Detecting Data Theft Using Stochastic Forensics

Jonathan Grier

Abstract

We present a method to examine a filesystem and determine if and when files were copied from it. We develop this method by stochastically modeling filesystem behavior under both routine activity and copying, and identifying emergent patterns in MAC timestamps unique to copying. These patterns are detectable even months afterwards. We have successfully used this method to investigate data exfiltration in the field. Our method presents a new approach to forensics: by looking for stochastically emergent patterns, we can detect silent activities that lack artifacts.

1 Background

Theft of corporate proprietary information, according to the FBI and CSI, has repeatedly been the most financially harmful category of computer crime (CSI & FBI, 2003). Insider data theft is especially difficult to detect, since the thief often has the technical authority to access the information (Yu & Chiueh, 2004; Hillstrom & Hillstrom, 2002). Frustratingly, despite the need, no reliable method of forensically determining if files have been copied has been developed (Carvey, 2009, p. 217). Methods do exist to detect particular actions often associated with copying, such as attaching a removable USB drive (Carvey, 2009; Carvey & Altheide, 2005). Methods also exist that can detect copying when given a network trace of the activity (Liu *et al.*, 2009), or when given the media to which the files were copied to (Chow *et al.*, 2007). However, no method has yet been discovered that given only a filesystem can determine if its files were copied. Carvey summarizes this problem: (Carvey, 2009, p. 217), "there are no apparent artifacts of this process [of copying data]. . . . Artifacts of a copy operation. . . are not recorded in the Registry, or within the file system, as far as I and others have been able to determine."

In this paper, we develop a method to do exactly that: analyze a filesystem to determine if and when its files were copied. We report on the foundations of our method (Section 3), simulated trials (Section 4), its mathematical basis (Section 5), and usage in the field (Section 6).

2 Can we use MAC timestamps?

Farmer and Venema's seminal work (Farmer, 2000; Venema, 2000; Farmer & Venema, 2004) describes reconstructing system activity via MAC timestamps. MAC timestamps are filesystem metadata which record a file's most recent *M*odification, *A*ccess, and *C*reation times. By plotting these on a timeline, investigators can reconstruct filesystem activity, and hence computer usage, of a particular time. An investigator can also plot a histogram of filesystem activity, showing amount of activity per time period (Casey, 2004).

Seemingly, we should be able to use MAC timestamps to detect data exfiltration. However, as mentioned above, the standard methods of MAC timestamp analysis fail to do this. Neither timelines nor histograms can distinguish copying from other forms of file access.

Moreover, Microsoft Windows NTFS systems do not update a file's access timestamp when it is copied. Unlike Unix based systems, which implement copy commands in user code via standard reads of the source file and writes to the destination file (Sun Microsystems, Inc., 2009a,b; Free Software Foundation, Inc., 2010), Windows provides a dedicated `CopyFile()` system operation (Microsoft Corporation, 2010a). Thus, Unix based filesystems do not distinguish copying a file from other forms of accessing it; both are done via `read()`, and both update the file's access timestamp. (This was experimentally confirmed using the `cp` command on a Linux 2.6.25 ext3 system.) Windows, however, distinguishes between the two at the system level. Our experiments (performed on a Microsoft Windows XP Professional 5.1.2600 system) confirm that Windows indeed does not update the access timestamp of the source file when copying it, making file copying seemingly invisible.

3 Emergent patterns caused by copying

To be able to detect copying, we must refine our model of its filesystem activity. For the rest of this paper, we concern ourselves with the copying of an entire folder with numerous subfolders and files; we believe this to be the typical form of data exfiltration.

We can distinguish between the access pattern of copying and that of routine access. Routine file access is *selective*: individual files and folders are opened while others are ignored. It is also *temporally irregular*: files are accessed in response to user or system activity, followed by a lull in access until the next activity causes new file access. Copying of folders, however, is *nonselective*: every file and subfolder within the folder is copied. It is furthermore *temporally continuous*: files are copied sequentially without pause until the entire operation is complete. Copying folders is also *recursive*: copying one folder invokes the copying of all subfolders, which each invoke copying of their subfolders, and so on, while routine activity is *randomly ordered* (see Table 1).

