

Certificate Based Authorization Simulation System

Jie Dai and Jim Alves-Foss
Center for Secure and Dependable Software
University of Idaho
{jiedai, jimaf}@cs.uidaho.edu

Abstract

Using certificates for distributed authorizations in computer network systems has been discussed in the literature. However real implementations of the concept are rarely seen. In our certificate based authorization simulation system (CBASS) project, we prototyped a computer system including some of the emulated functions of an operating system such as machine, user and file management, and emulated applications. These system management functions and applications use certification instead of conventional access control list mechanism for resource access control. It is our purpose to explore a practical way to build the distributed authorization into computer systems in order to solve authorization problems which exist in present open and large distributed computing environments. In this paper, we present our design and implementation of the CBASS.

1 Introduction

Authorization is an important issue in computer and network systems. The mechanism exists in operating systems, applications, or anywhere resource access control is concerned. Authorization is traditionally composed of two separate processes: authentication and access control. Authentication deals with the problem “who is the user?” and access control deals with the problem “what can the user do to a certain resource.” In traditional distributed computing networks, a domain based *access control list* (ACL) is the dominant mechanism for access control. This list specifies access rights of users to the resources in terms of read, write or execute permissions in data structures and inevitably brings problems to the existing authorization mechanisms. With the ACL based authorization mechanism, a system has to conduct authentication through challenge-response or by resolving a public-key certificate in order to verify the identity of an entity in the form of a name. However, as computer networks grow larger, names become an inappropriate media for identification. For instance, name collision is one of the problems. Even though global names can be used such as in the *public key infrastructure* (PKI),

they raise another issue of privacy because there is no assurance of anonymity [2]. Secondly, secure systems and services usually require an expressive authorization mechanism to implement the *least privilege principle* and multiple security policy models. ACL based authorization mechanism encounters difficulty to accomplish this, as the centrally controlled mechanism is coarsely grained. Additionally, policies or policy models in the ACL based systems are usually hard coded into the system software. This reduces the scalability of the system in respect of satisfying the ever-changing security requirements [3]. Finally, it is hard to localize authorization with static ACLs.

In order to solve above problems, researchers have developed techniques, namely *authorization certification* (AC) [4], trust management system [1], and logical programming and knowledge representation [6] for describing certificate delegation among applications. Our prototype extends these techniques by writing general authorization policies in certificates in a form of rules or facts and deploying the certificates to the resource owners and requesters to implement distributed *discretionary access control* (DAC) for a system. In our prototype, policies in ACLs are replaced by explicitly stated logical statements in certificate, which can securely move around from entity to entity. We have found an elegant way to separate authorization policies from the application and system software. This separation greatly helps to build system-wide policy engineering tools for human users (managers, individuals and system administrators) to change policies and policy models of a system. For more policy engineering issues in CBASS, please refer to [3]. In this paper, we present our design and implementation of the *Certificate Based Authorization Simulation System* (CBASS), discuss how certificates are used to provide non-transitive trust, solve Trojan Horse problem and manage authorization policies in DAC systems.

2 Certificate based authorization overview

In the real world, a certificate is a document attesting to the truth of certain stated facts or certifying that a person may officially practice in certain professions. Figure 1 shows a template of a typical certificate.

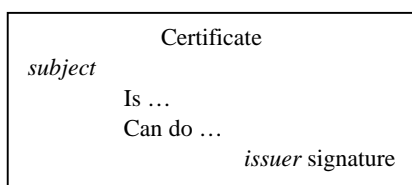


Figure 1. A typical certificate

In computer networks, a certificate usually refers to an electronically signed message used to delegate trust among applications. The delegation is implemented with the help of cryptographic techniques, such as generation and validation of public-private key pairs and digital signatures. The *subject* of an electronic certificate is usually a public key of an entity, which is the holder of the certificate and has obtained *permissions* to be something or to do something based on the facts stated in the certificate. The *issuer*, who may be the owner of a resource or a *trusted third party* (TTP), has the authority to delegate permissions. The signature on a certificate is generated by first hashing the contents of the certificate into a digest, then encrypting the digest with the issuer's private key.

