Flexible Access Control Lists in SFS

Names omitted to protect the innocent

December 6, 2001

1 Abstract

Sharing files with other users in a secure manner is a complex process under current Unix file systems. Permissions must be set appropriately, and providing access to specific users outside of the network requires privileged access. We present a flexible system of file sharing, built using the Self-Certifying File System framework. This allows unprivileged users to share directories with outside users specified by an access control list (ACL) that contains public keys. These flexible ACLs on top of SFS allow for a secure and convenient method for users to share files. Non-priviliged users can easily maintain access on their files without the intervention of an administrator with the same confidence as exporting and accessing their own files through SFS.

2 Introduction

The Self-Certifying File System (SFS) was designed to allow people to securely share and access files. To have the potential to scale such a system to span the Internet, key management and file system security were explicitly separated[4]. Thus by design, SFS maintains the same permissions structure as the system that is running the SFS server. To share files with a specific user, most Unix based file systems require that the user have an account on the server. If this is not the case, creating an account for a new user and adding him to a particular group requires administrator level privileges. This consequently limits the freedom with which normal users can share their files.

We outline a modification to SFS that would allow more flexible file sharing. A normal user is able to maintain his own access control list (ACL) for a directory to share its contents with anyone he chooses, without requiring intervention of the system administrator. Each ACL contains the public keys and permission settings (read and/or write) for those visitors that the user wishes to grant access. These visitors can then access these files from any SFS client with the same transparency as accessing ones own files through SFS.

Building this system on top of the existing SFS framework affords it the benefits of self-certifying pathnames and the authentication protocols of SFS. The modularity inherent in the design of SFS that separates key management from the file system allows us to similarly divide our modifications into two main groups: registering the public keys of visitors with the server's authentication database (key management) and checking visitor permissions when requesting files (file system security). Flexible ACLs on top of SFS allow for a secure and convenient method for users to share files. Non-privileged users can easily manage the visibility of their files to other users without the intervention of an administrator and with the same confidence as exporting and accessing their own files through SFS.

3 Related Work

Networked Filesystem[5] (NFS) is a protocol file system allowing users on client systems to mount remote file systems over a network. It does not make many security claims, and requires that users who access the files have accounts on the file server. Our work differs in an outside user's ability to read and write files securely from remote locations.

Self-Certifying Filesystem[4] (SFS) is a secure networked file system that relies on public/private cryptographic keys to enforce the security of file access. It provides functionality similar to NFS with added security. It also eliminates centralized control, which is heavily used for other authentication purposes such as in Verisign's central certification authority. SFS supports anonymous read and write access which allows users to share files with all external users, however, it does not allow for restrictions on outside users. There are no provisions for restricting access or write permissions to a limited set of outside users. This is the functionality SFS-ACL adds.

The Andrew Filesystem[2] (AFS), developed at Carnegie Mellon, is a distributed filesystem that uses ACLs and kerberos authentication to offer considerably more flexibility in directory protections than regular UNIX. Using ACLs, AFS can give one set of users read access to a directory, one set of users write access, and another set no access at all. Currently, AFS only allows ACLs to be applied to directories and not to files. Owners are allowed to modify the ACL list as well as users given permission to administrate that partiular ACL. Any user is allowed to create groups, but groups cannot contain groups. Sharing files amounts to listing the allowed users in the ACL, thus limiting the scope of allowed individuals to current users of the AFS domain where the shared directory is located. Sharing outside of the domain requires either sharing with everyone, or at the very least sharing with authenticated users of other domains. In all cases, a file system client must have an account on the system to access shared files. This differs from SFS-ACL which does not require visitors to have user accounts to be able to read and write exported files.

CODA[3] is a file system based on AFS. It adds the ability to use shared files on a disconnected workstation, but it does not change the access control structure of AFS.

