Understanding the Basics

jsecurity101.medium.com/wmi-internals-part-1-41bb97e7f5eb

July 5, 2022

Jonathan Johnson
Jul 5

.

8 min read

.

WMI Internals Part 1

Recently I have taken up an interest in WMI internals and thought I would write a blog series on some of my findings. This first release will cover the fundamentals of WMI and how to track back WMI activity to the WMI provider host process (WmiPrvse.exe), the executable responsible for executing WMI activity. This post is meant to give the information needed to understand part 2 of this series, which will cover the relationship between WMI and COM. That being said, this post will not cover everything WMI related — like permanent WMI event subscriptions, for example.

A lot of this information isn't new, so I would like to give credit early and direct everyone to the section below. As those write-ups/conversations helped my understanding of this technology tremendously.

WMI Vocabulary

Microsoft wanted to have their own technology that allowed them to gather information and manage assets across the enterprise, to accomplish this they implemented their own version of Web-Based Enterprise Management which they called Windows Management Instrumentation (WMI). WMI allows users and administrators to obtain information about objects, which in turn give information about things like the environment, computer, processes, etc. WMI also allows administrators to create their own objects, i.e. create a process, services, etc. In order to be successful at this, WMI uses the Common Information Model (CIM), which is a standard to represent various objects like the ones mentioned above. These objects are considered "managed objects".

WMI has 4 main components:

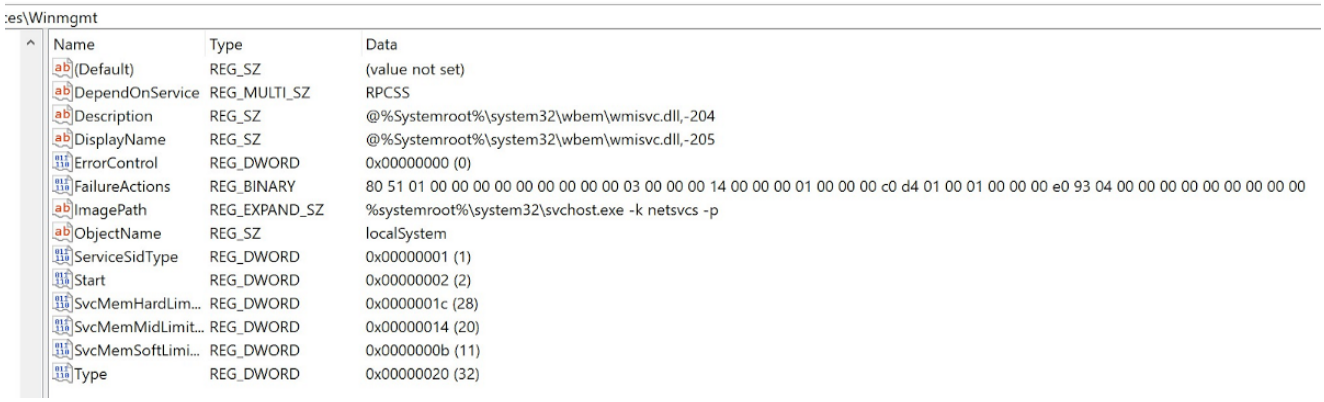
- COM servers that monitor managed objects. consist of a DLL (COM server) and a (MOF) file which serves as a definition for a WMI class. These providers are typically DLLs and can be found in `C:\Windows\System32\wbem\*`
- WMI class that represents objects like — processes, services, operating system, etc.
- This is the WMI service (Winmgmt). This service holds two components:

1. The CIM Object Manager (CIMOM). This component handles the connection between management applications and providers. This is considered the WMI Core.
2. The on-disk database "store" is known as the WMI/CIMOM Object Repository. The repository is organized by WMI namespaces. These namespaces look like root\cim2 and hold a collection of providers. The repository can be found at: C:\Windows\System32\wbem\Repository\

