Catching WMI lateral movement in an enterprise network

Jaco Blokker

Credits: Cat and mouse chase by Jeroen Moes (CC BY-SA 2.0)
http://creativecommons.org/licenses/by-sa/2.0/ and
https://www.flickr.com/photos/jeroenmoes/4265223393
About Jaco Blokker

15y infosec, 6y Senior member Blue team KPN

Event analysis

Develop, tune (network) detection, Snort
Network detection IDS ruleset:

github.com/KPN-CISO/Network-Detection

Why sharing it?
Agenda

- Method, approach
- Findings
- What worked and what not

…Starting point for future enhancements
About WMI

Dmtf standards
- WBEM: Web-Based Enterprise Management
- CIM: Common Information Model

Microsoft’s implementation:
Windows Management Instrumentation
WMI characteristics

Core OS component
- Read, manipulate, execute

Access defaults
- local administrator (allowed)
- hostbased firewall (blocked)

Transports
- WS-man (Winrm)
- RPC/DCOM [this research]
Different perspectives

System administrator
Attacker
Defender

How to detect + distinguish legit / non-legit?

Just theory?
<table>
<thead>
<tr>
<th>Technique</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APT32</td>
<td>APT32 used WMI to deploy their tools on remote machines and to gather information about the Outlook process. [65]</td>
</tr>
<tr>
<td>Astaroth</td>
<td>Astaroth uses WMIC to execute payloads. [35]</td>
</tr>
<tr>
<td>BlackEnergy</td>
<td>A BlackEnergy 2 plug-in uses WMI to gather victim host details. [14]</td>
</tr>
<tr>
<td>Cobalt Strike</td>
<td>Cobalt Strike can use WMI to deliver a payload to a remote host. [8]</td>
</tr>
<tr>
<td>Deep Panda</td>
<td>The Deep Panda group is known to utilize WMI for lateral movement. [49]</td>
</tr>
<tr>
<td>DustySky</td>
<td>The DustySky dropper uses Windows Management Instrumentation to extract information about the operating system and whether an anti-virus is active. [25]</td>
</tr>
<tr>
<td>Emotet</td>
<td>Emotet has used WMI to execute powershell.exe. [48]</td>
</tr>
<tr>
<td>Empire</td>
<td>Empire can use WMI to deliver a payload to a remote host. [10]</td>
</tr>
<tr>
<td>EvilBunny</td>
<td>EvilBunny has used WMI to gather information about the system. [48]</td>
</tr>
<tr>
<td>FELIXROOT</td>
<td>FELIXROOT uses WMI to query the Windows Registry. [14]</td>
</tr>
<tr>
<td>FIN8</td>
<td>FIN8's malicious spear phishing payloads use WMI to launch malware and spawn cmd.exe execution. FIN8 has also used WMIC during and post compromise cleanup activities. [64] [57]</td>
</tr>
<tr>
<td>FlawedAmmyy</td>
<td>FlawedAmmyy leverages WMI to enumerate anti-virus on the victim. [43]</td>
</tr>
<tr>
<td>GravityRAT</td>
<td>GravityRAT collects various information via WMI requests, including CPU information in the Win32_Processor entry (Processor ID, Name, Manufacturer and the clock speed). [20]</td>
</tr>
<tr>
<td>HALFBaked</td>
<td>HALFBaked can use WMI queries to gather system information. [28]</td>
</tr>
</tbody>
</table>
Define monitoring objectives

- Is it doable?
- Non-legit usage vs vulnerability detection
- Detect anomalies on network level

Support defender with:
- Evidence, context (who/what), attempts (success/failure)
Our very first attempt

- What does WMI look like from network perspective?

