

PowerShell for Forensics

by Washington Almeida

Organizations today handle more sensitive personal data than ever before. As the amount of sensitive personal data increases, the more they are susceptible to security incidents and breaches. One of the biggest challenges in Incident Response nowadays is in the incident detection phase. There are several tools available to help Forensics specialists on working into detection phase. Some of them are open-source and some of them are commercial tools. However, the Windows OS environment has a built-in tool that can support the forensic activities in live response: PowerShell.

PowerShell is an object-oriented automation engine and scripting language with an interactive command-line shell developed by Microsoft to help IT professionals in configuring systems and automating administrative tasks. Quite similar to the Perl language that has been introduced in Linux, Microsoft designed PowerShell to automate system tasks, such as batch processing, and to create systems management tools for commonly implemented processes in the MS Windows environment.

PowerShell language was developed by a Microsoft technical partner named Jeffrey Snover. While working on system administration tasks on Windows, Snover felt Windows lacked the power and flexibility to automate tasks, which was already available in the Unix environment. Thus, Snover developed the idea to create a standardized platform that used the .NET framework through objects and automation tasks.

Microsoft produced a new command-line interface along with a shell called Monad which the first beta was released on June 2005. In April 2006, Microsoft renamed Monad to Windows PowerShell and made it a core part of the Windows operating system environment. Microsoft released PowerShell 1.0 in the end of 2006 for Windows XP SP2, Windows Vista and Windows Server 2003 SP1. With Windows Server 2008, Microsoft offered PowerShell as an optional component. PowerShell 2.0 was released with Windows 7 and Windows Server 2008 R2. PowerShell 3.0 was released in 2012 as part of Windows 8/Server 2012. It completed the core language features and started to ship the ISE Editor. PowerShell 4.0 was released in 2013 as part of Windows 8.1/Server 2012R2. It added DSC (Desired State Configuration). PowerShell 5.0 was released in 2015 as part of Windows 10. This version is a major milestone with various new features and language extensions. Finally, PowerShell 5.1 was released in 2016 as part of Windows 10 Anniversary Update and Server 2016.

PowerShell commands or Cmdlets are based on .NET Framework objects, which mean that the objects carry multiple aspects or properties of the command. These Cmdlets let you access the file system and other Windows operating system data stores, such as the registry. PowerShell also provides access to Windows Management Instrumentation (WMI), which means that all the WMI commands that incident responders and information security professionals are familiar with can be run using PowerShell. Complete documentation about PowerShell can be found in the Microsoft URI at <https://docs.microsoft.com/en-us/powershell>.

PowerShell updates:

PowerShell updates are free and come as “.msu” packages, so they can be distributed just like any other Windows update. If you run an outdated version of PowerShell, you’re not just missing features, it’s also a potential security risk.

Understanding PowerShell help:

Later we will go into the details of PowerShell environment, but before you get started using PowerShell, you need to understand the syntax of its parameters, such as -ComputerName and -Class. Thus, for a better understanding of how PowerShell works, you can investigate the master cmdlet Get-

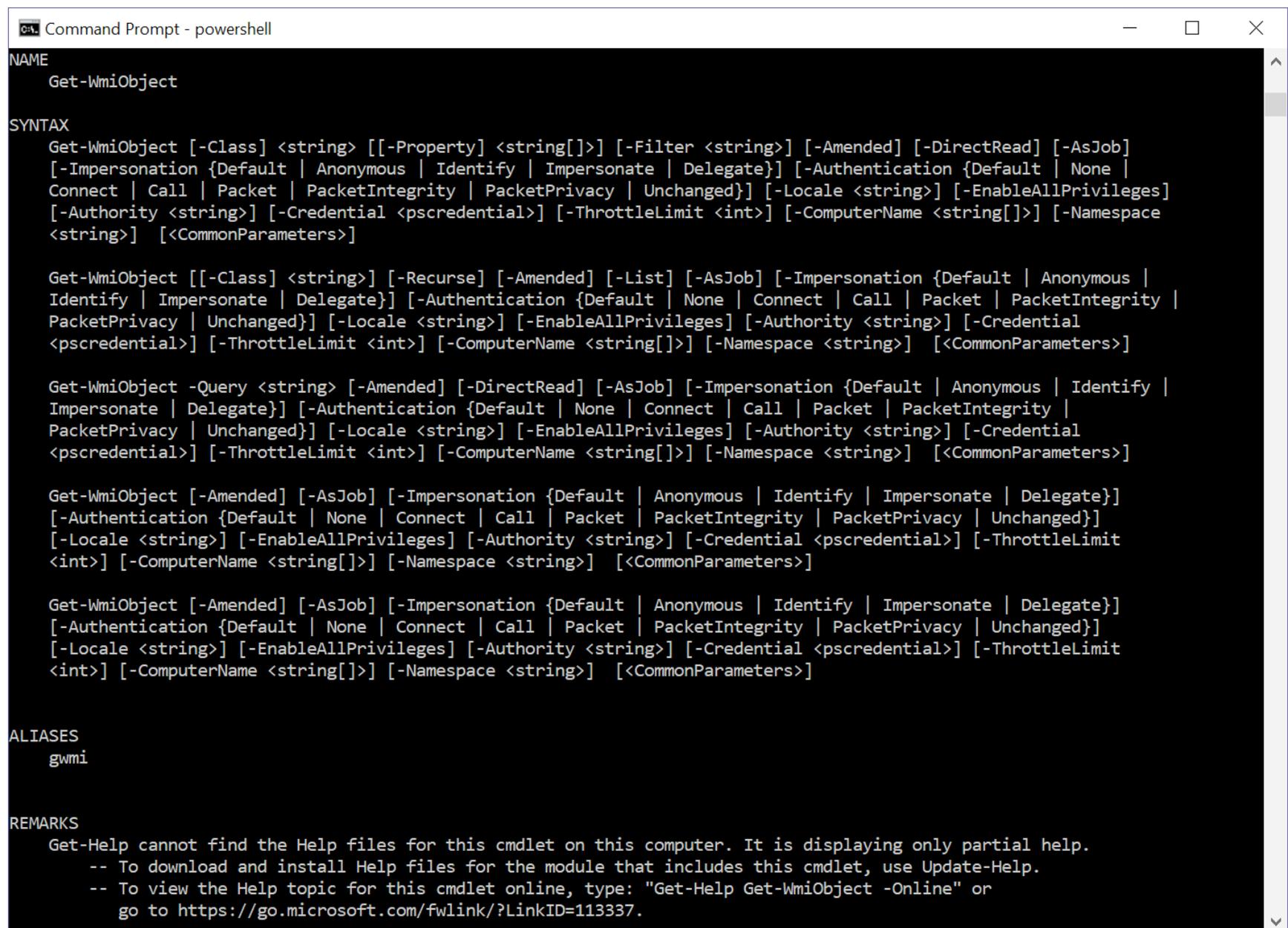
WmiObject (and all others) calling the help of it by Get-Help. To do this you can call the help for the cmdlet Get-WmiObject as follows:

```
C:\Users\Wash>powershell
```

```
Windows PowerShell
```

```
Copyright (C) 2016 Microsoft Corporation. All rights reserved.
```

```
PS C:\Users\Wash> Get-Help Get-WmiObject
```



```

Command Prompt - powershell
NAME
    Get-WmiObject

SYNTAX
    Get-WmiObject [-Class] <string> [[-Property] <string[]>] [-Filter <string>] [-Amended] [-DirectRead] [-AsJob]
    [-Impersonation {Default | Anonymous | Identify | Impersonate | Delegate}] [-Authentication {Default | None |
    Connect | Call | Packet | PacketIntegrity | PacketPrivacy | Unchanged}] [-Locale <string>] [-EnableAllPrivileges]
    [-Authority <string>] [-Credential <pscredential>] [-ThrottleLimit <int>] [-ComputerName <string[]>] [-Namespace
    <string>] [<CommonParameters>]

    Get-WmiObject [[-Class] <string>] [-Recurse] [-Amended] [-List] [-AsJob] [-Impersonation {Default | Anonymous |
    Identify | Impersonate | Delegate}] [-Authentication {Default | None | Connect | Call | Packet | PacketIntegrity |
    PacketPrivacy | Unchanged}] [-Locale <string>] [-EnableAllPrivileges] [-Authority <string>] [-Credential
    <pscredential>] [-ThrottleLimit <int>] [-ComputerName <string[]>] [-Namespace <string>] [<CommonParameters>]

    Get-WmiObject -Query <string> [-Amended] [-DirectRead] [-AsJob] [-Impersonation {Default | Anonymous | Identify |
    Impersonate | Delegate}] [-Authentication {Default | None | Connect | Call | Packet | PacketIntegrity |
    PacketPrivacy | Unchanged}] [-Locale <string>] [-EnableAllPrivileges] [-Authority <string>] [-Credential
    <pscredential>] [-ThrottleLimit <int>] [-ComputerName <string[]>] [-Namespace <string>] [<CommonParameters>]

    Get-WmiObject [-Amended] [-AsJob] [-Impersonation {Default | Anonymous | Identify | Impersonate | Delegate}]
    [-Authentication {Default | None | Connect | Call | Packet | PacketIntegrity | PacketPrivacy | Unchanged}]
    [-Locale <string>] [-EnableAllPrivileges] [-Authority <string>] [-Credential <pscredential>] [-ThrottleLimit
    <int>] [-ComputerName <string[]>] [-Namespace <string>] [<CommonParameters>]

    Get-WmiObject [-Amended] [-AsJob] [-Impersonation {Default | Anonymous | Identify | Impersonate | Delegate}]
    [-Authentication {Default | None | Connect | Call | Packet | PacketIntegrity | PacketPrivacy | Unchanged}]
    [-Locale <string>] [-EnableAllPrivileges] [-Authority <string>] [-Credential <pscredential>] [-ThrottleLimit
    <int>] [-ComputerName <string[]>] [-Namespace <string>] [<CommonParameters>]

ALIASES
    gwmi

REMARKS
    Get-Help cannot find the Help files for this cmdlet on this computer. It is displaying only partial help.
    -- To download and install Help files for the module that includes this cmdlet, use Update-Help.
    -- To view the Help topic for this cmdlet online, type: "Get-Help Get-WmiObject -Online" or
    go to https://go.microsoft.com/fwlink/?LinkID=113337.
    
```

