Global Information Assurance Certification Paper

Auditing Windows installed software through command line scripts

GIAC (GSNA) Gold Certification

Author: Jonathan Risto, jonathan.risto@hotmail.com
Advisor: Kees Leune
Accepted: September 7, 2016

Abstract

The 20 Critical Controls provides guidance on managing and securing our networks. The second control states there should be a software inventory of the products for all devices within the infrastructure. Within this paper, the auditor will be enabled to compare Windows system baseline information against the currently installed software configuration. Command line tools utilized will be discussed and scripts provided to simplify and automate these tasks.

1. Introduction

For any computer system, to successfully audit the software installed two items are required. The first being a baseline of the approved software available on the system in question, and the second being an accurate and current listing of programs for the same computer. While this is an easy statement to make, it is significantly harder to accomplish.

1.1. Control 2 – overview

The Critical Securit Controls, Control 2 require an organization 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). Within control 2, subcontrol 2.3 states an organization needs to deploy a software inventory system that tracks the approved software installations and notifies of any deviations from the approved baseline (Center for Internet Security, 2015).

However, there are challenges faced by all IT Security groups, budgets being one of them. With unlimited funds, it would be easy to implement a commercial solution that could easily resolve this control and provide the required details. However, resources are not unlimited, and not all organizations, especially those in the small and medium markets, cannot always afford commercial solutions.

Another limiting factor for the majority of organizations is the lack of qualified resources (Security Magazine, 2012). Without the resources to a manage new software installations, organizations either install a solution that is not used or more likely, are unable to install due to this limitation.

Jonathan Risto;jonathan.risto@hotmail.com

2. Auditing the control

To successfully audit the software inventory control, the first item that is needed is a reliable baseline of the system in question. Without accurately knowing what software is supposed to be on the system being audited, it is impossible to know if there are any changes from this baseline.

2.1. Collecting the information

There are numerous means available to collect the information needed. Commercial tools and publically available methods are accessible to the reader. However, for this work, a public and freely available collection process was necessary to ensure relevance and accessibility for the majority of users. To this end, the previously published work on collecting the software inventory on Windows systems was chosen. In this paper, free tools and command line methods are presented, as well as automated scripts are provided to the reader to automate the process are provided (Risto, 2016).

These scripts provide the baseline information for the systems under audit. All three collection methods were used to test the auditing and automation provided in this work.

3. Comparisons with PowerShell

PowerShell was created by Microsoft to assist with the management of operating system. It provides access to built-in commands, called cmdlets, grouped into modules within PowerShell. An in-depth description of PowerShell is outside the scope of this paper. However, Microsoft has a significant amount of information on PowerShell posted at (Microsoft, 2013).

Within PowerShell, one of the cmdlets that is available is Compare-Object, which is the command that used in this work. Detailed information on the command and all the options available can be found at both (SS64, 2015) and (Microsoft, 2016). For a basic comparison of two items, only the two elements to be compared are needed. The Compare-Object documentation calls these the referenceObject and the differenceObject

(Microsoft, 2016). Both of these need to be PowerShell reference objects. The command required to run the Compare-Object command is as follows:

```
Compare-Object $referenceObject $differenceObject
```

For this work, we require the Compare-Object command to compare two files. However, within the cmdlet, there is no direct means to compare two files. However, within PowerShell, there is another cmdlet that reads the contents of a file, which is the Get-Content cmdlet. This cmdlet permits the reading of a file, among other functions (Microsoft, 2016a). Using this functionality within the Compare-Object command is completed as follows:

```
Compare-Object $(Get-Content c:\temp\software_baseline.txt) $(Get-
Content c:\temp\new_software_listing.txt)
```

Within this command, the `$(Get-Content c:\temp\software_baseline.txt)` portion reads in the contents of the c:\temp\software_baseline.txt file and uses this as the reference object for the command. In a similar fashion, the `$(Get-Content c:\temp\new_software_listing.txt)` section of the command reads the content of the c:\temp\new_software_listing.txt and uses this as the difference object. When the command is run, the output is displayed on the screen, and it is similar to that shown in Figure 1. More details and information on the format of this output is contained in Section 3.1Compare-Object output format.

Figure 1 - Compare-Object output example

3.1. Compare-Object output format

When the PowerShell compare-object cmdlet is run, it provides an output that details differences between the two files as well as which file contained the line. The first portion of a baseline file is shown in Figure 2.

