Utilizing "AutoRuns" To Catch Malware

*What wants to **auto-**run on your Windows system today?*

Author: Jim McMillan, jimm.sans@bis.midco.net
Advisor: Pedro Bueno

Accepted: <DATE>

Abstract

Malware likes to survive- detect its survival attempts today! How do you know what is scheduled to start automatically on your Windows systems? Learn how to utilize freely available utilities to monitor the areas of a Windows system that malware may use to survive the boot sequence. In this paper, we will discuss how to monitor areas on a Windows system that are used to start processes at boot up and logon. We will look at scripting to automate the use of these utilities to monitor what is set to run automatically on Windows systems. Anti-virus and spyware software only detects what it knows about. With the tools and methods we discuss in this paper, you will be able to monitor what is set to start and be alerted to any changes. This information will provide you a better and faster approach to really determine what is happening when Windows systems start up.

1. Introduction

"Malware is a set of instructions that run on your computer and make your system do something that an attacker wants it to do". (Skoudis, 2004) It can perform a number of undesirable tasks on your computer. Malware is often referred to as malicious code because its programming intent is usually for something malicious. In his book, "Malware: Fighting Malicious Code", Ed Skoudis writes, "The malicious code doesn't have your best interests in mind." (Skoudis, 2004).

With many people and companies analyzing malware over the years, we have observed this fact about malware. It has been used to satisfy the agendas of criminals at the expense of many law abiding citizens and businesses. From the number of anti-virus and anti-spyware products available, we can safely say there is a high demand from consumers to protect themselves from malware. Many people and businesses just plainly do not want to have malware on their computers.

Motivation for the malware creator is primarily for financial gain. In a 2005 article, Information Week reported "more than 70 percent of virus writers are now writing spyware under contract, one more piece of evidence that hacking has evolved from mischievous hobby to money-making criminal venture". (Keizer, 2005) Malware authors have been financially motivated for many years.

The indication of financial motivation continues on into the decade. In an article from the 13th IEEE International Symposium on Pacific Rim Dependable Computing it is stated, "All these recent financial-motivated malware programs, after successfully infiltrating a machine, need to stay running on the infected machine for a long time and therefore must survive a system rebooted. This can be achieved by one of the following two ways: 1. by registering as an OS auto-start extension such as an NT service, a tray icon in Windows, or a Unix daemon/cron job; or 2. as an extension to an existing application that is either automatically run (such as the shell in Windows), or popular and commonly run by users (such as a Web browser)." (Dai, & Kuo, 2008)

While in September 2009, a more recent article on the TechNewsWorld website stated "While in the past malware authors have been driven by things like spite and the possibility of fame, in today's climate their motivation is almost always financial." (Debrosse, 2009) This is an indication that more malware authors are realizing the potential of their code to make money, which many people consider more important than the fame.

On the VirusList.com website, it is claimed that "Malware appears in any given environment when the following criteria are met: 1) The operating system is widely used, 2) reasonably high-quality documentation is available and 3) the targeted system is insecure or has a number of documented vulnerabilities." (Kaspersky Lab) For this reason, we are going to look at malware as it pertains to Microsoft Windows operating systems and applications that run on Windows.

As you can see, malware is a serious business. And as in any business, survival techniques are employed to keep the business in business. In this paper, we will discuss some of malware's survival techniques and how to detect them. We will look at some of the auto-run areas of a Windows system where third party code can be configured to run automatically. Then we will look at how malware can take advantage of these areas to survive a system reboot. We will discuss creating a basic process, and script, to monitor for changes in these areas. Finally, we will cover examples of things we may find with our process and how to utilize the process in our defense-in-depth strategy.

2. Detect malware initialized by auto-run techniques

2.1. Windows system startup

The Windows system startup is a complicated process that is difficult to explain in detail. The intent of this section is to familiarize ourselves with some portions of the startup process, not to learn the entire process in detail. If we use Google to search the Internet for information on the Windows system startup, we will find very little detail on the process from start to finish. In fact, even a search of Microsoft's site will turn up very little conclusive results.

In the book "Windows Internals, Fifth Edition" there are 28 pages that discuss the steps of the Windows startup. In those 28 pages, there about as many references back to process in chapters 3-7 and many references to various registry entries involved in the boot process. Toward the end of the authors' discussion, they state "In addition to the Userinit and Shell registry values in Winlogon's key, there are many other registry locations and directories that default system components check and process for automatic process startup during the boot and logon processes." (Russinovich, & Solomon, 2009)

If asked what we think the system is doing when it starts up, we would probably answer with a few common items. We would probably agree on things like the loading of hardware device drivers. Drivers for things we use to interface with the operating system to accomplish tasks, things like the mouse, keyboard, display and printers. Which is exactly right, all of these devices load drivers as part of the Windows system startup.

For example, on the root level of a Windows XP system partition there is a hidden file called boot.ini. This file is used to provide a boot menu at system startup and can be configured with certain boot options to tell the system how to boot. If we consider adding the /BOOTLOG option as described at http://technet.microsoft.com/en-us/sysinternals/bb963892.aspx, we can tell the system to write a log file to the filesystem location %SystemRoot%\Ntbtlog.txt. Analyzing this file we will see some of device drivers that are loaded during startup. If we review the registry keys under

HKLM\System\CurrentControlSet\ Services, we will find entries that contain the parameters for these drivers, as well as other drivers and services.

On The Microsoft TechNet website there is an article titled "System and Startup" by Paul Sanna. The article discusses some key registry locations, in addition to the Services registry keys. The article is on an older version of Windows, however, these registry locations still apply on today's versions of Windows. Sanna writes, "When Windows 2000 starts, a number of programs launch in the protected system context. To see the applications that launch for the operating system, open the Registry and inspect this entry.

Root Key: HKEY_LOCAL_MACHINE
Key: SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon

A handful of Registry entries control the applications and services that run at startup. It's useful to know where to find this list should you need to debug some problem at startup. The following list of keys show you where the run information is stored:

HKLM \SOFTWARE \Microsoft \Windows \CurrentVersion \Run
HKLM \SOFTWARE \Microsoft \Windows \CurrentVersion \RunOnce
HKCU \Software \Microsoft \Windows \CurrentVersion \Run
HKCU \Software \Microsoft \Windows \CurrentVersion \Runonce" (Sanna, 2001)

If we run the System Configuration Utility, Msconfig.exe, built into most Windows Operating Systems, we can view and edit some of these services and applications that run at system startup. However, due to many discussions that refer to the Autoruns Utility instead of the System Configuration Utility, we will not be discussing the System Configuration Utility in detail. It is only mentioned because it is built into Windows operating systems and readily available for limited use. If you would like more information on the System Configuration Utility, please visit URL http://support.microsoft.com/kb/310560.

According to the "Windows Internals" book, "The Autoruns tool examines more locations than Msconfig and displays more information about the images configured to automatically run." (Russinovich, & Solomon, 2009). This fact is the reason we are discussing the Autoruns Utility for our monitoring and detection process.

As we venture into malware survival techniques and then on to a look at the Autoruns Utility, you will soon see that malware has a lot of opportunity to hide in the Windows startup process.

2.2. Malware survival techniques

2.2.1. How does malware survive the Windows boot process?

Since today's malware has financial motives driving its existence, it naturally makes sense that malware wants to live and thrive. Whether it is some sort of fake security software pestering us to make a purchase, some ransom demanding software or an elusive botnet agent awaiting commands, they all need a way to survive a power cycle. What better way to ensure survival than utilizing features of the OS that will assure automatic execution during the startup process?

In his book, "The Art of Computer Virus Research and Defense", Peter Szor discusses this point in a section on Viruses in Kernel Mode referring to the Windows NT Infis virus. He states, "The virus installs the proper Registry key to allow itself to load on next system startup: HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\inf". (Szor, 2005) An example of how an older virus uses the registry to ensure its survival.

Yet today, if you browse through malware characteristics on any anti-virus vendor's web site, you will find that much of the Windows related malware uses Windows startup processes to survive. To affirm this point, let's look at a few pieces of malware via the McAfee Avert Labs Threat Library (http://vil.nai.com/vil/default.aspx). The three pieces we will look at are: W32/Koobface.worm, W32/Conficker.worm, and Generic Rootkit.dt.dr.

Name: W32/Koobface.worm (http://vil.nai.com/vil/content/v_148955.htm)
Type: Virus
SubType: Win32
Discovery Date: 08/03/2008
Related autorun characteristics:
The following registry keys are added:
•HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Active Setup\Installed Components\Intelli Mouse Pro Version 2.0B\StubPath: "%WinDir%\System32\splm\ncsjapi32.exe"

Author Name, email@address

•HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce\*Intelli Mouse Pro Version 2.0B*: "%WinDir% \System32\splm\ncsjapi32.exe"
•HKEY_USERS\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced\Hidden: "2"
•HKEY_USERS\Software\Microsoft\Windows\CurrentVersion\Run\Intelli Mouse Pro Version 2.0B: "%WinDir% \System32\splm\ncsjapi32.exe"
•HKEY_USERS\Software\Microsoft\Windows\CurrentVersion\RunOnce\*Intelli Mouse Pro Version 2.0B*: "%WinDir% \System32\splm\ncsjapi32.exe"

Name: W32/Conficker.worm (http://vil.nai.com/vil/content/v_153464.htm)
Type: Virus
SubType: Worm
Discovery Date: 11/24/2008
Related autorun characteristics:
When executed, the worm copies itself using a random name to the %Sysdir% folder. (Where %Sysdir% is the Windows system folder; e.g. C:\Windows\System32)
It modifies the following registry key to create a randomly-named service on the affected system:
•HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\{random}\Parameters\"ServiceDll" = "Path to worm"
•HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\{random}\"ImagePath" = %SystemRoot%\system32\svchost.exe -k netsvcs

Name: Generic Rootkit.dt.dr (http://vil.nai.com/vil/content/v_241386.htm)
Type: Trojan
SubType: Dropper
Discovery Date: 11/10/2009
Related autorun characteristics:
The malware then creates the following registry entries to ensure its execution as a Windows service when the infected machine reboots:
•HKEY_Local_Machine\System\CurrentControlSet\Services\zacypxeepnjv7
ImagePath = "%System%\Drivers\[Random file name].sys"
DisplayName = "[Random filename]"

As you can see, these pieces of malware utilize auto-run areas of the Windows system to survive. Each one relates to at least one registry entry of our Windows system startup discussion. Obviously malware has some common use areas, but what other areas are available in the Windows system startup? Let's take a look at the Autoruns Utility to get an idea of those areas and how we can monitor them.

Author Name, email@address

2.3. The Autoruns Utility

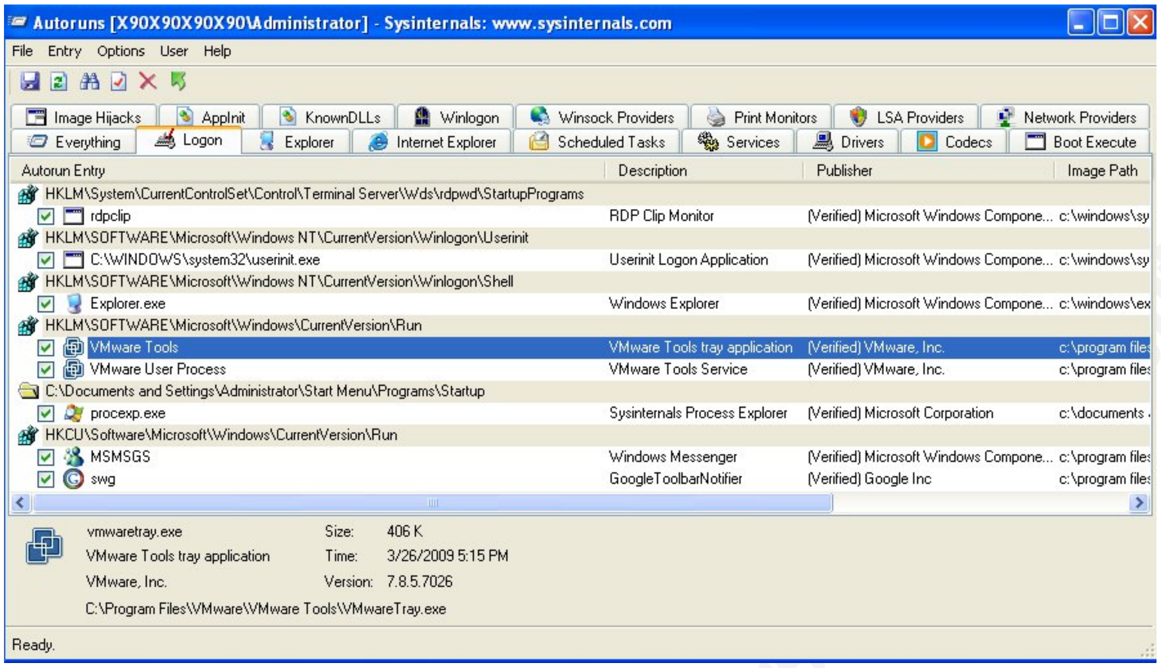
Autoruns is one of the many great Windows utilities created by Mark Russinovich and Bryce Cogswell of Windows Sysinternals. "This utility, which has the most comprehensive knowledge of auto-starting locations of any startup monitor, shows you what programs are configured to run during system bootup or login, and shows you the entries in the order Windows processes them. These programs include ones in your startup folder, Run, RunOnce, and other Registry keys." (Russinovich, & Cogswell, 2009)

Autoruns can be very beneficial in hunting down the malware that uses Windows startup areas to initialize. The information produced by Autoruns can be somewhat overwhelming at first, especially the first time you see it run. There are a surprisingly large number of processes that start during the Windows initialization process.

The Autoruns utility download includes a set of programs we will discuss, a GUI version (autoruns.exe) and a command line version (autorunsc.exe). We will take a look at both applications, but will quickly realize the benefits of the command line version as we get into our monitoring process. You can download the Autoruns Utility from Microsoft at http://technet.microsoft.com/en-us/sysinternals/bb963902.aspx. Step-by-step instructions for installation and configuration of the process we are going to build, including Autoruns, can be found in the Appendix (Section 4).

2.3.1. The Autoruns GUI interface

The Autoruns GUI interface, as seen in Figure 1, provides a nice view into all of the areas that are processed when a Windows systems starts.



Autorun Entry	Description	Publisher	Image Path
HKLM\System\CurrentControlSet\Control\Terminal Server\Wds\rdpwd\StartupPrograms			
rdpclip	RDP Clip Monitor	(Verified) Microsoft Windows Compone...	c:\windows\sy
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Userinit			
C:\WINDOWS\system32\userinit.exe	Userinit Logon Application	(Verified) Microsoft Windows Compone...	c:\windows\sy
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Shell			
Explorer.exe	Windows Explorer	(Verified) Microsoft Windows Compone...	c:\windows\ex
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run			
VMware Tools	VMware Tools tray application	(Verified) VMware, Inc.	c:\program file
VMware User Process	VMware Tools Service	(Verified) VMware, Inc.	c:\program file
C:\Documents and Settings\Administrator\Start Menu\Programs\Startup			
procexp.exe	Sysinternals Process Explorer	(Verified) Microsoft Corporation	c:\documents .
HKCU\Software\Microsoft\Windows\CurrentVersion\Run			
MSMSGS	Windows Messenger	(Verified) Microsoft Windows Compone...	c:\program file
swg	GoogleToolbarNotifier	(Verified) Google Inc	c:\program file

```
vmwaretray.exe                 Size:     406 K
VMware Tools tray application  Time:     3/26/2009 5:15 PM
VMware, Inc.                   Version:  7.8.5.7026
C:\Program Files\VMware\VMware Tools\VMwareTray.exe
```

Figure 1

It gives several tabs to view results for each auto-run area, including one tab to see everything. In Figure 1, the Logon tab is selected and shows all auto-run entries that get processed when a user logs on. As you can see, Autoruns.exe shows us a lot of information about the processes that are configured to run when Windows starts.

By default Autoruns.exe scans the auto-run areas for the computer and the user that is logged on to the computer. Autoruns displays the entry location/entry name (Autorun Entry), the description obtained from the file properties (Description), the signing publisher obtained from the file properties (Publisher) and the path to the file (Image Path). There is also a check box that gives us the ability to directly enable or disable entries from within Autoruns.exe.

By selecting an item in the list, we can obtain further information about that item. The additional information is displayed in the information pane at the bottom of the Autoruns window. In Figure 1, we can see the additional information available about the VMware Tools entry that is selected. This information includes some of the file properties such as file size, last modified date and version.

Author Name, email@address

Through the menu bar or a right-click context menu, we can perform certain actions on the selected entry or apply different options to simplify the view. Some actions that can be performed include:

- Delete – will delete the entry location. The entry in the registry will be deleted, not the file on disk where the entry pointed. If the entry location is a file path, whatever the path points to will be deleted.
- Copy – will copy the information of the selected entry to the clipboard in tab delimited format.
- Jump to – will open Regedit to the location of the selected entry.
- Search Online – will open a web browser and perform a web search for the process selected with the default browser.
- Process Explorer – will open Sysinternals Process Explorer to the process of the selected entry. The Process Explorer application needs to be running or in the environment path. More information on Process Explorer can be obtained at http://technet.microsoft.com/en-us/sysinternals/bb896653.aspx.
- Properties – will open the properties page for the file in the Image Path.

