Creating an Agent-less Host Intrusion Detection System using PowerShell and WMI

Jared Atkinson & Matt Graeber

Introduction

Matt Graeber - @mattifestation

- R&D Capability Lead – Veris Group – Adaptive Threat Division
- Former:
  - Malware Reverse Engineer at FireEye (FLARE Team)
  - Read Team Operator at government red team
  - U.S. Navy Linguist
- Cloud and Datacenter Management MVP – PowerShell
  - Neither a cloud nor a datacenter expert, FYI.
- Creator of PowerSploit, PowerShellArsenal, etc.

Introduction

Jared Atkinson - @jaredcatkinson

- Hunt Technical Lead – Veris Group – Adaptive Threat Division
- Former
    - U.S. Air Force Hunt (2011 – 2015)
- 2015 Black Hat Minesweeper Champion
- Moderator of the PowerShell.com "Security Forum"
- Developer of
    - PowerForensics
    - WMIEvent
    - Uproot IDS

What is Uproot?

- Uproot (www.github.com/Invoke-IR/Uproot)
  - Host based Intrusion Detection System built on permanent WMI event subscriptions
  - Leverages WmiEvent module to easily manage subscriptions
- WmiEvent (www.github.com/Invoke-IR/Uproot)
  - PowerShell module that abstracts the complexities of permanent WMI event subscriptions

Why are we here?

- Matt – Some colleagues were investigating a breach involving WMI persistence and I was asked how one would effectively detect the creation of permanent WMI event subscriptions.
- Jared – As a consultant, we are often not allowed to dictate configuration changes or software additions, but are responsible for near real-time monitoring. Permanent WMI event subscriptions offer support across all versions of Windows (past and present) for monitoring system changes as they happen.

WMI Eventing Refresher – Event Classes

Two types of event classes:

- Extrinsic:
  - “not linked to changes in the WMI data model”[1] – i.e. provider specific
  - Does not require a polling interval – i.e. no missed firings
  - Limited set
  - E.g. RegistryKeyChangeEvent
- Intrinsic:
  - “occurs in response to a change in the standard WMI data model”[1]
  - Requires polling interval – i.e. can miss firings
  - Limited only by the classes present in the WMI repository
  - E.g. __InstanceCreationEvent

[1] – “Determining the Type of Event to Receive” https://msdn.microsoft.com/en-us/library/windows/desktop/aa390355(v=vs.85).aspx

WMI Eventing Refresher – Events

- Local WMI events
  - Register-WmiEvent, Register-CimIndicationEvent
- Permanent WMI events
  - Set-WmiInstance, New-CimInstance
  - Requires the following instances:
    1. __EventConsumerClass – e.g. CommandLineEventConsumer
    2. __EventFilter – WMI event query
    3. __FilterToConsumerBinding

WMI Eventing Refresher – __EventFilter

- Intrinsic event filter example:
  - `SELECT * FROM __InstanceModificationEvent WITHIN 5 WHERE TargetInstance ISA 'Win32_Service' and TargetInstance.State = 'Running‘`
  - `SELECT * FROM __InstanceCreationEvent WITHIN 10 WHERE TargetInstance ISA 'Win32_StartupCommand'`

- Extrinsic event filter example:
  - `SELECT * FROM Win32_VolumeChangeEvent WHERE EventType = 2`
  - `SELECT * FROM Win32_ProcessStartTrace WHERE ProcessName LIKE '%chrome%'`

WMI Eventing Refresher – __EventConsumer

Standard event consumers

- LogFileEventConsumer
- ActiveScriptEventConsumer
- NTEventLogEventConsumer
- SMTPEventConsumer
- CommandLineEventConsumer

“Signature” Development – Methodology (1/3)

- Identify what you’d like to detect.
    - i.e. Identify common attacker actions
    1. Service creation
    2. Registry persistence – think Autoruns
    3. Lateral movement
    4. WMI persistence
    5. Etc.
- Consider if there is already current detection
    - Event log entries
    - Command-line auditing
    - Applocker

“Signature” Development – Methodology (2/3)