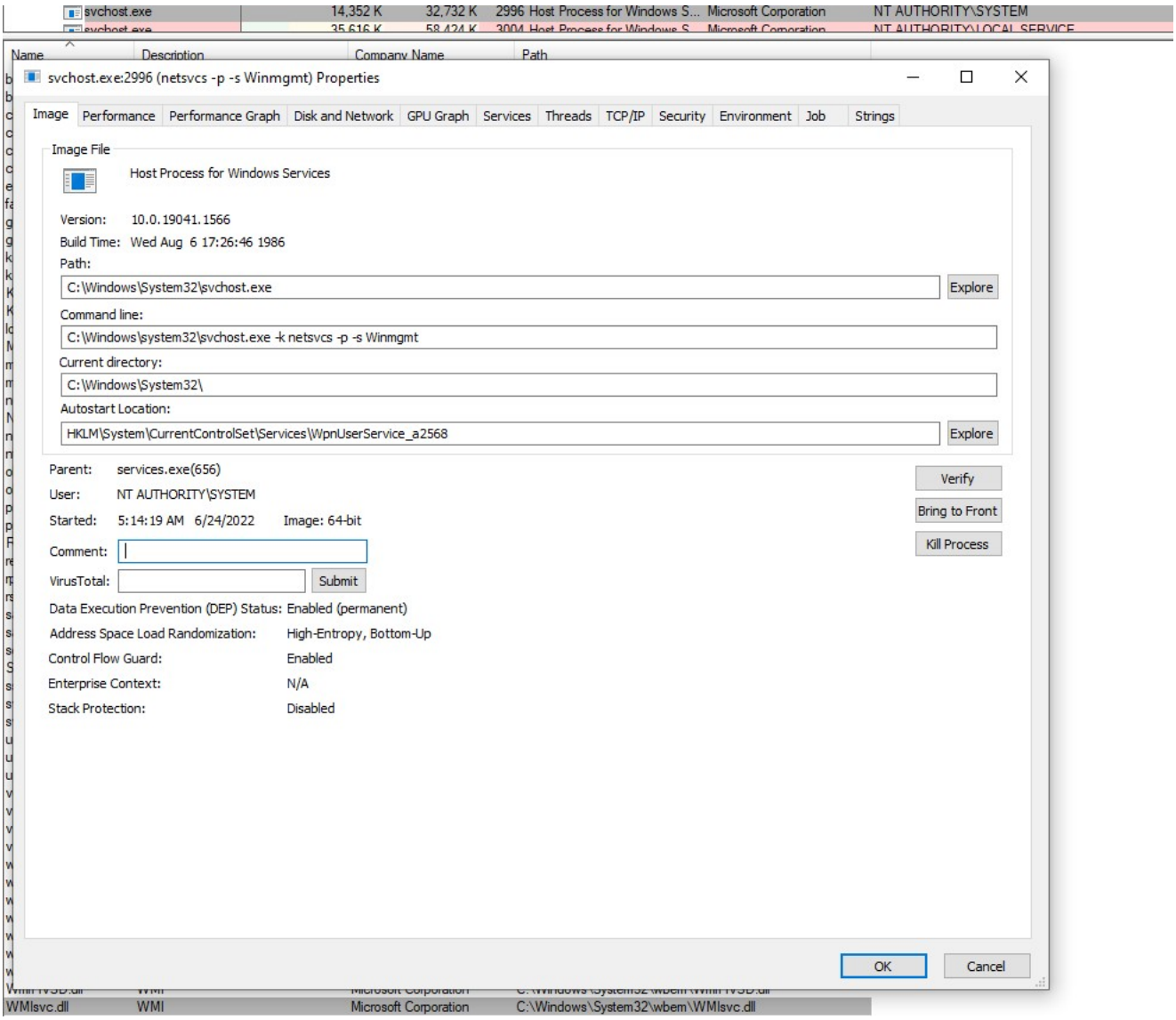
   The client application that interacts with the WMI infrastructure. This can be a regular binary (EXE), a VBScript, a PowerShell script, etc. We will see an example of this within the walkthrough.

