AccountabilityFS: A file system monitor for forensic readiness

Rune Nordvik, Yi-Ching Liao, and Hanno Langweg
Norwegian Information Security Laboratory
Gjøvik University College, Norway
Email: {rune.nordvik, yi-ching.liao, hanno.langweg}@hig.no

***Abstract*—We present a file system monitor, AccountabilityFS, which prepares an organization for forensic analysis and incident investigation in advance by ensuring file system operation traces readily available. We demonstrate the feasibility of AccountabilityFS in terms of performance and storage overheads, and prove its reliability against malware attacks.**

I. Introduction

Forensic analysis and incident investigation can be expensive if an organization is not well-prepared. The cost can be greatly reduced by continuously gathering information that crime investigators or incident responders require and preserving the collected information in a legally acceptable manner. Even though preparing for forensic analysis and incident investigation might appear to be burdensome, the advantages outweigh the disadvantages. For instance, a data breach occurs when an insider violates data security policies and steals sensitive information by copying files to a USB drive. Organizations that fail to prepare for the link between the user and file system operations have to pay high price for the data breach, and may have no lead to figure out who is responsible.

Current system loggers cannot provide enough information for forensic analysis and incident investigation, and crime investigators and incident responders demand fine-grained information for crime scene reconstruction and root cause analysis. Process activity tracking, which monitors and records information about the execution of programs, can be a suitable method to achieve forensic readiness. Comprehensive process activity tracking should cover several system resources, such as file system, network, and memory [1]. In this paper, we focus on tracking file system operations, and present a file system monitor, AccountabilityFS, which tracks system-wide file system operations for forensic readiness.

Prior research on file system monitor mostly implemented through file integrity monitoring and system call interposition. File integrity monitoring is insufficient for forensic readiness due to lack of fine-grained activity traces. Compared to implementing system call interposition in user space being vulnerable to forgery and manipulation, in-kernel interposition is a better way to provide admissible evidence for forensic readiness. Since in-kernel interposition requires modification of kernel source code, which is not publicly available on Microsoft Windows, most previously research employed system call interposition to track file system operations remains limited in Unix-like operating systems.

Since most desktop and laptop computers use Windows according to the usage share of operating systems, we implement AccountabilityFS for modern versions of Windows, including Windows 7, 8 and 8.1 (32-bit and 64-bit). We demonstrate the feasibility of AccountabilityFS for tracking file system operations by analysing its performance and storage overheads. For admissibility of collected traces, we evaluate the reliability of AccountabilityFS through malware attack simulation. We show that AccountabilityFS can suitably prepare an organization for forensic analysis and incident investigation in advance by tracking system-wide file system operations.

II. Related Work

File integrity monitoring is a commonly used technique for intrusion detection and policy compliance. Tripwire [2], a file system integrity checker, monitors unauthorized or unexpected modifications within file systems by identifying changes through the message digests and meta-data of designated file system objects. I3FS [3] enhances Tripwire with real-time support by reducing the performance overhead, and improves the system robustness through a loadable kernel module for file systems. Even though monitoring the modifications within file systems is beneficial for intrusion detection and policy compliance, for traceability, incident investigation and forensic analysis demand more fine-grained activity records than merely file integrity verification.

For fine-grained activities, previously research mainly employed system call interposition to track file system operations. A file system tracing package for Berkeley UNIX [4] tracks file system operations through system call hooking inside the kernel. Modified BSD kernel [5] intercepts file system relevant system calls to analyse the BSD file system. DFSTrace [6] supports long-term file system activity monitoring through instrumenting system calls in kernel. Tracefs [7], implemented as a loadable kernel module, intercepts file system relevant system calls by stacking on top of the underlying file system. //TRACE [8] captures I/O system calls through dynamic library interposition [9], which utilizes LD_PRELOAD environment variable to interpose on system calls. While tracing through dynamic libraries causes significant performance overhead and is easily compromised in user space, in-kernel system call interposition requires the kernel source code, which is not publicly available on Windows.