This recursive nature of copying results in an additional trait. To copy a folder, the system must enumerate the folder's contents. Modern filesystems implement folders as special types of files called *directories*; to enumerate a folder's contents, the system accesses and reads the directory file. Thus, copying will invariably access a directory before accessing its files and subfolders. What's more, since this is a data read and not a file copy, Windows NTFS *does* update the access time of the *directory* when its contents are enumerated. Our experiments confirmed that on both the above Windows and Linux systems, copying a folder updates the access time of the folder's *directory* and all subdirectories.

Copying Folders	Routine Access
Nonselective (all subfolders and files accessed)	Selective
Temporally Continuous	Temporally Irregular
Recursive	Random Order
Directory accessed before its files	Files may be accessed without directory
On Windows: Directory timestamps updated, but not file	Both directory and file timestamps updated

Table 1: Differences in access timestamp updates between copying folders and routine activity

Thus, although, as stated above, copying creates no individual artifact, it does create distinct *emergent patterns*. A filesystem examined immediately after copying occurs will show the five characteristics enumerated in Table 1.

However, we cannot yet apply this technique in the field: MAC timestamps, notorious for being quickly overwritten, are unreliable. And other types of recursive access besides copying may also cause such emergent patterns. We address these problems in Section 4 and Section 7.

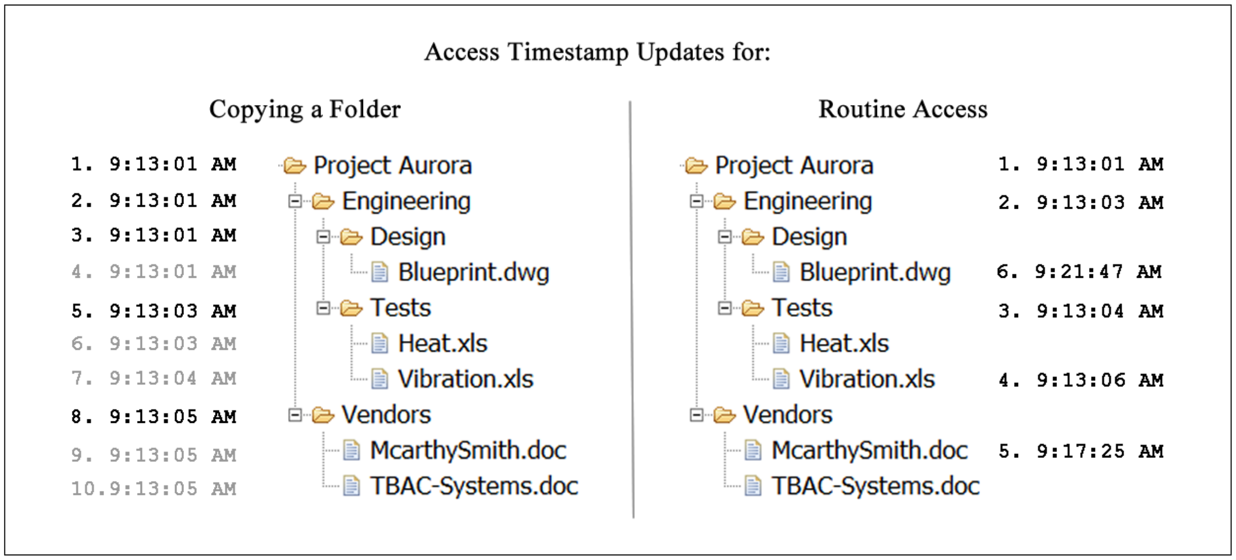
Access Timestamp Updates for:
Copying a Folder
Routine Access
1. 9:13:01 AM Project Aurora
2. 9:13:01 AM Engineering
3. 9:13:01 AM Design
4. 9:13:01 AM Blueprint.dwg
5. 9:13:03 AM Tests
6. 9:13:03 AM Heat.xls
7. 9:13:04 AM Vibration.xls
8. 9:13:05 AM Vendors
9. 9:13:05 AM McarthySmith.doc
10. 9:13:05 AM TBAC-Systems.doc
Project Aurora 1. 9:13:01 AM
Engineering 2. 9:13:03 AM
Design
Blueprint.dwg 6. 9:21:47 AM
Tests 3. 9:13:04 AM
Heat.xls
Vibration.xls 4. 9:13:06 AM
Vendors
McarthySmith.doc 5. 9:17:25 AM
TBAC-Systems.doc

Figure 1: The left side shows the access timestamp updates that would occur upon copying folder *Project Aurora*. Updates that would occur on Linux but not on Windows are shown in gray. The right side shows updates that might occur during typical user activity, which, unlike folder copying, is selective, temporally irregular, and randomly ordered (see Table 1).

