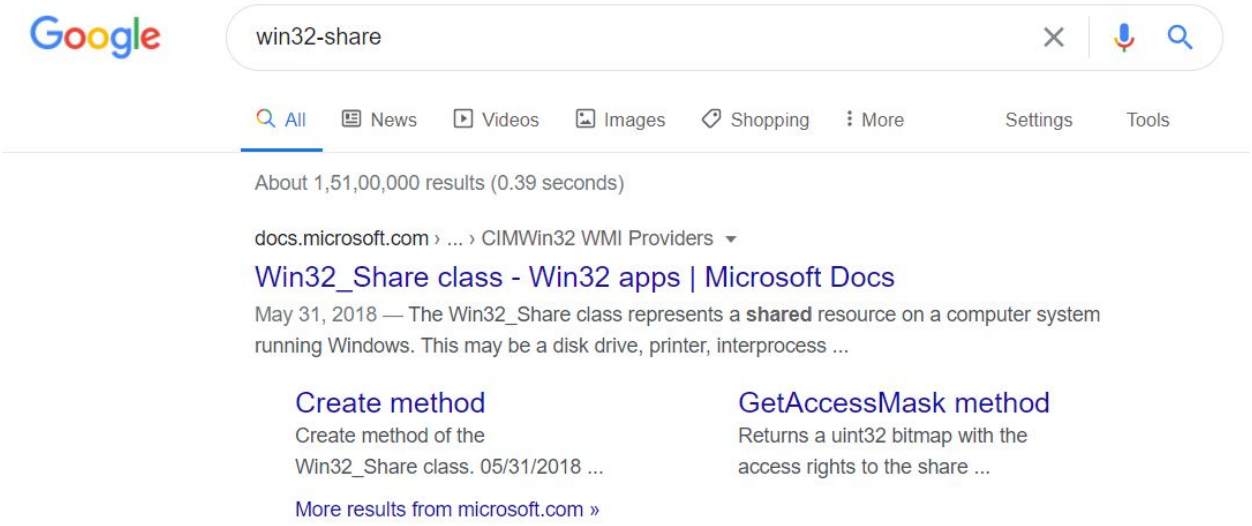
Note: [WMICLASS] - A way for accessing the static properties and methods of a class.

Command: $c = [wmiclass]"win32_share"
$c.methods

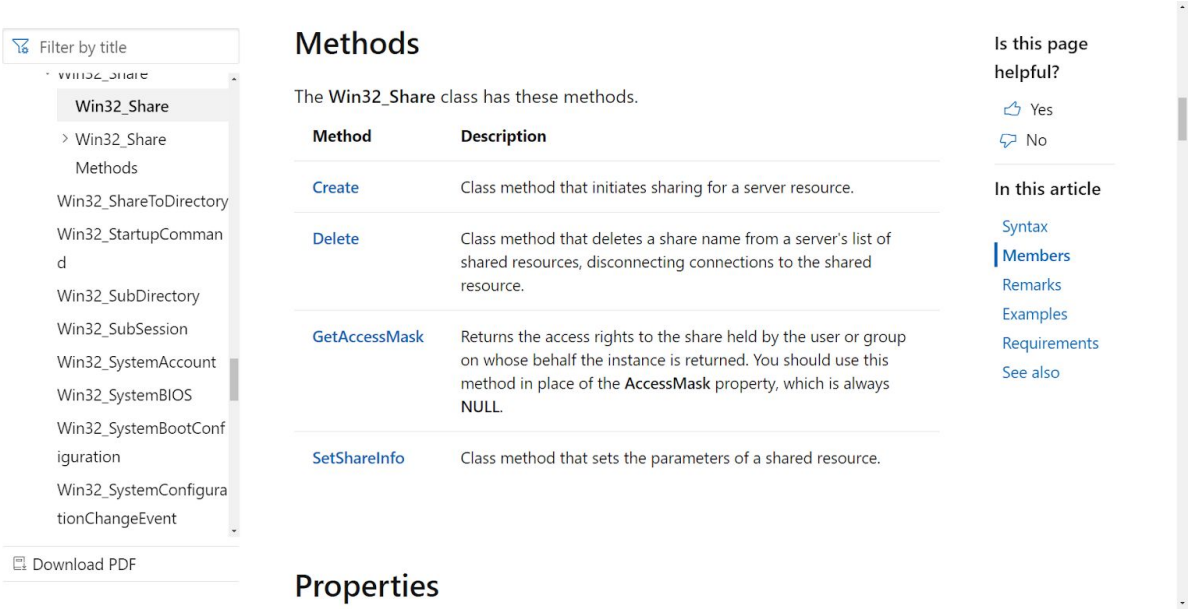
```
PS C:\Users\Administrator> $c = [wmiclass]"win32_share"
PS C:\Users\Administrator> $c.methods


Name           : Create
InParameters   : System.Management.ManagementBaseObject
OutParameters  : System.Management.ManagementBaseObject
Origin         : Win32_Share
Qualifiers     : {Constructor, Implemented, MappingStrings, Static}

Name           : SetShareInfo
InParameters   : System.Management.ManagementBaseObject
OutParameters  : System.Management.ManagementBaseObject
Origin         : Win32_Share
Qualifiers     : {Implemented, MappingStrings}

Name           : GetAccessMask
InParameters   :
OutParameters  : System.Management.ManagementBaseObject
Origin         : Win32_Share
Qualifiers     : {Implemented, MappingStrings}

Name           : Delete
InParameters   :
OutParameters  : System.Management.ManagementBaseObject
Origin         : Win32_Share
Qualifiers     : {Destructor, Implemented, MappingStrings}



PS C:\Users\Administrator>
```

