Snapshot Filtering Based On Resource-Usage Profiles

Frank Adelstein[1] and Carla Marceau
ATC-NY
{fadelstein,carla}@atc-nycorp.com

Abstract
Live forensic tools provide investigators with new sources of information. Unfortunately, the amount of data gathered by such tools can be overwhelming, with a low signal-to-noise ratio. The authors use an innovative method of monitoring the resource use of running processes to build a profile of the application's normal resource use, which they then exploit to filter out extraneous, forensically uninteresting data from a list of open file handles and dynamically loaded libraries attached to a process. Preliminary results show a dramatic reduction in the number of file and registry handles and DLLs, greatly reducing the forensic haystack, allowing the investigator to more easily spot the needles.
Keywords: live forensics, filtering, DLLs, open handles, normal resource usage, profiling

1. Introduction

As the field of live forensics matures, the investigator is gaining access to an abundance of information about the state of the system under investigation. He can observe not only the currently executing processes and their files, but *all* resources associated with each process, including registry keys, network connections, and less interesting, esoteric items such as shared memory segments and mutual exclusion handles.

The problem is that the investigator obtains too much information. Consider the files the process has open, which typically include many files needed by standard libraries (DLLs) working on behalf of the process. Processes using the network typically invoke DNS libraries and associated look-up functions. The configuration files associated with these functions are *not* forensically significant when used in the standard way. Such files provide little value to a forensic investigation but clutter the data space, rather than help the investigator zero in on useful evidence.

This paper focuses on the problem of data overload. Simply put, there is too much evidence available in digital forensic cases, and following dead-ends slows the analysis process. A typical Windows process can have hundreds of loaded DLLs and handles. For example, in one test Windows Explorer had 117 DLLs loaded, none of which were of particular interest.

The current state of the art in forensics is filtering based on fixed patterns, which is at best tedious and error-prone. There are standard search patterns to use (e.g., email addresses, credit card numbers, or names relevant to a case), but the overall method tends to be ad-hoc. Generally, one can search for either information to keep or information to eliminate. Such searches are easy to perform if one knows ahead of time what to keep or eliminate. If one does *not* have a list of relevant search terms, then something must create a list to allow a program to automatically eliminate useless information.

What is needed is a way to selectively filter out the extraneous pieces of information, while showing the useful information. However, the choice of what to filter depends on the particulars of the process under examination.

ATC-NY has addressed this problem with an advance in digital evidence analytics that helps investigators focus on potentially useful resources, such as files and registry keys peculiar to a

[1] Correspondence author

process. To do so, we employ a tool, AppMon, which observes a running application (across multiple invocations, by multiple users) and builds an abstract *profile* of the normal resource use of that application. As a proof of concept, we used the profile with our live forensic tool, OnLine Digital Forensic Suite™ (OnLineDFS), to filter out resources that the application often uses. Our hypothesis is that the remaining information will be more interesting and much easier to peruse.

The rest of the paper is organized as follows. A description of OnLineDFS and the process data it gathers is presented in Section 2, Section 3 describes the resource profile and how it is generated. Section 4 presents the results, and Section 5 discusses the conclusions and future work.

2. Gathering process data with OnLineDFS

OnLineDFS is a Web-based tool created by ATC-NY to gather volatile data from a live system with minimal impact to the running system [1]. It was one of the earliest products to gather volatile data in a systematic way. Online DFS addresses both UNIX™ and Windows systems, but in this paper we focus on Windows computers.

OnLineDFS provides extensive information about running processes. It dispatches *transient utilities* to run on the target system to gather information. The utilities generally send results in plain-text or XML. The user interface presents the results to the user in HTML, through a web browser. The "detailed process list" page combines results gathered from several utilities.

In addition to general data about the computer, OnLineDFS gathers the following detailed information about each process: process name, process ID, user, command line, priority, start time, memory statistics, execution times (user, kernel, elapsed), open network ports, running threads, loaded DLLs, and open handles.