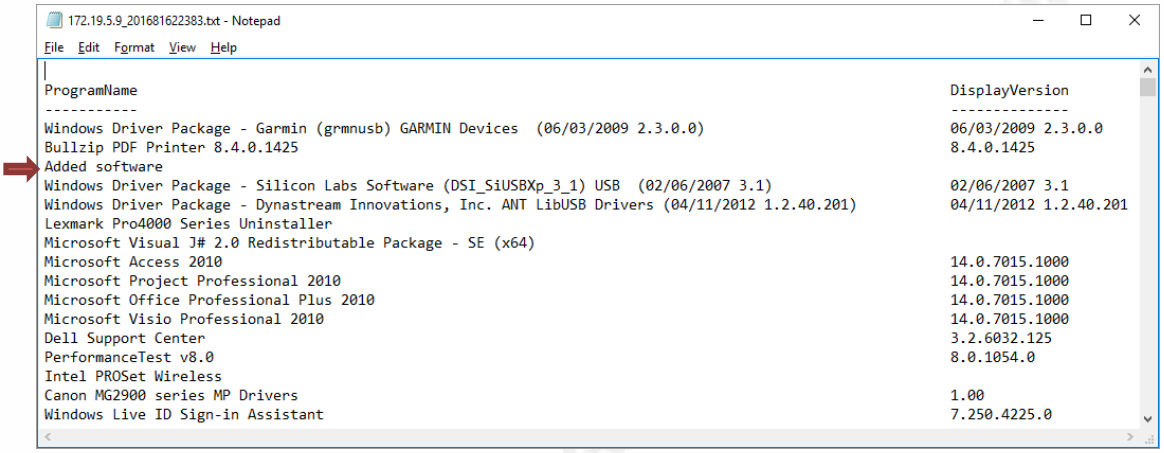


Figure 2 - PowerShell installed software baseline file example

Figure 3 shows the currently installed software as saved when the script was run. As you can note, both are very similar in listed software and version numbers of the portion of the file shown.

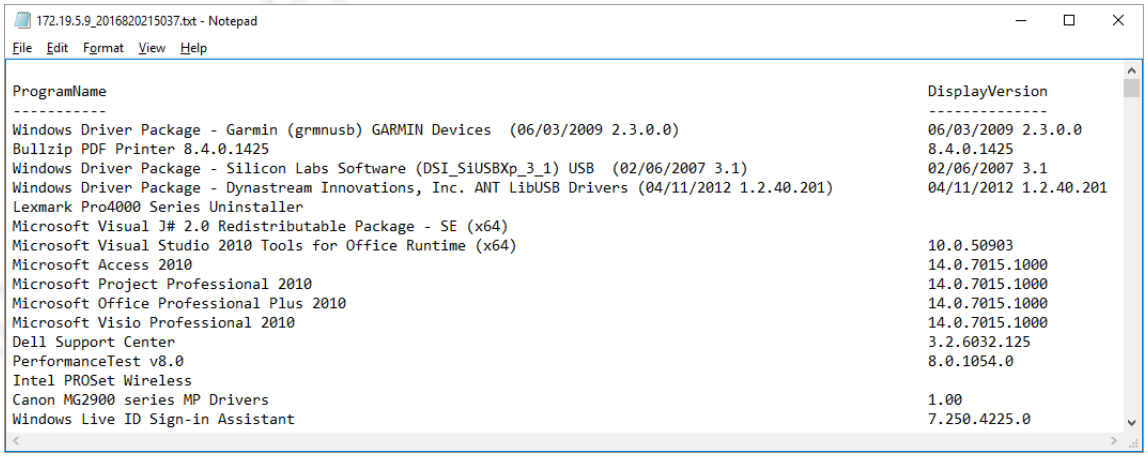


Figure 3 - PowerShell current installed software example

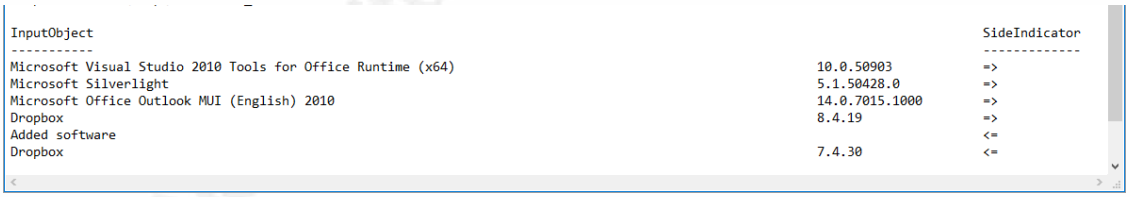
To ensure that the new software inventory would include additional items that were not in the baseline, the line Added Software was inserted into the file, as highlighted in Figure 2 by the arrow on the left-hand side. This entry was added to show how the

comparison outputs information on a piece of software missing from the current inventory.

As well, three lines of software were deleted from this baseline file. The deleted lines were for the programs Microsoft Visual Studio 2010 Tools, Microsoft Silverlight, and Microsoft Office Outlook MUI (English) 2010. The deletion of these lines was done to show how the system outputs the information of new items in the current software inventory file.

An additional piece of software was updated on the system in question, to show how the comparison outputs information for a similar situation.