- PS> Get-Wmiobject Win32_computersystem – Computername WIN-J0GNFCAISH2.testing.local – Credential <lookwhoistalking> Ipconfig.exe

- [Not authorized]. We knew.
<table>
<thead>
<tr>
<th>No.</th>
<th>Source</th>
<th>Source Port</th>
<th>Destination</th>
<th>Destination Port</th>
<th>Interface UUID</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>27</td>
<td>10.1.1.102</td>
<td>49751</td>
<td>10.1.1.101</td>
<td>135</td>
<td>99fcfec4-5260-101b-bbcb-00aa0021347a, 99...</td>
<td>DCERPC</td>
</tr>
<tr>
<td>28</td>
<td>10.1.1.101</td>
<td>135</td>
<td>10.1.1.102</td>
<td>49751</td>
<td>DCERPC</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>10.1.1.102</td>
<td>49751</td>
<td>10.1.1.101</td>
<td>135</td>
<td>DCERPC</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>10.1.1.101</td>
<td>135</td>
<td>10.1.1.102</td>
<td>49751</td>
<td>DCERPC</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>10.1.1.102</td>
<td>49752</td>
<td>10.1.1.101</td>
<td>135</td>
<td>0000001a-0000-0000-c000-000000000046</td>
<td>DCERPC</td>
</tr>
<tr>
<td>51</td>
<td>10.1.1.101</td>
<td>135</td>
<td>10.1.1.102</td>
<td>49752</td>
<td>DCERPC</td>
<td></td>
</tr>
</tbody>
</table>

**Abstract Syntax: IOXIDResolver V0.0**

- Interface: IOXIDResolver UUID: 99fcfec4-5260-101b-bbcb-00aa0021347a
- Interface Ver: 0
- Interface Ver Minor: 0
- Transfer Syntax[1]: 32bit NDR V2
Initial payload filter

- c4 fe fc 99 60 52 1b 10 bb cb 00 aa 00 21 34 7a

- Turned it into a rule like:
  (“guess this is WMI!”;
  content:”|c4 fe fc 99 60 52 1b 10 bb cb 00 aa 00 21 34 7a|”)

- Risk of false positive?
Figure 2. Simple RPC call
Map onto RPC preprocessor

Figure 2. Simple RPC call
Detection pattern changed into

(msg:“guess this is WMI !”;
dce_iface:99fcfecd-5260-101b-bbcb-00aa0021347a; …)

> Did a re-test: triggers again!

Preprocessor abstracts away: used endianess type
Differentiate legit - non-legit

- Up to here: 1st detection pattern defined
- Next, few suggestions to distinguish:
  - Based on time? Used credentials? traffic path?
- What is expected to be legitimate traffic in the enterprise?
Differentiate legit - non-legit

- Engage with system administrators
- Establish a policy if not already there:
  “We shall administer <this and that> using WMI only from < *endpoints > “

*) Typically steppingstone-like

Pitfall: change management
Payload pattern and policy combined

Whitelist approach:

alert tcp !$legitimate_sources any -> $protected_targets 135
...
(msg:"guess this is WMI !"
; dce_iface:99fcfec4-5260-101b-bbcb-00aa0021347a; .. )
Next: offer our rule for re-test

Revisited by redteam

Various clients
- Windows powershell
- Linux Impacket (low-level network protocol library)
- Using unauthorized account
Results

- Windows client triggered the rule as expected
- However: “Linux” based client did not ;-(
- What next?
[MS-WMI] Protocol Initialization

- “The client MUST call the IWbemLevel1Login::NTLMLogin method.

- The IWbemLevel1Login interface allows a user to connect to the management services interface in a particular namespace.

- The interface MUST be uniquely identified by the UUID {F309AD18-D86A-11d0-A075-00C04FB68820} “

3.1 Server Details WMI

CLSID_WbemLevel1Login ({8BC3F05E-D86B-11D0-A075-00C04FB68820})
CLSID_WbemBackupRestore ({C49E32C6-BC8B-11D2-85D4-00105A1F8304})

The following GUIDs are used for the interfaces:
IID_IWbemLevel1Login ({F309AD18-D86A-11d0-A075-00C04FB68820})
IID_IWbemServices ({9556DC99-828C-11CF-A37E-00AA003240C7})
IID_IWbemBackupRestore ({C49E32C7-BC8B-11d2-85D4-00105A1F8304})
IID_IWbemBackupRestoreEx ({A359DEC5-E813-4834-B91E-000D34B00000})