As noted earlier, the last two items often include hundreds of entries. For DLLs, OnlineDFS reports the version number and path for each loaded DLL. Most DLLs are loaded from either "C:\Windows\system32" or a directory related to the program such as "C:\Program Files\Microsoft Office\..." In addition, related system libraries tend to have similar version numbers (e.g., 5.01.266.2180 or 11.00.5601.0000). While an investigator can spot outliers by sorting the results by version or path, the process of doing so is tedious and error-prone.

Online DFS reports the type and path for all open file and registry key handles. The registry contains a wealth of information; but, many keys are not forensically significant. For registry keys, the path is similar to a file path, e.g., `\REGISTRY\MACHINE\SYSTEM\ControlSet001\Control\Nls\CodePage`.

3. Resource-usage profiles

From a potentially very large number of resources in use by a process, we wish to filter out resources that are unlikely to be of interest in an investigation. We begin by observing that each process is devoted to a single application. The resources used by an application fall into four groups:

Application-specific. The application needs a given resource—e.g., file or registry key and obtains it by name. An example would be the application's initialization file. On Windows systems, these files are typically found in the `%homedrive%\program files` folder.

Application/user-specific. The application maintains certain data per user—e.g., user mailboxes maintained by Outlook. These files are typically kept in the `%appdata%` directory on Windows XP systems.

Operating-system-specific. A system library routine invoked through the application needs a particular registry key and obtains it by name. An example would be the Windows initialization file or a registry key describing the local machine.

Process-specific. Some resources, such as user-specified files and temporary files, are peculiar to the process. Other executions of the same application would not use files with these names.

Based on this observation, we can now see that the forensic investigator is interested in just two categories of file. The **process-specific category** is the most interesting when the focus of the investigation is a person using the computer as a common tool. This category includes local copies of Web pages downloaded by a Web browser process and attachments to emails the target computer is either sending or receiving. **Application-user-specific** files are also of interest, especially when more is known about the application. This category includes mailboxes, contact lists, cookies, and (Web) bookmarks.

Further, since the application must be able to find all but the process-specific resources by name, what is really needed are the names by which the process obtains its application-, system-, and application/user-specific resources.

The names of many resources are easy to guess—especially files and registry keys required by an application. For example, we would expect Outlook to use files whose pathnames begin with `%homedrive%\program files\microsoft office` or `%appdata%\microsoft\outlook`. However, that criterion is not adequate for large, complex applications. In fact, Outlook also commonly uses files that a forensic investigator might not have predicted. A partial list includes:

```
%allusersprofile%\application data\microsoft\network\connections\pbk
%allusersprofile%\application data\microsoft\office\data\opa11.bak
%allusersprofile%\application data\microsoft\office\data\opa11.dat
%appdata%\microsoft\office\mso1033.acl
%appdata%\microsoft\office\msout11.pip
%appdata%\microsoft\systemcertificates\my\certificates
%homedrive%\program files\common files\microsoft shared\ink\penusa.dll
%homedrive%\program files\messenger\msmsgs.exe
```

A more systematic approach to obtaining the names of resources that an application consistently uses is to empirically collect data on the application's resource use and create a *resource-usage profile*. Such a profile should be independent of individual users, computers, or sites and should respect the application's use of environment variables. We have developed instrumentation to collect data on an application's use of resources and software that reads the resulting traces and creates empirical profiles of an application's use of files, DLLs, registry keys, and network addresses [9]. Within each resource category, the profile classifies each resource as application-specific, system-specific, or per-process. (The per-application category also includes application/user-specific resources).

The algorithm used to construct the profile is straightforward. First, for each trace, collect the set of files that are accessed and how they are used (e.g., read or written). Then correlate names and usage across traces to find names that are commonly used. "Commonly" means that the names appear in traces from a certain number of distinct IP addresses. Requiring that names be used on different computers prevents the inclusion of most user- and computer-specific resources.

Such names are assumed to be either application- or system-specific. System-specific files are located in certain well-known places in the file system. The remaining common files are deemed application-specific. Note that some names that the algorithm deems "application-specific" may not actually be known to the application. For example, profiles for Outlook typically include files used by anti-spam add-ins, because they are so commonly used with Outlook.