Microsoft Windows and Samba[1] utilize the Server Message Block[6] (SMB) protocol (newer versions call this CIFS or Common Internet Filesystem) that allows the sharing of files and resources over low-level network protocols. Sharing files under Windows operating systems require administrator privileges to add either current users or guests to ACLs. Samba allows users to integrate Unix and Windows systems into a unified networking environment. Permissions to share directories are independent of the underlying Unix system the software is installed on. Samba thus runs relatively independent of the underlying filesystem. While the software does take security into consideration, it still has several known vulnerabilities and remains a work in progress. In addition, setting up and negotiating the Windows and Unix filesystems is a challenging task. SMB differs from SFS-ACL in its requirement on administrative priviledges to export files to visitors.

4 Design

In designing our system, our topmost goals were usability and transparency. Users should be able to share files with minimal effort, while resting assured that security of the overall system has not been compromised.

Our proposed modifications to SFS cover two main areas: user authentication and file system security. In this section we first define the parties involved in our system and then describe a usage scenario. We follow with our proposed changes to the SFS architecture in the authentication subsystem and the file access mechanism.

4.1 Participants & Usage Scenario

4.1.1 Participants

Under our proposal, there are two interesting parties. The *guardians* and the *visitors*.

- A guardian is a user who has an account on the SFS file server. He has enough privileges to add his own public key to the SFS file server's authentication list. This key allows him to remotely access exported directories through SFS. He would like to grant access to these files to his friends, but his friends do not have login accounts on the SFS server. We will call these friends *visitors*.

- A visitor is someone who wishes to access files owned by a user on the file server (a guardian). Each visitor has his own public/private key pair that he can use to access files on the SFS server once he has been properly registered. Visitors may or may not have their own user accounts on the server.

4.1.2 Usage Scenario

In our scenario, guardian G_n grants read/write access to visitor V for directory (D) ~/sharedfiles. D is already exported to G_n using SFS. Here are the steps of the process, including key registration, and file access.

- V generates a public/private key-pair, and gives the public key PK_V to G_n via some verifiable channel(perhaps a PGP signed email).
- G_n creates a file in D called `ACL` and adds the line: "PK_V" to the file, followed by whatever permissions G_n wishes to grant V.
- G_n adds PK_V to the authentication server's table of authorized public keys. The authentication server adds a mapping from PK_V to G_n if one does not already exist.
- V logs into the SFS server, and authenticates himself with PK_V/SK_V. The authentication finds PK_V in its database and gives this session the credentials $\{PK_V\}$.
- V reads and writes files in D. When the SFS server is asked to perform these operations on files in D, it must check that the owner of the ACL, G_n has permission to read/write the files, and that PK_V is in the file D/`ACL` and also has read/write permissions specified in the file. If both of these are true, then the SFS server allows the operation to proceed.

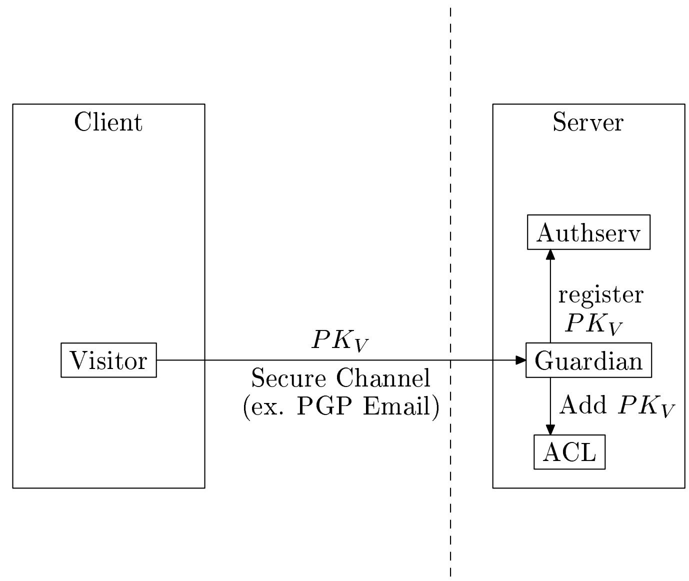

Figure 1: Visitor Key Registration

4.2 User Key Mangement

4.2.1 Key generation

We first address key creation, since a user must be able to generate key pairs, independent of auth server registration. Users must be able to generate and store their public key to send to a guardian if they wish to access another user's files. A new method of key registration was added to allow guardians to register visitors, as shown in figure 1. The new protocol remains similar to the original, though now a user can register the public keys of other users with the authserver, independent of whether or not these users have accounts on the machine running the SFS server.