As a result, the eForensics reader can have information about almost all possible combinations to get information of objects using Get-WmiObject. The reader also can see the ALIASES gwmi, which means you can use the alias gwmi instead of Get-WmiObject for PowerShell operations. You may prefer to have a complete help information of the cmdlet Get-WmiObject. In such case, use the following:

PS C:\Users\Wash> Get-Help Get-WmiObject -full

```

Command Prompt - powershell
PARAMETERS
- Amended
    Required?           false
    Position?          Named
    Accept pipeline input? false
    Parameter set name (All)
    Aliases             None
    Dynamic?           false

- AsJob
    Required?           false
    Position?          Named
    Accept pipeline input? false
    Parameter set name (All)
    Aliases             None
    Dynamic?           false

- Authentication <AuthenticationLevel>
    Required?           false
    Position?          Named
    Accept pipeline input? false
    Parameter set name path, class, WQLQuery, query, list
    Aliases             None
    Dynamic?           false

- Authority <string>
    Required?           false
    Position?          Named
    Accept pipeline input? false
    Parameter set name path, class, WQLQuery, query, list
    Aliases             None
    Dynamic?           false
  
```

Beyond the information you receive with the command **Get-Help Get-WmiObject** you realize that adding the clause “-full” you can have the information about all parameters, which is partially shown in the screenshot taken above.

Essential PowerShell Configuration:

Initially, PowerShell works in a limited mode and only accepts interactive commands. To fully leverage its capabilities, a small set of settings control what you can do and what you cannot do. Before you start working with PowerShell on a new computer, you should review these settings (and adjust them if necessary).

PowerShell is locked down by default but you can decide which areas you want to open. Basically, you need to worry about two items: scripting and remote access.

Initially, PowerShell can only execute interactive commands, but no scripts. Scripts work like batch files that can contain many commands that run in an ordered sequence.

Even if you don't plan to run scripts yourself, you should consider enabling scripts right away. Many of the commands are script-based, and when you don't allow PowerShell to run scripts, you may run into a lot of limitations. Let me explain how to enable this feature.

```
PS C:\Users\Wash> Get-ExecutionPolicy
```

Restricted

```
PS C:\Users\Wash> Get-ExecutionPolicy -list
```

```
Scope                ExecutionPolicy
```

```
-----
```

```
MachinePolicy        Undefined
```

```
UserPolicy           Undefined
```

```
Process              Undefined
```

```
CurrentUser          Undefined
```

```
LocalMachine         Undefined
```

Get-ExecutionPolicy reports the effective setting. When you add the parameter `-List`, you discover that there are five settings. PowerShell evaluates them from top to bottom, and if all are `Undefined`, script execution is disabled ("Restricted"). This configuration is the default for the brand-new computers.

MachinePolicy: this parameter is controlled via Group Policy. This one should always be "Undefined". If you see anything else, contact the Active Directory Administrator of the company. Execution policy works like a seat belt for you, not like a security boundary for your company. Users should always be able to change this setting.

UserPolicy: Same as `MachinePolicy` but just per user.

Process: Applies to the currently running instance of PowerShell only, and does not affect other instances. This parameter is not often used.

CurrentUser: This is your setting. It applies only to you and no one else.

LocalMachine: If the Active Directory Administrator of your company accidentally set this setting in one of the top two scopes, then point them to this setting: it works like a general preference for all users, yet users can still change it in one of the scopes above. That's where centrally managed settings should be set. Remember, you need Admin privileges to make changes on this.

So, to enable scripts, change the setting for CurrentUser. PowerShell cmdlets typically come in pairs, so the verb "Get" reads, and the verb "Set" changes, like shown below:

```
PS C:\Users\Wash> Set-ExecutionPolicy -Scope CurrentUser -
ExecutionPolicy RemoteSigned -Force
```

```
PS C:\> Get-ExecutionPolicy
```

```
RemoteSigned
```

```
PS C:\> Get-ExecutionPolicy -List
```

Scope	ExecutionPolicy
-----	-----
MachinePolicy	Undefined
UserPolicy	Undefined
Process	Undefined
CurrentUser	RemoteSigned
LocalMachine	Undefined

When you enter these commands, depending on which host you use, you get different levels of help. In the PowerShell console, you can press TAB to get auto completion suggestions. Also, you can press TAB multiple times for more choices, and press SHIFT+TAB to go back. If you are in the ISE Editor, you don't have to really do anything since Intellisense opens automatically, and you just need to pick from the choices described below.

RemoteSigned: You can execute scripts that are stored on your computer, or on any other computer that is part of your domain. You cannot execute scripts that you downloaded from the Internet, received via Email, or that are stored on computers outside your domain – unless they carry a valid digital signature (which scripts typically seldom do).

Unrestricted: You can execute any script. If it falls under any of the “potentially hostile” categories listed above, a dialog pops up and asks whether you are sane.

Bypass: You can execute any script. Period. No questions asked.

Restricted: You cannot execute any script.

Undefined: Resets the scope. PowerShell continues to evaluate the child scopes. If they are all “Undefined”, then PowerShell defaults to “Restricted”

AllSigned: Every single script needs to carry a valid digital signature. Remember, this affects not just scripts you launch. Often, applications run PowerShell scripts in the background. This setting is useful only for very locked down systems like teller machines.

All settings persist until you decide to change them again, excepting for the scope “Process”.

For Remote Access on PowerShell you need to have local Administrator privileges on the remote system you want to access and work with PowerShell. Sometimes this feature is useful while working on forensics activities, but it must be previously configured. For the purpose of this article, this subject will not be covered. However, if the eForensics reader comes to face the need to enable this feature, consult Microsoft PowerShell Documentation.

Live response:

Live response is an area that deals with collecting information from a live machine in order to identify if an incident has occurred.

Such information includes artifacts such as process information, connection information, files opened by processes (i.e. ransomware), and any artifact to establish the fact that an incident has occurred.

As the goal of live response is to identify incidents as quickly as possible, the Forensics specialists must know what artifacts they should collect, which vary from case to case.

About the environment:

To take a first dip in the PowerShell environment I invite the eForensics reader to join us and use PowerShell to identify my machine and the environment.

Firstly, we open a command line interface and launch the following simple command powershell as shown below.

```
Microsoft Windows [Version 10.0.15063]
```

```
(c) 2017 Microsoft Corporation. All rights reserved.
```

```
C:\Users\Wash>powershell
```

```
Windows PowerShell
```

```
Copyright (C) 2016 Microsoft Corporation. All rights reserved.
```

```
PS C:\Users\Wash>
```

When launching PowerShell command line the prompt is changed with the characters PS in the beginning of the prompt, which means you are now working under the PowerShell environment.

So, instead of telling the eForensics reader about my computer environment, let us use PowerShell to get some information regarding my machine. The command I will be using is "Get-WmiObject -Class Win32_ComputerSystem".

```
PS C:\Users\Wash> Get-WmiObject -Class Win32_ComputerSystem
```

```
Domain           : WORKGROUP
```

```
Manufacturer     : Alienware
```

```
Model            : Alienware 13 R3
```

```
Name             : DESKTOP-7HH470E
```

```
PrimaryOwnerName :
```

```
TotalPhysicalMemory : 17046671360
```

I am working on an Alienware machine 13 inches equipped with 16Gb of RAM memory, identified by TotalPhysicalMemory.

