The Power and Implications of Enterprise Incident Response with PowerShell

GIAC (GSEC) Gold Certification

Author: Robert Adams, robert.louis.adams@gmail.com

Advisor: Rob VandenBrink

Accepted: December 17th, 2015

Abstract

PowerShell is a Windows scripting language that can be leveraged to deliver enterprise-wide incident response. Although PowerShell's remoting technology offers a secure, flexible, and scalable solution, there are implications that need to be considered. This paper will address authentication, performance, and the integrity of data as it relates to incident response. A virtualized lab environment will be used to illustrate key research findings, with the ultimate goal of addressing both the appropriate and inappropriate uses of PowerShell remoting. This will enhance the security practitioner's ability to perform incident response, while understanding the most suitable use-cases for using PowerShell.

1. The Scenario

The marketing department of Company X has been the target of a phishing attack.

The Security Operations Center (SOC) starts receiving the reports: suspicious e-mails are arriving in employee mailboxes. The emails are spoofed. They also happen to contain a malicious attachment.

After reverse engineering the attachment the security analysts figure it out. The attacker is using VB macros to drop a malicious file on the user's disk, which then makes an outbound connection to download a malicious binary. Further analysis reveals that this hacker is writing to a Windows key.

A few dozen users and machines are potentially compromised. ***What do you do?***

1.1 Introducing PowerShell

Solving the above scenario with PowerShell turns out to be simple. The mean-time-to-response (MTTR) is an important metric that characterizes the efficiency of a security team's response efforts. PowerShell is an ideal candidate that can be used to increase a SOC's response time. With the aid of PowerShell's remoting feature, a simple script can be written to look for the values in the registry.

Script execution is not limited to one machine at a time. PowerShell remoting, once configured across the enterprise, can be leveraged to collect data from multiple endpoints at the same time.

2. Analysis in a Virtualized Lab

This section assumes prior knowledge of PowerShell. Appendix A contains an overview of PowerShell and PowerShell remoting (PSRemoting) for those who need a refresher on the functionality of PowerShell.

The technique mentioned above, of using PowerShell to respond to a phishing incident, can be further examined in a lab environment. A set of Windows virtual machines has been configured and joined to a test domain. A Windows Server 2012 R2 image is configured as the domain controller (DC), and there are three Windows 8.1 Enterprise hosts that are all joined to the domain. One of the three hosts will serve as the security analyst's machine, while the other two represent employees' workstations.

The following diagram illustrates the architecture of the test environment:

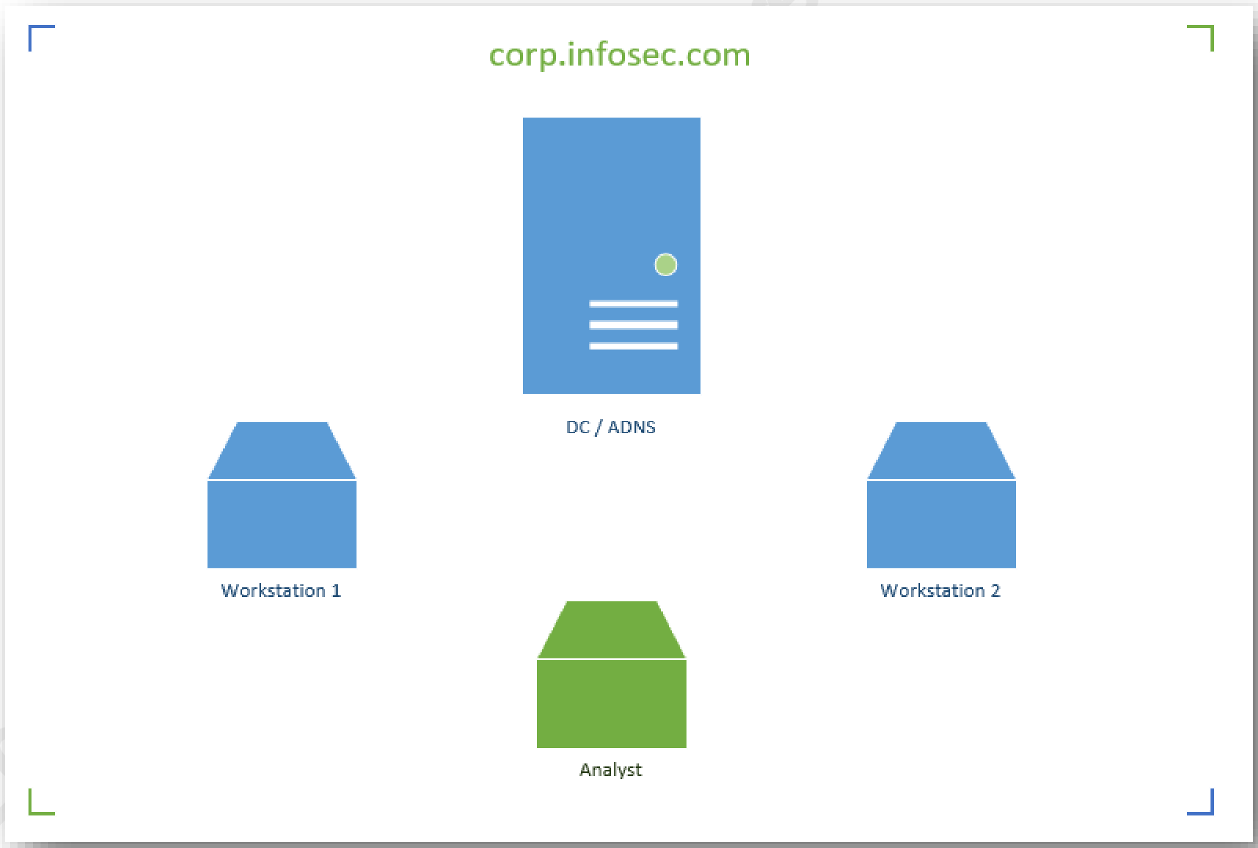
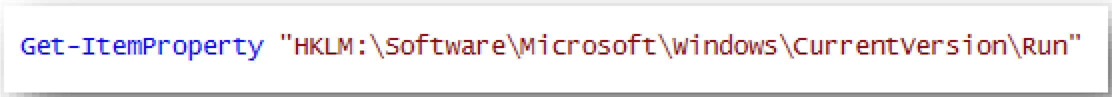


Figure 1 - Test Domain (corp.infosec.com)

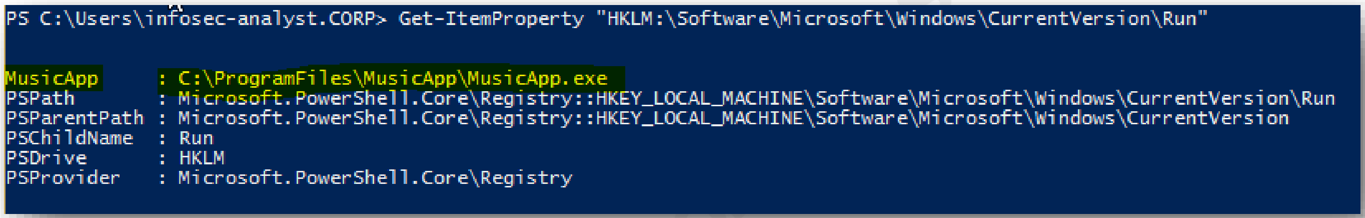
Using PowerShell, the security analyst can write a simple script to check the values of the registry key of interest: HKLM:\Software\Microsoft\Windows\CurrentVersion\Run.



```
Get-ItemProperty "HKLM:\Software\Microsoft\Windows\CurrentVersion\Run"
```

Figure 2 - PowerShell's Get-ItemProperty

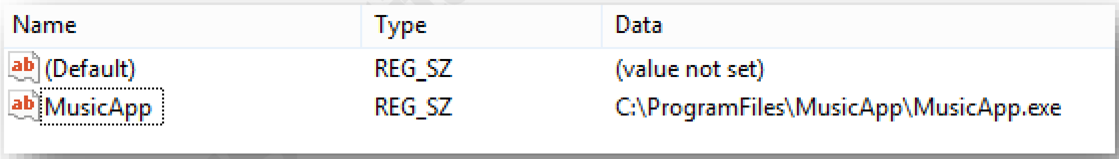
The code above simply makes use of PowerShell's **Get-ItemProperty** cmdlet to retrieve the values of the specified location. The result of running the cmdlet on the analyst's virtual machine is:



```
PS C:\Users\infosec-analyst.CORP> Get-ItemProperty "HKLM:\Software\Microsoft\Windows\CurrentVersion\Run"

MusicApp     : C:\ProgramFiles\MusicApp\MusicApp.exe
PSPath       : Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run
PSParentPath : Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion
PSChildName  : Run
PSDrive      : HKLM
PSProvider   : Microsoft.PowerShell.Core\Registry
```

