# Global Information Assurance Certification Paper

# Windows Installed Software Inventory

*Gathering the Information Needed For the 20 Critical Controls*

*GIAC (GCCC) Gold Certification*

Author: Jonathan Risto, jonathan.risto@hotmail.com
Advisor: Richard Carbone
Accepted: August 20, 2016

Abstract

The 20 Critical Controls provide a guideline for the controls that need to be placed in our networks to manage and secure our systems. The second control states there should be a software inventory that contains the names and versions of the products for all devices within the infrastructure. The challenge for a large number of organizations is the ability to have accurate information available with minimal impact on tight IT budgets. This paper will discuss the Microsoft Windows command line tools that will gather this information, and provide example scripts that can be run by the reader.

# 1. Introduction

To understand and know what programs are installed on the computers within an organization, regular monitoring and analysis are needed. To accomplish this tools are required that can query the systems in question and provide detailed information to administrators.

## 1.1. Overview of the control

The goal of Control 2 of the Critical Security Controls is to "Actively manage (inventory, track, and correct) all software on the network so that only authorized software is installed and can execute, and that unauthorized and unmanaged software is found and prevented from installation or execution." (Center for Internet Security, 2015) Control 2.3 states that organizations should deploy inventory tools that track versions of the applications installed on the system in question (Center for Internet Security, 2015). In their 2016 Internet Security Threat Report, Symantec reports that 75% of legitimate web sites have vulnerabilities. Within the report, it also states that fifteen percent of websites have critical vulnerabilities that would be trivial to exploit (Symantec, 2016). It is the opinion of the author that the vast majority of these sites must not know these vulnerabilities are present or not know that the software installed leaves such a massive security gap in place.

## 1.2. Challenges collecting information

As with many IT implementations, there are numerous reasons why a specific piece of hardware or software does not get implemented. One problem that continually occurs is the lack of money to purchase and implement a solution. Budgets are limited, and there are always items that need to be purchased. If it were possible to start monitoring this control with minimal expense, it would permit faster adoption and implementation.

Secondly, the resources needed to manage any new software or hardware is a continued pressure point for organizations, as IT departments are perpetually looking for

staff (Security Magazine, 2012) (Ponemon Institute LLC, 2014). By providing a solution that does not depend heavily on personnel and automates as much as possible, it will help avoid an added burden on existing IT staff.

# 2. Installed Windows Programs

The Windows operating system provides numerous ways to accomplish the same task. For example, three different methods will be discussed for collecting the installed software information from the command line. These methods are

1) *Psinfo.exe*, a program provided by the Microsoft Sysinternals group;
2) Windows Management Instrumentation, or WMI; and
3) PowerShell.

The Windows operating system is also a challenging operating system, as depending on how the query is performed, the results may differ slightly. Specific details about each method's limitations will be discussed in the following sections.

## 2.1. Command line collection methods

For this paper, the primary criterion for selecting the gathering method was to use programs and practices that are quickly and readily available to the administrator. Also, it was desired to have a means that any level of reader – novice to expert - can employ. Finally, as cost is always a limiting factor for any organization, methods that were free were desired.

With these criteria in mind, this work examines various methods built into the operating system by Microsoft to provide the listing of installed programs. A further extension of this list was permitted with tools from Microsoft's Sysinternals used, as these tools are freely available and can to be used without a Graphical User Interface (GUI). The reasoning behind removing a GUI was to permit the tools to be scripted and automated as much as possible.

## 2.2. Microsoft programs – Psinfo.exe and Psinfo64.exe

The first method used to collect the installed applications on a Windows system was the Microsoft Sysinternals programs *psinfo.exe* and *psinfo64.exe*, for 32 and 64-bit systems respectively (Microsoft, 2016). The Sysinternals web site was created by the group to store and make available the advanced system utilities they created (Microsoft, 2016).

These two tools collect the information from the system registry keys and runs on Windows NT/2000 or higher. While a lot of information is available from these programs, only the installed applications information was needed for this work. Psinfo has an option, the *–s* applications switch, which will provide a listing of the installed programs for the computer that is queried (Russinovich, 2010). An example of how this program can be run is shown in Figure 1.

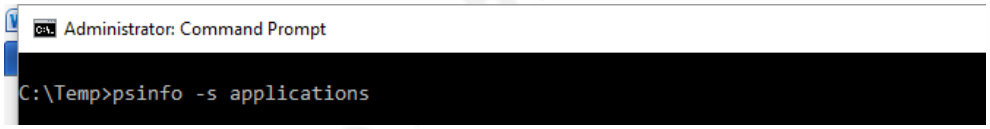


**Figure 1 - Psinfo command line**

The output of this tools a listing of applications and version numbers, illustrated in Figure 2.