4 Digging for footprints

Although we have identified distinct emergent patterns caused by copying, we should be skeptical about using them in real world investigations. Timestamps are notoriously ephemeral: like footprints, they are swept away by newer activity (Farmer & Venema, 2004). If an investigation is performed weeks or months after the data theft, do we have any hope of unearthing these emergent patterns in timestamps?

Surprisingly, the answer is yes: we can indeed detect them even months after the copying, and even when the date of the alleged copying is unknown. To do so, we must make two observations: First, while normal system activity (ignoring things like intentional tampering or resetting the system clock) can increase access timestamps to more recent times, it cannot decrease them. Thus, although access timestamps are extremely volatile (as each access overwrites the previous timestamp), *they nonetheless maintain an invariant of always increasing monotonically*.

Second, filesystem activity is by no means uniformly, or even normally, distributed over files. Activity more closely resembles heavy-tailed distributions, such as a Pareto distribution (Wikipedia, 2010): a small amount of files generally account for a large portion of activity, with a significant amount of files undergoing negligible activity (Vogels, 1999; Gribble *et al.*, 1998; Ferguson, 2002). Farmer and Venema (Farmer & Venema, 2004, p. 4) report that over periods as long as a year, the majority of files on a typical server are not accessed at all.

Consequently, if a folder was copied, we can expect to find the following, even if several weeks or months have elapsed since the time of copying:

- Neither the copied folder, nor any of its subfolders, have access timestamps less than the time of copying.
- A large number of these folders have access timestamps equal to the time of copying.

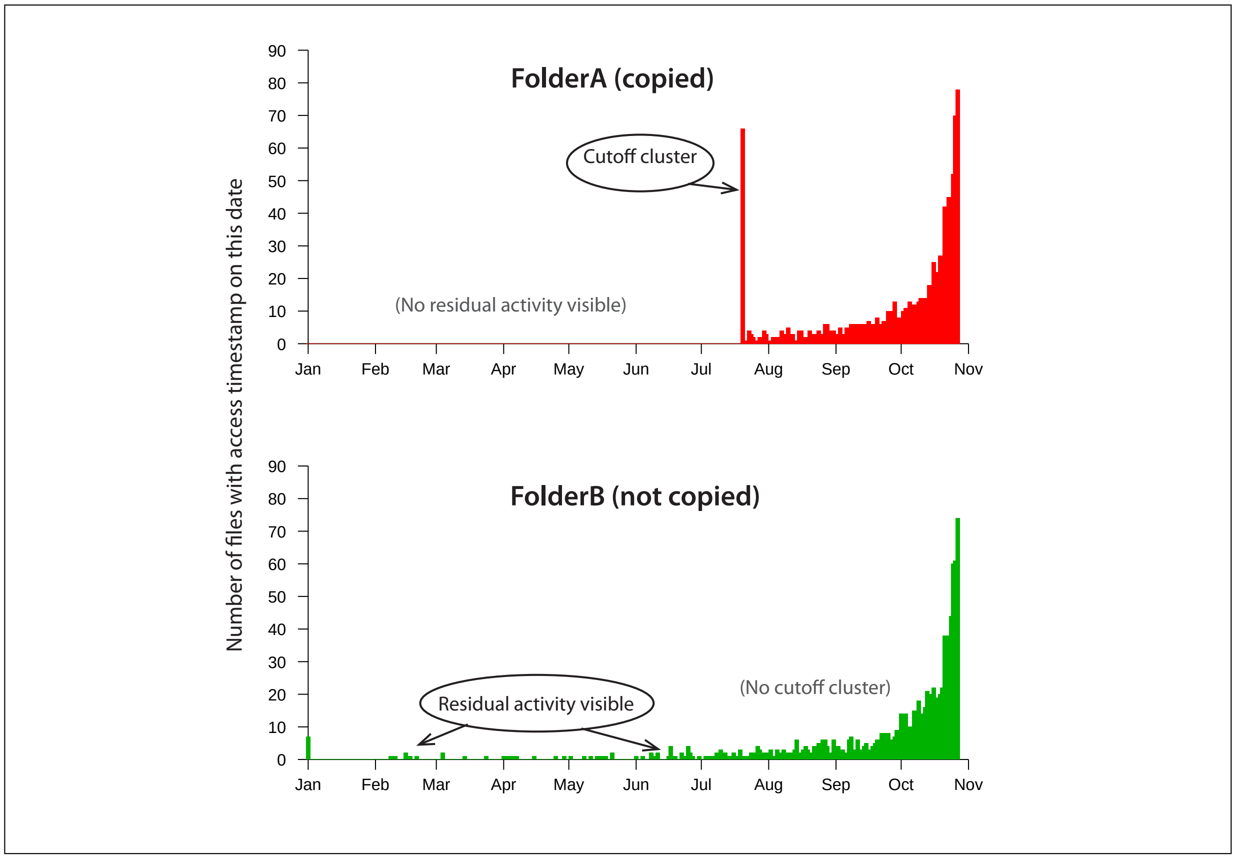

Figure 2: Access timestamps of two folders after 300 days of simulated activity. Note that the cutoff cluster of *FolderA*, caused by its copying on day 200, is clearly visible even 100 days later, even with a large amount of subsequent activity.

- On Windows, file timestamps will *not* resemble folders' timestamps. Specifically, many files will have access timestamps *before* any of the folders.

Copying thus creates an artifact which we call a *cutoff cluster*: a point in time which no subfolder has an access timestamp prior to (hence a *cutoff*), and which a disproportionate number of subfolders have access timestamps equal to (hence a *cluster*). We generally expect a folder to have a number of rarely accessed subfolders, which cause the cutoff cluster to remain detectable for several weeks or months (or until the next act of copying). Conversely, in the absence of copying (or other nonselective, recursive access), we expect to find some folders with access timestamps extending far back in time, consistent with a heavy-tailed distribution.

To explore this, we simulated a model filesystem containing two folders, *FolderA* and *FolderB*. Each folder has 1000 children (files or subfolders), created at the start of the simulation, and is accessed approximately 100 times a day. File access is distributed randomly using a Pareto distribution. *FolderA* is copied 200 days after the start of the simulation; *FolderB* is never copied. After 300 days of simulation, we tabulated the date of most recent access for each file; that is, the files' access timestamps. Both folders had more than half of their files accessed within the final two weeks of the simulation. Nonetheless, we are able to identify a clear cutoff cluster occurring at the time of copying of *FolderA* (see Figure 2). (An interactive version of the simulator is available at `http://www.vesaria.com/datatheft/sim/` .)

Note that, on Windows, a cutoff cluster is invisible unless we first filter the histogram to include only subfolders (and not files).

In short, *if a large folder is copied, it will result in a cutoff cluster; this cutoff cluster can be detected months after the date of copying*, even with low resolution timestamps and substantial amounts of noise.

5 Quantitative analysis of cutoff clusters

We now proceed to define metrics allowing us to quantitatively detect and measure cutoff clusters. For the remainder of this paper, we concern ourselves with systems such as Windows NTFS, which do not update file access timestamps on copy. Modifying our method for use with systems such as Linux ext3, which do update file access timestamps on copying, is straightforward.

We use the conventional model of a filesystem as a tree, with each subfolder a child of its parent folder. For each folder f, we define

$$D(f) = \{f\} \cup \{x | x \text{ is a descendant folder of } f\} \ .$$

That is, $D(f)$ is the set of f and all of its descendant folders. Note that only folders, and not files, are members of f. For a given time t, we partition $D(f)$ into four disjoint subsets:

$$\begin{aligned}
Db_t(f) &= \{x | x \in D(f) \wedge access\_timestamp(x) < t \wedge creation\_timestamp(x) < t\} \\
De_t(f) &= \{x | x \in D(f) \wedge t \leq access\_timestamp(x) \leq t + \varepsilon \wedge creation\_timestamp(x) < t\} \\
Da_t(f) &= \{x | x \in D(f) \wedge access\_timestamp(x) > t + \varepsilon \wedge creation\_timestamp(x) < t\} \\
Di_t(f) &= \{x | x \in D(f) \wedge creation\_timestamp(x) \geq t\} \ .
\end{aligned}$$

ε should be somewhat greater than the expected duration of copying; a good initial value is 1000 seconds.