Figure 3 - The Result of Get-ItemProperty

One value is returned: *MusicApp.exe* is set to run each time the system starts. The same information can be seen by selecting the same key-pair value from Windows's registry editor (regedit):



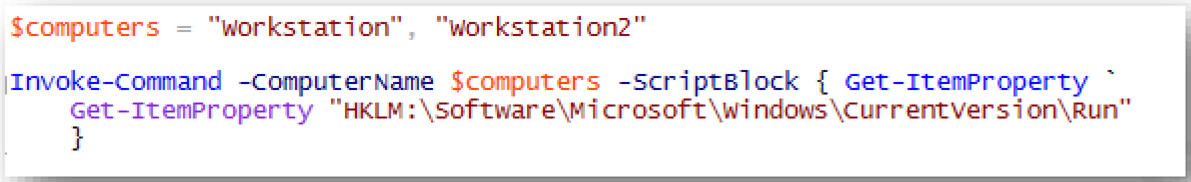
Name	Type	Data
(Default)	REG_SZ	(value not set)
MusicApp	REG_SZ	C:\ProgramFiles\MusicApp\MusicApp.exe

Figure 4 - Window's Regedit

2.1 Running the Command on Remote Machines

In the scenario, there is intelligence that the phishing campaign ultimately seeks to compromise machines and to establish persistency. Using the same PowerShell cmdlet as before, coupled with PowerShell remoting, the values from the registry can be concurrently retrieved from a number of systems.

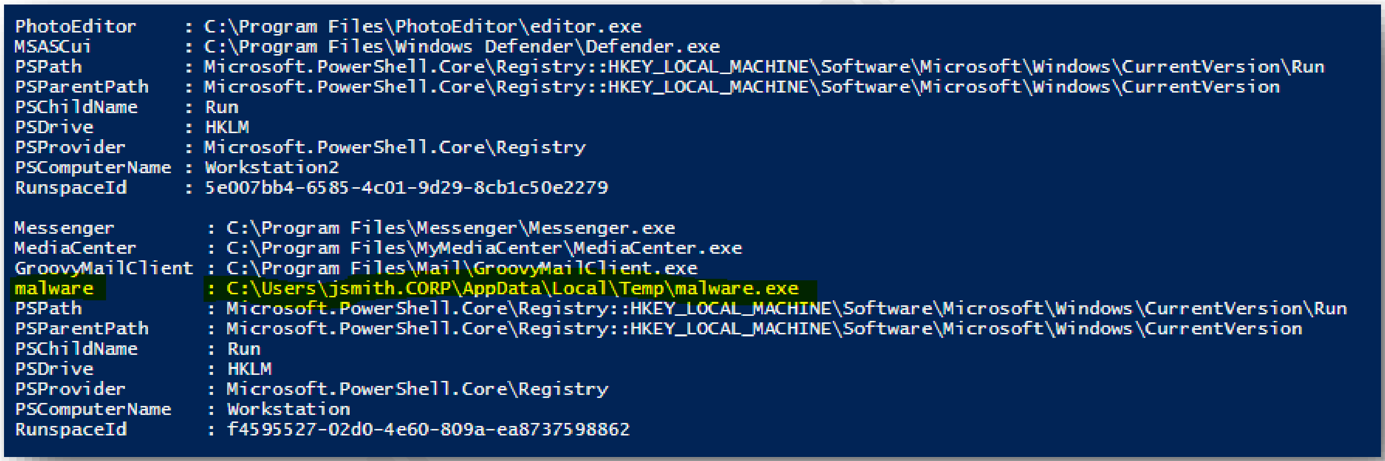
The following illustrates retrieving the *CurrentVersion**Run* registry values from both workstations in the virtualized lab environment. The code is executed from the security analyst's machine:



```
$computers = "Workstation", "Workstation2"

Invoke-Command -ComputerName $computers -ScriptBlock { Get-ItemProperty `
    Get-ItemProperty "HKLM:\Software\Microsoft\Windows\CurrentVersion\Run"
    }
```

Figure 5 - PowerShell Remoting via Invoke-Command

The script utilizes an array data structure to store each machine that data will be collected from, and then instantiates a remote connection to each system to retrieve the values from the CurrentVersion\Run registry key. The results:



```
PhotoEditor    : C:\Program Files\PhotoEditor\editor.exe
MSASCui        : C:\Program Files\Windows Defender\Defender.exe
PSPath         : Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run
PSParentPath   : Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion
PSChildName    : Run
PSDrive        : HKLM
PSProvider     : Microsoft.PowerShell.Core\Registry
PSComputerName : Workstation2
RunspaceId     : 5e007bb4-6585-4c01-9d29-8cb1c50e2279

Messenger         : C:\Program Files\Messenger\Messenger.exe
MediaCenter       : C:\Program Files\MyMediaCenter\MediaCenter.exe
GroovyMailClient  : C:\Program Files\Mail\GroovyMailClient.exe
malware           : C:\Users\jsmith.CORP\AppData\Local\Temp\malware.exe
PSPath            : Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run
PSParentPath      : Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion
PSChildName       : Run
PSDrive           : HKLM
PSProvider        : Microsoft.PowerShell.Core\Registry
PSComputerName    : Workstation
RunspaceId        : f4595527-02d0-4e60-809a-ea8737598862
```

Figure 6 - The Result of Invoke-Command

There is a grouping of results that correspond to each machine. In the second grouping, it is apparent that there is a suspicious registry key value: it looks like "*malware.exe*" has made its way into the registry settings to maintain a foothold in the system. The path of the executable is also shown, revealing that the malicious code resides under **jsmith's** local application data folder.

2.2 PowerShell Coupled with Third-Party Binaries

The previous example is quite simplistic. For demonstrative purposes, there is an assumption that there is a static indicator of compromise (IOC) related to the targeted phishing campaign. The reality is that there are usually numerous indicators tied to a given threat, and these indicators are often difficult to track due to their polymorphic nature.

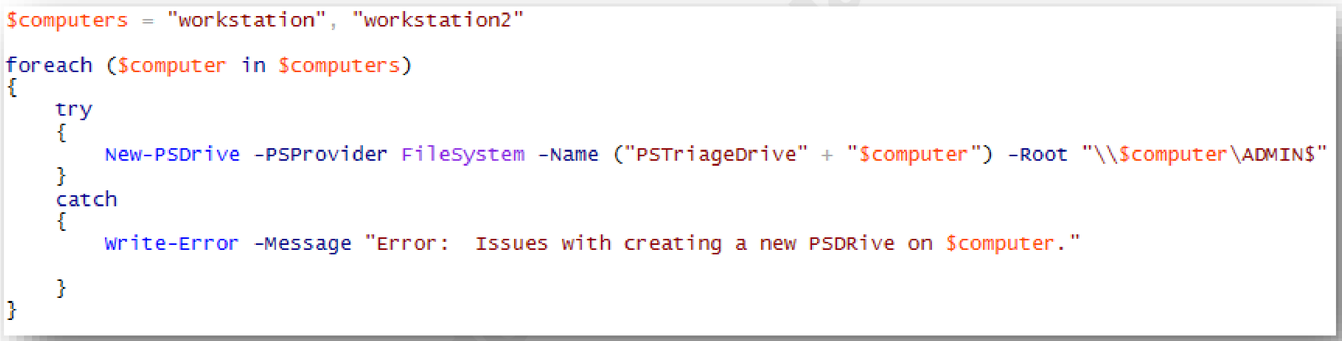
Polymorphic malware is designed to change constantly, ultimately seeking to evade detection. This shape-shifting characteristic is achieved through many methods: name changes, compression, and through encryption (Rouse, 2007). Interestingly, even though the malcode often changes, the main intent of the malicious software usually does not. As it relates to the targeted phishing example, the malware may evolve through various mutations. However, the code would still be delivered through the same vector, and still seek to ultimately gain persistency on victim machines.