When the comparison is run within PowerShell, the output created looks analogous to the output in Figure 4. In this output, the PowerShell command provides the name and versions of each piece of software that is different in the baseline or current software inventory files. The cmdlet also provides a column for the title of SideIndicator; that shows, albeit somewhat cryptically, in which of the two files the changes are located.



InputObject		SideIndicator
Microsoft Visual Studio 2010 Tools for Office Runtime (x64)	10.0.50903	=>
Microsoft Silverlight	5.1.50428.0	=>
Microsoft Office Outlook MUI (English) 2010	14.0.7015.1000	=>
Dropbox	8.4.19	=>
Added software		<=
Dropbox	7.4.30	<=

Figure 4- PowerShell comparison output example

Within the sample output of Figure 4, the SideIndicator column contains either a => or a <= symbol. What these are cryptically detailing to the person running the script is that in the case of <=, that the line was only contained within the baseline file used in the comparison. If a => is shown, this means that the line is only contained within the current inventory file.

To be able to understand how the output relates back to situations outlined previously, some further analysis is needed. Four test cases are required. First, an application only appears in the baseline file. Secondly, software is only listed in the latest inventory file. Thirdly, a program exists in both files and is the same version number.

Jonathan Risto;jonathan.risto@hotmail.com

Moreover, lastly, a piece of software exists in both files but with different version numbers in each file.

The first test case is illustrated in the output from the comparison by the added software depicted in Figure 2, with the arrow beside it. By adding a new line of this file, it is the equivalent of having a piece of software that appears only in the baseline file, our first test case. The results of this can be seen in the fifth line of Figure 4, where the Added Software is shown with an <= symbol. This symbol indicates that the filename is only found in the baseline file, and not the recently gathered inventory.

The second test case, information is only contained in the latest inventory file, is illustrated by the first three lines in Figure 4. These specific lines were deleted from the baseline file and therefore were only contained in the latest software inventory completed on the system. The output file lines show the => symbol, which graphically shows this information.

The third test case is trivial in nature, as there are numerous examples of this within the comparison files. As shown in Figures 2 and 3, multiple items appear in both files that are identical, and within the output file are not listed. So by having them listed in both files, the comparison operation does not include them in the output file.

The fourth test case is the most complex one. It requires the comparison to examine the entire entry of each line, not just a partial match of program names. An example of this is shown in Figure 5 and Figure 6. These two excerpts are from further down the baseline file and the current inventory file than previously shown. Both highlight that Dropbox exists in the files, but the version number is different. The comparison output shows this in lines four and six, with line 4 indicating the version associated with the current inventory with the symbol of => and line 6 indicating the version related to the baseline file, and provides the graphical indication of where the <= located this.

Jonathan Risto;jonathan.risto@hotmail.com

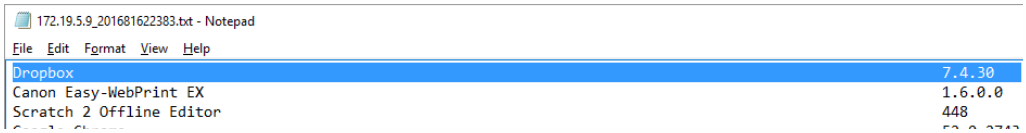


Figure 5 – Dropbox software version in baseline file

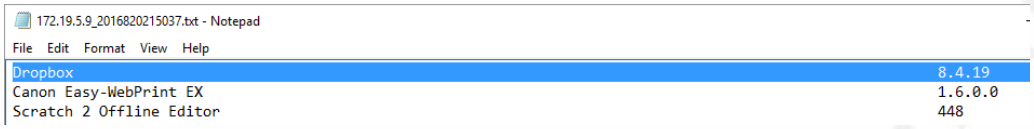


Figure 6 – Dropbox software version in current inventory file

The comparison in test case four correctly identifies that there are differences in the files, and shows which difference is located in the correct files. In an audit, this would either highlight that the baseline file has not been kept updated, or that a piece of software has been upgraded without permission on the computer in question.

4. Comparisons through Windows command line

Within the Windows operating system components, there is a program c:\windows\system32 directory call fc.exe. What this name is short for File Compare, and is detailed by Microsoft at (Microsoft, 2016b). The command has numerous options, but for this paper, minimal optional parameters are needed. By default, the program compares the two files as ASCII sources, which for this work is perfect. The only switch that used is the /w option.