To track file system operations on Windows, display-only file server [10], built on FileMon [11], defends against information theft by intercepting I/O Request Packet (IRP) requests associated with reading and creating files. The filter

driver of FileMon can also intercept all requests to an existing file system device for performance improvement [12]. Capture [13], an application behavioural analysis tool, utilizes minifilter driver as a file monitor which resides between the I/O manager of Windows kernel. A minifilter driver can intercept requests before they reach the intended resource, and monitor system-wide file system activities without hacking the kernel. Capture-honeypot client [14] also employs minifilter driver to monitor states of changes on the file system. Tarantula [15], a behaviour-based malware detection system, identifies system files and registry as critical system resources, and utilizes minifilter driver to prevent malicious activities from accessing the critical system resources. Compared to hooking native kernel API functions, a minifilter driver is more recommendable by Microsoft due to its compatibility and reliability. However, these file system monitors are either outdated or incompatible with modern versions of Windows.

III. Design Goals

To meet the two objectives of forensic readiness: maximizing the usefulness and minimizing the cost [16], we set the following design goals to guide us throughout the implementation of AccountabilityFS for forensic readiness:

A. Feasibility

To meet the objective of minimizing the cost for forensic readiness, AccountabilityFS should cost no extra system overhead before activation, and cause unnoticeable performance and storage overheads during tracking. In other words, AccountabilityFS should record system-wide file system operations without interfering with business processes.

B. Comprehensiveness

To provide a detailed full picture of file system activities, AccountabilityFS has to record all the relevant file operations without collecting the irrelevance. Complete and accurate information allows malicious file system operations to be traced back to one or more processes or programs, which ensures the traceability of file system operations.

C. Reliability

To provide admissible evidence, AccountabilityFS should secure the collected traces from being forged, manipulated or planted to incriminate others. Moreover, since the credibility of AccountabilityFS will cause severe impact on the authenticity of collected traces, AccountabilityFS should defend itself against errors or attacks during tracking.

D. Flexibility

To keep up with evolving hacking techniques, AccountabilityFS needs to provide flexibility for different tracking scenarios. Furthermore, system administrators should be able to activate and deactivate AccountabilityFS dynamically without any disruption. However, for reliability concerns, AccountabilityFS must be activated and deactivated with administrator privilege.

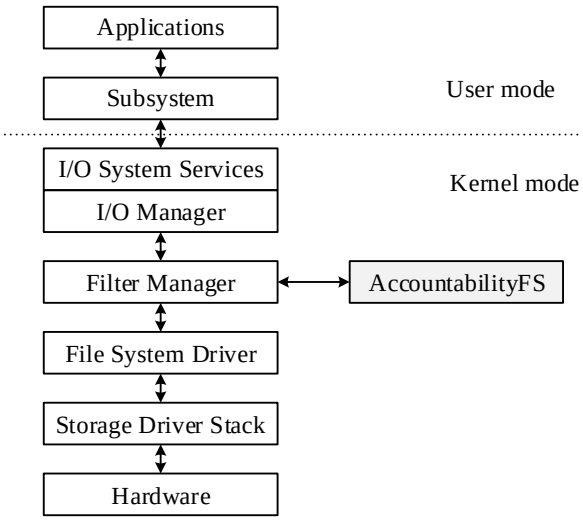

Fig. 1. A simplified Windows I/O architecture with AccountabilityFS

TABLE I. File system operations tracked by AccountabilityFS

Major function codes	File system operations
IRP_MJ_CREATE	Create/Open a file or I/O device
IRP_MJ_READ	Read data from the specified file or I/O device
IRP_MJ_WRITE	Write data to the specified file or I/O device
IRP_MJ_SET_INFORMATION	File system operation information
IRP_MJ_QUERY_INFORMATION	Retrieve file information for the specified file
IRP_MJ_CLEANUP	Cleanup outstanding I/O requests
IRP_MJ_CLOSE	Close the last open object handle

IV. Implementation

To implement AccountabilityFS, we choose C as the programming language, and employ Microsoft Visual Studio 2013 and Windows Driver Kit 8.1 as development tools. Currently AccountabilityFS tracks file system operations successfully on Windows 7, 8 and 8.1 (32-bit and 64-bit). We explain the implementation details as follows:

A. Architecture

Fig.1 presents a simplified Windows I/O architecture with AccountabilityFS [17]. Applications send I/O requests to subsystem, and the subsystem passes I/O requests by calling I/O system services exported by the I/O manager. The I/O manager generates I/O request packets (IRPs) for I/O requests to kernel-mode drivers. Afterwards, the filter manager intercepts IRPs, and calls the preoperation callback routines registered by AccountabilityFS. The file system driver processes the modified IRPs from the filter manager, and forwards them to the storage driver stack, which prepares requests for hardware.