Do cross-check

https://github.com/CoreSecurity/impacket/
Detection pattern category

Rpc stage:
- Rule – AppID for service winmgmt

Indicator WMI, presumed success
- IID Iwbemlevel../opnum 6
- IID Iwbemservices/.. (Bonus)
Ruleset becomes (condensed)

‘RPC’ stage

- #100: dce_iface:000001a0-0000-0000-c000-0000000000046; dce_opnum: 4; dce_stub_data; content: "|5e f0 c3 8b 6b d8 d0 11 a0 75 00 c0 4f b6 88 20|

WMI’ stage; Golden Rule:

- #110: dce_iface:F309AD18-D86A-11d0-A075-00C04FB68820; dce_opnum: 6

- #114: dce_iface:9556dc99-828c-11cf-a37e-00aa003240c7
- #116, #120: likewise for IID’s IEnumwbemobject/IWbemclassobject
Detect brute force attempts

- Indicator WMI call (rule #110) and subsequent calls may not happen
- Force multiple failed attempts:
  - PS C:>\wmic /node: "10.1.1.101" process call create "cmd.exe /c ipconfig.exe"
Server replies with ‘fault PDU’


Version: 5
Version (minor): 0
Packet type: Fault (3)
Packet Flags: 0x03
Data Representation: 10000000 (Order: Little-endian, Char: ASCII, Float: IEEE)
Frag Length: 32
Auth Length: 0
Call ID: 2
Alloc hint: 32
Context ID: 0
Cancel count: 0
Status: nca_s_fault_access_denied (0x80000005)

[Expert Info (Note/Response): Fault: nca_s_fault_access_denied]
[Fault: nca_s_fault_access_denied]
[Severity level: Note]
[Group: Response]
Opnum: 4
[Request in frame: 8]
[Time from request: 0.873886000 seconds]
Fault PDU structure

- pubs.opengroup.org/
- onlinepubs/9629399/chap12.htm

The fault PDU

The IDL declaration of the fault PDU is as follows:

typedef struct {
  /* start 8-octet aligned */
  /* common fields */
  u_int8 rpc_vers = 5; /* 00:01 RPC version */
  u_int8 rpc_vers_minor; /* 01:01 minor version */
  u_int8 PTYPE = fault; /* 02:01 fault PDU */
  u_int8 pfc_flags; /* 03:01 flags */
  byte packed_drep[4]; /* 04:04 NDR data rep format label*/
  u_int16 frag_length; /* 08:02 total length or fragment */
  u_int16 sub_length; /* 08:02 length of auth_value */
  u_int32 call_id; /* 12:04 call identifier */
  /* end common fields */
  /* needed for request, response, fault */
  u_int32 alloc_hint; /* 16:04 allocation hint */
  p_context_id_t p_cont_id; /* 20:02 proc context, i.e. data rep */
  /* needed for response or fault */
  u_int8 cancel_count; /* 22:01 received cancel count */
  u_int8 reserved; /* 23:01 reserved, m.b. */
  /* fault code */
  u_int32 status; /* 24:04 run-time fault code or zero */
  /* always pad to next 8-octet boundary */
  u_int8 reserved2[4]; /* 28:04 reserved padding, m.b. */
  /* stub data here, 8-octet aligned */
  ;
  ;
  ;
  ;
  /* optional authentication verifier */
  /* following fields present iff auth_length != 0 */
  auth_verifier_co_t auth_verifier; /* xx:yy */
} rpoconn_fault_hdr_t;
RPC access denied

(msg:" RPC PDU - fault_access_denied response 0x00000005`; flow:to_client, ... ;

content:"|05 00 03|`; offset:0; depth:3;

byte_test:4,=,0x00000005,24,dce;

metadata:service dcerpc; ... )
Test the rule set

- Engage with system administrators
- They did the heavy lifting!