We can notice that there are a total of four methods which we can use.

Step 2: We will Google win32_share and find more information about all the methods.



Link: https://docs.microsoft.com/en-us/windows/win32/cimwin32prov/win32-share



We can observe, all the information is available on the docs.microsoft.com site.

Create Method Link:
https://docs.microsoft.com/en-us/windows/win32/cimwin32prov/create-method-in-class-win32-share

Filter by title

Methods
- Create method
- Delete method
- GetAccessMask method
- SetShareInfo method

Win32_ShareToDirectory
Win32_StartupCommand
Win32_SubDirectory
Win32_SubSession
Win32_SystemAccount
Win32_SystemBIOS
Win32_SystemBootConf

Download PDF

Syntax

Managed Object Format | Copy

```
uint32 Create(
  [in]           string                  Path,
  [in]           string                  Name,
  [in]           uint32                  Type,
  [in, optional] uint32                  MaximumAllowed,
  [in, optional] string                  Description,
  [in, optional] string                  Password,
  [in, optional] Win32_SecurityDescriptor Access
);
```

Parameters

Path [in]

Local path of the Windows share.

Example, "C:\Program Files".

Name [in]

Is this page helpful?
Yes
No

In this article
- Syntax
- Parameters
- Return value
- Remarks
- Examples
- Requirements
- See also

We can notice that there are only three syntaxes (Yellow ones) are mandatory to mention rest are optional. (Purple ones)

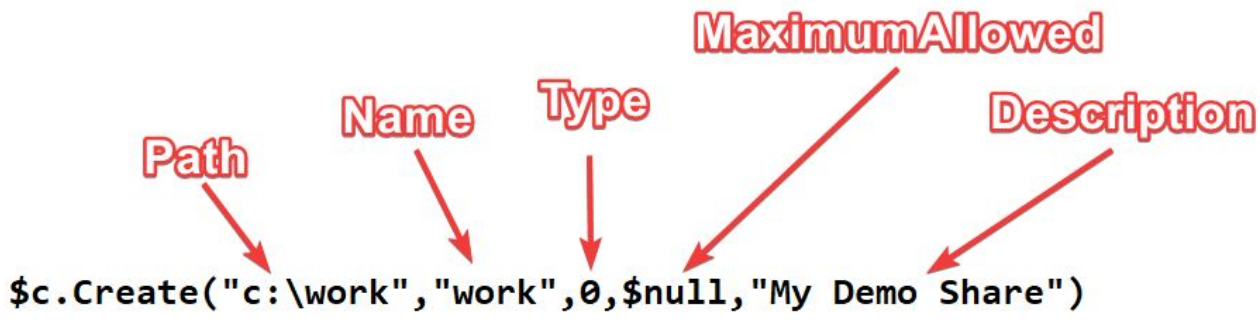
Step 3: We will share a folder using the win32_share class. Create a folder in C:\ drive i.e. **work**.

