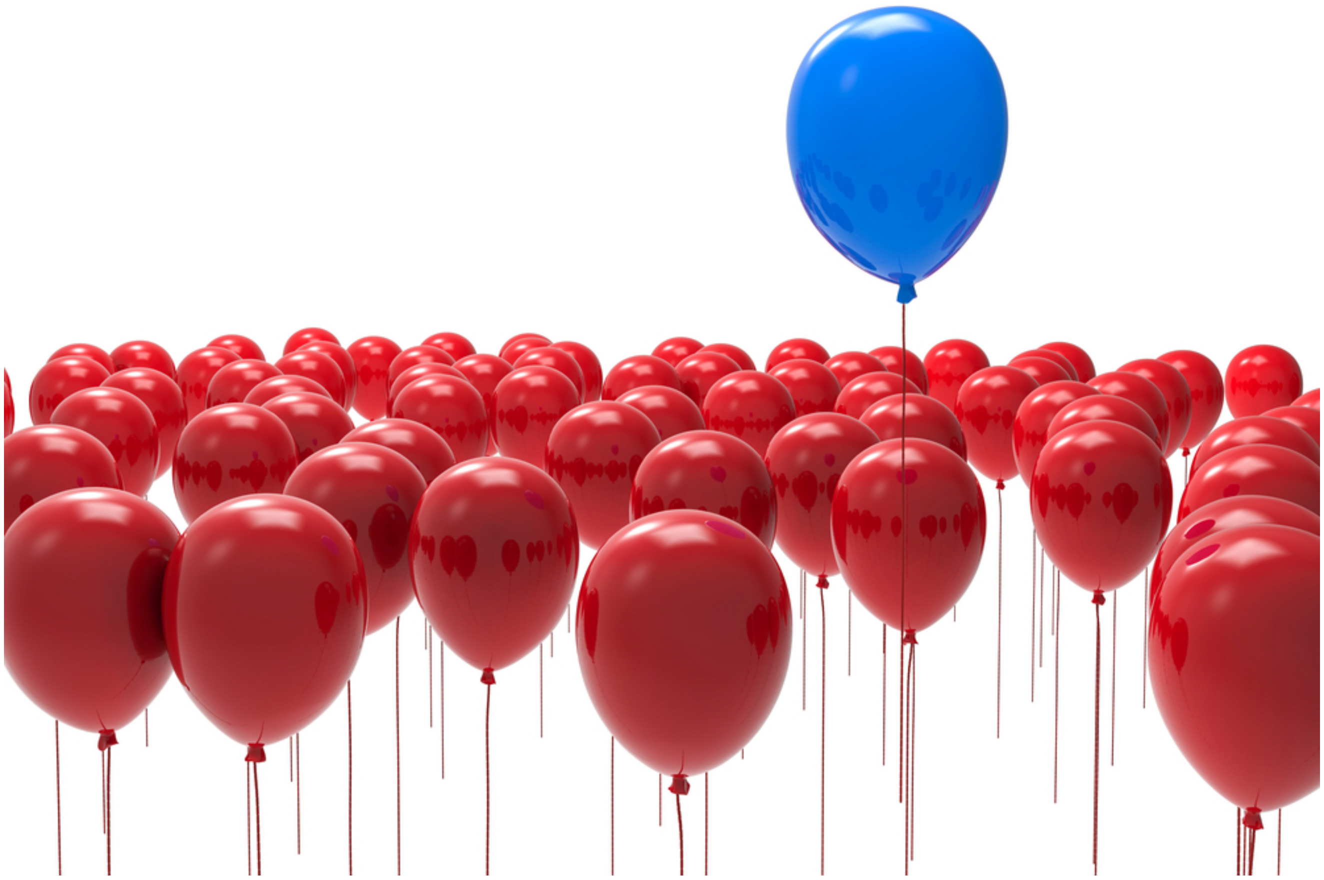
Detecting Lateral Movement via WinRM Using KQL.

in.security/detecting-lateral-movement-via-winrm-using-kql/

May 3, 2021



Over the past few months we've been looking a little more into the detection methods we might use to identify strange activity within a given environment.

A lot of this research stems from questions asked by our clients following a technical engagement, or questions from students that have taken our Hacking Enterprises training. With the arrival of our new Defending Enterprises training this year where we look at detection methods from our 'Top 10 in-the-field attacks', a lot of this research naturally evolved and we found this element particularly interesting.