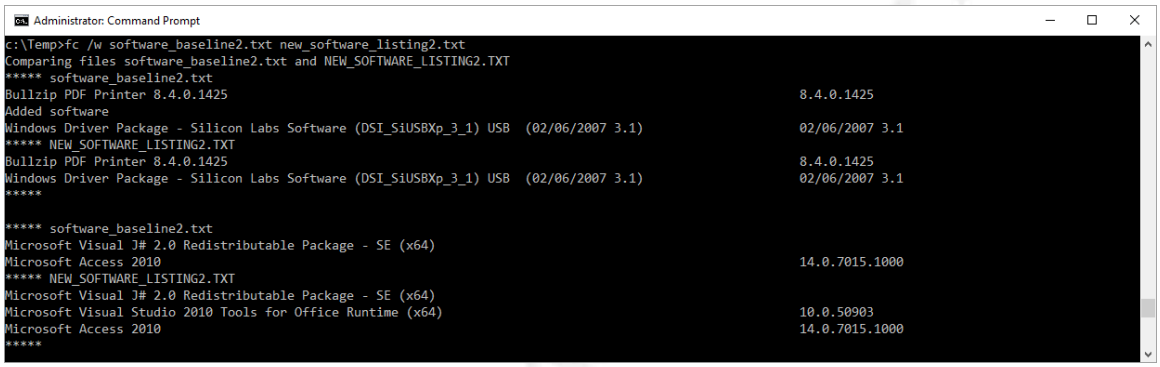
The /w option instructs the program to compress any multiple whitespace characters that are found together and treat them as a single space, as well as ignore the whitespace at the beginning and the end of the file (Microsoft, 2016b). Given that the creation of the file is based on programs outside of the control of the user, it cannot be guaranteed how the formatting appears, nor if there are blank lines at the start and end of a file. With this option enabled, the issues and concerns regarding blank lines within the two comparison files are removed.

To run the command as needed for this paper, the following command is used:

```
fc /w c:\temp\software_baseline.txt c:\temp\new_software_listing.txt
```

in the above command, the `c:\temp\software_baseline.txt` specifies the location of file filename 1, which is the baseline file for the comparisons. The `c:\temp\new_software_list.txt` option specifies the location of filename 2, which is the current inventory file in this paper.

Once the command above is run, it generates an output similar to that shown in Figure 7.



Administrator: Command Prompt
c:\Temp>fc /w software_baseline2.txt new_software_listing2.txt
Comparing files software_baseline2.txt and NEW_SOFTWARE_LISTING2.TXT
***** software_baseline2.txt
Bullzip PDF Printer 8.4.0.1425 8.4.0.1425
Added software
Windows Driver Package - Silicon Labs Software (DSI_SiUSBXp_3_1) USB (02/06/2007 3.1) 02/06/2007 3.1
***** NEW_SOFTWARE_LISTING2.TXT
Bullzip PDF Printer 8.4.0.1425 8.4.0.1425
Windows Driver Package - Silicon Labs Software (DSI_SiUSBXp_3_1) USB (02/06/2007 3.1) 02/06/2007 3.1
*****
***** software_baseline2.txt
Microsoft Visual J# 2.0 Redistributable Package - SE (x64)
Microsoft Access 2010 14.0.7015.1000
***** NEW_SOFTWARE_LISTING2.TXT
Microsoft Visual J# 2.0 Redistributable Package - SE (x64)
Microsoft Visual Studio 2010 Tools for Office Runtime (x64) 10.0.50903
Microsoft Access 2010 14.0.7015.1000
*****

Figure 7 - Output example of the fc.exe command

The details regarding how to decipher this output are found in Section 4.1 fc.exe Output format.

4.1. fc.exe Output format

Once the fc.exe command has been run, when differences are found within the files, it generates an output similar to that seen in Figure 8. This output was generated by running the following command:

```
fc /w software_baseline2.txt new_software_listing2.txt
```

