Seminar in Distributed Computing WS 05/06

Decentralized User Authentication in a Global File System

Max Meisterhans - mmax@student.ethz.ch

Overview

- Centralized Control
- Self-certifying File System (SFS)
- Goals for the Implementation
- Trust Model/Security
- User Authentication
- Access Control Lists and Authorization
- Implementation
- Evaluation
- Future Work
- Conclusion

Centralized Control

Central Authorities

- **LAN**

  Central authorities to coordinate secure collaboration are reasonable

- **Internet**

  Doesn‘t provide the same administrative structures like in a LAN.

Centralized Control

Centralized authentication

- **Kerberos**
  - Based on shared-secret cryptography
  - Creating accounts impossible without involving a Kerberos administrator
- **AFS combined with Kerberos**
  - Cross-realm authentication based on Kerberos allows remote users
  - Every file server must be enumerated on client machines

Centralized Control

Certificate-based Systems

- **SSL**
  - Relies too heavily on Certification Authorities (CAs)
  - CAs all demand a similarly exacting certification process
- **Taos**
  - Provides a secure distributed computing environment with global naming and global file access
  - User authentication based on CAs, which are issuing certificates which map a public key to a name
  - CAs can be arranged into a tree structure

Centralized Control

Central Authorities

- Hinder deployment
- Stifle innovation
- Complicate cross-administrative realm collaboration
- Exclude valid network resources
- Create single points of failure
- Put everyone at the mercy of the authority

- Certificates allow general forms of delegation, but often require more infrastructure than is necessary to support a network file system

SFS – Self-certifying File System

- Each SFS file system has a name of the form “/sfs/Location:HostID“
  - Location: DNS name of the server
  - HostID: cryptographic hash of the server‘s public key
- HostID specifies a unique, verifiable public key with which clients can establish secure communication channels to servers
- A user runs an SFS client that mounts the SFS under “/sfs“
- Automatic Mounting: When a user references a non-existing directory, the SFS client tries to contact the machine named by “Location“. If everything is correct, the client creates the referenced directory in “/sfs/“ and mounts the remote file system there.
- Given a host on the network, anyone can generate a public key, determine the corresponding HostID, run the SFS server software, and immediately reference that server by its self-certifying hostname on any client in the world.
- Authentication Server provides a user authentication service

SFS – Self-certifying File System

- Key Exchange and Server Authentication

$$\text{host ID} = \{PK_s, \text{hostname}\}_{\text{SHA-1}}$$

$$\text{session ID} = \{K_{cs}, K_{sc}, PK_t, N_0\}_{\text{SHA-1}}$$

1. $C \rightarrow S$: hostname, host ID
2. $S \rightarrow C$: PK_s, PK_t, N_0
3. $C \rightarrow S$: $\{\{K_{cs}, K_{sc}\}_{PK_t}\}_{PK_s}$, session ID, N_0

SFS – Self-certifying File System

ACL - Example (1)

- Creating a personal group on the authentication server:

```
$ sfskey group -C charles.cwpeople
```

SFS – Self-certifying File System

ACL - Example (2)

- Adding members to a group:

```
$ sfskey group \
   -m +u=james \
   -m +u=liz@bu.edu,gnze6... \
   -m +g=students@mit.edu,w7abn9p... \
   -m +p=anb726muxau6phtk3zu3nq4n463mwn9a \
  charles.cwpeople
```

SFS – Self-certifying File System

ACL - Example (3)

- Making a user an owner:

```
$ sfskey group \
    -o +u=george@sun.com,heq38... \
  charles.cwpeople
```

SFS – Self-certifying File System

ACL - Example (4)

- Constructing an ACL and placing it on the directory:

```
$ cat myacl.txt
ACLBEGIN
user:charles:rwlida:
group:charles.cwpeople:rl:
ACLEND
$ sfsacl -s myacl.txt /courseware
```

SFS – Self-certifying File System

Summary

- Secure, global, decentralized file system permitting easy cross-administrative realm collaboration
- Uses self-certifying hostnames – a combination of the server‘s DNS name and a hash of its public key (calculated with SHA-1)
- Provides a global namespace over which no authority has control
- Authentication server provides a generic user authentication service to other SFS servers
- Can scale to groups with tens of thousands of members

Goals

- Allowing people to grant access to specific users and groups in remote administrative domains
- Provide Access Control Lists (ACLs) that can contain remote principles
- Authentication server can respond to an authentication request without contacting any remote authentication servers
- User authentication without requiring certificates