However, pathnames often include login names; using the "raw" pathname would cause the algorithm to miss common names. To address this problem, the instrumentation also collects the definition of standard OS environment variables during execution of the process; for each file, it

records all the ways of writing its name using the environment variables. (Note: A given application may maintain some file pathnames in registry keys. Our algorithm does not currently find such keys or extract the pathnames.) For example, the pathname `c:\documents and settings\fadelstein\application data\microsoft\office\msoutll.pip` might also be written `%homedrive%:\documents and settings\%username%\application data\microsoft\office\msoutll.pip` or `%appdata%\microsoft\office\msout11.pip`. Names are correlated across traces to find common variants in distinct traces. The profile contains the most general common variant name—in this case, the name beginning `%appdata%`.

This algorithm results, then, in a profile of the application-specific and OS-specific resources (files, DLLs, registry keys, and network addresses) that are commonly used in conjunction with a given application.

3.1. How OnLineDFS uses the AppMon profile

OnLineDFS uses the AppMon profile in a straightforward way. Once the investigator has selected a profile to apply to a process, OnLineDFS reads the profile, and then iterates over all resources associated with the target process. Resources found in the profile are suppressed; resources not in the profile are displayed to the investigator. As a final step, all of the "esoteric" handles, including shared memory segments, semaphores, and mutual exclusion ("mutants") handles, are also suppressed. OnlineDFS uses the profiles to perform data reduction only, it does not "flag" any of the remaining data as abnormal. Our goal is to reduce the amount of time an investigator must spend examining the details associated with the process data.

3.2. Effects of Imperfect Profiles

The technique of constructing resource-usage profiles based on empirical data collected from a set of computers has a long history in the field of intrusion detection (a partial list of papers in this field includes [2, 3, 4, 5, 6, 7, 8, 10, 11]). Variations from the empirical profile are symptoms of anomalous code, which may indicate an attack.

Anomaly detection based on empirical profiles is subject to both false positives and false negatives. A *false positive* occurs when the program exhibits behavior that does not occur in the profile; for example, it may be caused by an uncommon but perfectly legitimate exception. False positives occur in resource usage profiles, but their effect is innocuous: a false positive simply adds one to the length of a list of files or registry keys that the investigator must consider. A *false negative* occurs when the program exhibits behavior that is in the profile but is nonetheless anomalous. Mimicry attacks [12] succeed by imitating normal behavior. In the context of forensic filtering, a false negative occurs when a file or registry key normally used by the process is filtered out, although it should not be. For example, the program may often write to a certain registry key. In the target process, the user explicitly causes a write to that registry key, but the registry key is hidden because of the filtering. Note that the only resources that can be hidden in this way are those normally used by the program. Therefore, this can only be a cause of concern in cases where the malicious action of the user is focused on such a resource. For example, a user might move a file containing illicit material to the location of an application file that he knows the application will not use on his behalf. However, we do not consider this a major concern. Current digital forensic tools can identify file type/file name mismatches, such as a jpeg image file with a .dll suffix. While this process is slow for large disk images, it is what is currently done. The method presented in this paper is intended to reduce the time investigators must spend examining certain resource lists; it does not replace existing techniques and methods.

The investigator is first presented with the unfiltered results. He then applies the filter to reduce the amount of information to review. If no promising leads are found, the investigator

must revert back to the unfiltered results and examine the remaining data. There should not be a significant impact from a legal point of view. The filters are analogous to performing a regular expression search for email addresses or URLs on a disk image. In both cases, the defense would have access to the raw data and could investigate it as they wish.

Many uses of anomaly detection are based on sequences of kernel calls [2, 3, 4, 6, 7, 8, 10, 11]. A concern in such systems is the sensitivity of the profile to small changes in the code. We expect the resource usage profiles described in this paper to be relatively insensitive to such changes. Patches and minor revisions to a program do not typically change its use of registry keys and files. In fact, we have found that most revisions to a program have little or no effect. We commonly create new resource usage profiles only for major revisions to a program (e.g., from Firefox 2 to Firefox 3). Similarly, small changes to the operating system have no perceived effect.

