39

PMES: Privilege management and enforcement system for secure distributed resource sharing

K.J. Maly[a], *A. Gupta*[a], *I.B. Levinstein*[a], *R. Mukkamala*[a]
B. Kvande[a], *S. Nanjangud*[a], *M. Olson*[a]
R. Whitney[b], *R. Chambers*[b]

[a] *Department of Computer Science, Old Dominion University, Norfolk, Virginia 23529-0162, USA*

[b] *The Continuous Electron Beam Accelerator Facility (CEBAF), Newport News, Virginia 23606, USA*

Abstract

PMES provides privilege management and enforcement methods for end user access to distributed mass storage data and information via the Global Internet using X-windows interfaces, and supporting authentication and encryption. It permits anyone owning or managing a resource (not only a system administrator) to control fine-grained privileges on a variety of networked resources. The system consists of a Java-based privilege management interface, a server which stores privilege information, and an enforcement system. In this paper, we describe the system, emphasizing the the implementation of the management interface, the server, and the interaction of the two.

Keywords

Data security, DCE, distributed resource sharing, Kerberos, privilege management.

1 INTRODUCTION

Today, networked resources and their constituencies may span the globe. Modern research projects may involve dozens of geographically distributed collaborators who access distributed information, devices, and even communication channels. To support such widely

N. Terashima et al. (eds.), *Advanced IT Tools*

distributed activities it is necessary to provide the means to control access to distributed resources by distributed users. Such a control system depends on the services of authentication brokers such as Kerberos (Kohl and Neuman 1993) or S-key servers, but these brokers only provide mutual guarantees of the identities of user and application, guarantees which permit the user to ask for services but do not control the variety of ways in which a user may access a resource. After authentication it is up to the resource to decide what access a particular user is allowed.

The research on which we are reporting is in service of a project to develop management and access methods for end user access to distributed mass storage data and information via the Global Internet including X-windows based interfaces, authentication and encryption (Burati and Pato 1996, Diffie and Hellman 1976, Needham and Schroeder 1978, Neuman and Ts'o 1994). The initial work of the overall project is for the distribution of Experimental Physics and Industrial Control System (EPICS) code to the EPICS Collaboration community and Continuous Electron Beam Accelerator Facility (CEBAF) data to the CEBAF User community.

The project which has been in progress for the last one year has under gone several rounds of design review. The preliminary versions of the system architecture and functional specification have appeared in (Maly et al 1996a). In that paper, we described the different components of our privilege management and enforcement system (PMES). One of the first challenges that we faced during the implementation of the system was the limited access control capabilities provided by Unix. We overcame this limitation through a special command called **execute** which made the access controls of Unix transparent to the PMES users. The details of this innovation were reported in (Maly et al 1996b). Since the last report, we have implemented the other components of the system. During this process, we revised some of the component interface specifications. In this paper, we describe the details of the implementation of the privilege management and enforcement components of our system.

The paper is organized as follows. Section 2 summarizes the system architecture. Section 3 describes the implementation of the client component. The details of the kernel implementation are included in section 4. The implementation of the privilege enforcement system are described in section 5. Finally, section 6 describes the current status and future directions.

2 PMS FUNCTIONAL SPECIFICATION

Our system (Privilege Management and Enforcement System or PMES) consists of three interacting subsystems which are replicated across networks and cells: the kernel, which stores privilege information which it then provides to validate user access to resources; the Privilege Enforcement System which permits valid users to access the resources to which they are entitled; and the PMS Interface (refer to (Maly 1996) for further details). Figure 1 illustrates the interconnections among the components.

The privilege management component, which manages access to resources, is based on the concept of a **resource set**. A resource set is a 2-tuple, *(RESOURCES, USERS)*. *USERS* is a set of user identifiers, for example login identifiers or group identifiers. *RESOURCES* is, in turn, a set of 2-tuples, *(resource, permissions)*, where *resource* is some

