March 21, 2017

An efficient method to avoid path lookup in file access auditing in IO path to improve file system IO performance

Arun Vishnu P K
Hewlett Packard Enterprise

Ramesh Kannan K
Hewlett Packard Enterprise

Rajkumar Kannan
Hewlett Packard Enterprise

An efficient method to avoid path lookup in file access auditing in IO path to improve file system IO performance

Abstract:

One of the biggest challenges in metadata management schemes that sit outside the filesystem layer is their ability to index meaningful path information of files that are being referenced in an external system like a database or in a metadata journal file. Path to a file is a critical requirement that allows both meaningful interpretation of the locality of the file and its metadata and also secondly allows for more efficient user mode services that can transform the file or its metadata. Additionally path information is very essential in compliance systems where audit logs need to tell what happened to a file and where it is located. However when the data path is being audited from layers such as protocols, it becomes harder to reconstruct the entire path information for all the files given that the protocol layers do not directly integrate with the underlying Filesystem. The protocol layers would then need to rely on system cache to get the path data and sometimes this may not be possible making it required for the protocol to actually do an expensive reverse path walk, reconstructing the path. This actually heavily degrades the performance of the system. In this paper we discuss a mechanism that allows us to record enough information about the file using the unique ID of itself and its parent in the protocol layer such that if and when required the path information can be reconstructed based on a reliable reverse lookup in a database or a file based journal system. The idea is to have enough information to reconstruct the path at a later time and outside the system where the information was initially originated from. The paper also talks of keeping this system consistent under all conditions.

DESCRIPTION OF INVENTION:

Problems Solved:

File access auditing (FAA) is an essential requirement for high secure data centers and also for organizations which conform to various regulatory compliance requirements. Since FAA happens in the IO path for every IO request, it is a critical that the auditing framework takes less resources to log audit information.

In general an audit log contains information like client details, operation details, operation status and the file object details. One of the crucial information in audit record is the absolute path of the file object in order to uniquely identify the file object on which the IO operation is performed.

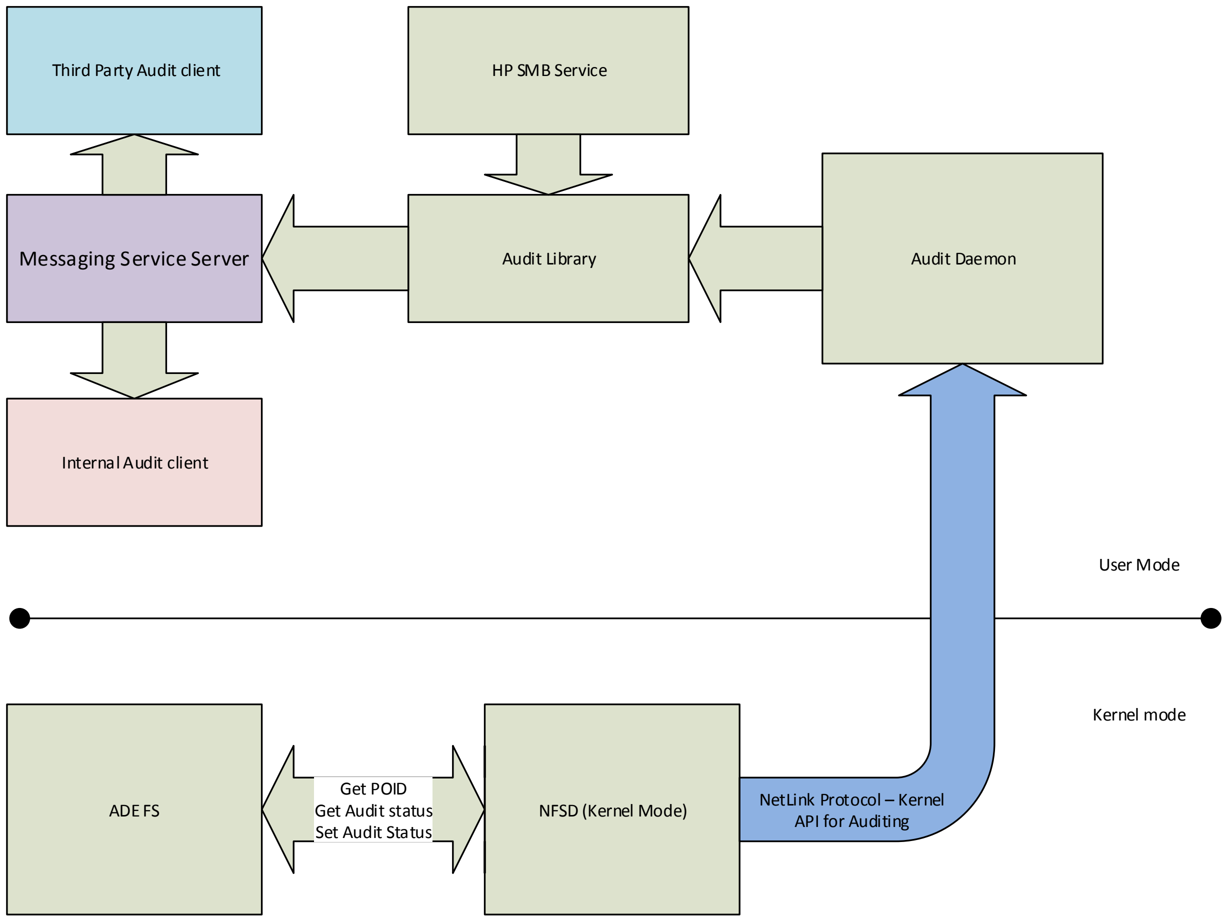
Finding the absolute path needs traversing the file path from the location of file being audited till the root of the filesystem/mount point. In the typical Linux kernel, an object name dentry carries the name information and each dentry maintains a pointer to its parent dentry. A module which needs to form the path has to lock the dentry and then traverse to the parent dentry and do complex string operation to form a complete file path.