PowerShell remoting can also be used to run third-party executables on a number of target machines. The pseudocode for achieving this functionality is as follows:

1. Create a PowerShell drive (PSDrive) that references each target's $ADMIN share
2. Use Copy-Item to move the required binary (.exe) to the newly created PSDrive
3. Use Invoke-Command to execute the binary on each target machine

The previous code snippet can be modified to look for evidence of persistency using Microsoft's SysInternal's Autoruns:

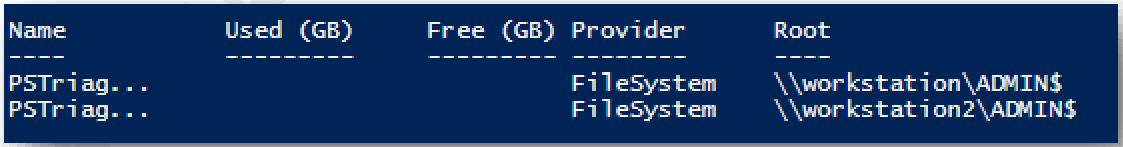
For each target in the $computers array, a new PSDrive is created that references each computers' ADMIN share.



```
$computers = "workstation", "workstation2"

foreach ($computer in $computers)
{
    try
    {
        New-PSDrive -PSProvider FileSystem -Name ("PSTriageDrive" + "$computer") -Root "\\$computer\ADMIN$"
    }
    catch
    {
        Write-Error -Message "Error:  Issues with creating a new PSDRive on $computer."

    }
}
```

Figure 7 - Creating PSDrives

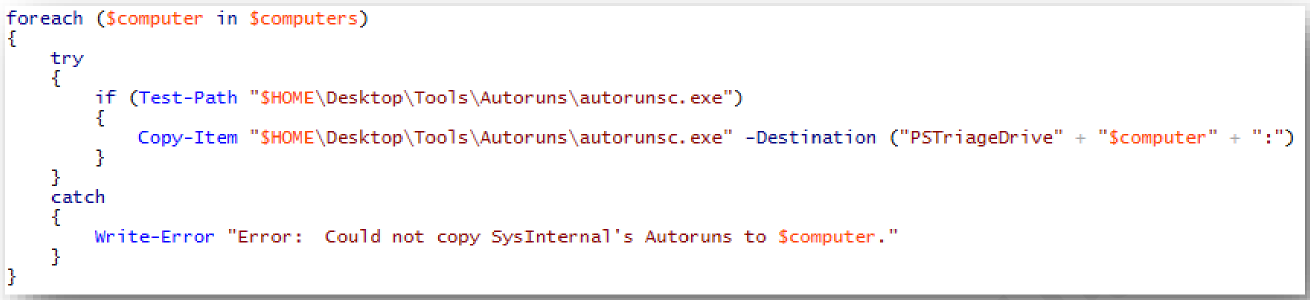
The results are illustrated:



```
Name           Used (GB)     Free (GB) Provider      Root
----           ---------     --------- --------      ----
PSTriag...                             FileSystem    \\workstation\ADMIN$
PSTriag...                             FileSystem    \\workstation2\ADMIN$
```

Figure 8 - The Result of New-PSDrive

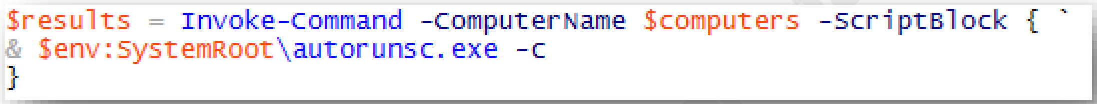
The list of computers is then enumerated a second time (for simplicity of demonstration), and the "*autorunsc*" executable is copied to each computers' ADMIN share:



```
foreach ($computer in $computers)
{
    try
    {
        if (Test-Path "$HOME\Desktop\Tools\Autoruns\autorunsc.exe")
        {
            Copy-Item "$HOME\Desktop\Tools\Autoruns\autorunsc.exe" -Destination ("PSTriageDrive" + "$computer" + ":")
        }
    }
    catch
    {
        Write-Error "Error:  Could not copy SysInternal's Autoruns to $computer."
    }
}
```

Figure 9 - Copying Autoruns to Machines

Finally, **Invoke-Command** is used to execute "*autorunsc.exe*" on each target system:

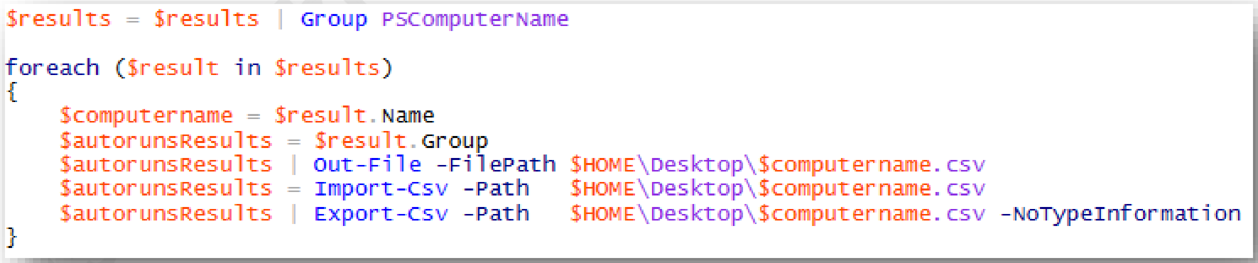


```
$results = Invoke-Command -ComputerName $computers -ScriptBlock { `
& $env:SystemRoot\autorunsc.exe -c
}
```

Figure 10 - Executing Autoruns Remotely

(*See Appendix B for an explanation the parameter/switch that is being used above, along with other useful parameter options).*

The results of "*autoruns.exe*" can now be examined for each target system. The resulting code will produce the results for each system as a .csv file.



```
$results = $results | Group PSComputerName

foreach ($result in $results)
{
    $computername = $result.Name
    $autorunsResults = $result.Group
    $autorunsResults | Out-File -FilePath $HOME\Desktop\$computername.csv
    $autorunsResults = Import-Csv -Path   $HOME\Desktop\$computername.csv
    $autorunsResults | Export-Csv -Path   $HOME\Desktop\$computername.csv -NoTypeInformation
}
```

Figure 11 - Capturing the Results in CSV Format

A resulting .csv file is created for each remote system that was instructed to execute autoruns:

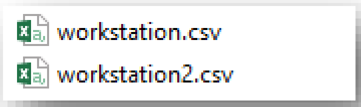
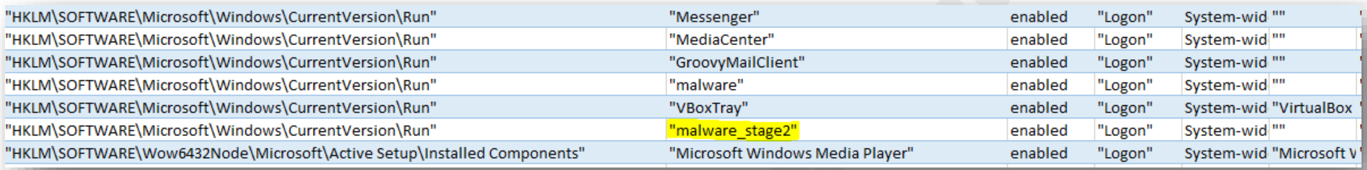

workstation.csv
workstation2.csv

Figure 12 - Resulting CSVs

Each file can be inspected to find other anomalies. Using Microsoft Excel to convert the .csv files into a table is very useful for sorting (and spotting anomalies):



"HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run"	"Messenger"	enabled	"Logon"	System-wid	""
"HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run"	"MediaCenter"	enabled	"Logon"	System-wid	""
"HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run"	"GroovyMailClient"	enabled	"Logon"	System-wid	""
"HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run"	"malware"	enabled	"Logon"	System-wid	""
"HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run"	"VBoxTray"	enabled	"Logon"	System-wid	"VirtualBox
"HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run"	"malware_stage2"	enabled	"Logon"	System-wid	""
"HKLM\SOFTWARE\Wow6432Node\Microsoft\Active Setup\Installed Components"	"Microsoft Windows Media Player"	enabled	"Logon"	System-wid	"Microsoft V

Figure 13 - Inspecting Autoruns Output

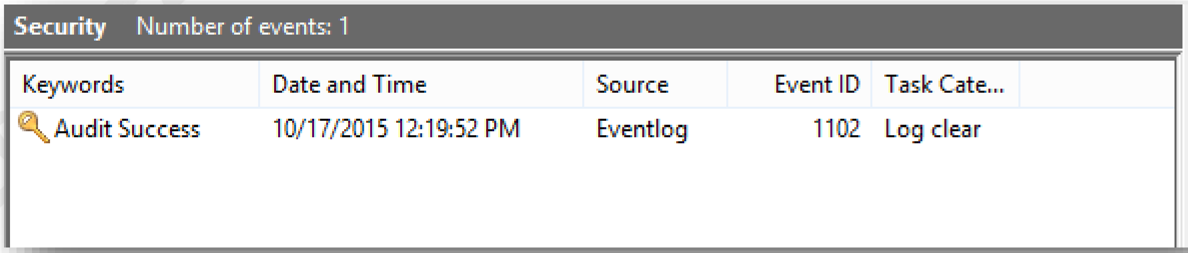
In the case, there is evidence of a stage 2 binary that has been registered on the system.