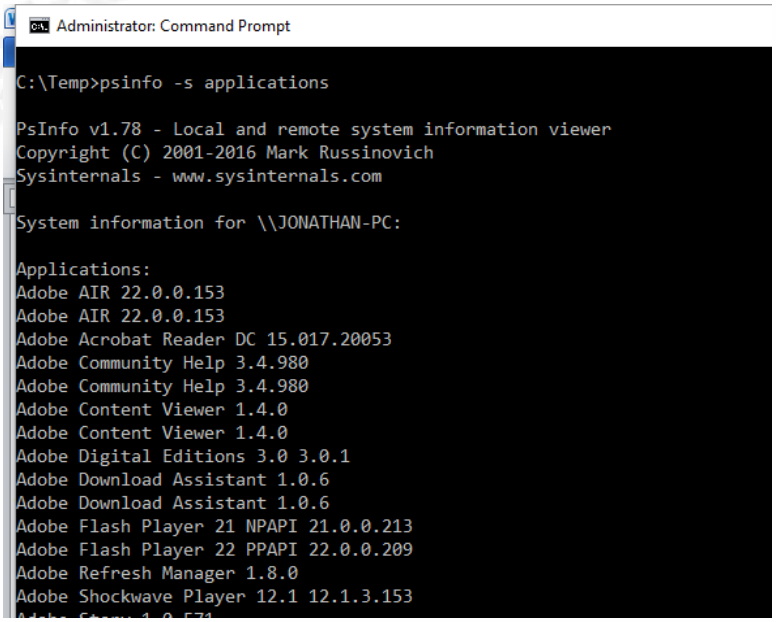


**Figure 2 - Psinfo example output**

Jonathan Risto;jonathan.risto@hotmail.com

Psinfo interrogates the *HKLM/System* registry key to get the listing of installed programs by using the Windows remote registry service (Russinovich, 2010). For the program to work, either on a local machine or on a remote system, the computer must have the remote registry service running for the tool to work.

### 2.2.1. Enabling remote registry

To quickly check if the remote registry service is running or not, from either the command prompt or the Windows run location on the start menu, type *services.msc,* as shown in Figure 3.

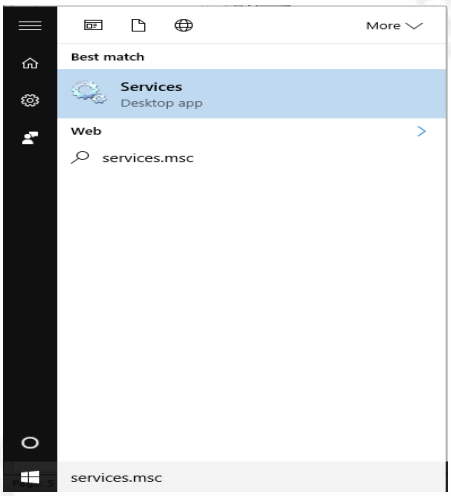


**Figure 3 – Launching services.msc**

Once the services program is started ensure that the items are sorted alphabetically, and then scroll down to the Remote Registry entry as shown in Figure 4.

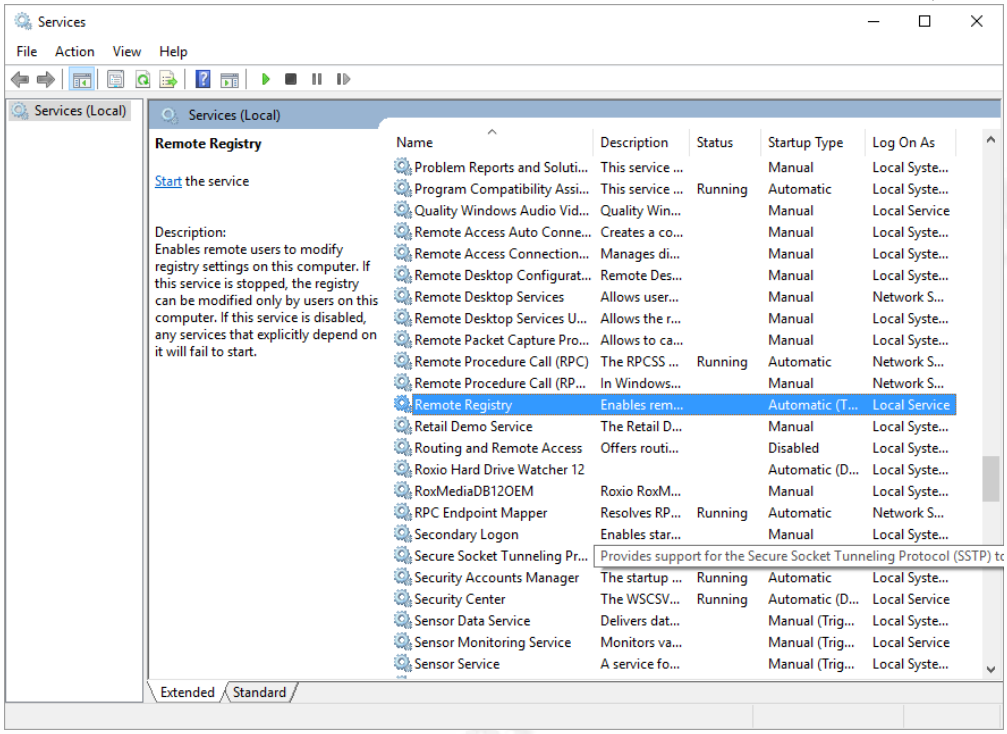


**Figure 4 - Remote registry service information**

Once the service is located, a double click will expand the entry, similar to Figure 5, and enable the service to be started if needed.