B. Operations Tracked

AccountabilityFS tracks file system operations by intercepting major function codes in its preoperation callback routine. Table I summarizes the major function codes we specify to intercept along with the file system operations.

TABLE II. TRACE STRUCTURE OF ACCOUNTABILITYFS

Field	Description
host	Host name, retrieved from Windows registry
pname	Name of the process in full path
time	Event start time in ISO8601 time format
msg	Operation type, retrieved from IoStatus parameter
msgid	Message identifier, used as globally unique identifier for event
pri	Event priority (ERROR/WARN/DEBUG/CRIT)
pid	Process identifier for the event
file.path	Target object in full path, null for the IRP_MJ_CLEANUP request
cmdline	Command line used to launch the process, null if unavailable
user.id	Security Identifier (SID) of user
user.name	User name
action	Action type (e.g., create, write, read, close and remove)
status	Action execution status (e.g., success and failure)
native	Includes other important information non-compulsory in CEE CLS

C. Trace Structure

To facilitate trace analysis for incident investigation and forensic analysis, AccountabilityFS records file system operations as Common Event Expression (CEE) [18] events, and represents the collected traces in Common Log Syntax (CLS) with JavaScript Object Notation (JSON) and UTF-8 encoding. Table II illustrates the trace structure of AccountabilityFS. Currently we extend the CEE CLS by storing parent process identifier, intercepted major function codes, action execution status in detail, target object type, CPU number, volume serial number, volume path, and media type in the native field.

D. Flexibility Enhancement

In addition to comprehensive file system operation tracking, AccountabilityFS supports whitelisting/blacklisting functionalities for flexibility enhancement. System administrators can include or exclude the processes, programs, or even files for tracking by specifying them in the set-up information (INF) file. For instance, the command `HKR, ,"whitelistpname",0x00010000,"EXPLORER.EXE"` excludes the EXPLORER.EXE program from tracking by setting the registry value of whitelistpname.

E. Protection Mechanism

We employ discretionary access control lists (DACLs) to protect AccountabilityFS against unauthorized access. We set access control entry (ACE), a pair of account and access-rights in the DACLs, to restrict built-in users from gaining access to AccountabilityFS system file, collected traces, and configuration settings in registry. For reliability enhancement, we employ the GRR rapid response framework [19] for remote logging if internet connection is available. The GRR rapid response framework secures the trace transfer with Advanced Encryption Standard (AES) 256-bit encryption, and authenticates the transmitter through X.509 certificate.

V. EVALUATION

To evaluate AccountabilityFS for forensic readiness, we utilize the four design goals described in section III as the evaluation criteria for quality assurance. We present the evaluation results as follows:

TABLE III. PERFORMANCE OVERHEAD OF ACCOUNTABILITYFS

Evaluation scenario	AccountabilityFS	CPU usage		
		Kernel	User	Idle
Minimum activity	Deactivated	3.53%	0.91%	95.56%
	Activated	3.97%	0.75%	95.28%
Maximum activity	Deactivated	9.03%	8.42%	82.55%
	Activated	16.09%	8.1%	75.81%

A. Feasibility

To analyse the performance and storage overheads, we deploy the experimental testbed using VMware virtual machine installed with Windows 8.1 64-bit operating system, which utilizes a single CPU core and 1 GB of memory. We employ the CPU usage as the indicator of performance overhead, which is retrieved by Kernrate, CPU profiler tool available from Windows Driver Kit 7.1. We evaluate the performance overhead in two scenarios: minimum activity, which simply executes Kernrate for 300 seconds, and maximum activity, which unzipping a 1.5 GB file. We run each scenario with and without tracking twice, and calculate the average of kernel-mode CPU usage, user-mode CPU usage, and idle CPU usage in percentage, which is shown in Table III. As for the storage overhead, AccountabilityFS requires an average of 0.1036 MB per second to record file system operations in the scenario of minimum activity, and 0.3825 MB per second in the scenario of maximum activity.