As we mentioned, there are also some options available to refine the view. Those options include:

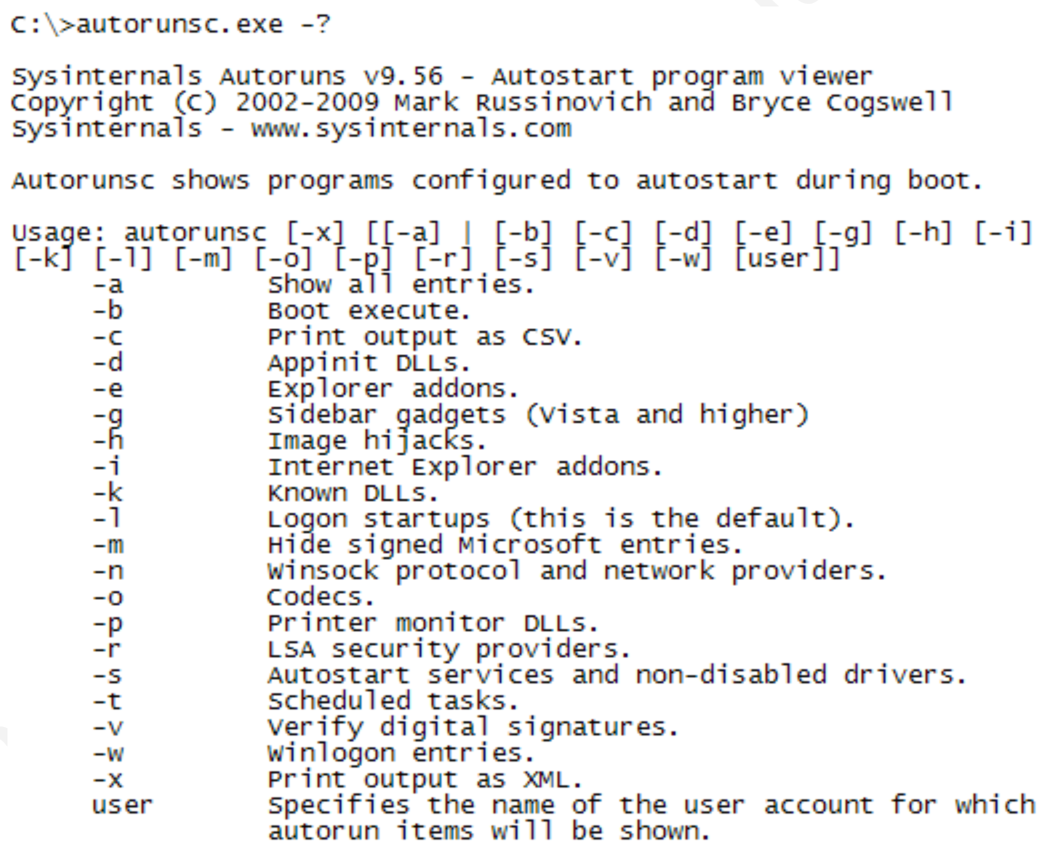
- Include Empty Locations – Autoruns will not show entries for the locations that do not contain any information. If you would like to see these entries, enable this option.
- Hide Microsoft and Windows Entries – will hide entries that are associated with Microsoft products. This should be combined with the Verify Code Signatures option to prevent hiding files with invalid signatures.
- Hide Windows Entries – will hide the entries that are associated with the Windows OS. This should be combined with the Verify Code Signatures option to prevent hiding files with invalid signatures.
- Verify Code Signatures – will attempt to verify the publishers signature (if available) to make sure it is signed with a valid signing certificate. This option has been turned on in Figure 1.

As you can see, the Autoruns GUI provides us with a lot of information with a lot of different options. The Autoruns GUI, by itself, is a good utility to manually look into a system that may be infected with malware and determine if anything looks amiss. But wait there is more; Autoruns has a command line version.

2.3.2. The AUTORUNSC command line interface

The Autoruns command line program, Autorunsc.exe, is capable of providing the same information as the GUI program. This gives us an ideal situation to expand the GUI's functionality and usefulness by providing us a means to automate and control desired output. Like when you need to analyze the auto-run output from a remote computer, a capability not found in the GUI version.

Autorunsc.exe gives us several options to control the information it can retrieve. Figure 2 shows the options that are available to us.



```
C:\>autorunsc.exe -?

Sysinternals Autoruns v9.56 - Autostart program viewer
Copyright (C) 2002-2009 Mark Russinovich and Bryce Cogswell
Sysinternals - www.sysinternals.com

Autorunsc shows programs configured to autostart during boot.

Usage: autorunsc [-x] [[-a] | [-b] [-c] [-d] [-e] [-g] [-h] [-i]
[-k] [-l] [-m] [-o] [-p] [-r] [-s] [-v] [-w] [user]]
      -a          Show all entries.
      -b          Boot execute.
      -c          Print output as CSV.
      -d          Appinit DLLs.
      -e          Explorer addons.
      -g          Sidebar gadgets (Vista and higher)
      -h          Image hijacks.
      -i          Internet Explorer addons.
      -k          Known DLLs.
      -l          Logon startups (this is the default).
      -m          Hide signed Microsoft entries.
      -n          Winsock protocol and network providers.
      -o          Codecs.
      -p          Printer monitor DLLs.
      -r          LSA security providers.
      -s          Autostart services and non-disabled drivers.
      -t          Scheduled tasks.
      -v          Verify digital signatures.
      -w          Winlogon entries.
      -x          Print output as XML.
      user        Specifies the name of the user account for which
                  autorun items will be shown.
```

Figure 2

If you look at the tabs in the GUI version, you will see a command line option to produce the results of each tab. However, there are a few instances where we get a

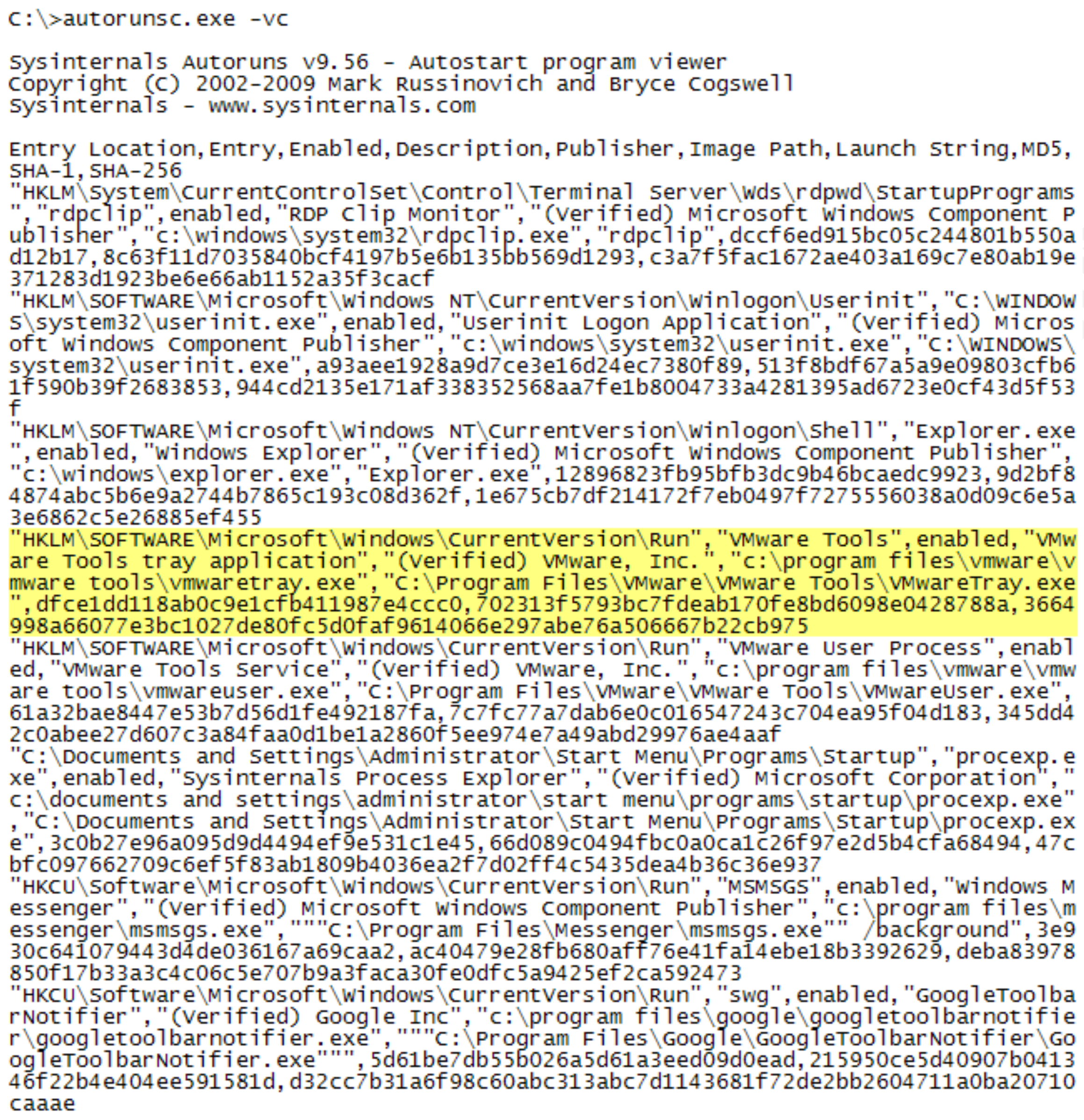
Author Name, email@address

couple of tabs combined into one option. The Services and Drivers tabs get combined under the “-s” option, whereas the Winsock Providers and Network Providers tabs get combined under the “-n” option. You will also notice a few options to control format of the output and the amount of output.

The “-m” option allows you to reduce the output to non-Microsoft entries, leaving you with all of the third party entries. The “-v” option allows you to verify the publisher’s signatures for applications that are digitally signed. Remember to use the “-v” option anytime you use the “-m” option. This will protect against programs masquerading as Microsoft programs.

The formatting options provide some convenient functionality as well. The “-c” option allows us to save output in comma separated format. This is a very useful option, especially when you are dealing with a remote computer. You can redirect output to a file and then load the output file into a spreadsheet. This will allow you to utilize sorting and searching features for faster analysis. It also proves to be nice when trying to detect changes, which we will discuss later.

Figure 3 shows us an example of the output from the Autorunsc.exe command, running against the same computer as we saw in Figure 1. The command is given options so it will provide Login Startups (“l” is the default, so “-l” not needed in our command), Verify Digital Signatures (“-v”) and output as CSV (“-c”).



```
C:\>autorunsc.exe -vc

Sysinternals Autoruns v9.56 - Autostart program viewer
Copyright (C) 2002-2009 Mark Russinovich and Bryce Cogswell
Sysinternals - www.sysinternals.com

Entry Location,Entry,Enabled,Description,Publisher,Image Path,Launch String,MD5,
SHA-1,SHA-256
"HKLM\System\CurrentControlSet\Control\Terminal Server\Wds\rdpwd\StartupPrograms
","rdpclip",enabled,"RDP Clip Monitor","(Verified) Microsoft Windows Component P
ublisher","c:\windows\system32\rdpclip.exe","rdpclip",dccf6ed915bc05c244801b550a
d12b17,8c63f11d7035840bcf4197b5e6b135bb569d1293,c3a7f5fac1672ae403a169c7e80ab19e
371283d1923be6e66ab1152a35f3cacf
"HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Userinit","C:\WINDOW
S\system32\userinit.exe",enabled,"Userinit Logon Application","(Verified) Micros
oft Windows Component Publisher","c:\windows\system32\userinit.exe","C:\WINDOWS\
system32\userinit.exe",a93aee1928a9d7ce3e16d24ec7380f89,513f8bdf67a5a9e09803cfb6
1f590b39f2683853,944cd2135e171af338352568aa7fe1b8004733a4281395ad6723e0cf43d5f53
f
"HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Shell","Explorer.exe
",enabled,"Windows Explorer","(Verified) Microsoft Windows Component Publisher",
"c:\windows\explorer.exe","Explorer.exe",12896823fb95bfb3dc9b46bcaedc9923,9d2bf8
4874abc5b6e9a2744b7865c193c08d362f,1e675cb7df214172f7eb0497f7275556038a0d09c6e5a
3e6862c5e26885ef455
"HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run","VMware Tools",enabled,"VMw
are Tools tray application","(Verified) VMware, Inc.","c:\program files\vmware\v
mware tools\vmwaretray.exe","C:\Program Files\VMware\VMware Tools\VMwareTray.exe
",dfce1dd118ab0c9e1cfb411987e4ccc0,702313f5793bc7fdeab170fe8bd6098e0428788a,3664
998a66077e3bc1027de80fc5d0faf9614066e297abe76a506667b22cb975
"HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run","VMware User Process",enabl
ed,"VMware Tools Service","(Verified) VMware, Inc.","c:\program files\vmware\vmw
are tools\vmwareuser.exe","C:\Program Files\VMware\VMware Tools\VMwareUser.exe",
61a32bae8447e53b7d56d1fe492187fa,7c7fc77a7dab6e0c016547243c704ea95f04d183,345dd4
2c0abee27d607c3a84faa0d1be1a2860f5ee974e7a49abd29976ae4aaf
"C:\Documents and Settings\Administrator\Start Menu\Programs\Startup","procexp.e
xe",enabled,"Sysinternals Process Explorer","(Verified) Microsoft Corporation","
c:\documents and settings\administrator\start menu\programs\startup\procexp.exe"
,"C:\Documents and Settings\Administrator\Start Menu\Programs\Startup\procexp.ex
e",3c0b27e96a095d9d4494ef9e531c1e45,66d089c0494fbc0a0ca1c26f97e2d5b4cfa68494,47c
bfc097662709c6ef5f83ab1809b4036ea2f7d02ff4c5435dea4b36c36e937
"HKCU\Software\Microsoft\Windows\CurrentVersion\Run","MSMSGS",enabled,"Windows M
essenger","(Verified) Microsoft Windows Component Publisher","c:\program files\m
essenger\msmsgs.exe","""C:\Program Files\Messenger\msmsgs.exe"" /background",3e9
30c641079443d4de036167a69caa2,ac40479e28fb680aff76e41fa14ebe18b3392629,deba83978
850f17b33a3c4c06c5e707b9a3faca30fe0dfc5a9425ef2ca592473
"HKCU\Software\Microsoft\Windows\CurrentVersion\Run","swg",enabled,"GoogleToolba
rNotifier","(Verified) Google Inc","c:\program files\google\googletoolbarnotifie
r\googletoolbarnotifier.exe","""C:\Program Files\Google\GoogleToolbarNotifier\Go
ogleToolbarNotifier.exe""",5d61be7db55b026a5d61a3eed09d0ead,215950ce5d40907b0413
46f22b4e404ee591581d,d32cc7b31a6f98c60abc313abc7d1143681f72de2bb2604711a0ba20710
caaae
```

Figure 3

Output from Autorunsc.exe provides all the information we have seen in the GUI output, plus some. In Figure 3, we see four additional fields: Launch String, MD5, SHA-1 and SHA-256. The Launch String field shows the actual command that is issued to launch the process. This may often be the same as the Image Path field, but may contain additional command arguments from time to time. The other fields, as their name indicates, are MD5, SHA-1 and SHA-256 hashes, respectively, of the file located in the Image Path. These hashes will be very important to us as we will see later.

2.4. Using Autorunsc Output to detect malware

2.4.1. Theory behind the process

The idea behind detecting malware with Autoruns is to detect changes in the auto-run areas of a Windows system over time. Change detection is accomplished by comparing a current snapshot image to a previous snapshot image and then analyzing the differences. The changes will hopefully be minimal and will allow for quick detection of something that looks out of place.

The process involves coming up with a schedule as to when to make snapshots, compare snapshots and rotate the snapshot images. At the scheduled time, the process will run and take a new snapshot of the auto-run areas. Once the new snapshot is made it will compare it to an older snapshot, probably the previous one. If you run a very tight change control process, you may compare to a baseline image. In the process we discuss later, we will be rotating the snapshots.

As far as the frequency of the process, the time period between snapshot comparisons can vary from system to system due to varying threat levels. You really have to determine what time period is best for you. Comparing snapshots once a day for the light computer user may be enough. For heavy Internet and e-mail usage, you may want to schedule a task to run more than once a day. The frequency all depends on your exposure to threats, other defenses/mitigations in place and your overall paranoia level.

2.4.2. Creating a snapshot of the auto-run areas

To create a snapshot of the auto-run areas we will be using the command line utility and redirect the output to a file. The GUI version has a compare function built in to it. However, for our automation techniques later on, we are going to stick with the command line application, Autorunsc.exe, for now.

Referring back to the options we have available for our command, let's build a command that will dump everything, verify signatures and format the output in CSV format. To do this we will use the "-a" option to show everything. The "-v" option to verify digital signatures of the publishers. And finally, the "-c" option to format the output in a comma separated format with one entry per line. We could add the "-m" option if we would like to minimize the output included in the snapshot, but we will leave it out so we can identify any changes to Microsoft entries.

We want our output to be in CSV format for a couple of good reasons. First, all the information about a single entry is on a single line of output. This will make the job of comparing the same entries in one snapshot to another snapshot easy. Since the output for an entry will remain the same as long as nothing changes, we will not be alerted to the entry. We will cover snapshot comparison in detail in the next section. Secondly, this format will make it convenient to import the data into a spreadsheet or database for analysis or other tasks.

When deciding what we want to see, we will want everything that our Windows system will attempt to start when it boots. By analyzing everything, we will be alerted to changes in any files that get modified. This includes system files that may have been modified by the installation of a service pack or, in worst case, by infection of malicious code.

Remember that we have hashes of the files in our output, so we can detect when an existing entry gets changed. When software updates are applied, it is often that a filename, path and registry entry gets reused. If the entry location, entry, description, image path, publisher and launch string doesn't change, how will we be alerted? Easy- if a file's content changes, so does the hash of the file. If a hash changes, so will the entry in our output. Bottom line, if a file's content is modified, we will be alerted.