Certificates were initially introduced into the Internet PKI X.509 standards for public key distribution. A certificate cannot only say "subject is ...", which binds a name with a public key; it can also say "subject can do ...", which binds a key with permissions. Therefore theoretically certificates cannot only be used for key distribution, they can also be used for capability distribution. Unlike conventional capabilities access control mechanism, certificates are digitally signed. Therefore certificates can be held by non-trusted parties and passed around from entity to entity. In the *simple public key infrastructure* (SPKI) / *simple distributed security infrastructure* (SDSI) [4], certificates are categorized into *name certificates* (NC) and *authorization certificates* (AC) based on their contents. NC identifies the key holder and therefore binds the public key with the name of an entity. Certificates used in the PKI are NC to distributed public keys of entities. (Instead of using a global name for an entity in the PKI, NC is proposed to use local names in the SPKI/SDSI [4].) AC establishes authorization permissions between a subject and a resource and therefore binds access rights with the subject. It is the AC which has been proposed to delegate authorizations among applications in order to ensure the integrity of the dynamic distributed authorization.

Certificate based authorization (CBA) discussed in this paper refers to an authorization infrastructure, where AC is used both for authentication and access control simultaneously. In the CBA, AC is signed capabilities. The public key of the subject in AC identifies the entity. The access permission is described with a logical programming language and signed with the issuer's private key. When an entity tries to access a resource, the entity attaches the AC

obtained from the resource or the resource representative previously to its request and sends them together to the resource. Based on certificates provided by the requester and policies established for accessing the resource, the resource verifies the authority of the signature of the AC and determines whether to permit the request. Since keys are used to represent identities and sign messages, the key-resolving process for the issuer's signature replaces the conventional name-based authentication process. Meanwhile compliance checking [1] between the credentials attached with a request with the policy of a resource solves the access control problem.

Compared to SPKI/SDSI and trust management approaches, CBASS implements all system authorization policies with explicit logical statements in certificates in a form of rules and facts, and therefore completely separate policies from system and application software. Meanwhile, since policies can be safely moved from entity to entity, CBASS provides more flexibility to localize authorization and establish corporative policies among trusted authorizers.

3 CBASS design

In the CBASS, we prototyped a CBA based system on the present UNIX platform. Instead of using the standard access control mechanism provided by UNIX, we use certificates explicitly declaring capabilities and policies for the users, files, and processes. Both policies and credentials for the access control are described in a similar form with a logical programming language, such as Prolog. Credentials and policies are specified with the declarative semantics of the language and turned into a logical program. Compliance checking is conducted by executing the program with the procedural semantics of the language. Access control can be achieved for each request, which becomes a query, through compliance checking process [1, 6].

3.1 Layer model

The CBASS is a system built with Java and Prolog upon the *Unix environment*.

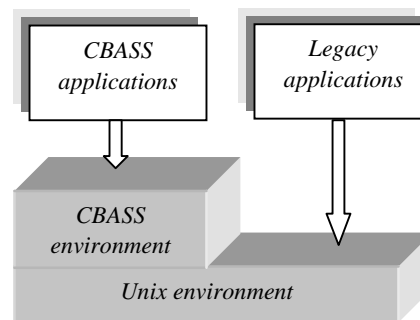


Figure 2. Layer model of CBASS

In the Figure 2, the system consists of CBASS environment, which conducts users, files, and machine management using certificates, and CBASS applications, which provide services based on the CBA mechanism. Since this is a prototype emulation system, Legacy applications still can use traditional ACL mechanism for authorization on the Unix environment.

3.2 System model

The CBASS emulates a distributed object-oriented network system comprised of *hosts* with various types of objects, including *user*, *machine*, *file server*, *application server* and *client*. The access control between any two objects is through certificates, which are managerially divided into credential certificates and policy certificates. *Credential certificates* are signed facts which can be name certificates or authorization certificates. *Policy certificates* are signed rules. Each of the objects has a certificate database for requesting resources and authorization. The objects in CBASS may be static files and running processes. A static file has a static certificate database, and for a running process, the certificate database is dynamically generated.