Now let us understand this basic PS command.

Get-WmiObject – Lists details of a WMI class.

-Class – instruct PS to a type of class that must be informed.

Win32_ComputerSystem – Is the WMI class of which I want to collect information.

As you work with PowerShell, you realize that it is relatively simple to use. You just need to know the arguments you should use to get the results you want to collect. How about the BIOS? How can I identify the BIOS of my system using PowerShell? We just need to change the class followed by the correct parameters as shown below.

```
PS C:\Users\Wash> Get-WmiObject -Class Win32_BIOS -ComputerName .
```

```
SMBIOSBIOSVersion      : 1.1.6
Manufacturer           : Alienware
Name                   : 1.1.6
SerialNumber           : 388Z7H2
Version                : ALWARE - 1072009
```

The eForensics reader must realize the **Get-WmiObject** is one of the most important cmdlets for general system management tasks and it is relatively easy to perceive since all critical subsystem settings are exposed through WMI into Windows environment.

What is the wallpaper that I am using in my desktop? Can PowerShell tell me about it? Yes.

```
PS C:\Users\Wash> Get-WmiObject -Class Win32_Desktop -ComputerName . | Select-Object -Property Wallpaper
```

```
Wallpaper
```

```
-----
```

```
C:\Users\Wash\Pictures\rainbow_alienware_red_by_darkangelkrys-d4xmzna.jpg
```

And how about the processor that equips my notebook?

```
PS C:\Users\Wash> Get-WmiObject -Class Win32_Processor -ComputerName . | Select-Object -Property Name
```

Name

```
Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz
```

Can PowerShell collect the information regarding the number of the logical processors in the Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz?

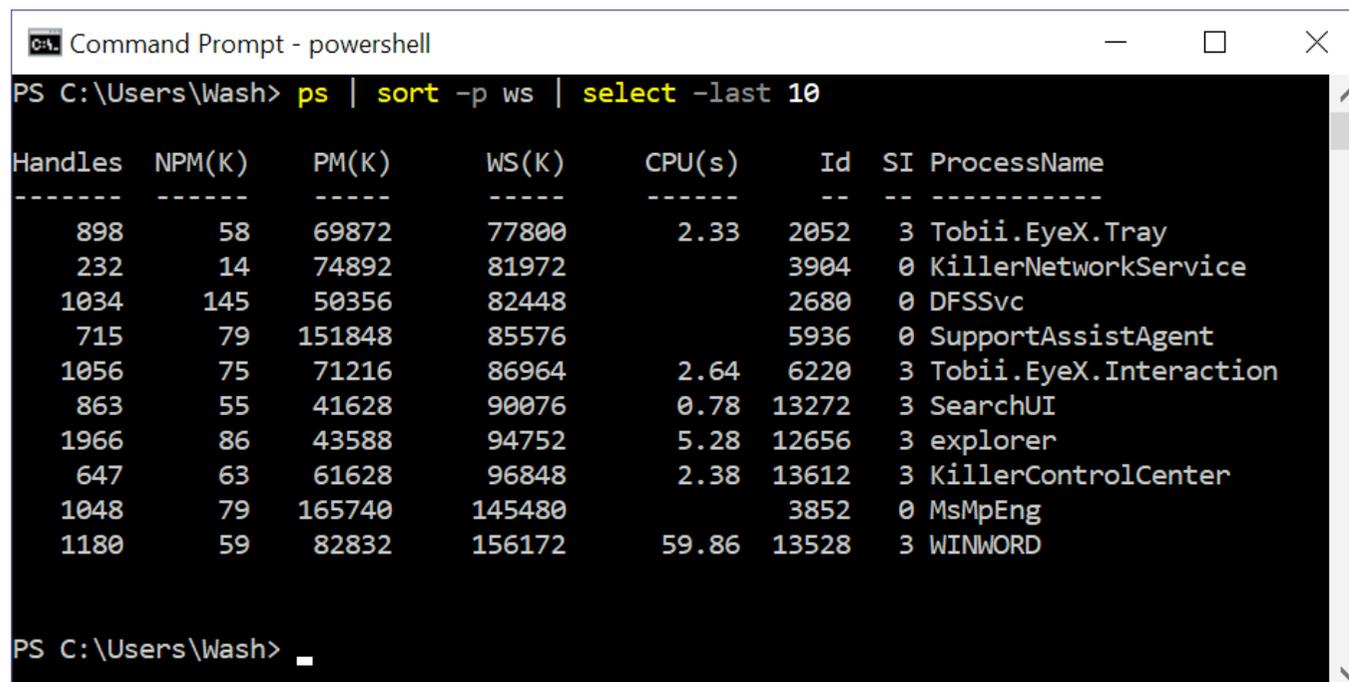
```
PS C:\Users\Wash> Get-WmiObject -Class Win32_Processor -ComputerName . | Select-Object -Property NumberOfLogicalProcessors
```

NumberOfLogicalProcessors

8

How about the top ten processes using the most memory?

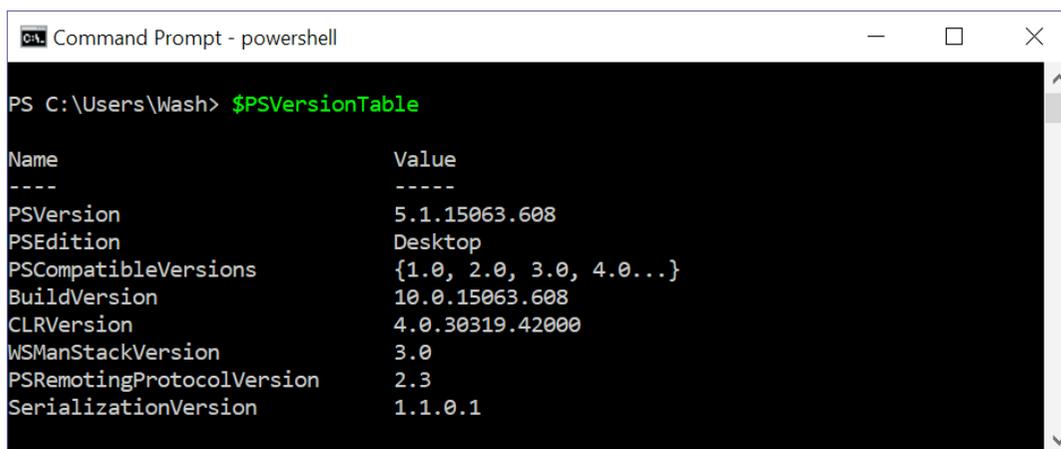
```
PS C:\Users\Wash> ps | sort -p ws | select -last 10
```



The examples above demonstrate how to collect specific information by using Get-WmiObject against an arbitrary computer. Notice that I specified the ComputerName parameter with the dot value (.), which represents the local computer. The eForensics reader may already realize that the PowerShell language is a very powerful tool that is available to forensic specialists in Windows environment system.

Also, depending on which version of PowerShell you are using, some commands may not be supported. So, it's always a good practice to check which version of PowerShell your system is using. To do so, just type \$PSVersionTable under PS command line prompt as below.

PS C:\Users\Wash> \$PSVersionTable

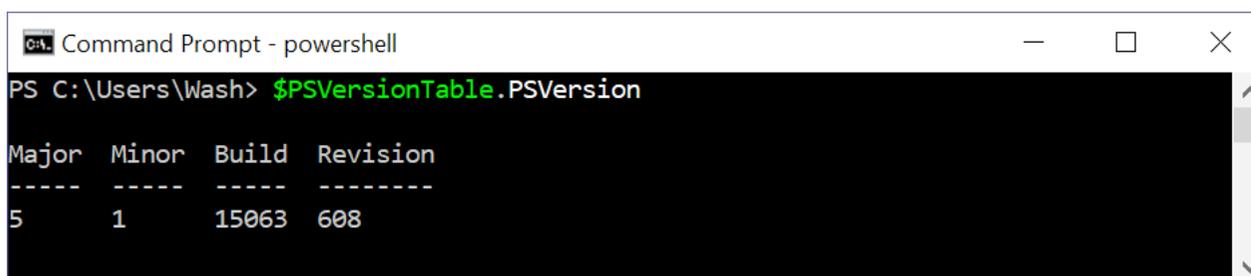


```

PS C:\Users\Wash> $PSVersionTable
Name                Value
----                -
PSVersion           5.1.15063.608
PSEdition           Desktop
PSCompatibleVersions {1.0, 2.0, 3.0, 4.0...}
BuildVersion        10.0.15063.608
CLRVersion          4.0.30319.42000
WSManStackVersion   3.0
PSRemotingProtocolVersion 2.3
SerializationVersion 1.1.0.1
  
```

Alternatively, you can collect only a specific item of the table above. Let us suppose we want just to collect the PSVersion information of the table. If you know how the information and tables are built-in into PowerShell, then it makes the task easier.

