Windows Forensic Analysis

Dedicated to incident response and computer forensic analysis topics, with respect to Windows 2000, XP, 2003, and Vista operating systems

WFA: Windows File Protection and Malware Detection

As of Windows 2000, the Windows operating systems have had a mechanism in place called "Windows File Protection[1]", or WFP. While not specifically billed as a security feature, WFP protects specific files by running in the background and "waking up" whenever a file system change event is generated, and checking its repository of "known-good" versions of files. By default on most systems, this repository will be found in the %SystemRoot%\system32\dllcache directory (may also be maintained on a network share or CD). If the file associated with the file system change event is protected by WFP (i.e., the file name is listed in sfcfiles.dll and there is a file by the same name in the repository) and has a different MD5 hash from the version in the repository, WFP will automatically replace the changed version with a copy of the "known-good" version of the file, and generate a System Event Log record (event ID 64001, source "Windows File Protection").

Windows systems also include the sfc.exe application, which allows an administrator to scan all protected files to verify their versions (on-demand scans, etc.).

On Vista systems, sfc.exe and the Windows Modules Installer service write to a log file name cbs.log[2] located in the %WinDir%\logs\cbs directory. Entries in this log file that were generated by sfc.exe include an "[SR]" tag, and as such, can be easily parsed and reviewed as part of a forensic examination. KB article 928228 includes a log except indicating that sfc.exe found an issue with a protected file.

See the Resources section for links to descriptions of the system file checker for Windows 2000, XP, and 2003.

A while ago, the Teddy Bear Hoax[3] was floating around. Users would receive an email stating that the file on their system named jdbgmgr.exe, whose icon was a teddy bear (see graphic), was really malware and needed to be deleted immediately. On Windows 2000 systems, this was a file protected by WFP, and if the user deleted the file, it would be immediately replaced. I ran into an example of this hoax while working in a full-time employment position as a security engineer…about a dozen employees received the hoax email, and all but one informed me of the email prior to doing anything. The remaining employee followed the instructions in the email, but Windows 2000 "woke up" and replaced the protected file automatically.

The list of protected files, including their paths, is maintained in sfcfiles.dll. Opening this file in PEView will allow you to see the list of protected files in the .data section of the PE file.

Malware

Many times malware authors will attempt to hide their programs from detection by giving the program a name that is very similar to the name of an important system file ("svclhost.exe" vs. "svchost.exe[4]"), or

[1] http://support.microsoft.com/kb/222193
[2] http://support.microsoft.com/kb/928228
[3] http://www.hoax-slayer.com/teddy-bear-virus-hoax.html
[4] http://support.microsoft.com/kb/314056

by giving the program the same name as an important system file, but writing it to a separate location within the file system (i.e., \Windows\temp vs. \Windows\system32).

However, some malware authors have been found using a method of disabling WFP, and having their programs replace a program or file protected by WFP. The two major advantages to the malware authors are that (a) the method uses "undocumented" Windows API functions, and (b) anti-virus vendors that analyze captured bits of malware are making to attempts to highlight this mechanism.

The "undocumented" method[5] used to temporary disable WFP is documented at BitSum.com (along with several other methods of "hacking" WFP). The web page describes an undocumented API function, *SfcFileException*, exported by sfc_os.dll at ordinal number 5.

Another method of "hacking" WFP is to modify the file entries in the sfcfiles.dll file. ElTorquiro described[6] one method, which was to replace the first bytes of the path to protected file with "\", followed by null termination (strings within the DLL are null terminated). Jeremy (at the BitSum site) described a similar method, which was to replace the first character of the path for the file to be replaced with a zero value.

According to the information at the BitSum site, by calling this undocumented API function, a malware author can disable WFP for one minute and put his own file on the system, overwriting a file protected by WFP. In fact, Win32/Caowy.G[7] does just that. While the AV vendor write-up does not mention the use of the undocumented API function, analysis of the unobfuscated version of the malware illustrates the use of that function. By using *SfcFileException*, the malware is able to replace spoolsv.exe, which on Windows XP systems is protected by WFP.