4. Results

We have compiled profiles for the following XP and Vista applications: Acrobat, Acrobat Reader, Excel, Firefox, MS Calculator, Outlook, WordPad, Word, and Notepad. The tests were performed using the 32-bit Windows XP operating system for our tests.

The processes were stock programs running no hidden functions. The processes executed 3 different common applications: Word, Excel, and Outlook. We selected those programs because they are commonly used and complex, generating a long list of DLLs, open files and registry keys. The test consisted of starting the program and performing some routine actions, such as entering text into Word or reading a message and opening the attached file. Then, we created a forensic snapshot under OnLineDFS using its "initial acquisition" feature. The initial results are promising.

In every case, the displays of both DLLs and keys shrank dramatically, as shown in Figure 1. The filtered list of DLLs generally fit on a single page, and the handle list, while longer, was much more easily perused. The filtering process did not remove any significant data, such as the .doc file that Word was editing or the Acrobat process started by Outlook on a PDF attachment. Note that some of the reduction in the number of handles, especially Outlook, is due to the automatic filtering of all "esoteric" handles, specifically events, which are unrelated to the resource profiling.

The number of files, however, remains almost the same. We believe this is caused by the way applications use files. The forensic snapshot represents a very small amount of the total run-time of the application. For most of an application's run time, few files are held open continuously. Most applications open a file, perform the read or write action, and then close the file. Therefore, it is unlikely that the forensic snapshot will record many open files. The forensic snapshot records many "device" files, e.g., \device\tcp, which is not considered a resource by our AppMon code and will not appear in our profile. And finally, the current version of our filtering code only removes the *first* occurrence of a resource that appears in the profile. Future versions of the code will remove all duplicates. We note that duplicate file and registry handles commonly occur. While the removal of duplicates will significantly improve the results shown in Figure 1, it is tangential to the improvements from profiling, as is the suppression of the shared memory and other esoteric handles. The results presented in Figure 1 are due only to the profiling.

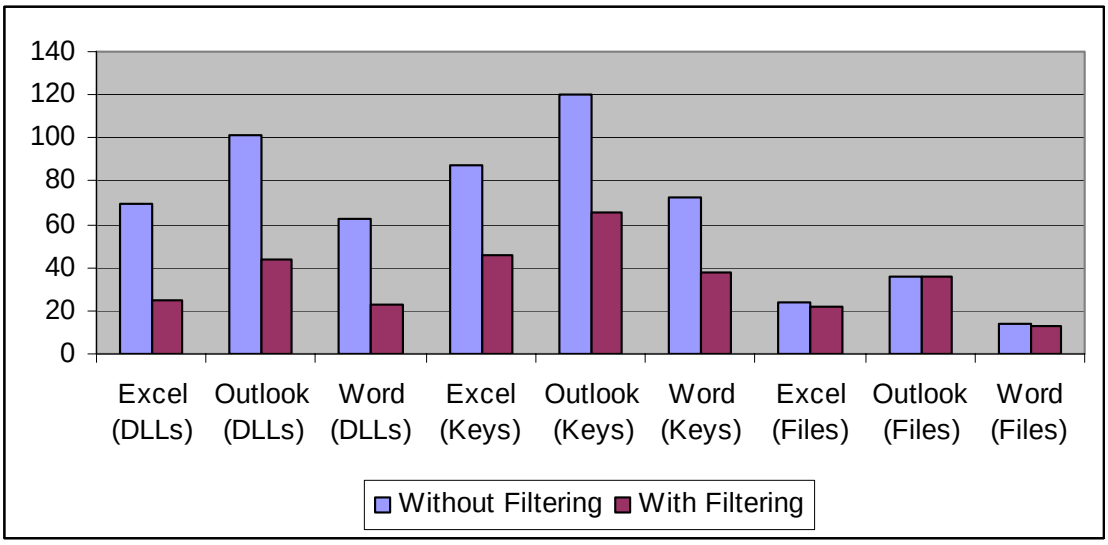

Figure 1. Number of profile entries with and without filtering

5. Conclusion and future work

The preliminary results are encouraging. In general, the filtering process significantly reduces the DLLs and handles displayed significantly. We used the Microsoft Office tools as examples because they are of general interest and tend to use a large number of DLLs and handles. By adopting a principled way to automatically filter uninteresting data from running processes, live forensic tools can provide a higher quality of data, which enables investigators to more quickly find forensically significant data, such as user-specified files written by a process or registry keys that contain process-state information.