4.2.2 Auth Server

The format of the database file stored in the auth server was changed to allow for the new functionality. SFS normally maps a public key to a set of credentials (a username). This was modified to insert a placeholder value in lieu of a username if a visitor is registered who does not have an account on the system. If, however, the visitor does have an account on the system, then the guardian need not register the key with the

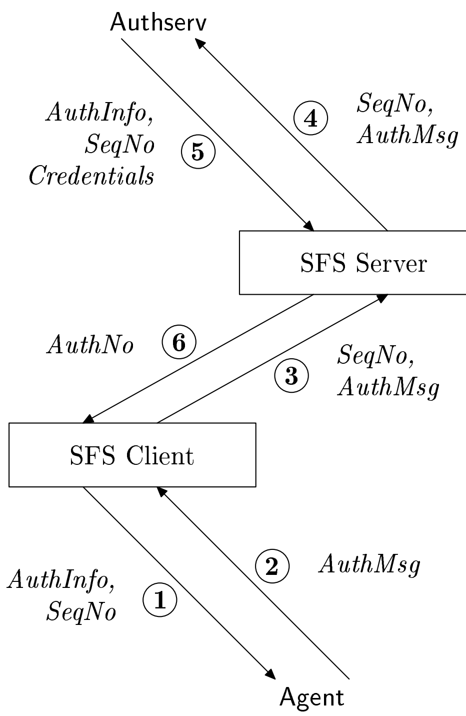

Figure 2: User Authentication Protocol

auth server, as a record will already exist for the visitor.

4.3 Authentication

During the authentication process, the auth server returns a session key, listing the credentials for the user. The user's credentials are returned to the server daemon along with an *AuthID* and a sequence number. For a user with an account on the system, these credentials are his username and public key. A visitor, however, will only have their public key as well as a meaningless placeholder value for their username, indicating that he has no account. The authentication process is thus identical to that of standard SFS, save for the fact that the public key is now included. The authentication process is shown in figure 2.

4.4 File Access

A guardian wishing to share his files places an ACL in the shared directory. In this file, he lists the public keys of each visitor, followed by the standard Unix permission abbreviations (`r`, `w`, etc) that he wishes to grant each visitor. The guardian can also opt to use the character `*` followed by a set of permissions to enable anyone using SFS specific access to that directory. This allows a user to export a single directory through SFS and create simple ACLs for this directory, while allowing for more varied permissions within subdirectories. An ACL can only be modified by the guardian, though all other files in the directory are accessible to his visitors.

Our modified SFS server performs the access checks at the point just before SFS hands RPC requests to the NFS loopback server. For each visitor request from the SFS-Client, the SFS-Server needs to verfy that the user has enough priviledges to perform the operation. First, the SFS server checks the ACL in the working directory to see if it contains PK_V. If no ACL exists in that directory, the request is denied. Otherwise, if it passes this test and the visitor is listed as having sufficient priviliges to perform the operation, the server forwards the request to the local NFS server using the UID of the guardian.

The SFS-Server issues the NFS request using the UID of the owner of the ACL file. In most cases, the owner of the ACL file should be the guardian of the shared directory. By using the guardian's UID for the NFS request, we give an approved visitor the credentials it needs to access files in the shared directory. In some special cases, the guardian might not want to grant access as himself, in particular, if the server administrator would like to export `/home`, he can place an ACL file containing `*, r` to allow all visitors to read the home directories, but change the owner of the ACL file to be *nobody* so that the NFS loopback server actually services the requests. We considered allowing multiply guardians per directory and specifying the UID to use in the ACL file, but this opens potential security risks. If the ACL file is writable my multiple guardians, one guardian could grant access to a visitor as a different guardian. This would allow a visitor to masquerade as a second guardian, in an unintended directory as long as the second guardian also listed the visitor as his own.

5 Implementation

5.1 Key Management

Two general changes had to be made in order to implement our new requirements for key management: first, a method by which users on a server can register visitor keys with their authserver; and second, an extension to the authserver's database format, in which a public key need not have a corresponding valid username on the system.

