Digital Investigation 9 (2012) S50–S59

Contents lists available at SciVerse ScienceDirect

Digital Investigation

journal homepage: www.elsevier.com/locate/diin

A general strategy for differential forensic analysis

Simson Garfinkel [a,*], Alex J. Nelson [b], Joel Young [a]

[a] Computer Science, Naval Postgraduate School, 900 N Glebe St., Arlington, VA 2203, USA
[b] Computer Science, University of California, Santa Cruz, 1156 High St, Santz Cruz, CA 95064, USA

Keywords:
Forensics
Differencing
Forensic strategies
Feature extraction
Temporal analysis

A B S T R A C T

The dramatic growth of storage capacity and network bandwidth is making it increasingly difficult for forensic examiners to report what is present on a piece of subject media. Instead, analysts are focusing on what characteristics of the media have *changed* between two snapshots in time. To date different algorithms have been implemented for performing differential analysis of computer media, memory, digital documents, network traces, and other kinds of digital evidence. This paper presents an abstract differencing strategy and applies it to all of these problem domains. Use of an abstract strategy allows the lessons gleaned in one problem domain to be directly applied to others.

Published by Elsevier Ltd.

1. Introduction

This paper describes *differential forensic analysis*, a practice that is increasingly used by digital forensic examiners but has not been formalized until now.

Differential forensic analysis compares two different digital forensic *images* (or, more generally, any pair of digital artifacts) and reports the differences between them. Focusing on the changes allows the examiner to reduce the amount of information that needs to be examined (by eliminating that which does not change), while simultaneously focusing on the changes that are thought to be the result of a subject's activities (for presumably, it was the activity of the subject that somehow transformed the first digital image into the second).

Differential analysis is widely practiced today. Reverse engineers attempt to infer the behavior of malware by comparing the contents of a hard drive before the malware is introduced with the hard drive captured after the malware infection. Sex offenders on many controlled release programs must submit their computers for regular analysis, so that an examiner can determine if the offender has visited a banned website. Network engineers compare month-to-month traffic summaries in an attempt to learn how demands on their networks evolve, as well as to identify the presence of malware.

No matter what specific modality is being examined, all of these use cases involve the collection of at least two digital objects—a *baseline* and a *final* image. Differential analysis reports the differences between the two—that is, what has changed. But despite the similarity of purpose, to date each differential analysis use case has been developed in isolation, with different procedures, tools and reporting standards.

We show that these scenarios can all be implemented using the same strategy. Furthermore, the strategy can cover scenarios apparently unrelated to computer forensics, such as reporting on the changes within a document or even file synchronization. The key to this strategy is the extraction of *features* from the digital artifacts in which each feature has a separately describable *name*, *location*, *content*, and possibly other metadata.

1.1. Contributions

This paper presents a principled study of differential analysis and then applies that work to multiple contexts, including the analysis of files on a computer's disk drive, the pattern of data sent across a network, and even reports

* Corresponding author. Tel.: +1 617 876 6111.
E-mail address: simsong@acm.org (S. Garfinkel).

1742-2876/$ – see front matter Published by Elsevier Ltd.
doi:10.1016/j.diin.2012.05.003

from other forensic tools. We show that a small set of well-chosen abstractions allows the same differential analysis strategy to be applied to all of these cases.

It is important to note that the tools we have written were created *before* we formalized our general strategy, not after. Although it would be quite elegant to have a single implementation of differential analysis and then to specialize that implementation for each modality, what actually happened is that we unwittingly wrote multiple implementations of the same abstract strategy each time we wrote another differential analysis program. Only after writing several different differential analysis tools were we able to appreciate the commonalities between the implementations and to realize that the strategy could be made general by an appropriate choice of abstraction.

2. Definitions, terminology and notation

In this section we introduce a consistent terminology for discussing differential analysis. We apply this terminology to prior work as well as to our own contributions.

Differential analysis. An analytical process that compares two objects (images) A and B and reports the differences between them. Although at first it might seem most sensible to report the differences as $(B - A)$, experience has shown that it is frequently more useful to report the differences as the series of operations R necessary to transform A into B:

$$A \underset{R}{\rightarrow} B \tag{1}$$

Typically A and B represent snapshots in time—A might be an image of a hard drive recorded before a computer is deployed, and B might be an image of the same drive after it has been compromised by an attacker. However both A and B might be two different systems that are based on a common object א:

$$\begin{array}{ccc} & \aleph & \\ \swarrow & & \searrow \\ A & \underset{R}{\rightarrow} & B \end{array} \tag{2}$$

Typically the operations R that are reported are a function of both the data formats and the needs of the examiner. If A and B are disk images and the examiner is evaluating the installation footprint of a new application, then R might be a list of files and registry entries that are created or changed. But if the examiner is looking for evidence of a malware infection, R might be a list of opcodes that are changed in existing executables.

Image. A byte stream from any data-carrying device representing the object under analysis. Practitioners will be familiar with *disk images*, *memory images* and *cell phone images*. Images may be *physical*, which can be thought of as a collection of sectors, or *logical*, which can be thought of as a collection of files.

Note that for the purposes of this abstract strategy the only real difference between a sector and a file is that sectors are constant-length collections of bytes and are identified by a number (which can be referred to either as a name or a location, depending on the context), while files are collections of bytes of variable length that are identified by strings (typically a path name consisting of one or more directory names and a final file name).

We use the term *image* to refer to any kind of digital artifact. In this article we occasionally use the word *object* as a synonym for image when warranted by context.

Baseline image (A). The image first acquired at time T_A.

Final image (B). The last acquired image, taken at time T_B.

Intermediary images (I_n). Zero or more images recorded between the baseline and final images. Image I_n is the nth image acquired.