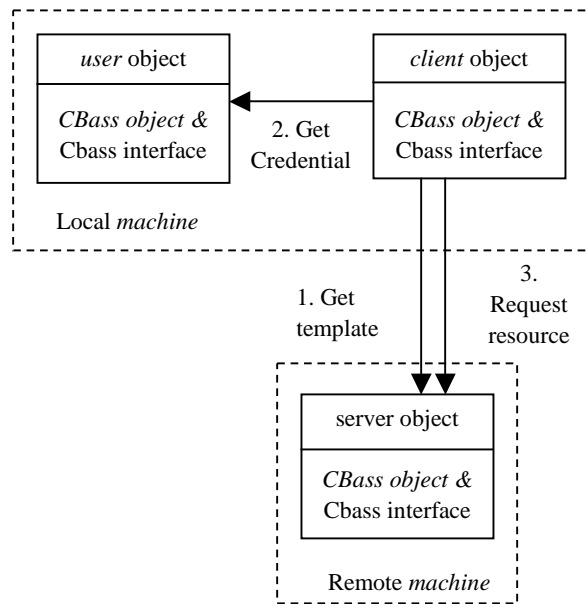


Figure 3. Template request diagram

Figure 3 shows how CBASS solves resource requesting and authorization with certificates. Objects are generally identified by *requesters* or *responders*. In the Figure 3, the *client* object is a *requester*, and the *user* and *server* objects are *responders*. The *client requester* initiates a request by asking the *server responder* a credential template which indicates *credential certificates* needed in order to authorize the request. After received the template, the *client* conducts certificate discovery, including searching its own certificate

database and getting credentials from other objects (e.g. *user* object) if necessary, then presents the certificates along with the request to the *server*. The *server* verifies the credentials against its policies and decides whether to acknowledge permission or to deny the request. In the CBASS, each object implements a *CBass interface* for communication with other objects and instantiates a *CBass object* for resource or service access control.

3.3 Object model

In the CBASS, each object has five functional modules: *certificate database* (CD), *certificate interpreter* (CI), *certificate verifier* (CV), *certificate manager* (CM) and *action* as illustrated in the Figure 4.

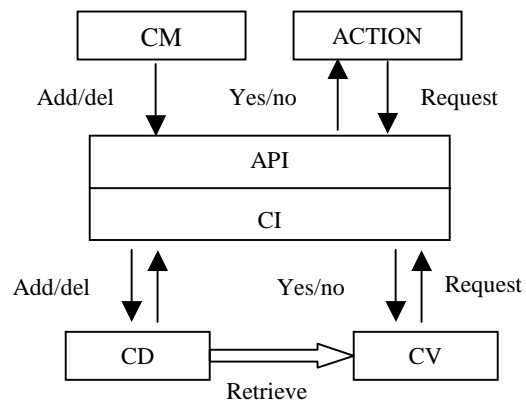


Figure 4. Object model in the CBASS

- CD stores policies and credentials, which are represented as rules and facts respectively in logical clause format.
- CI converts policies and credentials in the form of a certificate into the format of rule or fact assertions, and combines them into logical programs.
- CV accomplishes compliance checking by comparing the credentials obtained from the *action* with the policies stored in the CD by executing the logical program constructed in the CI.
- CM communicates with other objects in order to obtain or distribute certificates, and adds or deletes policies for the object;
- *Action* contains all other non-authorization functionalities specific to the object.

A generic *application programming interface* (API) is implemented to connect the *CBASS kernel* that contains CI, CD, and CV with CM or *action*. The algorithms for the kernel components are application and system independent, while the implementation of the CM and *action* usually has to rely on the specific application or system requirements. The API facilitates the integration of the CBASS authorization mechanism into the system and application

software. It allows the CM to administrate credentials in the CD. It also allows the *action* to obtain the authorization policy information for a resource access, and evaluate credentials carried in the security context for a request. In addition, the API supports certificate discovery [4].