B. Comprehensiveness

As shown in Fig.1, the I/O manager generates IRPs for all I/O requests, and the filter manager intercepts IRPs and calls the preoperation callback routines registered by AccountabilityFS. Therefore, in theory, AccountabilityFS can track file system operations we specify. In addition to employing specification-based testing with several test cases (e.g., remove, move, copy, open, create, write, and read file objects), we also utilize IOzone [20], a filesystem benchmark tool, to assure AccountabilityFS works as expected and records all the specified file operations.

C. Reliability

For admissibility of collected traces, we evaluate the reliability of AccountabilityFS through malware attack simulation. We utilize the same experimental testbed for feasibility evaluation, and use 500 malware samples confirmed by VirusSign [21] to evaluate the robustness of AccountabilityFS. The execution result shows that AccountabilityFS successfully tracks file system operations of all the malware samples without causing system crash.

D. Flexibility

Through supporting whitelisting/blacklisting functionalities, AccountabilityFS is capable of tracking file system operations by specified processes, programs, or even files. Moreover, since the filter manager provides safe removal for minifilter drivers, AccountabilityFS can be loaded and unloaded with administrator privilege while the system is still running without any disruption.

VI. Discussion

According to the feasibility evaluation results, AccountabilityFS can record system-wide file system operations without interfering with business processes when file system operations are not intensive. To reduce the storage overhead, in addition to making a trade-off between feasibility and comprehensiveness, we can consider data compression or storing the collected traces in compact binary format. As for comprehensiveness, even though we have assured AccountabilityFS works as expected and tracks file system operations we specify, the scope of file system operation tracking is still limited by major function codes intercepted, which can be extended after analysing the relevance and frequency of file system operations. With regard to reliability, AccountabilityFS proves its resistance to 500 malware samples confirmed by VirusSign while the GRR rapid response framework secures the trace transfer. However, we still need to conduct further security and vulnerability assessments to identify unknown problems for ensuring the collected traces as admissible evidence, such as simulating a DoS or insider attack. About flexibility, supporting whitelisting/blacklisting functionalities and dynamic deactivation also raise new security concerns to be addressed.

VII. Conclusions and Future Work

Process activity tracking, which monitors and records information about the execution of programs, is among the solutions for the problem of incomplete logging in incident investigation and forensic analysis. In this paper, we present a file system monitor for forensic readiness. We implement AccountabilityFS as a file system minifilter driver, since a minifilter driver is more recommendable by Microsoft due to its compatibility and reliability. To ensure the objectives of forensic readiness are met, we set four design goals for AccountabilityFS: feasibility, comprehensiveness, reliability, and flexibility. In addition to utilizing the four design goals throughout the implementation, we use them as evaluation criteria for assessing AccountabilityFS. The evaluation results show that AccountabilityFS archives the goals of comprehensiveness, reliability, flexibility, and partial feasibility if file system operations are not intensive.

We will continue the evaluation and examine the privacy implications of file system operation tracking. We will assess the feasibility of AccountabilityFS for other benefits of forensic readiness, including due diligence, regulatory compliance, and human error reduction. In the future, we plan to extend the tracking scope to memory management, network, interprocess communication, etc. We will make the source code of AccountabilityFS available to interested researchers under an appropriate open source licenses.

Acknowledgment

Yi-Ching Liao is supported by the COINS Research School of Computer and Information Security.

References

[1] Y.-C. Liao and H. Langweg, "A survey of process activity tracking system," *Norsk informasjonssikkerhetskonferanse (NISK)*, pp. 49–60, 2013.

[2] G. H. Kim and E. H. Spafford, "The design and implementation of tripwire: A file system integrity checker," in *Proceedings of the 2Nd ACM Conference on Computer and Communications Security*, ser. CCS '94. New York, NY, USA: ACM, 1994, pp. 18–29.

[3] S. Patil, A. Kashyap, G. Sivathanu, and E. Zadok, "I3fs: An in-kernel integrity checker and intrusion detection file system," in *Proceedings of the 18th Conference on Systems Administration (LISA 2004), Atlanta, USA, November 14-19, 2004*, L. Damon, Ed. USENIX, 2004, pp. 67–78.