PS C:\Users\Wash> \$PSVersionTable.PSVersion



```

PS C:\Users\Wash> $PSVersionTable.PSVersion
Major Minor Build Revision
-----
5      1      15063 608
  
```

As the eForensics reader can see, the variable \$PSVersionTable lists the versions of PowerShell ("PSVersion") and all its related components. We can just use a dot (".") to drill deeper into the specific information we want to collect. Now let us go ahead and see how we can use PowerShell for forensics purposes.

Working with PowerShell for forensics:

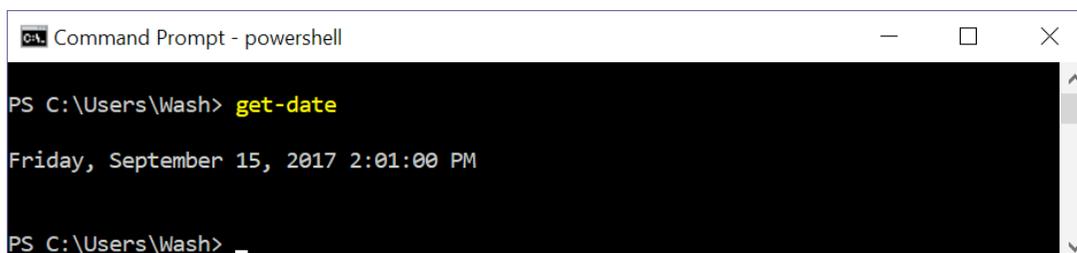
Writing about PowerShell can easily take us to several books separated by topics, which makes this article a real challenge in targeting PowerShell features for forensic activities.

So, let us suppose you face a breach in your systems. Thus, you want to retrieve as much information as possible to help you build a timeline of the intrusion, and hopefully figure out how the attackers gained access to your system and what parts of it they had access to. For this matter, PowerShell can prove to be very useful. Windows has integrated it in its newest operating systems, and steadily become Microsoft's preferred method for managing their core products. For example, the latest versions of Exchange, SharePoint, and even Windows Server 2012 can be managed almost entirely via PowerShell. As a matter of fact, many of the Microsoft GUI management tools provides for these applications are simply performing PowerShell operations in the background.

PowerShell provides a complete set of related cmdlets. We will explore some of them. The list of cmdlets that a forensics specialist can explore is very long. Also, some outputs will be omitted since they are very long.

Getting the date:

```
PS C:\Users\Wash> Get-Date
```

A screenshot of a Windows Command Prompt window titled "Command Prompt - powershell". The window has a black background and white text. The prompt shows the user at the C:\Users\Wash directory. The command 'get-date' has been entered and executed, resulting in the output: 'Friday, September 15, 2017 2:01:00 PM'. The prompt is now ready for the next command.

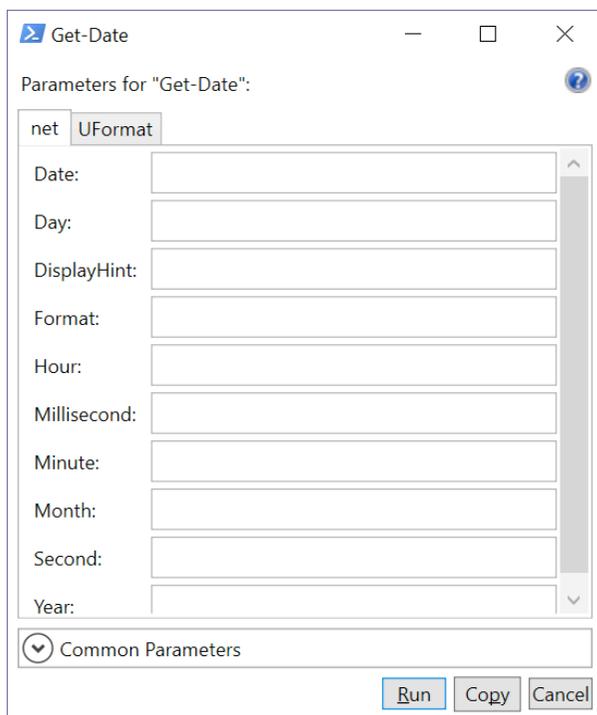
```
PS C:\Users\Wash> get-date  
Friday, September 15, 2017 2:01:00 PM  
PS C:\Users\Wash> _
```

Other PowerShell features: When in doubt about completing the parameters of a certain cmdlet, you can load the command panel even in command line mode. The command panel is a graphical window very similar to the PowerShell ISE command panel and allows you to view the optional parameters and mandatory cmdlets. For instance, let us do this Show-Command cmdlet.

```

Command Prompt - powershell
PS C:\Users\Wash> get-date
Friday, September 15, 2017 2:01:00 PM
PS C:\Users\Wash> Show-Command Get-Date
  
```

And Windows will open the cmdlet Get-Date in a GUI showing all possible parameters that can be used.



Collecting Network Information:

```
PS C:\Users\Wash> Get-NetTCPConnection -State Established
```

```

Command Prompt - powershell
PS C:\Users\Wash> Get-NetTCPConnection -State Established

LocalAddress      LocalPort RemoteAddress      RemotePort State      AppliedSetting
-----
172.20.6.78       64943    65.52.108.196      443       Established Internet
172.20.6.78       49777    23.210.204.101     443       Established Internet
172.20.6.78       49776    23.210.204.101     443       Established Internet
172.20.6.78       49775    23.210.204.101     443       Established Internet
172.20.6.78       49774    184.85.164.127     443       Established Internet
172.20.6.78       49773    23.210.205.143     80        Established Internet
172.20.6.78       49723    104.40.17.139      443       Established Internet
172.20.6.78       49722    104.40.17.139      443       Established Internet
172.20.6.78       49721    72.21.81.200       443       Established Internet
  
```

PS C:\> Get-NetNeighbor

The Get- NetNeighbor cmdlet gets the neighbor cache entries. The output expected will be something like what follows:

```

Command Prompt - powershell
PS C:\Users\Wash> Get-NetNeighbor

ifIndex IPAddress LinkLayerAddress State PolicyStore
-----
15 ff02::1:3 33-33-00-01-00-03 Permanent ActiveStore
15 ff02::1:2 33-33-00-01-00-02 Permanent ActiveStore
15 ff02::fb 33-33-00-00-00-FB Permanent ActiveStore
15 ff02::16 33-33-00-00-00-16 Permanent ActiveStore
12 ff02::1:3 255.255.255.255:655 Permanent ActiveStore
12 ff02::1:2 255.255.255.255:655 Permanent ActiveStore
12 ff02::fb 255.255.255.255:655 Permanent ActiveStore
12 ff02::16 255.255.255.255:655 Permanent ActiveStore
12 ff02::2 255.255.255.255:655 Permanent ActiveStore
12 fec0:0:0:ffff::3 255.255.255.255:655 Unreachable ActiveStore
12 fec0:0:0:ffff::2 255.255.255.255:655 Unreachable ActiveStore
12 fec0:0:0:ffff::1 255.255.255.255:655 Unreachable ActiveStore
  
```

Controlling the display (output) of information.

Information that you can collect through Windows PowerShell can be formatted to make it easier to see information. Using the pipe (|) we can pass the output of the command to several options. The pipe is an operator. Each command after the pipe receives an object from the previous command, performs some operation on the object, and then passes on to the next command in the pipeline. Some examples are shown below:

Get-Process | more

Get-Process | Format-List

Get-Process | Format-List | more

Get-Process | ConvertTo-HTML | Out-File "Processes.html"

Get-Process | Export-CSV "Processes.csv"

We will explore this feature later.

Collecting information about what was typed in sessions:

PowerShell automatically maintains a history of each session. We can save session history in XML or CSV format. By default, the history files are saved in the directory where the commands are executed, in this case c: \ users \ administrator, but the file can be saved anywhere.

We have four Windows PowerShell cmdlets that allow us to work with **Get-History**.

```

Command Prompt - powershell
Wash - eForensic article - 09/2017>get-command -noun history

CommandType      Name                               Version      Source
-----
Cmdlet           Add-History                        3.0.0.0     M...
Cmdlet           Clear-History                      3.0.0.0     M...
Cmdlet           Get-History                        3.0.0.0     M...
Cmdlet           Invoke-History                    3.0.0.0     M...