3.4 Peer model of authorization

In the CBASS-based system, every object can be an *authorizer*, a *representative* and/or a *customer* as illustrated in Figure 5. When an object controls resources that others want to use, it is the *authorizer*. The *authorizer* is the ultimate authority over usage of these resources, and maintains local policies that determine what usages are to be allowed. The object can also serve as a *representative* when another *authorizer* defers trust to it in a policy certificate. The credentials delegated by the *representative* can then be used for authorization decisions on the *authorizer* side. When the object wants to use resources that other objects control, it plays the role of a *customer*.

As an *authorizer*, the object conducts the credential evaluation in its CV, adds/deletes policies to its own certificate database and distributes credentials to other objects through CM.

As a *customer*, the object accomplishes its certificate discovery for a certain request in the CV. The certificate discovery is a dynamic process conducted by the CM of the object. The process may include retrieving certificates from a resource authorizer or representatives in addition to its own databases.

As a *representative*, the object re-delegates credentials or distributes authorization by CM. Protocols need to be developed to serve the purpose.

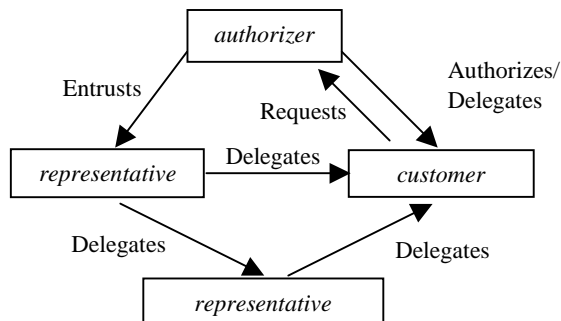


Figure 5. Direct and indirect authorization

In the Figure 5, an *authorizer* object can authorize a *customer* object by distributing credential to the *customer* directly or through intermediate *representatives*. The later indirect distribution method is that the *authorizer* entrusts part of its authority to a certain *representative*, and allows the *representative* to delegate credentials to other intermediate *representative* or *customer* object. Therefore, a certificate delegation chain may be formed.

4 CBASS implementation

CBASS is an emulated system applying certification for authorization in the operating system and applications. The implementation of the CBASS is divided into three parts: *CBASS API*, *system emulation*, and *application implementation*.

4.1 CBASS API

As we have learned from the CBASS design, every object has a CD, CV, CI, and CM to be able to play the roles of the *authorizer*, *customer*, and *representative* in the CBA based system. Since CD, CV, and CI are system and application independent, we implemented them as methods of a public *CBass* class. Each CBASS object has a *CBass* object and the CM and *actions* of the object can reuse CD, CV, and CI functions for specific system or applications. The *CBass* class is encapsulated in a package of CBASS which also contains other classes such as *Cbass Interface*, *cert* and etc.

4.2 System emulation

The system emulation in the CBASS prototype implements *machine*, *user*, and *file system* management objects upon UNIX. Instead of using a reference monitor [7], each objects has a certificate database in a form of two files, *object_name.pf*, and *object_name.cf*, for authorization and requesting resources respectively. When an object is instantiated, the static certificates stored in the two files are dynamically loaded to form its runtime certificates.

Figure 6 shows the relationship among the objects in CBASS. The *machine* object is the initial interface to the system. The object keeps passwords for all of its registered human users and maintains its authorization policy users and certificates for the communication with other *machine* objects. When a human user logs in, a *user* object is created by the *machine* object. The *user* object obtains its credential certificates from the *machine* and the static certificate database. From the machine, a credential certificate was generated stating that the logged in human user is local to the machine; from the static certificate database of the human user, credentials may indicate that which group the human user belongs to or which role the user plays. Then the *user* object stores the certificates in a runtime buffer. By presenting these credentials to the *file system* management object, the *user* may access (i.e. read, write, execute, create, delete and etc.) a file or directory by means of compliance checking between the credential certificates with the policy certificates of the *file system* management object.