[4] S. Zhou, H. Da Costa, and A. J. Smith, "A file system tracing package for berkeley UNIX," in *Proc. USENIX Summer Conference*. University of California, 1985.

[5] J. K. Ousterhout, H. Da Costa, D. Harrison, J. A. Kunze, M. Kupfer, and J. G. Thompson, "A trace-driven analysis of the UNIX 4.2 BSD file system," in *Proceedings of the Tenth ACM Symposium on Operating Systems Principles*, ser. SOSP '85. New York, NY, USA: ACM, 1985, pp. 15–24.

[6] L. Mummert and M. Satyanarayanan, "Long term distributed file reference tracing: Implementation and experience," Carnegie Mellon University, Tech. Rep. CMU-CS-94-213, 1994.

[7] A. Aranya, C. P. Wright, and E. Zadok, "Tracefs: A file system to trace them all," in *Proceedings of the FAST '04 Conference on File and Storage Technologies, March 31 - April 2, 2004, Grand Hyatt Hotel, San Francisco, California, USA*, C. Thekkath, Ed. USENIX, 2004, pp. 129–145.

[8] M. P. Mesnier, M. Wachs, R. R. Sambasivan, J. Lpez, J. Hendricks, G. R. Ganger, and D. R. O'Hallaron, "//TRACE: Parallel trace replay with approximate causal events," in *5th USENIX Conference on File and Storage Technologies, FAST 2007, February 13-16, 2007, San Jose, CA, USA*, A. C. Arpaci-Dusseau and R. H. Arpaci-Dusseau, Eds. USENIX, 2007, pp. 153–167.

[9] T. W. Curry, "Profiling and tracing dynamic library usage via interposition," in *USENIX Summer 1994 Technical Conference, Boston, Massachusetts, USA, June 6-10, 1994, Conference Proceeding*. USENIX Association, 1994, pp. 267–278.

[10] Y. Yu and T.-c. Chiueh, "Display-only file server: A solution against information theft due to insider attack," in *Proceedings of the 4th ACM Workshop on Digital Rights Management*, ser. DRM '04. New York, NY, USA: ACM, 2004, pp. 31–39.

[11] M. Russinovich and B. Cogswell, "FileMon for windows," 2006. [Online]. Available: http://technet.microsoft.com/en-us/sysinternals/bb896642.aspx

[12] D. S. Roselli, J. R. Lorch, and T. E. Anderson, "A comparison of file system workloads," in *Proceedings of the General Track: 2000 USENIX Annual Technical Conference, June 18-23, 2000, San Diego, CA, USA*. USENIX, 2000, pp. 41–54.

[13] C. Seifert, R. Steenson, I. Welch, P. Komisarczuk, and B. Endicott-Popovsky, "Capture - a behavioral analysis tool for applications and documents," *Digital Investigation*, vol. 4, Supplement, pp. 23–30, Sep. 2007.

[14] C. Seifert and R. Steenson, "Capture - honeypot client (capture-HPC)," 2006. [Online]. Available: https://projects.honeynet.org/capture-hpc

[15] S. Romana, S. Phadnis, H. Pareek, and P. R. L. Eswari, "Behavioral malware detection expert system - tarantula," in *Advances in Network Security and Applications*, ser. Communications in Computer and Information Science. Springer Berlin Heidelberg, Jan. 2011, no. 196, pp. 65–77.

[16] J. Tan, "Forensic readiness," *Cambridge, MA:@ Stake*, pp. 1–23, 2001.

[17] Microsoft, "Filter manager concepts (windows drivers)," 2014. [Online]. Available: http://msdn.microsoft.com/en-us/library/windows/hardware/ff541610(v=vs.85).aspx

[18] W. J. H. A. Chuvakin, J. T. J. R. Marty, and R. M. Mcquaid, "Common event expression," The MITRE Corporation, Tech. Rep., 2008.

[19] M. I. Cohen, D. Bilby, and G. Caronni, "Distributed forensics and incident response in the enterprise," *Digital Investigation*, vol. 8, Supplement, pp. S101–S110, Aug. 2011.

[20] W. Norcott and D. Capps, "IOzone," 2006. [Online]. Available: http://www.iozone.org/

[21] VirusSign, "VirusSignList," Mar. 2014. [Online]. Available: http://samples.virussign.com/samples