3. Looking Under the PowerShell Remoting Hood

PowerShell remoting is that simple. Once configured, it can be leveraged to collect data from multiple endpoints in parallel. However, it is important to examine what traces are left behind on the machine that is being remoted to. As a result of pulling registry values remotely, there are system modifications and new log entries that are written to disk.

This can be illustrated by clearing the security log of the workstation machine and re-running the **Get-ItemProperty** command from the analyst machine:

Prior to execution, only 1 event appears in the security log:



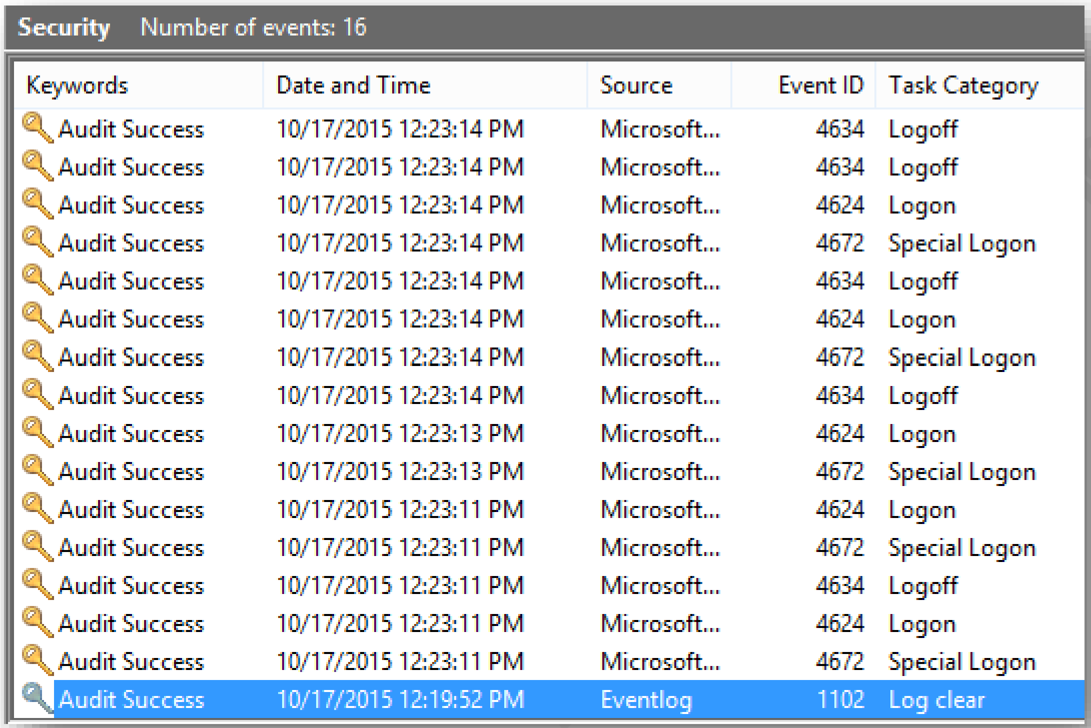
Security Number of events: 1

Keywords	Date and Time	Source	Event ID	Task Cate...
Audit Success	10/17/2015 12:19:52 PM	Eventlog	1102	Log clear

Figure 14 - Clearing the Security Log

After execution, the following events have populated into the security event log:

15 **new** entries were made as a result of executing a **single** PowerShell command remotely on the workstation machine:



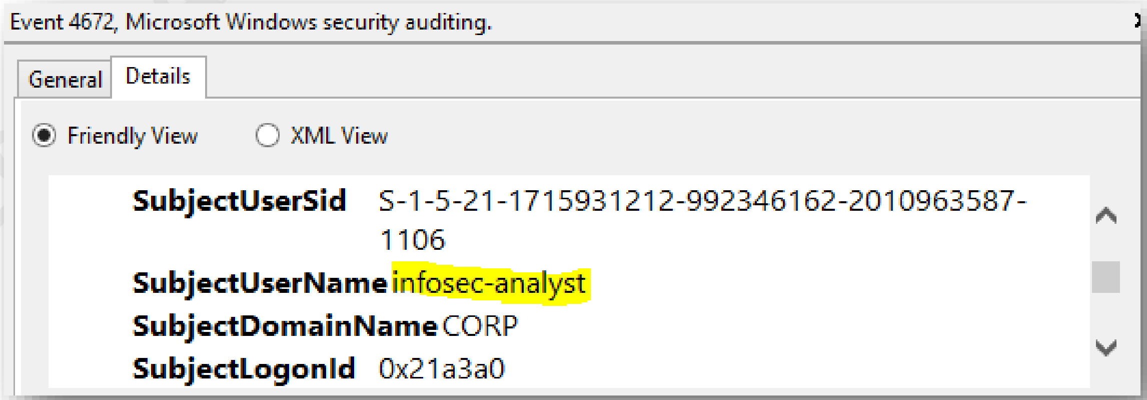
Security Number of events: 16

Keywords	Date and Time	Source	Event ID	Task Category
Audit Success	10/17/2015 12:23:14 PM	Microsoft...	4634	Logoff
Audit Success	10/17/2015 12:23:14 PM	Microsoft...	4634	Logoff
Audit Success	10/17/2015 12:23:14 PM	Microsoft...	4624	Logon
Audit Success	10/17/2015 12:23:14 PM	Microsoft...	4672	Special Logon
Audit Success	10/17/2015 12:23:14 PM	Microsoft...	4634	Logoff
Audit Success	10/17/2015 12:23:14 PM	Microsoft...	4624	Logon
Audit Success	10/17/2015 12:23:14 PM	Microsoft...	4672	Special Logon
Audit Success	10/17/2015 12:23:14 PM	Microsoft...	4634	Logoff
Audit Success	10/17/2015 12:23:13 PM	Microsoft...	4624	Logon
Audit Success	10/17/2015 12:23:13 PM	Microsoft...	4672	Special Logon
Audit Success	10/17/2015 12:23:11 PM	Microsoft...	4624	Logon
Audit Success	10/17/2015 12:23:11 PM	Microsoft...	4672	Special Logon
Audit Success	10/17/2015 12:23:11 PM	Microsoft...	4634	Logoff
Audit Success	10/17/2015 12:23:11 PM	Microsoft...	4624	Logon
Audit Success	10/17/2015 12:23:11 PM	Microsoft...	4672	Special Logon
Audit Success	10/17/2015 12:19:52 PM	Eventlog	1102	Log clear

Figure 15 – Newly Generated Security Events

Specifically, there are **5** logon events (event ID 4624), **5** logoff events (event ID 4634), and **5** special logon events (event ID 4672); all of which occurred within a 2 second time frame.

Examining one of the special logon events reveals some intelligence about who instantiated the logons:



Event 4672, Microsoft Windows security auditing.

SubjectUserSid S-1-5-21-1715931212-992346162-2010963587-1106
SubjectUserName infosec-analyst
SubjectDomainName CORP
SubjectLogonId 0x21a3a0

Figure 16 - Inspecting the Logon Event (4672)

Additionally, a user profile folder is also created on machines that receive PowerShell remoting commands. This also leaves remnants behind that are indicative of what privileged account was used to invoke the remote commands:

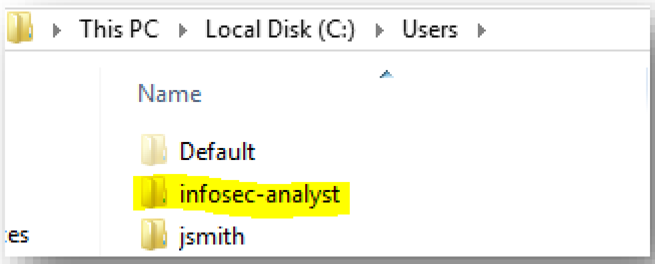


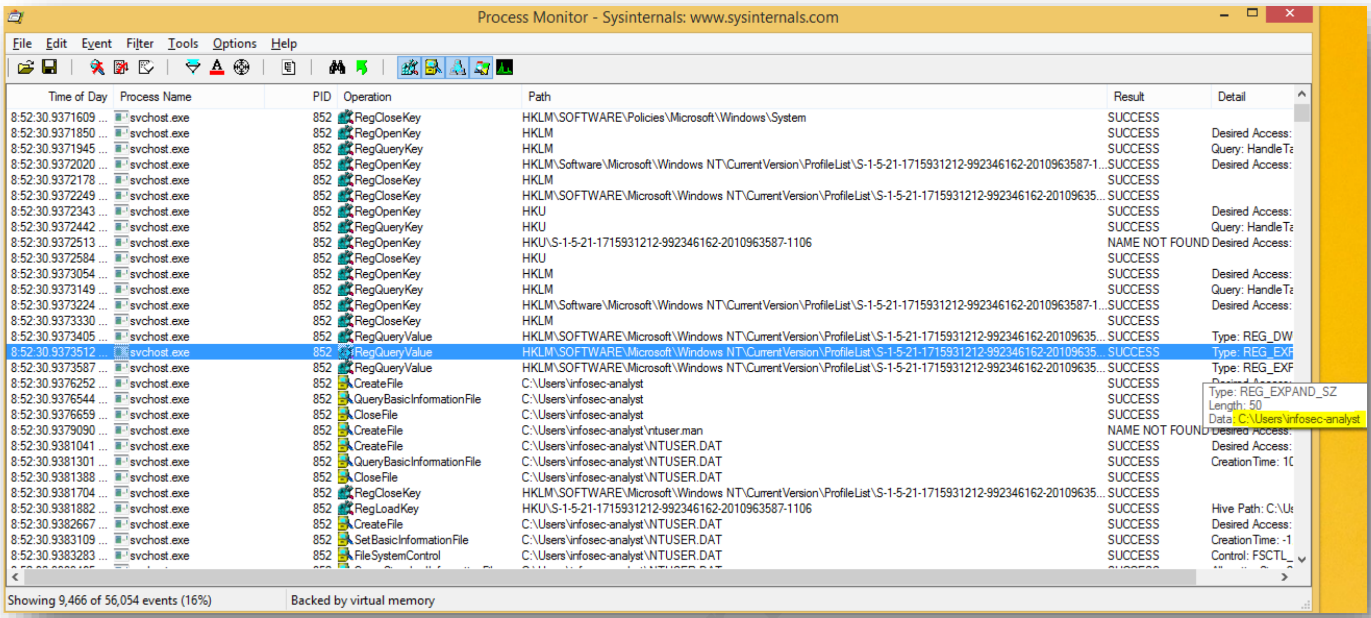
Figure 17 - Newly Created Users Folder

As a result of remoting to the "workstation" machine, a new folder was created within the system's "C: \ Users" directory, conveniently sharing the same name as the security analyst's privileged domain account.

Other modifications to disk can be captured live by using the **Process Monitor** utility. Process Monitor is Windows monitoring application that shows live information about a system's file system, registry, and networking.

Process Monitor can be used to show system modifications on a system that is currently receiving remote PowerShell commands.

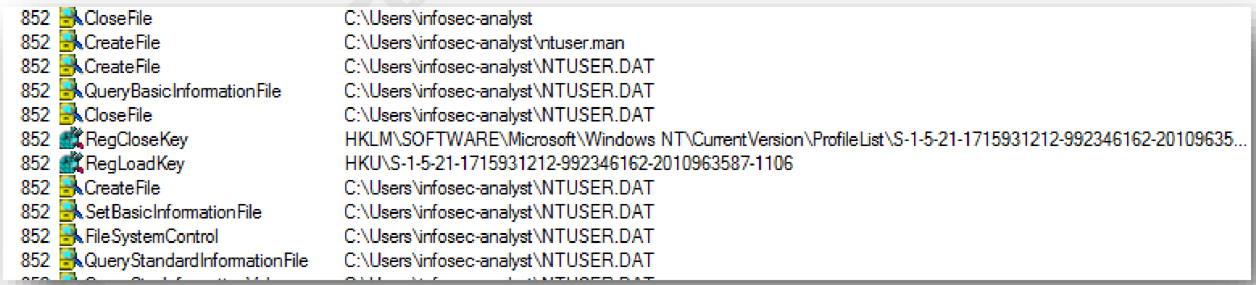
Process Monitor is used to capture system changes on the receiving machine:



```
Process Monitor - Sysinternals: www.sysinternals.com
File  Edit  Event  Filter  Tools  Options  Help

Time of Day      Process Name   PID  Operation                  Path                                                                                              Result           Detail
8:52:30.9371609 ... svchost.exe 852  RegCloseKey                HKLM\SOFTWARE\Policies\Microsoft\Windows\System                                                  SUCCESS
8:52:30.9371850 ... svchost.exe 852  RegOpenKey                 HKLM                                                                                              SUCCESS          Desired Access:
8:52:30.9371945 ... svchost.exe 852  RegQueryKey                HKLM                                                                                              SUCCESS          Query: HandleTa
8:52:30.9372020 ... svchost.exe 852  RegOpenKey                 HKLM\Software\Microsoft\Windows NT\CurrentVersion\ProfileList\S-1-5-21-1715931212-992346162-2010963587-1...SUCCESS  Desired Access:
8:52:30.9372178 ... svchost.exe 852  RegCloseKey                HKLM                                                                                              SUCCESS
8:52:30.9372249 ... svchost.exe 852  RegCloseKey                HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProfileList\S-1-5-21-1715931212-992346162-20109635... SUCCESS
8:52:30.9372343 ... svchost.exe 852  RegOpenKey                 HKU                                                                                               SUCCESS          Desired Access:
8:52:30.9372442 ... svchost.exe 852  RegQueryKey                HKU                                                                                               SUCCESS          Query: HandleTa
8:52:30.9372513 ... svchost.exe 852  RegOpenKey                 HKU\S-1-5-21-1715931212-992346162-2010963587-1106                                                 NAME NOT FOUND   Desired Access:
8:52:30.9372584 ... svchost.exe 852  RegCloseKey                HKU                                                                                               SUCCESS
8:52:30.9373054 ... svchost.exe 852  RegOpenKey                 HKLM                                                                                              SUCCESS          Desired Access:
8:52:30.9373149 ... svchost.exe 852  RegQueryKey                HKLM                                                                                              SUCCESS          Query: HandleTa
8:52:30.9373224 ... svchost.exe 852  RegOpenKey                 HKLM\Software\Microsoft\Windows NT\CurrentVersion\ProfileList\S-1-5-21-1715931212-992346162-2010963587-1...SUCCESS  Desired Access:
8:52:30.9373330 ... svchost.exe 852  RegCloseKey                HKLM                                                                                              SUCCESS
8:52:30.9373405 ... svchost.exe 852  RegQueryValue              HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProfileList\S-1-5-21-1715931212-992346162-20109635... SUCCESS          Type: REG_DW
8:52:30.9373512 ... svchost.exe 852  RegQueryValue              HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProfileList\S-1-5-21-1715931212-992346162-20109635... SUCCESS          Type: REG_EXF
8:52:30.9373587 ... svchost.exe 852  RegQueryValue              HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProfileList\S-1-5-21-1715931212-992346162-20109635... SUCCESS          Type: REG_EXF
8:52:30.9376252 ... svchost.exe 852  CreateFile                 C:\Users\infosec-analyst                                                                          SUCCESS
8:52:30.9376544 ... svchost.exe 852  QueryBasicInformationFile  C:\Users\infosec-analyst                                                                          SUCCESS
8:52:30.9376659 ... svchost.exe 852  CloseFile                  C:\Users\infosec-analyst                                                                          SUCCESS
8:52:30.9379090 ... svchost.exe 852  CreateFile                 C:\Users\infosec-analyst\ntuser.man                                                               NAME NOT FOUND   Desired Access:
8:52:30.9381041 ... svchost.exe 852  CreateFile                 C:\Users\infosec-analyst\NTUSER.DAT                                                               SUCCESS          Desired Access:
8:52:30.9381301 ... svchost.exe 852  QueryBasicInformationFile  C:\Users\infosec-analyst\NTUSER.DAT                                                               SUCCESS          CreationTime: 10
8:52:30.9381388 ... svchost.exe 852  CloseFile                  C:\Users\infosec-analyst\NTUSER.DAT                                                               SUCCESS
8:52:30.9381704 ... svchost.exe 852  RegCloseKey                HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProfileList\S-1-5-21-1715931212-992346162-20109635... SUCCESS
8:52:30.9381882 ... svchost.exe 852  RegLoadKey                 HKU\S-1-5-21-1715931212-992346162-2010963587-1106                                                 SUCCESS          Hive Path: C:\Us
8:52:30.9382667 ... svchost.exe 852  CreateFile                 C:\Users\infosec-analyst\NTUSER.DAT                                                               SUCCESS          Desired Access:
8:52:30.9383109 ... svchost.exe 852  SetBasicInformationFile    C:\Users\infosec-analyst\NTUSER.DAT                                                               SUCCESS          CreationTime: -1
8:52:30.9383283 ... svchost.exe 852  FileSystemControl          C:\Users\infosec-analyst\NTUSER.DAT                                                               SUCCESS          Control: FSCTL_

Type: REG_EXPAND_SZ
Length: 50
Data: C:\Users\infosec-analyst

Showing 9,466 of 56,054 events (16%)      Backed by virtual memory
```