```

Using Get-History, the Forensics Expert can see what was typed in the console.

```

Command Prompt - powershell
Wash - eForensic article - 09/2017>get-history

Id CommandLine
--
1 ps | sort -p ws | select -last 10
2 $PSVersionTable
3 $PSVersionTable.PSVersion
4 get-disk
5 Get-ItemProperty HKLM:\Software\Wow6432Node\Microsoft\Windows\CurrentVersi...
6 clear
7 Get-ItemProperty HKLM:\Software\Wow6432Node\Microsoft\Windows\CurrentVersi...
8 get-date
9 Show-Command Get-Date
10 Get-Process | Out-GridView
11 Get-Module -ListAvailable
12 function prompt {"Wash - eForensic article - 09/2017>"}
13 Get-Random -minimum 1 -maximum 101
14 clear
15 Get-Random -minimum 1 -maximum 50
16 clear
17 ($a = "Wash","Marta","Anna") | Get-Random
18 ($a = "Wash","Marta","Anna","eForensics Reader") | Get-Random
19 ($a = "Wash","Marta","Anna","eForensics Reader") | Get-Random
20 ($a = "Wash","Marta","Anna","eForensics Reader") | Get-Random

```

Collecting System Information:

PS C:\Users\Wash> Get-Process

The Get-Process cmdlet gets the processes on a local or remote computer. By default, Get-Process returns a process object that has detailed information about the process and supports methods that let you start and stop the process. You can also use the parameters of Get-Process to get file version information for the program that runs in the process and to get the modules that the process loaded.

The output expected will be something like what follows:

```

Command Prompt - powershell
PS C:\Users\Wash> Get-Process

Handles  NPM(K)  PM(K)  WS(K)  CPU(s)  Id  SI  ProcessName
-----  -
190      9       3456   8388   0.00    3732  0  AdminService
254      19      13052  19528  0.08    14192  1  AlienFusionController
276      31      18624  29848  0.00    5960  0  AlienFusionService
532      37      27836  46236  0.64    12164  1  AlienwareAlienFXController
192      13      2464   11032  0.47    9720  1  AlienwareOn-ScreenDisplay
433      22      13832  32392  2.61    5092  1  ApplicationFrameHost
185      13      5744   9420   0.03    12400  1  AWCCApplicationWatcher32
167      22      9200   10016  0.02    12412  1  AWCCApplicationWatcher64
351      35      26924  31352  0.09    11696  1  AWCCServiceController
743      24      11268  1812   2.09    11656  1  AWSoundCenterSvc32
209      12      6424   2264   1.11    9908  1  AWSoundCenterSvc64
199      12      4180   1268   0.28    1308  1  AWSoundCenterUILauncher
327      15      3204   16000  0.13    10084  1  browser_broker
43       4       3148   3240   0.02    13828  1  cmd
    
```

Windows Event Logs can be collected by cmdlet Get-EventLog. If used without a parameter, the forensics specialist will be invited to supply the name of the Log he/she wants to collect. I suggest the eForensics reader launch Get-EventLog with the parameter "-list" in order to see all Logs available to be collected.

Some existing cmdlets create incredible outputs, such as out cmdlets. To list the "out" cmdlets: Get-command out-

Out-Default - Send the output to the standard formatter and the default output cmdlet.

Out-File - Outputs a file.

Out-GridView - Output the output to an interactive table in a separate window.

Out-Host - Sends the output to the command line.

Out-Null - Erases output instead of sending it to the console.

Out-Printer - Outputs to a printer.

Out-String - Sends out a string of strings.

Get-Process | Out-GridView brings the following output:

Handles	NPM(K)	PM(K)	WS(K)	CPU(s)	Id	SI	ProcessName
238	9	4588	9416		3,732	0	AdminService
214	17	12660	17936	0.08	10,648	3	AlienFusionController
288	31	18932	29976		5,960	0	AlienFusionService
563	39	28276	48892	0.75	6,696	3	AlienwareAlienFXController
206	14	2500	11424	0.30	12,684	3	AlienwareOn-ScreenDisplay
522	26	12240	33812	3.09	6,308	3	ApplicationFrameHost
185	12	5652	9360	0.02	6,528	3	AWCCApplicationWatcher32
167	22	9120	9800	0.00	9,064	3	AWCCApplicationWatcher64
353	35	26836	31052	0.16	15,308	3	AWCCServiceController
617	24	11116	1284	1.61	9,572	3	AWSoundCenterSvc32
209	12	5284	420	0.95	10,292	3	AWSoundCenterSvc64
199	11	4040	152	0.27	6,848	3	AWSoundCenterUILauncher
372	16	3448	17112	0.09	6,520	3	browser_broker
43	4	2112	2892	0.02	6,488	3	cmd
131	8	1552	6884		1,000	0	conhost
241	14	5872	17308	3.17	1,968	3	conhost
107	7	5320	92	0.05	3,116	3	conhost
107	7	5328	9004	0.02	5,456	3	conhost

PS C:\Users\Wash> Get-EventLog -list

```

PS C:\Users\Wash> Get-EventLog -list

Max(K) Retain OverflowAction      Entries Log
-----
20,480  0 OverwriteAsNeeded      4,101 Application
512    7 OverwriteOlder         29 Dell
20,480  0 OverwriteAsNeeded      0 HardwareEvents
512    7 OverwriteOlder         0 Internet Explorer
20,480  0 OverwriteAsNeeded      0 Key Management Service
128    0 OverwriteAsNeeded      4 OALerts
Security
20,480  0 OverwriteAsNeeded      1,327 System
15,360  0 OverwriteAsNeeded      47 Windows PowerShell

PS C:\Users\Wash>

```

Now let us suppose we want to collect the Logs regarding the Dell log entries.

PS C:\Users\Wash> Get-EventLog Dell

```

C:\> Command Prompt - powershell
PS C:\Users\Wash> Get-EventLog Dell

Index Time          EntryType Source                InstanceID Message
-----
32 Sep 13 21:07 Information Update              0 Windows service started.
31 Sep 13 21:07 Information DigitalDelivery      0 Windows service started.
30 Sep 13 21:07 Information Update              0 Windows service started.
29 Sep 11 21:05 Information Update              0 Driver Updates Found: 0
28 Sep 11 21:05 Information Update              0 Scanning System for Driver Updates
27 Sep 08 10:55 Information Update              0 Windows service started.
26 Sep 08 10:55 Information DigitalDelivery      0 Windows service started.
25 Sep 08 10:55 Information Update              0 Windows service started.
24 Sep 08 10:44 Information Update              0 Driver Updates Found: 0
23 Sep 08 10:44 Information Update              0 Scanning System for Driver Updates
22 Sep 04 12:29 Information Update              0 Driver Updates Found: 0
21 Sep 04 12:29 Information Update              0 Scanning System for Driver Updates
20 Sep 01 00:53 Information Update              0 Installation of Qualcomm QCA61x4A And QCA9377 W...
19 Sep 01 00:51 Information Update              0 Installing Qualcomm QCA61x4A And QCA9377 WiFi A...
18 Sep 01 00:51 Information Update              0 Download of Qualcomm QCA61x4A And QCA9377 WiFi ...
17 Sep 01 00:50 Information Update              0 Driver Updates Found: 1
16 Sep 01 00:50 Information Update              0 Scanning System for Driver Updates
15 Aug 31 22:56 Information DigitalDelivery      0 Got 0 pending downloads from Dell servers.
14 Aug 31 22:55 Information DigitalDelivery      0 Got 0 entitlements from Dell servers.
13 Aug 31 22:45 Information Update              0 Windows service started.
12 Aug 31 22:45 Information DigitalDelivery      0 Windows service started.
11 Aug 31 22:45 Information Update              0 Windows service started.
10 Aug 31 22:20 Information DigitalDelivery      0 Windows service stopped.
9 Aug 31 23:59 Information Update              0 Windows service started.
8 Aug 31 23:59 Information DigitalDelivery      0 Windows service started.
7 Aug 31 23:59 Information Update              0 Windows service started.

