CS632: Group 05
Distributed Access Control System Project Report

Arbaz Khan
arbazk@iitk.ac.in
CSE Department

Mayank Dang
mayankd@iitk.ac.in
CSE Department

Indian Institute of Technology, Kanpur

Final report
10^{th} Nov, 2012

Abstract

In this report we present a decentralized access control system based on authorization certificates. The system is designed to be scalable, provide high availability and overcome the problem of SPOF (Single Point of Failure) that is characteristic of a centralized access control system. We also describe a method to decentralize the Certificate Store. The system is designed to provide the user with node transparency - the user can connect to any AC (access control) node to gain access to the distributed system.

1 Introduction

Internet enables global sharing of resources across organizational boundaries. In the domain of resource sharing, distributed systems offer many advantages. These systems are fault tolerant, provide faster access and high availability, have low setup cost and are tremendously scalable. However, an organization has many types of resources and often organizations can only offer selective role-based access to some or all of its resources. Hence, there is a need of a light-weight access control system that can be used to validate users and provide access based on the assigned privileges.

A Distributed Access Control System (DACS) is such a system that provides access control for resources spread out at multiple locations. Most traditional implementations of DACS are based on a centralized control server that acts as intermediary between the end user and the distributed system. All requests for resources by a client are processed by DACS and the server provides access based on the permissions stored at the server. However, due to a centralized model, the existing centralized access control mechanisms fail to provide scalability and high availability, the very essential features that form the base of Cloud Computing.

Hence, a decentralized scalable model for DACS is required. The model should overcome the problem of SPOF ((Single Point of Failure) and hence some redundancy has to be added. We use digital signatures to sign the certificates for correctly isdentifying a valid certificate. Digital Signature Algorithm based on RSA is used for signing the certificate. Certificates are issued to clients in encrypted format. Since , RSA cant handle encrypt large plaintext messages, AES 16-bit encryption algorithm is used to encrypt the certificate data and sent along with the corresponding AES key encrypted with RSA public key of the destination.

1.1 Problem Statement

Design a scalable, fault tolerant, decentralized access control system that provides fast access and high availability. The design should overcome the problem of SPOF (Single Point of Failure).

1.2 Related Work

A decentralized access control system based on authorization certificates was presented by [1]. They used CRL's (Certificate Revocation Lists) to store the list of certificates on the access servers for which the privileges have been revoked. The access metadata like privileges, expiry date, etc. is carried in a certificate having a digital signature of the document. The signature is used to verify the

identity of the user. [2] used access control lists for storing and validating access rights. [4] described file sharing in distributed file systems. Their DACS was an implementation over the Network File System. [3] also used signatures to create an access control system and used a policy definition language to express access rights.

2 Algorithm

Decentralized Access Control servers do not require the maintenance of most access control information on the side of the mechanism. Information required by the user is presented to the Access Control at the time of access control. That means, that the user has a 'certificate' containing all the requisite profile information:

- client id
- privileges for different resources
- expiry date of the certificate
- user public key

The client sends the certificate at the time of request. The server then checks if the user is authenticated, ie, it checks for the validity of the certificate. To check for the validity of a (signed) certificate, the AC node verifies the signature of the certificate with the public key of the issuing node (the node that issued the certificate). For this purpose, each AC node is required to know the public key of every other AC node. This exchange of public keys is done when the DACS is started.

```
<certi>
 <dacs>
  <id>
  </id>
  <privileges>
  </privileges>
  <expiry>
  </expiry>
  <pubkey>
  </pubkey>
 </dacs>
 <sign>
 </sign>
</certi>
```

Figure 1: Certificate format

If the certificate is valid, the AC server replies with a success message, indicating the client to proceed with actual requests. The permissions that a user is allowed are modelled as a string. For each subsequent requests, the server checks if the permission requested is a subset of the allowed permissions. The allowed permissions would in practice be maintained as Access Control Lists (a list of permissions for each resource) once the user is authenticated.

Since the AC nodes do not require any explicit maintenance or moderator at the back end, we have a moderator user who can issue certificates to others. The moderator has an initial certificate having maximum privileges. Any user first has to request the moderator to assign a certificate for that user. The moderator (*parent*) compiles a certificate for the user (*child*) and sends it for issuance to the server. The server, after checking the privileges and the expiry date of the parent and the user, then creates a certificate for the *child*. It stores the certificate in its **Certificate Store** and sends a passphrase back to the *parent*. The parent gets the passphrase and tells it to the *child*. The child can request *any* AC node for its certificate. The AC node maps the passphrase of the *child* to corresponding AC node (which issued the certificate of the *child*) and retrieves back the certificate from that AC node. It then sends the certificate back to the child, after encrypting the certificate with the public key of the child. This ensures that the certificate can only be retrieved by the user for which it was intended.

2.1 Certificate Issuance

- A user (can be any user, not necessarily the moderator) sends its certificate (*parent* certificate) along with the information comprising the newly issued certificate (permitted operations, expiry date, public key of the intended user) to the AC node.
- The *parent* certificate is verified by the AC node.
- The AC node ensures that the requested operations and the expiry date for the *child* are a subset of the *parent's* operations and expiry date respectively.
- It creates a new certificate for the child and signs it with its own private key. The certificate is kept in the Certificate Store of the AC node and a passphrase is returned to the *parent*.