Figure 18 - SysInternal's Process Monitor

There are numerous modifications that are related to the PowerShell remoting activity. The activity is easy to identify by using Process Monitor's find capability to search for "infosec-analyst", the name of the account that issued the remote commands. Furthermore, filtering on the process ID (PID) associated with the searched criteria yields **1,121** system changes. For example:



```
852  CloseFile                  C:\Users\infosec-analyst
852  CreateFile                 C:\Users\infosec-analyst\ntuser.man
852  CreateFile                 C:\Users\infosec-analyst\NTUSER.DAT
852  QueryBasicInformationFile  C:\Users\infosec-analyst\NTUSER.DAT
852  CloseFile                  C:\Users\infosec-analyst\NTUSER.DAT
852  RegCloseKey                HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProfileList\S-1-5-21-1715931212-992346162-20109635...
852  RegLoadKey                 HKU\S-1-5-21-1715931212-992346162-2010963587-1106
852  CreateFile                 C:\Users\infosec-analyst\NTUSER.DAT
852  SetBasicInformationFile    C:\Users\infosec-analyst\NTUSER.DAT
852  FileSystemControl          C:\Users\infosec-analyst\NTUSER.DAT
852  QueryStandardInformationFile C:\Users\infosec-analyst\NTUSER.DAT
```

Figure 19 - Process Monitor Output

3.1 Commodity vs. APT

Leveraging PowerShell during a response can be extremely beneficial for both the security and performance gains outlined thus far. However, there are nuances that need to be seriously considered before making the decisions to enable PowerShell remoting and before creating, or leveraging pre-existing PowerShell frameworks. There are two scenarios that illustrate the need for deliberate decision making as it relates to this topic.

Imagine a scenario where a SOC aggregates, prioritizes, and responds to anti-malware alerts. The team finds that most of the time these alerts correspond to commodity malware (adware, bundled software, ransomware, dated Trojans, etc.), and spends little time making a determination between false positive (FP) and true positive (TP). The threat is relatively low, and the team feels confident in leveraging their PowerShell IR framework to look for evidence to support the anti-virus' claim.

On the other hand, there is a less-common scenario that occurs when the SOC receives a report of a potential advanced persistent threat (APT). This type of threat is on the opposite end of the spectrum, and represents a potential adversary that is more mature, sophisticated in their methodologies, and possibly targeting the team's organization specifically.

Should the team feel the same about using their PowerShell IR tool to retrieve data from hosts that are suspected to be involved in this APT-related intelligence report?

PowerShell remoting IR tools are great for protecting credentials and at offering scalable, performant solutions, but they may not be the best choice when responding to advanced adversaries. When a set of commands are executed remotely via PowerShell, there are several changes made to disk that represent the activity that was just executed. These changes modify the integrity of disk, and can also provide attribution to an adversary that may be active on a compromised host.

3.2 The Cause and Effects of Response Efforts

It is clear that PowerShell's remoting technology can be leveraged to assist in incident response efforts. It is also clear that instantiating remote PowerShell sessions is rather noisy, resulting in modification to the target's disk. It is important to understand that using PowerShell during incident response is considered *active* response. It is also imperative to understand passive versus active response, counterintelligence, and the overall risk decisions that need to be made before engaging.

An interview was conducted with Microsoft's Senior Security Analyst, Brian Hooper, who has an extensive background in computer security and forensics. Prior to his current role at Microsoft, Brian served as the Lead Instructor and Content Developer for security training at Lockheed Martin. Specifically, Brian taught on the topics of Intelligence-Driven Defense and the Cyber Kill Chain.

The interview was introduced with the following question, "**Why is attribution important before initiating incident response efforts?**"

It was immediately made clear that "attribution" was not the most suitable word for the question, but instead, defenders should look for *any* information that can serve as tips as to what is happening on the network. This information could simply stem from a network-based intrusion detection system (IDS) alert, or a non-anomaly on the host (such as an application crash) (Hooper, 2015).

Brian noted that defenders should ask themselves, "What am I dealing with here?" "How bad do I really think it is?" (Hooper, 2015)

Understanding where the criticality falls on the "bad guy scale" is crucial. This information is then used to determine what type of response is appropriate: active or passive. And remember, PowerShell remoting is most definitely active.

The interview continued down this path when the following question was posed, **"Can you describe the risk decision process around whether passive or active response is more appropriate?"**

Brian explained that security defenders should frame the risk decision with the following question. "If there are truly indications of a bad guy on a box, how would you treat the situation differently?" The bottom line is that active response should be carried out with caution. Even something like nbtstat, the netBIOS equivalent of a ping, touches the box. Admittedly, the chances of an advanced adversary doing anything that would draw awareness to a nbtstat, or a ping, is extremely low. However, the main point is that defenders need to be knowledgeable about their toolsets, and need to understand the cause-and-effect of each one (Hooper, 2015).

Conversely, active response is not limited to "touching the box". Brian underscored this point with an example: A security analyst has acquired a filename IOC and wants to derive related intelligence. The analyst accomplishes this by performing a Google search on the file name. Brian then asked, "Is the search passive or active?"

Brian went on to explain that the bad guys can purchase ad keywords on the Google search engine that relate to their malware. Consequently, the security analyst in this scenario has now fed intelligence back to the adversary by performing the Google search. Data derived from web searches is fed back to adversary via Google AdSense. (Hooper, 2015).

The act of feeding intelligence back to an adversary during response prompted the next question.

"Can you describe/speak to counter intelligence?"

Counter intelligence boils down to thinking about each and every action when dealing with a potential advanced adversary and understanding what information may be tipped off. There is no perfect solution that will work for every organization's response team. The reality is that deciding the appropriate response action needs to be treated as a risk decision.

Brian highlights this fact with another scenario. If a response team identifies a compromised host with the presence of an active adversary, which of the following actions are more appropriate? Should the response team immediately isolate the box by taking it offline? This is typically a sound plan for teams that have the forensic capability. Or should the response team keep the box online in order to perform live triage and to potentially gain additional intelligence? (Hooper, 2015)

Taking a box offline can potentially tip off the adversary. The adversary will more than likely know that some response has been initiated when this approach is taken. Also, taking the machine offline also prevents the defenders from possibly learning more about the adversary. Brian states, "If you learn more about him [the adversary] today, there's a better chance to defend against him in the future [when he comes back]." (Hooper, 2015)

The final question was asked, "**What does advance adversary response look like?"**

Brian stated, "If you truly think it's advanced adversary, the white gloves come on. You have to be incredibly cautious about everything you do. If it looks like APT, take it offline [in most cases]." (Hooper, 2015)

The bottom line is that there is no one-size-fits-all solution for every organization. Every organization has a unique footprint and is equipped with varying capabilities. The key is to use the best tool to achieve the goal, and to understand the cause-and-effect of each and every action during response.

Conclusion

PowerShell is currently shaping the threat landscape and influencing the way defenders respond. Malware authors are incorporating PowerShell into their weaponization techniques, and security professionals are using the very same technology to perform incident response. From the defender's perspective, PowerShell offers a secure, scalable, and efficient IR solution. However, it is important to understand that PowerShell-based security frameworks should not be considered "one-size-fits-all" security tools.

The use of PowerShell remoting, which is often the key feature to emerging PowerShell response frameworks, is a method of active response. The risk decision process that depicts the appropriate level of response for a given incident needs to be taken seriously before leveraging such PowerShell frameworks. Invoking remote PowerShell sessions to retrieve evidence can cause great modification to a host disk, which can potentially disrupt the integrity of forensic evidence.

Counterintelligence also influences the risk decisions that defenders need to make before engaging in response. It is important to ask, "What are the effects of this tool?" "What will be the result of running this command?" "Will this particular activity leak any information back to the adversary?"

PowerShell is just as the name implies—a powerful scripting language that is starting to gain increased popularity within the security community. But with any tool, the power that it offers needs to be understood, and most certainly, not abused. Unfortunately, even a well-meaning individual can cause harm with this tool if their level of understanding is not sufficient. The key component is thorough understanding. Once a user truly learns the intricacies of this incredible tool, it will provide them with the computing muscle they need to achieve and protect their goals.

Bibliography

Graeber, M., Bialek, J., Campbell, C., Lundeen, R., Atkinson, J., & Hajduk, B. (2015, September 30). *PowerSploit - A PowerShell Post-Exploitation Framework*. Retrieved from GitHub: https://github.com/PowerShellMafia/PowerSploit

Hofferle, J. (2012, July 23). *An Introduction to PowerShell Remoting: Part One*. Retrieved from Hey, Scripting Guy! Blog: http://blogs.technet.com/b/heyscriptingguy/archive/2012/07/23/an-introduction-to-powershell-remoting-part-one.aspx

Holmes, L. (2015, June 9). *PowerShell ♥ the Blue Team*. Retrieved from Windows PowerShell Blog: http://blogs.msdn.com/b/powershell/archive/2015/06/09/powershell-the-blue-team.aspx

Hooper, B. (2015, 11 9). Understanding Active vs. Passive Incident Response; a Look into Counter Intelligence. (R. Adams, Interviewer)

Hull, D. (2014, July 18). Kansa: A PowerShell-Based Incident Response Framework.