```

Getting the username of the person currently logged on to a computer:

```

C:\> Select Command Prompt - powershell
PS C:\Users\Wash> Get-WmiObject -Class Win32_ComputerSystem -Property UserName -ComputerName .

GENUS           : 2
CLASS           : Win32_ComputerSystem
SUPERCLASS      :
DYNASTY         :
RELPATH         :
PROPERTY_COUNT  : 1
DERIVATION      : {}
SERVER         :
NAMESPACE       :
PATH            :
UserName        : DESKTOP-7HH470E\Wash
PSComputerName  :

```

Finding the names of installed applications on the current computer:

Get-WmiObject -Class Win32_Product -ComputerName . | Format-Wide -Column 1

```

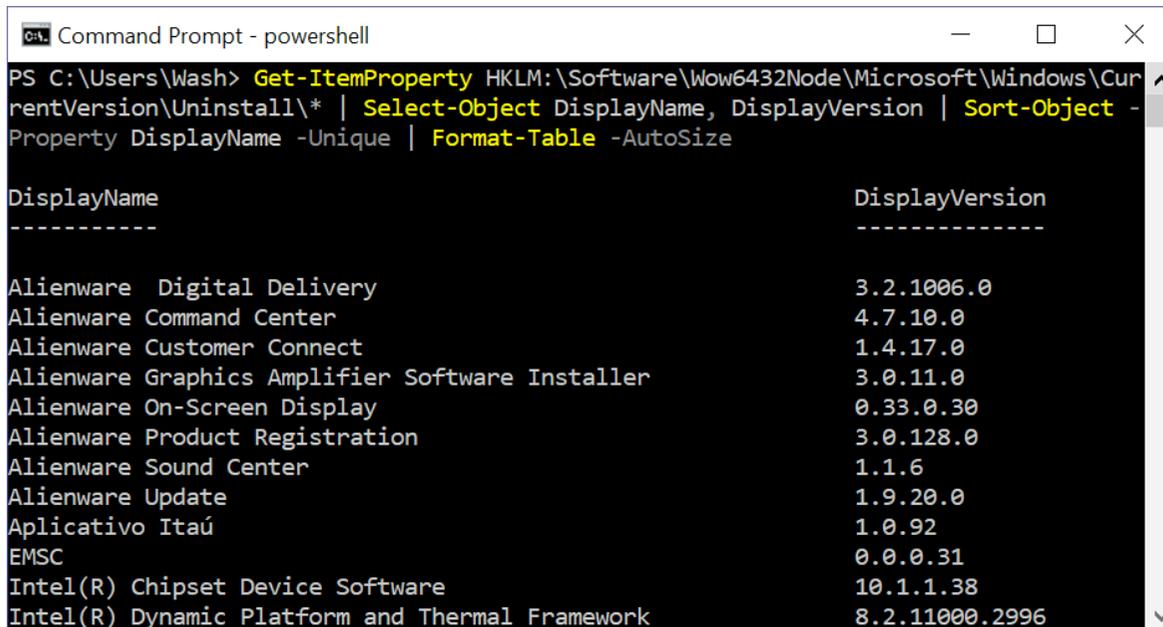
C:\> Command Prompt - powershell
PS C:\Users\Wash> Get-WmiObject -Class Win32_Product -ComputerName . | Format-Wide -Column 1

Aplicativo Itaú
Office 16 Click-to-Run Extensibility Component
Office 16 Click-to-Run Localization Component
Office 16 Click-to-Run Extensibility Component 64-bit Registration
Office 16 Click-to-Run Licensing Component
DSC/AA Factory Installer
LauncherSetup
ProductDaemonSetup
Dell Foundation Services
Microsoft VC++ redistributables repacked.
Microsoft Visual C++ 2015 x64 Minimum Runtime - 14.0.23918
Tobii EyeX Intro
Tobii Bundle Requirements
Microsoft Visual C++ 2013 x86 Minimum Runtime - 12.0.21005

```

Finding the names of installed applications on the current computer and its versions:

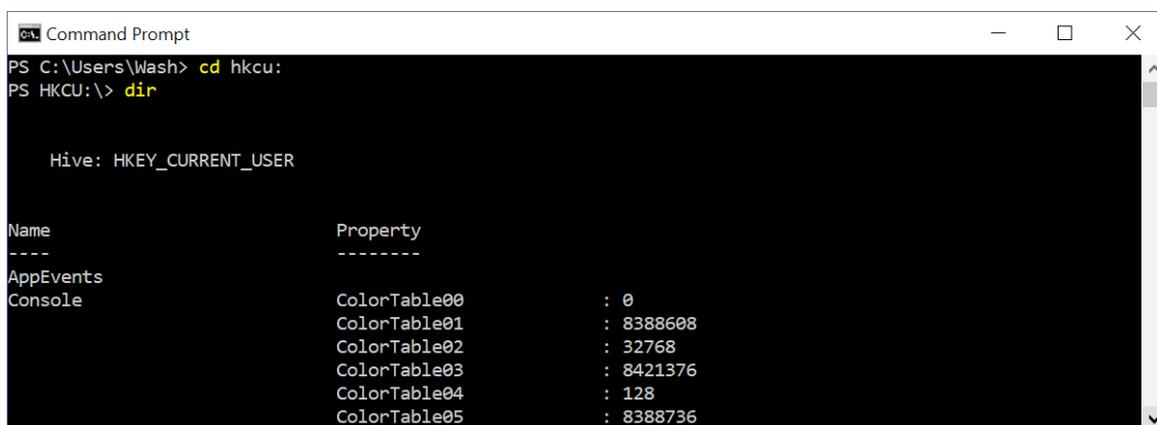
```
Get-ItemProperty HKLM:\Software\Wow6432Node\Microsoft\Windows\CurrentVersion
\Uninstall\* | Select-Object DisplayName, DisplayVersion | Sort-Object -Property
DisplayName -Unique | Format-Table -AutoSize
```



```
PS C:\Users\Wash> Get-ItemProperty HKLM:\Software\Wow6432Node\Microsoft\Windows\Cur
rentVersion\Uninstall\* | Select-Object DisplayName, DisplayVersion | Sort-Object -
Property DisplayName -Unique | Format-Table -AutoSize
```

DisplayName	DisplayVersion
Alienware Digital Delivery	3.2.1006.0
Alienware Command Center	4.7.10.0
Alienware Customer Connect	1.4.17.0
Alienware Graphics Amplifier Software Installer	3.0.11.0
Alienware On-Screen Display	0.33.0.30
Alienware Product Registration	3.0.128.0
Alienware Sound Center	1.1.6
Alienware Update	1.9.20.0
Aplicativo Itaú	1.0.92
EMSC	0.0.0.31
Intel(R) Chipset Device Software	10.1.1.38
Intel(R) Dynamic Platform and Thermal Framework	8.2.11000.2996

We can navigate into Windows Registry the same way we navigate into the file system:



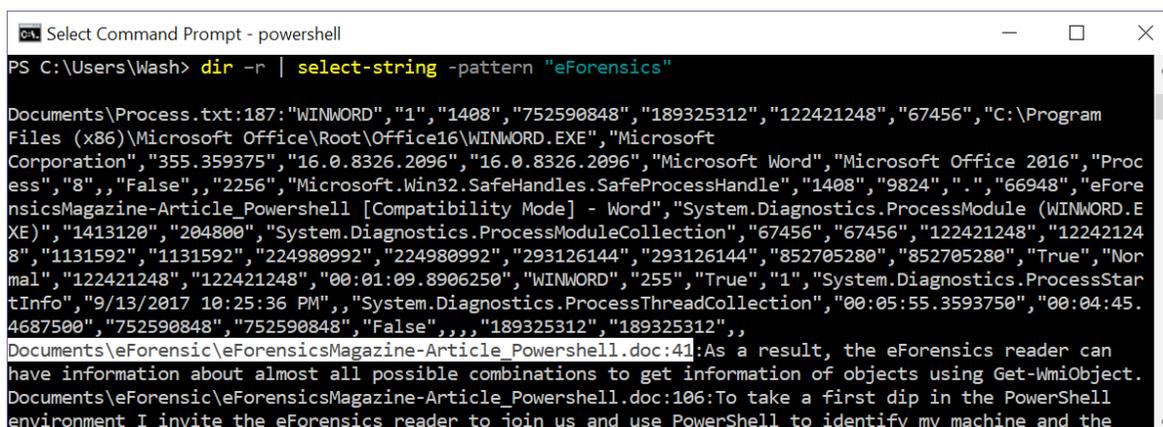
```
PS C:\Users\Wash> cd hkcu:
PS HKCU:\> dir
```

```
Hive: HKEY_CURRENT_USER
```

Name	Property
AppEvents	
Console	ColorTable00 : 0
	ColorTable01 : 8388608
	ColorTable02 : 32768
	ColorTable03 : 8421376
	ColorTable04 : 128
	ColorTable05 : 8388736

The Forensics Expert can also search recursively for a certain string within files:

```
PS C:\Users\Wash> dir -r | select-string -pattern "eForensics"
```



```
PS C:\Users\Wash> dir -r | select-string -pattern "eForensics"
```

```
Documents\Process.txt:187:"WINWORD", "1", "1408", "752590848", "189325312", "122421248", "67456", "C:\Program
Files (x86)\Microsoft Office\Root\Office16\WINWORD.EXE", "Microsoft
Corporation", "355.359375", "16.0.8326.2096", "16.0.8326.2096", "Microsoft Word", "Microsoft Office 2016", "Proc
ess", "8", "False", "2256", "Microsoft.Win32.SafeHandles.SafeProcessHandle", "1408", "9824", ".", "66948", "eFore
nsicsMagazine-Article_Powershell [Compatibility Mode] - Word", "System.Diagnostics.ProcessModule (WINWORD.E
XE)", "1413120", "204800", "System.Diagnostics.ProcessModuleCollection", "67456", "67456", "122421248", "12242124
8", "1131592", "1131592", "224980992", "224980992", "293126144", "293126144", "852705280", "852705280", "True", "Nor
mal", "122421248", "122421248", "00:01:09.8906250", "WINWORD", "255", "True", "1", "System.Diagnostics.ProcessStar
tInfo", "9/13/2017 10:25:36 PM", "System.Diagnostics.ProcessThreadCollection", "00:05:55.3593750", "00:04:45.
4687500", "752590848", "752590848", "False", "189325312", "189325312",
Documents\eforensic\eforensicsMagazine-Article_Powershell.doc:41:As a result, the eForensics reader can
have information about almost all possible combinations to get information of objects using Get-WmiObject.
Documents\eforensic\eforensicsMagazine-Article_Powershell.doc:106:To take a first dip in the PowerShell
environment I invite the eForensics reader to join us and use PowerShell to identify my machine and the
```