Security / Trust Model

- SFS is a collection of clients and servers that provide several services:
  - a global file system
  - remote execution
  - user authentication
- communication with Remote Procedure Calls
- each server has its own private key
- clients always explicitly name the servers public key using self-certifying hostnames

Security / Trust Model

SFS guarantees the following security properties for connections between SFS clients and servers:

- **Confidentiality**

  A passive attacker, who is able to observe network traffic, can only accomplish traffic analysis

- **Integrity**

  An active attacker can, at best, only effect a denial of service

- **Server Authenticity**

  The Server must prove its identity. Once a connection has been established, the client trusts the server who it claims to be

User Authentication

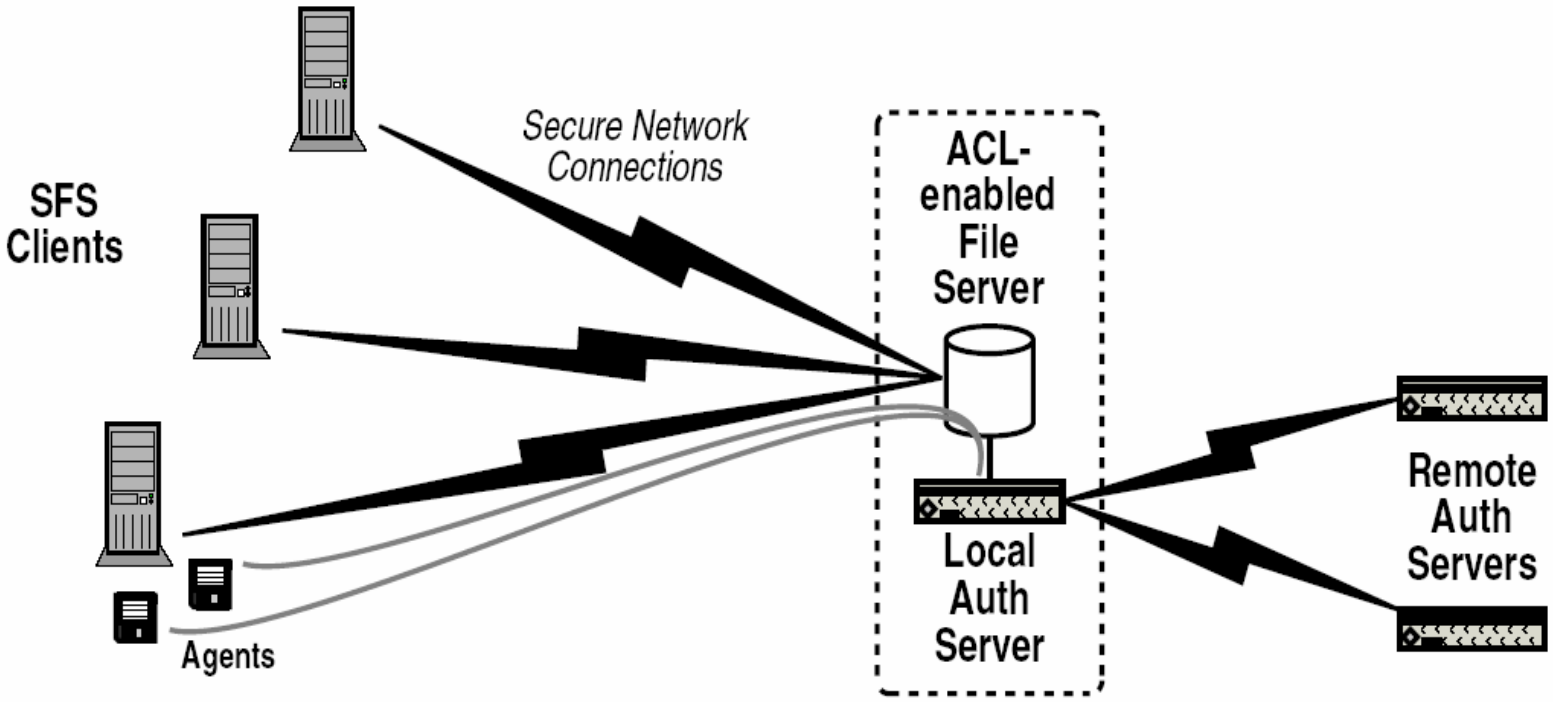


Figure 1: Overview of the SFS authentication architecture

User Authentication

User Authentication is a multi-step operation:

1. The SFS agent signs an authentication request on behalf of the user with his private key

2. The user sends his request to the file server, which passes it, as opaque data, on to the local authentication server

3. The authentication server verifies the signature and issues credentials to the user

4. The authentication server hands these credentials back to the file server

User Authentication

Authentication server

- Two main functions
  - It provides a generic user authentication service to other SFS servers
  - It provides an interface for users to manage the authentication name space
- Named using self-certifying hostnames
- Main challenge: How to retrieve remote user and remote group definitions

