Zettabyte File System Autopsy: Digital Crime Scene Investigation for Zettabyte File System

Andrew Li
Department of Computing
Macquarie University
Sydney, Australia
andrew.li@students.mq.edu.au

Abstract

Files stored on a computer are managed by the file system of the operating system. When a computer is used to store illegal data such as child pornography, it is important that the existence of the illegal data can be proven even after the data is deleted. In this study, a new functionality is added to the Zettabyte File System (ZFS) debugger, which digs into the physical disk of the computer without using the file system layer of the operating system. This new functionality enables digital crime scene investigators to retrieve any data from the disk, including deleted files. This paper briefly presents an explanation of ZFS internals and describes the approach taken to arrive at the new ZFS debugger functionality. By using this new functionality, we find that the content and all the metadata (file size, owner, creation time, etc) of a deleted file can be retrieved directly from the disk without going through the file system layer of the operating system.

1 Introduction

Files in a computer are stored on digital storage such as a hard disk. A file system is a layer of the operating system that sits on top of the hard disk. It is like a filing cabinet for an operating system. It organises files in a way that enables the operating system to efficiently access files with minimal effort and translate the raw data on the disk to a format that can be understood by humans, like file name and directory name.

The job of a digital crime scene investigator is to carry out computer forensic examination pertaining to legal evidence found in computers and digital storage. The digital storage cannot be modified during the forensic examination, any modification performed on the digital storage evidence is considered as contaminated evidence which cannot be used in court. This creates a need for a tool which can access the digital storage directly, without going through the file system layer in the operating system.

The Zettabyte File System (ZFS) is a new file system type developed by Sun Microsystems[1]. The ZFS file system debugger (ZDB) is part of the ZFS software suite that is used to diagnose and gather ZFS file system statistics. In this study, we present a new feature of the ZFS debugger which allows a digital crime scene investigator to access files directly from the hard disk without intervention of the operating system. Readers may think of the new feature of the ZDB as a tool that can grab a chunk of raw data from the hard disk, and translating it into file and directories which are human readable.

The remainder of the paper is organized as follows. We first present related work in Section 2. Section 3 outlines the ZFS internal which describes the innards of the different layers of the file system. Section 4 describes the design and implementation of the new feature of ZDB. Section 5 presents future work. Finally Section 6 concludes the paper.

2 Related Work

ZFS is still fairly new and there is no publicised forensic tool for the ZFS file system as yet. An initial proposal[2] for a new ZFS forensic tool has been posted to the Open Solaris Security Discuss Mailing List[3] in November 2007. The number of responses from the Open Solaris community has

[1] http://www.sun.com/software/solaris/zfs.jsp
[2] http://blogs.sun.com/efi/entry/proposal_open_solaris_forensic_toolkit
[3] http://opensolaris.org/os/community/security/

indicated that there is a need for a ZFS forensic tool.

File system examination on common Unix and Linux file systems can be done by using open source tools such as The Sleuth Kit[4] and The Coroner's Toolkit[5]. These tools read the hard disk directly and translate the raw data into file system structure that the tool understands. These tools can work on the Linux file systems Ext2 and Ext3, the Microsoft FAT file system, the Berkerly Fast File System (also known as the Unix File System or UFS), the Hierarchical File System by Apple Computer and the Windows NT File System by Microsoft. We have tried applying these tools on ZFS, but it does not work because the ZFS structure is different to all the traditional file systems mentioned above.

Different file systems behave differently in the way they store files, delete files, and the way the file metadata (file owner, group, size, modified time, access control list, etc) is stored. File system forensic examination on different file systems has been explored in *File System Forensic Analysis* [Carrier and Brian, 2005] and *Forensic Discovery* [Farmer and Venema, 2005]. These studies presented detail file system analysis on common file systems like Ext2, Ext3, and UFS, which have provided file system forensic concept toward our new ZDB feature in the present paper.

The article *ZFS On-Disk Data Walk* [Brunning, 2008] uses the ZFS file system debugger (ZDB) and the Solaris Modular debugger (mdb) to walk through the ZFS file system layers. His study uses ZDB and mdb to trace a pointer from the disk to the actual physical file content of a file. The approach is similar to ours in that we perform all activities in ZDB by taking the active uberblock through the various layers of the file system, until the file content and metadata is pointed to by the uberblock is reached. This will become clearer as we explain our new ZDB extension in Section 4.