- Be aware: keep policy implementation up-to-date
Test blueprint

- Targets { Windows 2012R2/2016 }
- Clients { Windows cli:wmic, ps:get-wmiobject, imp:wmiquery, imp:wmiexec }
- Fully privileged/authorized + unprivileged account
- Result from client perspective: success, failure
### Visualize results as heatmap

<table>
<thead>
<tr>
<th>RPC</th>
<th>WMI</th>
<th>WMI</th>
<th>Reject</th>
</tr>
</thead>
</table>

- **RPC**
  - “Success”
  - “Denied”

- **WMI**

- **Reject**

Use case →
Workaround for reject rule

(msg:"RPC generic reject"; content:"|05 00 03|"; offset:0; depth:3; byte_test:4,=,0x00000005,24,little; ...)

Note:
- “access denied” common as part of server-client negotiation
- better: use as correlation, apply with threshold
- maybe better: indicator higher in protocol stack
Re-test compare both versions reject rule

<table>
<thead>
<tr>
<th></th>
<th>RPC</th>
<th>WMI</th>
<th>WMI</th>
<th>Reject</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request denied</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Use case →
• Improved rule set
• Testing involved both system administrators & redteam
• Rules fire when expected to fire, and when not
Are we done and ready to deploy?

#1 en #2 major concern for a security analyst?
How do we know and find out?

- False positives
- False negatives
Assess the ruleset quality

- From attacker defender perspective

- What means are left for an attacker to evade detection defender to assess the ruleset pro-actively?
WMI security

- Namespace (S/DACL, securable objects)
- Transport level
  - DCOM/RPC
    - Impersonation
  - Authentication level options
    - Server and client require Mutual agreement
    - None / connect / call / pkt / pktintegrty / pkt_privacy
    - “Privacy”: encrypts argument values

https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-rpce/425a7c53-c33a-4868-8e5b-2a850d40dc73
Options and defaults

- Dcomcnfg.exe
- UI to registry
- Machine wide
- Process wide

- Default level: “Connect”
Setup pristine lab environment

Client / Server
- W 2016 domain controller + member + standalone
- W 10 standalone
- Linux client (impacket)

Encryption
- [d] default
- [ec] client only
- [ecs] client + server
Client methods

1. Get-CimInstance -ClassName Win32_OperatingSystem -CimSession $csd
2. Get-WmiObject win32_computersystem -ComputerName _ -Credential adm
3. wmic /node:<> /user:"administrator" cpu get name
4. wmiquery.py <>/administrator@<> -file wql.file
5. pth-wmic -U Administrator<> //<> "select Name from Win32_UserAccount"
6. like @1, with invalid password
7. like @ 5, with invalid password
8. like @2, valid credentials, however not authorized
Force clientside encryption ("ec")

<table>
<thead>
<tr>
<th>Use case →</th>
<th>Win PS</th>
<th>Wmic</th>
<th>Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WMI</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WMI</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reject</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Force pkt_privacy on both sides

Win PS
RPC
WMI

Wmi

Linux

Use case →
Extend the ruleset

- iwbemlevel1login: f309ad18-d86a-11d0-a075-00c04fb68820
  Rule #112 -> "|18 ad 09 f3 6a d8 d0 11 a0 75 00 c0 4f b6 88 20|

- IWebmServices: 9556dc99-828c-11cf-a37e-00aa003240c7
  Rule #115 -> "|99 dc 56 95 8c 82 cf 11 a3 7e 00 aa 00 32 40 c7|"
Re-run with extended rule set

<table>
<thead>
<tr>
<th></th>
<th>Win PS</th>
<th>Wmic</th>
<th>Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>RPC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WMI</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WMI</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reject</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Use case →
Takeaways

- Network level detection is doable
- Github.com/KPN-CISO/Network-Detection
- Cross-team collaboration is key
- Based on testing so far, happy with FP
- FN a concern

- Future research evasion techniques, improve detection, resolve open ends
Final thoughts...

“One sunny day
A few lacking rules became a big takeaway
To overcome the annoyance and frustration
With the support of both admins and reds
We fulfilled the promise  nothing is beyond our reach
Now it’s time to call on you to have a look and make it better
Administrator, defender or attacker, the role doesn’t matter
Suggesting to combine it with a Belgian beer
Let me say it loud and clear
I feel confident we can work it out together!“