In our command, we are adding the option to verify digital signatures of the publishers to help prioritize our analysis of changes. If the digital signature of a file is verified successfully, chances are pretty high that it is valid. This gives us the opportunity to focus on the other changes that have no digital signatures or unverifiable ones.

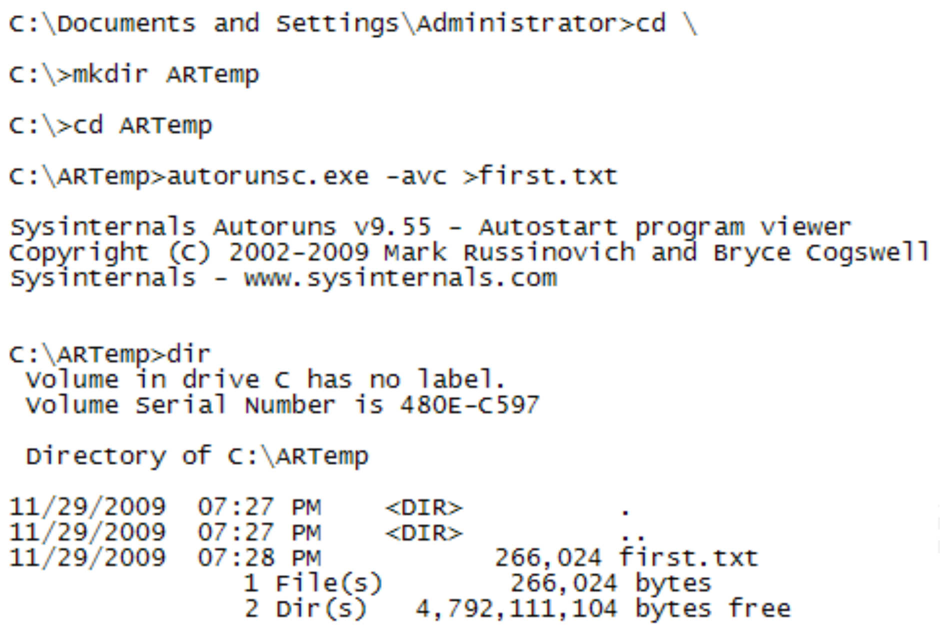
The command, with the options we discussed, will be "autorunsc.exe -avc". This will give us the output we desire and provide for the greatest chance of malware activity detection. To get the output to a snapshot file for comparison, we will add a redirect to the command, such as "autorunsc.exe –avc >{unique filename}". If you are unfamiliar with command line redirection operators, you can find more information on them on Microsoft's web site at URL: http://technet.microsoft.com/en-us/library/bb490982.aspx.

2.4.3. Comparing two snapshots

Comparing snapshots and analyzing the differences, or changes, between two snapshots over a period of time will be what helps us catch malware activity. However, before we can compare two snapshots, we need to have two snapshots. To discuss snapshot comparison, it would be most beneficial to discuss an example where the snapshots have some differences.

For our example, we will install the Google Toolbar between running our snapshots. Snapshot files contain a lot of information, everything that starts automatically, so we will not be looking at the contents of these files directly. We have already looked at a subset of what the files contain when we were discussing the Autoruns utility. At this time, we will be interested in looking at the differences between the two snapshot files.

First, we will issue our command and redirect it to a file to use as a previous, or baseline, snapshot. To do this, let's make a temporary directory in the root of C:\ called ARTemp and change directories so we are in that directory. Then issue our command as seen in Figure 4, directing the output to a file (first.txt in this case). After issuing the autorunsc.exe command, we will have a comma delimited file for our previous snapshot.

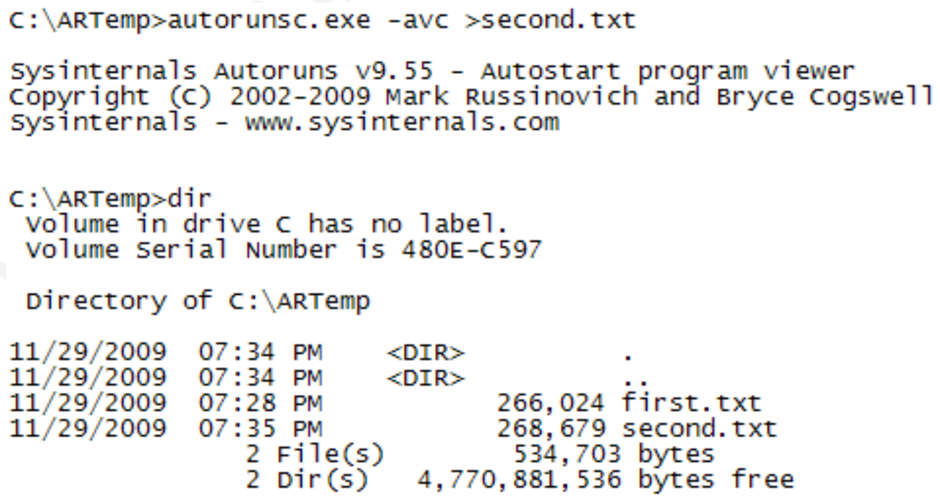


```
C:\Documents and Settings\Administrator>cd \

C:\>mkdir ARTemp

C:\>cd ARTemp

C:\ARTemp>autorunsc.exe -avc >first.txt

Sysinternals Autoruns v9.55 - Autostart program viewer
Copyright (C) 2002-2009 Mark Russinovich and Bryce Cogswell
Sysinternals - www.sysinternals.com


C:\ARTemp>dir
 Volume in drive C has no label.
 Volume Serial Number is 480E-C597

 Directory of C:\ARTemp

11/29/2009  07:27 PM    <DIR>          .
11/29/2009  07:27 PM    <DIR>          ..
11/29/2009  07:28 PM           266,024 first.txt
               1 File(s)        266,024 bytes
               2 Dir(s)   4,792,111,104 bytes free
```

Figure 4

Next, we will install the Google Toolbar. We are not going to discuss the install in detail. That is outside the scope of this paper. We will just go to Google’s site, navigate to the download page, download the installer and install it. Of course we will virus scan the downloaded installer prior to executing it.

After the install is complete, we will issue our command to create another snapshot file to compare to our previous snapshot. To do this, let’s issue our command again as seen in Figure 5. This time we created another file called second.txt. You will notice the second.txt file is slightly larger than the first.txt file. What has changed?



```
C:\ARTemp>autorunsc.exe -avc >second.txt

Sysinternals Autoruns v9.55 - Autostart program viewer
Copyright (C) 2002-2009 Mark Russinovich and Bryce Cogswell
Sysinternals - www.sysinternals.com


C:\ARTemp>dir
 Volume in drive C has no label.
 Volume Serial Number is 480E-C597

 Directory of C:\ARTemp

11/29/2009  07:34 PM    <DIR>          .
11/29/2009  07:34 PM    <DIR>          ..
11/29/2009  07:28 PM           266,024 first.txt
11/29/2009  07:35 PM           268,679 second.txt
               2 File(s)        534,703 bytes
               2 Dir(s)   4,770,881,536 bytes free
```

Figure 5

Author Name, email@address

To find out, we need to do a comparison of the snapshot files, first.txt and second.txt. How do we do this without manually comparing line by line?

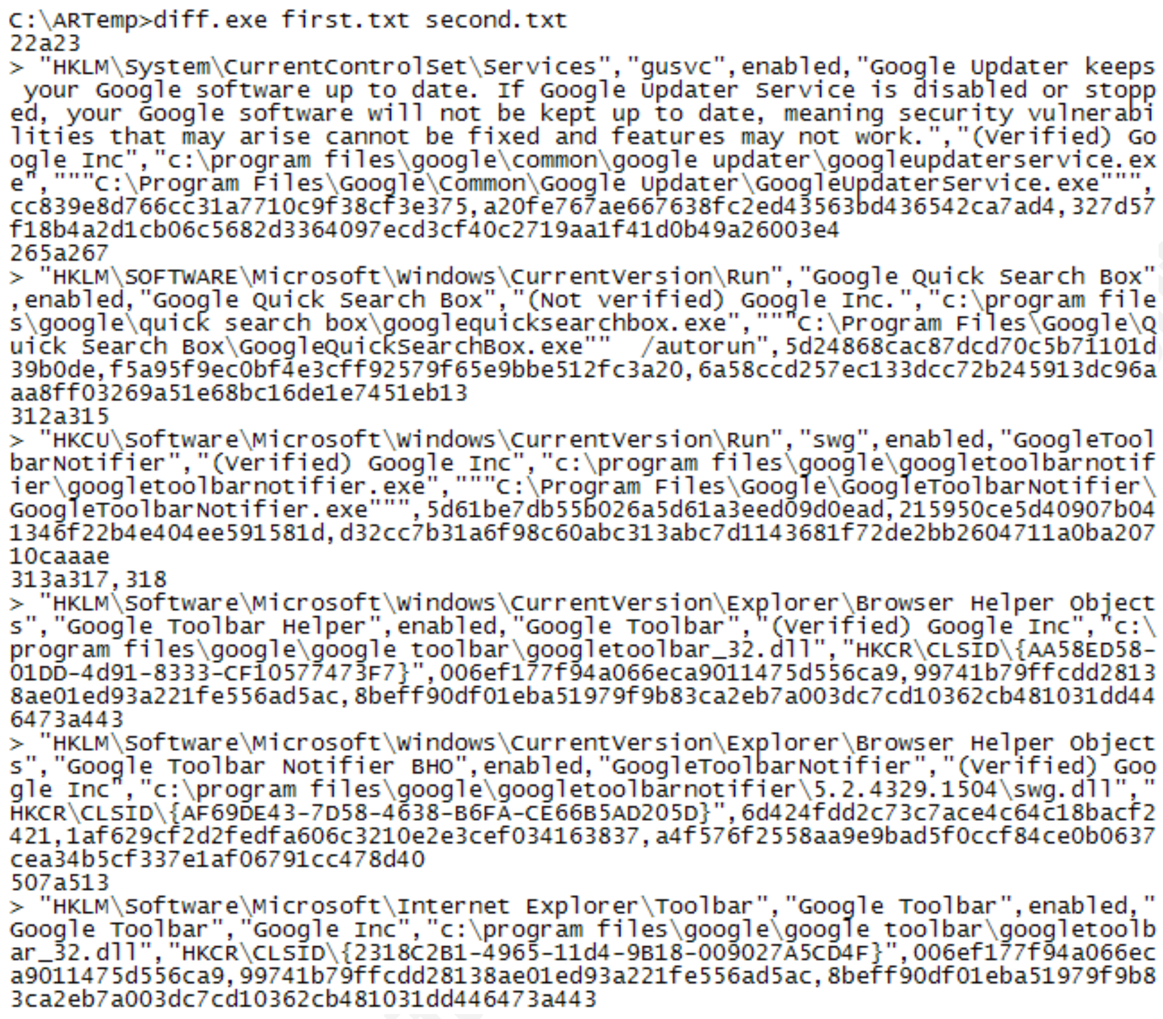
One option would be to use the Windows built-in file comparison utility, fc.exe. However, using this utility presents us with a couple of challenges. First, the utility's output is not ideal to use within a script for parsing of the results, as we will do later. Secondly, we are trying to monitor for malware attempting to hide on our system. If the malware cloaks itself from the built-in Windows commands, how can we detect it?

To overcome these challenges, we will discuss using the diff.exe command from DiffUtils for Windows. Diff will give us output of the differences in our two files that is easy to parse. It will also provide us with programs and libraries that are separate from the Windows operating system. However, before we get into the actual file comparison, let's take a quick look at what Diff is and how to get it.

Diff started out as a GNU utility for Linux. According to GNU.org, "Computer users often find occasion to ask how two files differ. Perhaps one file is a newer version of the other file. Or maybe the two files started out as identical copies but were changed by different people. You can use the diff command to show differences between two files, or each corresponding file in two directories. diff outputs differences between files line by line in any of several formats, selectable by command line options. This set of differences is often called a 'diff' or 'patch'. For files that are identical, diff normally produces no output; for binary (non-text) files, diff normally reports only that they are different." (Free Software Foundation, 2008)

To run diff.exe on Windows we need to get the DiffUtils for Windows package and the Dependency files from SourceForge. Download these files and extract them according to the instructions in the Appendix, section 4.3. While we are downloading the files, we will grab the documentation file to have for future reference.

Now we are ready to perform our comparison, we run the diff.exe program and give it our two snapshot files. The diff.exe command has many options but for our needs we just need to give it the two files we wish to compare. Figure 6 shows the output from the diff.exe command, "diff.exe first.txt second.txt", we use to compare the previous file (first.txt) to our current snapshot file (second.txt).



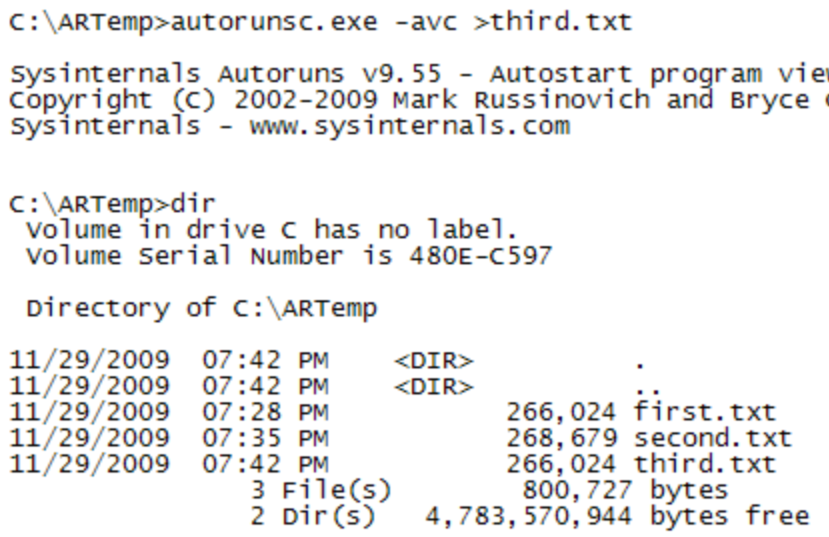
```
C:\ARTemp>diff.exe first.txt second.txt
22a23
> "HKLM\System\CurrentControlSet\Services","gusvc",enabled,"Google Updater keeps
 your Google software up to date. If Google Updater Service is disabled or stopp
ed, your Google software will not be kept up to date, meaning security vulnerabi
lities that may arise cannot be fixed and features may not work.","(Verified) Go
ogle Inc","c:\program files\google\common\google updater\googleupdaterservice.ex
e","""C:\Program Files\Google\Common\Google Updater\GoogleUpdaterService.exe""",
cc839e8d766cc31a7710c9f38cf3e375,a20fe767ae667638fc2ed43563bd436542ca7ad4,327d57
f18b4a2d1cb06c5682d3364097ecd3cf40c2719aa1f41d0b49a26003e4
265a267
> "HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run","Google Quick Search Box"
,enabled,"Google Quick Search Box","(Not verified) Google Inc.","c:\program file
s\google\quick search box\googlequicksearchbox.exe","""C:\Program Files\Google\Q
uick Search Box\GoogleQuickSearchBox.exe""  /autorun",5d24868cac87dcd70c5b71101d
39b0de,f5a95f9ec0bf4e3cff92579f65e9bbe512fc3a20,6a58ccd257ec133dcc72b245913dc96a
aa8ff03269a51e68bc16de1e7451eb13
312a315
> "HKCU\Software\Microsoft\Windows\CurrentVersion\Run","swg",enabled,"GoogleTool
barNotifier","(Verified) Google Inc","c:\program files\google\googletoolbarnotif
ier\googletoolbarnotifier.exe","""C:\Program Files\Google\GoogleToolbarNotifier\
GoogleToolbarNotifier.exe""",5d61be7db55b026a5d61a3eed09d0ead,215950ce5d40907b04
1346f22b4e404ee591581d,d32cc7b31a6f98c60abc313abc7d1143681f72de2bb2604711a0ba207
10caaae
313a317,318
> "HKLM\Software\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Object
s","Google Toolbar Helper",enabled,"Google Toolbar","(Verified) Google Inc","c:\
program files\google\google toolbar\googletoolbar_32.dll","HKCR\CLSID\{AA58ED58-
01DD-4d91-8333-CF10577473F7}",006ef177f94a066eca9011475d556ca9,99741b79ffcdd2813
8ae01ed93a221fe556ad5ac,8beff90df01eba51979f9b83ca2eb7a003dc7cd10362cb481031dd44
6473a443
> "HKLM\Software\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Object
s","Google Toolbar Notifier BHO",enabled,"GoogleToolbarNotifier","(Verified) Goo
gle Inc","c:\program files\google\googletoolbarnotifier\5.2.4329.1504\swg.dll","
HKCR\CLSID\{AF69DE43-7D58-4638-B6FA-CE66B5AD205D}",6d424fdd2c73c7ace4c64c18bacf2
421,1af629cf2d2fedfa606c3210e2e3cef034163837,a4f576f2558aa9e9bad5f0ccf84ce0b0637
cea34b5cf337e1af06791cc478d40
507a513
> "HKLM\Software\Microsoft\Internet Explorer\Toolbar","Google Toolbar",enabled,"
Google Toolbar","Google Inc","c:\program files\google\google toolbar\googletoolb
ar_32.dll","HKCR\CLSID\{2318C2B1-4965-11d4-9B18-009027A5CD4F}",006ef177f94a066ec
a9011475d556ca9,99741b79ffcdd28138ae01ed93a221fe556ad5ac,8beff90df01eba51979f9b8
3ca2eb7a003dc7cd10362cb481031dd446473a443
```

Figure 6

Our output shows us that the Google Toolbar adds six entries to our auto-run areas. We know this because of the six entries that diff.exe found in second.txt file that were not in first.txt file. Notice each additional entry is prefixed by a “>” symbol. If we think about the order of the input files on the command line, the “>” symbol can be considered a pointer to the file where the entry appears. Thus, second.txt would have these entries in comparison to first.txt. The rest of the files entries are the same, hence no display of the entries. If there had been an entry in first.txt that was not in second.txt, the “<” symbol would have appeared before the entry.

