A File Provenance System

Salmin Sultana
Purdue University
ssultana@purdue.edu

Elisa Bertino
Purdue University
bertino@cs.purdue.edu

ABSTRACT

A file provenance system supports the automatic collection and management of provenance i.e. the complete processing history of a data object. File system level provenance provides functionality unavailable in the existing provenance systems. In this paper, we discuss the design objectives for a flexible and efficient file provenance system and then propose the design of such a system, called FiPS. We design FiPS as a thin stackable file system for capturing provenance in a portable manner. FiPS can capture provenance at various degrees of granularity, can transform provenance records into secure information, and can direct the resulting provenance data to various persistent storage systems.

Categories and Subject Descriptors

C.5 [**Computer System Implementation**]: General

General Terms

Reliability, Security

Keywords

Data Provenance, Operating Systems, File System

1. INTRODUCTION

Provenance refers to the history of ownership and the actions performed on a data object. Provenance has been widely used in the scientific and grid computing domains in order to document workflows, data generation, and processing. For scientific experiments, provenance contains input datasets, experimental procedures, parameters, etc. information which are sufficient to enable reproduction and validation of results [1]. A number of domain specific provenance systems, such as Chimera [2], ESSW [3] have been developed for various experimental systems. They capture provenance for scientific data and record provenance at the semantic level of application. Other application level provenance systems capture provenance at the level of business objects, lines of source code or other units with semantic meaning to the application. The fundamental problem with domain-specific approaches is that the data object and the provenance are managed by two separate data management systems (i.e. data by file system and provenance by database system) [5]. Consequently, data and provenance do not remain tightly coupled and thus the provenance may not completely reflect the data processing history. Moreover, the application specific provenance systems require the users to manually track provenance by building appropriate provenance collection tools. In this context, it has been suggested that the provenance collection should be a responsibility of the operating system (OS) that can also generate system level provenance meta data [5].

In this paper, we propose the design of a *file provenance system*, named as FiPS, which is a file system that not only manages files but also transparently captures, stores and manages the file provenance. FiPS autonomically collects sufficient metadata in order to recreate a file i.e. to reenact the series of actions that generated the content of the file. We design FiPS as a thin layer operating between the *Virtual File System* (VFS) and the underlying file system. In contrast to a system-call based provenance approach [5], we intercept file system calls passed through the VFS layer and then generate provenance records. System call level approaches often fail to see how a system call activity is translated into multiple actions in the lower layers of the OS. Memory-mapped I/O can only be traced at the file system level. In addition, server-side operations of network file system (NFS) are performed directly in the kernel, not through system calls.

To make the provenance system useful, we also incorporate *flexibility* and *security* into our design. To be flexible, FiPS allows provenance to be captured based on fine-grained conditions. This feature also helps capture and store only desired information, making the solution efficient in space and time. To be *secure*, FiPS can apply appropriate security function on a file provenance in order to protect it from unauthorized access. The system can also write provenance to a local or networked storage. Our implementation is still a work in progress. To layer FiPS on top of any conventional file system, we implement our functionalities on the stackable wrapper file system *Wrapfs* [6]. We use in-kernel Berkeley DB [4] to manage granularity and security policies.

2. DESIGN OBJECTIVES

Based on the features of the existing file system provenance solutions and their limitations, we outline the following design goals required to build a robust file provenance system:

Portability: The file provenance system should capture provenance for any file system, without modifying the OS or the provenanced file system. FiPS is designed as a stackable filesystem and thus can be layered on top of any conventional file system. In addition, FiPS is to be implemented as a kernel module which requires no kernel modification in order to collect provenance.

Efficiency: It is essential that provenance capture and management do not add too much overhead to the file system operations with respect to space and time. The provenance system should provide fast, high-throughput provenance operations in order to avoid impacting operating system and application performance. The system must record enough provenance metadata to serve the desired purpose but not any unintended information. Hence, it should distinguish between data objects that are required to be provenanced and data objects that are not. On the other hand, capturing provenance information by intercepting system calls often misses information about how a system call activity is translated into multiple actions in the lower layers of the OS. Also NFS servers cannot work with system call level logging since they operate directly in kernel, not through system calls. Finally, it is more natural to manage file system provenance in terms of file system instead of system calls. We design FiPS as a thin layer between the Virtual File System (VFS) and any other file system which results in space and time efficiency.