We define a metric $Cluster_t(f)$, indicating the relative size of the cutoff cluster, and thus the likelihood that folder f was copied on time t, as follows:

$$Cluster_t(f) = \begin{cases} 0, & \text{if } |Db_t(f)| > 0 \\ |De_t(f)| / (|De_t(f)| + |Da_t(f)|), & \text{otherwise.} \end{cases}$$

This metric ranges between 0 and 1, indicating the size of the cutoff cluster, relative to the maximum size theoretically possible.

We furthermore define $Mag_t(f)$, indicating the sample size, and thus the confidence of $Cluster_t(f)$, as follows:

$$Mag_t(f) = \begin{cases} \infty, & \text{if } |Db_t(f)| > 0 \\ |De_t(f)| + |Da_t(f)|, & \text{otherwise.} \end{cases}$$

$Mag_t(f)$ is on a nominal scale, and is defined to be infinite when $Cluster_t$ is zero. $Mag_t(f)$ can be interpreted as: *the more subfolders f has, the larger our sample, and the more confident we are.*

We can define a second confidence metric as follows. Given a set S of folders, let $Files(S)$ be the set of all files contained in any folder in S. We define a confidence metric $|Abn_t(f)|$, where

$$Abn_t(f) = \{x | x \in Files(D(f)) \wedge access\_timestamp(x) < t - \delta\}.$$

$|Abn_t(f)|$ can be interpreted as *if a large number of files in f have access timestamps less than t, while no subfolders do, we become very confident that f was indeed copied.* High values of $|Abn_t(f)|$ give great confidence in $Cluster_t(f)$, because they show that the historical file activity is too sparse to have created a cutoff cluster by chance. A good value for δ is 10 days; it should be large enough to distinguish the time of the alleged copying from prior historical behavior. Note that $|Abn_t(f)|$ is only applicable to Windows NTFS and similar systems that do not update file access timestamp on copying (see Section 2 above). On systems such as Linux ext3 which update the file access timestamp, we need to substitute $|Abn_t(f')|$, where f' is a folder similar to f that is known to not have been copied.

In short, these metrics quantitatively measure the cutoff cluster: $Cluster_t(f)$ indicates the cutoff cluster's relative size, $Cluster_t(f) \times Mag_t(f)$ its absolute size, and $|Abn_t(f)|$ its abnormality.

6 Field results

We successfully used these metrics as part of an investigation of suspected data theft. At the time of the investigation ($t_{investigation}$), it was suspected that *FolderQ* had been surreptitiously copied during a window 30 - 60 days before $t_{investigation}$. To investigate this, we computed the metrics on several top level folders, for all t in the range $(t_{investigation} - 180 \text{ days}, t_{investigation})$. $Cluster_t(FolderQ)$ was greater than 0.3 at t_1 ($\approx t_{investigation} - 50$ days), with $Mag_t > 5000$ and $|Abn_t| > 50000$, forensically supporting the suspicion. *FolderR* also had a non-zero $Cluster_t$ value at t_2 ($\approx t_{investigation} - 70$ days), which subsequent investigation determined was due to authorized copying. All other folders examined had zero $Cluster_t$ values for all t in the range $(t_{investigation} - 180 \text{ days}, t_{investigation})$ (see Table 2).

	FolderQ	*FolderR*	*FolderS*	*FolderT*	*FolderU*
A priori hypothesis	Suspected of being copied	Not suspected of being copied			
$\lvert D(f)\rvert$	≈ 6000	≈ 7000	≈ 800	≈ 300	≈ 50
Maximum $Cluster_t$	$> \mathbf{0.3}$ (at $t = t_1$)	$> \mathbf{0.9}$ (at $t = t_2$)	0	0	0
Indication	**Copied at $\mathbf{t_1}$**	**Copied at $\mathbf{t_2}$**	Not copied		
Mag_t	> 5000 ($t = t_1$)	> 6000 ($t = t_2$)	∞	∞	∞
$\lvert Abn_t\rvert$	> 50000 ($t = t_1$)	> 20000 ($t = t_2$)	> 1500	> 3000	> 500
Results	Suspicion supported forensically	Subsequent investigation determined this copying was authorized	Not copied		

Table 2: Metrics applied to field investigation. All values are over range $(t_{investigation} - 180 \text{ days}, t_{investigation})$ unless otherwise noted.