Menrige, M. (2014, May 29). *Black Magic: Windows PowerShell Used Again in New Attack*. Retrieved from TrendMicro Blog: http://blog.trendmicro.com/trendlabs-security-intelligence/black-magic-windows-powershell-used-again-in-new-attack/

Microsoft Developer Network. (2015, August 9). *Enable-PSRemoting*. Retrieved from Microsoft Developer Network: https://technet.microsoft.com/en-us/library/hh849694.aspx

Rouse, M. (2007, April). *Polymorphic Malware Definition*. Retrieved from TechTarget SearchSecurity: http://searchsecurity.techtarget.com/definition/polymorphic-malware

Russinovich, M. (2015, October 26). *Autoruns for Windows v13.5*. Retrieved from Windows Sysinternals: https://technet.microsoft.com/en-us/sysinternals/bb963902.aspx

Scholer, B. (2012, September 16). *Enable Powershell Remoting via Group Policy*. Retrieved from Briantist: http://www.briantist.com/how-to/powershell-remoting-group-policy/

Trend Micro. (2015). Banking Malware VAWTRAK Now Uses Malicious Macros, Abuses Windows PowerShell. *TrendLabs Security Intelligence Blog*.

Appendix A – PowerShell Primer

PowerShell has been influencing the security community, and shaping the threat landscape. There are a handful of defender tools built in PowerShell, but at the same time, there have been legitimate attacks written in PowerShell. For example, the VAWTRAK banking malware leverages PowerShell in its attack vector. The malware makes its way onto victims' computers via phishing e-mails, which then prompt users to open up a seemingly innocent Microsoft Office attachment. Once opened, Visual Basic (VB) macros are used to drop a PowerShell script (Trend Micro, 2015).

Although PowerShell can be leveraged for malicious intent, it can also be leveraged to greatly enhance security operations and incident response efforts. PowerShell's remoting technology allows security professionals to securely connect to other systems in their environment in order to retrieve critical pieces of data during triage. PowerShell remoting leverages TCP for communication and Kerberos for authentication. An enterprise can easily be configured for PowerShell remoting with Group Policy.

PowerShell remoting also protects important credentials. This is achieved because the act of instantiating a remote PowerShell connection does not create a delegation token, therefore, preventing the exposure of the credential's hash (Pilkington, 2013). However, specific use-cases are more appropriate for using PowerShell's remoting than others. Credentials may be protected, but executing commands remotely inevitably generates new artifacts on the target host. Would this be ideal when performing incident response functions against a potential APT actor?

What is PowerShell Remoting?

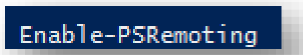
PowerShell comes with its own remoting technology. This feature essentially allows users to execute commands on remote systems in a secure fashion. The remoting technology is built upon the WSMan protocol, which is Microsoft's implementation of the Web Services for Management. Additionally, PowerShell remoting uses Windows Remote Management Service (WinRM) for the communication and security piece (Hofferle, 2012).

PowerShell remoting, PSRemoting, really shows its benefits when used at-scale. The machines that are the recipient of commands handle all of the processing, preventing overhead concerns on the system issuing the commands. For example, PSRemoting allows a system administrator to retrieve the status of specific services and processes that are running on *hundreds* of machines in a matter of minutes.

Types of Remoting

How exactly is PowerShell remoting enabled? There are essentially two options. Firstly, a system can be enabled for remoting by simply leveraging the **Enable-PSRemoting** command.

Robert Adams | ROBERT.LOUIS.ADAMS@GMAIL.COM



```
Enable-PSRemoting
```

Figure 20 - Enable-PSRemoting

Issuing this command as an administrator will make all the necessary changes to allow that system to be a receiving endpoint for commands that are issued remotely. What exactly happens to the system when PowerShell remoting is enabled?

The **Enable-PSRemoting** command makes use of another PowerShell command, **Set-WSManQuickConfig**; which performs the following:

- Starts the WinRM service
- Sets the startup type on the WinRM service to 'Automatic'
- Creates a listener to accept requests on any IP address
- Enables a firewall exception for WS-Management communications
- Registers the Microsoft.PowerShell and Microsoft.PowerShell.Workflow session configurations
- Registers the Microsoft.PowerShell32 session configuration on 64-bit computers
- Enables all session configurations
- Changes the security descriptor of all session configurations to allow remote access
- Restarts the WinRM service to make the preceding changes effective

It is important to note that on Windows Server 2012 and greater, PowerShell remoting is enabled by default. This means that only client machines will need to be configured for PowerShell remoting (Microsoft Developer Network, 2015).

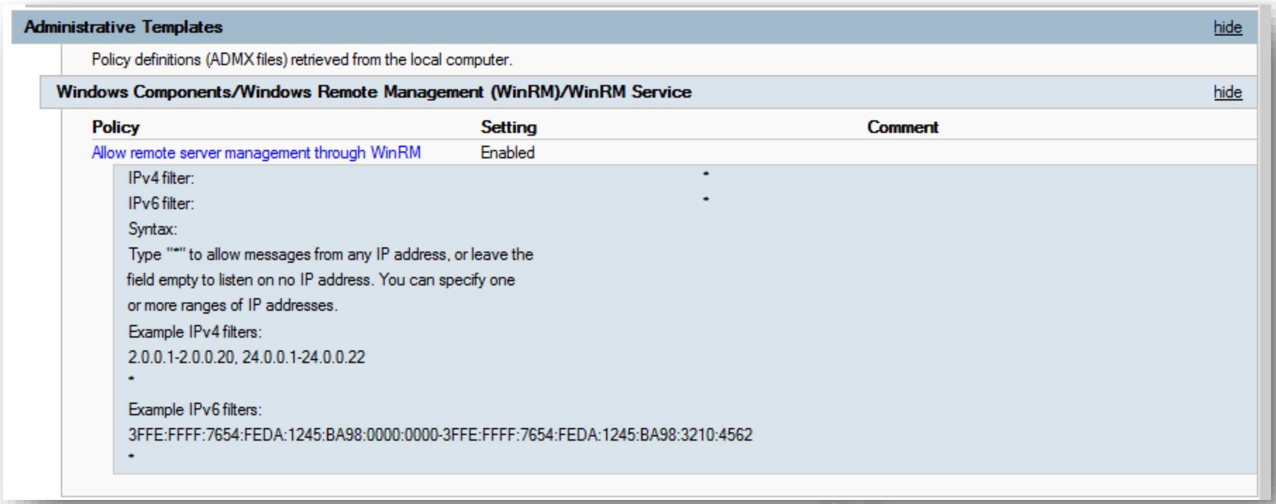
In a scenario where *many* hosts will need to be configured to receive commands via PSRemoting, the Enable-PSRemoting cmdlet is not a scalable solution. Instead, leveraging Microsoft Group Policy should be considered.

There are templates that can be used to easily configure PowerShell remoting via Group Policy. The two templates are:

1. Policies -> Administrative Templates -> Windows Components -> Windows Remote Management (WinRM) -> **WinRM Service**
2. Policies -> Administrative Templates -> Network -> Network Connections -> Windows Firewall -> **Domain Profile**

These configuration settings need to be configured to allow remote server management through WinRM, and to allow automatic configuration listeners on specified IP addresses. Additionally, the firewall configuration needs to be configured to allow inbound TCP traffic on port 5985 (Scholer, 2012).