Listing installed hotfixes -- QFEs, or Windows Update files:

Get-WmiObject -Class Win32_QuickFixEngineering -ComputerName .

```

PS C:\Users\Wash> Get-WmiObject -Class Win32_QuickFixEngineering -ComputerName .
Source      Description      HotFixID      InstalledBy      InstalledOn
-----      -
DESKTOP-7H... Update          KB4022405     NT AUTHORITY\SYSTEM  9/1/2017 12:00:00 AM
DESKTOP-7H... Security Update  KB4034662     NT AUTHORITY\SYSTEM  9/1/2017 12:00:00 AM
DESKTOP-7H... Security Update  KB4038806     NT AUTHORITY\SYSTEM  9/14/2017 12:00:00 AM
DESKTOP-7H... Security Update  KB4038788     NT AUTHORITY\SYSTEM  9/14/2017 12:00:00 AM
    
```

Start-Up Processes:

Another useful thing to remember when carrying out a forensic investigation is to check for strange start-up processes. This can be easily done with the "Get-CimInstance" and its ability to access CIM instances of a class from a CIM server.

```

PS C:\Users\Wash> Get-CimInstance win32_service -Filter "startmode = 'auto'"
ProcessId Name          StartMode State  Status ExitCode
-----
5960      AlienFusionService  Auto    Running OK     0
3732      AtherosSvc          Auto    Running OK     0
2660      AudioEndpointBuilder Auto    Running OK     0
2760      Audiosrv            Auto    Running OK     0
2712      BFE                 Auto    Running OK     0
444       BITS                Auto    Running OK     0
896       BrokerInfrastructure Auto    Running OK     0
2144      CDPSvc              Auto    Running OK     0
3768      ClickToRunSvc       Auto    Running OK     0
2712      CoreMessagingRegistrar Auto    Running OK     0
3748      cplspcon            Auto    Running OK     0
3756      CryptSvc            Auto    Running OK     0
896       DcomLaunch          Auto    Running OK     0
7224      DDVCollectorSvcApi  Auto    Running OK     0
    
```

Recently Modified Files:

Using the "Get-ChildItem" we can retrieve the items and child items in one or more specified locations. By doing so recursively and then applying a filter to the modified date of the file it is possible to get a list of files modified within a certain period; for the following example we will use within the last 2 days:

```

Command Prompt - powershell
PS C:\Users\Wash> Get-ChildItem -Recurse C:\Users\Wash\Documents\ | ? {$_.lastwritetime -gt
(Get-Date).AddDays(-2)}

Directory: C:\Users\Wash\Documents

Mode                LastWriteTime         Length Name
----                -
d-----          9/13/2017  11:44 PM             eForensic

Directory: C:\Users\Wash\Documents\eForensic

Mode                LastWriteTime         Length Name
----                -
-a-----          9/13/2017  11:44 PM    4986368 eForensicsMagazine-Article_Powershell.doc

PS C:\Users\Wash>

```

To list the modules on your server or workstation run the cmdlets:

Get-Module -ListAvailable

```

Command Prompt - powershell
PS C:\Users\Wash> Get-Module -ListAvailable

Directory: C:\Program Files\WindowsPowerShell\Modules

ModuleType Version      Name                               ExportedCommands
-----
Script      1.0.1        Microsoft.PowerShell.Operation.V... {Get-OperationValidat...
Binary      1.0.0.1      PackageManagement                 {Find-Package, Get-Pa...
Script      3.4.0        Pester                             {Describe, Context, I...
Script      1.0.0.1      PowerShellGet                      {Install-Module, Find...
Script      1.2          PSReadline                        {Get-PSReadlineKeyHan...

Directory: C:\Windows\system32\WindowsPowerShell\v1.0\Modules

ModuleType Version      Name                               ExportedCommands
-----
Manifest    1.0.0.0      AppBackgroundTask                 {Disable-AppBackgroun...
Manifest    2.0.0.0      Appx                               {Add-AppxPackage, Get...

```

And you will see that there are several interesting modules and some of them we will use throughout this material.

Remote Machines:

PowerShell has a very interesting and relatively new management platform called WinRM. From a PowerShell standpoint, WinRM provides the platform that allows for running PowerShell commands directly on remote machines. WinRM is included by default on Vista and higher, and can be installed on

XP and Server 2003 R2. However, the WinRM service is not running by default on workstation platforms (Vista/7/8), though it is started automatically on Server 2008 and 2012. For more information please refer to the official documentation informed in the beginning of this article.

The cmdlet Invoke-Command runs the commands on remote computers and displays the result on your console. This is extremely useful in environments with multiple servers or computers and you want to run commands remotely. However, as explained before, for security reasons the Forensics Expert may not find this feature enabled on the customer's environment.

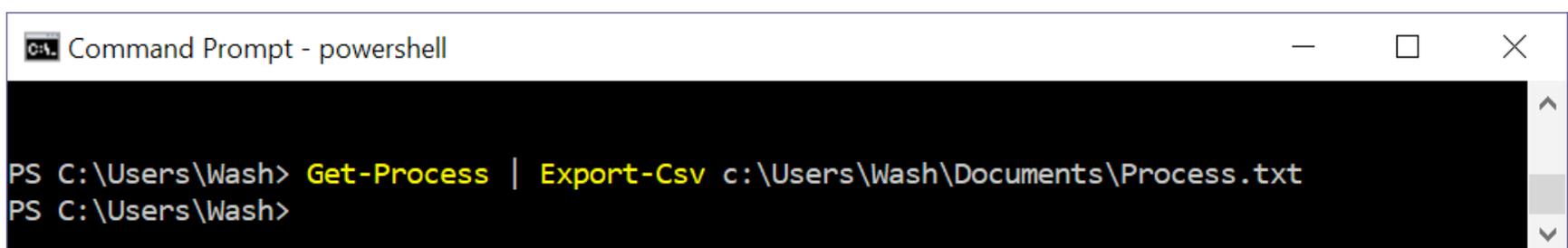
An example in my office would be the following:

```
Invoke-Command -ComputerName Wash-Station-1 -ScriptBlock {Get-WmiObject Win32_Bios
```

The result would be the BIOS information of the computer named Wash-Station-1.

Saving collected Data as a Comma-Separated Values File:

Another point to mention is that PowerShell provides multiple cmdlets to export the output of its commands and store them in a convenient way. Some examples of these cmdlets are:



```
Command Prompt - powershell
PS C:\Users\Wash> Get-Process | Export-Csv c:\Users\Wash\Documents\Process.txt
PS C:\Users\Wash>
```

This command uses Get-Process to grab information about all the processes running on the computer, then uses Export-Csv to write that data to a file named Process.txt that will be written in the folder c:\Users\Wash\Documents.

The resulting text file is shown below.

```

Process - Notepad
File Edit Format View Help
#TYPE System.Diagnostics.Process
"Name","SI","Handles","VM","WS","PM","NPM","Path","Company","CPU","FileVersion","ProductVersion","Descriptor"
"AdminService","0","190","76947456","8536064","3543040","9168",,,,,,,,,,"Process","8",,,,,,,,,,"190","3732","."
"AlienFusionController","1","254","221589504","17137664","13361152","19128","C:\Program Files\Alienware\C
"AlienFusionService","0","291","510410752","30408704","19042304","31776",,,,,,,,,,"Process","8",,,,,,,,,,"291",
"AlienwareAlienFXController","1","527","334168064","42491904","28434432","38056","C:\Program Files\Alienw
"AlienwareOn-ScreenDisplay","1","192","101515264","11100160","2523136","13584",,,,"0.546875",,,,,,"Process"
"ApplicationFrameHost","1","789","2199243849728","49074176","25612288","36440","C:\Windows\system32\Appli
"AWCCApplicationWatcher32","1","185","122617856","9314304","5885952","12872","C:\Program Files\Alienware\
"AWCCApplicationWatcher64","1","167","519507968","9760768","9420800","22800","C:\Program Files\Alienware\
"AWCCServiceController","1","351","647262208","28958720","27570176","36232","C:\Program Files\Alienware\C
"AWSoundCenterSvc32","1","755","159625216","4198400","11538432","24464",,,,"2.625",,,,,,"Process","8",,,,,,
"AWSoundCenterSvc64","1","209","113725440","1830912","6578176","12432",,,,"1.46875",,,,,,"Process","8",,,,,,
"AWSoundCenterUILauncher","1","199","113004544","831488","4210688","11664",,,,"0.28125",,,,,,"Process","6",
"browser_broker","1","327","2199131443200","15982592","3231744","14984","C:\Windows\system32\browser_brok
"cmd","1","43","2199041986560","2572288","3223552","3800","C:\Windows\system32\cmd.exe","Microsoft Corpor
"conhost","0","131","2199087853568","7053312","1654784","8544",,,,,,,,,,"Process","8",,,,,,,,,,"131","4584","."
"conhost","1","107","2199112658944","1286144","5496832","7736","C:\Windows\system32\conhost.exe","Microsc
"conhost","1","128","2199116103680","10133504","5820416","9640","C:\Windows\system32\conhost.exe","Microsc

```

By default, data is saved in ASCII format. What if you'd prefer to save the data in Unicode format (or maybe UTF7 or UTF8)? Just add the `-encoding` parameter followed by the desired format like "Unicode".