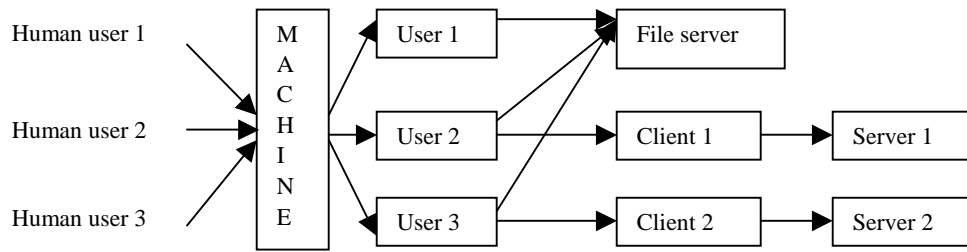


Figure 6. CBASS architecture

4.3 Application design

In the CBASS, a *user* object can also request services from an application on the local or remote *machines* by invoking the *client* object of the application. Instead of masquerading the *user*, the intermediate *client* object combines its own credentials with the credentials sent by the *user* to form a new request and a credential chain, and presents them to the application *server*. If the credentials passed by the *client* object comply with the policy certificates of the *server* object, then the *user* object may use the service. If the credentials passed by do not comply with the policies, the request would be declined. CBASS allows a service request process with a long invocation chain, where intermediate objects receive high-level requests from initiating clients and perform some series of low-level operations on a number of other services. We have implemented an application called *Cat* to display files and directories, and an application called *Clock* to display a host system clock. The following example describes how polices and credentials are generated and combined in the CBASS.

Alice is a user of a machine called *Robot*. The *Robot* can access static credential certificate database file called *Alice.cf*, which has a Prolog assertion of

```
permit(Alice, read, student_files).
```

After *Alice* logged into the *Robot*, *Robot* creates a *user* object for *Alice* and adds two credential certificates

```
permit(Alice, read student_files). // loaded from Alice.cf (1)
```

```
is_local(Alice, robot). // generated dynamically during the
login process (2)
```

to the *user Alice* object. When the *user Alice* object invokes the *Cat* that is an application to display files and directories, credentials owned by the *user Alice* object is passed on to the *Cat* object. Meanwhile another certificate

```
invoke(Alice, Cat). (3)
```

is automatically generated by the *Cat* object. The *Cat* then sends a request

```
authorize(Alice, read, filename). (4)
```

as a query along with above credentials (1), (2) and (3) to the *file server*, which is the file system management object on the *Robot*. The *file server* has authorization policies stored in the *FS.pf* file, in which there exists a policy as:

```
authorize(U, read, filename) :- is_local(U, M), permit(U,
read student_files), invoke(U, Cat). (5)
```

By executing the prolog file comprised of the policy (5) and the credentials (1) (2) and (3), the answer to the query (4) from the *file server* is “yes” and therefore the file is read and displayed by the *Cat* object to the user *Alice*.

This is a simplified example which demonstrates how certificates are used in the CBA and which is based on several assumptions. For instance, we assumed that the *file server* trusted *Cat*, and *Cat* trusted the *user Alice* object, therefore whether the *user Alice* is allowed to invoke *Cat* is not verified on the *Cat* side.

5 Discussion

The CBASS prototype establishes that the CBA does provide a flexible mechanism in a finer granularity for authorization to secure the large and open distributed network systems in the following aspects:

Non-transitive trust: the flexible certification description language allows the CBA to model various policies for a system. The policies cannot only be applied to the static objects, such as users and files, but also to the dynamic invocation processes. Therefore, an intermediate process cannot simply impersonate its invoker as most authorization systems do, but has to pass on the credential chain of previous invokers along with its own identification to the next process in order to verify the access permission at the end server. Therefore the CBASS implemented one of the important properties for authorization where trust should not be transitive. This can be seen from the example in section 4.3, in which the *file server* trusts *Cat*, the *Cat* trusts the *user Alice* object, but the *file server* does not trust *Alice* unless *Alice* has appropriate certificates such as `be(Alice, student)`, and *Alice* accesses the file through a proper invocation path.