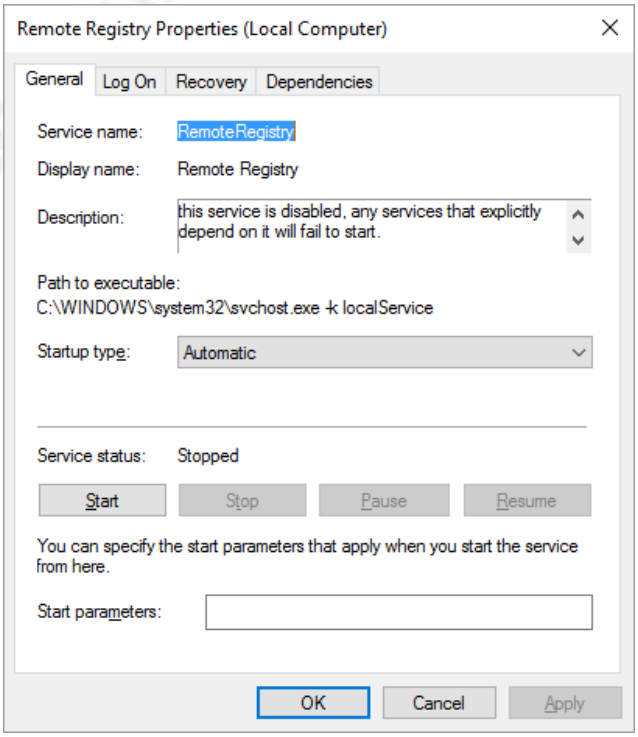


**Figure 5 - Service information display**

Jonathan Risto;jonathan.risto@hotmail.com

## 2.3. Windows Management Instrumentation

According to Microsoft, Windows Management Instrumentation (WMI) is their implementation of Web-Based Enterprise Management (WBEM), an industry initiative to standardize the collection of management information (Microsoft, 2016a). WMI is based on Common Information Model (CIM), put out by the Distributed Management Task Force (DMTF) (Wikipedia, 2016).

Within WMIC, which stands for Windows Management Instrumentation Command-line, numerous commands have been built into the tool to detail various settings and configurations of the computer (Morrison, 2012). Given that WMIC can be called remotely, an administrator can query any system under their control.

For this paper, the *product* option will be used. This command can return the following items: name, description, install date, vendor and version. Given that this work is interested in verifying the names and versions of software on the system, the *Name* and *Version* options will be used.

To put all these options together into a single command line entry is easy. To run the commands and options outlined above, it would need the following command:

*wmic product get name, version*

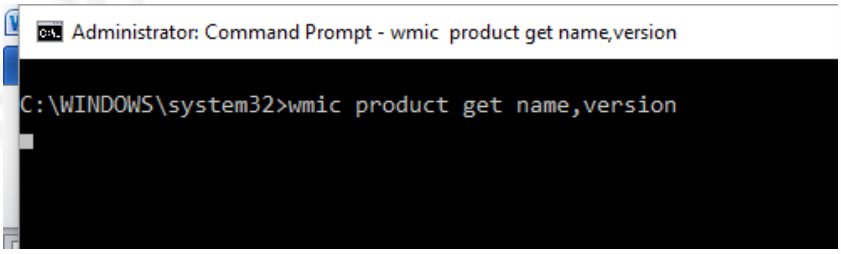


**Figure 6 - WMIC command line**

Running this command returns a large number of entries, with an example shown in Figure 7.

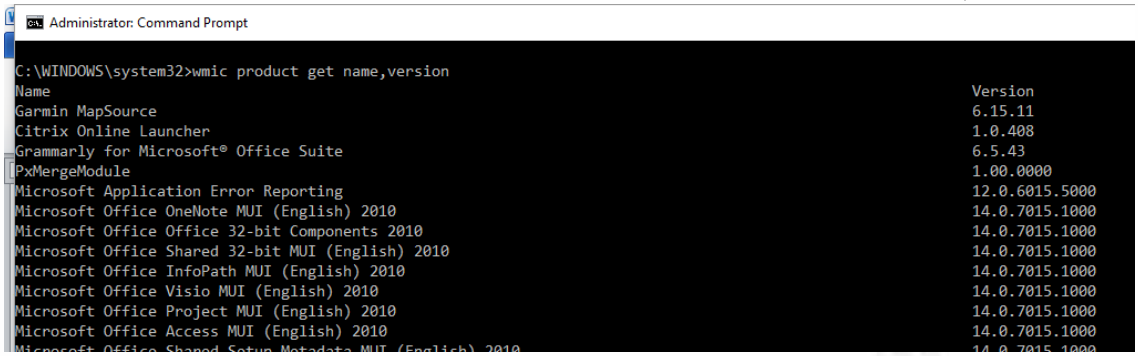


**Figure 7 - WMIC output**

## 2.4. PowerShell

PowerShell is a command line shell that Microsoft created to assist with the management of systems. It originated from Microsoft's Monad shell, which morphed into PowerShell v1 (Snover, 2007) (Snover, 2002). PowerShell provides access to a large number of built-in commands, called cmdlets, which are grouped together in modules within the environment. Details on the PowerShell modules and cmdlets are beyond the scope of this work, but information posted by Microsoft on these PowerShell topics can be found at (Microsoft, 2013).

The PowerShell functionality that is used in this paper is the ability to open a registry key location and read its contents. The two places that are accessed by PowerShell for this are:

*'HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\'*

and

*'HKLM\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\'*

Both of these registry locations store information about programs that have been installed on the system in so long as the programs register themselves correctly.

The problem, so to speak, with the registry, is that each program that accurately writes to this location creates this information in a new sub-key. An example of a sub-key is shown in Figure 8 and Figure 9. Within each of these Figures, it is possible to see that there is a significant number of sub-keys that need to be looked at to try and extract

the desired information of program name and version number. On the computer used for testing, there were 447 sub-keys located in *HKLM\SOFTWARE\Microsoft\Windows \CurrentVersion\Uninstall* alone. To extract the required information from each of the sub-keys, PowerShell needs to iterate through each potential sub-key and determine if there is a value present for *ProductName* and *DisplayVersion* fields in each registry entry.