3 Overview of ZFS Internals

This section presents an overview of the ZFS internals. It will provide sufficient ZFS information for readers to understand the extension that will be made to the ZFS file system debugger (ZDB) in Section 4.

[4] http://www.porcupine.org/forensics/tct.html

[5] http://www.sleuthkit.org/sleuthkit/

The ZFS file system is a new technology that provides dynamic storage which can grow and shrink without the need to re-partition the underlying storage. It does that by eliminating the concepts of partitions and volumes in traditional file systems. A ZFS file system consists of a common storage pool made up of writable storage media. The concept of files and directories are replaced by objects. A complete listing of all ZFS objects can be found in the *ZFS On-Disk Specification* [Sun Microsystems, 2006].

ZFS is comprised of seven components: the SPA (Storage Pool Allocator), the DSL (Dataset and Snapshot Layer), the DMU (Data Management Layer), the ZAP (ZFS Attribute Processor), the ZPL (ZFS POSIX layer), the ZIL (ZFS Intent Log), and ZVOL (ZFS Volume). We will concentrate on SPA, DMU, DSL and ZAP as they are more relevant to our study. For a complete description on all components, please see the *ZFS On-Disk Specification* [Sun Microsystems, 2006].

The Storage Pool Allocator (SPA) component of ZFS contains virtual devices (vdevs) which make up the ZFS storage pools. The virtual devices are described by virtual device label (vdev label). The vdev label contains an array of uberblocks which provide the file system with information necessary to access the content of the storage pool. The uberblock is equivalent to the superblock in traditional Unix file systems, as it contains block pointers that describe blocks of data on disk.

The Data Management Layer (DMU) consumes blocks and groups them into objects. With the exception of low level infrastructure in SPA, everything in ZFS is an object. Objects are defined by structures called dnode. A dnode describes and organizes a collection of blocks making up an object. A file system is described by a group of objects called object sets.

The Dataset and Snapshot Layer (DSL) describe and manage the relationship between object sets. In DSL, object sets are grouped hierarchically into Dataset Directories. Each dataset object points to a DMU object set which contains the actual object data.

The ZFS Attribute Processor (ZAP) is a module that operates the object used to store properties for a dataset, file system object and pool properties. A ZAP object is a DMU object used to store attributes.

The relationship of SPA, DMU, DSL and ZAP components is illustrated below in Figure 1. Below, is a brief description of the remaining