Flexibility: Traditionally provenance-aware file systems collect a specific detail of the provenanced objects which in some cases are inadequate. On the other hand, PASS captures a good amount of information for each provenanced object which results in a huge volume of provenance records when integrated with an end-to-end provenance solution. To be productive, the provenance system should be flexible by supporting a wide range of fine grained policies on provenance collection. The policies should allow one to specify the granularity of provenance information to be captured based on applications, users, file names, attributes, etc.

Security: The system must capture and store provenance in a way so that the information is kept secure against attacks and subversion. Besides, the provenance information may require access control to be protected from unauthorized user access. Our in-kernel system design provides stronger security. Moreover, we incorporate security policies used to apply appropriate security mechanisms (e.g. encryption, signature) while sending provenance to persistent storage.

Redundancy Elimination: Recording provenance for file system operations may result in large amount of data which are difficult to store, manage and query efficiently and effectively. Hence, mechanisms for provenance pruning or compression should be provided, e.g., for replacing a part of the provenance graph with the end result of the modification.

Queries on Provenance: Collecting data provenance is not useful unless the provenance can be accessed and utilized easily. Hence, the file provenance system must provide support for a structured storage of provenance which in turn will facilitate provenance queries. The management system should also respond quickly to the relationship queries leading to the generation of ancestry or descendancy graphs.

3. FIPS - THE PROPOSED PROVENANCE FRAMEWORK

FiPS is designed to collect and store provenance for the data objects at a file granularity. However, the provenance can be tracked easily at finer or coarser granularities by defining the granularity policies accordingly. We define the *provenance of a data object (file) as the documented history of the actors, process, operations, inter-process/operation communications, input/output data, the hardware and OS environment related to the creation and modification of the object.* The complete provenance of a data object form a directed acyclic graph (DAG), referred to as the *provenance graph.*

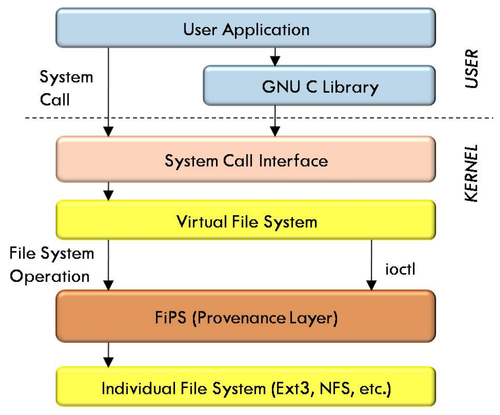
We design FiPS as a stackable file system [6] that can work on top of any underlying file system. Figure 1(a) shows how FiPS is placed between the *Virtual File System* (VFS) and any other file system. In a traditional file system, the system calls related to file operations invoke VFS calls which in turn invoke underlying file system procedures. When integrated, FiPS intercepts the VFS calls, extract arguments and other necessary information from kernel data structures, and translates them into in-memory provenance records. While recording the provenance metadata for a data object, the level of details to be stored is determined according to the associated granularity policy. At the end, FiPS sends the in-memory provenance records to a persistent storage, either to a local disk or to a remote server.

Figure 1(b) shows the detailed architecture of the FiPS layer. The key components are: the provenance *logger* which records the provenance metadata and translates them into in-memory provenance records, and the provenance *writer* that stores in-memory records into persistent storage. To control the granularity and security of provenance, the components act upon two databases: the *granularity policy DB* and the *security policy DB*, respectively. Below, we briefly discuss the components and the policy databases:

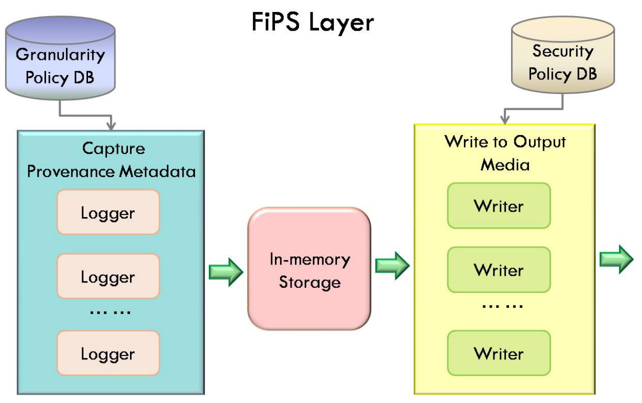
Logger: The role of a logger is to record provenance data or VFS calls and then to ensure that these records are passed to the destination file system for attachment to the appropriate files. FiPS supports multiple *logger* threads where the intercepted VFS calls pertaining to an application/process will be handled by one *logger*. Such a design will increase the speed of the provenance tracking for simultaneous processes and make it easier to deal with the granularity policies.