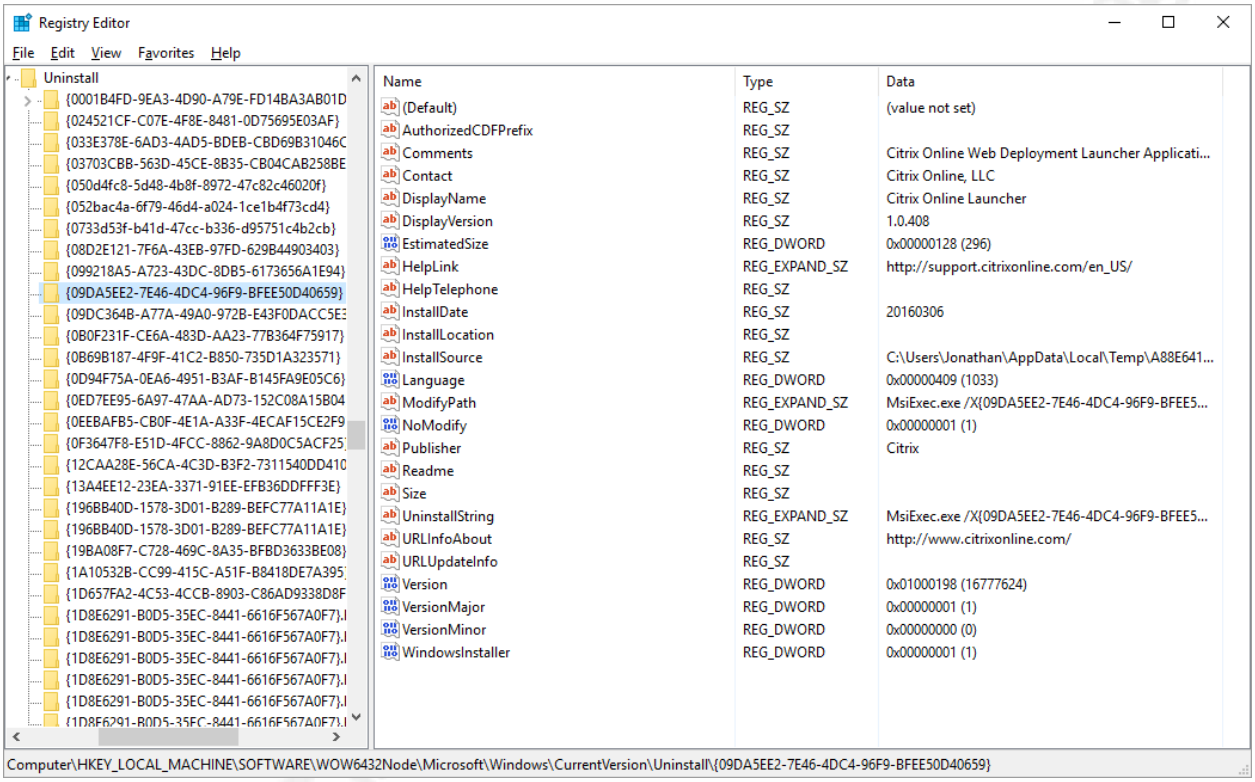


**Figure 8 - Registry value example in Software\WOW6432 location**

Not all sub-keys have the values being requested, which adds uncertainty to the requests being made. Examples without the Name value are shown in Figure 10, without the version information is provided in Figure 11 or without any of the required information is seen in Figure 12.