Microsoft Azure Sentinel is fast becoming our go-to SIEM as it not only brings the accessibility of cloud services, but a wealth of functionality at a relatively low price point. In this blog we're going to take a brief look at the power of the Kusto Query Language (KQL).

Detecting Lateral Movement Through WinRM

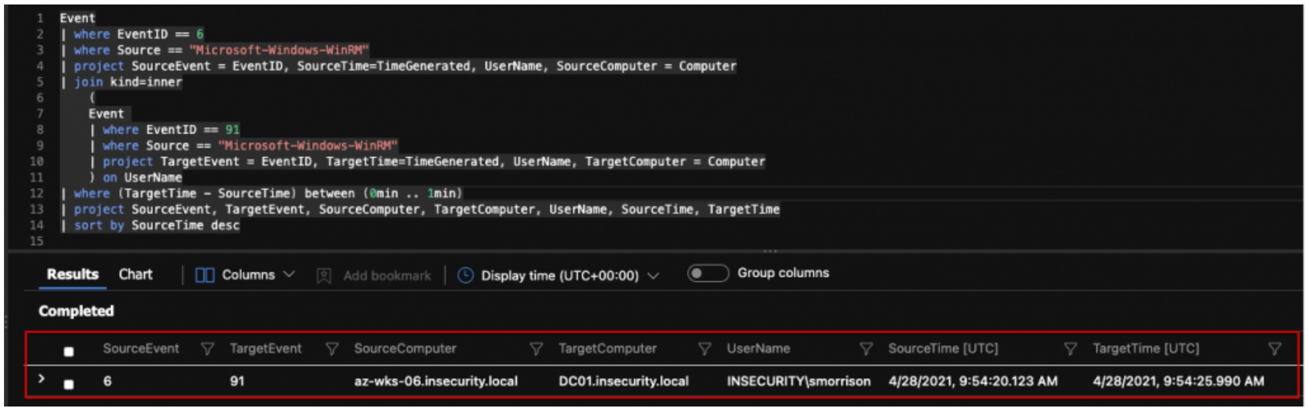
Under the context of internal network monitoring, we wanted a quick and easy method to identify when a WinRM or PowerShell Remoting session had been instigated, but we also wanted an idea of where the user both had originated from, and targeted. As PowerShell Remoting uses WinRM to establish connections, the same Indictor of Attack (IOA) could be used.

When a WinRM connection is initialised *EventID 6* will be recorded (the source host) and when a WinRM connection is received *EventID 91* is recorded (the target host). Both events will be logged in ***Microsoft-Windows-WinRM/Operational*** (Windows Remote Management through the GUI).

Therefore, to chain together such events we can use a Time Window Join operation to map *source* > *target* using a query such as the following:

In short, this query will look for *Event ID 6* in the *Microsoft-Windows-WinRM* log and, if found, a second query is executed that looks for *Event ID 91*, again in the *Microsoft-Windows-WinRM* log, but the events have to occur within 1 minute of each other (in larger, busy environments this timing may need to be tuned). If both events are identified (matched on username), details of the event are displayed, as shown in the example below.

```
Event
| where EventID == 6
| where Source == "Microsoft-Windows-WinRM"
| project SourceEvent = EventID, SourceTime=TimeGenerated, UserName, SourceComputer
= Computer
| join kind=inner
    (
    Event
    | where EventID == 91
    | where Source == "Microsoft-Windows-WinRM"
    | project TargetEvent = EventID, TargetTime=TimeGenerated, UserName,
TargetComputer = Computer
    ) on UserName
| where (TargetTime - SourceTime) between (0min .. 1min)
| project SourceEvent, TargetEvent, SourceComputer, TargetComputer, UserName,
SourceTime, TargetTime
| sort by SourceTime desc
```



```
Event
| where EventID == 6
| where Source == "Microsoft-Windows-WinRM"
| project SourceEvent = EventID, SourceTime=TimeGenerated, UserName, SourceComputer = Computer
| join kind=inner
    (
    Event
    | where EventID == 91
    | where Source == "Microsoft-Windows-WinRM"
    | project TargetEvent = EventID, TargetTime=TimeGenerated, UserName, TargetComputer = Computer
    ) on UserName
| where (TargetTime - SourceTime) between (0min .. 1min)
| project SourceEvent, TargetEvent, SourceComputer, TargetComputer, UserName, SourceTime, TargetTime
| sort by SourceTime desc
```