We also plotted histograms of the data: *FolderQ* and *FolderR* showed cutoff cluster patterns similar to the simulated *FolderA* shown in Figure 2 above; the other folders did not. Our method thus detected copying occurring approximately 2 months beforehand, and demonstrated the absence of copying for approximately 6 months beforehand.

7 Distinguishing different forms of recursive, nonselective access

These metrics can identify folder copying and distinguish it from routine activity. Besides folder copying, there are other types of recursive, nonselective access, such as searching folders for particular files, scanning them for viruses, or even using the POSIX `ls -lR` command to generate a recursive directory listing. While we have not yet used our method to distinguish between these activities, we're investigating doing so via these fingerprinting characteristics:

- *File access.* Are all, some, or none of the file access timestamps updated? Copying, depending on the system, updates either all files or only folders' (see Section 2 above), whereas virus scanning may update only certain types of files (e.g. executable), and searching typically updates only a subset of files having a common subsequence in their name.
- *Skipped folders and files.* What types of folders and files are skipped? Possibilities include ones beginning with periods, NTFS Alternate Data Streams, NTFS hidden files, NTFS system files, Windows `Thumbs.db`, and OS X `DS_Store`.
- *Tree traversal method.* Is the recursion performed breadth first, depth first, or in another order?
- *Sibling visit order.* What order are siblings visited in? Filesystem order may be the most common, but alphabetical or other orders may be used as well. When a folder contains both files and subfolders, is one accessed before the other?
- *Speed.* At what rate are folders and files accessed? Does the rate depend on the number of entries? On the size of files? It should be noted that a copy command may recursively enumerate all descendants of a subfolder before copying any of them, and so the timestamp updates may happen much faster than the actual copying.

We should note that our informal experience is that system activity, such as Windows Volume Snapshot Service, as well as most (but not all) backup software, does not modify access timestamps, and hence is irrelevant to the above metrics. Likewise, modern file searches, which may use indexes such as Windows Search Services (Microsoft Corporation, 2010b), do not necessarily access the individual folders and files being searched. We have found, however, that graphical shells (such as Microsoft Windows Explorer) automatically access various well known User Profile folders (such as `Documents and Settings\<User Name>`) in ways which we have not fully explored; further research is required before applying these metrics to such folders.

Like any forensic method, cutoff clusters are a component of an investigation, not a replacement. The absence of a cutoff cluster provides strong evidence that copying has not taken place. When a cutoff cluster is found, an investigator will use other means, both digital and human, to investigate its time and circumstances. A cutoff cluster occurring in the middle of a workday, caused by an employee who frequently uses the Unix command line, with no suspicious activity occurring at the time, may most likely be due to a Unix `ls -lR` or the like. The investigator will attempt to confirm this by finding other cutoff clusters occurring regularly in folders used by that employee. In another case, a cluster occurring late at night may prompt examination of the building exit records, which show that an employee who usually leaves at 5 PM stayed late that night for no apparent reason. Further investigation may show that employee had just previously expressed anger at a poor performance review. This may prompt a forensic examination of that employee's PC, revealing further information as the investigation progresses.

8 Future work

We are exploring the following improvements to our method:

- We would like to perform scientific trials of our method, evaluating how accurately it detects copying in a blinded test on real world filesystems (see Garfinkel *et al.*, 2009). However, benchmarking this requires external a priori knowledge of if and when copying took place, which the standard corpora (such as DigitalCorpora.org) lack. In general, this problem often makes such corpora, without their accompanying histories, unsuitable for scientifically testing real world phenomena.

- Our metrics are currently on a nominal scale. We'd working to incorporate a model of expected routine filesystem activity to yield a true probability value.