User Authentication

Authentication server - Interface

- The authentication server presents an RPC interface which supports three basic operations:
  - LOGIN: allows an SFS server to obtain credentials for a user
  - QUERY: allows a user or another authentication server to query the authentication database
  - UPDATE: allows a user to modify records in the authentication database

User Authentication

Authentication server – User/group records

User Record:

User Name	Public Key
ID	Privileges
GID	SRP Information
Version	Audit String

Group Record:

Group Name	Owners
ID	Members
Version	Audit String

User Authentication

Authentication server – Naming users and groups

```
p=bkfce6jdbmdbzfbct36qgvmpfwzs8exu
u=alice
u=bob@cs.cmu.edu,fr2eisz3fifttrtvawhnygzk5k5jidiv
g=alice.friends
g=faculty@stanford.edu,7yxnw38ths99hfpqnibfbdv3wqxqj8ap
```

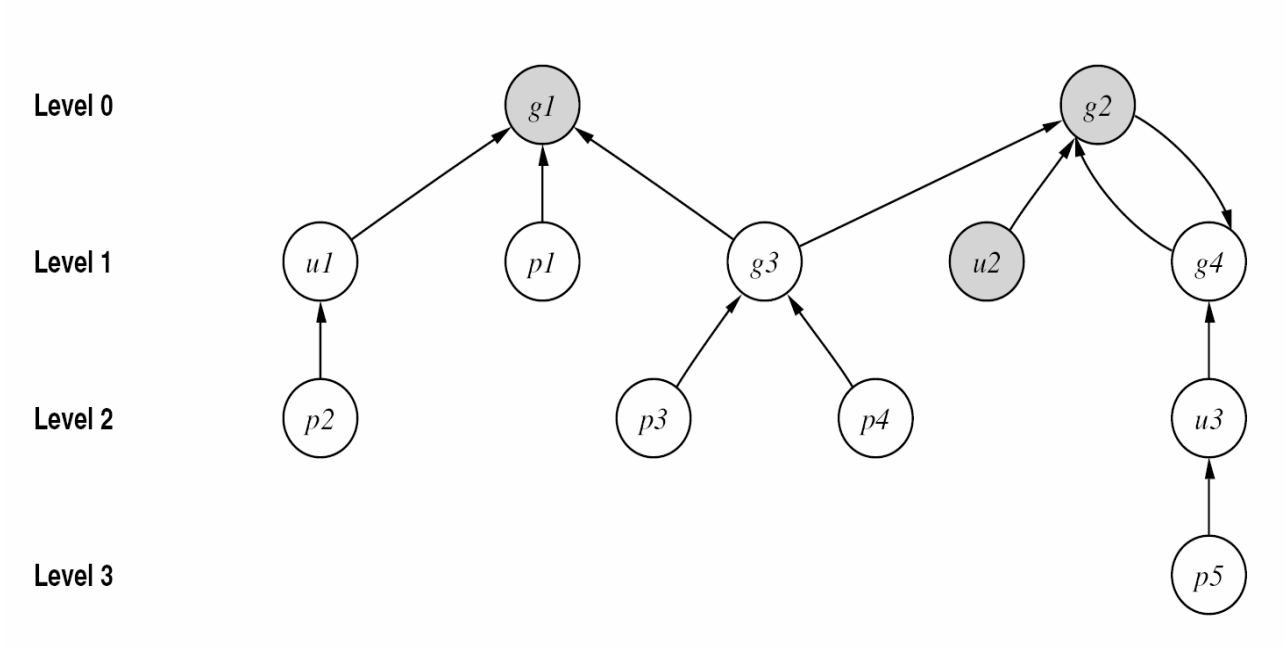
Figure 2: Example user and group names in SFS. Both `bob` and `faculty` are remote names, which include self-certifying hostnames.

Naming users and groups with self-certifying hostnames delegates trust to the remote authentication server.

This is important because it allows the remote group‘s owners to maintain the group‘s membership lists, but this implies that the local server must trust those owners.