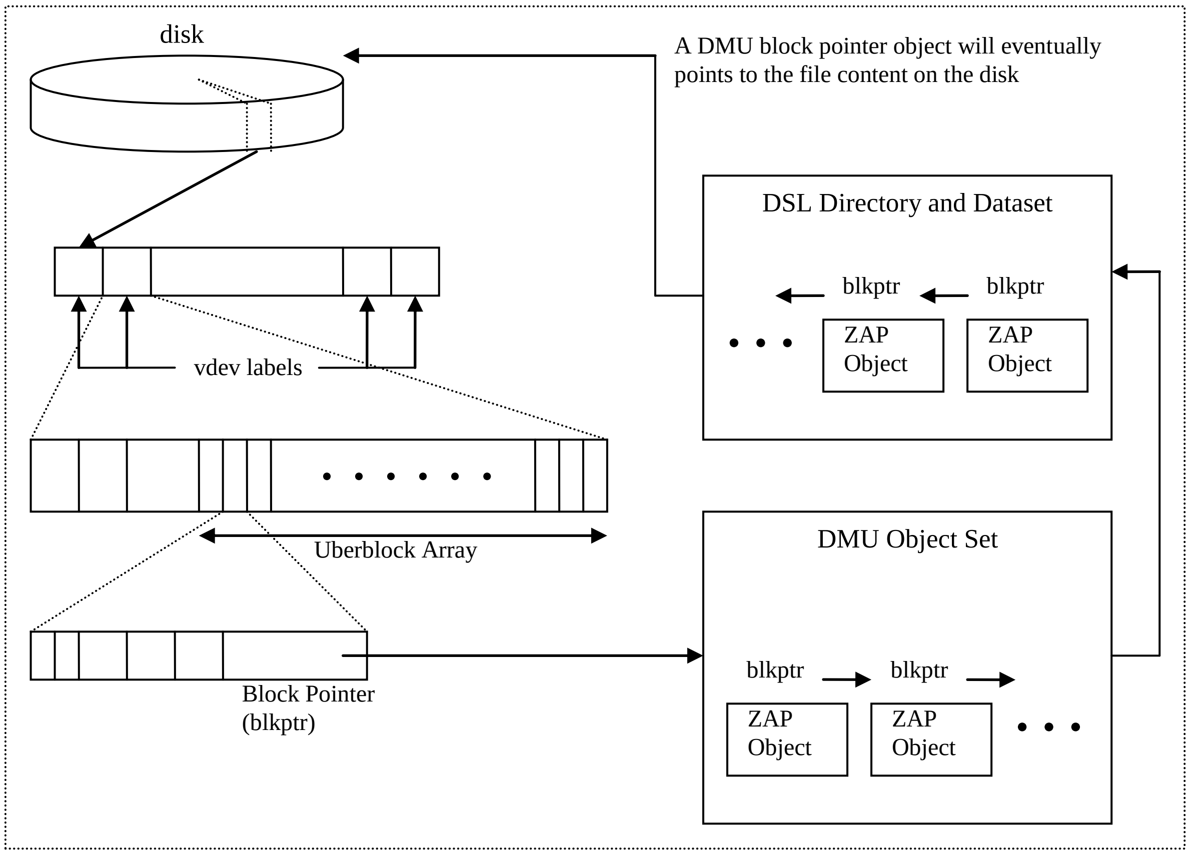

Figure 1 Relationship of ZFS components

components of ZFS which are not directly related to our study, but are included to show the complete architecture oz ZFS.

The ZFS POSIX Layer (ZPL) makes the file system POSIX compliant. It provides a set of POSIX services for the file system.

The ZFS Intent Log (ZIL) records all transactions of the file system. Its purpose is to replay the log records in the event of a machine panic or power failure. This prevents inconsistency in the file system.

ZFS Volumes (ZVOL) provides a mechanism for creating logical volumes in ZFS.

4 New ZDB Feature

In this section, we present the extension we made to the ZFS file system debugger (ZDB) which enables a user to traverse through the file system to get to the actual data stored on the disk.

Section 4.1 specifies the requirement for building the new ZDB extension; Section 4.2 provides a high level overview of the new ZDB extension; Section 4.3 explains the extension in more detail by referring to the OpenSolaris ZFS and ZDB source code. Readers may refer back to Figure 1 to help understand the procedure and the ZFS layout.

4.1 Requirement For ZDB Extension

The source code of ZFS and ZDB are open sourced under the Common Development and Distribution License (CDDL[6]) Version 1.0. The header files of the ZFS structures mentioned in this section can be found at uts/common/fs/zfs/sys/*.h in the OpenSolaris source code[7] and the code for ZDB is found at uts/cmd/zdb/zdb.c. To compile and build any part of the OpenSolaris source tree, a copy of the Sun Studio 12 is required. It can be downloaded from: http://www.opensolaris.org/os/community/tools/sun_studio_tools/sun_studio_12_tools/.

[6] http://opensolaris.org/os/licensing/opensolaris_license/
[7] http://opensolaris.org/os/downloads/on/

4.2 Overview of New ZDB Extension

At a high level, the following steps are carried out by our new ZDB to retrieve the file content of a newly created file without using the file system layer of the operating system.

1. Create a file with known content in the top directory of a mounted ZFS file system
2. Display the file content with the Unix cat command
3. Remove the file that was just created
4. Retrieve the active ZFS uberblock and its block pointer with ZDB
5. Retrieve the dnode for the metadata object set
6. Retrieve the Object Directory dnode and its ZAP object
7. Retrieve the DSL Directory object
8. Retrieve the DSL Dataset object
9. Using the DSL Dataset dnode, retrieve the ZFS file system object set
10. Using the ZFS file system object, get the Master dnode and its ZAP object
11. From the ZAP object of the Master dnode, get the root directory dnode of the ZFS file system
12. From the block pointer of the root directory, find the object id of our target file
13. Using the address stored in the object id dnode, retrieve the block of data directly from the disk and output the raw data. This should match the content of the file we created in Step 1

In summary, the above procedure retrieves a chunk of data from the disk which contains the file content that we are searching for.