Command: mkdir C:\work
ls C:\

```
PS C:\Users\Administrator> mkdir C:\work


    Directory: C:\


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-----       10/13/2020   8:05 AM                work


PS C:\Users\Administrator> ls C:\


    Directory: C:\


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-----       11/14/2018   6:56 AM                EFI
d-----        5/13/2020   5:58 PM                PerfLogs
d-r---       11/14/2018   4:10 PM                Program Files
d-----       10/12/2020   9:00 AM                Program Files (x86)
d-r---       10/12/2020   8:03 AM                Users
d-----       10/12/2020   8:01 AM                Windows
d-----       10/13/2020   8:05 AM                work


PS C:\Users\Administrator>
```

Step 4: Sharing the work folder.

Command: $c.Create("c:\work","work",0,$null,"My Demo Share")




```
PS C:\Users\Administrator> $c.Create("c:\work","work",0,$null,"My Demo Share")


__GENUS          : 2
__CLASS          : __PARAMETERS
__SUPERCLASS     :
__DYNASTY        : __PARAMETERS
__RELPATH        :
__PROPERTY_COUNT : 1
__DERIVATION     : {}
__SERVER         :
__NAMESPACE      :
__PATH           :
ReturnValue      : 0
PSComputerName   :



PS C:\Users\Administrator>
```

We have received "**ReturnValue: 0**" which means the command executed successfully and now the "**C:\work**" folder is shared across the network. If we don't receive the return value to 0 then, one can google the return value i.e 22,24, etc to find out the cause of the failure of the command.

We can find all the return values definition on the following link:
https://docs.microsoft.com/en-us/windows/win32/cimwin32prov/create-method-in-class-win32-share

Return value

Returns one of the values listed in the following list, or any other value to indicate an error. For additional error codes, see **WMI Error Constants** or **WbemErrorEnum**. For general **HRESULT** values, see System Error Codes.

Success (0)

Access denied (2)

Unknown failure (8)

Invalid name (9)

Invalid level (10)

Invalid parameter (21)

Duplicate share (22)

Redirected path (23)

Unknown device or directory (24)

Net name not found (25)

Other (26 4294967295)

Step 5: We can verify the share by running the **Get-WMIObject** cmdlet.

Command: Get-wmiobject win32_share

```
PS C:\Users\Administrator> Get-wmiobject win32_share

Name   Path       Description
----   ----       -----------
ADMIN$ C:\Windows Remote Admin
C$     C:\        Default share
IPC$              Remote IPC
work   c:\work    My Demo Share


PS C:\Users\Administrator>
```

We can observe that we have successfully shared the ‘work’ folder.

Step 6: Checking the available methods for the share folder **work**

Commands: $work = Get-wmiobject win32_share -filter "name = 'work'"
$work | get-member -MemberType Method

```
PS C:\Users\Administrator> $work = Get-wmiobject win32_share -filter "name = 'work'"
PS C:\Users\Administrator> $work | get-member -MemberType Method


   TypeName: System.Management.ManagementObject#root\cimv2\Win32_Share

Name          MemberType Definition
----          ---------- ----------
Delete        Method     System.Management.ManagementBaseObject Delete()
GetAccessMask Method     System.Management.ManagementBaseObject GetAccessMask()
SetShareInfo  Method     System.Management.ManagementBaseObject SetShareInfo(System.UInt32


PS C:\Users\Administrator>
```

We can invoke three methods i.e Delete, GetAccessMask and SetShareInfo

Step 7: Delete the shared folder.

