Offensive WMI - The Basics (Part 1)

0xinfection.github.io/posts/wmi-basics-part-1

August 29, 2021

2021-08-29

This blog post is the first of a many part series on WMI and is intended for fairly new audiences. A basic understanding of Powershell will definitely help the reader while going through the blog, however, it is not a requirement. That's it, let us jump into the real stuff.

Introduction

Why WMI?

WMI is a set of specifications from Microsoft that was designed for fast and efficient administration when it comes to Windows systems. And as you might know, a rule of security says "that anything useful for administration is also great at being abused by evil-doers". WMI can *really* do a lot of things – from gathering statuses of computers and configuring settings to running applications and executing code. Moreover, WMI is present on all available Windows OS versions, so targeted surface is quite broad here.

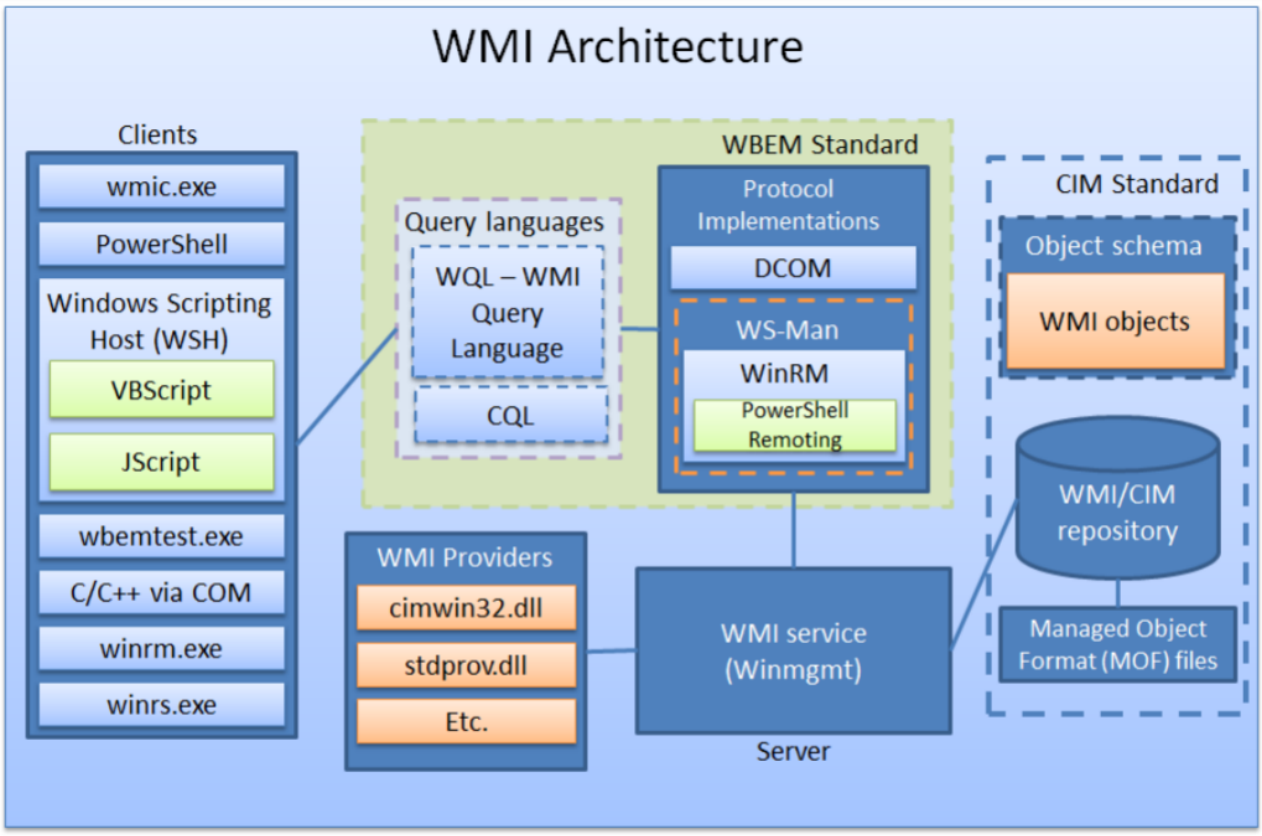
What is WMI?

Let's quickly go over a few important terms. WMI stands for **Windows Management Instrumentation**, which is Microsoft's implementation of the CIM (Common Information Model) & WBEM (Web-Based Enterprise Management) – both of which are *DMTF*

(Distributed Management Task Force) standards in general. WMI gives us a neat and uniform interface for applications/scripts to manage a computer (maybe remote and local) – incl. processes, services, etc.

The WMI Architecture

Understanding the architecture is very essential to understanding how the entire WMI ecosystem works. A broad overview of the architecture of WMI is depicted below (sourced from Graeber's talk from BHUSA 15):



Let's break down the major components one by one:

- **Clients/Consumers**: These are essentially the end consumers which interact with WMI classes for querying data, running methods, etc. Few prominent clients include `wmic.exe`, `wbemtest.exe`, `winrm.exe`, VBScript/JScript and ofc Powershell cmdlets.
- **Query Languages**: Just like SQL provides you with a way to query a database, WMI too has WQL (WMI Query Language) / CQL for querying the WMI service. When it comes to managing remote boxes, the WBEM standard kicks in – which include DCOM and WS-Man (don't worry if you don't understand these terms, read on). WQL is basically SQL syntax for WMI, so is not case-sensitive.

