Decentralized User Authentication In a Global File System

Paper By:
Michael Kaminsky
George Savvides
David Mazières
M. Frans Kaashoek

Presentation by:
Rahul Potharaju
EECS 345

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 that allows remote users
- Every file server must be enumerated on client systems

Certificate Based Systems

SSL

- Relies heavily on Certification Authorities
- Certification process is very similar

Taos

- Global naming and file access
- CAs map a public key to a name

Disadvantages of Centralized Control

- Hinder deployment
- Complicate cross-administrative realm collaboration
- Create single points of failure
- Put every one 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

Kerberos

- Was developed at M.I.T. and is based on the Needham-Schroeder authentication protocol.
- It is a security system that allows clients in setting up a secure channel with any server that is part of a distributed system.
- Security Is based on shared secret keys.
- Two components → **Authentication Server** and **Ticket Granting Service**

Self-Certifying File System - SFS

- After framing the SFS, it was extended to validate the functions of the authentication server. This was achieved by making the SFS compatible with ACLs

- The ACLs for the files are stored in the first 512 bytes. Though there is a performance overhead, it helps in demonstrating the usefulness of the authentication server.

Creating a personal group on the authentication server:

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

New Group Name

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
```

Local User

Remote user maintained at bu.edu

Students at mit.edu

Hash of a user who does not belong to the organization

Group Name

Self-Certifying File System - SFS