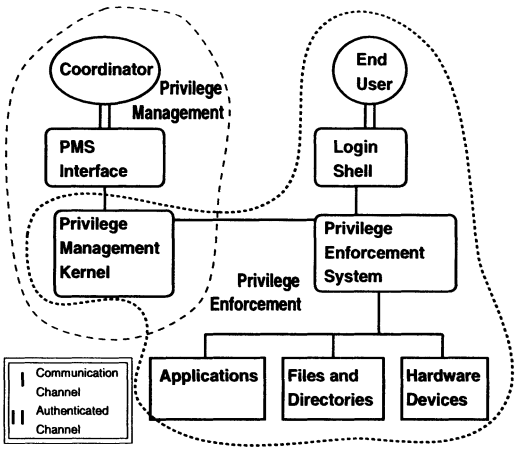

Figure 1 PMES Architecture

resource and *permissions* is a collection of permissions which have been granted on that resource. A resource may, in turn, be a tuple such as a *(program, machine)* pair which could be the basis for a right to execute a certain program but only on a particular machine. In short, a resource set identifies a selection of resources, a selection of users, and a collection of access privileges that the users have on the resources. Henceforth, we will usually refer to a resource set simply as a set.

The PMS kernel maintains the PMS access control database. Upon receiving requests from the PMS interface via the PMS daemon, the kernel interacts with the database server and makes the appropriate changes. The database is later accessed by the privilege enforcement component (PES) to either permit or deny access to a privileged resource.

The coordinator, responsible for the privilege mangement, interacts with the PMS interface to query and make changes to the underlying privilege management kernel. The PMS interface is referred to as PMS client or simply as client in the rest of the paper. The PMS kernel is referred to as the server.

Resource types currently supported by our system are directories and files, programs (executables), machines and devices. Currently supported privileges are read, write, execute, admin, and delete access as well as the combinations of permission to run a program on a particular host machine and to access a file with a particular application.

In the rest of the paper, we describe the implementation details of these components.

3 PRIVILEGE MANAGEMENT SYSTEM: CLIENT IMPLEMENTATION

The PMS client consists of a set of utilities used to interact with the PMS server that manages the privilege database. The client architecture has a layered modular design to maximize the portability and flexibility of the client and the security service it uses.

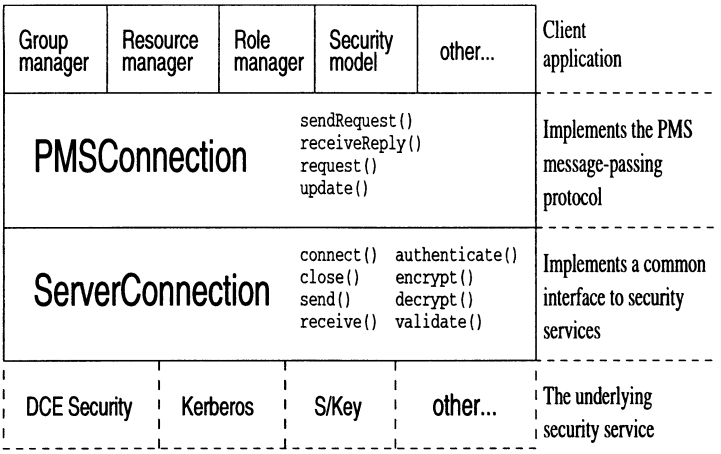

Figure 2 PMES Client Implementation

The architecture consists of three layers (Figure 2). The base component, ServerConnection, implements a common interface for the underlying security service. The middle component, PMSConnection, uses the services of ServerConnection to implement the PMS specific communication protocol. The application components reside on top of PMSConnection and interact with the coordinator to manipulate the privilege database.

3.1 The ServerConnection Component

The ServerConnection component has a common security service interface, regardless of the currently installed security service it uses. This base component is responsible for setting up a secure communication channel between the client and the server. It performs such operations as authentication of users and servers, encryption and decryption of data to prevent eavesdropping, and validation of received data to prevent against interception and modification of data in transit.

The services of the ServerConnection component are given to the user as Java methods. These Java methods are, however, implemented as C-functions calling the appropriate functions in the underlying security library. Because of security restrictions and hardware dependency of compiled C-code, the ServerConnection component has to be installed on every host computer the client application will be used on. The code will simply be recompiled for each different hardware platform, and once the code is in place the client application can be downloaded from a single common source and executed on any computer that has the Java-environment installed.

Having a common interface to the security service enables us to replace the particular underlying security library without braking the client application. With this approach the same client application can run on different locations having different security services without being rewritten.