Before moving on I would like to go back to the WMI service (Winmgmt) and speak as to how it is implemented and how tasks are carried out.

The WMI service (WinMgmt) is stored within **wmisvc**.dll which is loaded and runs inside of **svchost**.exe. We can see this if we look at WinMgmt configuration within the registry:



As well as confirm this within Process Explorer:



You might have seen another WMI binary on disk called WmiPrvSe (WMI Provider Host). This binary is used to load the correct COM servers (WMI providers) so that it may execute the task it was instructed to. This binary is launched via **C:\Windows\system32\wbem\wmiprvse.exe -secured -Embedding**, where its parent is a svchost process with the CommandLine of: **C:\Windows\system32\svchost.exe -k DcomLaunch -p**. This svchost is launched under services.exe.

An example of how a WMI call is made at a high level:

- WMI service (wmisvc.dll) is launched within the SVCHOST process via ()
- Management application (powershell.exe) executes WMI method
- WmiPrvSe is launched via , under the DCOMLaunch svchost process
- The WMI services loads the appropriate WMI provider into WmiPrvSe
- WmiPrvSe executes the function expressed by the method

There is a lot more that happens underneath the hood of WMI that include COM/RPC. Please see the Windows Internals book Part 2, specifically Chapter 10 for more information on this.

WMI Walkthrough

For me, WMI made a lot more sense after playing with the various cmdlets exposed through Windows. Let's do that.

First we need to identify which WMI class/method we want to interact with. Luckily there are two different WMI cmdlet types exposed to us via PowerShell. The WMI cmdlets and the CIM cmdlets. The CIM cmdlets are the "newer" and more preferred way of interacting with WMI, but the WMI cmdlets still hold their place, which we will see later.

I want to see if there is a WMI class that allows me to create a process. To do that I am going to see if there are any classes that expose a method that contains Create in it, to do so I run the following:

```
PS > Get-CimClass -MethodName *Create*

   NameSpace: ROOT/cimv2

CimClassName                         CimClassMethods      CimClassProperties----------
--                        ---------------      ------------------Win32_Process
{Create, Terminat... {Caption, Description, InstallDate, Name...}Win32_BaseService
{StartService, St... {Caption, Description, InstallDate, Name...}Win32_Service
{StartService, St... {Caption, Description, InstallDate, Name...}…
```

Here we can see that there is a WMI class called that holds a method called . This classes provider lives within the namespace. However; we currently don't know what the WMI provider is, so let's find that out next.

WMI providers, as mentioned above, are essentially just COM servers. Which means that they are stored in the registry behind a class identifier (CLSID). By obtaining a provider instance and filtering on the WMI class we are curious about, we may pull that CLSID out.

```
{d63a5850-8f16-11cf-9f47-00aa00bf345c}
```

We can then search for that CLSID within the HCKR hive in the registry:

```
C:\WINDOWS\system32\wbem\cimwin32.dll
```

Great, now we have a lot of great information about the WMI class and method we want to invoke:

WMI Class: Win32Process
Method: Create
Provider: cimwin32.dll
Namespace: ROOT/cimv2

Lastly, I need to see the parameters I need to pass through in order to successfully create the process. There are a couple of ways to achieve this but let's first see if we can leverage the WMI cmdlets to give us this information.

```
Name                        CimType Qualifiers
ReferenceClassName----                          ------- ----------
-----------------CommandLine                    String {ID, In,
MappingStrings}CurrentDirectory             String {ID, In,
MappingStrings}ProcessStartupInformation Instance {EmbeddedInstance, ID, In,
MappingStrings}ProcessId                    UInt32 {ID, MappingStrings, Out}
```

Great, here I can see that there are 3 "In" parameters (**CommandLine**, **CurrentDirectory**, **ProcessStartupInformation**) and 1 "Out" parameter (**ProcessId**). To get more information about the parameters I need to pass to a method I typically open up the provider's MOF file. In this case `C:\Windows\System32\wbem\cimwin32.mof` .