In a digital crime scene investigation, this new feature of ZDB will be useful because the investigator can use this tool to examine the disk media without the file system layer in the middle which can intervene with the examination. In a normal day to day operation, when a file is accessed via the operating system through the file system layer, the metadata of the file will be modified. The last access time, modification time, file owner, file size, and permission may change due to the nature of the file system. With the new ZDB feature, the file system is not invoked when the file is being accessed. Therefore, there is no record of the file being accessed, thus nothing on the file system will be updated and the file content and metadata remains untouched.

4.3 Detail Analysis of New ZDB Extension

This section provides a detailed explanation of our new version of ZDB. The new ZDB traverse through the various layers of ZFS using data structures from the ZFS source code. As it is a complex layout, readers may wish to refer back to Figure 1 and the high level overview in Section 4.2 when reading this section.

The first step (Step 4 of Section 4.2) of the new ZDB extension is to retrieve an active uberblock from the uberblock array within the vdev label of the ZDB pool. Each uberblock is stored in an uberblock_t structure defined in the header file uberblock_impl.h[8]. The active uberbock contains a block pointer structure blkptr_t that is used to locate, describe and verify blocks on disk. Block pointers are defined in the header file spa.h[9]. The block pointer contains copies of data virtual address which describes the metadata in a ZFS file system.

The next task (Step 5 of Section 4.2) of the new ZDB is to make use of the data virtual address from the uberblock block pointer. This address points to a location on the disk that stores the metadata which describes the metadata object set. This metadata is described by the dnode_phys_t structure defined in dnode.h[10]. This shows the relationship between the SPA layer and the DMU layer. As mentioned previously in Section 3, almost everything in ZFS is an object and all objects are described by a dnode. The dnode_phys_t contains another block pointer. Similar to the block pointer from the uberblock, this block pointer also contains data virtual addresses. This time the address points to a location on the disk containing an array of dnodes which makes up the metadata object set. The metadata object set is described by an objset_phys_t structure defined in dmu_objset.h[11].

The new ZDB will now retrieve the Object Directory dnode within the metadata object set (Step 6 of Section 4.2). An object directory is a ZAP object, it stores attributes for a ZFS object. The ZAP object used here is described by the structure mzap_phys_t and it is defined in zap_impl.h[12]. The ZAP object contains details of the root DSL directory for the storage pool. It describes all the top level dataset within the pool.

[8] uts/common/fs/zfs/sys/uberblock_impl.h
[9] uts/common/fs/zfs/sys/spa.h
[10] uts/common/fs/zfs/sys/dnode.h
[11] uts/common/fs/zfs/sys/dmu_objset.h
[12] uts/common/fs/zfs/sys/zap_impl.h

The DSL Directory object is stored somewhere in the metadata object set that was retrieved initially from the uberblock. The ZAP object contains the location of the DSL Directory object inside the metadata object set. Recall that the metadata object set is an array of dnode, ZDB will now retrieve the dnode to obtain the DSL Directory object (Step 7 of Section 4.2). This object is described by the dsl_dir_phys_t structure defined in dsl_dir.h[13]. This DSL Directory object gives us the next piece of information for retrieving the DSL Dataset object.

The new ZDB now retrieves the DSL Dataset object using information from the DSL Directory object (Step 8 of Section 4.2). The DSL Dataset object is stored in the structure dsl_dataset_phys_t which is also defined in dsl_dir.h. The dsl_dataset_phys_t contains a blkptr_t. This blkptr_t contains data virtual address of the root dataset of the file system. ZDB will now grab this chunk of data from the disk (Step 9 of Section 4.2) and use it for the next step.

Like everything else, the root dataset of the file system is another object dnode. This dnode contains a block pointer which will lead to the Master node. It maybe necessary to go through a few level of indirection to get to the Master node. The blkptr_t from the root dataset contains a variable dn_nlevels that specifies the level of indirection. If the dn_nlevels is one, it means that the blkptr_t points to another blkptr_t which points to the Master node. Our ZDB will trace through the blkptr_t chain to arrive to the Master node and retrieve the ZAP object of the Master node (Step 10 of Section 4.2).

Once our ZDB gets to the Master node, the Master node contains a data virtual address which points to another array of dnode. Note that this is the second array of dnode, the first array is the array of dnode that makes up the metadata object set obtained from the uberblock. The ZAP object of the Master node contains an object id which tells us where the root directory of the ZFS file system is located. Using the object id, we can locate the root directory from the Master node dnode array (Step 11 of Section 4.2).

The root directory dnode from the Master node dnode array contains a bonus buffer. This bonus buffer is a znode_phys_t structure that contains attributes like time stamps, ownership, and size of the file or directory (Step 12 of Section 4.2). This znode_phys_t structure is defined in znode.h[14], its purpose is similar to an inode for a UFS file system.

We have now arrived to the final step to retrieve the data block on the disk. The root directory dnode from the Master node dnode array contains a block pointer that points to the target file. Our ZDB will use the data virtual address to retrieve a block of data from the disk (Step 13 of Section 4.2). This data will be the content of the file that we are searching for. That completes our extension to ZDB.

In summary, the new ZDB make frequent use of the data virtual address from blkptr_t inside a dnode. This virtual address points to different layers of the ZFS file systems and eventually leads us to the target file we are searching for. Since almost everything in ZFS is an object, just about every step involves dealing with dnode, which is what ZFS uses to store any object.

5 Future Work

In this study, we have introduced extension in ZDB which takes only the active uberblock and traces it back to the data on disk. When investigating a disk taken from a real crime scene investigation, all files which have been stored inside the file system will need to be recovered. The code in our study was developed with this in mind to ease future enhancement. Majority of the code which performs the file system traversal have already been completed in this study. The future release of our new ZDB will incorporate this code into a loop which loops through the array of uberblock so that each uberblock can lead back to the actual data stored on disk, giving the investigator the file metadata and content of every file stored on the disk.

Examination of ZFS snapshots will need to be included in future releases of our ZDB. The technique used on a ZFS snapshot will be similar to what has been done in this study.

Finally, the code from this study could be turned into a set of library function calls. This will enable other system utilities to perform direct file system access and will make the code in ZDB cleaner and easier to maintain, because the complexity has been transferred to the library functions. But the security implications of this will need to be further researched.

To have our new ZDB feature included in future releases of OpenSolaris, it will need to go

[13] uts/common/fs/zfs/sys/dsl_dir.h

[14] uts/common/fs/zfs/sys/znode.h

through a process like all other open source projects. All code will need to be posted to the OpenSolaris community for code review. Once the code is reviewed by the OpenSolaris community, the code will need to be submitted via an online application form[15]. After submitting the code, the code will go through another code review process by developers from Sun Microsystems. See the Improving OpenSolaris[16] webpage for a complete description of the code submission process.

6 Conclusion

The work described in this paper presents a proof of concept that a digital forensic tool for ZFS is achievable, unlike Ext3 and UFS2 where the relationship between the file and the data on disk is removed when a file is deleted, making it harder to trace the data back to the disk. File retrieval is done by using our new feature in ZDB, which travels through the various layers of the ZFS file system until it reaches the target file stored on disk. This means that the data on disk is being accessed directly without intervention from the file system layer operating system.

This new feature of ZDB is designed to help a digital crime scene investigator to retrieve evidence from an operating system with a Zettabyte File System. It enables investigators to retrieve data that has been deleted or hidden, which cannot be seen under normal operating system operations. Our new ZDB achieves this by tracing through virtual addresses stored in ZFS block pointers to dig into the ZFS file system layers until the target data is reached. By doing so, the file system layer of the operating is not invoked and the data stored on the disk can be accessed directly. This enables investigators to gather reliable crime scene evidence.

References

Bruning, Max. June 2008. *ZFS On-Disk Data Walk*. In OpenSolaris Developer Conference. June 25-27, 2008 Prague.

Carrier, Brian. March 2005. *File System Forensic Analysis.* Addison Wesley Professional.

Farmer, Dan. & Venema, Wietse. 2005. *Forensic Discovery*. Addison-Wesley Professional.

Sun Microsystems, Inc. 2006. *ZFS On-Disk Specification*. Sun Microsystems, Inc

[15] http://bugs.opensolaris.org/

[16] http://opensolaris.org/os/communities/participation/