User Authentication

Resolving group membership



membership graph for local groups g1 and g2

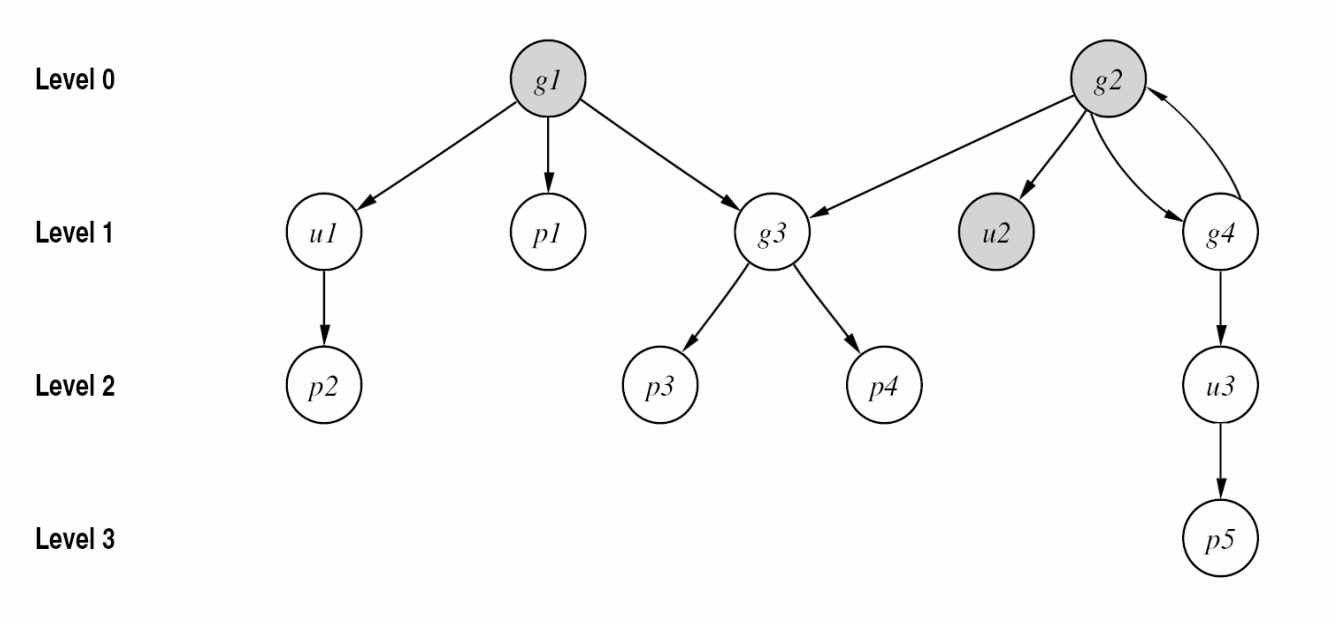
User Authentication

Resolving group membership

- Credentials that the authentication server issuues may contain a list of groups, but these groups must be local
- The server resolves each local group into a set of public keys by fetching all of the remote users that the local group contains
- Consequences:
  - Remote principals cannot appear directly on ACLs
  - Advantages:
    - The authentication server knows which remote users and groups to fetch
    - The authentication server fetches only users and groups that are necessary to issue credentials to a user
    - Because no public keys appear in ACLs, there is no need to update them, if a public key is changed

User Authentication

Resolving group membership



containment graph for local groups g1 and g2

User Authentication

Constructing the containment graph

- Resolving group membership reduces to the problem of constructing the containment graph given a set of local groups
- Problems
  - Groups can name remote users and groups
  - Traversing the containment graph must be efficient
  - The containment graph changes
- Addressing the problem by splitting the authentication task into:
  - Constructing the graph
  - Issuing credentials

User Authentication

Constructing the containment graph

- The authentication server constructs the graph and caches the records in the background
- The authentication server issues credentials when a user accesses the file system
- Updating the cache
  - Breadth-first traversal
  - Cache update cycle every hour
- Cache entries:

g1: *u1, p1, g3*
g2: *g3, u2, g4*
u1: *p2*
g3: *p3, p4*
g4: *u3, g2*
u3: *p5*

User Authentication

Constructing the containment graph

- Optimizations
  - Store connections to the remote authentication servers during an update cycle
  - Authentication servers only transfer the changes made since the last update
  - Remote authentication servers can transform user names into their corresponding public key hashes
- Performance of updating the cache
  - Bytes to fetch
  - Time to traverse the containment graph

User Authentication

Constructing the containment graph

- Freshness
  - The cache update scheme has eventual consistency
  - Trade-off between efficiency and freshness
- Revocation
  - Problems for users who have their public key hashes on another user‘s group record or ACL

User Authentication

Credentials

- Given the user‘s public key, the authentication server uses its database to determine the credentials.