After getting to the point of where the Win32_Process class is defined we see a lot of great information:

```
[Dynamic,Provider("CIMWin32") :
ToInstance,SupportsCreate,CreateBy("Create"),SupportsDelete,DeleteBy("DeleteInstance")
 : ToInstance,UUID("{8502C4DC-5FBB-11D2-AAC1-006008C78BC7}") : ToInstance]class
Win32_Process : CIM_Process{[Read : ToSubclass,Privileges{"SeDebugPrivilege"} :
ToSubclass,MappingStrings{"Win32API|Tool Help Structures|MODULEENTRY32|szExePath"} :
ToSubclass] string ExecutablePath;[Read : ToSubclass,Privileges{"SeDebugPrivilege"} :
ToSubclass,MappingStrings{"Win32|WINNT.H|QUOTA_LIMITS|MaximumWorkingSetSize"} :
ToSubclass] uint32 MaximumWorkingSetSize;[Read :
ToSubclass,Privileges{"SeDebugPrivilege"} :
ToSubclass,MappingStrings{"Win32|WINNT.H|QUOTA_LIMITS|MinimumWorkingSetSize"} :
ToSubclass] uint32 MinimumWorkingSetSize;...
```

Firstly, there are a lot of read instructions, which showcases that we can probably use this same class to get a process object by instantiating the Win32_Process class. We will do this later, what we care about now however is the Create method. We see there is a "Constructor" qualifier which means that there is a call that will create an instance of this class. Looking at the information for the Constructor method, we see it refers to **Create**.

```
[Constructor,Static,Implemented,Privileges{"SeAssignPrimaryTokenPrivilege",
"SeIncreaseQuotaPrivilege", "SeRestorePrivilege"} : ToSubclass,ValueMap{"0", "2",
"3", "8", "9", "21", ".."} : ToSubclass,MappingStrings{"Win32API|Process and Thread
Functions|CreateProcess"} : ToSubclass] uint32 Create([In :
ToSubclass,MappingStrings{"Win32API|Process and Thread Functions|lpCommandLine "} :
ToSubclass] string CommandLine,[In : ToSubclass,MappingStrings{"Win32API|Process and
Thread Functions|CreateProcess|lpCurrentDirectory "} : ToSubclass] string
CurrentDirectory,[In : ToSubclass,MappingStrings{"WMI|Win32_ProcessStartup"} :
ToSubclass] Win32_ProcessStartup ProcessStartupInformation,[Out :
ToSubclass,MappingStrings{"Win32API|Process and Thread
Functions|CreateProcess|lpProcessInformation|dwProcessId"} : ToSubclass] uint32
ProcessId);
```

This definition has the same information that the WMI cmdlet holds, except it says that one parameter (**ProcessStartupInformation**) is passed in via a **Win32_ProcessStartup** instance. Taking a look at this class, I can see that I can create my own instance of this class, specify a wide range of **ProcessStartup** options. One that stood out was the parameter.

```
[Abstract,Locale(1033) : ToInstance,UUID("{8502C4DB-5FBB-11D2-AAC1-006008C78BC7}") :
ToInstance]class Win32_ProcessStartup : Win32_MethodParameterClass

{

[Write : ToSubclass,MappingStrings{"Win32API|Process and Thread
Structures|STARTUPINFO|wShowWindow"} : ToSubclass] uint16 ShowWindow;...class
Win32_ProcessStartup : Win32_MethodParameterClass{[Description("The ShowWindow
property specifies how the window is to be displayed to the user.") : Amended
ToSubclass,Values{"SW_HIDE", "SW_NORMAL", "SW_SHOWMINIMIZED", "SW_SHOWMAXIMIZED",
"SW_SHOWNOACTIVATE", "SW_SHOW", "SW_MINIMIZE", "SW_SHOWMINNOACTIVE", "SW_SHOWNA",
"SW_RESTORE", "SW_SHOWDEFAULT", "SW_FORCEMINIMIZE"} : Amended ToSubclass] uint16
ShowWindow;
```

There is an option to specify **Sw_Hidden** (or value 0). Let's do that because I figured that starting a hidden notepad process is what any regular hacker would do.