Command: $work.Delete()

```
PS C:\Users\Administrator> $work.Delete()


__GENUS          : 2
__CLASS          : __PARAMETERS
__SUPERCLASS     :
__DYNASTY        : __PARAMETERS
__RELPATH        :
__PROPERTY_COUNT : 1
__DERIVATION     : {}
__SERVER         :
__NAMESPACE      :
__PATH           :
ReturnValue      : 0
PSComputerName   :



PS C:\Users\Administrator>
```

We have deleted the shared folder. Verifying it.

Command: Get-wmiobject win32_share

```
PS C:\Users\Administrator> Get-wmiobject win32_share

Name   Path       Description
----   ----       -----------
ADMIN$ C:\Windows Remote Admin
C$     C:\        Default share
IPC$              Remote IPC


PS C:\Users\Administrator>
```

We have successfully removed the ‘work’ shared folder.

So we have successfully shared a folder using the win32_share class method manually and removed it.

This is where we could also use **the Invoke-WMIMethod** cmdlet to share the folder without adding additional steps.

Step 8: We will share a folder using **Invoke-WMIMethod**. Create a folder in C:\ drive i.e **office**.

Command: mkdir C:\office
ls C:\

```
PS C:\Users\Administrator> mkdir C:\office


    Directory: C:\


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-----       10/13/2020   9:46 AM                office


PS C:\Users\Administrator> ls C:\


    Directory: C:\


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
d-----       11/14/2018   6:56 AM                EFI
d-----       10/13/2020   9:46 AM                office
d-----        5/13/2020   5:58 PM                PerfLogs
d-r---       11/14/2018   4:10 PM                Program Files
d-----       10/12/2020   9:00 AM                Program Files (x86)
d-r---       10/12/2020   8:03 AM                Users
d-----       10/12/2020   8:01 AM                Windows
d-----       10/13/2020   9:42 AM                work


PS C:\Users\Administrator>
```

Step 9: Sharing the office folder using **Invoke-WMIMethod**

Command: Invoke-WmiMethod -Class win32_share -Name Create -ArgumentList @($null, "My office Files",$null,"work",$null,"c:\office",0)

```
PS C:\Users\Administrator> Invoke-WmiMethod -Class win32_share -Name Create -ArgumentList @($null
, "My office Files",$null,"work",$null,"c:\office",0)


__GENUS          : 2
__CLASS          : __PARAMETERS
__SUPERCLASS     :
__DYNASTY        : __PARAMETERS
__RELPATH        :
__PROPERTY_COUNT : 1
__DERIVATION     : {}
__SERVER         :
__NAMESPACE      :
__PATH           :
ReturnValue      : 0
PSComputerName   :



PS C:\Users\Administrator>
```

Step 10: We can verify the share by running the Get-WMIObject cmdlet.

Command: Get-wmiobject win32_share

```
PS C:\Users\Administrator> Get-wmiobject win32_share

Name   Path       Description
----   ----       -----------
ADMIN$ C:\Windows Remote Admin
C$     C:\        Default share
IPC$              Remote IPC
work   c:\office  My office Files


PS C:\Users\Administrator>
```

The office folder is shared successfully.

Step 11: Delete the shared folder and verify it.

Commands:

(Get-WmiObject -Class Win32_Share -ComputerName . -Filter "Name='work'").InvokeMethod("Delete",$null)

Get-WmiObject -Class Win32_Share

```
PS C:\Users\Administrator> (Get-WmiObject -Class Win32_Share -Filter "Name='work'").InvokeMethod("Delete",$null)
0
PS C:\Users\Administrator> Get-WmiObject -Class Win32_Share

Name   Path       Description
----   ----       -----------
ADMIN$ C:\Windows Remote Admin
C$     C:\        Default share
IPC$              Remote IPC


PS C:\Users\Administrator>
```

We have successfully shared and deleted the folder i.e office. Using Invoke-WMIMethod cmdlet.

References:

- Invoke-WMIMethod (https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.management/invoke-wmimethod?view=powershell-5.1)