The line previous to the entries indicates our deductions are correct. For example, prior to our first entry we see 22a23. The 22a23 means the entry following this line was found after a match of line 22 in first.txt and appears as an additional line, line 23, in second.txt. The concept behind the "<" and ">" symbols is all we need to understand. If you would like more information on diff.exe output, please refer to the documentation file we downloaded from SourceForge.

To see an example of what it looks like when diff.exe shows removed items, let's uninstall the Google Toolbar. After we go through add/remove programs to perform the uninstall, we run our autoruns.exe command for another snapshot as seen in Figure 7.

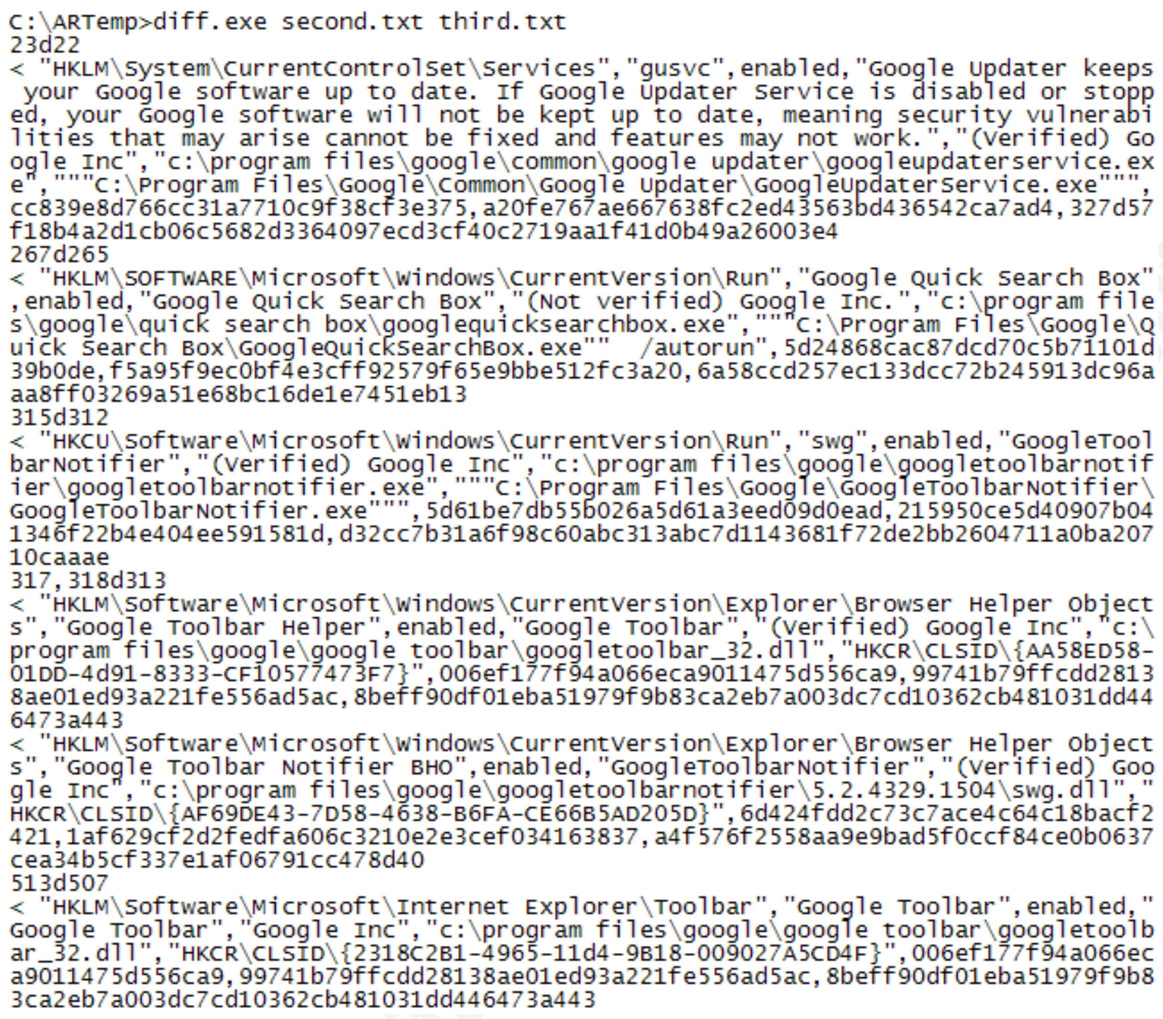


```
C:\ARTemp>autorunsc.exe -avc >third.txt

Sysinternals Autoruns v9.55 - Autostart program viewer
Copyright (C) 2002-2009 Mark Russinovich and Bryce Cogswell
Sysinternals - www.sysinternals.com


C:\ARTemp>dir
 Volume in drive C has no label.
 Volume Serial Number is 480E-C597

 Directory of C:\ARTemp

11/29/2009  07:42 PM    <DIR>          .
11/29/2009  07:42 PM    <DIR>          ..
11/29/2009  07:28 PM           266,024 first.txt
11/29/2009  07:35 PM           268,679 second.txt
11/29/2009  07:42 PM           266,024 third.txt
               3 File(s)        800,727 bytes
               2 Dir(s)   4,783,570,944 bytes free
```

Figure 7

As you can see, we redirect the output to a file called third.txt. If we look at the file size in the directory listing, the size of third.txt is now the same as first.txt. This would possibly indicate that the uninstall process removed the six entries it added during the install.

But we can't say for sure what happened until we look at the difference between our last snapshot and our current snapshot. If we run diff.exe and compare our previous snapshot file (second.txt) to our current snapshot file (third.txt), we see the output from our command as shown in Figure 8.



```
C:\ARTemp>diff.exe second.txt third.txt
23d22
< "HKLM\System\CurrentControlSet\Services","gusvc",enabled,"Google Updater keeps
 your Google software up to date. If Google Updater Service is disabled or stopp
ed, your Google software will not be kept up to date, meaning security vulnerabi
lities that may arise cannot be fixed and features may not work.","(Verified) Go
ogle Inc","c:\program files\google\common\google updater\googleupdaterservice.ex
e","""C:\Program Files\Google\Common\Google Updater\GoogleUpdaterService.exe""",
cc839e8d766cc31a7710c9f38cf3e375,a20fe767ae667638fc2ed43563bd436542ca7ad4,327d57
f18b4a2d1cb06c5682d3364097ecd3cf40c2719aa1f41d0b49a26003e4
267d265
< "HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run","Google Quick Search Box"
,enabled,"Google Quick Search Box","(Not verified) Google Inc.","c:\program file
s\google\quick search box\googlequicksearchbox.exe","""C:\Program Files\Google\Q
uick Search Box\GoogleQuickSearchBox.exe""  /autorun",5d24868cac87dcd70c5b71101d
39b0de,f5a95f9ec0bf4e3cff92579f65e9bbe512fc3a20,6a58ccd257ec133dcc72b245913dc96a
aa8ff03269a51e68bc16de1e7451eb13
315d312
< "HKCU\Software\Microsoft\Windows\CurrentVersion\Run","swg",enabled,"GoogleTool
barNotifier","(Verified) Google Inc","c:\program files\google\googletoolbarnotif
ier\googletoolbarnotifier.exe","""C:\Program Files\Google\GoogleToolbarNotifier\
GoogleToolbarNotifier.exe""",5d61be7db55b026a5d61a3eed09d0ead,215950ce5d40907b04
1346f22b4e404ee591581d,d32cc7b31a6f98c60abc313abc7d1143681f72de2bb2604711a0ba207
10caaae
317,318d313
< "HKLM\Software\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Object
s","Google Toolbar Helper",enabled,"Google Toolbar","(Verified) Google Inc","c:\
program files\google\google toolbar\googletoolbar_32.dll","HKCR\CLSID\{AA58ED58-
01DD-4d91-8333-CF10577473F7}",006ef177f94a066eca9011475d556ca9,99741b79ffcdd2813
8ae01ed93a221fe556ad5ac,8beff90df01eba51979f9b83ca2eb7a003dc7cd10362cb481031dd44
6473a443
< "HKLM\Software\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Object
s","Google Toolbar Notifier BHO",enabled,"GoogleToolbarNotifier","(Verified) Goo
gle Inc","c:\program files\google\googletoolbarnotifier\5.2.4329.1504\swg.dll","
HKCR\CLSID\{AF69DE43-7D58-4638-B6FA-CE66B5AD205D}",6d424fdd2c73c7ace4c64c18bacf2
421,1af629cf2d2fedfa606c3210e2e3cef034163837,a4f576f2558aa9e9bad5f0ccf84ce0b0637
cea34b5cf337e1af06791cc478d40
513d507
< "HKLM\Software\Microsoft\Internet Explorer\Toolbar","Google Toolbar",enabled,"
Google Toolbar","Google Inc","c:\program files\google\google toolbar\googletoolb
ar_32.dll","HKCR\CLSID\{2318C2B1-4965-11d4-9B18-009027A5CD4F}",006ef177f94a066ec
a9011475d556ca9,99741b79ffcdd28138ae01ed93a221fe556ad5ac,8beff90df01eba51979f9b8
3ca2eb7a003dc7cd10362cb481031dd446473a443
```

Figure 8

Our output indicates the six items listed were removed from third.txt. The “<” symbol indicates they were in the second.txt file and no longer in the third.txt file. If we compare the items from this output to the output after our install, we can see the same items that were added (“>”) after the install were deleted (“<”) after the uninstall.

As we progress from here, we should keep our file order in our diff.exe command constant, such as “diff.exe {previous snapshot} {current snapshot}”. By doing so we can consider lines starting with “>” as additions and lines starting with “<” as deletions.

We can see that by using Autoruns to make frequent snapshots and Diff to compare them, we can detect changes in the auto-run areas on our Windows systems. Now, let’s talk about how we can automate this process.

2.5. Bringing it all together with scripting techniques

2.5.1. Scripting Autorunsc and Diff

Now that we have covered the basic concept of making and comparing snapshots of Windows auto-run areas, let's talk a little about automating the process. Continuing to do this as a manual process would be too inefficient and tedious, especially on a system where not many changes are expected. If we have empty reports day after day, we may become complacent and stop running the process on a regular basis. What we need to do now is put these commands to work in a script.

We have created and used two commands from the command prompt, one to dump all auto-run information and another to compare snapshots of this information. We need to consider how we would go through the whole process of running our commands while maintaining a current snapshot and the previous snapshot we need for a historical comparison. To start building our automated process, let's put the process together in the logic we would process our tasks.

When working out our automation logic we should be thinking along the lines of having the following tasks:

- Initialize constants and variables – this will be the first part of our script that will initialize any variables and constants we want to use. This will include working paths, filenames, etc.
- Make current snapshot – this is where we will issue the Autorunsc.exe command to make our current snapshot. We will redirect the output to a snapshot file so we can compare it to an older snapshot file.
- Compare snapshots – in this section we will issue the diff.exe command to compare our snapshots. At this point, you may be wondering "where is this older snapshot we are comparing to our current snapshot?" We will cover file handling coming up shortly.
- Build report – here is where we will parse the results of our diff.exe command and add the information we want to our report.
- Display report – finally we will want to output the report in some manner.

Before we start discussing a script to accomplish these tasks, there are a few other considerations that we need to think about. As mentioned earlier, what about file handling? How do we manage the files we are creating so we have that previous snapshot? We also need to think a little about report logic. What exactly do we want to see in the report? What about error handling? What types of errors can we account for and capture in our script to make future troubleshooting easier.

File Handling: In our script, we are relying on two snapshots to help us detect malware infections. One of the current auto-run state and one from a previous state at some point in time. To have a previous snapshot, what better to use than the last snapshot we made for our current snapshot? Which brings us to a file management decision- do we want to manage files before the current snapshot is made or afterwards?

To make this decision, let's look at a snapshot rotation method where the current snapshot becomes the previous snapshot. We already know we need two snapshots to do a comparison. Let's call the files snap.curr (current snapshot) and snap.last (previous snapshot). If we want to set up our files for the next run, we will want to replace the previous snapshot with the current snapshot. We will also want to delete the current snapshot file to prepare for our new snapshot file. To accomplish this we can delete the snap.last file and then rename snap.curr to snap.last.

We have two options for the timing of this file preparation step. We can perform it either at the start or the end of our script. One thing to consider in the decision of where to make the rotation is snapshot retention. If we rotate the two files at the end of the script, we lose our previous snapshot right away and are only left with the current one (now named as the previous). If we rotate them at the start of the script, we retain our two snapshots until the next time the script executes. Retaining the snapshots for as long as possible will be beneficial if any investigation of the snapshots is required due to a report entry. For this reason, we will be rotating the files at the start of our script.

One last thing to consider for file handling is what happens the first time the script runs or the previous snapshot file is missing. At first run, there will not be a previous snapshot in which to compare the current snapshot and we must account for this. The first part of the file rotation task should check for an existing snapshot file (snap.curr). If it is missing we should create one so our diff command will have two files to work with. By making this check we will cover both situations, a first run and a run where the file is missing.

This is the most basic file handling that needs to be accomplished. You can become as elaborate as you need with snapshot history. It's just a matter of translating your desires into the script.

Report Logic: While building the report from your difference file you need to consider what is important for you to see. Our script will report any addition or deletion entry. However, this may be too much information in some environments. Once again, you can make the report as elaborate or as simple as needed for your environment. There may be times when there are no changes between snapshots, so you may not want to display a report at all. The "what" and "when" is completely up to you. We will be discussing report examples later in an effort to make the decisions easier. Refining the process over time will be a common task.

Error handling: Our script in this paper has some basic error handling built in to it. The error handling in our script is for assuring the necessary folder structure is in place. You should consider what other errors may arise in your environment and add/remove whatever error handling you desire.

Environments vary from place to place, so should the script included in this paper. You should take the script and customize it to your needs. However, if you feel this is beyond your ability, the script can be implemented as-is.

There are a wide variety of scripting languages and programming styles. This script is basic and can be rewritten in your preferred language and style. However, in our analysis we will be looking at a VBScript implementation.

2.5.2. Complete working VBScript

In the appendix, section 4, we have the complete steps to implement our process on a stand-alone computer. For the script to function properly, we need to create the file system structure as documented in the Appendix, section 4.1. Next we need to download the supporting files (Autoruns and Diff) in accordance with the directions in the

Appendix, sections 4.2 and 4.3. Finally we will need to copy the VBScript, in Appendix section 4.4, into a file called ARCheck-SA.vbs or whatever you wish to call it.

We should probably use a name that has some meaning when we see it. ARCheck-SA means "**A**uto-**R**un **Check**er-**S**tand-**A**lone". The ".vbs" file extension will associate the file with the Windows Scripting Host, which is important. You should not deviate from this unless you are an advanced user and know what you are doing.

When we discussed our automation logic we defined several tasks that were needed to accomplish the process we are creating. We touched on five main tasks: define variables, make snapshot, compare snapshots, build report and display report. Then we discussed tasks for file and error handling. If you look at our script in the appendix, you can pull out the following six tasks we will discuss while explaining the script.

1. Variables – look for comment "Set up variables and constants"

At the start of our script we define several variables and constants. We assign default, or starting, values to these variables. Throughout our script you will see naming conventions used for our variables and constants. These naming conventions are used so we can help identify what type of data may be stored in the variable. The following list of three character variable prefixes describes the value types stored in our variables:

- boo – Boolean variable for TRUE or FALSE values.
- int – Integer variable to hold numeric values.
- str – String variable to hold string data.
- obj – Object variable to hold a reference to a system object. Examples of objects used are: System information objects to pull host and user names; File System objects to access files on a file system; and Shell objects to run our commands.

The constants and variables in the script are explained in the Appendix, section 4.5.

2. File and error handling – look for comment "Verify file system is in order"

After we get our variables and constants configured, we can start with our file management portion of the script. While performing file management, the script starts performing some error checking on the folder structure it requires. Our script relies on a certain folder structure to be in place to write temp files, manage our snapshot files and save our report files.

The first thing our script does in the file handling section is to check to see if the strOutputPath exists. If it doesn't exist, the script will attempt to create the folder. At this point, our error handling kicks in and checks to see if the folder creation was successful. If it was not successful, we call the ErrHandler procedure passing it the error number, a description of "Output Path" and the strOutputPath variable. The procedure will display an error and the script will exit.

Once we make sure the strOutputPath is establish, we do the same check on the strReportPath. If it doesn't exist, we attempt to create the path and check for creation errors. If there was an error, we once again call the ErrHandler procedure passing it the error number, a description of "Report Path" and the strReportPath variable. The procedure will display an error and the script will exit.

Once we know our folder structure is in place, we prepare the snapshot files by performing the file rotation as we discussed in the file handling logic. We delete the previous snapshot file (strPreviousFile). Then we move the old current snapshot file (strCurrentFile) to make it the previous snapshot (strPreviousFile). Or if the file is missing, we create an empty previous snapshot file to compare with the current snapshot we are about to make.

3. Make snapshot – look for comment "Make snapshot"

Once we know the folder structure requirements are in order and the snapshot rotation has completed, we can run our Autorunsc.exe command to create our current snapshot file. We set the strCommand variable with the Autorunsc.exe command we created. We the call the RunCommand procedure passing it the strCommand variable, the window style of 0 (zero) to run as a hidden window and the booWait variable to have the script wait for the program to complete before continuing.