On 5 July 2007, a McAfee AVERT Labs Blog post (*http://www.avertlabs.com/research/blog/index.php/2007/07/05/wfp-hack-redefined/*) described W32/Crimea, which accessed the *SfcFileException* function. The undocumented API function is even mentioned in this[8] Polk County, IA, Supplemental Expert Report from 19 Dec 2006.

Analysis

Most analysts and security personnel are fully aware that AV applications are not silver bullet solutions to the issue of malware. Many examinations have revealed malware samples that when scanned with several AV products, or posted to the *virustotal.com* web site, are not identified at all.

Even so, there are processes that an analyst can use to "narrow down the field" a bit. For example, an analyst can use an application to parse through the dllcache directory (after mounting an image as a read-only file system), collecting all of the file names and sizes, and computing MD5 hashes for each file (and in addition, possibly also collecting file version information from the resource sections of Windows PE files, if available). Then, that application would then parse through the entire file system (skipping the dllcache directory, of course), locating files of the same names as those found in the dllcache directory, and comparing sizes and hashes. This process would allow the analyst to quickly (with a degree of automation and

[5] http://www.bitsum.com/aboutwfp.asp

[6] http://www.msfn.org/board/WFP-app-for-removing-individual-files-t98306.html&mode=linearplus

[7] http://www.ca.com/us/securityadvisor/virusinfo/virus.aspx?id=73218

[8] http://www.sonic.net/~undoc/comes_v_microsoft/Supp_Rpt_Andrew_Schulman.pdf

accuracy) detect any protected files that had been replaced within the file system.

> Mounting the image as a read-only file system via a tool such as SmartMount,[9] or accessing the system remotely via F-Response[10] (also read-only) protects the files accessed from having their last access times modified by this analysis process. That way, hash values can be computed and compared, and if the values do not match, the file metadata (i.e., last access time) is retained for later analysis.

An application such as WFPChecker v.1.01 (GUI excerpt illustrated in figure 1) allows an examiner to automate this analysis by selecting the location of the %SystemRoot% (most often C:\Windows\system32 directory) and the resulting .csv file. WFPChecker will then locate the dllcache directory within the %SystemRoot%, parse the file names and sizes for all files in that directory and compute their MD5 hashes.

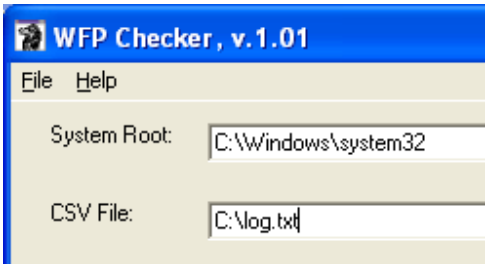

Figure 1: WFPChecker v1.01 GUI excerpt

The application will then parse through the file system starting at the root of the drive (in the example illustrated in figure 1, C:\) looking for *all* files having the same name, computing their MD5 hashes, and comparing the hashes and file sizes. In order to optimize the search, WFPChecker starts in the %SystemRoot% directory first, as a number of the protected files are located in that directory.

[9] http://www.asrdata.com/SmartMount/
[10] http://www.f-response.com

The application then writes its results in a comma-separated value (.csv) format so that the examiner can open the file in Excel for easy analysis. The headers written to the resulting .csv file are file name, full path to the file, and the results of the comparison. If a match is found, "Match" is written; if there is a mismatch in size and/or hash values, "MisMatch", followed by the size and hash values, is written. The examiner can open the .csv file in Excel and sort on any of the columns for easier analysis and viewing.

When WFPChecker is run, information about the status of the scan is displayed in the GUI. WFPChecker will also save a log file of its activity, in the same location as the .csv file, with the same name, but with a different file extension. This allows the examiner mount an image file (for example, D:\case\image.dd) using SmartMount, as a read-only file system (i.e., "F:\"), and save the results to D:\case\ for easy case management.