The ServerConnection currently has the following public methods:

send(data)	send a raw data packet
data receive()	receive a raw data packet
connect(host,port)	set up connection with server, authenticating both sides
close()	close the connection with the server
encrypt(data)	encrypt the data using the current encryption algorithm implemented
data decrypt()	decrypt the data
validate(data)	check that the data came from the trusted server, and that it was not tampered with.

3.2 The PMSConnection Component

This component is purely written in Java and downloaded with the client application. It uses the services of ServerConnection and implements the specific request-reply and update message passing protocol defined between the PMS client and server.

The PMSConnection component has services for sending requests, receiving replies, and sending updates to the server. When the user calls the component's services, the type of the request and its arguments are given as parameters to the method. The request and reply data will be sent to and received from the server in a secure manner using ServerConnection's security-enabled services.

This class has the following public methods:

sendRequest(req, arglist)	sends a request to the server
data receiveData()	receives data from the server
data request(req, arglist)	combines the two above methods
update(req, paramlist)	sends an update to the server

All components in the client application requiring communication with the PMS server use the services of the PMSConnection class.

3.3 The Application Components

The application modules are written in Java and the "binary" can therefore be executed directly on any host that has the Java environment installed. Application modules that need to communicate with the server use the services of PMSConnection. Since these are implemented as request-reply and update methods, the application modules can be written to appear as close to the specifications as possible. Performance is enhanced by caching all data that is received from the server. Updates to the privilege database is sent instantly to the server for consistency reasons.

By having a defined interface for the ServerConnection class that PMSConnection relies upon, it can be replaced without breaking the application. This enables us to change the desired (or required) security system simply by replacing ServerConnection with another implementation of this class as long as the interface is maintained.

4 IMPLEMENTATION OF PMS KERNEL

The PMS kernel maintains a PMS database which consists of a set of NIS maps which are used to determine the access rights granted to a user. We create and modify the ASCII

file associated with the NIS map and make use of NIS commands to propagate the maps to the NIS slaves.

In the current implementation we maintain the following maps.

Group ⇒ Users

Group ⇒ Resources

Coordinator ⇒ Managed Groups

NIS uses ASCII files as input and converts these files into DBM format, which are called NIS maps. Whenever a client request is received by the PMS kernel, the whole stream is passed to a set of Tcl procedures. In the Tcl procedures the option field is extracted and the relevant action executed. The modifications if any are made in the NIS input ascii files. These files are then converted to NIS maps and propagated to NIS slaves.

All modifications to the PMS database are effected through the PMS client. We define a protocol for communication between the PMS client and the PMS kernel. The protocol consists of requests and acknowledgements. All communication between the PMS client and the kernel is using secure RPC. The client and the kernel mutually authenticate each other at the beginning of each transaction.

The kernel accepts PMS client requests and operates on the PMS database by updating,modifying or retrieving information. The server interprets the following classes of requests: Updates, Queries, and Modifiers.

The client communicates the request by sending a stream of characters which contains the necessary information to interpret and execute the request. The format of the client request is:

option args ...

Currently, we support the following options.

- Request for the resource-sets managed by a coordinator
- Query the users in a particular resource-set
- Query the resources in a particular resource-set
- dd/Delete a user to a resource-set
- Add/Delete a group to a coordinator's resource-set
- Add/Delete a resource to a resource-set
- Grant/Revoke priveleges to use a particular resource in a resource-set

As part of the PMS protocol the server communicates with the client with the help of a series of predefined acknowledgements. These acknowledgements convey the present status of the request and the database, back to the client. Both positive and negative acknowledgements are used.

5 IMPLEMENTATION OF THE PRIVILEGE ENFORCEMENT SYSTEM (PES)

The PES consists of the execute command and some of the PMES kernel functions. The code for the PMES kernel is written in Tcl. The code for the execute command is written in POSIX.1 compliant standard C.

The privilege granting system can be invoked explicitly using **execute** command inter-

face, or implicitly through a UNIX shell like interface. In either case, the access verification process is transparent to users. Here, every command executed by the users needs to be preceded by the **execute** command. The execute command interfaces to the PMS system via a Distributed Name Sever like service to verify access control information. In order to use the execute command, users need to specify their privilege group either by an argument specifying the privilege group to be used for a particular transaction, or optionally this may be set in an environment variable at login. On execution, the execute command acquires the real user id (uid) of the user and looks up the appropriate login id via the name service. It then looks up that login id in a PMS specific access control name service database, acquires the groups and privileges of the user, and grants/denies access as appropriate. The execute command maintains an audit trail of the command executed and the group that was used to grant the privilege for the purpose of cross verification and accounting.

