Teleporter: An Analytically and Forensically Sound Duplicate Transfer System

By

Kathryn Watkins, Mike McWhorter, Jeff Long and William Hill

From the proceedings of

The Digital Forensic Research Conference

DFRWS 2009 USA

Montreal, Canada (Aug 17th - 19th)

DIGITAL INVESTIGATION 6 (2009) S43–S47

available at www.sciencedirect.com

ScienceDirect

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

Digital Investigation

Teleporter: An analytically and forensically sound duplicate transfer system

Kathryn Watkins, Mike McWhorte, Jeff Long, Bill Hill*

MITRE Corporation, 7515 Colshire Drive, Mailstop T730, Mclean, VA 22102, USA

ABSTRACT

This research seeks to minimize the transmission of unnecessary data when sending forensically sound drive images from remote client locations back to a central site. Files such as operating-system files, applications, and media files are likely to exist on many hard drives. The concept combines common data from local stores with unique data from the remote site to create new, forensically sound media images. Data that can be used from local stores need not be sent over the link. The result is substantially reduced amounts of data that must be transmitted between the two sites and stored at the central location.

This paper describes Teleporter, a highly optimized file transfer method. The described research will also refine the understanding of the term forensically sound and introduce the term analytically sound.

1. Introduction

Various law enforcement entities obtain hard drive images for analysis over bandwidth limited WAN connections. When no additional technologies can be applied to better utilize the speed of the connection and no additional bandwidth can be attained, critical operations that rely on intelligence gathered from the timely analysis of drives are hindered. Improving the transmission process will result in tactical gains for law enforcement.

Although it might seem that an ideal tool to address this problem would compare files on the remote client hard drive to a list of existing files at the server-side and send the difference, this approach would not create a forensic duplicate and would miss analytically significant information that is outside of the file system such as slack space data or deleted/partial files. To enable a bit-for-bit forensic copy, the transfer tool must work directly with disk blocks.

While the process will operate at the disk level, critical insights gained from understanding the file system geometry opens the possibility for performance improvements. If the tool merely substitutes transmission of blocks with transmission of block ID codes, the savings is simply the difference in size of those two; a modest benefit. The first insight is that under some circumstances blocks assigned to a file will be contiguous. If the tool can send an identifier that indicates a common sequence of blocks, there is a greatly increased potential savings. Two methods of identifying sequences were tested. The first method works at a high level of abstraction (files) and can work at a lower level when needed. The second method works at a low level of abstraction (physical sectors or logical clusters) and can work at a higher level when needed. The first method was proven to have better gains and thus implemented.

A second insight is that large regions of the drives are ''empty''—they contain no user data. Drive formatting typically fills blocks with repeating sequences which are often but not always zeros. If, similarly to above, the tool could send an identifier which indicated a sequence of these blocks, there would be significantly less data to transmit.

* *Corresponding author*. +1 703 983 5685.

E-mail addresses: kwatkins@mitre.org (K. Watkins), mcwhorter@mitre.org (M. McWhorte), jefflong@mitre.org (J. Long), bill@mitre.org (B. Hill).

1742-2876/$ – see front matter © 2009 Digital Forensic Research workshop. Published by Elsevier Ltd. All rights reserved.
doi:10.1016/j.diin.2009.06.012

Teleporter offers a refinement of the term forensically sound. The accepted definition of forensically sound is a bit-for-bit copy of a drive. Teleporter produces the same result but in a new way. It creates a bit-for-bit copy of a drive image by using data from other images. This research also introduces the term analytically sound. An analytically sound image doesn't remove all possible previously seen data. Instead it allows analysts to skip initial and timely pre-conditioning steps that forensic tools take by removing only known good (known file filter) data. This drive can still be rebuilt into a forensically sound duplicate of the original but in its current state is ready for analysis.

Proving the fidelity of the definition of forensically sound and proving the validity of the term analytically sound will further the science of computer forensics.

2. Prior work

This research combines previous research in low bandwidth distributed file systems with research in hash reliant backup and archival systems.

Projects such as rsync (Tridgell and Mackerras, 1997), a utility for synchronizing files, and the Low Bandwidth File System (LBFS) (Muthitacharoen et al., 2001), a distributed file system have laid much groundwork for optimizing transfers using hashing. Combining these with research ideas such as SIF (Manber, 1994), a tool to find similar files in large file systems make up the work of Teleporter.

Rsync, in an oversimplified explanation, has a client and server-side. The client-side breaks a file into fixed length chunks and sends those chunks to the server-side. The server-side then compares those fixed length chunks to hashes of all overlapping chunks of the same file. If any of these sliding-window hashes match the client-side chunk-hashes, the client can avoid sending those specified chunks of data. This works well for synchronizing known sets of files.

LBFS enables low-latency access to files and applications over low-bandwidth connections by providing an optimally synchronized file system. Like rsync, LBFS, uses hashing to identify known pieces of files. Because the idea of LBFS is to make data available to the remote client, it utilizes a large persistent file cache on the client side. Because this research only needs to optimally transfer and then store data at the server-side, it can streamline these client-side operations.

Policroniades and Pratt (2004) provide quantitative comparisons of duplication detection schemes that rely on hashing. Single Instance Storage (Bolosky et al., 2000) introduces the ability to store files only once while Venti (Quinlan and Dorward, 2002), a block level storage system, introduces the idea of storing physical blocks of data only once. Teleporter combines these two ideas in both transmission and storage.

Backup systems such as Patiche (Cox et al., 2002) and using XML to describe files (Tolia et al., 2003) lent insight that made certain properties of Teleporter more viable.

These prior works combined with understanding of file system geometry described in (Carrier, 2005) has made this research possible.

3. Method

Teleporter achieves two goals that will both decrease the amount of data that needs to be transmitted. It can reduce a drive to its most minimal state by referencing the parts of the drive that already exist at a server-side. It also provides a drive that needs minimal pre-processing before an analyst can begin work on the image while remaining forensically sound, thus analytically sound. These outcomes are achieved individually with a specific implementation focused on either the minimalistic state or the analytic state depending on an entities needs or as a hybrid of the two. Although the minimalistic state was implemented for this research, the hybrid state will be explained to understand all possible benefits.

As previously stated, Teleporter works by leveraging large data stores that forensic investigative entities already maintain. The current implementation of Teleporter builds a single instance file storage from the current holdings. It then forward deploys hash databases representative of the single instance storage to clients. The clients then use these databases to compare data on incoming drives to those already stored at the server-side. This architecture allows work that could previously only be conducted at the server-side to be moved to the client.

Forensic analysis tools perform a pre-processing of drives to narrow the base of analysis. This is done by redacting known files from the larger drive. Current forensic tools do this by maintaining a hash database of known files. Because they do not maintain the files that correspond to the hashes, this processing must be done by the analysis tool so that the drive image can be rebuilt upon completion of analysis. Teleporter is able to do this pre-processing at the client side of the operation because it maintains a copy of all known files at the server-side and because it maintains how the files are laid out on the drive being analyzed.

In addition to eliminating the need to send known files, Teleporter is able to identify any file or partial file that it has previously seen. As a result, all files that contribute to reducing a drive to its analytic state would also be redacted when building the minimalistic state. Thus, known files identified by existing forensic programs as being insignificant to analysis will be flagged on the server-side's database.

Three things are then possible when the server-side receives a skeleton image. The drive can be completely rebuilt to its original state. For testing purposes hashes of the original client drive and the resulting drive rebuilt from the skeleton image were compared to prove forensic soundness. The skeleton image can be stored in the reduced state until needed for analysis. Or, ideally, the server-side can rebuild the drive adding only files that it does not have marked with the analytic flag. This builds a drive that is immediately ready for analysis. All files that would be redacted during pre-processing with analysis tools have already been done. The resulting drive contains all files, slack space, and partial files that will be of interest.

The last step of the process is to add the previously unknown files and partial files to the server-side database.

The community accepted definition of a forensically sound drive is as follows: A 'forensically-sound' duplicate of a drive

is, first and foremost, one created by a method which does not, in any way, alter any data on the drive being duplicated. Second, a forensically sound duplicate must contain a copy of every bit, byte and sector of the source drive, including unallocated 'empty' space and slack space, precisely as such data appears on the source drive relative to the other data on the drive. Finally, a forensically-sound duplicate will not contain any data (except known filler characters) other than which was copied from the source drive (Murr, 2006). Teleporter does this but in a new way.

4. Implementation

4.1. Hash algorithm

Teleporter uses the SHA-256 hash algorithm because of its collision-resistant properties (Gilbert and Handschuh, 2005; Schneier, 1996). Teleporter is not dependant on any particular hash algorithm. Any one-way hash function (Naor and Yung, 1989) can be substituted, provided that it is collision-resistant with respect to security and size where size refers to the server-side data store. As related to the birthday attack (McKinney, 1966) the number of files in the server-side data store must be less than $2^{N/2}$ where N is the length of the hash to avoid a probable collision (>50%). For example, if the implementation requires to have a less than 1% chance of collision when using a 256 bit hash, the number of items in the repository must be fewer than $4.8e10^{37.}$

4.2. Index

The server-side of Teleporter uses a MySQL database with two tables. One table is for known files and the other is for known disk clusters. The known_files table contains only one field – the file's hash value. Since the files are stored in a hash tree, the hash value also serves as the file name and path to the file. The hash value is stored in binary format using MySQL's BINARY data type. It is indexed to provide faster lookup times.

The known_clusters table contains two fields – the cluster hash and the cluster's contents. The cluster's hash value is stored as a BINARY field, and the content of the cluster is stored as a binary large object (BLOB). The hash value field is indexed for faster lookup times.

The client side of Teleporter uses a database that contains the hashes of both the files and clusters (both physical and logical hashes). The client-side database contains no data other than the hashes.

4.3. File system parser

At its core, Teleporter is a Java application. However, it relies on an external support application which is file system specific. Since every file system stores objects differently, a parser must be written for every file system that Teleporter supports. The parser reads the disk image and generates an XML file describing the layout of the disk. Teleporter uses this XML file to read the image and access files. This design was chosen to make the software more modular. If Teleporter needs to be extended to support other file systems, the developer need only write a parser for that file system. The new parser can be deployed without any modifications to Teleporter's source code.

The current version of Teleporter includes an NTFS parser. It is a modified version of an open source utility called ntfsls which has been altered to produce an XML file as its output.

4.4. File and cluster storage

In addition to the database, the server-side must maintain a repository of known files to rebuild the original disk image. The repository's directory structure is arranged in a hash tree configuration where files are named according to their hash values and placed in directories corresponding to what those values are. This improves access time since there is no lookup step required to locate a file. It also allows for scalability since the leaves of the tree can be distributed evenly across several physical volumes.

Individual disk clusters are stored in the database as BLOBs. Typically, pulling files from a file system is faster than pulling them from a database. However, due to the relatively small size of a disk cluster and the number of clusters that must be processed, the extra overhead of opening and closing millions of small files causes a significant decrease in speed. Our testing indicated a fivefold increase in speed when disk clusters were stored in the database rather than on the file system.

When the server-side receives a skeleton image, it is not necessary to rebuild this image into its original state. The image can be permanently stored as its skeleton and rebuilt only when needed in its original state. This offers a percentage of space savings that correlates to the percentage of transmission time savings.

4.5. Protocol communication

In an effort to reduce the transmission times of drive images and to use the uplink bandwidth as efficiently as possible, there is no communication between the client and server until transmission is initiated. This property is unique to Teleporter in comparison to other file transfer and backup systems. The client-side process of creating a skeleton image from the original is done in a stand-alone environment. This is accomplished by the pre-deployment of the client-side database. At the point of transmission, the sender and receiver can choose whatever method of file transfer best utilizes the available bandwidth.

4.6. Processing files

Teleporter recognizes known files by their logical hash values – the hash value of the file as it is read from the file system thus excluding slack space. As stated in Section 4.2, both the sender and receiver maintain a database of hash values for every known file. When new disk image is scanned at the client location, a hash value is computed for each file and compared to the known hash database.

If a file is unknown, it, and any slack space, is copied to the skeleton image along with a record describing how it is stored on disk so that the file segments can be properly placed at the

server-side and so that a physical hash of the area containing the file can be validated. If a file is known, it is not copied to the skeleton image, but replaced with a record containing its logical hash value, any slack space data, and where it is stored on disk. Information regarding where it is stored on disk is important for maintaining forensic integrity. The goal is block by block fidelity.

4.7. *Processing free space*

After processing all of the files, Teleporter begins scanning the "free space"—sections of the disk which the file system considers to be empty. This area often contains partial files and residual copies of deleted files. These must be captured as well in order to preserve the forensic integrity of the image. Teleporter scans these areas and records the contents of every unused disk cluster to ensure that every block of data on the disk is accounted for.

In addition to recognizing known files, Teleporter can also recognize known disk clusters. Like the files, Teleporter recognizes known disk clusters by their hash values. As previously described, both the sender and receiver maintain a list of hash values for every known cluster. When the free space is scanned on the new client drive, a hash value is computed for each disk cluster and compared to the known clusters database. If a cluster is unknown, its contents and location are stored in the skeleton image. If a cluster is known, only its hash value and location are stored.

4.8. *Additional compression*

LBFS (Muthitacharoen et al., 2001) boasts an order of magnitude in savings when combining their implementation with traditional compression and caching algorithms. Teleporter produces similar levels of savings when a standard compression algorithm is applied to the skeleton image. The unknown files and disk clusters are likely to contain long runs of zeroes (sparseness), making the skeleton image a good candidate for compression. All drives are compressed as a last step before sending.

4.9. *Update problem*

A problem exists in maintaining synchronization between the client and server-side databases. Teleporter was designed to require no communication between the server and client unless the client is sending an image. Achieving the requirements creates a problem in how updates of the hash database will be sent to the client side. Two options are available for addressing this problem and choosing an update option is dependent upon the architecture of the system and the network that supports it.

The first option is that every time an image is sent to the server-side, the client-side assumes that the server-side has correctly ingested each new file. By making this assumption, both sides can update their own hash databases when a new file is seen. The advantage to this scheme is that it uses less bandwidth than other update options. The disadvantage is that this only works when there is one client and one server.

A second option to address the update problem is sending periodic updates to the client side. Unlike the previous update solution this ensures proper synchronization between the server-side and client-side operations. The disadvantage to this solution is that it introduces additional communication between the server and client. Additionally, duplicates that could have been identified in drives ingested between identification of the file and updates being sent will be missed.

5. Testing and performance

5.1. *Testing results*

Fourteen drives were available for testing Teleporter. A physical hash—a hash encompassing all physical portions of a drive was calculated at both the client side and the server-side of the testing. Should one bit have been misplaced the test would have been considered a failure. Furthermore, it was ensured that drives were not best-case-scenarios. Although all drives used Microsoft Windows XP in order to represent a genre of drives likely to be encountered, no two drives were identical. Additionally, some drives were built from clean installs while some were installed over older operating systems without initial complete formatting. Each drive went through the described ingestion process and the hash database was then re-deployed to include knowledge gained from the previous drives. Savings ranged between 30 and 95%. The thirty percent drive being the one with files from a previous operating system still existing and ninety-five percent drive being from a one with only a base Windows XP installation.

5.2. *Expected shortcomings*

A production implementation of Teleporter would ingest all existing data in the server-side data store. A resulting, comprehensive, known hash database is then sent to the client side. A deployment that is not able to leverage an existing data store will only show savings based on knowledge gained from the previous subset of drives sent as was shown in testing of the initial version of Teleporter. Although this implementation is not ideal, performing testing in this way allowed gained understanding of the rates at which Teleporter learns and to better forecast expected savings when leveraged against a much larger data store. A full implementation would expect to see between 50 and 70% savings per drive. This expected percentage is a correlation between the numbers of files that a forensic analyst normally eliminates (on a per operating-system basis) before analysis and our expectancy of Teleporter to ingest all of those known files.

6. Improving results: future work

6.1. *Send indexes of hashes*

The current version of Teleporter sends full hashes as the metadata to represent a chuck of known data. Small gains can be achieved by sending indexes of hashes instead of the hashes themselves. This is a sound assumption because the

hash is chosen as $2^{N/2}$ where N is the length of the hash. As described in Section 4.1, this is done to ensure collision resistance. Thus we know that the number of files is less than the number of possible hashes.

6.2. *Pattern matching*

While much can be gained by sending hashes or indices that represent known chunks of data, more could be gained if a single piece of metadata could represent some pattern of chunks of data. We begin to accomplish this by sending hashes of files instead of just hashes of clusters—think sequences of clusters as files. The logic follows that additional savings can be found by sending hashes representative of sequences of files.

We have verified that when identical options are chosen within a standard Microsoft Windows XP installation, the initial Operating-System files are laid out identically from disk to disk. A problem exists in that clusters are intentionally left empty between some files. The file system will then fill these empty clusters with fragmented files. The existing implementation of Teleporter requires sending information about where the file or cluster is stored on a physical disk. Because of these empty clusters between files, file location information in respect to physical clusters would still be required to be sent. Even after an Operating System is updated, some of this original pattern would still exist. One could send an identifier of the sequence, the length of consecutive data on the new disk that follows the sequence, and locations of files on the disk.

7. Conclusion

Teleporter reduces the amount of data that must be sent across slow WAN links by 50–70% in initial testing. Teleporter also proves that one can create a forensically sound duplicate of a drive using pieces of other drives. Lastly, Teleporter offers an analytically sound drive, one that is available for analysis much quicker than was previously possible.

REFERENCES

Bolosky WJ, Corbin S, Goebel D, Douceur JR. Single Instance Storage in Windows 2000. In: fourth Usenix Windows System Symposium; Aug 2000.

Carrier B. File system forensic analysis. Addison Wesley; 2005.

Cox LP, Murray CD, Noble BD. Pastiche: making backup cheap and easy. SIGOPS Oper Syst Rev 2002;36(SI):285–98.

Gilbert H, Handschuh H. Security analysis of SHA-256 and sisters, selected areas in cryptography 2003 NIST Cryptographic Hash Workshop; 2005

Manber U. Finding similar files in a large file system. In: Proceedings of the USENIX winter technical conference; 17.21 1994.

Murr Michael. Forensically sound duplicate. Forensic Computing, <http://forensiccomputing.blogspot.com/2006/08/forensically-sound-duplicate.html>; 02 Aug 2006.

Muthitacharoen A, Chen B, Mazières D. A Low-bandwidth network file system. In: Proc. 18th SOSP; Oct. 2001.

McKinney EH. In: Generalized birthday problem, vol. 73. American Mathematical Monthly; 1966. p. 385–7.

Naor M, Yung M. Universal one-way hash functions and their cryptographic applications. In: 21st annual ACM symposium on theory of computing; 1989.

Policroniades C, Pratt I. Alternatives for detecting redundancy in storage systems data. In: Proceedings of the 2004 Usenix conference; June 2004.

Quinlan S, Dorward S. Venti: a new approach to archival storage. In: Proc. of the Conference on file and storage technologies (FAST); January 2002.

Schneier B. Applied cryptography. 2nd ed. John Wiley & Sons; 1996.

Tolia N, Kozuch M, Satyanarayanan M, Karp B, Bressoud T, Perrig A. Opportunistic use of content addressable storage for distributed _le systems. In: Proceedings of the 2003 USENIX annual technical conference; May 2003. p. 127–40.

Tridgell A, Mackerras P . The rsync algorithm. Tech. Rep. TR-CS-96-05. Australian National University; 1997.