The key registration was accomplished by creating a new RPC procedure, *VREGISTER*, that was a parallel to the standard *REGISTER* procedure save that the authserver knew the calling user was not registering himself, but rather a visitor. Our new database format is still a simple flat text file, except that, for each public key, the following field now may or may not hold a valid username. For visitors without accounts on the system, this field will instead hold a meaningless placeholder value instead.

5.2 Auth Server

In addition to parsing the extended database format as above, the authserver was modified so that it passes a public key along with that user's username to the SFS server daemon at the time of user authentication.

5.3 File Access

The process for file access is the most significant change in our implementation for ACLs, and thus required the most extensive code modifications. Our scheme for checking ACL access control is implemented as follows:

- For each disk volume, the SFS server daemon maintains a global table mapping filehandles to the filehandles of their parent directory. Our system requires such a data structure in order to locate the relevant ACL file for every file operation and no natively implemented method allows us to do this efficiently.

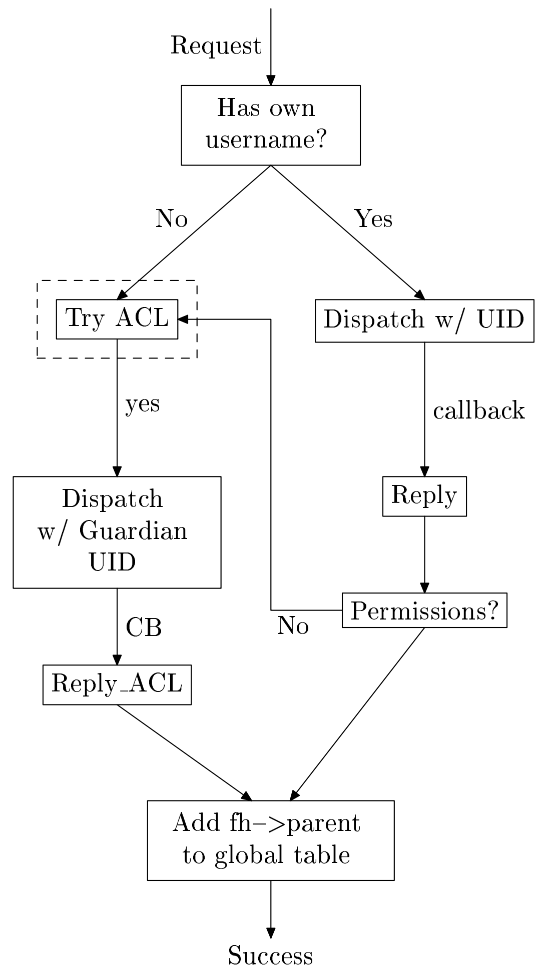

Figure 3: Flow of operation for file operations

- When a client first requests access to a file, through any of a number of NFS calls, the server uses a two-step procedure to determine if that client is authorized for the file.

  1. The SFS server first checks if the client has a real username/UID, that is, if the user attempting access has an account on the server; if so, the server simply passes the request along to NFS, using the real UID (figure 3).
  2. If the first test is not successful, either because the user does not have an ac-

count, or because the desired access is not authorized under that account, the SFS server checks if an ACL file exists in the relevant directory. For directory operations, the filehandle for this operation is readily available. File operations, on the other hand, require that we find the parent directory's filehandle, which is accomplished using the globle table described above. If an ACL file is located in the directory, the system checks if her public key is listed in the ACL and also that she has sufficient permissions for the requested operation. If both of these requirements are satisfied, the SFS server makes a request to NFS using the guardian's UID, obtained from determining the owner of the ACL file. The result of this call—success or failure—is passed back to the client. (figure 4)

On a successful NFS operation involving a new filehandle, an entry is added to the global table, mapping the returned filehandle to the filehandle of the parent that was used in making the request.