Changing the PowerShell Prompt:

The Forensics Expert may want to change the PowerShell prompt while analyzing the machine. This is not complex and can be done with the following embedded function as shown below:

```
function prompt {"Wash - eForensic article - 09/2017>"}
```

The result is the prompt changing just after launching the function.

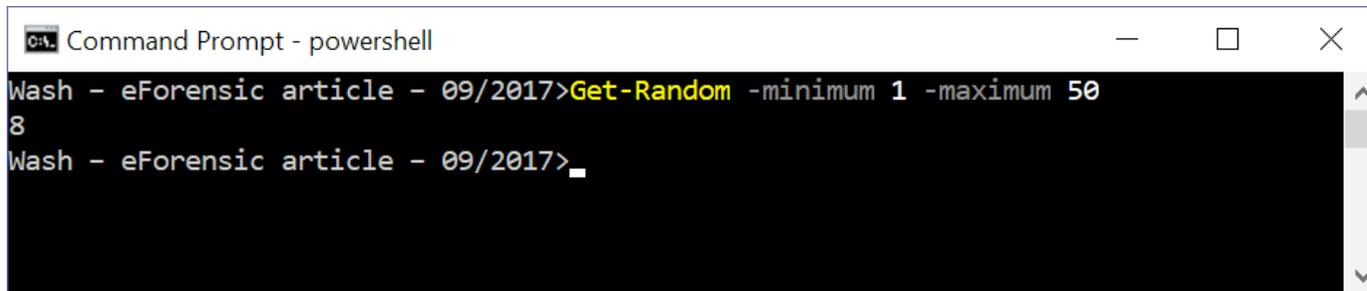
```

Command Prompt - powershell
PS C:\Users\Wash> function prompt {"Wash - eForensic article - 09/2017>"}
Wash - eForensic article - 09/2017>

```

Other PowerShell features:

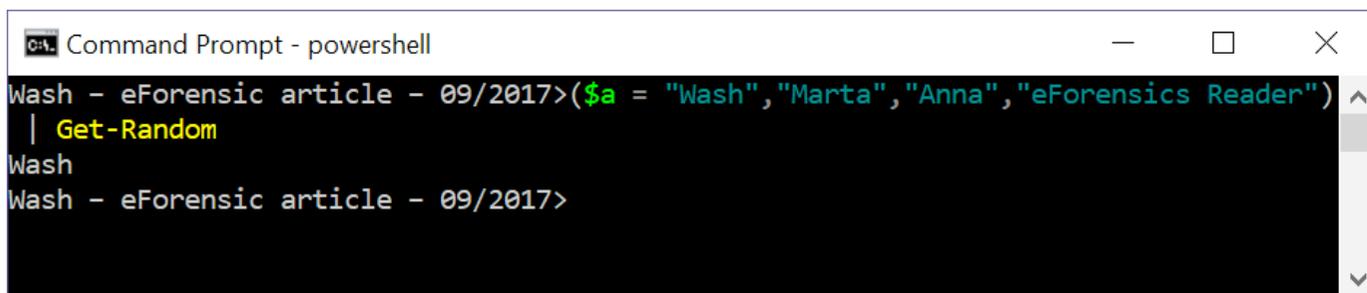
PowerShell can also make random sweeps through the System.Random Object. As an example, let us draw a number from 1 to 50.



```

Command Prompt - powershell
Wash - eForensic article - 09/2017>Get-Random -minimum 1 -maximum 50
8
Wash - eForensic article - 09/2017>
  
```

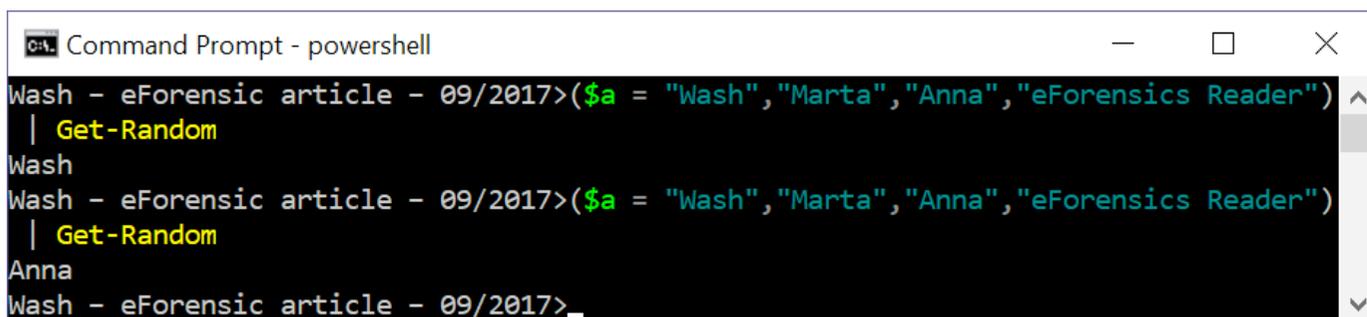
Let us now draw a name between Wash, Marta, Anna and eForensic Reader.



```

Command Prompt - powershell
Wash - eForensic article - 09/2017>($a = "Wash","Marta","Anna","eForensics Reader")
| Get-Random
Wash
Wash - eForensic article - 09/2017>
  
```

Wash was drawn. Let us repeat.



```

Command Prompt - powershell
Wash - eForensic article - 09/2017>($a = "Wash","Marta","Anna","eForensics Reader")
| Get-Random
Wash
Wash - eForensic article - 09/2017>($a = "Wash","Marta","Anna","eForensics Reader")
| Get-Random
Anna
Wash - eForensic article - 09/2017>
  
```

Now Anna was drawn. Another interesting feature is that you can use a text file and draw content. The following command would make it.

```
Get-Content C:\Users\Wash\Scripts\Wash-list.txt | Get-Random
```

PowerShell ISE:

You can choose to load the PowerShell Integrated Scripting Environment (ISE), a PowerShell programming environment that makes it easy to develop scripts because you can execute commands, write, test, and debug scripts in a Windows-based graphical user interface.

PowerShell and Cyber Security:

Any scripting language can be used to spread malicious code and PowerShell is not an exception. Because of this, Windows PowerShell by default does not allow the scripts to be run. It is controlled by what is called an execution policy, as explained earlier.

I strongly recommend that you enable PowerShell scripts to run if, and only if, you have full understanding of the security implications that may affect your environment as well as running its features on remote access. Do not do this if you do not quite understand how to operate with PowerShell for scripting and the potential impacts by running via remote access.

Summary:

The information you want to collect with PowerShell in a forensic investigation may vary from case to case, however, we know that PowerShell can bring any and all information from the investigated environment using the appropriate query. Remember we always must have a plan. So plan the activity before starting into it.

PowerShell comes with a large number of in-built commands to satisfy the live response need. Developers can also utilize the PowerShell APIs to create additional Cmdlets if required.

Since PowerShell is based on .NET Framework, it also enables PowerShell commands access to large collection of .NET classes. The .NET classes already provide access to various Windows system resources.

Another major advantage of using PowerShell for live response is its ability to completely automate, which is always beneficial as it becomes more efficient and scalable.

About the author: Washington Almeida



Washington Almeida is an electronic engineer specializing in digital forensics and cyber security with more than 25 years of experience working for large companies in cases involving intellectual property infringement, computer network intrusion, social networking monitoring, among others. Cyber security professional also works with sophisticated systems invasion testing, helping companies to improve the security of their assets.

In the assistance of the Justice, he is licensed by the “Tribunal de Justica de São Paulo” and “Tribunal Regional do Trabalho da 2ª Região” to work as digital forensic expert appointed by the judge.

Wash Web page: www.washingtonalmeida.com.br

Washington Almeida e-mail: wualmeida@outlook.com