The following configuration settings are illustrated using the Windows Group Policy editor:



Administrative Templates — hide

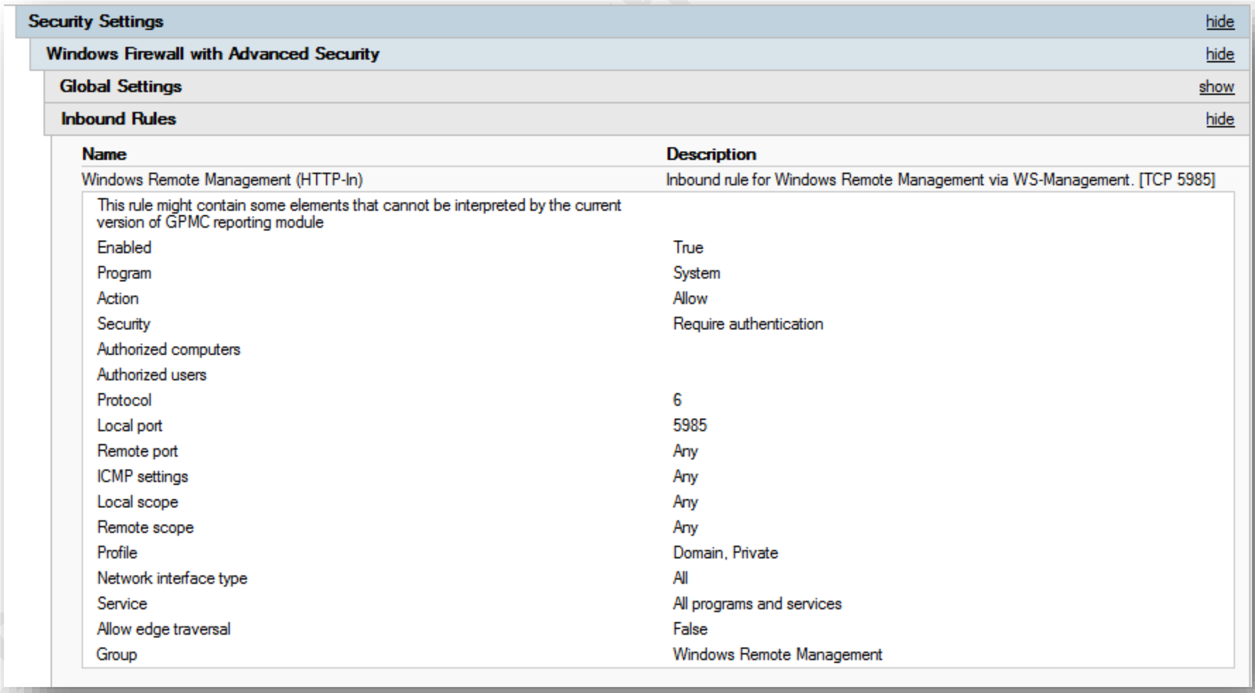
Policy definitions (ADMX files) retrieved from the local computer.

Windows Components/Windows Remote Management (WinRM)/WinRM Service — hide

Policy	Setting	Comment
Allow remote server management through WinRM	Enabled	

```
IPv4 filter:                                   *
IPv6 filter:                                   *
Syntax:
Type "*" to allow messages from any IP address, or leave the
field empty to listen on no IP address. You can specify one
or more ranges of IP addresses.
Example IPv4 filters:
2.0.0.1-2.0.0.20, 24.0.0.1-24.0.0.22
*
Example IPv6 filters:
3FFE:FFFF:7654:FEDA:1245:BA98:0000:0000-3FFE:FFFF:7654:FEDA:1245:BA98:3210:4562
*
```

Figure 21 - Windows GPO Editor



Security Settings — hide

Windows Firewall with Advanced Security — hide

Global Settings — show

Inbound Rules — hide

Name	Description
Windows Remote Management (HTTP-In)	Inbound rule for Windows Remote Management via WS-Management. [TCP 5985]

This rule might contain some elements that cannot be interpreted by the current version of GPMC reporting module

Enabled	True
Program	System
Action	Allow
Security	Require authentication
Authorized computers	
Authorized users	
Protocol	6
Local port	5985
Remote port	Any
ICMP settings	Any
Local scope	Any
Remote scope	Any
Profile	Domain, Private
Network interface type	All
Service	All programs and services
Allow edge traversal	False
Group	Windows Remote Management

Figure 22 - WinRM Firewall Rule

Once the GPO is properly configured, it can be attached to a policy and applied across a single, or multiple domains.

Enabling PowerShell remoting across an environment should be approached systematically. For example, the GPO should be tested and applied to a sub group first. A collection of test computers can be added to a new security group, and then the test GPO can be applied to the newly created SG.

It is also important for management to understand the business impact and justification of leveraging PowerShell remoting across the environment. This justification can also be leveraged when taking the request through the organization's change request process, or change advisory board.

Frameworks that Leverage PowerShell Remoting

There is a clear and upward trend in PowerShell remoting. More and more security frameworks, written entirely in PowerShell, with the use of its remoting technology, are appearing within the security community. This is sign that the community at large acknowledges the performance and security value-add of PowerShell, and are making deliberate decisions to incorporate the tool and technology into common workflows.

Kansa is an incident response frame work that leverages PowerShell to perform enterprise-wide response. It was developed by Dave Hull, who at the time, was a security analyst at Microsoft. The primary intent of Kansa is to enable security professionals to collect data from numerous endpoints in an expedient fashion. This is achieved through the use of PowerShell remoting and the security it provides: Kerberos authentication in a way that protects elevated credentials.

Kansa is an open-source, modular framework that allows easy customization and extensibility. The tool is equipped with several modules out-of-the-box, which empowers security analysts to retrieve and correlate data across several hosts. For example, Kansa will return information about the systems' local administrators group, networking data, hotfix installation and dates, system and security logs, and much more. Kansa can also reference third-party binaries and execute those on target systems, under the umbrella of the secure PowerShell remoting framework. The only caveat is that the binary has to be copied to the target's machine (Hull, 2014).

Kansa is an excellent example of what a Security Operations Center (SOC) can create to leverage the utility of PowerShell remoting; once it has been enabled through GPO, for example. There are other frameworks that are gaining popularity as well. Other tools include Invoke-IR, PSRecon, and PowerShell Empire. These are other examples of PowerShell tooling that makes great use of PSRemoting.

See Appendix C for links to these other tools.

How Attackers Leverage PowerShell

There is an abundance of benefits for using PowerShell for defense. Interestingly, PowerShell's many attractive characteristics also appeal to attackers. Security analysts at TrendMicro have identified and researched several threat actors using PowerShell. The associated malware is often delivered via an e-mail attachment, and once downloaded, PowerShell is used to bypass execution policies and to retrieve additional payloads. In one case, the PowerShell-based malware captured critical system data and sought to steal user passwords (Menrige, 2014).

PowerShell has also become a crucial toolset for penetration testers. PowerSploit is a collection of PowerShell modules that have been developed specifically for pen testers. There are modules for every phase of an assessment, and some noteworthy features include remote DLL injection, persistency capabilities, anti-virus bypassing, and exfiltration (Graeber, et al., 2015).

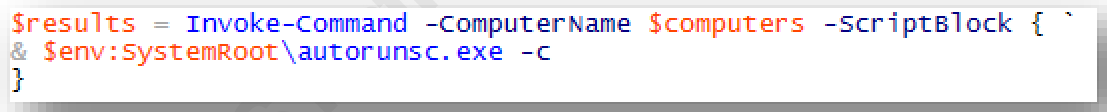
There is a slew of new PowerShell (version 5) security features that are designed to give the defenders a way to combat these attack vectors.

Starting with PowerShell version 4, users have the option of turning on a granular level of logging, called **transcriptions** (which is facilitated through the "Start-Transcript" command). PowerShell version 5 has enhanced this feature by allowing transcripts to capture remote sessions and other non-interactive uses. Better yet, the transcription feature can be enabled with GPO (Holmes, 2015).

Deep Script Block Logging is another security-centric feature that can be configured to provide invaluable insights into what PowerShell is doing on a system. Microsoft's Lee Holmes describes it as, "When you enable script block logging, PowerShell records the content of all script blocks that it processes." "This provides complete insight into the script-based activity on a system – including scripts or applications that leverage dynamic code generation in an attempt to evade detection." (Holmes, 2015)

Finally, there is a new feature called **Antimalware Scan Interface Integration** (AMSI). Essentially, the new version of PowerShell will automatically interact with the client's anti-virus engine to double check PowerShell script activity against known definitions. This ultimately helps by providing an additional layer of protection against a variety of attacks (Holmes, 2015).

Appendix B – Autorunsc.exe Command Line Switches (from example)



```
$results = Invoke-Command -ComputerName $computers -ScriptBlock { `
& $env:SystemRoot\autorunsc.exe -c
}
```

Command Line Switch/Parameter	Description
-c	Print output as csv.
-v [rs]	Queries Virus Total for malware matches based on file hash. Extremely useful feature for easily spotting malware.
-vt	Accepts the Terms of Service for using Virus Total.
-a *	Specifies all autostart entry points.
b	Boot execute.
h	Image hijacks.
t	Scheduled tasks.
-s	Verify digital signatures.

(Russinovich, 2015).

Appendix C – Other PowerShell Remoting Tools

Tool	Reference Link
Invoke-IR	http://www.invoke-ir.com/
PSRecon	https://blog.logrhythm.com/digital-forensics/psrecon/
PowerShell Empire	http://www.powershellempire.com/

Robert Adams | ROBERT.LOUIS.ADAMS@GMAIL.COM