First let's create a Win32_ProcessStartup instance with the hidden parameter and pass it in.

Lastly, let's invoke the Create method:

```
ProcessId ReturnValue PSComputerName— — — — — — — — — — — — — — — — — —2432 0
```

After invoking this, we can see that this process was spawned under WmiPrvse.exe:

WmiPrvSE.exe	3,928 K	12,072 K	10128 WMI Provider Host	Microsoft Corporation	NT AUTHORITY\NETWORK SERVICE
notepad.exe	2,312 K	13,180 K	15444 Notepad	Microsoft Corporation	DESKTOP-02SN8AH\TestUser

If we look at the WmiPrvse.exe binary, we see that the Win32_Process provider DLL — cimwin32.dll was loaded:

advapi32.dll	Advanced Windows 32 Base API	Microsoft Corporation	C:\Windows\System32\advapi32.dll
bcryptprimitives.dll	Windows Cryptographic Primitives ...	Microsoft Corporation	C:\Windows\System32\bcryptprimitives.dll
cimwin32.dll	WMI Win32 Provider	Microsoft Corporation	C:\Windows\System32\wbem\cimwin32.dll
clbcatq.dll	COM+ Configuration Catalog	Microsoft Corporation	C:\Windows\System32\clbcatq.dll
combase.dll	Microsoft COM for Windows	Microsoft Corporation	C:\Windows\System32\combase.dll
fastprox.dll	WMI Custom Marshaller	Microsoft Corporation	C:\Windows\System32\wbem\fastprox.dll
framedynos.dll	WMI SDK Provider Framework	Microsoft Corporation	C:\Windows\System32\framedynos.dll
gdi32.dll	GDI Client DLL	Microsoft Corporation	C:\Windows\System32\gdi32.dll
gdi32full.dll	GDI Client DLL	Microsoft Corporation	C:\Windows\System32\gdi32full.dll
kernel.appcore.dll	AppModel API Host	Microsoft Corporation	C:\Windows\System32\kernel.appcore.dll
kernel32.dll	Windows NT BASE API Client DLL	Microsoft Corporation	C:\Windows\System32\kernel32.dll
KernelBase.dll	Windows NT BASE API Client DLL	Microsoft Corporation	C:\Windows\System32\KernelBase.dll
locale.nls			C:\Windows\System32\locale.nls
msvcp_win.dll	Microsoft® C Runtime Library	Microsoft Corporation	C:\Windows\System32\msvcp_win.dll
msvcrt.dll	Windows NT CRT DLL	Microsoft Corporation	C:\Windows\System32\msvcrt.dll
ncobjapi.dll		Microsoft Corporation	C:\Windows\System32\ncobjapi.dll
ntdll.dll	NT Layer DLL	Microsoft Corporation	C:\Windows\System32\ntdll.dll
oleaut32.dll	OLEAUT32.DLL	Microsoft Corporation	C:\Windows\System32\oleaut32.dll
powrprof.dll	Power Profile Helper DLL	Microsoft Corporation	C:\Windows\System32\powrprof.dll
profapi.dll	User Profile Basic API	Microsoft Corporation	C:\Windows\System32\profapi.dll
R000000000006.clb			C:\Windows\Registration\R000000000006.clb
rpcrt4.dll	Remote Procedure Call Runtime	Microsoft Corporation	C:\Windows\System32\rpcrt4.dll
sechost.dll	Host for SCM/SDDL/LSA Lookup ...	Microsoft Corporation	C:\Windows\System32\sechost.dll
SortDefault.nls			C:\Windows\Globalization\Sorting\SortDefault.nls
sspicli.dll	Security Support Provider Interface	Microsoft Corporation	C:\Windows\System32\sspicli.dll
ucrtbase.dll	Microsoft® C Runtime Library	Microsoft Corporation	C:\Windows\System32\ucrtbase.dll
umpdc.dll			C:\Windows\System32\umpdc.dll
user32.dll	Multi-User Windows USER API Cli...	Microsoft Corporation	C:\Windows\System32\user32.dll
user32.dll.mui	Multi-User Windows USER API Cli...	Microsoft Corporation	C:\Windows\System32\en-US\user32.dll.mui
userenv.dll	Userenv	Microsoft Corporation	C:\Windows\System32\userenv.dll
wbemcomn.dll	WMI	Microsoft Corporation	C:\Windows\System32\wbemcomn.dll
wbemprox.dll	WMI	Microsoft Corporation	C:\Windows\System32\wbem\wbemprox.dll
wbemsvc.dll	WMI	Microsoft Corporation	C:\Windows\System32\wbem\wbemsvc.dll
win32u.dll	Win32u	Microsoft Corporation	C:\Windows\System32\win32u.dll
WmiPrvSE.exe	WMI Provider Host	Microsoft Corporation	C:\Windows\System32\wbem\WmiPrvSE.exe
wmiutils.dll	WMI	Microsoft Corporation	C:\Windows\System32\wbem\wmiutils.dll
ws2_32.dll	Windows Socket 2.0 32-Bit DLL	Microsoft Corporation	C:\Windows\System32\ws2_32.dll