Common baseline. A single image that is a common ancestor to multiple final images. For example, א in Equation (2) is a common baseline for A and B.

Image delta ($B - A$). The differences between two images, typically between the baseline image and the final image.

Differencing strategy. A strategy for reporting the differences between two or more images.

Differencing strategies and algorithms that implement those strategies have long been applied to programs, text, and word processing files (Horwitz, 1990), and are widely available in tools such as Unix diff and Microsoft Word. Traditionally there has been little distinction between the tool that implements the algorithm and the algorithm itself, and both have been developed for specific differencing tasks.

This paper presents a *general* strategy for differential analysis. By general we mean that the strategy can be equally applied to other articles of forensic interest, such as memory images and network packet dumps. For example, if A and B are collections of packets sent over a network on two successive days, the examiner might be interested in an R that describes changes to metadata describing network flows—for example, that a web server that was previously listening on one IP address and port was moved to another location, or that a protocol that was previously protected with SSL is no longer using encryption. On the other hand, a differential analysis of a Microsoft Word document at two points in time might report that some paragraphs have been changed while others have been moved—an analysis performed by Word's "Compare Documents" feature, or the Unix command-line diff utility on text files.

Feature (f). A piece of data or information that is either explicitly extracted from the image or otherwise dependent upon data within the image. For example, an email address from an address book, a URL from a browser cache, the hash value of a sector, and a histogram of port frequency use within a set of packets are all examples of features.

Feature in image ((A, f)). Features typically are found in images. In this case, feature f is found in image A.

Feature name (NAME (A, f)). Every feature may have zero, one or multiple names. For example, if a feature is the contents of a file, the feature name might be the file name.

Feature location (LOC (f)). The location in the image in which the feature is found. If the feature is the contents of a file, the address might be the sectors where the file resides. Some features can have both *logical* and *physical* addresses. For example, a file's location may be either its file name (which is the file's logical address in the file

system), its inode number, or a set of sector numbers (which are the file's physical address on the media). A feature might thus have multiple locations.

Feature extraction ($F()$). The process of deriving one or more features from bulk data. Differential analysis is rarely applied on a byte-for-byte basis to bulk data. While it is certainly possible to compare two objects byte-for-byte and report where the bytes differ, it is more useful to transform the data through some kind of grouping, data reduction, or feature identification step.

In order to manage complexity, differencing algorithms extract features as atomic units of one or more bytes, and report differences between A and B as differences in their extracted features. For example, the Unix diff program first extracts lines from the files being compared. Many systems that perform differencing on files in a forensic context use the cryptographic hash of a long sequence of bytes as a feature.

Extract feature set $F(A)$. The set of features extracted from an image. For example, the Unix diff program treats each file as an ordered set of extracted lines.

Feature set delta ($F(B) - F(A)$). The differences between the feature sets extracted from two images. The delta is not necessarily symmetric.

Transformation set (R). The specific set of operations that are applied to A to produce B. For example, the Unix diff program can generate a "patch file" that is a set of changes which, when applied to the first file, will produce the second. This can be indicated with the notation shown in Equation (2).

2.1. Processing multiple images

Clearly, any sequence of differential analyses performed on a baseline image, one or more intermediary images, and a final image can be reduced to a set of pairwise comparisons between an image I_n and image I_{n+1}. For this article we shall therefore only consider differences between pairs of images.

3. Prior work

Although differential analysis is practiced by a growing number of forensic examiners, there is very little that has been published on the subject in the forensic context. For this reason we expand the prior work section to include related concepts from version control systems and file synchronization systems.

3.1. Differential analysis with diff

Differential analysis of text and binary objects has a long history in computer science dating back to the original Unix diff command (Thompson and Ritchie, 1975), the algorithm for which was described by Hunt and McIlroy (1976). Of particular note for this article, the original diff command could generate an *edit script* that will transform the first file being compared into the second using the Unix *ed* editor.

Modern versions of the diff command implement variants of the Myers Difference Algorithm (frequently called simply the *diff algorithm*) which operates by finding the longest common substrings in a set of symbols, where each symbol is typically taken to be a line of text (Myers, 1986).

It is possible to use the diff command for forensic differential analysis. The examiner must first prepare for each snapshot a text file containing the name of each allocated file, that file's time stamp, size and hash value. The two files are then compared with diff. This is the essence of Tripwire (Kim and Spafford, 1994). Today this operation can be performed simply using diff with the output of md5deep (Kornblum, 2011) or SleuthKit's fls command (Carrier, 2005). The main limitation of this approach is that it tends to over-report differences. For example, a file that is renamed will be reported as one file being deleted and a new file with the old file's modification date, size and hash value being created elsewhere in the file system.

3.2. Manual forensic differencing

Examiners can use existing forensic tools such as EnCase (Guidance Software, Inc., 2011) and FTK (Access Data, 2011) to perform forensic differencing, but as these tools have no support for automated differencing, the differencing must be manually performed, or automated with macros.

For example, using EnCase it is possible to compare the file systems on two drives by computing the hash values for the files of one drive and then using this as a filter to view the files on the second drive. New files can be found by filtering for files on the second drive that do not have hashes in the file set from the first drive; files that have been renamed are those for which the hash value is the same but the file name has been changed.

Levy (2012) has developed an EnScript that analyzes two disk drives and reports files on the second that are not on the first. This is done using file hash values as an EnScript filter.

3.3. Automated forensic differencing

White (2008) presented a system at AAFS2008 for identifying the changes to a Windows registry resulting from the installation of a program. The system consists of a program to ingest two Microsoft RegEdit-generated Registry patch files and produce an XML file containing the differences between the two files, and a second program using that XML file to create a Windows Registry Dataset (WiReD) file containing "changes to the Registry caused by application installation, de-installation, execution or other modifying operations." The NIST (2009) follow-up publication indicated that the WiReD dataset is "based on an FBI database schema."