Solution to the Trojan Horse problem: According to Gallagher [5], there is an inherent deficiency in the *discretionary access control* (DAC), which restricts access to objects based solely on the identity of subjects. In most

systems implementing the DAC, programs run by a user impersonate the user and have the DAC access rights of the user. This fundamental flaw leaves a hole for Trojan Horse attacks. However, with the CBA the influence of the Trojan Horse can be effectively limited by carefully written policies and credential certificates for the critical files, which are usually the targets for the Trojan Horse attacks. For instance, policies of critical files such as *.profile* or *password* file should be set to control not only who is eligible for the access, but also how (i.e. through a specific invocation path). In this way, even if an innocent user executes a Trojan Horse program, the critical files on his/her system are free from the attack because the access is not through an intended invocation path.

Better policy management: in the CBASS, the dynamic runtime certificate database for an object is built from the information in the static credential files and invocation relationships. Therefore, exploiting explicitly stated credential and policy certificates stored in files, CBASS can separate policy implementation from the system and application software. This separation would provide possibilities for end users instead of software developers to deal with policies exclusively without much knowledge of the underlying system or application software. With the help of the Computer Aided Policy Engineering (CAPE) tools designed in the CBASS, end users can safely change and analyze their policies of a system [3].

6 Summary and future work

In this paper, we explained our understanding of a certificate based authorization application system, introduced the exploration on the design and implementation of the distributive authorization in CBASS and discussed how the CBASS is different from traditional ACL-based computer systems. We described how the CBASS is capable of realizing non-transitive trust on the network and limiting the influence of the Trojan Horse attacks that occur in the DAC systems. Additionally, we also pointed out that policy management and deployment is critical for a CBA based system, as authorization policies and credentials are distributed all over the system and even across networks. To build system-wide policy engineering tools and guarantee the policy consistency in order to enable end users to manipulate their policies correctly is the key to put the CBA into practice [3].

CBASS is now able to conduct compliance checking for the authorization in applications and systems, and provide a user-friendly policy management interface for human users to add, delete and modify policy and credential certificates in the distributed certificate database files. However, CBASS is still at its experimenting stage, where many functions need to be developed and system performance needs to be improved. Design of the certificate

delegation and discovery is underway. Future research will focus on the system independent policy engineering tools which will enable end users to conduct policy engineering for their specific systems. Meanwhile the scalability of the CBA approach is of a great concern throughout our future research and design.

7 Acknowledgement

Thanks to Cory Stone, Jonathan Fox, Casey Gepford and Kris Provant for the discussion and implementation of the CBASS project.

8 References

- [1] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis, "The Role of Trust Management in Distributed System Security," Chapter in *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, (Vitec and Jensen, eds.), Springer-Verlag, 1999. Available at: <ftp://ftp.research.att.com/dist/mab/trustmgt.ps>.
- [2] R. Clarke, "Conventional Public Key Infrastructure: An Artefact Ill-Fitted to the Needs of the Information Society", Xamax Consultancy Pty Ltd, Canberra, Australia, November 13, 2000.
- [3] J. Dai, and J. Alves-Foss, "Authorization Policy Engineering in the CBASS," Center for Secure and Dependable Software, Technical Report (00-10 TR), Department of Computer Science, University of Idaho, December, 2000.
- [4] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen, "SPKI Certificate Theory," RFC 2693, The Internet Society, Sept. 1999, Available at: <ftp://ftp.isi.edu/in-notes/rfc2693.txt>
- [5] P. R. Gallagher, "A guide to understanding discretionary access control in trusted systems," Technical Report NCSC-TG-003 version-1, National Computer Security Center, September 1987.
- [6] N. Li, B. Grosf, and J. Feigenbaum, "A Practically Implementable and Tractable Delegation Logic", In Proc. of the 21st IEEE Symposium on Security and Privacy, Oakland CA, May 2000, Available at: <http://cs1.cs.nyu.edu/ninghui/>
- [7] Department of Defense Computer Security Center, "Department of Defense Trusted Computer System Evaluation Criteria," *DoD 5200.28-STD*, December 26, 1985