One caveat we in the script, that we haven’t discussed yet, is the added (and undocumented) command line parameter “/accepteula”. This parameter will prevent a first time use license agreement from being display and waiting for the user to accept the agreement. If we do not use this command line option, it may prevent our script from running and ultimately detecting malware.

4. Compare snapshots – look for comment “Compare snapshots”

Now that we have made a new snapshot, we are ready to compare the snapshots and make a difference file for our report building task. We once again set the strCommand variable with the diff.exe command we created. The RunCommand procedure is called and passed the strCommand variable, the window style of 0 (zero) to run as a hidden window and the booWait variable to have the script wait for the program to complete before continuing.

5. Build report – look for comment “Build Report”

The report building task can now process our difference file and we can build a list of entries to report. In this task we build the report header, parse the difference file creating the report body and then write the report body to the report file.

The first part of this task opens an object to the report file for writing the report of our findings. Once the file is open a header is written to the file based on the information we have collected from the system. This information includes hostname, username, version number, a notice if there is no previous history, the time the report ran and the report filename.

After we build the header, we open an object to read the difference file. We parse the difference file line by line, or entry by entry, storing the entry in the strOutput variable. We use the start of each entry to determine what part of the report the entry will be written, if it is written at all. The script adds entries starting with a “>” symbol to the strAdditions variable and entries starting with a “<” symbol to the strDeletions variable. Other entries are ignored and not reported in any output.

Finally the script writes the strAdditions variable to our report file. Unless there are no entries to report, in which it will write an entry saying there are "No deletions to report." This is followed by a similar process for the strDeletions variable.

6. Display report – look for comment "Display Report"

During the report building process we used a boolean variable called booChanges to track if changes were discovered. If there were any additions or deletions detected in the difference file this variable was set to TRUE. If no additions or deletions were detected then this variable was set to FALSE.

The strCommand variable is set to run notepad.exe with our report file as input. If the booChanges variable is set to TRUE then the script calls the RunCommand procedure. The procedure is passed the strCommand variable, a 7 for window style (normal window in the background) and the booNoWait variable. The 7 to open notepad.exe in the background is to prevent interruption of anything the user may be working on. The booNoWait variable is to have the script continue and complete without waiting on the user to close the notepad program.

Just a few more small tasks to clean up our system resources

After the report is built and displayed, there are a couple more items the script does before exiting. The script cleans up the difference file that we do not need to keep by deleting the file located in the variable strDiffFile. It closes the system objects it had open and then calls the ExitScript procedure to end the processing.

Extra Code found at the end of the script: Procedures

At the end of the script you will find the scripting for the procedures used by the script. There are three of them; ErrHandler, ExitScript and RunCommand.

The ErrHandler procedure is used when the script catches an error that it is programmed to handle. The script is programmed to handle file errors when we are checking our required file system structure. If a file error is encounter the script calls this procedure and passes three variables; iErrNum – the built-in Err number as reported by WSH, strErrDesc – a programmer description of the error and strErrFile – the path that was being used when the error occurred. The procedure uses this information to display an error message and then it calls the ExitScript procedure.

The ExitScript procedure does just what its name indicates. It causes the script to terminate. It first frees the open file system object (objFSO), then it calls the WSH quit command to terminate the script gracefully.

The RunCommand procedure also does just what its name indicates. It runs a command, stored in the strCmd variable, which is passed to the procedure when called. In addition to the command, the procedure is passed two other variables; intWinStyle and booWaitDone. The intWinStyle variable defines the style of the window to run the command. Our script uses two styles; 0 - hidden and 7 - normal window in background. The booWaitDone variable is a boolean variable that determines if script processing waits for the command to finish or not.

2.6. Change detection

2.6.1. Detecting a change

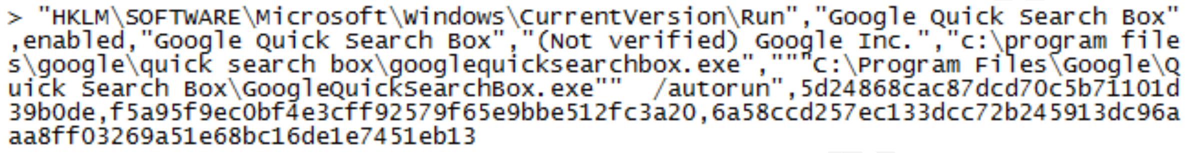
After running this script regularly over a period of time we will start seeing various reports being generated. Obviously, the reports that tell us “No changes were detected…” are the reports we like to see. However, this will not always be the case. We will see reports that have changes that are normal due to software installations, updates or uninstallations. We will see reports alerting us of changes because items a user may modify, such as their Startup folder or Scheduled Tasks. Unfortunately, we may also see reports that are due to malware activity.

As time goes by, hopefully we all will get better at understanding what is causing the reports to be generated. For now, let’s discuss some of the items we may see in our reports. Then we will move into some of the things that can be done to help determine what is going on.

As we get into the examples, let’s remember an important part of creating our diff command. If we keep the input files in the same order, previous snapshot file then current snapshot file, we can quickly determine if the entry is an addition or deletion. Using this order, any line starting with a “>” symbol will be an addition and any line starting with a "<" symbol will be a deletion. The report is designed to keep additions grouped together in one section and deletions in another.

Install/Uninstall Example:

For the first example, we will revisit our Google Toolbar install and uninstall that we discussed earlier. If you look back at the output from the diff command we used in Figure 6, which we used to compare snapshots around our Google Toolbar install, you will see the second entry of output is the same as you see in Figure 9.



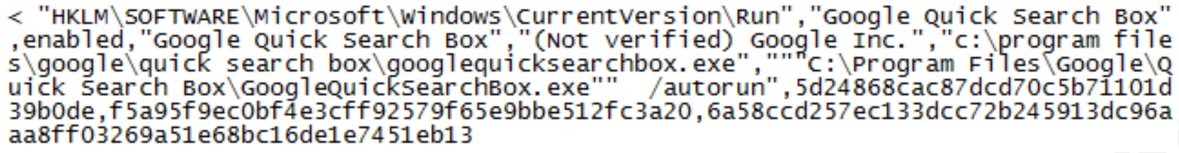
```
> "HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run","Google Quick Search Box"
,enabled,"Google Quick Search Box","(Not verified) Google Inc.","c:\program file
s\google\quick search box\googlequicksearchbox.exe","""C:\Program Files\Google\Q
uick Search Box\GoogleQuickSearchBox.exe""  /autorun",5d24868cac87dcd70c5b71101d
39b0de,f5a95f9ec0bf4e3cff92579f65e9bbe512fc3a20,6a58ccd257ec133dcc72b245913dc96a
aa8ff03269a51e68bc16de1e7451eb13
```

Figure 9

Our general analysis by reading the entry in Figure 9, as ordered, tells us:

- The entry is an additional auto-run entry as the ">" symbol indicates
- The Entry Location is "HKLM\SOFTWARE\Microsoft\Windows\Current Version\Run"
- The Entry is called "Google Quick Search Box"
- The entry's status is "enabled"
- It has a Description of "Google Quick Search Box"
- The Publisher is "Google Inc." and the digital signature was "(Not Verified)"
- The application location, or Image Path, is:
  "c:\program files\google\quick search box\googlequicksearchbox.exe".
- The application, or Launch String, that gets executed is:
  "C:\Program Files\Google\Quick Search Box\GoogleQuickSearchBox.exe /autorun"
- Finally, Autoruns gives us the three hashs (MD5,SHA-1,SHA-256) for the application "googlequicksearchbox.exe":
  "5d24868cac87dcd70c5b71101d39b0de"
  "f5a95f9ec0bf4e3cff92579f65e9bbe512fc3a20"
  "6a58ccd257ec133dcc72b245913dc96aaa8ff03269a51e68bc16de1e7451eb13"

As we recall, the uninstallation of the Google Toolbar produced very similar results as compared to the installation results. As you see in Figure 10, the only difference in the uninstall output is at the start of the entry. In the uninstall results, we see the “<” symbol which tells us that the entry was a deletion.

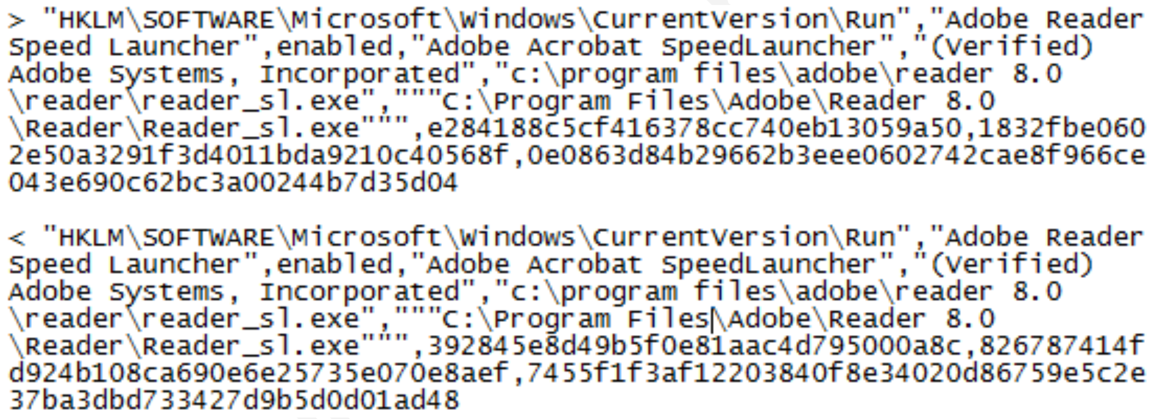


```
< "HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run","Google Quick Search Box"
,enabled,"Google Quick Search Box","(Not verified) Google Inc.","c:\program file
s\google\quick search box\googlequicksearchbox.exe","""C:\Program Files\Google\Q
uick Search Box\GoogleQuickSearchBox.exe""  /autorun",5d24868cac87dcd70c5b71101d
39b0de,f5a95f9ec0bf4e3cff92579f65e9bbe512fc3a20,6a58ccd257ec133dcc72b245913dc96a
aa8ff03269a51e68bc16de1e7451eb13
```

Figure 10

Upgrade Example:

Considering how the reports look for installing and uninstalling software, how do you think the reports for updating software will look? If you are thinking there will be an addition entry and a deletion entry, you are correct. When software gets updated, you will often see coinciding entries as in Figure 11. There will be one entry that removes the old program information and another entry that adds the new program information.



```
> "HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run","Adobe Reader
Speed Launcher",enabled,"Adobe Acrobat SpeedLauncher","(Verified)
Adobe Systems, Incorporated","c:\program files\adobe\reader 8.0
\reader\reader_sl.exe","""C:\Program Files\Adobe\Reader 8.0
\Reader\Reader_sl.exe""",e284188c5cf416378cc740eb13059a50,1832fbe060
2e50a3291f3d4011bda9210c40568f,0e0863d84b29662b3eee0602742cae8f966ce
043e690c62bc3a00244b7d35d04

< "HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run","Adobe Reader
Speed Launcher",enabled,"Adobe Acrobat SpeedLauncher","(Verified)
Adobe Systems, Incorporated","c:\program files\adobe\reader 8.0
\reader\reader_sl.exe","""C:\Program Files\Adobe\Reader 8.0
\Reader\Reader_sl.exe""",392845e8d49b5f0e81aac4d795000a8c,826787414f
d924b108ca690e6e25735e070e8aef,7455f1f3af12203840f8e34020d86759e5c2e
37ba3dbd733427d9b5d0d01ad48
```

Figure 11

In the report output, we see two entries that look very similar to one another. This output comes from a comparison of snapshots made before and after an update to the Adobe Reader Software. After analyzing the output of these two entries, we can deduce the following:

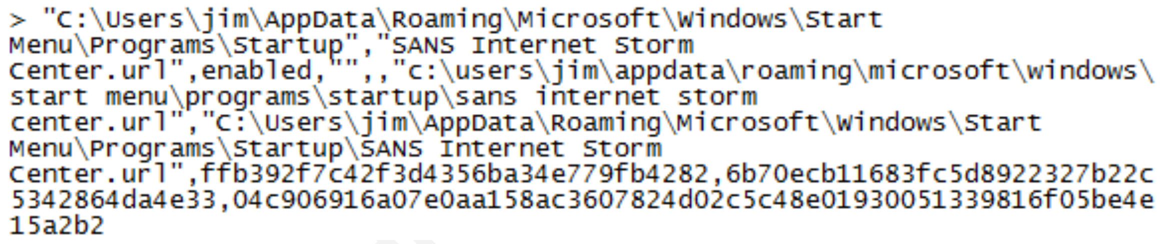
- The starting symbols (“>” and “<”) indicate one is an addition and one is a deletion.
- The location, entry, status, description, publisher, image path and launch path are identical for both entries.
- The application’s hash values (MD5, SHA-1 and SHA-256) are different from one entry to the other. This tells us that the application file has been modified.

In this software update example, the path to the application remained the same. Some applications may show a different path for the updated install, especially if the version of the application is used in the path. The path may change for other reasons as well, such as a user defined installation location.

Be careful not to just dismiss these types of entries as upgrades. Due to the nature of parasitic viruses, the hash may change because malware modified the file. If an unexpected entry appears, be sure you scrutinize it thoroughly using the methods we talk about a little later.

Start Folder Example:

In Figure 12, we see how a report entry will look if a user creates an entry in their Startup folder. You will see in this example that user Jim created a URL shortcut in his User Profile startup folder.



```
> "C:\Users\jim\AppData\Roaming\Microsoft\Windows\Start
Menu\Programs\Startup","SANS Internet Storm
Center.url",enabled,"",,"c:\users\jim\appdata\roaming\microsoft\windows\
start menu\programs\startup\sans internet storm
center.url","C:\Users\jim\AppData\Roaming\Microsoft\Windows\Start
Menu\Programs\Startup\SANS Internet Storm
Center.url",ffb392f7c42f3d4356ba34e779fb4282,6b70ecb11683fc5d8922327b22c
5342864da4e33,04c906916a07e0aa158ac3607824d02c5c48e01930051339816f05be4e
15a2b2
```

Figure 12

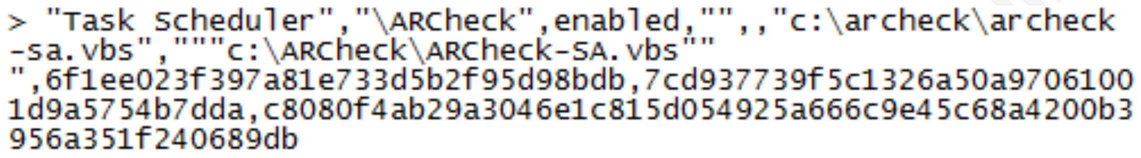
From the output we can determine:

- The starting symbol (“>”) indicates this is an addition.
- The location “C:\Users\jim\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup” tells us this is the start menu for a user profile on a Windows Vista or 7 system. On an XP system the folder would be “C:\Documents and Settings\jim\Start Menu\Programs\Startup”.
- The entry is named “SANS Internet Storm Center.url”.
- The image path and launch string is “C:\Users\jim\AppData\Roaming\Microsoft \Windows\Start Menu\Programs\Startup\SANS Internet Storm Center.url” which could tell us this entry is a URL shortcut.

We could assume this entry is a shortcut to the SANS Internet Storm Center website, but we are basing that on the name of the shortcut we see in our output, which is a bad approach! To determine what this shortcut does, we would have to investigate the shortcut itself.

Scheduled Task Example:

In Figure 13, we see an example of what you will see in a report when a new Scheduled Task is created. In this example, a task called "ARCheck" was created to run "c:\ARCheck\ARCheck-SA.vbs". From this point we could go investigate the Scheduled Tasks on the system to see what the VBScript will do when it runs.

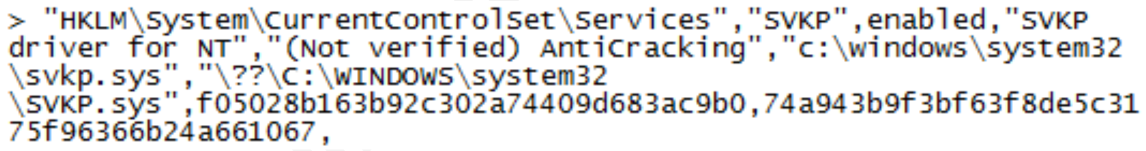


```
> "Task Scheduler","\ARCheck",enabled,"",,"c:\archeck\archeck
-sa.vbs","""c:\ARCheck\ARCheck-SA.vbs""
",6f1ee023f397a81e733d5b2f95d98bdb,7cd937739f5c1326a50a9706100
1d9a5754b7dda,c8080f4ab29a3046e1c815d054925a666c9e45c68a4200b3
956a351f240689db
```

Figure 13

Malware Infection Example:

In this example, figure 14, we see what an entry in our report would resemble if it detects malware. With malware entries there is not very much commonality from one report to another. Often with the entry you will see something that stands out, but not always.



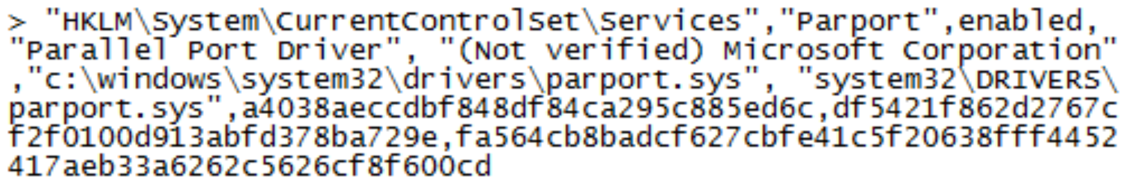
```
> "HKLM\System\CurrentControlSet\Services","SVKP",enabled,"SVKP
driver for NT","(Not verified) AntiCracking","c:\windows\system32
\svkp.sys","\??\C:\WINDOWS\system32
\SVKP.sys",f05028b163b92c302a74409d683ac9b0,74a943b9f3bf63f8de5c31
75f96366b24a661067,
```

Figure 14

Some malware writers try very hard to make their malicious code look like it is suppose to be there. They use official looking names, like "Security Tool" or "AntiVirus 2009". This is where our investigation process is important, especially for entries we do not know. We will discuss investigation and research after we discuss one more report item.

