

A Smart Card Login Module for Java Authentication and Authorization Service

EXTENDED ABSTRACT

Laurent Gauteron and Pierre Girard

Gemplus, Product Security Group,
B.P. 100, 13881 Gémenos Cedex, France
{Laurent.Gauteron, Pierre.Girard}@gemplus.com

1 Introduction

The latest Java security package JAAS gives developers a way to control the behaviour of an applet and its access to the local resources on a per-user basis in addition to the classical per-origin and per-signer basis. This authorisation service is supported by an authentication component which decides who is the current user of the Java Virtual Machine.

Smart cards have been widely recognized as a efficient way to greatly improve the security of a user authentication process. In particular, it is far better than a classical password scheme. Password schemes are quite simple, but face so-called dictionaries attacks. Even if the research space is huge (like if the password is alphanumeric and 8 characters long), an attacker can limit its exhaustive search to a limited list of passwords (its dictionary) that have a good probability to be chosen by the average user. The dictionary of likely password include all the English (or local language) words along with popular first names, date of birth, and so on. Practical experience show that it extremely difficult to instruct the users to chose good password. It has also been proven that giving random password to the users is a bad idea because they are difficult to remember and, most often, they are finally written on a note sticked under the user's keyboard, if not on the screen. In addition, passwords are sensitive to other attacks such as replay or eavedropping.

Fortunately, the JAAS designers have carefully disconnected the authentication task from others part of the package: this task is performed by a login module and custom ones can be added when needed. This is the classical concept of Pluggable Authentication Module (PAM).

In this paper we describe an implementation of a JAAS login module which provides a strong authentication of users by smart cards. We also detail the environment needed to manage the users base.

The paper is organized as follows: we first recall the basics on Java security and the JAAS features. In a second step we introduce the architecture of our solution which is detailed later on in two parts: one dealing with the login module, and the other dealing with the user management. Finally we present an application using JAAS that we have implemented to challenge our smart card authentication solution.

2 Java security and JAAS

On the Java platform, security is managed at the class level. Each loaded class is accepted only if it is verified successfully by the bytecode verifier. The origin of the class (i.e. its IP address) is identified and if the code appears to be signed, the signature is verified. The next step is to associate to the class a set of permissions that is determined from the security policy associated with the JVM. When a sensitive call is performed by one method belonging to the class, the access controller compares the actual class permissions with the requested permission. If the resource can be accessed by the class, the check simply returns, if not it throws a security exception.

This (oversimplified) summary of security verifications in Java shows that the security policy is code-centric, that is the permissions are granted according to code properties (signature and origin). The main reason for that is the primary purpose of Java security: defense of local resources against hostile mobile code coming from outside.

This approach is perfect for applications such as web browsers with embedded JVM. However, when it comes to classical multi-user security, this approach is no longer appropriate. In this case, the aim is to grant different rights to the same class depending on which user is currently using the class. JAAS adds this second approach and authorizes to write code-centric and user-centric security policies.

To benefit from extra privileges granted to a class for a given user, this user should authenticate himself to the JVM through a login module. The module to use can differ from application to application. The JVM use a configuration file to select the correct identification module in a given context. More than one login module can be used, in this case, the user need to be successfully authenticated by all login modules.

The login phase itself tries to authenticate a user represented by an object of class `Subject`. Once authenticated, an action can be performed using the additional rights granted to a user by using the static method `doAs` of `Subject` class. This method associate an authenticated user with the current access control context. Hence, subsequent access control checks will be based upon both code and user.

3 Architecture of a smart card based authentication

Figure 1 gives an overview of our login module and its environment. The login module in itself implement the `javax.security.auth.spi.LoginModule` interface in pure Java using the OpenCard Framework packages to communicate with a user's smart card which holds his keys and credentials.

4 The login module

We have chosen to use the latest public key enabled smart card in the Gemplus product range: the GPK8000 card. As card services are available for this card, it is far easier to exploit the GPK8000 by this mean than by dealing with low level Application Protocol Data Unit.

The authentication protocol between the card and the login module is based on a classical challenge/response exchange based on public key cryptography. The card

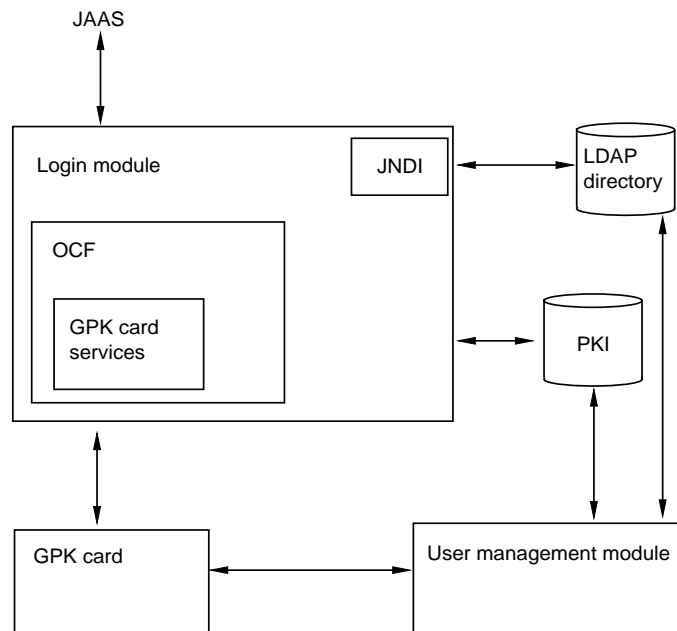


Fig. 1. The authentication architecture

contained a private/public RSA key pair which is used to sign a random challenge sent by the login module. The login verifies the card signature and if the verification is correct, considers the user as authenticated.

For the login module, there are two remaining issues: how to associate a user with a public key and how to obtain the user's public key. The association between a user and its public key is usually done by a Certification Authority who issues signed certificates which tie a user identity with its public key. The Public Key Infrastructure also maintains a list of revoked certificates.

The certificates can be carried by the user's card, or obtained somewhere on the local network in a LDAP directory or in a directory maintained by the Certification Authority. In any case, the login module needs to contain the public key of the Certification Authority. This key will be used to verify the certificates.

5 The environment

In addition to the above mentioned PKI and directory, a tool is needed to administrate the users base. This tool maintain the list of authorized users and help the administrator to add and remove users, update their rights, etc.

This tool outputs the security policy of the JVM, and personalizes or modifies users' smart card. It also communicates with the PKI and the directory to obtain new certificates for new users, to revoke certificates of deleted users, etc.

6 Using JAAS

The smart card login module and the user centric security policy usage is demonstrated through a public mail access application.

This application runs continuously on publicly accessible terminals allowing users to check their mail inside a campus or a wide organisation.

They authenticate themselves with their smart card (potentially their smart ID or organization badge) and then check their mail. With a user-centric security policy and JAAS, it is not necessary for the users to log-on the underlying operating system. Once authenticated, they are authorized to access their own files, but the access controller guaranty that access to other files is prohibited.

7 Conclusion

We have introduced a new smart card base login module for JAAS and its support environment. The check mail application demonstrates the interest of a user centric security policy and tests our login module.

References

1. Charlie Lai, Li Gong, Larry Koved, Anthony Nadalin, and Roland Schemers. User authentication and authorization in the java platform. In *Fifteenth Annual Computer Security Application Conference*, pages 285–290. IEEE Computer Society, December 1999.