- The authentication server supports three credential types:
  - Unix credentials
  - Public key credentials
  - Group list credentials

ACLs and Authorization

- Once the user has the credentials, the SFS server can make access control decisions based on those credentials.
- The file system needs the ability to map symbolic group names to access rights.
- An ACL is a list of entries that specify what access rights the file system should grant to a particular user or group of users.
- Four different types of ACL entries:
  - User names
  - Group names
  - Public key hashes
  - Anonymous

ACLs and Authorization

- Access rights

Permission	Effect on files	Effect on directories
r	read the file	no effect
w	write the file	no effect
l	no effect	change to the directory and list its files
i	no effect	insert new files/dirs into the directory
d	no effect	delete files/dirs from the directory
a	modify the file's ACL	modify the directory's ACL

Figure 4: Access rights available in SFS ACLs

- The ACL server does not support negative permissions; once an ACL entry grants a right to the user, another entry cannot revoke that right.

```
ACLBEGIN
sys:anonymous:rl:
group:author1.sosp2003:wlida:
ACLEND
```

Implementation

Authentication server

- To improve scalability, the server has a Berkeley DB backend, which allows it to efficiently store and query groups with thousands of users.
- Berkeley DB is also used to store the authentication server‘s cache

Implementation

ACL-enabled file system

- Files are stored on the server‘s disk using NFSv3. This offers portability to any OS that supports this file system.
- File ACLs are stored in the first 512 bytes of the file and directory ACLs in a special file in the directory called `.SFSACL`
- Use of a text-based format for the ACLs.
- Permissions
  - When the server receives a request, it retrieves the necessary ACLs and decides whether to permit the request
- Caching
  - The server caches ACLs to avoid issuing extra NFS requests
  - The server caches the permissions granted to a user for a particular ACL based on his credentials

Implementation

Usage

- Tools for manipulating of groups and ACLs

Evaluation

Authentication server

- The number of bytes that the authentication server must transfer to update its cache depends on the number of remote records that it needs to fetch
- Group records are fetched using a QUERY RPC
- Connecting to the remote authentication server requires two RPCs
- Because the implementation caches secure channels, only one channel is established during an update cycle

Evaluation

Authentication server

Experimental Results:

To transfer	*Q*	*R*	*S*	*M*	*O*	*B*
0 users	72	136	40	0	216	208
10000 users	72	136	40	10000	216	408632
0 changes	72	108	40	0	180	180
1000 changes	72	108	40	1000	180	40720

B: Total number of bytes transferred
Q: Size of the RPC request
R: Size of the Reply
M: Number of users in the group (or number of changes to the group)
S: Size of a single user (or change)
O: RPC overhead incurred for each additional 250 users

Evaluation

Authentication server

Total number of bytes transferred for a particular group size or number of changes is given by the following formula

$$B = Q + R + (M \times S) + \left\lfloor \frac{M}{251} \right\rfloor \times O$$

B: Total number of bytes transferred
Q: Size of the RPC request
R: Size of the Reply
M: Number of users in the group (or number of changes to the group)
S: Size of a single user (or change)
O: RPC overhead incurred for each additional 250 users

Evaluation

Authentication server

- The RPC overhead is insignificant, only a few percent of the total bytes transferred
- The Authentication server can reasonably support group sizes up to tens of thousands

Evaluation

ACL-enabled file system

- Penalty due to ACL mechanisms
- Results of running a benchmark:

Phase	Original SFS seconds	ACL SFS with caching seconds (slowdown)	ACL SFS without caching seconds (slowdown)
create	15.9	18.1 (1.14X)	19.3 (1.21X)
read	3.4	3.5 (1.03X)	4.3 (1.26X)
delete	4.8	5.1 (1.06X)	6.0 (1.25X)
Total	24.1	26.7 (1.11X)	29.6 (1.23X)

Figure 6: LFS small file benchmark, with 1,000 files created, read, and deleted. The slowdowns are relative to the performance of the original SFS.

NFS request	Original SFS (NFS RPCs)	ACL SFS with caching (NFS RPCs)	ACL SFS without caching (NFS RPCs)
lookup	1	1	3
access	1	1	3
read	1	1	2
Total	3	3	8
Predicted slowdown	1.00X	1.00X	2.67X

Figure 7: Cost of reading a file during the read phase of the Sprite LFS small file benchmark, expressed as the number of NFS RPCs to the loopback NFS server.

Future Work

- Add the ability to update the cache more or less frequently
- Add the option to restrict users from naming certain remote principals

Conclusion

- Generality is sacrificed for ease-of-use and simplicity of implementation
- The authentication server does not require an infrastructure for managing certificates
- Issuing credentials does not require contacting remote sites during file access
- Experiments demonstrate that the server can scale to groups with tens of thousands of users
- Assumption of formation of a trusting group.