Unsigned Microsoft DLL Example:

From time to time, you may run across entries that have Microsoft Corporation as the Publisher but the signature is "Not Verified". In figure 15, we see an example of a Windows XP Parallel Port driver that is not digitally signed.



```
> "HKLM\System\CurrentControlSet\Services","Parport",enabled,
"Parallel Port Driver", "(Not verified) Microsoft Corporation"
,"c:\windows\system32\drivers\parport.sys", "system32\DRIVERS\
parport.sys",a4038aeccdbf848df84ca295c885ed6c,df5421f862d2767c
f2f0100d913abfd378ba729e,fa564cb8badcf627cbfe41c5f20638fff4452
417aeb33a6262c5626cf8f600cd
```

Figure 15

Sometimes we will run across files from Microsoft, or other vendors, that are not signed. This may occur when the files are from an older software implementation. The more current the software is, the more likely it will have files that are signed.

However, we will need to research these entries (covered in the next section). At this point we do not know why the entry is “Not Verified”. It could be that Microsoft didn’t sign the file, the file is signed with a fake certificate or the file has been modified by malware.

Other examples:

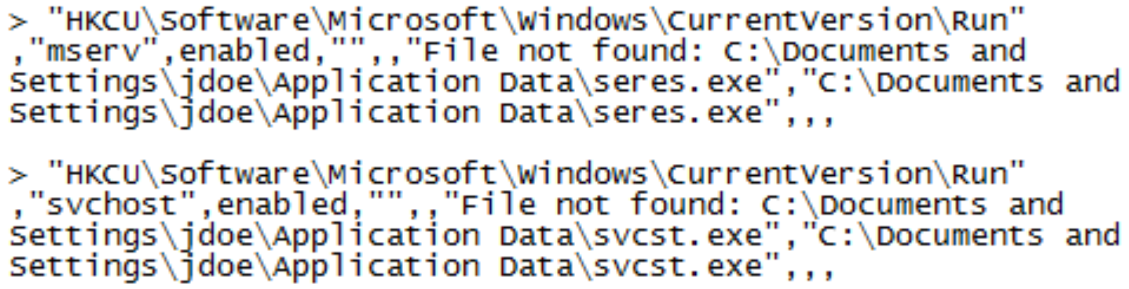
There are also times we will see “orphaned” entries in our reports. What do we mean by “orphaned” entries? Well, from time to time we will see entries that have an image path that is reporting “File not found: {file system path}”. There are a few situations where this can occur.

One situation where this can happen is when the user account running the script does not have permissions to the location of the image path. At times you will need to check the image path with an account that has appropriate permissions to see if the file actually exists. Sometimes the file may no longer exist because it was deleted, by a user or an application.

When an application gets uninstalled, not all of the uninstall programs remove all of the registry entries, causing registry rot. At times you may find these entries in your report after an application uninstall. If you do, you may have to manually delete the registry entry. However, be very cautious when doing so. Manually editing the registry can cause many problems for a Windows system. If you decide to edit the registry, always consider backing it up first.

There is one scary situation to think about in a report. When our system is infected with a trojan or rootkit, the malware may hide files from the Windows APIs (application program interfaces). Some really evil pieces of malware put hooks into the Windows processes that retrieve file, registry and process information. These hooks are used to sanitize the information that Windows is reporting back to the requesting program. So when Autoruns requests file information about an entry, the Windows processes, influenced by the malware, may report that a file doesn’t exist even when it does. At that point Autoruns doesn’t think the file exists and tells us “file not found”.

The last situation we are going to discuss is when anti-virus software detects and removes a piece of malware. When anti-virus software detects a piece of malware it can often remove the executables the malware has planted on the system. Unfortunately, anti-virus software often leaves registry entries that were created by the malware infection. In Figure 16, we can see entries from a computer that was infected with malware and then caught and removed by anti-virus software.



```
> "HKCU\Software\Microsoft\Windows\CurrentVersion\Run"
,"mserv",enabled,"",,"File not found: C:\Documents and
Settings\jdoe\Application Data\seres.exe","C:\Documents and
Settings\jdoe\Application Data\seres.exe",,,

> "HKCU\Software\Microsoft\Windows\CurrentVersion\Run"
,"svchost",enabled,"",,"File not found: C:\Documents and
Settings\jdoe\Application Data\svcst.exe","C:\Documents and
Settings\jdoe\Application Data\svcst.exe",,,
```

Figure 16

These entries appeared in an initial report from the previously infected computer. If this hadn’t been the initial report, our process would have caught the virus shortly after the infection had occurred. The investigation into these files proved the files no longer existed and the “File not found” message was not due to permissions or malware trickery.

How do we know these files were part of a previous malware infection? Other than knowing from experience, we can research these entries using trusted Internet resources.

2.6.2. How can I research items found in my reports?

Often we will see things in our reports that will look suspicious. Fortunately, there are some good research sites available for us to utilize in our investigation efforts.

We will discuss two decent options for malware research: ThreatExpert.com and VirusTotal.

“ThreatExpert is an advanced automated threat analysis system designed to analyze and report the behavior of computer viruses, worms, trojans, adware, spyware, and other security-related risks in a fully automated mode.

In only a few minutes ThreatExpert can process a sample and generate a highly detailed threat report with the level of technical detail that matches or exceeds antivirus industry standards such as those normally found in online virus encyclopedias.” (ThreatExpert Ltd, 2009)

A search at the ThreatExpert web site for the files in Figure 16, seres.exe and svcst.exe, indicates the files are mostly identified as a threat. This is handy information to know in our investigation. It may prompt us to research further on the malware in which ThreatExpert indicates these files are associated. Researching your anti-virus software’s website and working with the vendor may be a good idea.

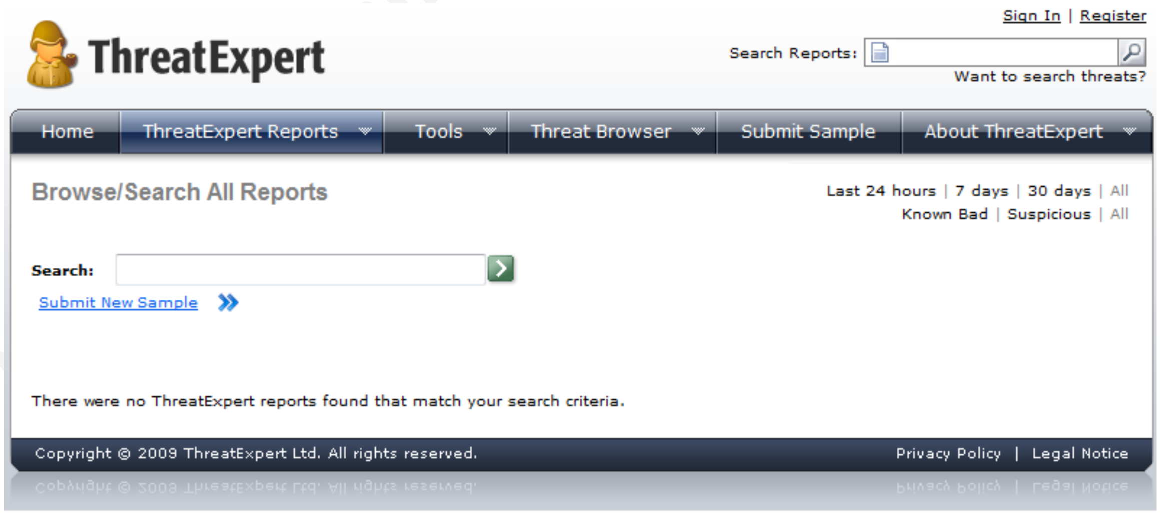
Unfortunately in the example in Figure 16, the files were really gone and not available for further investigation. But be aware that some malware files mask their existence from being detected. Just because the report tells you “File not found” doesn’t mean the file doesn’t exist.

Let’s look at a more recent report and see why we cannot rely on just one Internet resource for our investigation. Figure 17 shows us some interesting files as additions to the auto-run area of our Windows system.

```
> "HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run"
,"06529123",enabled,"",,"c:\documents and settings\all
users\application data\06529123\06529123.exe","C:\DOCUME~1
\ALLUSE~1\APPLIC~1\06529123
\06529123.exe",ea14ddb220fb9b22f2cd0721ac24cdb3,d809d0e5321
952b225fb057c5c17354949809d95,a0c8a3f558114ff99d9f224eef056
4e40f7bbaa592d5dc403eb7ca1c91e16edb

> "HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run"
,"75139732",enabled,"",,"c:\documents and settings\all
users\application data\75139732\75139732.exe","C:\Documents
and Settings\All Users\Application Data\75139732
\75139732.exe",b6813e0e437c339e5514ec431862a5f7,56da96181e7
1203353731a226b3752fd366b22c0,c1ffceacd927ea9abbd4013a966b6
c135fb4986c2cf6271aaf4b728acb10850d
```

Figure 17

Here we see two new entries under the Run registry entry. One called "06529123" and the other called "75139732", both being enabled. Neither of them have a publisher, so they are not digitally signed. The executables for each are found in the All Users profile. These files should have our curiosity running wild. Programmers usually give their programs descriptive names and signing executable files is becoming very standard these days (unfortunately not standard enough). But still, the naming convention is not very descriptive which stands out. It almost seems like random naming, which is common in malware detection avoidance techniques. A search for the file names on ThreatExpert.com returns nothing, as seen in Figure 18.



There were no ThreatExpert reports found that match your search criteria.

Copyright © 2009 ThreatExpert Ltd. All rights reserved.

Figure 18

With ThreatExpert not giving us any information on these files, let's take a look at VirusTotal.com as a resource. "VirusTotal is a service that analyzes suspicious files

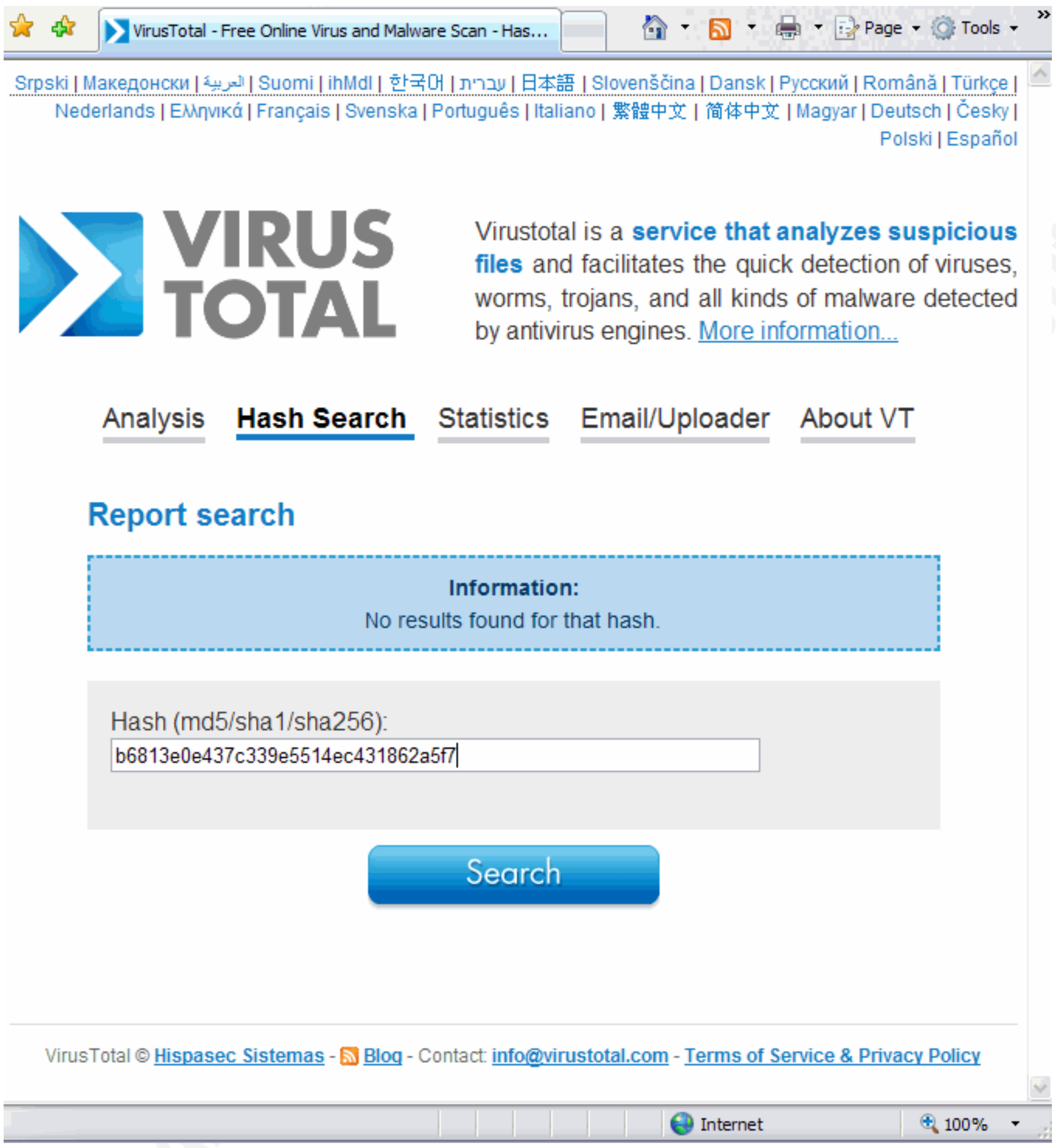
Author Name, email@address

and facilitates the quick detection of viruses, worms, trojans, and all kinds of malware detected by antivirus engines." (VirusTotal)

VirusTotal utilizes antivirus engines from about 40 different companies. Queries can be performed via a web interface in two manners: 1) by submitting a file directly, or 2) via a hash search. The hash search queries a database of hashes created from other files submitted to VirusTotal. If you submit a file to VirusTotal, the file properties, including hashes, will be stored in VirusTotal's database. The file, if not already in the database, will then be scanned by over 40 different malware products (commercial and open source). If the file is in the database, you can view the previous results or have it rescanned.

Remember, hashes are used to verify that no changes have been made to a file. Hashes are based on file content. If you change the contents of a file, the hash of the file will change as well. If a hash of one file matches the hash of another file (regardless of filename), there is a very high probability that the files are identical.

In our output in Figure 17, we have hash values for our files. If we go to the VirusTotal website and submit our hashes, we can see if VirusTotal is aware of our files. In Figure 19, we can see what it looks like when VirusTotal is not familiar with a hash.


VirusTotal - Free Online Virus and Malware Scan - Has...
Page
Tools
Srpski | Македонски | العربية | Suomi | ihMdl | 한국어 | עברית | 日本語 | Slovenščina | Dansk | Русский | Română | Türkçe | Nederlands | Ελληνικά | Français | Svenska | Português | Italiano | 繁體中文 | 简体中文 | Magyar | Deutsch | Česky | Polski | Español
VIRUS TOTAL
Analysis
Hash Search
Statistics
Email/Uploader
About VT
Search
Internet
100%

Virustotal is a **service that analyzes suspicious files** and facilitates the quick detection of viruses, worms, trojans, and all kinds of malware detected by antivirus engines. More information...

Report search

Information:
No results found for that hash.