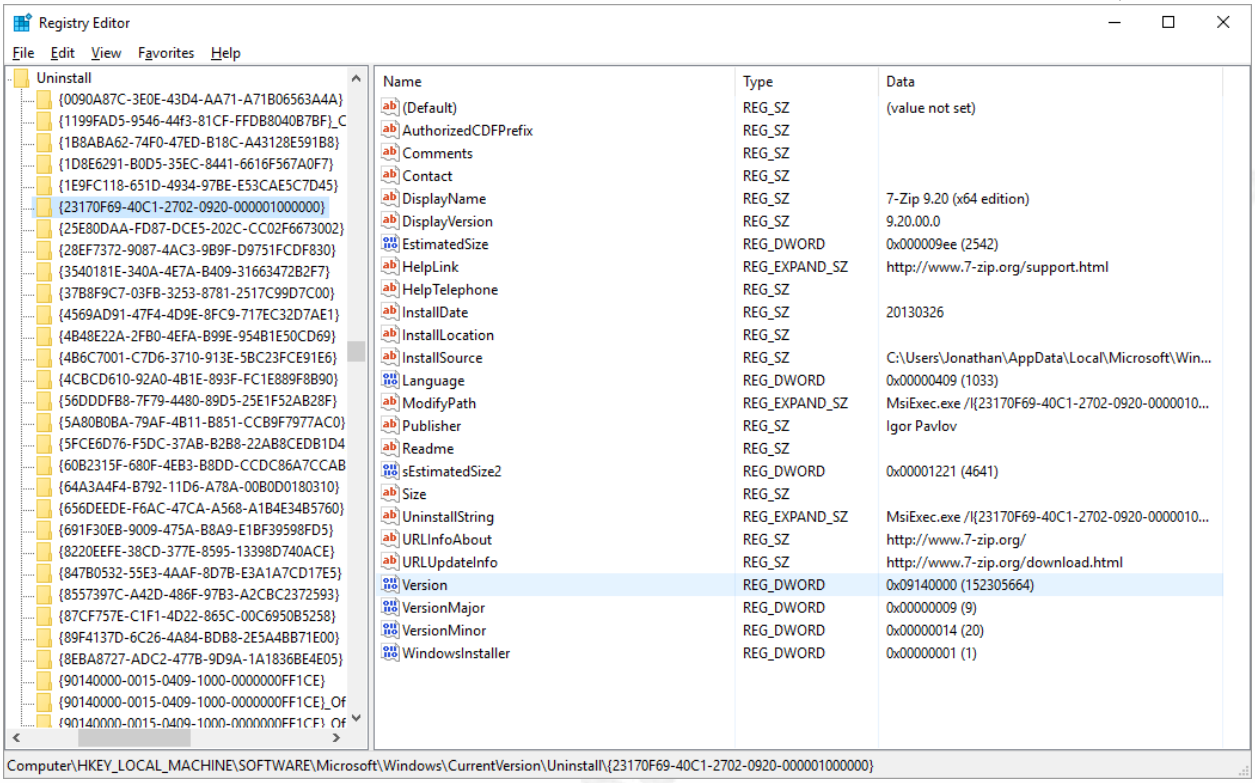


**Figure 9 - Registry example from the Software\Microsoft location**

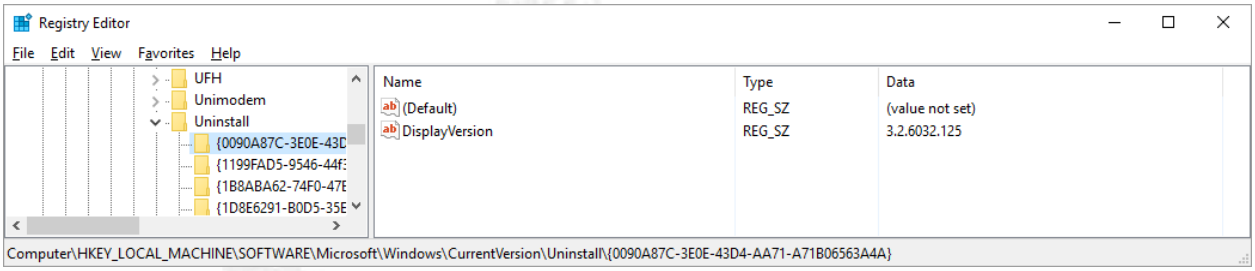


**Figure 10 - Example of a registry sub-key that does not have the DisplayName value**

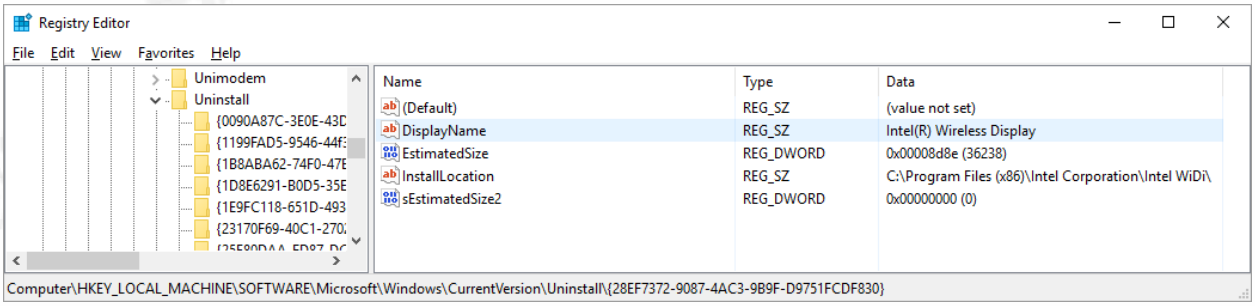


**Figure 11 - Example of a registry sub-key that does not have a ProductVersion value**

Jonathan Risto;jonathan.risto@hotmail.com

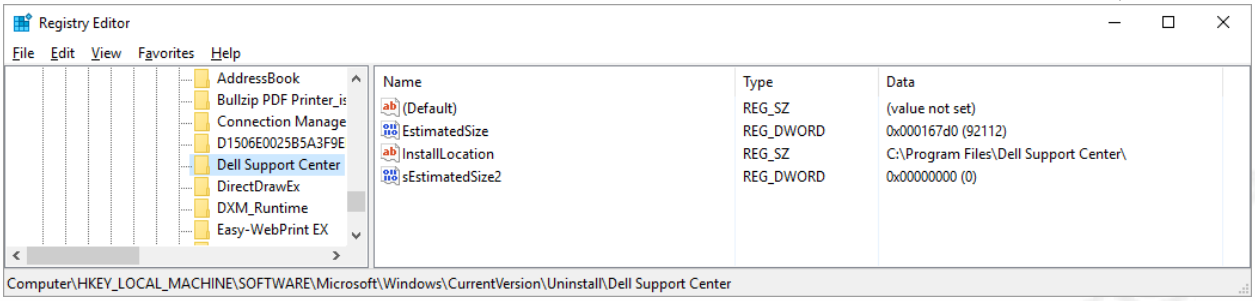


**Figure 12 - Example of a registry sub-key that does not have any required information**

With the examples and problems illustrated in Figures 5 through 9, the PowerShell commands used need to recognize and handle unknown values (the key names), as well as conditions when the desired values are not present (names of version). Contrary to WMIC or Psinfo collection methods, as both utilized built in functionality to extract the required information. Within PowerShell however, we are accessing the information directly, and therefore need to be able to handle any error and any abnormal conditions.

## 3. Putting it all together – Scripting

While most of the commands that have been discussed can be typed from the command prompt, it is much easier and creates fewer problems if the files are placed into a script that can be called quickly and repetitively. To that end, a script to simplify the collection process has been included in Appendix A through C.