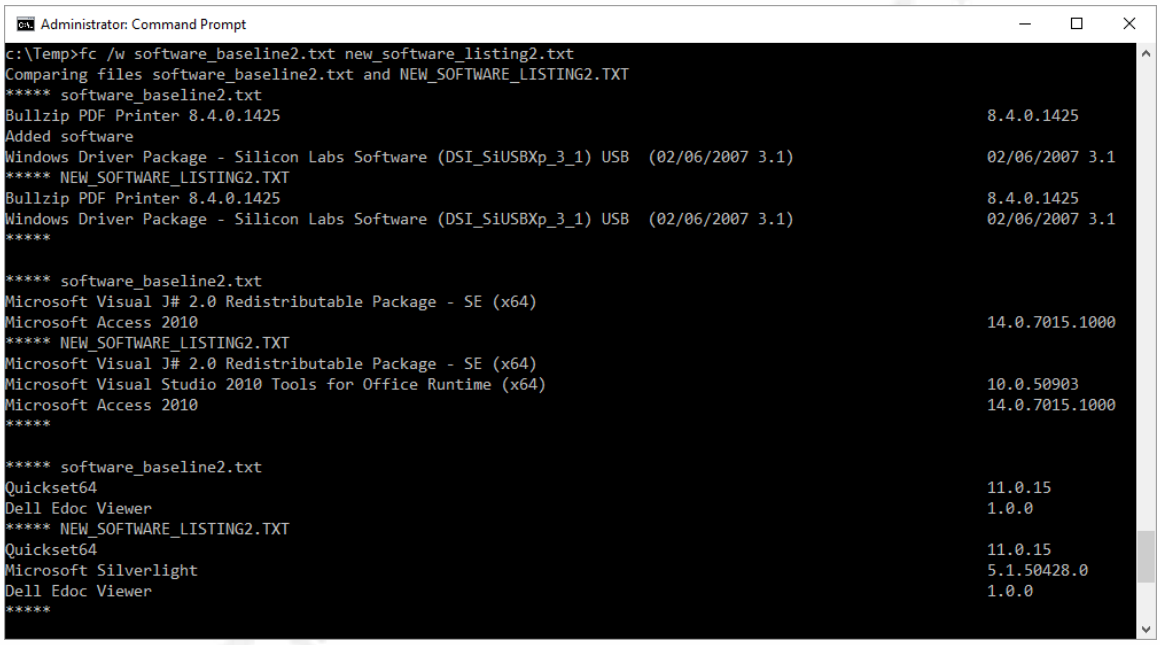


Figure 8 - fc.exe output example

The some of the contents of software_baseline2.txt are shown in Figure 9. This picture is very similar to Figure 2, however, in this case, the file is an ASCII file, as it was processed through the type command before being displayed. This filtering was done to ensure that no Unicode characters were present in the file. Similar actions were taken to create the file `c:\temp\new_software_listing2.txt`, which is shown in Figure 10.

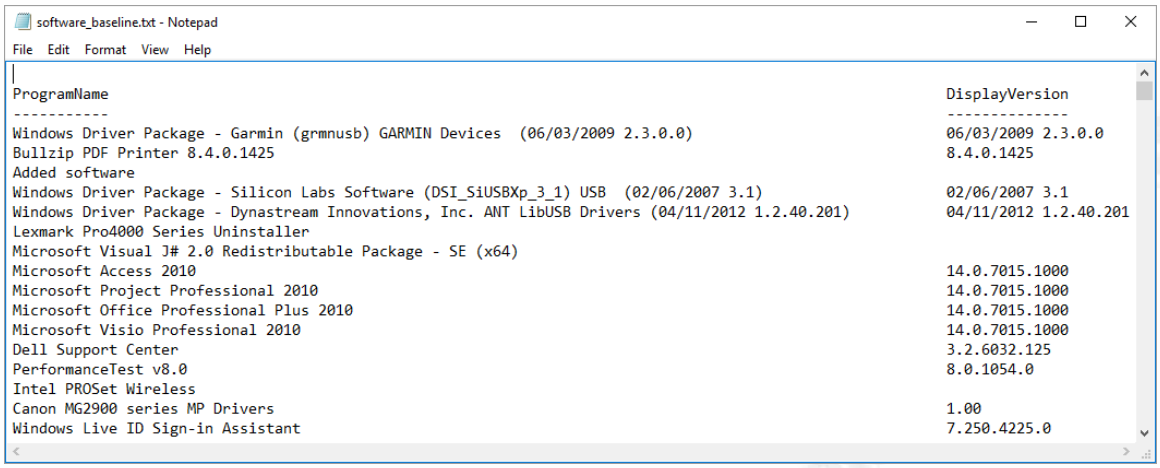


Figure 9 - Contents of software_inventory2.txt file

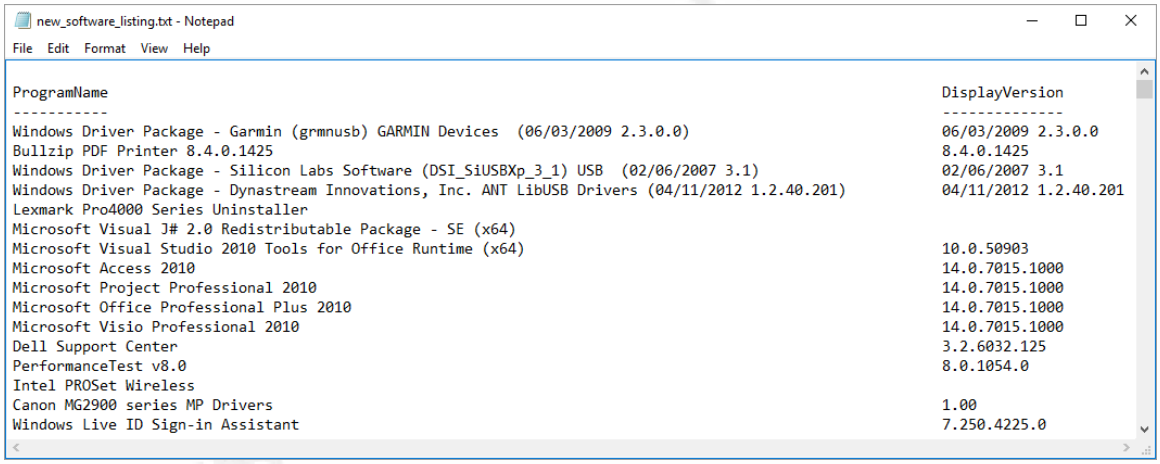


Figure 10- Contents of the new_software-listing.txt file

When the fc.exe program finds a difference between two items it is comparing, it lists the differences discovered in the two files, in a somewhat confusing fashion. In Figure 8, it lists as one of the first lines

```
***** software_baseline2.txt
```