Creating an Owner:

Name of the new owner

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

Local User

Constructing an ACL and placing it on the directory:

$cat myacl.txt
ACLBEGIN   Begin Statement
user: charles:rwlida: From user onto group
group: charles.cwpeople:rl:
ACLEND   End Statement
$sfsacl –s myacl.txt /courseware   Directory Name

Self-Certifying File System – SFS Notable Points

- 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) to other SFS servers

- Provides a global namespace over which no authority has control

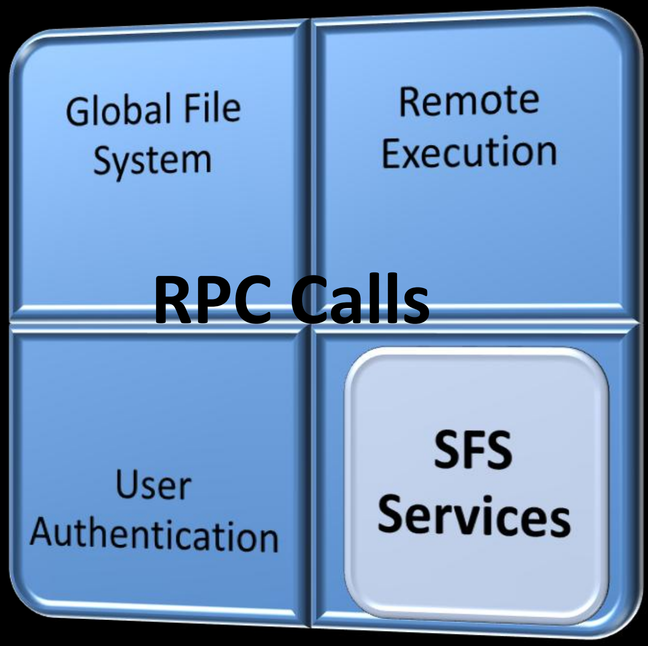
- Authentication server provides a generic user authentication service

- Can scale to groups with tens of thousands of members

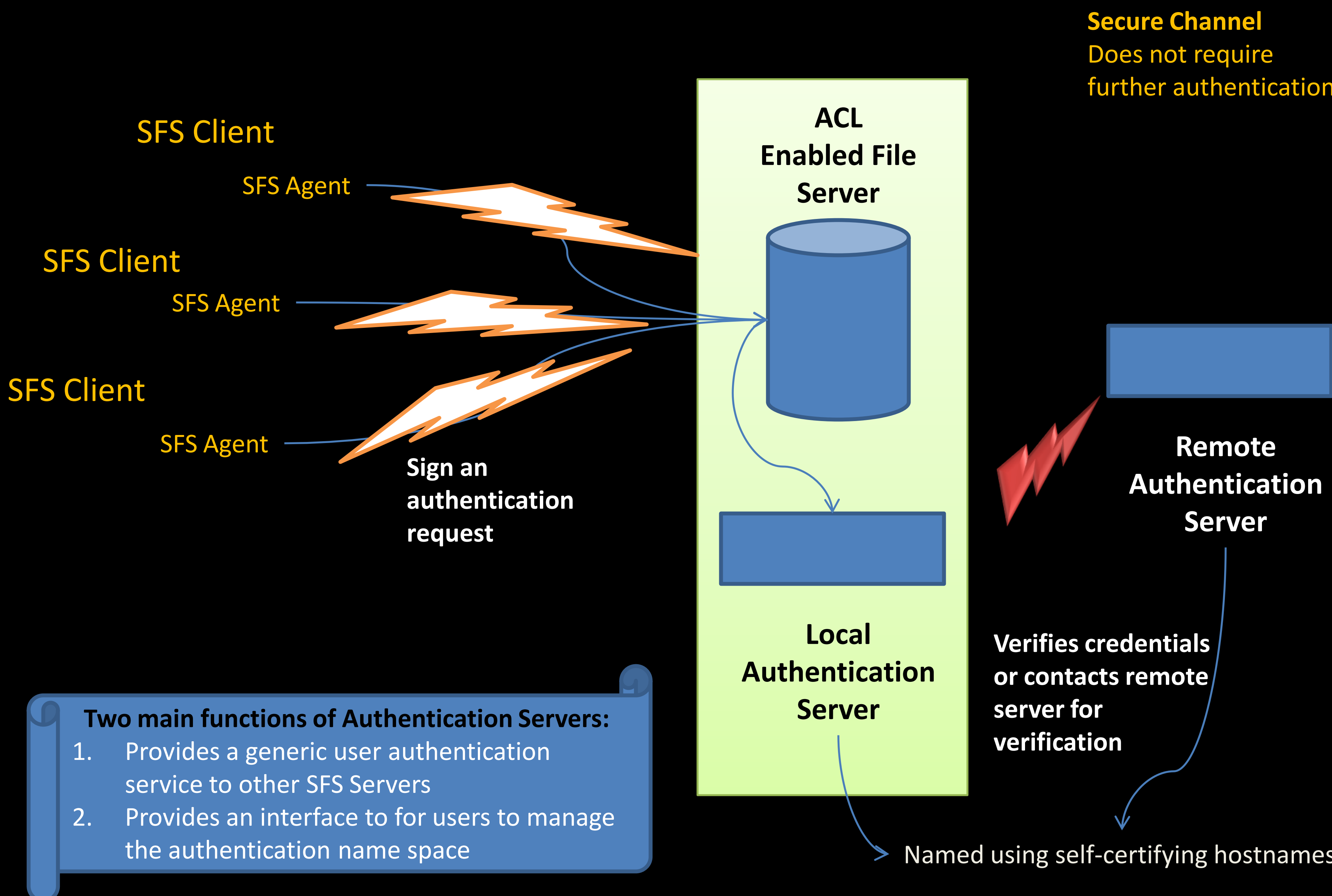
Security and Trust Model

Guarantees provided by SFS:

- Confidentiality
  - An attacker can sniff as much traffic as he wants – he'll end up doing traffic analysis!
- Integrity
  - An attacker an insert/delete etc. but at the max can cause a DoS attack
    - Is this good?
- Server Authenticity
  - When the client initiates the connection, the server must first prove that it knows the private key that pairs with the public key in the self-certifying hostname.



Overview of the SFS Architecture



User Authentication

Authentication Server Interface

RPC Interface

LOGIN
QUERY
UPDATE

User Records

Analogous to UNIX: User Name, ID, GID

User Name
ID
GID
Version

Public Key
Privileges
SRP Information
Audit String

Public Key → ASCII armored SHA-1 hashes

Version → Indicates number of times updated

Audit String → Who last updated this record?

SRP Information → For users who want to use the Secure Remote Password protocol

Group Records

Group Name
ID
Version

Owners
Members
Audit String

Naming Users and Groups