Each of these scripts has automated the process to collect the installed programs from the three methods discussed previously. They automate the running of the commands. The scripts also request some user input, specifically the IP address of the computer that the user wishes to run the script. These scripts also save the output that can be viewed either from the command line or loaded into the program of the user's choice.

Some input error checking has been performed on the scripts, when possible. For example, within the PowerShell script, the user is forced to put the IP address in the correct format of ***x.x.x.x***. If another input other than a number is entered, the program requests the information again, as it is invalid input.

The scripts have been documented heavily, to permit the user to follow what the commands within are doing. If the code is from another source, this information is

recorded in the script, as well as the location where the code was obtained. For example, the *psinfo.exe* and *psinfo64.exe* commands generate some duplication in their outputs, either individually or collectively. A simple, but powerful, two-line sequence was found that iterates through two text files and removed any duplicate entries.

# 4. Conclusion

This paper has provided three scripts that collect the system information in three different manners. While the information obtained from each script is different, due to inconsistencies in information sources, consistently using the same means to collect the data is essential to enable comparisons.

Each of the scripts provided is interactive, as it requires the entry of the IP address of the system to be scanned. With minimal changes, an administrator could change all of the scripts to accept the IP address as part of the script call, thereby allowing the scripts to be scheduled when desired.

The PowerShell script collects the most extensive listing of information, and could arguably be stated the best script to use to enumerate the software installed on a Windows system. The details include information on operating system patches that have been applied to the system as well as software installed. Details regarding patches applied to Microsoft software are also contained within the PowerShell output.

The Psinfo script provides information on installed software from the registry, as well as information some of the patches installed on the system for Microsoft software. However, it does not contain operating system patches.

Using the WMIC collection method is by far the slowest collection method used within this paper. It does provide a detailed listing of software. However, it does not include any information on the operating system patches nor the Microsoft software patches.

Regardless of the script used, having a baseline of installed software is a crucial step in addressing the control. Corporate policies may prevent the use of one option or another, so this work does provide multiple means for administrators to obtain the details required to baseline the systems in question.

Jonathan Risto;jonathan.risto@hotmail.com

# References

Brumfield, K. (2014, January 19). *Simple way to temporarily bypass PowerShell execution policy*. Retrieved July 30, 2016, from Ken Brumfield's Blog: https://blogs.technet.microsoft.com/ken_brumfield/2014/01/19/simple-way-to-temporarily-bypass-powershell-execution-policy/

Center for Internet Security. (2015). *The CIS Critical Security Controls for Effective Cyber Defense.* Arlington: Center for Internet Security. Retrieved July 30, 2016

Microsoft. (2013, October 17). *Microsoft.PowerShell.Core Module*. Retrieved July 30, 2016, from Microsoft Developer Network: https://technet.microsoft.com/en-us/library/hh847840.aspx

Microsoft. (2016). *Microsoft Sysinternals.* Retrieved July 30, 2016, from Microsoft: https://technet.microsoft.com/en-us/sysinternals/default.aspx

Microsoft. (2016a). *About WMI.* Retrieved July 30, 2016, from Microsoft Developer Network: https://msdn.microsoft.com/en-us/library/aa384642(v=vs.85).aspx

Morrison, B. (2012, February 17). *Useful WMIC Queries.* Retrieved July 30, 2016, from Ask the Performance Team Blog: https://blogs.technet.microsoft.com/askperf/2012/02/17/useful-wmic-queries/

Ponemon Institute LLC. (2014, February). *Understaffed and at Risk: Today's IT Security Department.* Retrieved July 30, 2016, from Ponemon Institute: http://www.hp.com/hpinfo/newsroom/press_kits/2014/RSAConference2014/Ponemon_IT_Security_Jobs_Report.pdf

Russinovich, M. (2010, April 28). *PsInfo v1.77.* Retrieved July 30, 2016, from Windows Sysinternals: https://technet.microsoft.com/en-us/sysinternals/psinfo

Security Magazine. (2012, August 16). *Cyber Security -- Staffing*. Retrieved July 30, 2016, from Security Magazine: http://www.securitymagazine.com/articles/83412-study--63-percent-of-companies--it-departments-are-understaffed

Snover, J. (2002, August 8). Monad Manifesto. Retrieved July 30, 2016, from https://msdnshared.blob.core.windows.net/media/MSDNBlogsFS/prod.evol.blogs.msdn.com/CommunityServer.Components.PostAttachments/00/01/91/05/67/Monad%20Manifesto%20-%20Public.doc

Jonathan Risto;jonathan.risto@hotmail.com

Snover, J. (2007, March 18). *Monad Manifesto – the Origin of Windows PowerShell.* Retrieved July 30, 2016, from Windows PowerShell Blog: https://blogs.msdn.microsoft.com/powershell/2007/03/18/monad-manifesto-the-origin-of-windows-powershell/

Symantec. (2016, April). *Internet Security Threat Report, Volume 21.* Retrieved July 30, 2016, from Symantec: https://www.symantec.com/content/dam/symantec/docs/reports/istr-21-2016-en.pdf

Wikipedia. (2016, July 24). *Windows Management Instrumentation.* Retrieved July 30, 2016, from Wikipedia: https://en.wikipedia.org/wiki/Windows_Management_Instrumentation

Jonathan Risto;jonathan.risto@hotmail.com

# Appendix A

## PSinfo.exe and PSinfo64.exe script

For this script to work, the *psinfo.exe* and *psinfo64.exe* files must be in the *c:\temp* directory. To change this, any reference to *c:\temp* within the batch file must be modified to the new location of these files. When the script is run, it will request the IP address of the system to query.