Completed

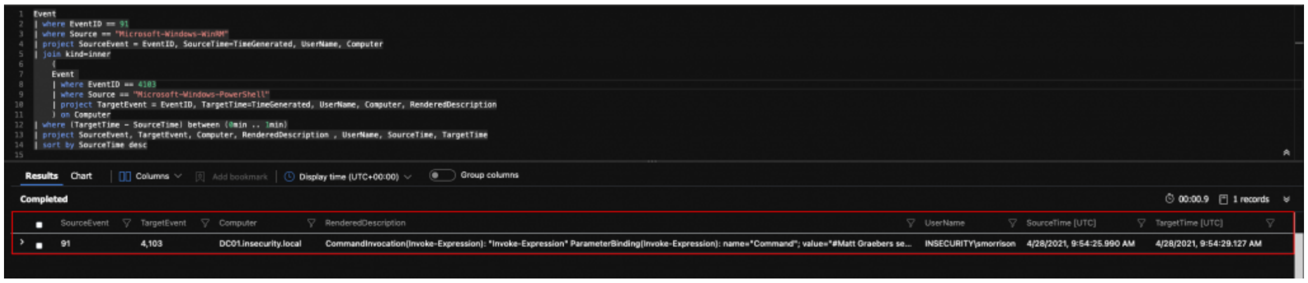
SourceEvent	TargetEvent	SourceComputer	TargetComputer	UserName	SourceTime [UTC]	TargetTime [UTC]
6	91	az-wks-06.insecurity.local	DC01.insecurity.local	INSECURITY\smorrison	4/28/2021, 9:54:20.123 AM	4/28/2021, 9:54:25.990 AM

Identifying Further Activity Through PowerShell Logging

We can further investigate any activities by collating *EventID 91* (a WinRM connection has been received – i.e. the target host) with PowerShell logging *EventID 4103* (Module Logging), to see what may have been executed after the connection was made.

As per the previous query, we're using a *join* statement with much of the same underlying logic. In this instance if *Event ID 91* is identified, a second query is executed that looks for *Event ID 4103* within 1 minute of the initial connection. If both events are identified, details are displayed, as shown in the example below.

```
Event
| where EventID == 91
| where Source == "Microsoft-Windows-WinRM"
| project SourceEvent = EventID, SourceTime=TimeGenerated, UserName, Computer
| join kind=inner
    (
    Event
    | where EventID == 4103
    | where Source == "Microsoft-Windows-PowerShell"
    | project TargetEvent = EventID, TargetTime=TimeGenerated, UserName, Computer,
RenderedDescription
    ) on Computer
| where (TargetTime - SourceTime) between (0min .. 1min)
| project SourceEvent, TargetEvent, Computer, RenderedDescription , UserName,
SourceTime, TargetTime
| sort by SourceTime desc
```



```
Event
| where EventID == 91
| where Source == "Microsoft-Windows-WinRM"
| project SourceEvent = EventID, SourceTime=TimeGenerated, UserName, Computer
| join kind=inner
    (
    Event
    | where EventID == 4103
    | where Source == "Microsoft-Windows-PowerShell"
    | project TargetEvent = EventID, TargetTime=TimeGenerated, UserName, Computer, RenderedDescription
    ) on Computer
| where (TargetTime - SourceTime) between (0min .. 1min)
| project SourceEvent, TargetEvent, Computer, RenderedDescription , UserName, SourceTime, TargetTime
| sort by SourceTime desc
```

Completed 00:00.9 1 records

SourceEvent	TargetEvent	Computer	RenderedDescription	UserName	SourceTime [UTC]	TargetTime [UTC]
91	4,103	DC01.insecurity.local	CommandInvocation(Invoke-Expression): "Invoke-Expression" ParameterBinding(Invoke-Expression): name="Command"; value="#Matt Graebers se...	INSECURITY\smorrison	4/28/2021, 9:54:25.990 AM	4/28/2021, 9:54:29.127 AM

We always recommend establishing a decent baseline to know what normal activity looks like before diving into trying to identify the abnormal, otherwise it becomes a very difficult task to spot the haystack let alone the needle!

If this has you interested, why not check out some of our other blue orientated posts below.

- Detecting Pass-the-Ticket (PtT) Attacks
- PsExec. I thought we were friends
- Getting Started with Sysmon for Linux
- What the Heck PsExec!
- Detecting Lateral Movement via WinRM Using KQL