This wording indicates the file from which that the lines come. Next is a listing of items from the file. In this example, it is the following:

```
Bullzip PDF Printer                    8.4.0.1425
Added software
Windows Driver Package - Silicon Labs Software (DSI_SiUSBXp_3_1) USB
(02/06/2007 3.1)                       02/06/2007 3.1
```

Jonathan Risto;jonathan.risto@hotmail.com

This listing shows the program names from the file, as well as the version numbers in two columns. The reason these items are listed is that there was a difference between this listing and the listing in the compared listing.

Next, the output lists the next file used in the comparison, in this case, the file shows the following `***** NEW_SOFTWARE_LISTING2.TXT`. The lines that follow this show the lines from the file that differ from the lines in the baseline file. In this example, it shows

```
Bullzip PDF Printer                 8.4.0.1425
Windows Driver Package - Silicon Labs Software (DSI_SiUSBXp_3_1) USB
(02/06/2007 3.1)                    02/06/2007 3.1
```

The difference in the two listings is that there is an additional line in the baseline file. It is the line of `Added Software`. The fc program lists the relevant sections of each file, and the reader is left to see what the difference is. The same format is repeated for each difference found in the files, in the same format. At the end of each listing of, the program places `*****` on a new line to show this is the end of the section in question.

5. Automation - scripting it all together

The items that have been discussed in sections 3 and 4 of this paper outline the commands and how they work, but those commands as shown are run in isolation and do not automate the control in any way. It is excellent to have a means that can perform the work for the administrator, but what is needed is a simple way to have the functionality executed by an administrator. As part of this paper, two scripts have been created, one a PowerShell script and one a command line batch script.

5.1. PowerShell script

PowerShell has many features and commands available within it that help simplify scripting in that language. From simple comparisons to checking the existence of a file, their presence contribute to improving the value and use of the script created.

The PowerShell script is provided in Appendix B in its entirety and is not reproduced here. In this section, an overview of the functionality that is used within the script is detailed.

When the script runs, the first thing it asks for input is the baseline file location. Once this information is entered, the script checks to ensure that file exists. This check is done using the *if (![System.IO.File]::Exists($orig_file))* commands as shown on line 5 of the script. This statement calls the built-in function and method to check if the file entered by the user exists.

If the file does not exist, the program then enters the *do* loop in line 8 and informs the user that the file as entered does not exist, and asks for the filename again. A valid filename must be provided to exit the loop.

Once a valid filename has been provided, the script continues and asks for the collection method desired. The choice should be the same way as the baseline file was created. If not, a comparison between the two files is similar to comparing an apple and an orange. The program checks to ensure that a valid choice has been entered, and if not, it goes into the loop on line 19 and informs the user of a bad entry and stays in that loop until a valid entry is detected.

Once the method is chosen, the program runs the correct script as discussed in Section 2.1. Within each of scripts, the user is requested to enter the IP address for the system being audited to collect the current information from that system. This information is stored in the c:\temp directory.

The script then, in lines 41 thru 43 determines what the newest file is within the c:\temp directory, and stores this information. Given that the program just saved out the latest information from running the wmic, PowerShell or batch script, this is the most up-to-date information on the system being audited.

Jonathan Risto;jonathan.risto@hotmail.com

At this point, the script asks for where the output from the comparison is to be stored. The comparison is then performed between the baseline file and the newly collected information, with the information stored at the input location. This comparison is performed using the

Compare-Object $(Get-Content $orig_file) $(Get-Content $new_filename) | out-file -filepath $out_file –append

command as discussed in Section 3. The output is the same as described in Section 3.1 and is stored where the user has specified.

5.2. Command line script

Windows batch line scripting provides the ability to run numerous commands together, while also some of the core functionality of if constructs, variables and goto statements.

Once run, the batch file asks where the baseline file is located, and checks to see that the entry provided was not blank. This test is performed at line 13 of the script in the if statement. If the entry was blank, the program goes back to the start and requests the information again. The if construct from lines 17 thru 27 checks to see if the entered file exists on the system. If it does, the program then jumps to the next section of the program. If not, the user is informed that the file entered does not exist, and is asked to provide the information again.

The script then asks what the collection method for the software inventory is (wmic, PowerShell or psinfo). Once the selection method is chosen, in lines 33 thru 42, the script jumps to the correct section based on the input of wmic, PowerShell or psinfo. If none of those values are entered, the script is at line 44, and inform the user that an invalid entry was performed, and goes back to line 30.