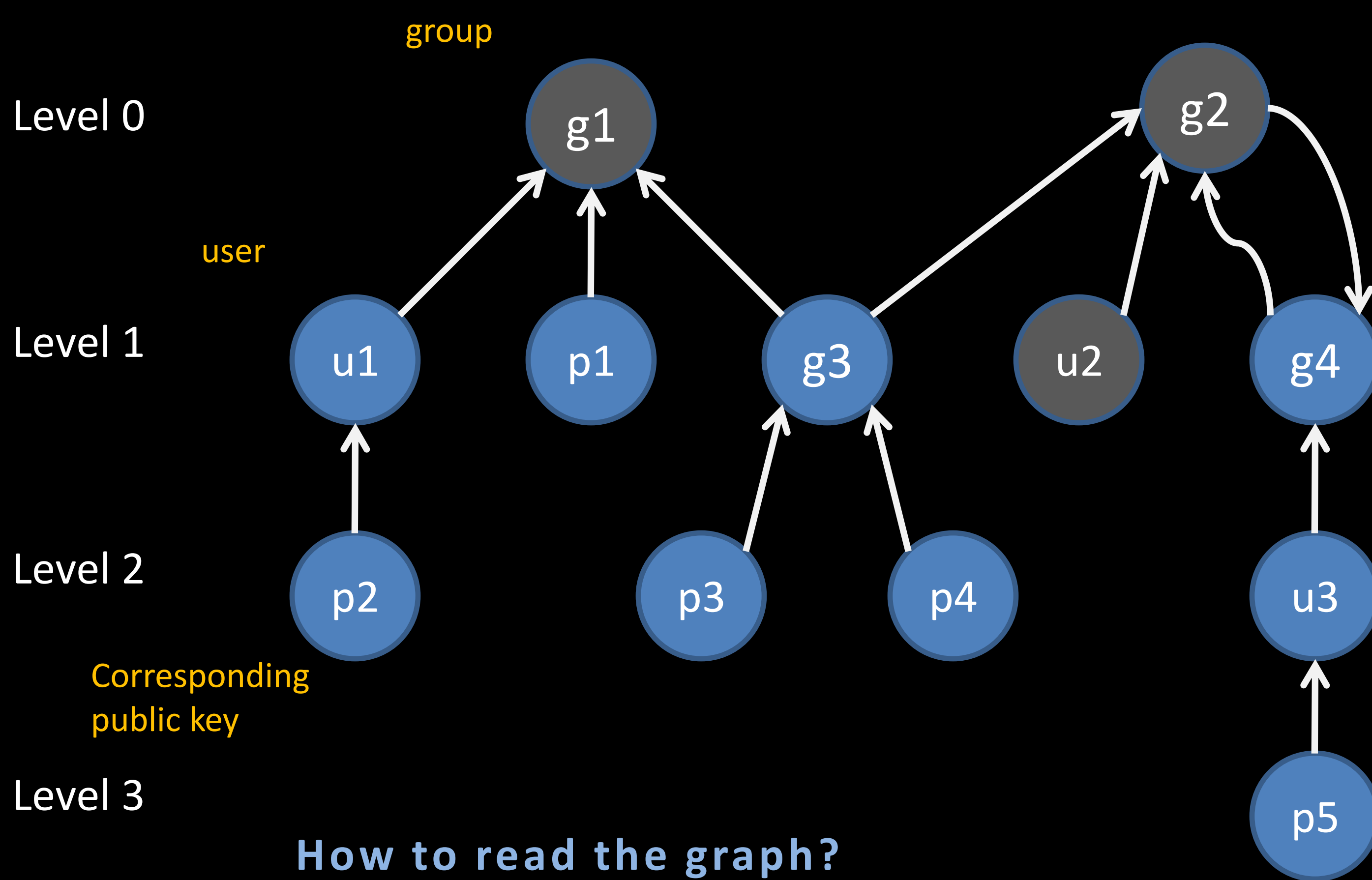
p = bkfce6jdbmdbzfbct36qgvmpfwzs8exu
u = alice
u = bob@cs.mcu.edu, fr2eisz3fifttrtvawhnygzk5k5jidiv
g = alice.friends
g = faculty@stanford.edu, 7yxnw38ths99hfpqnibfbdv3wqxqj8ap

- Self-certifying hostnames delegates trust to the remote authentication server.
- 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

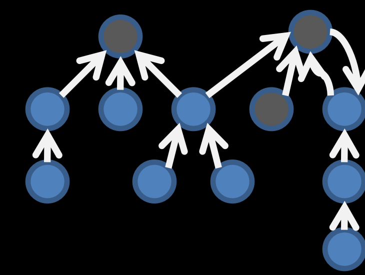


How to read the graph?

p2 “belongs” to user u1
u1 “belongs” to group g1
G3 “belongs” to group g1 and g2

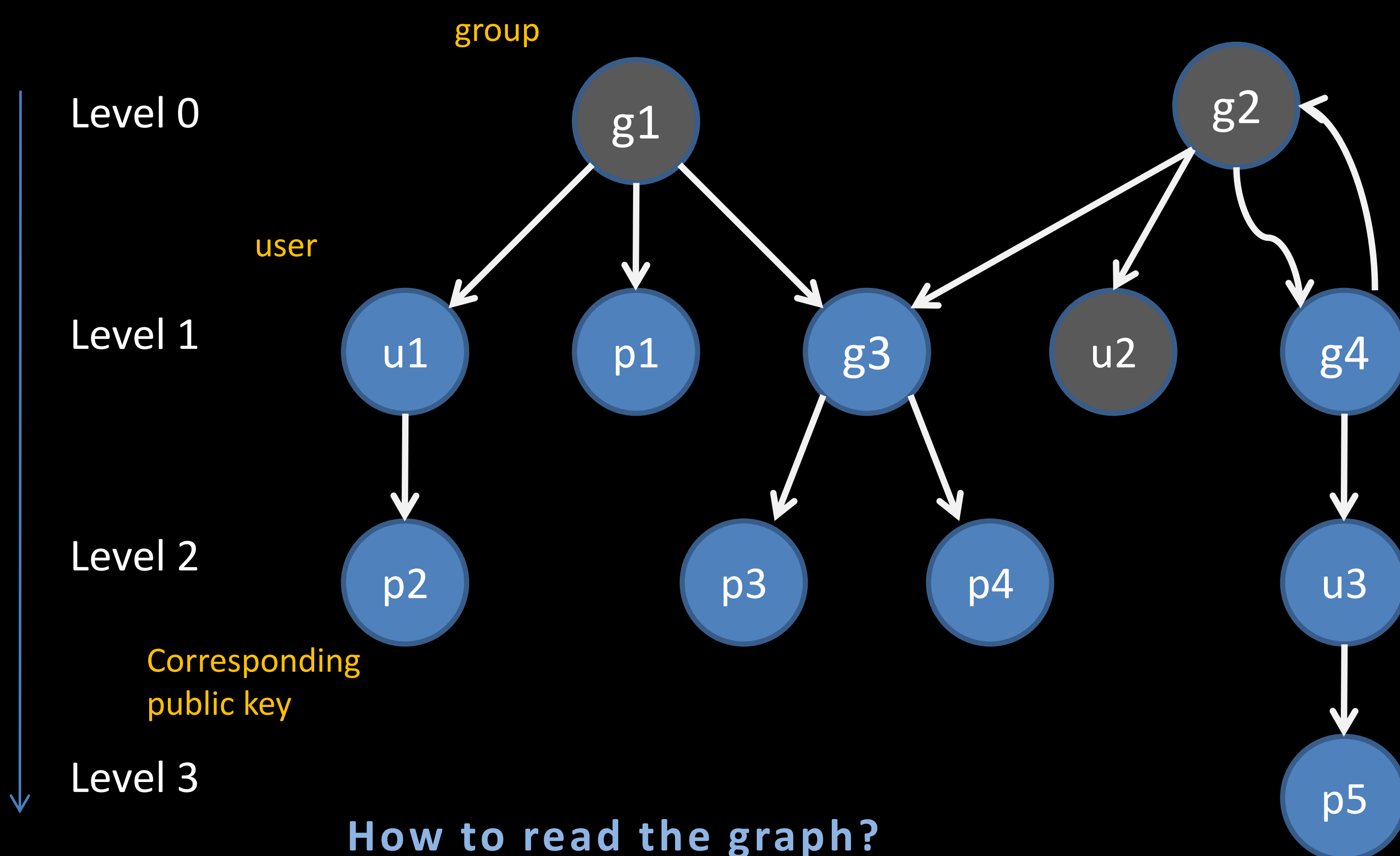
All is good… But its like ________________…
To address this problem, the authentication server constructs a **complementary graph** to construct the membership graph

User Authentication



Resolving Group Membership

Containment Graph



How to read the graph?

p2 "belongs" to user u1
u1 "belongs" to group g1
G3 "belongs" to group g1 and g2

g1 "contains" u1
u1 "has" p2 as his public key
g1 "contains" another group g3

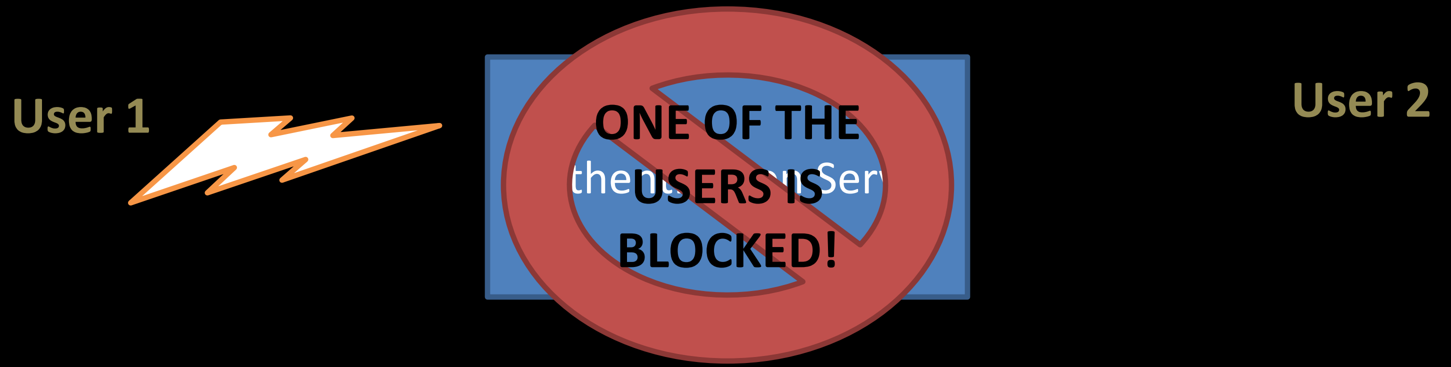
User Authentication

Resolving Group Membership

Challenges in Constructing the Containment Graph:

- Groups can name remote users and groups
    - Because the graph could contain remote users and groups a large number of remote authentication servers have to be contacted

- Traversing the containment graph must be efficient



- Containment graph changes
    - The world is dynamic after all!

User Authentication

Resolving Group Membership

So how are the challenges resolved?

- Split the authentication task into two parts:
  - Constructing the graphs
    - Uses Pre-fetching and caching
  - Issue Credentials
    - Does this only when a user tries to access the file system
- Cache is stored to disk so that the server can resume state after restarts

User Authentication

Resolving Group Membership