Watkins et al. (2009) described a system using a combination of differencing and similarity matching to determine the minimal change set necessary to transport a copy of a disk image from one location to another. The *Teleporter* uses a shared database of files and disk sectors at both the sending and receiving location as a code book and, when possible, sends these codes rather than the entire disk sectors. Using our notation, Teleporter treats the disk sent as B, assumes that A is null, and creates R with reference to the code book.

3.4. Differencing outside of forensics

Others have applied differential analysis to a variety of circumstances. Apiwattanapong et al. (2004) describe an algorithm for differencing object-oriented programs; Lindholm et al. (2006) present a simple XML tree differencing algorithm, while Duley et al. (2010) provide an algorithm for differencing Verilog HDL. More recently, Loh and Kim (2010) introduce *LSdiff*, "a program differencing tool to identify systematic structural differences."

Many file formats, including network configuration files, are not readily parsed with regular expression based tools such as diff. Weaver and Smith (2011) are building a context-free analog to the classic grep and hierarchical differencing tools able to better search structured documents. Weaver et al. (2011) focused on IGTF PKI and Cisco IOS policies. Their strategy was to parse the files into policy trees, develop change tables summarizing the trees, and then mine the tables for reporting.

3.5. Data synchronization

Data synchronizers are programs that manage pairs of datasets such that changes to a *master* file system are reflected in the *replica*. In practice there are two approaches to implementing data synchronization. The first approach is to employ differencing: deltas between the master and replica are computed, such that the differences contain sufficient detail to modify the replica to make it a copy of the master. The second approach is to start with two identical databases, track changes to the master, and replay those changes to the replica.

The popular rsync (Tridgell and Mackerras, 1996) file system replication tool implements differencing-based synchronization, while MySQL database replication implements change-tracking synchronization. (In the case of MySQL synchronization, the changes are data-modifying SQL statements that are sent from the master database to the replicants.) Unison (Pierce and Vouillon, 2004) uses differential analysis to create both the initial sync and a snapshot of each system's metadata. The file system and snapshot are then compared to create a list of change deltas that are used to accelerate its change-based synchronization scheme.

3.6. Revision control systems

Closely related to the problem of data synchronization is the problem of keeping multiple copies of source code repositories consistent. Two broad strategies have emerged for revision control. These are centralized (e.g., RCS (Tichy, 1982), Subversion (Pilato et al., 2008)) and decentralized (e.g., git (Grosenbach, 2007) and Darcs (Roundy, 2005)). In centralized systems, a master repository maintains a full history of all changes and folds in changes on command from subordinate (users') copies. In decentralized systems, there is no master repository. Instead each user keeps their own copy of the repository and changes are merged as needed from other repositories. Jacobson (2009) has partially formalized the merging process for one of these systems (darcs) using group theory over compositions of invertible functions. Each function represents a particular change needed for the merge.

3.7. Timeline analysis

In a forensic context, differential analysis often implicitly involves the analysis of timestamps as well.

Marrington et al. (2011) developed a system called CAT Detect that used a rule-based system to detect events in the computer activity timeline—for example, a user that deletes a file before it is created. Although the system does not use differential analysis, it could be readily extended to do so.

Olsson and Boldt (2009) presented a tool for the visual display of timelines. Such a tool could be combined with a differential analysis tool to display only changes between two snapshots, rather than all time elements within an image. By displaying the results of differential analysis, temporal inconsistencies (Section 5.3) would be readily apparent as activity outside the region between the two snapshots.

Schatz et al. (2006) discussed an approach for correlating multiple timestamps on a computer for the purpose of validating timestamps.

4. Differencing in digital forensics

Unlike synchronization and revision control uses, forensic analysts frequently do not use differential reporting to create byte identical replicas of forensic case materials. Instead, analysts use differencing to understand the processes that resulted in the changes. For example, when analyzing a baseline network flow with one captured during an intrusion, the analyst may not care about the actual content. Indeed, if the actual content is encrypted, the analyst may not even be able to view it! Instead the analyst may look for differences between the destinations, ports, or average packet size of the connections. A carefully designed system that extracts and processes the specific features of interest can dramatically reduce the amount of data that needs to be processed, reducing processing and storage costs. Intelligent feature extraction also makes it easier for the developer to create reports that focus on features of interest to the analyst, rather than reports that are optimized for ease of data processing.

Existing differential analysis and synchronization systems carefully select the features that they extract and compare. The key insight in describing those systems with our strategy was the realization that decisions regarding which features to consider are frequently inherent in the implementations that we have created. For example:

- Unison does not synchronize deleted files or file fragmentation patterns as such "features" are invisible to the POSIX file API. Instead, it obtains a list of features by walking the file system and consulting a local database.
- Subversion does not (by default) synchronize file modification dates, because keeping such dates synchronized is not relevant to the system's original purpose of source code control. Subversion does synchronize file names, content, MIME type, and

whether or not files are executable; all of which are relevant to source code control.

Clearly, even existing systems distinguish changes that matter from those that do not. This is done through feature selection and extraction.

4.1. Use cases

This section presents three cases in which a forensic analyst might wish to determine the difference between disks in time or space: malware discovery and analysis; insider threat identification; and pattern of life analysis.

4.1.1. Malware discovery and analysis

Malware discovery and identification is the process of examining the contents of memory to determine if malicious code or files exist on the device. Disk differencing in time and space can quickly identify malicious files by analyzing registry hives, file additions and deletions, and abnormal file size growth. This process will not identify the most advanced or targeted malware, but it will uncover many instances.

4.1.2. Insider threat identification

Employee malpractice and espionage is a growing concern among corporations as well as governmental agencies. Automated techniques in disk differencing on forensic images can identify and verify harassment cases, policy-use violations, theft (intellectual property, money, identities), and a variety of other malpractices. Several issues exist in identifying or gathering evidence in insider threat cases. One problem is that forensic images are typically gathered only after misuse has already occurred, making it difficult to establish the baseline. Additionally, forensic images are usually not gathered right after the event immediately occurs. Instead, the disk images are produced days or months after the malicious activity has already taken place. This can make finding the evidence difficult or impossible.

By using disk differencing techniques one can identify the evidence quicker by identifying abnormalities in time and space. These automated techniques can correlate information from the case to the base image and final image to reduce the final image's size to only the pertinent information.

4.1.3. Pattern of life

To analyze an individual's "pattern of life" is to learn the individual's habits regarding work, play, meals, sleep, and other regular activity. Understanding routine behaviors can be useful for a variety of activities involving data collection or intervention.

Differential analysis can be a powerful tool for determining an individual's pattern of life for the simple reason that computer systems are intimately involved in so many habitual activities. Differential analysis can also identify if a computer is used by one or more people, if a single person is using multiple accounts, or even if an account has been hijacked. Differential analysis is useful in all of these cases because it can be used to create a set of deltas that occur over time. Such deltas can be readily assembled into profiles using appropriate statistical techniques.

4.2. Summarized reporting

Although it is relatively easy to simply calculate and report $F(B) - F(A)$ (that is, the list of all features that are different between the two images), such lists tend to be extraordinarily long and relatively difficult to interpret.

Based on our analysis of above use cases, combined with interviews conducted with users of differential reports, we have concluded that there is a need for a range of higher-level differential reports. These include:

- The introduction of new features. This might be the creation of new files or the appearance of a new email addresses in a document.
- The increase in count of an existing feature.
- The decrease in count of an existing feature.
- The removal of a feature from the image.
- A feature that is relocated from one location to another.

Frequently it is possible to suppress information in a report that is not of interest to an analyst. For example, analysts that are interested in finding a subject's email correspondents are frequently interested in new email addresses but not in the disappearance of email addresses: a new email address indicates a new connection, but the disappearance of an email address may simply indicate that a file was deleted and the sectors overwritten by another file.

5. A general strategy for differential analysis

This section presents our strategy. The algorithm operates by augmenting each feature with *metadata* and then determining the set of operations necessary to transform one set of extracted features into another.

5.1. Feature metadata

In addition to a sequence of bytes, each feature has the following associated metadata:

Location (mandatory): Each feature must have at least one location. There are few restrictions on the nature or form of the location. Features and locations have a one-to-many relationship, such that different features necessarily have different locations, but a single feature might have multiple locations if copies of it are present at multiple locations. Differential analysis algorithms can be dramatically more efficient if locations are sorted and then searched using a binary search algorithm. LOC(A, f) is a conceptual function that returns all of the locations of feature f in object A. In practice this function is implemented by searching A for all features and arranging them in a reverse index.

Name (optional): In many cases, it is useful for features to have names so that they can be referred to in the human-readable difference report. NAME (A, f) is a conceptual function that returns all of the names for feature f in object A.

Timestamp(s) and other metadata (optional): Features can also have one or more timestamps. Typically timestamps are extracted from the features themselves, although they may be extracted from elsewhere in the target media. Most obviously timestamps are useful for constructing timelines, but they can also be used to find temporal inconsistencies (Section 5.3). Beyond timestamps, features could have additional metadata as well.

5.2. Change primitives

Images A and B can be viewed as sets of features, $F(A)$ and $F(B)$. Differential analysis can then be reduced to identifying the specific operations for converting feature set $F(A)$ into $F(B)$. These operations are called *change primitives*. Change primitives operate on features, not images, and as such, do not explain image deltas but rather feature set deltas.

Unlike for revision control systems like Darcs that need to be able to revert changes, for forensic differencing the change primitives do not need to be invertible.

With respect to any feature f, the change primitive needed to transform $(A,f)\rightarrow(B,f)$ can be identified using the rules described in text in Table 1 and in set notation in Table 2. The full set of primitives can be efficiently calculated by enumerating all features, feature names, and feature locations in both objects, removing the features that are unchanged between A and B, identifying the features created and deleted, and finally identifying features that have moved, been renamed, had names added, and had names deleted.

5.3. Temporal inconsistencies

If A and B are from the same system and $T_B > T_A$, one would expect all new features in the feature set delta $F(B) - F(A)$ to be timestamped after T_A. If B contains features that predate T_A or postdate T_B, then there is an inconsistency. It is useful for analysts to identify and explain temporal inconsistencies when performing differential analysis:

1. Inconsistencies most obviously arise from tampering of some type—either evidence tampering (for example, the planting of new files with old file stamps), or system tampering (for example, changing the system's clock to a previous time) (Marrington et al., 2011).
2. Inconsistencies can be a result of unexpected system operation (for example, the Unix cp command will preserve the *mtime* (modification date) of a file that is copied, but the file's *ctime* (inode change time) will necessarily reflect the time that the copy is made (Garfinkel et al., 2010).
3. Inconsistencies may be in inherent in the manner that systems track time. For example, Nelson (in press) found that for Microsoft Windows Registry hives, the last-updated key, the hive header, and the hive file could have inconsistent mtimes, while Garfinkel and Rowe (2010) found Windows rounds many times to the hour.
4. Finally, inconsistencies can result from tool error.

Table 1
Change detection rules in English.

• If something did not exist and now it does, it was created
• If it did exist before and now it does not, it was deleted
• If it is in a new location, it was moved
• If more copies of it exist, it was copied
• If less copies of it exist, something got deleted
• Aliasing means names can be added or deleted

Table 2
Abstract Rules for transforming $A \rightarrow B$ (A into B) based on observed changes to features (f), feature locations (LOC (A, f)), and feature names (NAME (A, f)). Although the RENAME primitive is not strictly needed (it can be implemented with a ADDNAME and a DELNAME), it is useful to distinguish the two operations.

Rule	Change primitive for $A \underset{R}{\rightarrow} B$
$f \in F(A)$ and $f \in F(B)$	(No change)
$f \in F(A)$ and $f \notin F(B)$	DELETE f
$f \notin F(A)$ and $f \in F(B)$	CREATE f
$\lvert\text{LOC}(A,f)\rvert = 1$ and $\lvert\text{LOC}(B,f)\rvert = 1$ and $\text{LOC}(A,f) \neq \text{LOC}(B,f)$	MOVE $\text{LOC}(A,f) \rightarrow \text{LOC}(B,f)$
$\lvert\text{LOC}(A,f)\rvert < \lvert\text{LOC}(B,f)\rvert$	COPY $\text{LOC}(A,f) \rightarrow (\text{LOC}(B,f) \backslash \text{LOC}(A,f))$
$\lvert\text{LOC}(A,f)\rvert > \lvert\text{LOC}(B,f)\rvert$	DELETE $(\text{LOC}(A,f) \backslash \text{LOC}(B,f))$
$\lvert\text{NAME}(A,f)\rvert = 1$ and $\lvert\text{NAME}(B,f)\rvert = 1$ and $\text{NAME}(A,f) \neq \text{NAME}(B,f)$	RENAME $\text{NAME}(A,f) \rightarrow \text{NAME}(B,f)$
($\lvert\text{NAME}(A,f)\rvert \neq 1$ or $\lvert\text{NAME}(B,f)\rvert \neq 1$) and $n \notin \text{NAME}(A,f)$ and $n \in \text{NAME}(B,f)$	ADDNAME f, n
($\lvert\text{NAME}(A,f)\rvert \neq 1$ or $\lvert\text{NAME}(B,f)\rvert \neq 1$) and $n \in \text{NAME}(A,f)$ and $n \notin \text{NAME}(B,f)$	DELNAME f, n

5.4. Reporting

Once the features have been extracted, the change primitives enumerated, and temporal inconsistencies evaluated, the strategy enters the *reporting* stage. This stage has two primary missions: First to suppress irrelevant information and second, to emphasize the important differences.

As discussed above, source code control systems detect differences and propagate them to all clients so that all have identical copies of the same source code once local modifications are taken into account. That is, it requires that R be sufficiently rich to allow B to be generated from A.

Forensic examiners rarely have such a stringent requirement for differential analysis. Being already in possession of both A and B, their goal, instead, is to determine what has changed at the appropriate level of detail to enable their objectives (Section 4.1). Thus forensic differential analysis frequently requires that extraneous information be suppressed.

As previously noted (Section 4.2), the most straightforward way to suppress extraneous information is by simply not extracting unwanted features. But some kinds of

suppression (such as reporting increases in feature counts but not decreases) can only be done in the reporting stage. Some techniques are:

1. Present count statistics rather than the actual features. For example, a report differencing two network streams may only present the number of common IP address/port pairs rather than enumerating all of them.
2. Organize the features in hierarchies and allow the viewer to drill-down into the level of detail needed
3. Organize the features into timelines.

6. Tools we have written

In this section we present tools that implement variants of the general algorithm.

6.1. idifference—differences between two different disk images

One of the most basic differential analysis tasks is to compare two disk images and report the files that have been added, deleted, renamed, and altered. This is the basic functionality that Garfinkel (2009) introduced in the *idifference.py* program. However the program's initial implementation had subtle bugs that only became evident when we attempted to explain its behavior using the algorithm described in this article.

Our current *idifference.py* program is based upon Garfinkel (2012)'s DFXML toolset. The program reads a DFXML file associated for each disk image. (If a DFXML file is not available, the *fiwalk* program is run as a subprocess to produce a DFXML stream which is processed instead.) Each DFXML file contains a <fileobject> XML block for each allocated, deleted or orphaned file. These XML blocks are used to create Python *fileobject* objects. Each object can be queried for the corresponding file's length, contents, the hash of the contents, and metadata such as file modification time. A SAX-based framework makes it relatively easy to write Python programs that ingest and process XML files that are gigabytes in size.

This basic disk differencing implementation maintains two data structures for each disk image:

- *fnames[]*, a Python dictionary that maps complete path names to file objects.
- *inodes[]*, a Python dictionary that maps the *(partition, inode #)* pair to a unique file object.

These dictionaries reside in an instance of the *DiskState* class. When a disk image *B* is processed, the *DiskState* instance has the *fnames[]*, and *inodes[]* dictionaries associated with image *A*. Then for each file object *fi*, the following operations are performed:

1. If *fi* is not allocated, it is ignored. (This program only reports differences of allocated files.)
2. The program retrieves the file object associated with *fi's* file name from *A*. This can be calculated with: ofi = f-names[fi.filename()]
3. If there is no entry in the fnames[] array for the name fi.filename(), the file is new.
4. If ofi.sha1()!= fi.sha1(), then the file's contents have changed.
5. if ofi.mtime()!= fi.mtime(), the file's modification time has changed.
6. The fi file object is removed from fnames[].
7. Finally, the file object is added to the new_sha1s[], new_fnames[], and new_inodes[] dictionaries, which will represent the version of the disk image at time T_n when the disk image associated with time T_{n+1} is processed.

After all of the files in *B* are processed:

1. The fnames[] array contains a list of the file names that were present in *A* but not in *B*. These file names are reported as a list of files that were deleted.
2. inodes[] is set to new_inodes[]
3. fnames[] is set to new_fnames[]

This program views inodes as the features that are extracted from the disk image. Here the "name" of the feature is the path name. A change of the feature's name is a rename event, which strictly corresponds to a file that has been renamed.

6.2. rdifference—differences between two registry hives

We have also tailored a version of *idifference.py*, to describe the difference between two Windows Registry hive files. The new tool is called *rdifference.py*. This is possible because the Registry behaves much like a simple file system. The Registry as a whole exposes a hierarchical namespace, and each hive is "mounted" like a child file system at fixed points. For example, the system hive file mounts at HKEY_LOCAL_MACHINE\SYSTEM. Nelson (in press) designed an XML format, RegXML, to represent hives, and implemented a RegXML processing interface in the DFXML toolset. The same SAX model that supports *idifference.py* has an interface for hive *cells* similar to file-system inodes.

Several distinctions relevant to differencing exist between hives and more-complete file systems. File-system "Directories" and "Files" have analogous structures in hives, "Key" and "Value" cells respectively. Values have one feature file metadata lack: Content is explicitly *typed*, for instance as UTF-16 characters or binary. However, most file system metadata does not exist in hives:

- There are only mtimes; there are no access, creation, or change times.
- Keys and the hive header have mtimes, but values do not.
- There is no notion of an "inode" as a data structure always separate from cell content.

The general algorithm of *rdifference.py* is a subset of *idifference.py*. We presently omit rename detection, as cells have only their paths as unique identifiers. (It may be possible to measure subtree similarity to infer renames, but we leave that to future work.) The program presently reports:

- New and deleted hive cells, including cells that have precisely matching full paths (i.e., a cell being fully duplicated, including name).
- Deleted cells.
- Values with modified content or type.
- Keys with changed mtimes (from which one can infer a changed value's mtime).

It is important for tools to be cautious about assuming a unique path for each object in a file system analysis: Nelson (in press) observed hives that contained cells with non-unique paths. That is, some key in the hive had at least two children with completely matching names. This may cause a tool that walks a file system to fail in some manner, as path names are normally unambiguous. Hence, it is important to identify path components during processing by a characteristic that must be unique, such as a byte offset. These corner-case hives came from a research corpus of drives (Garfinkel et al., 2009) used and discarded by people around the world.

6.3. bulk_diff.py

The *bulk_diff* program compares histograms from two runs of the *bulk_extractor* program (Beverly et al., 2011) and reports on the differences between them. Written to address the real-world needs of forensic analysts, this program reports email addresses, URLs, IP addresses, domain names, and other information that are present in *B* that are not present in *A*.

The *bulk_diff* program was written to detect new activity on the part of a computer system subject to monitoring. For example, in the M57-Patents scenario (Garfinkel et al., 2009), five computers belonging to a simulated small company had their hard drives imaged every day for three weeks. The *bulk_diff* program makes it possible to rapidly infer what happened on any day n by running *bulk_extractor* on I_n and on I_{n+1} and comparing the results. URLs, email addresses, and other features found on I_{n+1} that are not on I_n necessarily correspond to activity that took place on day n. Features present on I_n that are not present on I_{n+1} have little to no probative value, as they merely correspond to files that were overwritten during the course of operation on day n. (While one might argue that the sudden drop of many features may correspond to activities on day n intended to remove evidence, a drop might just as easily correspond to an increase in legitimate activity.)

6.4. corpus_sync.py

Previously we noted that the problem of synchronizing two file systems is a special case of the differential analysis problem. This was an important realization in our group that has allowed us to work with our multi-terabyte forensic corpora in a more efficient manner.

Most file synchronization programs in use today (rsync and Unison), are designed for synchronization over a network and assume IP connectivity during the synchronization process. We are not aware of an offline file synchronization system—for example, a system that would allow synchronization using data carried on terabyte hard drives.

We therefore created a program called *corpus_sync* that synchronizes two file systems using a DFXML file that describes *B*, the master file system, a second DFXML file that describes *A*, the file system to be brought into agreement with *B*, and *DB*, a database of additional materials that can be retrieved by hash value (this database would presumably be placed on removable media and hand carried or shipped from *B* to *A*).

The *corpus_sync* program implements the abstract rules in Table 2 but it orders them and adds an additional step to handle the case where *file1* is renamed to be *file2* while *file2* is renamed to be *file1*. The algorithm is thus:

1. Read the DFXML files for *A* and *B* into memory.
2. Every file afi whose hash value is not in *B* is removed from the file system and from *A*.
3. For every file bfi in *B*:
   (a) If bfi's filename is in *A* and bfi's hash value is the same as the file in *A* with the same name, ignore bfi.
   (b) If bfi's hash value is not in *A*, *get* bfi from *DB*.
   (c) Let afi be a file in *A* that has the same hash value as bfi
   (d) Add the name bfi.filename()+ '.new' to afi.
4. For every file *filename*+.new that was created, rename the file to *filename*, erasing *filename* first if it already exists.

The DFXML files used with this program are generated by the *md5deep* program, allowing *corpus_sync* itself to be quite small. In practice, the *get* operation in step 3b can either create a list of files that need to be transferred, or can copy the file from the transfer media. On a modern 64-bit system we are able to process the DFXML file for a corpus of more than a million objects in under 300 s (not including the time necessary to make the file system alterations).

6.5. flowdiff—difference between two pcap files

We are developing a tool that uses the strategy described in this document to determine the differences between two sets of network packet capture flows. Each flow is processed with the tcpflow (Elson and Garfinkel, 2011) TCP session reconstruction tool. This tool was recently modified to output a DFXML file with a <fileobject> section for each flow. The feature extracted for each flow will consist of a signature for the protocol in question—for example, HTTP or POP. The addresses of each flow will be IP addresses and port number of the flow's origin. We expect that *flowdiff* will be available for download by April 2012.

7. Case study: file system differencing

In the M57-Patents scenario (Woods et al., 2011), one of the personas is a suspect for possessing (simulated) contraband imagery. In the scenario, benign cat pictures are proxies for illicit materials. Suspicions arose when a computer the persona was using was sold on Craigslist

Table 3
Summary statistics for changes in "Jo" computers from the M57-Patents scenario (Woods et al., 2011). Counts are based on inodes. The chosen times represent the base image (Nov. 12, 2009, start of day), the first work day (Nov. 16), next to last day of work for the machine (Nov. 19), and the last day of work for the machine due to software failures. All images were taken at the end of the day except for the Nov. 12 starting image.

	$T_{11-12\ \text{start}} \rightarrow T_{11-20}$	$T_{11-12\ \text{start}} \rightarrow T_{11-16}$	$T_{11-16} \rightarrow T_{11-19}$	$T_{11-19} \rightarrow T_{11-20}$
Files in former image	24,131	24,131	28,735	29,678
Files in latter image	30,497	28,735	29,678	30,497
New files	8546	5140	1157	2773
Deleted files	1900	200	98	1814
Renamed files	463	449	566	703
Files with modified content	1011	687	981	568
Files with changed file properties	3581	1906	4275	1784

without erasing the contents. There is a warrant to search that persona's electronic possessions for the contraband imagery. We show how, with a baseline image available, we can identify content created, removed, updated, modified, or viewed since deployment. This can greatly reduce the search scope for an investigator.

The scenario data include one or more images per computer for each day of operation for the (fictional) company, and baseline images for the initial four computers. It may be that administrators have a subset of these data, particularly a deployment image and an image at time of collection, so they could perform differential analysis. We used *idifference.py* to produce differences between several of the "Jo" images. Table 3 summarizes the change counts and describes the selected times. Table 3 also shows the count of inodes within the disk images, to show the reduction in files one would need to analyze by only looking at changed files. Rename operations make it difficult to get a precise match between the difference in file counts and the difference in new and deleted files. We include differences from disk images that would not commonly be collected, to show normal activity amounts.

Many of the differences appear to come from non-interactive processes. Software updates and Windows background activity dominate the output. However, since in this case the police know they are looking for illicit images, filtering the difference reports for the ".jpg" extension should trim down the results to what they need to show pictures placed on the drive. Indeed, the difference report of $T_{11-12\ \text{start}} \rightarrow T_{11-20}$ reports on a little over three hundred files ending with ".jpg" that changed. Many of those JPEGs turn out to be significant in the scenario. With full file paths provided in the difference report, investigators can quickly find files of interest.

8. Future research

As noted in Section 4.2, to facilitate interpretation for forensic purposes, high-level differential reports are needed. Research and tool building is needed for automatic generation of these differential reports emphasizing occurrence of new features, changes in counts, and disappearance of features. The *idifference.py* and *bulk_diff* tools discussed in Section 6 could be generalized to report changes in any features present in the DFXML regardless of if the DFXML images come from *fiwalk*, *bulk_extractor*, or some other tool. With an appropriate front end, the user could drill-in and select which feature types are of interest to build tailored reports.

By taking the union of *extract feature sets* produced by different *feature extraction* tools and performing differencing on the resulting feature sets, one can build a natural hierarchy of more powerful tools. For example, one could union the *fiwalk* and RegXML feature sets yielding a fusion between file system and Windows Registry features. A differencing tool could then identify co-occurrences in the differences. As discussed in Section 3, differential analysis often involves timeline analysis. Existing timeline approaches can be viewed as very high-level feature extractors—the timeline is the feature. Research is needed on differencing the timelines extracted from pairs (or series) of images.

Further research will always be needed in developing new features, ways of integrating features into metadata, and techniques for extracting features from images. When evaluating the digital evidence (cellphones, computers, etc.) for a newly captured criminal, at first glance there are not pairs of images to delta, just the current images. However, one can always perform differencing against stock images extracted from virgin equipment and OS installs. Changes in firmware and OS updates can help timeline the captive's behavior and location. Modulo legal issues, if the captive is subsequently released and re-caught, differencing can be performed between previous and new images. Research is needed on what features are relevant for deciding on holding or releasing.

9. Conclusion

All differencing tasks are fundamentally identical whether operating directly at the content level such as diff or with higher features such as the *bulk_diff* tool we introduce above. In this paper we introduce a common vernacular for discussing differencing and outline the state-of-the-art for both forensic and traditional applications. From this foundation we note the universal similarities and elicit a general strategy hidden in all extant differencing systems.

By explicitly leveraging this strategy, we propose that development of future differencing tools will be streamlined. To demonstrate, we outline the functionality of our new tools within the lens of our terminology and strategy. Furthermore, we expect that educating forensic analysts to use new differencing techniques will be easier if cached within the context of our strategy.

Acknowledgments

Portions of this work were funded by NSF Award DUE-0919593. We wish to thank Adam Russell for his useful inputs.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. Government. The U.S. Government is authorized to reproduce, distribute, or authorize reprints for any reason notwithstanding any copyright annotations thereon.

References

Access Data. Forensic toolkit (FTK); 2011.

Apiwattanapong T, Orso A, Harrold MJ. A differencing algorithm for object-oriented programs. In: Proceedings of the 19th IEEE international conference on automated software engineering. Washington, DC, USA: IEEE Computer Society; 2004. p. 2–13.

Beverly R, Garfinkel S, Cardwell G. Forensic carving of network packets and associated data structures. In: Proceedings of the 2011 DFRWS conference; 2011. p. S78–89.

Carrier B. The Sleuth Kit and Kutopsy: forensics tools for Linux and other Unixes; 2005.

Duley A, Spandikow C, Kim M. A program differencing algorithm for Verilog HDL. In: Proceedings of the IEEE/ACM international conference on automated software engineering. New York, NY, USA: ACM; 2010. p. 477–86. ASE '10.

Elson J, Garfinkel S. tcpflow; 2011.

Garfinkel S. Digital Forensics XML. Digital Investigation 2012;8:161–74.

Garfinkel S, Parker-Wood A, Huynh D, Migletz J. A solution to the multi-user carved data ascription problem. IEEE Transactions on Information Forensics and Security 2010;5(4):868–82.

Garfinkel S, Rowe N. Global analysis of drive file times. In: Fifth international workshop on systematic approaches to digital forensic engineering. Oakland, CA: IEEE; 2010. p. 97–108.

Garfinkel SL. Automating disk forensic processing with SleuthKit, XML and Python. In: Proceedings of the fourth international IEEE workshop on systematic approaches to digital forensic engineering. Oakland, CA: IEEE; 2009. p. 73–84.

Garfinkel SL, Farrell P, Roussev V, Dinolt G. Bringing science to digital forensics with standardized forensic corpora. In: Proceedings of the 9th annual digital forensic research workshop (DFRWS). Quebec, CA: Elsevier; 2009.

Grosenbach G. Git source code control. 1st ed. PeepCode; 2007.

Guidance Software, Inc. EnCase Forensic; 2011.

Horwitz S. Identifying the semantic and textual differences between two versions of a program. In: Proceedings of the ACM SIGPLAN 1990 conference on programming language design and implementation. New York, NY, USA: ACM; 1990. p. 234–45. PLDI '90.

Hunt JW, McIlroy MD. An algorithm for differential file comparison. Technical report 41. Bell Laboratories; 1976.

Jacobson J. A formalization of Darcs patch theory using inverse semigroups. Technical report CAM09-83. UCLA; 2009.

Kim GH, Spafford EH. The design and implementation of tripwire: a file system integrity checker. In: Proceedings of the 2nd ACM conference on computer and communications security. New York, NY, USA: ACM; 1994. p. 18–29. CCS '94.

Kornblum J. md5deep and hashdeep—latest version; 4.1.2011. http://md5deep.sourceforge.net/ [last accessed 18.02.12].

Levy J. Differential EnScript v1; 2012.

Lindholm T, Kangasharju J, Tarkoma S. Fast and simple XML tree differencing by sequence alignment. In: Proceedings of the 2006 ACM symposium on document engineering. New York, NY, USA: ACM; 2006. p. 75–84. DocEng '06.

Loh A, Kim M. LSdiff: a program differencing tool to identify systematic structural differences. In: Proceedings of the 32nd ACM/IEEE international conference on software engineering, vol. 2. New York, NY, USA: ACM; 2010. p. 263–6. ICSE '10.

Marrington A, Baggili I, Mohay G, Clark A. CAT Detect (computer activity timeline detection): a tool for detecting inconsistency in computer activity timelines. In: Proceedings of the 2011 DFRWS conference; 2011. p. S52–61.

Myers E. An O(ND) difference algorithm and its variations. Algorithmica 1986;1:251–66.

Nelson AJ. RegXML: XML conversion of the Windows registry for forensic processing and distribution. In: Chow KP, Shenoi S, editors. Advances in digital forensics VIII. Springer; IFIP Advances in Information and Communication Technology, in press.

NIST. National Software Reference Library; 2009. http://www.nsrl.nist.gov/Documents/NSRL-CFS-April-2009.pdf.

Olsson J, Boldt M. Computer forensic timeline visualization tool. In: Proceedings of the 2009 DFRWS conference; 2009. p. S78–87.

Pierce BC, Vouillon J. What's in Unison? A formal specification and reference implementation of a file synchronizer. Technical report; 2004.

Pilato C, Collins-Sussman B, Fitzpatrick B. Version Control with Subversion. 2nd ed. O'Reilly Media, Inc.; 2008.

Roundy D. Darcs: distributed version management in Haskell. In: Proceedings of the 2005 ACM SIGPLAN workshop on Haskell. New York, NY, USA: ACM; 2005. p. 1–4. Haskell '05.

Schatz B, Mohay G, Clark A. A correlation method for establishing provenance of timestamps in digital evidence. In: Proceedings of the 2006 DFRWS conference; 2006. p. S98–107.

Thompson K, Ritchie DM. diff – differential file comparator. UNIX programmer's manual. 6th ed.; 1975.

Tichy WF. Design, implementation, and evaluation of a revision control system. In: Proceedings of the 6th international conference on software engineering. Los Alamitos, CA, USA: IEEE Computer Society Press; 1982. p. 58–67. ICSE '82.

Tridgell A, Mackerras P. The rsync algorithm. Technical report TR-CS-96-05; ANU computer science technical reports; 1996.

Watkins K, McWhorter M, Long J, Hill W. Teleporter: an analytically and forensically sound duplicate transfer system. In: Proceedings of the 2009 DFRWS conference; 2009.

Weaver GA, Foti N, Bratus S, Rockmore D, Smith SW. Using hierarchal change mining to manage network security policy evolution. In: Proceedings of the 11th USENIX conference on hot topics in management of internet, cloud, and enterprise networks and services. Berkeley, CA, USA: USENIX Association; 2011. p. 8. HotICE'11.

Weaver GA, Smith SW. Context-free grep and hierarchical diff. In: LISA '11: 25th large installation system administration conference. USENIX; 2011.

White D. Tracking computer use with the windows registry dataset; 2008.

Woods K, Lee C, Garfinkel S, Dittrich D, Russell A, Kearton K. Creating realistic corpora for forensic and security education. In: 2011 ADFSL conference on digital forensics, security and law. Richmond, VA: Elsevier; 2011.