As in PASS, all files and processes in our system are considered provenanced kernel objects. The logger generates a provenance record for each provenance related VFS call and stores the record in an intermediate storage. This intermediate storage may be a buffer or in-kernel Berkeley Database (KBDB). The metadata to be captured in the provenance record may vary depending on the associated granularity policy. For example, a provenance record may include a subset of Process ID, input files, User ID, command line, kernel version, etc. information. In addition, granularity policies may specify when the system should not capture provenance (e.g. for a particular application, file, etc.) or when the intermediate results should not be recorded, etc. Unlike PASS, our system can distinguish between provenanced and non-provenanced file systems and does not retain provenance in-memory for non-provenanced files.

Granularity Policy DB: This database stores the granularity policies which determine how much information has to be captured as provenance. The policies may be associated with a process, application, user, file or file attributes. The policy database may be populated with policies during



(a)



(b)

Figure 1: (a) FiPS as a stackable file system. (b) Architecture of the FiPS layer.

the OS start or with application specific policies when the application executes.

We design this database as an in-kernel Berkeley DB. The policy database has the *object* and the unique *object ID* (e.g. file and inode) as the key and a 4 or 8 byte value containing the policy bits as the data. The policy bits include: (i) OBJECT - specifies the object for which the policy is defined; (ii) ATTRIBUTES - the set of attributes of the respective objective, used to check the condition in the policy; (iii) ACTION - specifies the action taken if the condition is satisfied. The actions include *capture-all*, *no-capture*, *capture-set*, *no-intermediate-data*, etc.

Security Policy DB: This database, also designed as a KBDB, stores security policies which are applied to the in-memory provenance records when they are stored in a persistent storage. The design of the database is similar to the granularity policy DB with a difference in the taken actions. The actions here include applicable security mechanisms, such as *encryption*, *signature*, etc.

Writer: A writer writes out the in-memory provenance records to the persistent storage. The output media may be a regular file on local disk, a raw device or a socket. When writing to a socket, the writer connects to a TCP socket at a remote location and sends the provenance over the network. This is particularly useful when storing provenance in a remote NFS server. To ensure desired security for the provenance, the writer first finds out the security policy associated with the provenanced data object, performs the security action and then writes the secure provenance to the destination storage. The system activates multiple writers to fasten the performance.

In addition, we include a *redactor* which compresses and prunes long term history for older files. The redactor may be a user level or a system process. Provenance pruning/compression may be managed by the system wide retention policies or user-specified policies for a file, a group of files, or a directory. The redactor periodically examines the file system and uses the policies to decide when and which files can be left to carry on with a compressed provenance graph. This approach allows users to indicate the files for which it is required to keep detailed provenance and versioning information.

4. CONCLUSION

We present the modular design of a low-overhead and flexible file provenance system (FiPS) that collects provenance by operating below the VFS layer. Unlike system call tracing, FiPS can handle memory mapped I/O and NFS server operations easily. FiPS incorporates granularity policies which provides flexibility in provenance capturing and security policies to ensure the desired security. Currently, we are implementing the system which is bringing forth various design and implementation issues. In future, we plan to extend the system to collect provenance at virtual machine monitors.

5. ACKNOWLEDGMENTS

The work reported in this paper has been partially funded by NSF under Award 1111512 “Privacy-Enhanced Secure Data Provenance”.

6. REFERENCES

[1] P. Buneman, S. Khanna, and W.-c. Tan. Why and where: A characterization of data provenance. *ICDT*, 1973:316–330, 2001.

[2] I. Foster, J. Vöckler, M. Wilde, and Y. Zhao. Chimera: A virtual data system for representing, querying, and automating data derivation. In *Proc. of the Conference on Scientific and Statistical Database Management (SSDBM)*, pages 37–46, 2002.

[3] G. Janée, J. Mathena, and J. Frew. A data model and architecture for long-term preservation. In *Proc. of the conference on Digital libraries*, pages 134–144, 2008.

[4] A. Kashyap. File system extensibility and reliability using an in-kernel database. Technical Report FSL-04-06, Master’s Thesis, Stony Brook University, 2004.

[5] K.-K. Muniswamy-Reddy, D. Holland, U. Braun, and M. Seltzer. Provenance-aware storage systems. In *Proc. of the USENIX Annual Technical Conference*, 2006.

[6] E. Zadok and I. Badulescu. A stackable file system interface for linux. Technical Report CUCS-021-98, Columbia University, 1998.