  A simple query may look like this:

  ```
  select * from win32_bios
  ```

  which gives us information about our BIOS.

- **Repositories**: These are the databases that we talked about previously that stores all static data (definitions) of classes. The repositories are defined by MOF (managed object format) files which define the structure, classes, namespaces, etc. The database files can be found under the `%WINDIR%\System32\Wbem\Repository` directory.

```
PS C:\Users\pew> dir C:\Windows\System32\Wbem\Repository\


    Directory: C:\Windows\System32\Wbem\Repository


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         8/24/2021   8:11 AM        4866048 INDEX.BTR
-a----         8/24/2021   8:06 AM          77060 MAPPING1.MAP
-a----         8/24/2021   8:11 AM          77100 MAPPING2.MAP
-a----         8/24/2021   7:56 AM          77060 MAPPING3.MAP
-a----         8/24/2021   8:11 AM       22945792 OBJECTS.DATA
```

- **MOF Files**: MOF files are basically used to define WMI namespaces, classes, providers, etc. You'll usually find them under the `%WINDIR%\System32\Wbem` directory with the extension `.mof`. In a later part of the series, we'll take a look at how we can write our own MOF files to extend the feature-set of WMI.

- **Providers**: Whatever is defined in the repositories can be accessed with the help of WMI providers. They are usually DLL files and associated with a MOF file – `cimwin32.dll`, `stdprov.dll`, etc to name a few, however, they can take the form of other types as well (Class, Event, Event Consumer, Method, etc). The providers are essential to the ecosystem because they monitor events and data from specific defined objects. Think of providers like drivers which provide a bridge between managed objects and WMI.

  In the screenshot below, the DLL files are the providers of the associated MOF files:

```
PS C:\Users\pew> dir C:\windows\System32\wbem


    Directory: C:\windows\System32\wbem


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----         8/24/2021   8:20 PM                AutoRecover
d-----         12/7/2019   1:52 AM                en
d-----         12/7/2019   1:52 AM                en-US
d-----         12/7/2019   1:14 AM                Logs
d-----         8/24/2021   8:21 PM                MOF
d-----         8/24/2021   7:56 AM                Performance
d-----         8/24/2021   7:51 AM                Repository
d-----         12/7/2019   1:14 AM                tmf
d-----         12/7/2019   1:14 AM                xml
-a----         12/7/2019   1:08 AM           2852 aeinv.mof
-a----         12/7/2019   1:51 AM          17510 AgentWmi.mof
-a----         12/7/2019   1:10 AM            693 AgentWmiUninstall.mof
-a----         12/7/2019   1:08 AM          40448 appbackgroundtask.dll
-a----         12/7/2019   1:08 AM           2902 appbackgroundtask.mof
-a----         12/7/2019   1:08 AM            852 appbackgroundtask_uninstall.mof
-a----         12/7/2019   1:09 AM           1724 AuditRsop.mof
-a----         12/7/2019   1:09 AM           1092 authfwcfg.mof
-a----         12/7/2019   1:08 AM          12120 bcd.mof
-a----         12/7/2019   1:10 AM           2626 BthMtpEnum.mof
-a----         12/7/2019   1:09 AM         161166 cimdmtf.mof
-a----          4/9/2021   6:50 AM        2058752 cimwin32.dll
-a----         12/7/2019   1:08 AM        2712842 cimwin32.mof
-a----         12/7/2019   1:10 AM           2088 CIWmi.mof
-a----         12/7/2019   1:08 AM           4966 classlog.mof
```

- **Managed Objects**: These are aliases of the resources in context, i.e. a managed object can be the service, process or OS being managed by WMI.

- **Namespaces**: Put simply, namespaces are logical divisions of classes meant for easy discovery and usage. They are divided into 3 groups:
    - system
    - core
    - extension

  and 3 types:
    - abstract
    - static
    - dynamic

  Few prominent namespaces that come by default are: `root\cimv2`, `root\default`, `root\security`, `root\subscription`, etc.

That's it with the architecture. Now let's learn a bit about using WMI with Powershell.