Once the script continues, it calls the correct collection script, as discussed in Section 2.1. Within each of scripts, the user is requested to enter the IP address for the system being audited to collect the current information from that system. This information is stored in the c:\temp directory.

Jonathan Risto;jonathan.risto@hotmail.com

Similar to the PowerShell script, this script, in line 78, checks for the most recent file in the c:\temp directory (the output location of the called script) and stores the latest filename for the comparison.

The location to store the comparison results is requested from the user in line 87. Before running the comparison, the script takes the two files being compared and ensures that there are no UNICODE characters in the files that may cause a problem in the fc program. This work is performed in lines 92 and 93 of the script. The reason for this is that the output from wmic is stored in UNICODE format, while the output from psinfo is not. The commands utilized have no impact on the output from psinfo but place the wmic output in a format that the fc command can use.

The comparison is run using the command *fc /w %orig_file%.v2 %new_file%.v2* . This comparison is performed on line 97 of the script. It is performed on the temporary files, indicated by the v2, instead of the original files as outlined above. The script them cleans up the temporary files and informs the user that the output is saved to the specified location.

The format and details of the comparison are the same as described in Section 4.1.

6. Conclusion

Auditing a computer system's software inventory is a complex task. It requires excellent documentation of the computer in question to understand what is approved to be installed on it. As well, it requires the current inventory to compare to the baseline, thus determining what has been modified from this approved baseline.

Auditing the system manually is possible, but a significant amount of time is needed to conduct a manual audit. Even by using the commands outlined in Sections 3 and 4, the ability to quickly verify numerous systems is limited by the capacity of the auditor to have access to the required inventories to compare.

It is a huge advantage for the auditor to automatically and swiftly review a system through automating the collection of the needed information, comparing the results and storing the results for offline analysis and inclusion in a report. A simple means that is

easily duplicated with minimal typing also aids in the reduction of errors from all involved parties.

The scripts also enable the administrator to make any changes by documenting them in detail. If desired it would even be possible to modify the scripts to accept any and all parameters as input, or to check multiple systems. Both of these examples would extend this work, but are beyond the scope of this paper.

This work has attempted to provide a simple yet practical means for collecting and required information from the computer being audited. At the same time, it has sought to make it easy for the auditor and system administrator to run the programs in question and interpret the results.

Jonathan Risto;jonathan.risto@hotmail.com

References

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 August 15, 2016, from Microsoft Developer Network: https://technet.microsoft.com/en-us/library/hh847840.aspx

Microsoft. (2016). *Using the Compare-Object Cmdlet*. Retrieved August 15, 2016, from Microsoft Technet: https://technet.microsoft.com/en-us/library/ee156812.aspx

Microsoft. (2016a). *Get-Content*. Retrieved August 15, 2016, from Microsoft Developer Network: https://technet.microsoft.com/en-us/library/hh849787.aspx

Microsoft. (2016b). *FC*. Retrieved August 15, 2016, from Windows XP Professional Product Documentation: https://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/fc.mspx?mfr=true

Risto, J. (2016, August 20). *Windows Installed Software Inventory.* Retrieved August 30, 2016, from SANS Reading Room: https://www.sans.org/reading-room/whitepapers/critical/windows-installed-software-inventory-37252

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

SS64. (2015). *Compare-Object*. Retrieved August 15, 2016, from SS64: http://ss64.com/ps/compare-object.html

Jonathan Risto;jonathan.risto@hotmail.com

Appendix A

Windows Batch file auditing

To run the verification batch file, run the following command at an administrative command prompt, assuming that the batch file is stored in the c:\temp directory.

C:\temp\audit_command_line.bat

There are no command line options required for this script. Details regarding the script and how it works are outlined in Section 5.2.