The invention proposes an efficient approach where the audit framework will never do a path lookup for logging audit record but it logs enough information sufficient for a post-processing tool to build the path when needed. The audit information can be plumped to a database to make the post-processing efficient.

Prior Solutions:

Description:

The below block diagram explains the different components involved in file access auditing and the audit event flow. The diagram covers both NFS and SMB protocols.



Further in the disclosure we selected NFS protocol auditing on the ADE file system to explain the proposed method. We recommend to turn on auditing for the desired shares before any operation. But enabling auditing in on a shares which is currently under usage is handled.

Along all mandatory information like, client, user, operation and start, we record three more items.

1) Object name
2) Object ID
3) Parent ID

Object name is the name of the object being audited. It can be name of a file or a directory. Object ID is a unique ID which represents the object. The parent ID is the unique ID of the parent directory where the current object located.

The Object ID will be provided by the ADE fs which is known as the permanent object ID (POID). The POID will be a unique identifier of the file/directory throughout its life and will not change even during rename operation. POID is 128 bit long and will not be reused in the life span of the filesystem.

In the context of auditing a Linux kernel based NFS protocol, a dentry is associated with every operation. The dentry carries name, inode and parent inode as standard and makes our approach much more efficient.

All create operations are recoded by default to keep the audit logs self-sufficient for any post-processing.

Below an example where a share “fs/vfs/fstore/nfs1” is being audited.

When the share get mounted on a client we log the share-root information

Name	POID	Parent POID
nfs1	10001	10000

Created directory dir1

Name	POID	Parent POID
Dir1	10002	10001

Created file foo.txt inside dir1

Name	POID	Parent POID
Foo.txt	10003	10002

Created file foot.txt inside root of the share

Name	POID	Parent POID
Foo.txt	10004	10001

Here we have two objects named foo.txt is available but with unique POID and parent POID.

Path Formation:

The audit events generated from the protocol server will be transferred to audit client, where the post process happens. The audit client will form the file path on reception of an audit event. IN general audit clients are dedicated servers which process and maintain the audit logs for future references.

Referring the above example, when audit client receives event

Name	POID	Parent POID
Foo.txt	10003	10002

It starts forming the path. First it takes the Object name from current event, which is "**Foo.txt**". Now the client will do a look up on the past logs to resolve the name for the parent POID, which is 10002. The past log

Name	POID	Parent POID
Dir1	10002	10001

Will provide you the name for 10002, which is Dir1 and path become **Dir/Foo.txt**. But the lookup will not stop as this event has a valid parent POID which is 10001. The process is repeated to figure the name for 10001 and path become **nfs1/dir1/foo.txt**, which is a complete path from share root.

Handle rename:

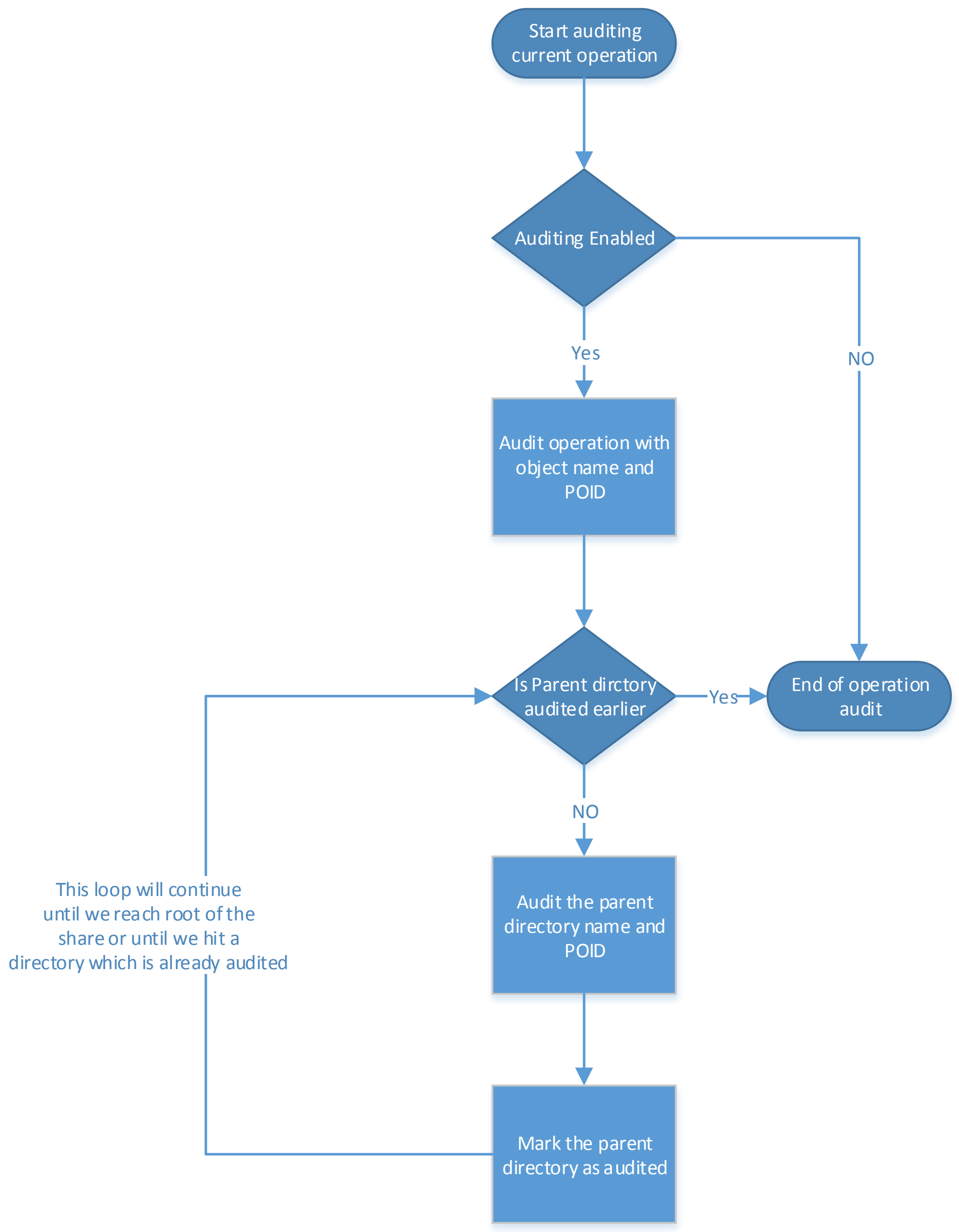
Rename is another challenge with a file path management. As per the above explained mechanism, the POID which represents a file will remain same even when the file is renamed. In order to make the audit information self-sufficient for future auditing, we will log old and new name on a successful rename operation.

But directory rename is much complicated as it affects the entire files and directory inside the directory being renamed. The FAA modules does not do any extra logging on a directory rename operation. But the audit client need to do address by a runtime name look up to locate all past logs affected by the rename and update the path against them. The look will be based on the POID of directory got renamed.

Enabling auditing on share with data:

The best practice is always to enable the auditing before starting any operation on the share. But this is not guaranteed. When auditing get enabled on share which already has a data, simply logging POID and parent POID will not be sufficient to build the complete path as we might not logged all directories involved in forming the path.

In order to handle this we maintain a flag on the inode. This flag be initialized as zero on newly allocated inode and will be set to one when we log the object. Also we traverse the parent's parent and so on in case the flag is not set. Below flow chart explain the control flow:


Start auditing current operation
Auditing Enabled
Yes
NO
Audit operation with object name and POID
Is Parent dirctory audited earlier
Yes
End of operation audit
NO
Audit the parent directory name and POID
Mark the parent directory as audited
This loop will continue until we reach root of the share or until we hit a directory which is already audited

Advantages:

1) Audit logs generated are self-sufficient to create the complete path. Audit clients need not apply any ID to path conversion logic to resolve the path. Also audit clients will not contact ADE fs to resolve a POID to path conversion.
2) Delete and rename operations are handled without any special workflow.
3) Performance impact on protocol IO is minimal.
4) Since the path is not logged in audit log, constructing the path in every IO is completely avoided thus the impact on IO due to audit logging is reduced to a great extent.

5) Not logging the complete path also avoids additional IO (to write the audit log record) and reduces the audit log size.

Disclosed by P K Arun Vishnu, Ramesh Kannan K and Kannan Rajkumar, Hewlett Packard Enterprise