Before we close out, remember earlier when we saw that we could get a WMI instance of a process via Win32_Process as well? Let's see if we can do that to get information about our newly created notepad process:

```
ProcessId Name HandleCount WorkingSetSize VirtualSize — — — — — — — — — — — — — — — —
— — — — — — — — — -15444 notepad.exe 190 13496320 2203470827520
```

You can also achieve this via `Get-WMIObject` :

```
PS C:\Users\TestUser\Desktop\AtomicTestHarnesses> Get-WmiObject -Class win32_process -Filter "ProcessId = 15444"


__GENUS                   : 2
__CLASS                   : Win32_Process
__SUPERCLASS              : CIM_Process
__DYNASTY                 : CIM_ManagedSystemElement
__RELPATH                 : Win32_Process.Handle="15444"
__PROPERTY_COUNT          : 45
__DERIVATION              : {CIM_Process, CIM_LogicalElement, CIM_ManagedSystemElement}
__SERVER                  : DESKTOP-02SN8AH
__NAMESPACE               : root\cimv2
__PATH                    : \\DESKTOP-02SN8AH\root\cimv2:Win32_Process.Handle="15444"
Caption                   : notepad.exe
CommandLine               : notepad.exe
CreationClassName         : Win32_Process
CreationDate              : 20220629072849.018453-420
CSCreationClassName       : Win32_ComputerSystem
CSName                    : DESKTOP-02SN8AH
Description               : notepad.exe
ExecutablePath            : C:\Windows\system32\notepad.exe
ExecutionState            :
Handle                    : 15444
HandleCount               : 190
InstallDate               :
KernelModeTime            : 468750
MaximumWorkingSetSize     : 1380
MinimumWorkingSetSize     : 200
Name                      : notepad.exe
OSCreationClassName       : Win32_OperatingSystem
OSName                    : Microsoft Windows 10 Pro|C:\Windows|\Device\Harddisk0\Partition2
OtherOperationCount       : 95
OtherTransferCount        : 2372
PageFaults                : 3504
PageFileUsage             : 2400
ParentProcessId           : 10128
PeakPageFileUsage         : 2824
PeakVirtualSize           : 2203473547264
PeakWorkingSetSize        : 13320
Priority                  : 8
PrivatePageCount          : 2457600
```

Conclusion:

During this post I wanted to set a baseline of knowledge that will carry on to other posts in this series “WMI Internals”. I find this important so that everyone has the same vocabulary and basic understanding of how things work. What I showed today wasn’t anything new but will be showcased in less basic examples in the following posts. There were some things purposefully left out for WMI, but I urge everyone to go to the resource section and check out the work of some phenomenal researchers. Thanks for tuning in, part 2 will dive deeper into WMI and COM relationships.

Resources: