Forensic Timeline Analysis of the Zettabyte File System

Dylan Leigh <research@dylanleigh.net>
Supervisor: A.Prof. Hao Shi <hao.shi@vu.edu.au>
Centre for Applied Informatics, College of Engineering and Science, Victoria University, Melbourne

***Abstract*—During forensic analysis of computer systems, it is often necessary to construct a chronological account of events, including when files were created, modified, accessed and deleted. Timeline analysis is the process of collating and analysing this data, using timestamps from the filesystem and other sources such as log files and internal file metadata.**

The Zettabyte File System (ZFS) uses a novel and complex structure to store file data and metadata across multiple devices. Due to the unusual structure and operation of ZFS, many existing forensic tools and techniques cannot be used to analyse ZFS filesystems.

In this project, it has been demonstrated that four of the internal structures of ZFS can be used as effective sources of timeline information. Methods to extract these structures and use them for timeline analysis are provided, including algorithms to detect falsified file timestamps and to determine when individual blocks of file data were last modified.

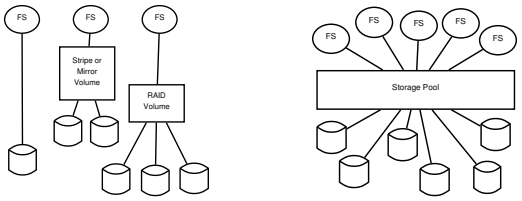
I. Objectives

The objectives of this project are to identify internal ZFS metadata which can be used for timeline analysis, and provide methods to collect this metadata. It must also be determined how this metadata is related to the creation and modification time of files, and how long any useful metadata will be retained. Finally, this project should provide at least one effective method which uses ZFS metadata to detect falsified file timestamps.

The knowledge obtained from the study should be used to provide practical guidelines and procedures for forensic investigators, and shared amongst the community to provide the basis for future forensic research and development of ZFS forensic software. It is intended that at least one publication is produced from this research in the area of timeline forensics, plus one possible publication related to the study of the ZFS on-disk format.

II. Background: ZFS

ZFS was designed[1] to meet the needs of enterprise server storage and surpass many limitations of existing filesystems. ZFS is a popular filesystem for many applications due to its scalability, high availability and guaranteed data integrity [2].



(a) Traditional FS/Volume (b) ZFS Pool

Figure 1. Structure of Traditional Filesystems and Volumes compared to ZFS Pools

A. Pooled Storage

Traditional volume management involves mapping individual filesystems to volumes and disks;. filesystems are fixed in location on the disk or volume.

ZFS uses a different approach: all disks are allocated to a "pool"; and filesystems are created and mounted as required administratively. Data is stored within different devices by ZFS to optimise performance and maintain redundant copies to prevent loss of data from device failure.

Creating a new ZFS filesystem on a pool is a lightweight process similar to creating a directory on other filesystems. Figure 1 shows this pooled structure of the Zettabyte File System compared to "traditional" file systems.

B. Guaranteed Data Integrity

Data integrity and prevention of corruption was a key design consideration of ZFS. ZFS checksums all data and metadata and transparently detects and recovers from corruption using the redundant copies. Even with a "pool" of a single device, the checksums detect corruption which would occur without warning on other filesystems [2].

ZFS uses a copy-on-write transaction operation for all writes, which ensures that no data is ever overwritten in place and the disk is never left in an inconsistent state. All disk writes by ZFS will either complete atomically or not at

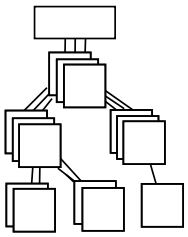


Figure 2. Simplified ZFS Object Tree

all. This prevents corruption in the event of power failure or a hardware fault. ZFS can also recover faster from such faults as no file system check is required when the system is restarted.

C. Internal Structure

ZFS stores all data and metadata within a tree of objects, stored within a tree of blocks. Each "Block Pointer" reference to another block can point to up to three identical copies of the target block, which ZFS spreads across multiple devices in the pool in case of disk failure.

Each physical device contains a "device label" with an array of 128 "Uberblocks", the root of the block and object trees. Each transaction is committed to a new Uberblock (overwriting the oldest one in the array).

Each Uberblock contains a Block Pointer pointing to the root of the object tree, the "Meta Object Set". Apart from the metadata contained within the label, all data and metadata is stored within the object tree, including internal metadata such the free space maps and delete queues.

III. ZFS Forensics Literature

Beebe et al's *Digital Forensic Implications of ZFS* [3] is an overview of the forensic differences between ZFS and more "traditional" file systems; in particular it outlines many forensic advantages and challenges of ZFS. However, it is based on theoretical examination of the documentation and source code, noting that "we have yet to verify all statements through our own direct observation and reverse engineering of on-disk behavior".

Data recovery on ZFS has been examined extensively by other researchers. Max Bruning's *On Disk Data Walk* [4] demonstrates how to access ZFS data on disk from first principles; Bruning has also demonstrated techniques for recovering deleted files[5].

Li [6], [7] presents an enhancement to the ZDB filesystem "debugger" which allows it to trace target data within ZFS media without mounting the file system. Finally, some ZFS anti-forensics techniques are examined by Cifuentes and Cano [8].

Table I
ZFS Structures and Components used for Timeline Analysis

Structure	Component	Effective
Uberblocks	TXG and Timestamp	Sometimes
File Objects	Object Number	Sometimes
	Generation TXG	Always
File Data Block Pointers	Birth TXG	Always
	Vdev Number	Never
Spacemaps	Spacemap TXG	Never

Publications

This research has directly produced one publication [9] discussing timeline analysis of ZFS; it has also been used in a related project to extend existing super-timeline software to to process ZFS events which has produced another publication by the author [10].
Additional publications are planned for release in 2015 using thesis material which has not yet been published, as well as additional case studies based on more detailed simulations.

IV. Experiment Methodology

A. Primary Experiments

ZFS metadata was collected from test systems which were configured to use ZFS pools with 1,2,3,4,5,7 and 9 devices, in a variety of pool configurations. Disk activity was simulated using data obtained from surveys of activity on corporate network file storage [11], [12], [13].

Three experiments were conducted for each pool configuration: a control with no tampering, one where timestamps were altered by changing the system clock backwards briefly by one hour, and one where the Unix "touch" command was used to alter the modification and access times of one file backwards by one hour. Tampering was conducted 5 hours into each simulation.

Each simulation was conducted for 24 hours. Metadata was collected every 30 minutes, using the ZDB (ZFS debugger) tool. After each simulation had concluded, a final set of metadata was collected, which was examined to determine if any timestamp falsification could be detected. Table I shows the structures and the components of those structures which were used for timeline analysis after each simulation.

Experiments were repeated as time permitted; in total 85 simulations were completed (not including the "Large File" experiments) and over 110GB of ZFS metadata was collected.

B. Large File Experiments

A second set of 22 experiments were conducted to verify behaviour of files larger than

128KB observed during the primary experiments. These were conducted in a similar manner to the primary experiments, with the addition of a "large file" which was created at the start of the simulation and appended to every second.

No tampering was used; instead the analysis after the simulation compared internal metadata from the large file with other files. By matching the "birth" Transaction Group ID (TXG) from individual blocks of the large file with birth TXG from highest level block pointers in other files, it is possible to determine the time at which the individual blocks of the large file are written, which cannot generally be determined via other means.

V. Results and Key Findings

Table II shows the results of performing timeline analysis on all experiments involving tampering. In experiments without tampering; all methods were negative and there were no false positives.

The most significant findings for ZFS timeline analysis are listed below:

1) Comparing the birth TXG ID from the highest level file data Block Pointers is an effective and reliable method for detecting false timestamps. As birth TXG ID is closely related to the time and order of modification of blocks, they are useful for other timeline analysis techniques as well.
2) The birth TXG from level 0 Block Pointers in files containing multiple blocks can be used to determine the last modification time of the individual blocks in a large file.
3) The Generation TXG and Object Number of a file object can be used to determine the creation order of objects, and are effective at detecting files which have a false creation time. The Object Number method is not reliable as these numbers may be reused; the effect of this needs to be confirmed via further research. The use of Generation TXG is reliable.
4) Uberblocks may be used to detect falsified timestamps but only if collected quickly, before the relevant Uberblock is overwritten. In a pool under a constant write load, Uberblocks are typically overwritten after 10.6 minutes, except for pools with more than 4 disks and no redundancy where Uberblocks may persist for several hours.
5) Spacemaps and the Vdev number do not appear to be effective as a source of events for timeline analysis, due to the many transient objects caused by the copy-on-write mechanism of ZFS and the high rate at which Spacemaps are condensed.

A. Other Contributions

The potential for false positives (from timestamps which are physically false but logically true, such as from copying old files) are discussed, as well as the effects of file attribute changes and tampering with internal ZFS metadata.

This project also contributes to forensic practice by providing methods and commands for analysis of ZFS devices, including general computer forensic procedures for ZFS as well as those specific to timeline analysis. In particular, techniques to safely import a ZFS pool from forensic disk images without corrupting the host system or the system under examination are provided.

VI. Future Work

Although this is the first time internal ZFS metadata has been examined for timeline forensics, the scope of this study is extremely limited and much more work needs to be done to validate the techniques presented here under varied conditions.

A. Survey

Most importantly, data from production systems needs to be obtained to verify our results with real systems. The next phase of this project will involve a survey to obtain metadata from real systems. Volunteers would be asked to submit anonymized ZDB data (with paths and names removed). A script has already been developed to perform this anonymisation and ethical approval obtained for the survey.

B. Tampering with Internal Metadata

A determined forger could manually alter internal ZFS structures to match any modified file timestamps this on actual disks yet, although the effects are theoretically discussed.

C. Other ZFS Structures

Only four ZFS structures have been examined at this stage. There are many other ZFS objects and structures which could be used for timeline analysis.

D. Deleted ZFS Structures

The copy-on-write nature of ZFS is likely to leave past copies of many objects. Regions of the disk which are no longer referenced could be examined to see if they contain old ZFS objects or Block Pointers.