- Our metric currently assumes that folders can be represented as a tree. Many real world filesystems cannot be represented as a tree, because they allow a folder to have multiple parents, such as through symbolic links or Windows 7 Libraries (Kiriaty, 2009; Microsoft Corporation, 2010c). We feel our metric could be extended to these as well.

- Before computing the metrics, certain folders which may be omitted from copying (such as hidden or permission restricted folders) must be prefiltered from $D(f)$. Since these folders are relatively rare, we feel we could dispense with the need to manually prefilter by using a fuzzy threshold for Db_t, instead of a steep cutoff at zero.

- Finally, we think there are other activities that, although they may fail to deterministically create identifiably unique artifacts, nonetheless result in stochastically emergent patterns. We feel that stochastic forensics may enable investigation of these otherwise silent activities.

9 Experimenting with access timestamps

In the course of our experiments, we've found access timestamp behavior to be quite mercurial. Here are the experimental pitfalls we encountered and solutions.

- *Systems may, for performance reasons, decline to update an access timestamp.* Since maintaining accurate access timestamps may involve substantial performance costs, and isn't deemed system critical, systems may decline to update them. In many systems, this is user configurable (Microsoft Corporation, 2003a). In particular, some versions of Microsoft Windows ship configured to disable access timestamp updates (Carvey, 2009, p. 205). Complicating things further, some systems may selectively update the timestamps, for instance updating only when the newer timestamp differs from the previous one by a certain threshold.

  The recommended solution is to check system configuration and documentation before experimenting, and to exhaustively observe system behavior under different scenarios.

- *Systems may, for performance reasons, defer writing updates of access timestamps to the filesystem.* Even when filesystems do maintain accurate access timestamps, they may cache the updates in memory before writing them to a disk (Microsoft Corporation, 2003b). Thus, if a filesystem is examined before a system has been shutdown properly, its access timestamps may not be accurate.

- *Systems may report updated access timestamps even before writing them to disk.* In cases when updates are deferred, queries to the system for access time may return the updated value stored in memory, even though it is has not yet been written to disk. Thus, an experimenter may find one value if he queries the operating system, and another value if he directly examines the filesystem.

- *Querying a file's access timestamp may itself update it.* For instance, we have found that using Windows Explorer to display a file's access timestamp will cause the access timestamp to be updated to the current time.

These last three problems can be solved by not using the standard operating system facilities to query access time, but instead shutting the operating system down normally and then directly examining the filesystem image using specialized tools. Admittedly, this makes experimentation cumbersome.

10 Conclusions

As noted, copying of data has no known artifacts. Nonetheless, we can reliably detect *emergent patterns* unique to copying, even months after its occurrence. Statistical mechanics, which treats objects as individually unpredictable and looks for patterns which nonetheless emerge stochastically, gives us insight beyond the classical laws from which it derives. Similarly, we believe stochastic forensics provides us with means to analyze hitherto undetectable activity.

11 Acknowledgments

The author would like to thank Dan Farmer, Wietse Venema, Matt Bishop, and Igor Mandel for their encouragement and suggestions.

References

Carvey, Harlan. 2009. *Windows Forensic Analysis DVD Toolkit, Second Edition*. Syngress Publishing.

Carvey, Harlan, & Altheide, Cory. 2005. Tracking USB storage: Analysis of Windows artifacts generated by USB storage devices. *Digital Investigation*, **2**(2), 94 – 100.

Casey, Eoghan. 2004. *Digital Evidence and Computer Crime*. Orlando, FL, USA: Academic Press, Inc.

Chow, K. P., Law, Frank Y. W., Kwan, Michael Y. K., & Lai, Pierre K. Y. 2007. The Rules of Time on NTFS File System. *Pages 71–85 of: SADFE '07: Proceedings of the Second International Workshop on Systematic Approaches to Digital Forensic Engineering*. Washington, DC, USA: IEEE Computer Society.

CSI, & FBI. 2003. *2003 Computer Crime and Security Survey*. Tech. rept.

Farmer, Dan. 2000. What Are MACtimes? *Dr. Dobb's Journal of Software Tools*, **25**(10), 68, 70–74.

Farmer, Dan, & Venema, Wietse. 2004. *Forensic Discovery*. Addison Wesley Professional.