> There is a second version of WFPChecker available called WFPChecker v.101F, which performs all of the same functions as WFPChecker, with the added capability of collecting file version information from the resource section of executable file images, if this information is available. This provides the examiner with some additional information that can be used to determine the validity and source of various files.

However, this process might also present the examiner with false positives, as older archived versions of files (for example, afd.sys) may still be resident within the file system, albeit not "protected" by WFP. The examiner should ensure that enough information is collected during the scanning process to provide for such results, and that the analysis of the data addresses these results accordingly. Also, this analysis methodology should be considered supplementary to other forms of malware

detection analysis, such as AV scanning and hash value comparison.

The examiner should remember that the file sfcfiles.dll contains a list of protected files, which can be enumerated using BinText (as illustrated in figure 2) or strings.exe[11].

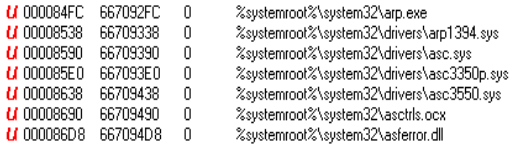

Figure 2: Sample of BinText output from sfcfiles.dll

The examiner could use this list of files instead of the list of files derived by reading the contents of the dllcache directory, as the list from sfcfiles.dll includes more complete path information. However, both techniques may result in false positives, and automated methods of analysis should take this into account. As a general rule, automation applied to analysis should strive for data reduction.

Malware Analysis

If the analyst is able to obtain a memory dump from the system, tools such as Volatility[12] may allow her to extract the actual Windows portable executable (PE) file for the malware from the memory dump, and parse the Import Table to determine if the API function at ordinal 5 from sfc.dll is accessed by the malware. This same technique can be used when analyzing suspicious files that do not appear to be native to the system, are not detected by AV scanning applications, etc.

Note: On Vista, the Windows Resource Protection mechanism includes the *SfcIsKeyProtected* API function, which can be used to determine if a Registry key is protected.

[11] http://technet.microsoft.com/en-us/sysinternals/bb897439.aspx
[12] https://www.volatilesystems.com/

In addition, examiners should include searches of the System Event Log for event ID 64001, with "Windows File Protection" as the source, as such records *may* indicate attempts to replace critical system files.

Summary

Windows File Protection was not intended to be a security mechanism, but it does offer useful functionality. In order to replace files protected by WFP, certain mechanisms need to be used (calling undocumented API functions, DLL injection, etc.) and these mechanisms will have indicators or leave artifacts when used. A level of complexity has been removed from implementing undocumented functions such as *SfcFileException* as usable code has become available on the Internet. However, live response and memory analysis techniques can be used to locate artifacts of the use of these mechanisms, providing for more automated and comprehensive analysis of systems, particularly in light of shortcomings faced with AV scanning applications.

Analysis Checklist

A checklist for analysis might appear as follows:

- Parse the System Event Log for Event ID 64001 records, or use a tool such as evtrpt.exe to determine if there are any such records in the log
- Mount the acquired image read-only and use a tool such as WFPChecker to scan for file disparities between those in the dllcache directory and those within the file system
- Parse a memory dump for unusual or suspicious executable file images, and examine the import table for indications of access to the *SfcFileException* API function
- Parse a memory dump and/or unallocated space for indications of event records with event ID 64001

Resources

- Registry Settings for Windows File Protection[13]
- Description of the Windows 2000 System File Checker[14]
- Description of the Windows XP and Windows Server 2003 System File Checker[15]

Harlan is a forensic analyst located in the metro DC area, and is the author of three books related to live response and forensic analysis of Windows systems. He also presents at conferences and provides training on those same topics, as well. He can be reached at keydet89@yahoo.com.

[13] http://support.microsoft.com/kb/222473/
[14] http://support.microsoft.com/kb/222471/
[15] http://support.microsoft.com/kb/222471/