2.2 Certificate Retrieval

- A user requests for its certificate by sending its passphrase for **any** AC node.
- The AC node uses the passphrase to find out the AC node that assigned the child's certificate.
- The AC node then retrieves the *child*'s certificate from the assigner-AC node.
- The AC node sends the certificate back to the user, after encrypting it with the user's public key.

- At the client's end, the encrypted certificate is decrypted using the user's private key.
- The retrieved certificate is used to access the distributed system by the user in the future.

2.3 Certicficate Verification

- The AC node confirms that the current date is not past the expiry date of the certificate.
- The AC node maps the user-id to the issuing AC node.
- It then verifies the signature of the certificate with the public key of the issuing node.
- It replies the node with success if everything is satisfied above.

2.4 Validating user permissions

- Once a node has been authenticated, the privileges are stored in the form of an Access Control List at the server.
- Each subsequent resource access request gets passed on to the underlying distributed system if the user has the required access privileges for the requested resource. Access Control List is used to check for the same.
- If the user does not have the required permissions, then the AC node replies with a failure.

2.5 Fault Tolerance and Reliability

An AC node stores the certificates in a Certificate Store. It is hence necessary for the server to retain that store even in the event of a failure at AC node.

This is done by maintaining R copies of a certificate on R different AC nodes (where R is the Replication factor). When an AC node creates a certificate for a child, it forwards the certificate to $R-1$ other AC nodes (AC nodes are arranged in an implicit serial order). In the event of AC node failure, the certificates can be retrieved by other AC nodes from any of the other $R-1$ running nodes (on which the certificates were replicated). The number of failures to deny Certificate Retrieval is alteast R. Increasing the replication factor R increases the reliability.

$<<< Insertfigure >>>$

2.6 Scalability

DACS based on a centralized server is not scalable. The reason is that in a centralized server, all the access requests pass through one server, thereby resulting in greater latency for large number of users. Access Control Lists (ACL) for all the users are stored on the centralized server and as a result, the server also faces large space crunch for storing the ACL's for large amounts of users. The centralized server is also a single point of failure. Although we can replicate the Certificate Store for eventual consistency, the service still would be off-line for some time in the event of a failure, since copying of the backup store would take some time. The current implementation does not have any bottlenecks. The individual AC nodes are not single points of failure, since the certificates are replicated on other AC servers too and can be retrieved from there in the event of a failure. As discussed above, the system is able to tolerate $R-1$ failures for a replication factor of R. The replication factor can be increased to increase reliability.

The number of AC nodes can be increased to any number. Greater the number of AC nodes, greater would be the increase in performance and availability for the end-users. The only constraint in arbitrary increase in AC nodes is the cost.

2.7 Fault Recovery

Any server failures can be categorized into hardware and software faults. Hardware faults are critical damages to every system. Hence, we focus on recovering from software faults here.

Certificate Store is a directory in which issued signed certificates are stored. It is safe from any software crashes. The only information lost on any software crash is the list of id's and public keys of all the other AC nodes in the mesh. We provide fault recovery mechanism, wherein this information is re-retrieved from all the other servers when the server recovers from the crash. The recovering server pings all the other AC servers. The other AC servers respond by sending their id's and public keys, which are then saved again. The other servers, in turn, update their information about the newly setup node.

Hence, our system is able to recover from any software fault without any loss of information, functionality or need to restart the whole system again.

2.8 Overcoming Denial of Service

The system described does not require the existence of special parent nodes for issuing certificates. Any node can request for a certificate for other nodes. This has several de-merits. For example, the system would be prone to Denial-of-service attacks, involving repeated certificate requests for child nodes. To get around this problem of Denial of Service attacks against AC nodes, we have incorporated a counter. Each user can now have

a fixed number of unclaimed certificates pending in the Certificate-Store. As soon as any parent requests for (1+limit)th certificate, the AC node would stop issuing certificates to that user. Hence, Denial of Service is also effectively handled.

3 Conclusions

In this report we have presented a decentralized access control system based on authorization certificates. The system is designed to be scalable, provide high availability and overcome the problem of SPOF (Single Point of Failure) that is characteristic of a centralized access control system. We have also described a method to decentralize the Certificate Store.

The only additional requirements on which the success of this model works are:

- A secure communication channel between different AC nodes. The certificates sent across the network should not be sniffed out by any external agent.
- Here, we have modelled the 'privileges' field by a string just for simulation. Actual implementations would use Access Matrix - information about each resource stored as a list.

References

[1] J. Arakawa and K. Sasada. A decentralized access control mechanism using authorization certificate for distributed file systems. In *Internet Technology and Secured Transactions (ICITST), 2011 International Conference for*, pages 148 –153, dec. 2011.

[2] Michael Kaminsky, George Savvides, David Mazieres, and M. Frans Kaashoek. Decentralized user authentication in a global file system. *SIGOPS Oper. Syst. Rev.*, 37(5):60–73, October 2003.

[3] Alexander Levine, Vassilis Prevelakis, John Ioannidis, Sotiris Ioannidis, and Angelos D. Keromytis. Webdava: An administrator-free approach to web file-sharing. In *In Proceedings of the IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), Workshop on Distributed and Mobile Collaboration*, 2003.

[4] Stefan Miltchev, Jonathan M. Smith, Vassilis Prevelakis, Angelos Keromytis, and Sotiris Ioannidis. Decentralized access control in distributed file systems. *ACM Comput. Surv.*, 40(3):10:1–10:30, August 2008.