Ferguson, Michael. 2002. *File System Numbers*. `http://www.cs.cornell.edu/courses/cs614/2002sp/File%20System%20Numbers.ppt`. Accessed 2010-03-09.

Free Software Foundation, Inc. 2010. *GNU `coreutils` implementation*. `http://ftp.gnu.org/gnu/coreutils/coreutils-8.4.tar.gz`. Accessed 2010-03-09.

Garfinkel, Simson, Farrell, Paul, Roussev, Vassil, & Dinolt, George. 2009 (August). Bringing science to digital forensics with standardized forensic corpora. *In: Proc. 9th Annual Digital Forensic Research Workshop (DFRWS).*

Gribble, Steven D., Manku, Gurmeet Singh, Roselli, Drew, Brewer, Eric A., Gibson, Timothy J., & Miller, Ethan L. 1998. Self-similarity in file systems. *SIGMETRICS Perform. Eval. Rev.*, **26**(1), 141–150.

Hillstrom, Kevin, & Hillstrom, Laurie Collier. 2002. Employee Theft. *In:* Hillstrom, Kevin, & Hillstrom, Laurie Collier (eds), *Gale Encyclopedia of Small Business*, 2nd edn. Farmington Hills, MI, USA: Gale Group/Thomson Learning.

Kiriaty, Yochay. 2009. *Understanding Windows 7 Libraries*. `http://windowsteamblog.com/blogs/developers/archive/2009/04/06/understanding-windows-7-libraries.aspx`. Accessed 2010-03-09.

Liu, Yali, Corbett, Cherita, Chiang, Ken, Archibald, Rennie, Mukherjee, Biswanath, & Ghosal, Dipak. 2009. SIDD: A Framework for Detecting Sensitive Data Exfiltration by an Insider Attack. *Hawaii International Conference on System Sciences*, **0**, 1–10.

Microsoft Corporation. 2003a. *Windows Server 2003 Resource Kit Registry Reference: `NtfsDisableLastAccessUpdate`*. `http://technet.microsoft.com/en-us/library/cc758569(WS.10).aspx`. Accessed 2010-03-09.

Microsoft Corporation. 2003b. *Windows Server 2003 Technical Reference: How NTFS Works*. `http://technet.microsoft.com/en-us/library/cc781134(WS.10).aspx`. Accessed 2010-03-09.

Microsoft Corporation. 2010a. *Microsoft Developer Network Online Documentation: `CopyFile` Function*. `http://msdn.microsoft.com/en-us/library/aa363851(VS.85).aspx`. Accessed 2010-03-09.

Microsoft Corporation. 2010b. *Microsoft Developer Network: Windows Search Overview*. `http://msdn.microsoft.com/en-us/library/aa965362(v=vs.85).aspx`. Accessed 2011-04-10.

Microsoft Corporation. 2010c. *Windows Developer Center: Learn About Windows 7: Libraries*. `http://msdn.microsoft.com/en-us/windows/ee658250.aspx`. Accessed 2010-03-09.

Sun Microsystems, Inc. 2009a. *Solaris `mv` command implementation*. `http://src.opensolaris.org/source/xref/onnv/onnv-gate/usr/src/cmd/mv/mv.c`. Accessed 2010-03-09.

Sun Microsystems, Inc. 2009b. *Solaris `writefile` implementation*. `http://src.opensolaris.org/source/xref/onnv/onnv-gate/usr/src/lib/libcmdutils/common/writefile.c`. Accessed 2010-03-09.

Venema, Wietse. 2000. File Recovery Techniques. *Dr. Dobb's Journal of Software Tools*, **25**(12), 74, 76–80.

Vogels, Werner. 1999. File system usage in Windows NT 4.0. *Pages 93–109 of: SOSP '99: Proceedings of the seventeenth ACM symposium on Operating systems principles*. New York, NY, USA: ACM.

Wikipedia. 2010. *Pareto distribution*. `http://en.wikipedia.org/wiki/Pareto_distribution`. Accessed 2010-03-09.

Yu, Yang, & Chiueh, Tzi-cker. 2004. Display-only file server: a solution against information theft due to insider attack. *In: Proceedings of the 4th ACM workshop on Digital rights management*. DRM '04. New York, NY, USA: ACM.