- A cache of filehandles to guardian UIDs is maintained for each session. After the ACL in a directory is examined to determine the proper guardian UID to use, if access is granted, the UID is added to this cache along with the filehandle of the directory. If access is forbidden, the filehandle is mapped to a UID of value 0. We maintain the cache as a linked-list and assume that more-recently used filehandles will be accessed sooner in the future and thus add or move the most recently accessed filehandles to the head of the list. Each entry is also assigned a timeout value, after which the ACL is re-examined in case the permissions have been altered. This timeout value protects our system against replay attacks, in which the attacker tries to use a filehandle that refers to a file for which he no longer has ACL permissions, because after the timeout period—configurable by the user—the filehandles are marked as dirty, and will be rechecked at the next access. The timeout period also allows ACL updates to propagated to the server.

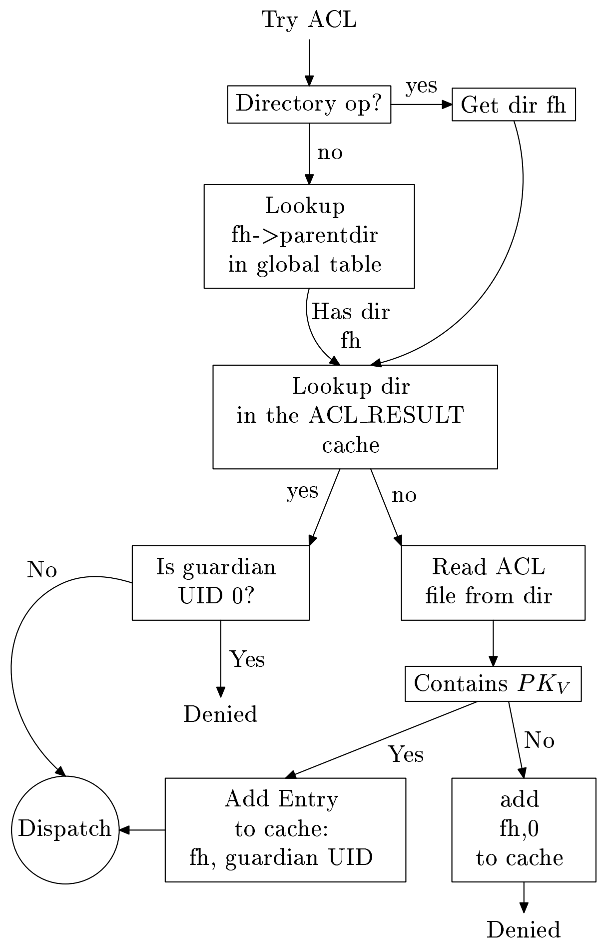

Figure 4: Flow of operation when checking ACL

6 Discussion

Beyond confirming the desired functionality of our system, it is difficult to quantitatively measure how well our system works. Having listed usability and transparency as our top level goals, we feel we have succeeded in developing a system that is easy for the average user to maintain. Sharing files merely involves invoking a command to add the visitor's public key to the authentication server and adding the key to the

ACL of the appropriate directory. This is significantly easier when compared with traditional Unix systems, and should remain as secure as the original SFS system.

The decision to use a global table mapping a filehandle to the filehandle of its parent directory was chosen over a more complex per-user design. The global table was easier to maintain than one that is stored for each user, but can potentially grow beyond a reasonable size. An ideal replacement policy would determine when filehandles became stale, removing those entries to make space for newer ones. The cost of such an operation, however, is somewhat expensive since it requires checking each entry in the table with the NFS server to determine its validity. On the other hand, tables stored per-client may contain duplicate entries, but could potentially reduce overhead and utilize knowledge of the client's activities to clear the table at opportune times. As a result, the per-client table is a more ideal solution than our global table.

The overall design of our system is not extremely complex in nature. However, in implementing it we realized that special considerations had to be made for various factors inherent in the underlying architecture. There were several special cases that had to be dealt with, concerning NFS operations and permissions, that we might have been able to avoid had the system been designed from scratch. In addition, existing protocols and architecture sometimes had to be circumvented, such as maintaining the table of parent filehandles.

6.1 Security

The overall security of any system that is composed of a conglomerate of different pieces, built on top of one-another, is always in question. We do not believe our changes have made the security of SFS any weaker as long as guardians are mindful of their ACL files.

The authentication system is almost identical to the original SFS so it should have the same secure properties of the unaltered SFS. We have only added public keys to the login credientials.