For example, to run a program called analyze using privileges as a researcher, the command line interface would be: **execute -researcher analyze <arguments>**

The same command could be executed in two steps by defining the group and then executing the command. This approach is useful if the users is going to execute a number of commands under the same PMS privilege group.

The PES is charged with controlling access to machines, devices, software, and data. Machine Access is limited by using the netgroup NIS map and standard password file restrictions. The PMES kernel adds users with privileges to certain machines to the netgroups controlling those machines and the UNIX login system handles the actual access control based on the contents of the netgroup NIS map. Access to devices, software, and data are all currently handled by the execute command.

The execute command handles all privileges the same way; the only differentiation it makes is between read and write permissions to file sets. The authorization data used by execute is stored in NIS maps by the PMES kernel. The data is read and privileges are determined at the time of execution. No system has yet been provided for the event of privilege modification after execution, nor for execution past set access times. The PES grants access by changing the user and group id's of the account needing to access the data. To grant write permissions, the User ID is changed to the User ID of the writeable file set. This effectively limits write access to one file set per execution. Read and Execute Access is granted by setting the primary or one of the supplimentary group ID's to the group ID of readable file set. Eventually a certain amount of automatic parameter determination will be done by the execute command. At present this is not implemented and file sets must be explicitly specified.

6 CONCLUSION

In this paper, we have highlighted the implementation details of our PMES system. To date we have established secure DCE cells (Mullan 1996) at both sites and have achieved cross-cell PMES control on selected resources for privileges which correspond to those of the underlying UNIX operating system. For example, a user who has no operating system level permission to run a particular application but who has been given permission via the PM Interface can run it using the *execute* command.

We are currently working on introducing more comprehensive privilege enforcement features such as concurrent management and time-based access control.

REFERENCES

Burati, M. and Pato, J. (1996). RFC 91.0: User-to-User Authentication – Functional Specification. January, 1996.

Diffie, W. and Hellman, M.E. (1976). New Directions in Cryptography. *IEEE Trans. Informations Theory*, Vol. IT-22, pp. 644-54.

Kohl, J. and Neuman, C. (1993). RFC 1510: The Kerberos Network Authentication Service (V5).

Mullan, S. (1996). RFC 92.0: DCE Interoperability with Kerberos. January, 1996.

Maly, K.J., Gupta A., Kvande, B., Levinstein, I.B., Mukkamala, R., and Olson, M. (1996). A Privilege Management System For A Secure Network. To appear in *Proc. Third International Workshop on Services in Distributed and Networked Environments (SDNE '96)*, 3-4 June 1996, Macau.

Maly, K.J., Gupta A., Kvande, B., Levinstein, I.B., Mukkamala, R., Nanjangud, S., and Olson, M. (1996). A Privilege Management And Enforcement System For Distributed Resource Sharing. To appear in *Proc. International Workshop on Enterprise Security*, June 19-21, Stanford University, California.

Needham, R.M. and Schroeder, M.D. (1978). Using encryption for authentication in large networks of computers. *Communications of ACM*, vol. 21, pp. 993-9, 1978.

Neuman, C., and Ts'o, T. (1994). Kerberos: An Authentication Service for Computer Networks. *IEEE Communications,* Vol. 32, No. 9, September 1994

BIOGRAPHY

Kurt J. Maly received the Dipl. Ing. degree from the Technical University of Vienna, Austria, and the M.S. and Ph.D. Degrees from the Courant Institute of Mathematical Sciences, New York University, New York, NY.

He is Kaufman Professor and Chair of Computer Science at Old Dominion University, Norfolk, VA. Before that, he was at the University of Minnesota, both as faculty member and Chair. His research interests include modeling and simulation, very high-performance networks protocols, reliability, interactive multimedia remote instruction, Internet resource access, and software maintenance. His research has been supported by DARPA, NSF, NASA, CIT, ARPA and the U.S. Navy among others.

Dr. Maly is a member of the IEEE Computer Society and the Association for Computing Machinery.