1. Prioritize utilization of extrinsic event classes
   - No chance of missing events – no polling interval required

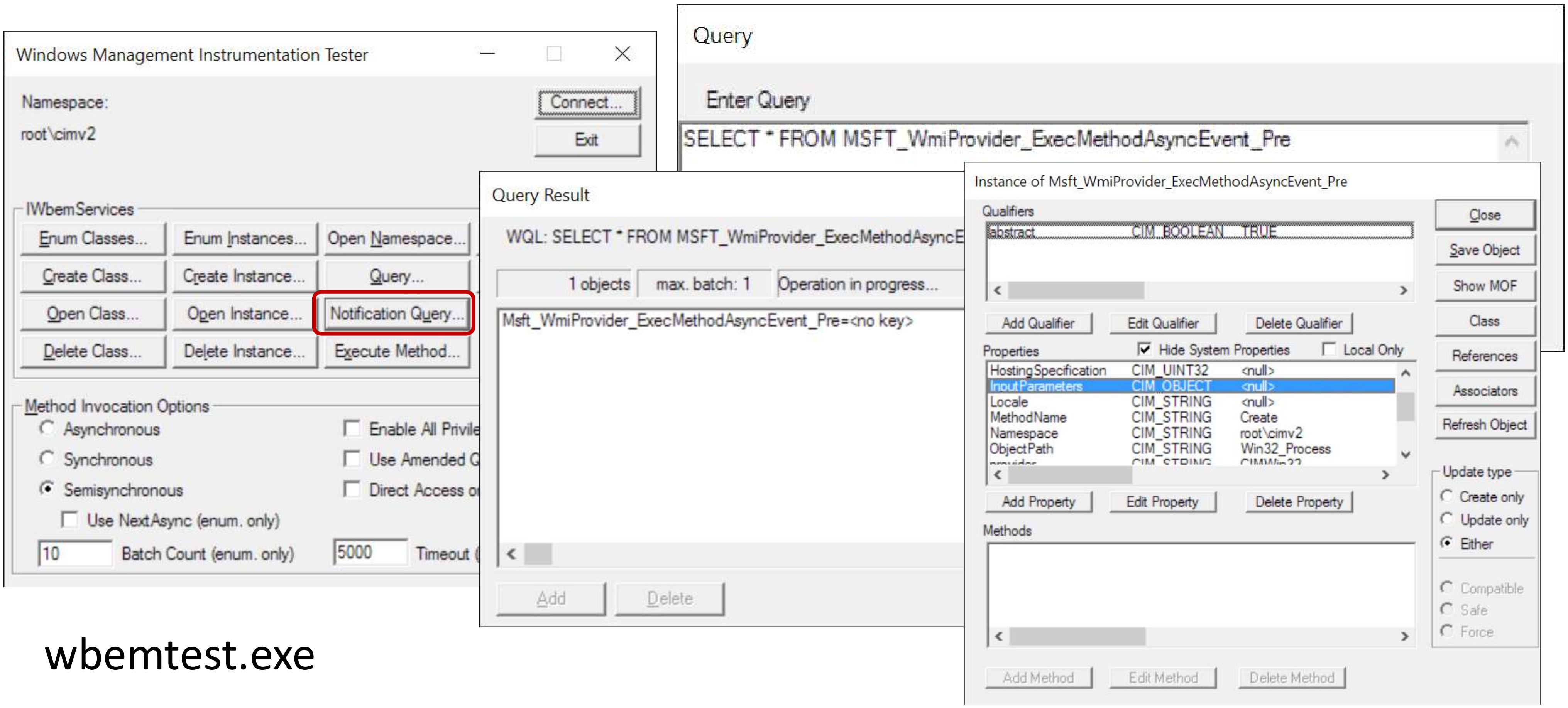
2. Fall back to intrinsic events if necessary

But how do I know what events are available???

- PowerShell, of course!

Demo time

“Signature” Development – Methodology (3/3)


Windows Management Instrumentation Tester

Namespace: root\cimv2

Method Invocation Options: [✓ Semisynchronous]

10 Batch Count (enum. only)

5000 Timeout

Query

Enter Query

```
SELECT * FROM MSFT_WmiProvider_ExecMethodAsyncEvent_Pre
```

Query Result

```
WQL: SELECT * FROM MSFT_WmiProvider_ExecMethodAsyncE
1 objects    max. batch: 1    Operation in progress...
Msft_WmiProvider_ExecMethodAsyncEvent_Pre=<no key>
```

Instance of Msft_WmiProvider_ExecMethodAsyncEvent_Pre

Qualifiers

```
abstract              CIM_BOOLEAN   TRUE
```

Properties [✓ Hide System Properties]

```
HostingSpecification  CIM_UINT32    <null>
InputParameters       CIM_OBJECT    <null>
Locale                CIM_STRING    <null>
MethodName            CIM_STRING    Create
Namespace             CIM_STRING    root\cimv2
ObjectPath            CIM_STRING    Win32_Process
```

Update type: [✓ Either]

wbemtest.exe

“Signature” Development - Scenario

- You have a good idea of attacker actions but you don’t have a specific WMI class for detection in mind.
  - E.g. lateral movement
  - Is there a Win32_LateralMovement class??? No. ☹
- Let’s explore a bit and see if there are any events that stand out.
- Some creativity required…

Demo time

“Signature” Development - Results

- As a result of exploring extrinsic events, we came up with some of the following signatures:

1. `SELECT * FROM MSFT_WmiProvider_ExecMethodAsyncEvent_Pre WHERE ObjectPath="Win32_Process" AND MethodName="Create"`
2. `SELECT * FROM MSFT_WmiProvider_ExecMethodAsyncEvent_Pre WHERE ObjectPath="StdRegProv"`
3. `SELECT * FROM Win32_ModuleLoadTrace WHERE FileName LIKE "%System.Management.Automation%.dll%"`
4. `SELECT * FROM __ClassCreationEvent`
5. `SELECT * FROM MSFT_WmiProvider_CreateInstanceEnumAsyncEvent_Pre WHERE ClassName="Win32_Process"`