COMMAND LINE SCRIPT LISTING – audit_command_line.bat

```
echo off
cls

:start
set /p orig_file=Enter the full path and name of the baseline file for the computer to be audited:
REM  getting the baseline file

echo You entered %orig_file%
echo.

if "%%~orig_file" == "" goto :start
REM if the file is blank, need to get a request again

if exist %orig_file% (
   echo The file %orig_file% exists
        goto :collection
        REM Check to see if the file is there, if it is, allow things to continue

) else (
   echo The file %orig_file% doesn't exist. Please enter a valid filename.
        echo.
        goto :start
```

```
25              REM if the file does not exist, go back and ask again for the filename
26      )
27
28
29      :collection
30      set /p query_means=Enter the collection method you wish to use to audit the information (type one of psinfo, PowerShell or wmic)
31      if "%query_means%"=="psinfo" (
32          goto :psinfo
33          )
34      if "%query_means%"=="powershell" (
35          goto :powershell
36          )
37      if "%query_means%"=="wmic" (
38          goto :wmic
39         )
40      REM checking for valid inputs and then going to the right location
41
42      echo you didn't enter a valid entry, please try again.
43      goto :collection
44      REM no valid value, go back and ask again
45
46      :wmic
47      call c:\temp\wmic_query.bat
48      rem need to use call command to run a script from within a script
```

```
goto :compare

:powershell
powershell /executionpolicy bypass /file c:\temp\software_enumeration.ps1
goto :compare

:psinfo
call C:\temp\psinfo.bat
goto :compare

:compare

REM now that we have the two files, need to compare them
for /f "delims=|" %%I in ('dir "c:\temp\*.*" /b /o:d') do set new_file=c:\temp\%%I

REM above finds the most recent file written to the directory, as if using PowerShell or psinfo, we cannot guarantee what the name is.
REM dir /o:d shows files in order oldest first. last value output from the command would be the newest file. We store that value for use

REM now that we have the old file, and the new file is created, we can compare. We are using the Windows command line program FC

set /p out_file=Enter the output file name that you wish to store the comparison results in. Please enter full path (e.g.
C:\temp\comparison.txt)

REM creating the output file name to use in the next comma
```

```
73
74          >%out_file% (
75          type %orig_file% > %orig_file%.v2
76          type %new_file% > %new_file%.v2
77
78          REM ensure that the files do not have any UNICODE entries in them. WMIC output stores in UNICODE. This ensures we don't touch the
79  originals.
80
81          fc /w %orig_file%.v2 %new_file%.v2
82          )
83
84          del %new_file%.v2
85          del %orig_file%.v2
86          REM clean up the temp files before leaving the script
87
88          echo The comparison results are stored in  %out_file%
89          REM above we open the output file first, and then run the command we want in that file. Easiest way to capture command outputs
```

Appendix B

To invoke the PowerShell script, enter the following command at an administrative command prompt, assuming that the script has been stored in the c:\temp directory.

PowerShell /executionpolicy bypass /file c:\temp\audit_powershell.ps1

The above command utilizes the */executionpolicy bypass* option when invoking PowerShell to ensure that group policies preventing scripts to be run do not cause a problem. This override is only for this single instance of PowerShell and does not require any changes to the policy settings. However, it does require an administrative command shell.

The */file c:\temp\audit_powershell.ps1* option instructs PowerShell to use the file specified and run the command contained therein.

Details on what happens within the script are provided in Section 5.1.

Jonathan Risto;jonathan.risto@hotmail.com

POWERSHELL SCRIPT LISTING – audit_powershell.ps1

```
$orig_file = read-host 'Enter the full path and name of the baseline file for the computer to be audited: '

#get baseline file name from user
if (![System.IO.File]::Exists($orig_file)) {
   #check to see if the file exists. If it doesn't, loop till it does

   do{
      $orig_file = read-host 'File does not exist. Enter the full path and name of the baseline file for the computer to be audited: '
      }
   while (![System.IO.File]::Exists($orig_file))
   }

#have a file that exists for a baseline.
$query_means = read-host 'Enter the collection method you wish to use to audit the information (type one of psinfo, powershell or wmic)'

if ($query_means -ne "wmic" -AND $query_means -ne "psinfo" -AND $query_means -ne "powershell")
{
   do {
      $query_means = read-host 'Invalid entry. Enter one of psinfo, powershell or wmic'
      }
   Until ($query_means -eq "wmic" -OR $query_means -eq "psinfo" -OR $query_means -eq "powershell")
   }
#ask for the collection method. If the value is not entered correctly, loop until it is
```

```
if ($query_means -eq "wmic") {
cmd /c c:\temp\wmic_query.bat
}

if ($query_means -eq "powershell") {
invoke-expression -Command 'powershell /executionpolicy bypass /file c:\temp\software_enumeration.ps1'
}

if ($query_means -eq "psinfo") {
cmd /c c:\temp\psinfo.bat
}

# the above runs the proper commands depending on what the user has selected

$newest_file = Dir c:\temp\ | Sort CreationTime -Descending | Select Name -First 1
$new_filename = "c:\temp\"
$new_filename += $newest_file.name


# the above queries the directory where the files are stored, sorts by creation time and selects the newest one.
# the filename is stored in $newest_file.name location variable
```

```
49          $out_file= read-host 'Enter the output file name where you wish to store the comparison results. Please enter full path (e.g.
50  C:\temp\comparison.txt)'
51
52          # creating the output file name to use in the next commands
53
54          $temp1 = "baseline file "
55          $temp1 += $orig_file
56          $temp2 = "compare file "
57          $temp2 += $new_filename
58          $temp1 | out-file $out_file
59          $temp2 | out-file $out_file -append
60           #putting some details on the two files compared to the start of the output file
61
62           Compare-Object $(Get-Content $orig_file) $(Get-Content $new_filename) | out-file -filepath $out_file -append
63
64           #-append command used on outfile to not overwrite the existing file information we created above.
65
66          # run the compare-object commandlet to compare the two files. Output is redirected to the file the user entered
67
68          write-host 'The comparison file was saved at ' $out_file '.'
69
```