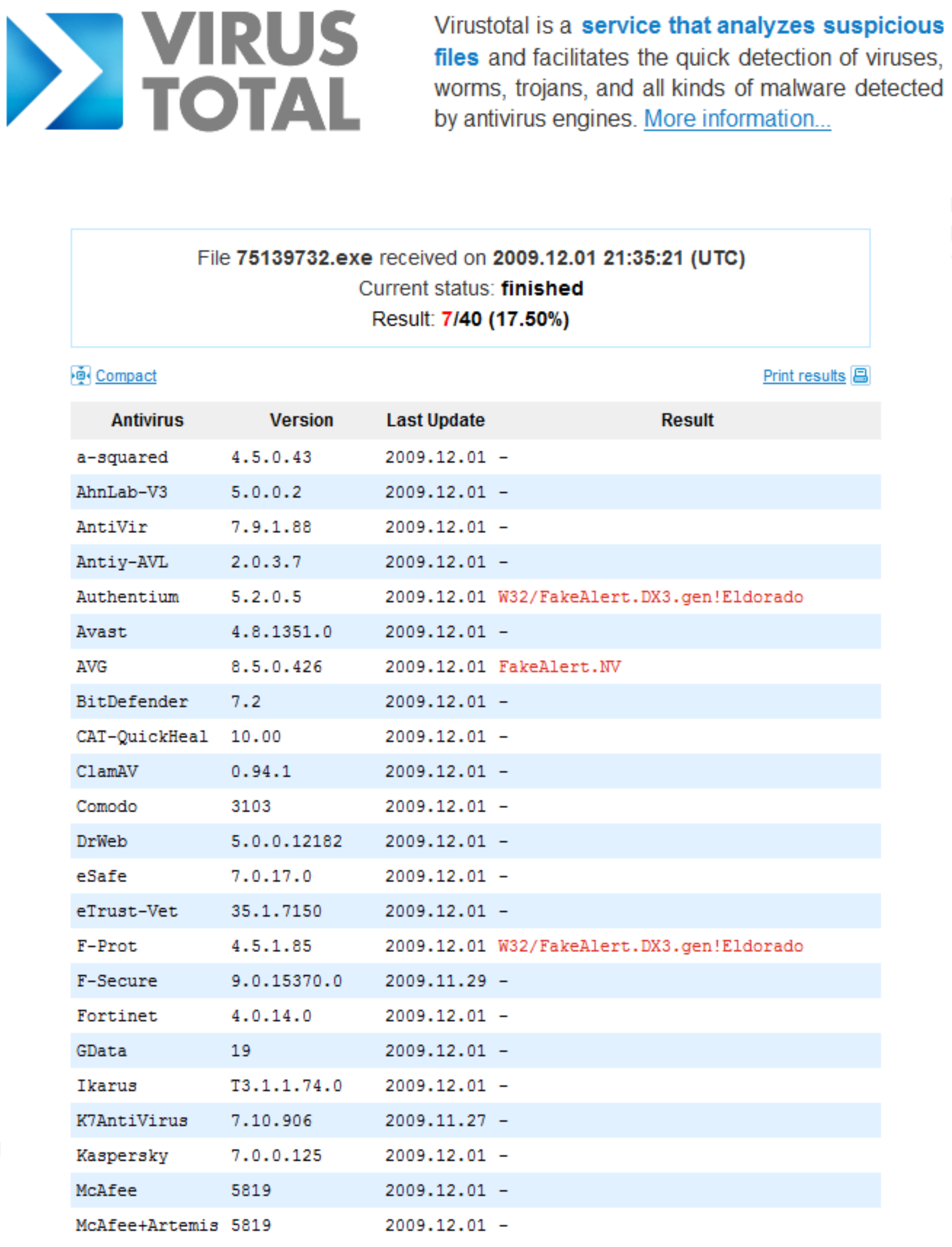
Hash (md5/sha1/sha256):
b6813e0e437c339e5514ec431862a5f7

VirusTotal © Hispasec Sistemas - Blog - Contact: info@virustotal.com - Terms of Service & Privacy Policy

Figure 19

Since we did not get a hit on the hash value of our files, we can submit the actual files to VirusTotal. If we get the executable file and submit it through the VirusTotal website, we will basically run it against 40 anti-virus tools to see if anyone detects it as malware. Figure 20 shows partial results from VirusTotal on the file 75139732.exe, the same as the hash from Figure 19.

Author Name, email@address



Virustotal is a service that analyzes suspicious files and facilitates the quick detection of viruses, worms, trojans, and all kinds of malware detected by antivirus engines. More information...

File **75139732.exe** received on **2009.12.01 21:35:21 (UTC)**
Current status: **finished**
Result: **7/40 (17.50%)**

Compact   Print results

Antivirus	Version	Last Update	Result
a-squared	4.5.0.43	2009.12.01	-
AhnLab-V3	5.0.0.2	2009.12.01	-
AntiVir	7.9.1.88	2009.12.01	-
Antiy-AVL	2.0.3.7	2009.12.01	-
Authentium	5.2.0.5	2009.12.01	W32/FakeAlert.DX3.gen!Eldorado
Avast	4.8.1351.0	2009.12.01	-
AVG	8.5.0.426	2009.12.01	FakeAlert.NV
BitDefender	7.2	2009.12.01	-
CAT-QuickHeal	10.00	2009.12.01	-
ClamAV	0.94.1	2009.12.01	-
Comodo	3103	2009.12.01	-
DrWeb	5.0.0.12182	2009.12.01	-
eSafe	7.0.17.0	2009.12.01	-
eTrust-Vet	35.1.7150	2009.12.01	-
F-Prot	4.5.1.85	2009.12.01	W32/FakeAlert.DX3.gen!Eldorado
F-Secure	9.0.15370.0	2009.11.29	-
Fortinet	4.0.14.0	2009.12.01	-
GData	19	2009.12.01	-
Ikarus	T3.1.1.74.0	2009.12.01	-
K7AntiVirus	7.10.906	2009.11.27	-
Kaspersky	7.0.0.125	2009.12.01	-
McAfee	5819	2009.12.01	-
McAfee+Artemis	5819	2009.12.01	-

Figure 20

The output in Figure 20 has been cut off, but we can see that only 7 out of 40 utilities flag this file as possibly being malware. The next step would be to submit these programs to your anti-virus vendors and/or other security sites such as TheatExpert and SANS Internet Storm Center (http://isc.sans.org).

VirusTotal offers a couple other options for submitting files that may be of interest. They have a small piece of software called VirusTotal Uploader that you can install on a computer. Then there is an option to make submissions via e-mail.

The Uploader program adds functionality to the "Send To" feature. After it is installed you can right click on a file, choose "Send To" and there will be a new option called VirusTotal, as seen in Figure 21. After selecting VirusTotal, the program will check the hash and either: 1) Open a browser to the hash results page or 2) Send the file and open the browser to the submission results page.

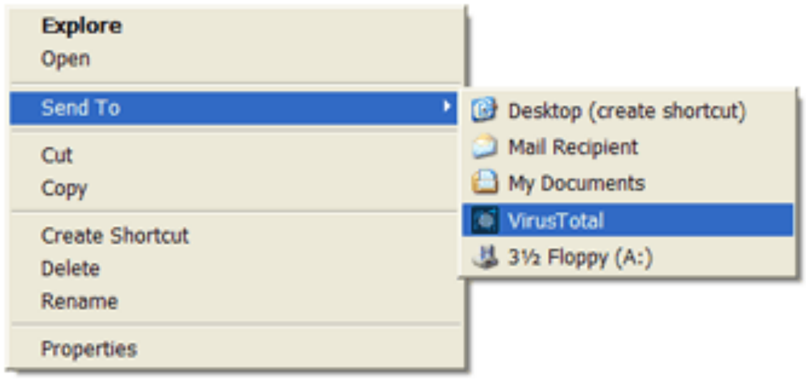


Figure 21

The option to submit via e-mail allows you to send a file and receive an e-mail response of the results. All you need to do is send an e-mail to scan@virustotal.com with a subject of SCAN and the file to scan as an attachment. Once the file is scanned, the results will be e-mail back to you. Information on these options can be found at http://www.virustotal.com/metodos.html.

If you suspect your computer has been infected, to protect yourself and the data on the computer, you would probably want to remove the computer from the network or Internet until you can properly clean it. Otherwise, you are running the risk of identity theft, participating in botnets or other unwanted activities.

Author Name, email@address

2.7. Advanced deployment considerations

2.7.1. Running in a small environment

When running this process in a SOHO type network environment, you may want to take further considerations into account. The environment may get to large to manage this process and analysis on a computer to computer basis. Some of the things you may want to consider include running from a central location, storing reports in a central location and adding some other mechanism for delivering the report.

If you have a computer or server that is accessible by other computers that run this process, consider setting up a shared location to use. By creating a network share, you can have computers run this process from a centralized location. When you need to update the process, you would only need to do so in one location. Not to mention that a central location would guarantee all of your computers are running the same process.

You could also use this shared location to save the report data. Our script includes the host name and date/time stamp in the report name. By modifying the script slightly, you could add additional functionality to write reports to a central location. This would make it easier for you to review the reports, or perform other tasks such as searching all of the reports for a particular thing (such as a hash of a know piece of malware).

When running this process on several computers, using the Notepad application may not be the best way to deliver the report. You may just want to rely on someone to look at the reports on a regular basis. Or maybe you want to code in some logic to deliver reports by e-mail. You could even do both, count on someone to review reports regularly and e-mail certain reports based on some predefined content.

2.7.2. Running in a medium to large environment

As the environment increases to a medium to large size, so does the complexity of this process. As we get into environments that have many computers spread across geographical locations, we need to start considering a more distributed and automated environment. We need to start considering "site" processing, possible white-listing of known good programs, additional scripts to automate tasks further and deployment automation.

When we start dealing with computers at various geographical locations, we should consider processing boundaries by geographical site. Creating multiple centralized areas, one per site, for our script to run and process can help reduce the failures that may occur when using a remote server. By creating a centralized point at each site, you can also customize the script for the particular location.

In larger environments we may run into a broader mix of applications, each having various update processes. Incorporating this process into your change management program could allow you to utilize a white-listing approach to minimize the number of entries on a report. If you make a list of known MD5, SHA-1 or SHA-256 hashes, you can compare the hash to an entry prior to writing it to the report. If you have matching hashes you know the change is authorized by your change management program, so you don't really need to see these entries in a report.

Another thing you could do is add other scripts into the mix to improve the overall process. You could possibly write a script that would traverse the reports looking for a certain item to see if any reports have it. For instance you could search for the hash of a newly discovered piece of malware. Or maybe you want to see how many users have registry entries (HKCU) with executables (.EXE) that auto-start. You can write scripts to archive older reports to keep the file system cleaned up. There are many things that can be scripted to make our jobs easier.

One other thing that can be scripted is deployment of the script. You can use many different tools that are common in a corporate environment. Things such as group policy, login scripts and software deployment tools are some options.

There is no end to the possibilities of functionality you could add to the process. Environments vary tremendously and this process can be customized for each of those environments. So take this script as a start and add to it as you see fit.

2.7.3. Run after malware alerts

There are a few considerations you should consider no matter of the size of your environment. First we should give consideration to force a run of our script after receiving a malware detection alert. After all, we cannot say for sure what the malware has done to our system and how our anti-virus software handled the infection attempt.

After the alert and prior to rebooting, manually execute the script and view the report. We should heavily scrutinize any entries in the report that is generated. It is possible that the attempt to infect our system could have been only partially prevented. Multiple items could have been attempted with only some of them being detected. For example, the SDBot worm (http://vil.nai.com/vil/content/v_140978.htm) attacked multiple vulnerabilities. What if some of the vulnerabilities were not widely known yet?

The important thing here is to make sure our security software did the job we thought it should do. By checking for additional entries we are protecting ourselves from other malware that may have been hidden in our auto-run areas to infect our systems on reboot. If you find something suspicious, you can investigate it as we did earlier.

At this point, you will have to make a decision based on your findings. You can accept the entries as legitimate, delete the entries or report findings to your anti-virus vendor. If unsure, you also have the option to seek expert advice from a local computer company.

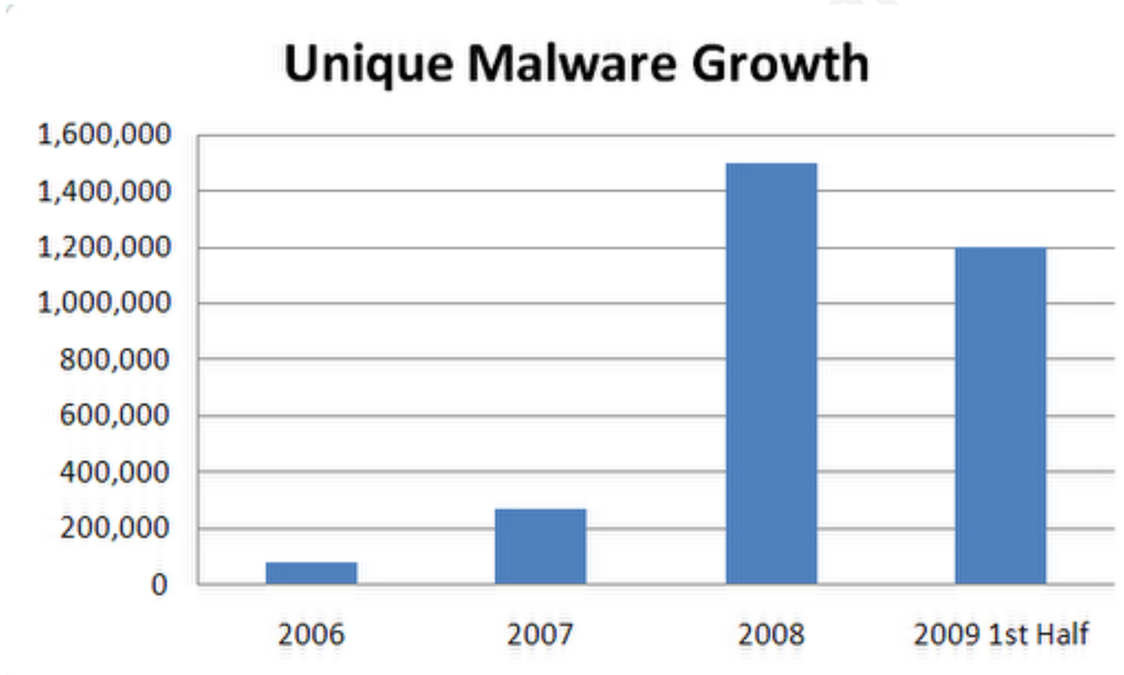
2.7.4. Updating AUTORUNS program

As with any software, Autoruns is updated on a regular basis to improve functionality or fix bugs. You should regularly monitor the Sysinternals page on Microsoft's website for updates to Autoruns.

The updates are not complex to perform. They consist of a file download and extraction to the install location. You can utilize the same instructions in the Appendix that we covered for the initial download and setup.

3. Conclusion

For many years, malware has been growing more and more into a business for cyber criminals. There are numerous pieces of malware appearing daily to be used for financial gain by malware authors. According to a Blog article by David Marcus posted on McAfee's AvertLabs website, malware growth has grown exponentially. While there is no silver bullet, or one product/process, to defend ourselves from malware, we can deploy a layered defense system to protect ourselves.



(Marcus, 2009)

In this paper, we discussed the basics of Windows auto-run areas and how malware uses these areas to survive. By hiding deep in these areas and automatically running at system startup, malware can do its bidding without our knowledge. Let's add to our defense-in-depth strategy and put another layer of malware detection in place.

By utilizing the freely available autorunsc.exe and diff.exe programs, we can build a process to monitor the Windows auto-run areas for changes. Using the script we discussed to automate this process makes it easy to find any changes (good or bad) to the auto-run areas on a Windows system. In return, we have another method to detect malware activity, even the activity of new and unknown malware.

References

Skoudis, E. (2004). Malware fighting malicious code. Upper Saddle River, NJ: Prentice Hall.

Keizer, G. (2005, April 04). *Hackers write spyware for cash, not fame*. Retrieved from http://www.informationweek.com/news/security/vulnerabilities/showArticle.jhtml?articleID=160403715

Dai, S, & Kuo, S. (2008). Mapmon: a host-based malware detection tool. *13th IEEE International Symposium on Pacific Rim Dependable Computing*, 350-356.

Debrosse, J. (2009, September 16). *Navigating the new cybercrime threatscape, part 2*. Retrieved from http://www.technewsworld.com/story/68128.html?wlc=1255138964&wlc=1258905198&wlc=1263757986

Kaspersky Lab. (n.d.). *Three criteria for malware existence*. Retrieved from http://www.viruslist.com/en/viruses/encyclopedia?chapter=153279591

Russinovich, M, & Solomon, D. (2009). *Windows internals fifth edition*. Redmond, WA: Microsoft Press.

Sanna, P. (2001, June 18). *System and startup settings*. Retrieved from http://technet.microsoft.com/en-us/library/bb742541.aspx

Szor, P. (2005). *The Art of computer virus research and defense*. Upper Saddle River, NJ: Addison Wesley.

Russinovich, M, & Cogswell, B. (2009, December 1). *Autoruns for windows v9.57*. Retrieved from http://technet.microsoft.com/en-us/sysinternals/bb963902.aspx

Free Software Foundation. (2008, October 22). *Diffutils*. Retrieved from http://www.gnu.org/software/diffutils/diffutils.html

ThreatExpert Ltd, Initials. (2009). *Welcome to threatexpert*. Retrieved from http://www.threatexpert.com

VirusTotal, . (n.d.). *Virustotal*. Retrieved from http://www.virustotal.com/

Marcus, D. (2009, July 22). *Malware is their business…and business is good!*. Retrieved from http://www.avertlabs.com/research/blog/index.php/2009/07/22/malware-is-their-businessand-business-is-good/

4. Appendix

4.1. File System Configuration

This section describes the file system structure for the script we discuss. Create the following directories on your system. The description is for information purposes and is provided to explain the purpose of the folders.

Folder	Description
C:\ARCheck\	Base folder for process. Holds the script and other programs/libraries.
C:\ARCheck\Process	Folder to retain snapshot history and working files.
C:\ARCheck\Reports	Folder to store the reports that get generated.

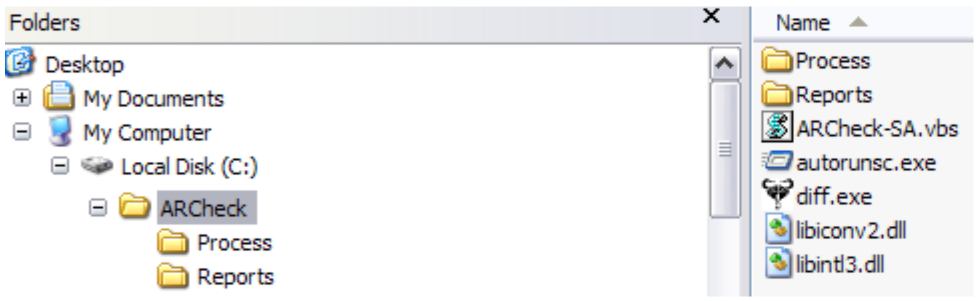
You can modify this folder structure to fit your needs and your environment. There are three variables defined for these folders toward the start of the script. Listed below are the three variables and associated folder:

strPath variable – points to the base folder C:\ARCheck

strOutputPath variable – points to the working folder C:\ARCheck\Process

strReportPath variable – points to the report folder C:\ARCheck\Reports

If you change the folder structure, you will need to modify the script variables to match. The script included in this paper will verify the subfolders under C:\ARCheck exist. If they do not, the script will attempt to create them. As long as the strPath variable points to a valid location with appropriate permissions, all else should be ok. After we complete our directory structure and the remaining steps in this appendix, we will have a folder structure that looks like the following:



We are now ready to download/create and configure the files we need for our process. These are the files as seen in the right pane of the above window.