Using WMI with Powershell

Now that we're done with the theory part, let's quickly spawn a PS terminal. It is important to remember that up to v2 of Powershell, there are only a few cmdlets to interact with WMI. We'll quickly check our Powershell version and change the version to 2:

```
PS C:\Users\pew> $PSVersionTable.PSVersion

Major  Minor  Build  Revision
-----  -----  -----  --------
5      1      19041  906


PS C:\Users\pew> powershell -version 2
Windows PowerShell
Copyright (C) 2009 Microsoft Corporation. All rights reserved.

PS C:\Users\pew> $PSVersionTable.PSVersion

Major  Minor  Build  Revision
-----  -----  -----  --------
2      0      -1     -1
```

Now, lets run `Get-Command -CommandType Cmdlet *wmi*` in a Powershell prompt. This leads us to:

```
PS C:\Users\pew> Get-Command -CommandType Cmdlet *wmi*

CommandType     Name                                               Version    Source
-----------     ----                                               -------    ------
Cmdlet          Get-WmiObject                                      3.1.0.0    Microsoft.PowerShell...
Cmdlet          Invoke-WmiMethod                                   3.1.0.0    Microsoft.PowerShell...
Cmdlet          Register-WmiEvent                                  3.1.0.0    Microsoft.PowerShell...
Cmdlet          Remove-WmiObject                                   3.1.0.0    Microsoft.PowerShell...
Cmdlet          Set-WmiInstance                                    3.1.0.0    Microsoft.PowerShell...
```

> **TIP**: The names of the commands are pretty self-explanatory (and we'll dig into more later as well). At any point in time, you can use Powershell's standard syntax: `help <command>` to get more info on what the specific command does. e.g. you might want to try `help Invoke-WmiMethod` to view what the command does – very similar to Linux manpages.

From Powershell v3 onwards, MS introduced CIM cmdlets which make use of WS-MAN and CIM standards to manage objects. Having access to CIM cmdlets has advantages in 2 contexts:

- In machines where WMI/DCOM itself is blocked from running (maybe due to a host-based firewall rule?) but WinRM/WS-MAN (Windows Remote Management) is enabled, we can still use CIM to do exactly what we can do with WMI.
- CIM itself is an industry-standard and is implemented cross-platform, which means it can be used to work with non-Windows boxes as well.

> Phew! That's a mouthful of words. Let's understand what the new terms here mean:
>
> **DCOM**: An alias for Distributed Component Object Model, DCOM is a proprietary Microsoft protocol for communication between software components on networked computers. WMI uses Distributed COM (DCOM) to connect to a remote machine. However, DCOM isn't that firewall-friendly.
>
> **WS-MAN**: WS-MAN or WS-Management is a DMTF standard that provides a common way for systems to access management information across the IT infrastructure. WS-MAN on the other hand uses HTTP, so is definitely firewall-friendly.

We'll redo what we did above, but after changing the Powershell version back to default (in my case I've Powershell v5):

```
PS C:\Users\pew> Get-Command -type Cmdlet *cim*

CommandType     Name                                               Version    Source
-----------     ----                                               -------    ------
Cmdlet          Get-CimAssociatedInstance                          1.0.0.0    CimCmdlets
Cmdlet          Get-CimClass                                       1.0.0.0    CimCmdlets
Cmdlet          Get-CimInstance                                    1.0.0.0    CimCmdlets
Cmdlet          Get-CimSession                                     1.0.0.0    CimCmdlets
Cmdlet          Invoke-CimMethod                                   1.0.0.0    CimCmdlets
Cmdlet          New-CimInstance                                    1.0.0.0    CimCmdlets
Cmdlet          New-CimSession                                     1.0.0.0    CimCmdlets
Cmdlet          New-CimSessionOption                               1.0.0.0    CimCmdlets
Cmdlet          Register-CimIndicationEvent                        1.0.0.0    CimCmdlets
Cmdlet          Remove-CimInstance                                 1.0.0.0    CimCmdlets
Cmdlet          Remove-CimSession                                  1.0.0.0    CimCmdlets
Cmdlet          Set-CimInstance                                    1.0.0.0    CimCmdlets
```

Repeating what we said above, CIM cmdlets can do *everything* that WMI cmdlets can. If we want to map the functionalities between both WMI cmdlets and CIM cmdlets, here's a tabular representation of the functionality comparison between both types:

Use \ Types	WMI Cmdlets	CIM Cmdlets
Get information about classes	`Get-WmiObject`	`Get-CimInstance`
Invoking a method	`Invoke-WmiMethod`	`Invoke-CimMethod`
Subscribing to an event	`Register-WmiEvent`	`Register-CimIndicationEvent`
Creating/updating instances of a class	`Set-WmiInstance`	`Set-CimInstance`
Deleting instances of a class	`Remove-WmiObject`	`Remove-CimInstance`

Running WMI queries with Powershell

Now that we know about the different cmdlets available for us to use, we can try running the sample WQL query above. We already know that `Get-WmiObject` can be used to get info about classes. So let's run the cmdlet with the `-Query` parameter:

```
Get-WmiObject -Query 'select * from win32_bios'
```

```
PS C:\Users\pew> Get-WmiObject -Query 'select * from win32_bios'


SMBIOSBIOSVersion : VirtualBox
Manufacturer      : innotek GmbH
Name              : Default System BIOS
SerialNumber      : 0
Version           : VBOX   - 1
```

Conclusion

This blog post was meant for giving an overview of what we'll be dealing with in the next parts of the series. There are a lot of technical buzzwords here but understanding them is necessary. I hope you enjoyed reading through so far, and I'm looking forward to our journey together into exploring WMI.

Adios amigo!