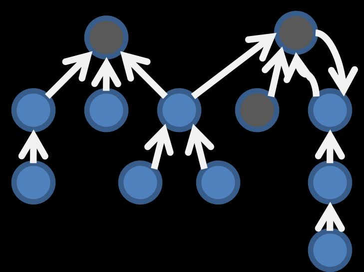
Updating the Cache

- Performs a breadth-first search and fetches the records in that order
- Never visits the same node twice (to detect graph cycles)
- Stores the reverse mappings (thereby yielding the membership graph)

Securely contacts remote server but not a problem because of the self-certifying hostnames i.e. Local → Remote Authentication is a breeze!

Cache Entries

- An adjacency list



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

Optimizations

- Store connections to the remote authentication servers during an update cycle update
- Authentication servers only transfer the changes made since the last update – Incremental Updates
- Remote authentication servers can transform usernames into their corresponding public key hashes

User Authentication

Resolving Group Membership

Performance Analysis

- Number of bytes to fetch
- Time required to traverse the containment graph
- Number of public key operations required to update the cache

Depends on whether there was already a copy cached → If there was, then fetch only the updated records else full fetch

Sum of the download latencies at each level – BFS!

Dependent on the number of unique servers in the containment graph

Freshness

- Freshness vs. Efficiency → Winner is Efficiency because delays are not acceptable
- Freshness vs. T(Cache Update) → Winner is Freshness because the other is less

Revocation

- All the servers that have a particular record have to be contacted – Most difficult!

User Authentication

Credentials

- Authentication → Process through which the AS issues credentials on behalf of the user

- As we've already seen:
  - User signs a request with his private key and sends it to the SFS server
  - SFS routes this request over to the local AS as part of LOGIN