4.2. Autoruns Download and Setup

This section describes the process to download and setup the Autoruns Utility to support our script. Once the folder structure is in place, perform the following steps to setup the Autoruns Utility our script requires.

1) Open the following Microsoft web page for the Autoruns Utility. http://technet.microsoft.com/en-us/sysinternals/bb963902.aspx

2) Click the green arrow to start the download for Autoruns.

3) When the File Download window prompts to open or save the file, click Save.

4) Save the file to a location of your choosing.

5) Once the file is saved, extract the contents.

6) Copy the "autorunsc.exe" file to the base folder you created in section 4.1, in our example it is C:\ARCheck. The other files are not needed for our process, but you may want to keep them around to use for manual investigations.

We now have the Autoruns Utility configured as we need it to be for our script. We can now move on to download the next utility we need, Diff Utils.

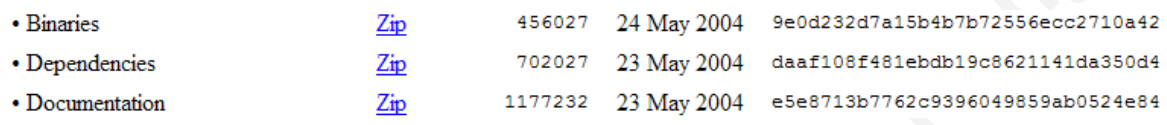
4.3. DiffUtils for Windows Download and Setup

This section describes the process to download and setup the DiffUtils for Windows package along with the supporting dependencies and documentation. We will need to download and extract two files for our diff.exe command to function correctly.

Author Name, email@address

We will also download and extract a third file so we have the DiffUtils documentation on hand.

1. Open the following SourceForge web page for DiffUtils for Windows. http://gnuwin32.sourceforge.net/packages/diffutils.htm
2. In the Download section, download the following three files and save them as you did the Autoruns download.



• Binaries	Zip	456027	24 May 2004	9e0d232d7a15b4b7b72556ecc2710a42
• Dependencies	Zip	702027	23 May 2004	daaf108f481ebdb19c8621141da350d4
• Documentation	Zip	1177232	23 May 2004	e5e8713b7762c9396049859ab0524e84

3. Extract the diffutils-2.8.7-1-bin.zip file and copy .\bin\diff.exe file to the base folder you created (C:\ARCheck).
4. Extract the diffutils-2.8.7-1-dep.zip file and copy .\bin*.dll files (libiconv2.dll and libintl3.dll) to the base folder you created (C:\ARCheck).
5. Extract the diffutils-2.8.7-1-doc.zip file to a location to save the documentation pdf for future reference.

We now have all the support files download, extracted and put in the correct location for our script. At this point, all we need to do is build the script file to execute.

4.4. ARCheck-SA.vbs script

Here is the commented script to copy and store in a file in the base folder (C:\ARCheck). For our example we named the file "ARCheck-SA.vbs". Copy the following script into a Notepad document and save the file in the base folder as "ARCheck-SA.vbs". At this point you will be able to run the script according to the schedule you have decided on.

```
'************ Start VBScript **********************************
'Turn on error checking
On Error Resume Next

'Set up variables and constants
'Constant to use in file opening for read only
```

Author Name, email@address

```
Const intForReading = 1
'Constant to use for file overwrite
Const booOverwrite = True
'Constant used to tell Shell Object to wait for command completion before
continuing
Const booWait = True
'Constant to tell Shell Object to not wait for command completion before continuing
Const booNoWait = False

'Versioning number to track program version and display in report
strVersion = "1.0.0"
'Path to program root folder
strPath = "C:\ARCheck\"
'Current system date and time for this report
strStart = Now
'Pull current system date for report
strDate = DatePart("m",strStart) & DatePart("d",strStart) & _
        DatePart("yyyy",strStart)
'Pull current time for report
strTime = FormatDateTime(strStart,4)
'Combine date and time into a string as mdyyyyhhmm
strDateTime = strDate & Mid(strTime,1,2) & Mid(strTime,4,2)

'Create object for system information
Set objSI = CreateObject("WinNTSystemInfo")
'Get host name from the system information object
strHost = UCase(objSI.ComputerName)
'Get user name from the system information object
strUser = UCase(objSI.UserName)
'Free memory in use for system information object
Set objSI = Nothing
'Check to see if user is blank.  If so assign it an unknown value
If strUser = "" Then strUser = "UNKNOWN"
```

Author Name, email@address

```
'Output path for snapshot working folder
strOutputPath = strPath & "Process\"
'Set name for current snapshot file
strCurrentFile =  strOutputPath & strHost & "-snap.curr"
'Set name for previous snapshot file
strPreviousFile = strOutputPath & strHost & "-snap.last"
'Set name for difference file that will be created to generate report
strDiffFile = strOutputPath & strHost & "-snap.diff"
'Set path of location to write reports
strReportPath = strPath & "Reports\"
'Set filename for report
strReportFile = strReportPath & strHost & "-" &  strDateTime & ".report"

'Create an object to work with the file system
Set objFSO = CreateObject("Scripting.FileSystemObject")

'Verify file system is in order
'Check to see if working folder exists
If Not objFSO.FolderExists(strOutputPath) Then
	'If not then create it
	objFSO.CreateFolder(strOutputPath)
	'If it cannot be created call Error Handling procedure
	If Err <> 0 Then Call ErrHandler(Err,"Output Path",strOutputPath)
End If

'Check to see if report folder exists
If Not objFSO.FolderExists(strReportPath) Then
	'If not then create it
	objFSO.CreateFolder(strReportPath)
	'If it cannot be created call Error Handling procedure
	If Err <> 0 Then Call ErrHandler(Err,"Report Path",strReportPath)
```

```
End If

'File handling for snapshot rotation
'Delete previous file
objFSO.DeleteFile(strPreviousFile)
'Check to see if there is a current snapshot file
If Not objFSO.FileExists(strCurrentFile) Then
        'If it doesn't exist then create an empty previous file, so we have
        'something (in this case nothing) in which to compare the current snapshot.
        objFSO.CreateTextFile(strPreviousFile)
        'Set indicator so we know this is the first run and there is no
        'historical snapshot
        booNoHistory = True
Else
        'If there is a current snapshot file, move it into the previous
        'snapshot file for historical comparison
        objFSO.MoveFile strCurrentFile,strPreviousFile
        'Set indicator so we know this is a subsequent run
        booNoHistory = False
End IF

'Make snapshot
'Set our command string for the command to make our current snapshot file
'Note the "/accepteula" parameter that is needed to avoid license agreement
'acceptance request that is seen the first time Autorunsc is run on a system
strCommand ="%ComSpec% /c " & strPath & "autorunsc.exe /accepteula -avc >" & _
                strCurrentFile
'Call procedure to run command with a Window style of 0 and wait for return
Call RunCommand(strCommand,0,booWait)

'Compare snapshots
'Set our command string for the command we want to run to compare our snapshots
```

Author Name, email@address

```
'and output the results into our difference file
strCommand = "%ComSpec% /c " & strPath & "diff.exe " & strPreviousFile & " " & _
                    strCurrentFile & " >" & strDiffFile
'Call procedure to run command with a Window style of 0 and wait for return
Call RunCommand(strCommand,0,booWait)

'Build Report
'Open report file for writing report, overwrite file if it exists
Set objReportFile = objFSO.CreateTextFile(strReportFile, booOverwrite)
'Create header for report
objReportFile.WriteLine("Report for Auto-Run areas on: " & strHost & " by " & _
          strUser)
objReportFile.WriteLine("========================================" & _
                    "=================================================")
objReportFile.WriteLine("ARAnalyzer Version: " & strVersion)
If booNoHistory Then objReportFile.WriteLine("*** No previous history " & _
                                                  "detected: reporting everything.")
objReportFile.WriteLine("Date/Time: " & strStart)
objReportFile.WriteLine("Report File: " & strReportFile)
objReportFile.WriteLine("========================================" & _
                    "=================================================")
objReportFile.WriteLine()

'Set indicator to detect if we have changes to report
booChanges = False
'Set variable to hold additions to be reported
strAdditions = ""
'Set variable to hold deletions to be reported
strDeletions = ""
'Open our difference file for processing information to include in report
Set objDiffFile = objFSO.OpenTextFile(strDiffFile, IntForReading)
'Enter loop to read through difference file line by line
```

```
Do Until objDiffFile.AtEndOfStream
                'Read an entry from the difference file
                strOutput = objDiffFile.ReadLine()
                'Check to see if the entry is an addition ">" or deletion "<"
                Select Case Left(strOutput,1)
                        Case ">" 'Addition
                                'Add addition entry to additions area for report
                                strAdditions = strAdditions & strOutput & vbCrLf
                                'Change indicator to indicate changes were detected
                                booChanges = True
                        Case "<" 'Deletion
                                'Add deletion entry to deletions area for report
                                strDeletions = strDeletions & strOutput & vbCrLf
                                'Change indicator to indicate changes were detected
                                booChanges = True
                        Case Else
                                'Do nothing - entry line was either header or whitespace
                End Select
Loop

'Check to see if changes were detected
If Not booChanges Then
        'If no changes, indicate that it report
        objReportFile.WriteLine("No changes were detected since last run.")
Else
        'If changes were detected
        'Write additions to report first
        objReportFile.WriteLine("Additions:")
        objReportFile.WriteLine("----------------------------------------------------" & _
                                        "--------------------------------")
        'Check to see if additions were detected
        If strAdditions = "" Then
```

Author Name, email@address

```
                'If no additions were detected tell that in report
                objReportFile.WriteLine("No additions to report." & vbCrLf)
        Else
                'If additions were detected write them to the report
                objReportFile.WriteLine(strAdditions)
        End If
        'Write deletions to report
        objReportFile.WriteLine("=================================" & _

        "===========================================================")
        objReportFile.WriteLine("Deletions:")
        objReportFile.WriteLine("----------------------------------------------------" & _
                                                "--------------------------------")
        'Check to see if deletions were detected
        If strDeletions = "" Then
                'If no deletions were detected tell that in report
                objReportFile.WriteLine("No deletions to report." & vbCrLf)
        Else
                'If deletions were detected write them to the report
                objReportFile.WriteLine(strDeletions)
        End If
End If

'Close file objects
objReportFile.Close
objDiffFile.Close

'Display report
'Here we will use notepad to display report if changes were detected
'Set up our command to open the report using notepad
strCommand = "notepad.exe " & strReportFile
'Check to see if changes were detected, if so then open report with notepad.
```

```
If booChanges Then Call RunCommand(strCommand,7,booNoWait)

'Cleanup filesystem and memory
'Delete difference file
objFSO.DeleteFile(strDiffFile)
'Free memory used by script
Set objReportFile = Nothing
Set objDiffFile = Nothing

'Exit Script by calling exit procedure
ExitScript

'Procedures and Functions
Sub ErrHandler(iErrNum,strErrDesc,StrErrFile)
        'This procedure will output an error message to the screen and the call the
        'script to exit our script.
        'Inputs
        '   iErrNum - error number returned
        '   strErrDesc - error description message set be script not system
        '   strErrFile - file path that error is concerning
        'Write message to screen
        WScript.Echo "Error with " & strErrDesc & vbCrLf & _
                "File: " & strErrFile & vbCrLf & "Error number: " & iErrNum
        'Add additional message based on error code return
        Select Case iErrNum
                Case 70 'Error number 70 is generated due to bad permissions
                        WScript.Echo "Access Denied!"
        End Select
        'Call procedure to exit script
        ExitScript
End Sub
```

```
Sub ExitScript()
	'This procedure returns memory and exits our script.
	Set objFSO = Nothing
	WScript.Quit
End Sub

Sub RunCommand(strCmd,intWinStyle,booWaitDone)
	'This procedure runs our commands
	'Inputs
	'   strCmd - our command line string to run
	'   intWinStyle - window style to run our comman in - 0 = hidden
	'   booWaitDone - should our command complete before we continue?
	'Create shell object to run our command
	Set objShell = createobject("wscript.shell")
	'Execute our command
	objShell.Run strCmd,intWinStyle,booWaitDone
	'Give the memory back
	Set objShell = Nothing
End Sub
'************ End VBScript ********************************
```

4.5. ARCheck-SA.vbs variables

The following list explains the purpose of some of the variables and constants you find in the script in section 4.4. These are some values you can customize for your environment. Following these are constants and variables that should not be changed.

- strVersion – Helps you track versions of the script. The version is used in the report so we know which version of the script created the report. You can use this Constant to perform version tracking as you customize the script over time.
- strPath – This is the base folder path where all the script and command files are located. See Appendix, Section 4.1, for information about our folder structure. This can remain pointed to the local hard drive, or can be changed

to a mapped drive or UNC path. You can customize this to fit the needs of your environment and structure desires. Holds a value of "C:\ARCheck\".

- StrOutputPath – gets assigned the output path for the Process folder described in the Appendix, section 4.1. Built off of the strPath variable and holds the value: "{strPath}Process\".
- strCurrentFile - gets assigned the output path and filename for our current snapshot file. Built off of the strOutputPath and strHost variables. Holds a value of "{strOutputPath}{strHost}-snapp.curr".
- strPreviousFile – gets assigned the output path and filename for our previous snapshot file. Built off of the strOutputPath and strHost variables. Holds a variable of "{strOutputPath}{strHost}-snapp.last".
- strDiffFile – gets assigned the output path and filename for our difference file. Built off of the strOutputPath variable. Built off of the strOutputPath and strHost variables. Holds a variable of "{strOutputPath}{strHost}-snapp.diff".
- strReportPath – gets assigned the output path for the Reports folder described in the Appendix, section 4.1. Built off of the strPath variable and holds the value: "{strPath}Reports\".
- strReportFile – gets assigned the output path and filename for our report file. Built off of the strReportPath, strHost and strDateTime variables. Holds the value of "{strReportPath}{strHost}-{strDateTime}.report".
- strCommand – defined in "Make snapshot", "Compare snapshots" and "Display reports" sections. Used to build the autorunsc.exe and diff.exe commands as we discussed earlier. It is also used to build the command to open our report with notepad.exe. If you wish to modify these commands, these are the areas where you do it.

The following variables and constants are used in the script. However these should not be changed.

- intForReading – This variable is used to set the IOMode when opening a file object. This is used when opening an object to the difference file. The IOMode value of 1 is for read only, hence the variable name.
- booOverwrite – used when performing operations on our output files. When we create our report file on the file system, we tell the object to overwrite the file if it exists.
- Err – built in variable that is used by Windows Scripting Host as a return variable for command success of failure codes.
- booWait – boolean variable used to tell the script to wait for our command to finish running before continuing with the script. Holds a value of True.
- booNoWait - boolean variable used to tell the script NOT to wait for our command to finish running before continuing with the script. Holds a value of False.
- strStart – gets assigned the current system time.
- strDate – gets assigned a MMDDYY value from the strStart variable.
- strTime – gets assigned a HHMM value from the strStart variable.
- strDateTime – concatenation of the strDate and strTime variables to use when formatting our report name.
- strHost – variable that holds the host named pulled from the System Information object.
- strUser – variable that holds the username pulled from the System Information object. If no username can be determined from the System Information object, this variable is assigned the value "UNKNOWN".
- booNoHistory – defined in "File and error handling" section. When determining if this is the first time the program is running on the system or if there was a missing previous snapshot. This variable is used in report building to inform you the script thinks the report is a first time run.

- booChanges – defined in “Build report” section. Determines if the report needs to be displayed. If there is an entry added to the report, this variable is set to TRUE to run the notepad.exe command to display our report.
- strAdditions - defined in “Build report” section. When parsing the difference file, addition entries are added to this variable. If this variable is not empty when building the report output, it will add the addition entries to our report. If this variable is empty, it will add an entry to the report telling us “No additions to report”.
- strDeletions - defined in “Build report” section. When parsing the difference file, deletion entries are added to this variable. If this variable is not empty when building the report output, it will add the deletion entries to our report. If this variable is empty, it will add an entry to the report telling us “No deletions to report”.
- strOutput - defined in “Build report” section. This variable holds each individual entry from our difference file as it is read and processed line-by-line.
- iErrNum – used in ErrHandler Procedure. The return code is from the built-in Err variable. Used when displaying error output.
- strErrDesc - used in ErrHandler Procedure. An error description passed when the procedure is called. Used when displaying error output.
- strErrorFile - used in ErrHandler Procedure. A string that is passed to the procedure and reports which file in which we are having errors. Used when displaying error output.
- strCmd – used in RunCommand Procedure. It is set to a command string that gets passed to our procedure. The RunCommand procedure the uses this string to execute our various commands when it is called.
- intWinStyle - used in RunCommand Procedure. It is set to the Windows style we pass to our procedure to tell what kind of Window to run the command in.

We use two different styles; a 0 (zero) tells it to use a hidden windows and a 7 tells it to open a regular window in the background.

- booWaitDone - used in RunCommand Procedure. Boolean variable that tells the command weather to wait or not to wait for completion of the command before continuing with our script.

Objects defined in script

- objSI – System Information object. Opens an interface to system properties for us to gather the hostname (strHost) and username (strUser).
- objFSO – File System object. Opens an interface to the file system so we can interact with the files.
- objReportFile – File object. Opens an interface with our report file so we can output out findings.
- objDiffFile – File object. Opens an interface with our difference file so we can parse the differences in our file and create a report.
- objShell – Shell object. Opens a shell interface to allow us to run our commands on the system.