Table II
FALSIFICATION DETECTION RESULTS BY FALSIFICATION METHOD

Detection Method	System Clock Tampering		Touch Command Tampering	
	True Positive	False Negative	True Positive	False Negative
Uberblock TXG (within 30 min.)*	15	11	11	16
Uberblock TXG (after 24 hours)	0	26	0	27
Block Pointer TXG	26	0	27	0
Generation TXG	26	0	0	27
Object Number	26	0	0	27
Spacemaps	0	26	0	27
Vdev Number	0	26	0	27
(Number of Experiments)	26		27	

E. Other Pool Configurations and Workloads

Both sets of experiments used simulated file activity based on a corporate network file storage workload. Other types of systems (e.g. desktop, database server, virtual machine storage) could be expected to have different patterns of file activity.

REFERENCES

[1] Bonwick J, Ahrens M, Henson V, Maybee M, Shellenbaum M. The zettabyte file system. In: Proc. of the 2nd Usenix Conference on File and Storage Technologies; 2003. Available from: http://users.soe.ucsc.edu/~scott/courses/Fall04/221/zfs_overview.pdf.

[2] Zhang Y, Rajimwale A, Arpaci-Dusseau AC, Arpaci-Dusseau RH. End-to-end data integrity for file systems: a ZFS case study. In: Proceedings of the 8th USENIX conference on File and storage technologies. USENIX Association; 2010. p. 3–3. Available from: http://research.cs.wisc.edu/adsl/Publications/zfs-corruption-fast10.pdf.

[3] Beebe NL, Stacy SD, Stuckey D. Digital forensic implications of ZFS. Digital Investigation. 2009;6:S99–S107. Available from: http://www.sciencedirect.com/science/article/pii/S1742287609000449.

[4] Bruning M. ZFS On-Disk Data Walk. In: OpenSolaris Developer Conference; 2008. Available from: http://www.osdevcon.org/2008/files/osdevcon2008-max.pdf.

[5] Bruning M. Recovering Removed File on ZFS Disk; 2008. (website). Available from: http://mbruning.blogspot.com.au/2008/08/recovering-removed-file-on-zfs-disk.html.

[6] Li A. Digital Crime Scene Investigation for the Zettabyte File System. Macquarie University; 2009. Available from: http://web.science.mq.edu.au/~rdale/teaching/itec810/2009H1/FinalReports/Li_Andrew_FinalReport.pdf.

[7] Li A. Zettabyte File System Autopsy: Digital Crime Scene Investigation for Zettabyte File System; 2009. Available from: http://web.science.mq.edu.au/~rdale/teaching/itec810/2009H1/WorkshopPapers/Li_Andrew_FinalWorkshopPaper.pdf.

[8] Cifuentes J, Cano J. Analysis and Implementation of Anti-Forensics Techniques on ZFS. Latin America Transactions, IEEE (Revista IEEE America Latina). 2012;10(3):1757–1766. Available from: "http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6222582".

[9] Leigh D, Shi H. Forensic Timestamp Analysis of ZFS. In: BSDCan 2014; 2014. Available from: http://www.bsdcan.org/2014/schedule/events/464.en.html.

[10] Leigh D, Shi H. Adding ZFS Events to a Super-Timeline. Digital Forensics Magazine. 2014 August;(20). Available from: http://digitalforensicsmagazine.com/index.php?option=com_content&view=article&id=949.

[11] Leung AW, Pasupathy S, Goodson GR, Miller EL. Measurement and Analysis of Large-Scale Network File System Workloads. In: USENIX Annual Technical Conference. vol. 1; 2008. p. 5–2. Available from: http://www.ssrc.ucsc.edu/Papers/leung-usenix08.pdf.

[12] Agrawal N, Bolosky WJ, Douceur JR, Lorch JR. A five-year study of file-system metadata. ACM Transactions on Storage (TOS). 2007;3(3):9. Available from: "http://dl.acm.org/citation.cfm?id=1288788".

[13] Roselli DS, Lorch JR, Anderson TE, et al. A Comparison of File System Workloads. In: USENIX Annual Technical Conference, General Track; 2000. p. 41–54. Available from: https://www.usenix.org/legacy/event/usenix2000/general/full_papers/roselli/roselli.pdf.

ACKNOWLEDGEMENTS

The assistance and guidance of my supervisor, Associate Professor Hao Shi, has been invaluable throughout this project. I am also grateful to many other staff and students at Victoria University who have provided advice, especially Prof. Xun Yi, Dr Gitesh Raikundalia, Dr Russell Paulet, Robert Moonen and Prof. Nalin Sharda, as well as Dr Ron van Schyndel of RMIT University.

I would also like to thank BSDCan for funding travel to and accommodation at the conference and providing me with an opportunity to speak to the BSD community. Discussions at BSDCan also enhanced this project; in particular Matt Ahrens of Delphix provided me with vital corrections and advice regarding ZFS processes and terminology.

BIOGRAPHY

Dylan Leigh is an honours student at Victoria University, Melbourne, where his research focus is filesystem forensics. He completed Computer Science and Engineering degrees at RMIT University, where he teaches several subjects including Computer & Internet Forensics. He has been a FreeBSD user since 2004.