While the tests were based on applications for which profiles already existed, an investigator could easily generate profiles *after* taking the forensic snapshot. Therefore, this filtering technique is not limited to processes for which a profile *currently* exists.

Incomplete profiles act like less efficient filters; they allow some useless data to be displayed, but still filter out most chaff. It is unlikely that profiles contain forensically useful data. An investigator who is interested in files or keys that the application always uses can turn off filtering (by default it is off).

Future work includes duplicate handle suppression, more extensive tests, additional profiles, and tools to automate data collection for profiles. While the profile generation is automated, users are required to execute the applications, performing tasks to select menus, click on buttons, and provide input. Another avenue for future work is to determine the effectiveness of profiles that were generated very quickly, for example, profiles that were created by starting an application and selecting only a few standard menu items such as File->new, File->Save As, and File->Print. Other work includes exploring different filtering methods and isolating application/user-specific resources.

6. References

[1] Frank Adelstein, "MFP: The Mobile Forensic Platform," *International Journal of Digital Evidence*, Volume 2, Issue 1, Spring 2003.

[2] S. Forrest, S. A. Hofmeyr and A. Somajayi, "A Sense of Self for UNIX Processes," in Proceedings of IEEE Symposium on Computer Security and Privacy, 1996.

[3] D. Gao, M. K. Reiter and D. Song, "On Gray-Box Program Tracking for Anomaly Detection," in Proceedings of USENIX Security Symposium, 2004, pp. 103-118.

[4] A. K. Ghosh, A. Schwatzbard and M. Shatz, "Learning Program Behavior Profiles for Intrusion Detection," in Proceedings of 1st USENIX Workshop on Intrusion Detection and Network Monitoring, 1999.

[5] K. A. Heller, K. M. Svore, A. D. Keromytis and S. J. Stolfo, "One Class Support Vector Machines for Detecting Anomalous Window Registry Accesses," in Proceedings of 3rd IEEE Conference Data Mining Workshop on Data Mining for Computer Security, 2003.

[6] S. A. Hofmeyr, S. Forrest and A. Somayaji, "Intrusion detection using sequences of system calls," in Journal of Computer Security 6 (3), pp. 151–180S. A. Hofmeyr, S. Forrest and A. Somayaji, "Intrusion detection using sequences of system calls," in Journal of Computer Security.

[7] W. Lee and S. J. Stolfo, "Data mining approaches for intrusion detection," in Proceedings of 7th USENIX Security Symposium, 1998.

[8] C. Marceau, "Characterizing the Behavior of a Program Using Multiple-Length N-grams," in Proceedings of New Security Paradigms Workshop, 2000.

[9] Carla Marceau and Rob Joyce, “Empirical privilege profiling,” in *Proceedings of the 2005 New Security Paradigms Workshop*, Lake Arrowhead, California, *September 20 - 23, 2005.*

[10] R. Sekar, M. Bendre, D. Dhurjati and P. Bollineni, "A fast automaton-based method for detecting anomalous program behaviors," in Proceedings of IEEE Symposium on Security and Privacy, 2001, pp. 144-155.

[11] U. o. N. M. Computer Science Department, Computer Immune Systems: Data Sets and Software, http://www.cs.unm.edu/~immsec/systemcalls.htm.

[12] D. Wagner and P. Soto, "Mimicry attacks on host based intrusion detection systems," in Proceedings of Ninth ACM Conference on Computer and Communications Security, 2002.

Note: The work on resource usage profiles was supported by DARPA contract W31P4Q-06-C-0134 and AFOSR STTR contract FA9550-06-C-0098.