The output of this script is also stored in the *c:\temp* directory. The output will be in the format of <IP address>_YEAR_MONTH_DAY_HOUR_MINUTE.txt. This format is to permit multiple collections from the same IP address to happen without overwriting the information, and to quickly provide a reference of when the script created the file.

To run this script, all that is needed is to call the *.bat* file from an administrative command prompt. The user's current credentials are passed along to the system being queried by the operating system. It has been assumed that the batch file is called psinfo.bat and has been placed in the *c:\temp* directory. The user must type the following command to invoke the batch file:

*C:\temp\psinfo.bat*

Jonathan Risto;jonathan.risto@hotmail.com

**CONTENTS OF BATCH FILE: psinfo.bat**

```
@ECHO OFF
cls
REM blank the screen to make it easier to follow what is happen

set /p computer=Enter the computer IP address you wish to query:
REM getting the user to enter the IP address of the system to be inventoried

for /F "tokens=1-4 delims=/ " %%i IN ('date /t') DO (
set DT_DAY=%%i
set DT_MM=%%j
set DT_DD=%%k
set DT_YYYY=%%l)
REM  the above lines date the output of the date comment and place the values into appropriate variables

for /F "tokens=1-4 delims=: " %%i IN ('time /t') DO (
set DT_hour=%%i
set DT_min=%%j)
REM  do the same for the time command

c:\temp\psinfo.exe -s applications \\%computer% >> C:\temp\%computer%.txt
c:\temp\psinfo64.exe -s applications \\%computer% >> C:\temp\%computer%.txt
REM above run the psinfo and psinfo 64 command, which return information on the system in question.

sort C:\temp\%computer%.txt >> C:\temp\%computer%_.txt
REM the above sorts the items in the file
```

```
REM  following two lines of code remove duplicate entries in a text file.
REM It was found at http://stackoverflow.com/questions/11689689/batch-to-remove-duplicate-rows-from-text-file

FOR /f "delims=" %%a IN (C:\temp\%computer%_.txt) DO SET $%%a=Y
(FOR  /F "delims=$=" %%a In ('set $ 2^>Nul') DO ECHO %%a)>C:\temp\%computer%_%DT_YYYY%%DT_MM%%DT_DD%%DT_hour%%DT_min%.txt

del %computer%_.txt
del %computer%.txt
REM remove temp files
```

# Appendix B

## WMIC script

For this script to work, it does not require any additional information to invoke it. All that is needed is the *.bat* file to be stored on the disk. When the batch file runs, it will request the IP address from the user. No other input is needed. It will query the remote PC using the credentials that invoked the program.

The output of this script is also stored in the *c:\temp* directory. The output will be in the format of <IP address>_<YEAR><MONTH><DAY><HOUR><MINUTE>.txt. For example, when the program is run it would create a file called *172.19.5.9_201607301134.txt* that contains all of the output from the command.

This naming convention was done to permit multiple collections from the same IP address to happen without overwriting the information, and to quickly provide a reference of when the file was created.

To change this setting, and reference to c:\temp found within the script must be modified.

To run this script, just type the following command at an administrative command prompt. The following command example assumes that the batch file is saved in the *c:\temp* directory and is called *wmic_query.bat*.

*c:\temp\wmic_query.bat*

Jonathan Risto;jonathan.risto@hotmail.com

**CONTENTS OF BATCH FILE: wmic_query.bat**

```
@echo off
cls
set /p computer=Enter the computer IP address you wish to query
REM getting the user to enter the IP address of the system to be inventoried

for /F "tokens=1-4 delims=/ " %%i IN ('date /t') DO (
set DT_DAY=%%i
set DT_MM=%%j
set DT_DD=%%k
set DT_YYYY=%%l)
REM  the above takes the date command output and populates the appropriate variables

for /F "tokens=1-4 delims=: " %%i IN ('time /t') DO (
set DT_hour=%%i
set DT_min=%%j
)
REM  the above does the same as the above date process, but for the time command

echo.
echo Starting the query. This may take a minute or two. Be patient.
REM Feedback to the user so they know what is happening

wmic /node:"%computer%" /OUTPUT:C:\temp\%computer%_%DT_YYYY%%DT_MM%%DT_DD%%DT_hour%%DT_min%.txt product get name,version
echo.
echo The file C:\temp\%computer%_%DT_YYYY%%DT_MM%%DT_DD%%DT_hour%%DT_min%.txt was written
echo.
```

```
29
30  REM  Run the WMIC command that uses the provided address and saves the file with machine IP and date/time as the filename in C:\temp\
```

# Appendix C

## PowerShell script

For this script to work, it does not require any additional information to invoke it. All that is needed is the *.ps1* file to be stored on the disk. When the command is run, it will request the IP address from the user. No other input is needed. It will query the remote PC using the credentials that invoked the program.

The output of this script is also stored in the *c:\temp* directory. The output will be in the format of <IP address>_<YEAR><MONTH><DAY><HOUR><MINUTE>.txt. For example, when the program is run it would create a file called *172.19.5.9_201607301134.txt* that contains all of the output from the command.

This naming convention was done to permit multiple collections from the same IP address to happen without overwriting the information, and to quickly provide a reference of when the file was created.

To change this setting, and reference to *c:\temp* found within the script must be modified.