Visitors are permitted to perform almost all file system operations availible over NFS. They can read, write, create and remove files as long as they have the necessary permissions to working directory. The ACL file, however, is special. It cannot be read, altered or removed by a visitor. This disallows a visitor from changing the permissions to a directory. Visitors are permitted to create directories. When a new directory is created, the ACL from the parent directory is copied to the new directory. Visitors are not permitted to remove directories. This would require them to remove the ACL file, and they are not permitted to do this as an SFS visitor.

More compilcated security issues arise in setting the directory permissions. In our system, visitors can at most have as much access as their guardians do, per directory. If a guardian has been willing to add a visitor to an ACL, then they are willing to allow a visitor to read and/or write files as themselves. As long as they understand these semantics, they should understand the cababilies they given their visitors. An important implication of this is that the ACL is the primary barrier for the guardian's security. If a guardian is not careful setting the permissions of his ACL files, another user on the system could add visitors to direcoties unintended for visitor access. However, if the guardian is careful setting up his ACL he should be well protected.

While we attempted to utilize the SFS machinery as closely as possible, the matter of accessing files as another user may result in security related problems that we have not considered.

6.2 Performance

With regard to performance, accessing ACLs can potentially be a bottleneck when executing file operations. We attempt to minimize such behavior with the use of the cache that maps filehandles to guardian UIDs, though such a design can result in delays in updating permissions when ACLs are modified.

We compared the performance of our system against that of an unmodified SFS setup, both when the client represents a real user, and when

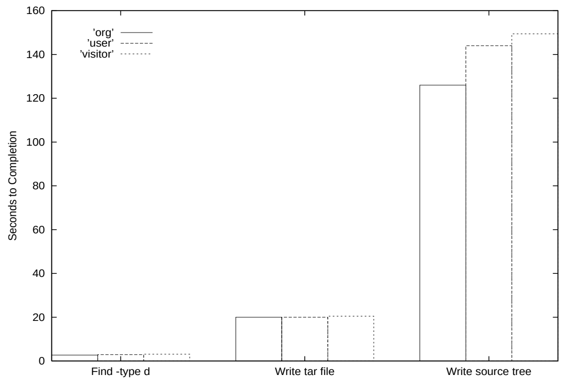

Figure 5: Time in seconds for running three different operations on three different client server setups. We ran these tests on a 600Mhz PIII server with 768MB RAM and a 550Mhz PIII client with 512MB RAM. Our changes do not adversely affect large file transfers, but performance is affected when using many different files.

it is a visitor. Our tests (See Figure 5) comprise writing a large tar file(the tarred SFS build tree), writing several small files(the SFS build tree), and executing a complex find command in an exported filesystem (find . -type d in the SFS build tree). As one can see, our ACL operations do not add much of an overhead in real-life situations; the largest performance loss was when writing many small files. We attribute this to the many LOOKUP operations required for all the files copied. LOOKUP requests are slower in SFS-ACL because of the extra work requried to read and parse the ACL file.

7 Conclusion

The modular architecture of SFS allowed us to implement a more flexible method of file sharing. By authenticating users through the SFS auth server, we allow any user with a public key to access specified directories. Visitors need not have an account on the server, and thus the hassle of managing special users and groups is avoided. File sharing is thus made significantly easier, while insuring that the security of the file system is not compromised.

References

[1] www-106.ibm.com/developerworks/linux/library/samba/index.html, November 2001.

[2] J. H. Howard. An overview of the andrew file system. In *Proceedings of the USENIX Winter 1988 Technical Conference*, pages 23–26, Berkeley, CA, 1988. USENIX Association.

[3] J. Kistler and M. Satyanarayanan. Disconnected operating in the coda file system, 1991.

[4] David Mazières, Michael Kaminsky, M. Frans Kaashoek, and Emmett Witchel. Separating key management from file system security. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP '99)*, Kiawah Island, South Carolina, December 1999.

[5] Sun Microsystems, Inc. NFS: Network file system protocol specification. *Internet Request for Comments*, (1094), 1989.

[6] M. SMB, F. Protocol, D. Microsoft, and C. Washington. Microsoft networks smb file sharing protocol.