  - Local AS if required will contact Remote AS else it will try to match the user's public key with the signature.
  - The AS determines the credentials of the user

- Three types of Credentials:
  - Unix credentials → Fields from the /etc/passwd. Gives only to users in the authentication database
  - Public Key Credentials → Text string containing an ASCII armored SHA-1 hash of the user's public key
  - Group List Credentials → List of groups to which the user belongs to

ACLs & Authorization

Credentials

- 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 → To name users with Unix accounts on the local machine
    - Group names → Refers to the SFS groups on the Local AS
    - Public key hashes → ASCII armored SHA-1 hashes used to match against Public Key credentials
    - Anonymous

ACLs & 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 directory and list 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

No negative permissions unlike AFS!
Once an ACL entry grants access to a user, another entry cannot revoke it

Implementation

Authentication Server

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

Authentication Server

- 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 ACL based on his credentials
- Caching
  - The server caches ACLs to avoid issuing extra NFS requests
  - The server caches the permissions granted to a user for a particular

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
  - They limit the number of owners and groups returned ➔ Some queries may require two or more Query RPC calls
- Connecting to the remote authentication server requires two RPCs
  - Because the implementation caches secure channels, only one channel is established during an update cycle ➔ Save one RPC per query
- They ran two experiments…

Evaluation

Authentication Server

Two Experiments

Local AS fetched the entire group because it didn't have anything in its cache

=

- Number of bytes transferred scales linearly with group size
- Total Groups transferred = 1001

--- Each group consisting of increasing number of users

- Users were represented using hashes of their public keys (34 bytes)
- Group names → 16 bytes
- Audit strings → 70 bytes
- Owners list → empty

Local AS had a cached copy

=

- Number of bytes transferred scales linearly with number of changes in the group since last update
- Varied the number of changes from 0 to 9990 in steps of 10
- Each change was simple – Add a user and a (+) sign

Evaluation

Authentication Server

Size of the RPC Request

Size of the reply

RPC Overhead per 250 users

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	10000	180	40720

Total bytes transferred

Size of a single user

Number of users in the group

$$B = Q + R + (M*S) + [M/251]*O = 400\ \text{KB}$$

Insignificant from the evaluation
For instance, to transfer 10000 users, the overhead was 8424 bytes
➔ Just 2% of the total bytes

Result is favorable – Looks like it can support MIT Athena group which is large

Evaluation

ACL-enabled File System

Benchmark create, reads, and deletes 1,000 1024 byte files and then flushes the cache

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.

Performance Penalty associated with the ACL mechanism!

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.

Comments

- Paper makes the reader feel as if they have nothing to hide. Reveals almost everything in the system
- If there's a drawback (such as the 512 byte overhead), they address it right away.
- Self certifying hostnames looks like a promising decision
- Too many sections! ☹
- No graphs ☺