To run this script, just type the following command at an administrative command prompt. The following command example assumes that the *ps1* file is saved in the *c:\temp* directory and is called *software_enumeration.ps1*.

*powershell -executionpolicy bypass –file software_enumeration.ps1*

The *-executionpolicy* bypass option is used to ensure that the script will run even if the security policy is configured to block PowerShell scripts. This exception allows the scripts to run and lowers the setting only for that PowerShell session, and not the entire system configuration (Brumfield, 2014).

Jonathan Risto;jonathan.risto@hotmail.com

**CONTENTS OF POWERSHELL SCRIPT: software_enumeration.ps1**

```
<#
This script to ask the user to provide the IP address they wish to query, and then runs the PowerShell commands to get the installed software on the
Windows system in question.

The output will be saved to a file called c:\temp\<IP ADDRESS>_<YEAR><MONTH><DAY><HOUR><MINUTE>.txt

The first portion of this PS script contains a function named Get-RemoteProgram authored by Jaap Brasser.
It polls through the registry keys to extract the required information
and is posted on Microsoft's Technet at https://gallery.technet.microsoft.com/Get-RemoteProgram-Get-list-de9fd2b4

Small modifications were needed to this code to make it work as necessary for this project.

Where the code was modified from original, comment lines start with JR show any new information within the function
#>
Function Get-RemoteProgram {
<#
.Synopsis
Generates a list of installed programs on a computer
```

```
21
22          .DESCRIPTION
23          This function creates a list by querying the registry and returning the installed programs of a local or remote computer.
24
25          .NOTES
26          Name: Get-RemoteProgram
27          Author: Jaap Brasser
28          Version: 1.2.1
29          DateCreated: 2013-08-23
30          DateUpdated: 2015-02-28
31          Blog: http://www.jaapbrasser.com
32          #>
33              [CmdletBinding(SupportsShouldProcess=$true)]
34              param(
35                  [Parameter(ValueFromPipeline=$true,
36                      ValueFromPipelineByPropertyName=$true,
37                      Position=0)]
38                  [string[]]
39                      $ComputerName = $env:COMPUTERNAME,
40                  [Parameter(Position=0)]
```

Jonathan Risto;jonathan.risto@hotmail.com

```
41      [string[]]$Property
42  )
43
44  begin {
45      $RegistryLocation = 'SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\',
46                          'SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall\'
47      $HashProperty = @{}
48                          #JR  original line follows
49      #$SelectProperty = @('ProgramName','ComputerName')
50                          #JR  modified line follows
51                           $SelectProperty = @('ProgramName','DisplayVersion')
52                           if ($Property) {
53          $SelectProperty += $Property
54      }
55  }
56
57  process {
58      foreach ($Computer in $ComputerName) {
59          $RegBase = [Microsoft.Win32.RegistryKey]::OpenRemoteBaseKey([Microsoft.Win32.RegistryHive]::LocalMachine,$Computer)
60          foreach ($CurrentReg in $RegistryLocation) {
```

```
61          if ($RegBase) {
62             $CurrentRegKey = $RegBase.OpenSubKey($CurrentReg)
63             if ($CurrentRegKey) {
64                $CurrentRegKey.GetSubKeyNames() | ForEach-Object {
65                   if ($Property) {
66                      foreach ($CurrentProperty in $Property) {
67                         $HashProperty.$CurrentProperty = ($RegBase.OpenSubKey("$CurrentReg$_")).GetValue($CurrentProperty)
68                      }
69                   }
70                  #JR commented out following line - not needed for this work
71                   #$HashProperty.ComputerName = $Computer
72                   $HashProperty.ProgramName = ($DisplayName = ($RegBase.OpenSubKey("$CurrentReg$_")).GetValue('DisplayName'))
73                   #JR added following new line as needed for this work
74                   $HashProperty.DisplayVersion = ($DisplayVersion = ($RegBase.OpenSubKey("$CurrentReg$_")).GetValue('DisplayVersion'))
75
76                   if ($DisplayName) {
77                      New-Object -TypeName PSCustomObject -Property $HashProperty |
78                      Select-Object -Property $SelectProperty
79                   }
80             }
```

```
                }
            }
        }
    }
  }
}

clear-host
#blanking the screen to make it easy to see output
$ip_resp = ''
$ip_addr= ''
#blank out the variable to ensure it is empty
DO
        {DO
                {
                $ip_addr = Read-Host 'Input the IP address of the systems you wish to scan. E.G. 10.11.12.13'
                } while ($ip_addr -notmatch '\p{Nd}+\.\p{Nd}+\.\p{Nd}+\.\p{Nd}+')

        #the above forces the user to user numbers in the format we want of x.x.x.
        write-host 'You entered' $ip_addr ', is this correct?
```

```
101             $ip_resp = Read-Host 'Y or N'
102             } #end of DO
103     Until ($ip_resp -eq "Y" -OR ($ip_resp -eq "y"))
104     #the above loops asking the same question until the user enters Y or y
105
106     $a = Get-Date
107     # get date/time information from the built in powershell command
108
109     $path = 'C:\temp\'
110     #change the above location to where you wish to have the output files stored
111     $filename = $path + $ip_addr +'__'+$a.Year+$a.Month+$a.Day+$a.Hour+$a.Minute+$a.Second+'.txt'
112     #creates the filename to be stored, by combining information and referencing sub values of the Get-Date output
113
114     Get-RemoteProgram -ComputerName $ip_addr| Out-File $filename
115     #the above calls the 3rd party function and writes out the contents to the filename specified
116
117     write-host ‘Collection complete. File ‘ $filename ‘ written
```