A security architecture for electronic commerce applications

Bart De Win, Jan Van den Bergh, Frank Matthijs, Bart De Decker, Wouter Joosen
Department of Computer Science, K. U. Leuven
Celestijnenlaan 200A, B-3001 Leuven, Belgium
{bartd, janv, frankm, bart, wouter}@cs.kuleuven.ac.be

Key words: electronic commerce, security, transparent

Abstract: On the Internet many electronic commerce applications can be used today, but most of them provide only weak security or even none whatsoever. A major cause of this problem is the variety of technologies used to create such applications. Most existing security architectures are not designed to work in different environments.
In this paper, we propose a security architecture for generic electronic commerce applications. This architecture is open enough to be able to cope with the different security and technology requirements of today's electronic commerce applications and it is ready for tomorrow's new systems.

1. INTRODUCTION

When we look at today's electronic commerce market we see an enormous variety of applications written in several languages running on top of different systems. Most of the applications work quite well, but it is hard to trust them since they lack strong security. There are different types of shortcomings in existing applications, among which:

- parties are not able to trust each other because of the uncertainty of the correct identity
- there is no simple way of allowing or denying access to certain resources in a system

- generation of proof of certain events is not possible
- storing sensitive information in a secure manner is not easy and adequate support is not always available

To solve these problems several good security systems have been built, but they all have their own restrictions that often do not correspond with the specific requirements of electronic commerce applications. Therefor we would like to have a security architecture that is capable of securing electronic commerce applications in different environments. For example since a variety of communication subsystems are used in electronic commerce, the security service should be able to co-operate with these different communication systems. And more important we would like to secure in a transparent way. This guarantees that developers only need to concentrate on the application and that existing applications do not have to be modified.

This paper presents an architecture that is capable of transparently securing electronic commerce applications in different environments. First we give an overview of the security requirements of modern electronic commerce applications. Then we discuss why existing security architectures are not satisfactory. Next we discuss an architecture that deals with this lack of functionality. Finally we end this paper with a conclusion and several ideas for future work.

2. REQUIREMENTS FOR A SECURITY SERVICE

In this section we will overview the requirements of a security service for electronic commerce applications. We will first analyse the security requirements, which are basically the security requirements of a distributed system, extended with some specific features. Afterwards, other non-functional requirements will be discussed.

2.1 Security requirements

On the Internet people cannot be sure of the other party's identity. Masquerading techniques pose a real problem since customers are not willing to spend money in stores they cannot trust. *Mutual authentication* guarantees the correct identity of both parties. However sometimes it suffices to be able to recover the correct identity of the misbehaving party. This allows for schemes where people can shop *anonymously*.

To prevent misuse an *authorisation* mechanism must be available. This authorisation can be based on the identity of the user or on a particular role

the user has in the system. Some applications also require *delegation of rights*. For instance, an agent searching the web to find the best deal must be able to access information for you under the same conditions.

It is often desirable to *log* certain events in the system. These events can be predefined security system events or user defined application events. Naturally, the log files should be secured to prevent unauthorised modification.

The client doesn't want to reveal secret information to the rest of the world. *Confidentiality* might be necessary in different circumstances.

Likewise guaranteeing the *integrity* of information is necessary. Imagine that it would be possible to change the amount of products a client wants to order, or the total amount to pay on an invoice.

Because of the intrinsic insecure character of public communication networks and because of the existing lack of trust in other parties, *generation of proof* is required frequently. The combination of non-repudiation and auditing offers the possibility of *automatic proof* generation to show that certain events happened. For example if a store sends an invoice with a certain amount to pay, then the client might need to prove that the store settled for that price.

Most of the above properties dealt with building a trust relationship between different parties and securing communication between them. However these efforts are useless if the application's data storage cannot be trusted. Therefor a general mechanism to securely store confidential information must be available.

2.2 Non-functional requirements

The previous paragraph described the functionality of the security service. The non-functional requirements, properties orthogonal to the functionality of a system, are also important.

In our opinion a very important non-functional requirement of a security service for electronic commerce applications is its *application independence*. It should be possible to add security to existing applications transparently. This separation of concerns helps the developer to focus on the functionality of the application without having to worry about the non-functional requirements (security in his case). In order to achieve this, the binding between the application and the service must be as minimal as possible. Of course some functionality of the security service, like the generation of proof, can not be used without the application's awareness.

Generality is another important requirement of the system. Security is not really a static business. A lot of researchers are working in this area and new techniques and algorithms will be designed. The service has to be *open*

enough to be able to work with this new technology in the future. As such it is wise to aim at a framework rather than at a specific solution.

Since the Internet is an open network with many different communication technologies the service must be *communication independent*. This is certainly true for electronic commerce applications since in practice different subsystems are already being used (e.g. HTTP, SSL, CORBA, etc.) and since different steps in an electronic commerce scenario often require different communication subsystems. The catalogue of a shop for example is frequently transferred via HTTP, while payment information is often sent via SSL or via a secure medium using proprietary communication systems.

The security service should be *dynamically configurable*. Not all communication paths will have the same requirements and it might be necessary to change them on the fly. Therefor it should be possible to specify the security policies for every method call separately.

Since electronic commerce applications are distributed systems, typical requirements in that world, like *scalability* and *fault-tolerance*, are also important here.

Last but not least, it is not necessary to reinvent the wheel. Several security architectures have good properties and we would like to *reuse* them as much as possible. Some of them will be discussed in the next section.

3. EXISTING SYSTEMS

In this section we take a closer look at some existing security architectures. We will discuss them briefly and explain why they were not satisfactory. Note that it is not our intention to discuss every existing security system. Instead we try to cover a great variety of systems. As far as we know no security architecture exists that satisfies all the requirements discussed in the previous section.

3.1 Sesame

SESAME [7] is an acronym for “A Secure European System for Applications in a Multi-vendor System” and it is the result of a European collaboration between Bull, ICL and Siemens together with some leading European research groups. It is a secure multi-domain distributed architecture that aims to provide a full set of security services to the user, across a multi-domain distributed environment.

SESAME is described to have no real limitations [1]. It is scalable across multi domains through its use of public key cryptography. It supports single

login for a user, and single storage for authentication and privilege information. Delegation and auditing across the SESAME system is provided, and it allows for plug in substitution of cryptographic algorithms and authentication schemes. SESAME seems to have all the necessary security requirements discussed in the previous section.

Unfortunately there are some non-functional disadvantages. The most important one is its obligatory application awareness, since applications must be changed (a process that is called *sesamization*) in order to use SESAME. Another problem with SESAME is its communication subsystem. An application uses the Secure Association Context Manager (SACM) for its secured client-server interaction. The communication technology used by the SACM can not be changed easily. Adapting SESAME to overcome these problems would require a lot of work.

3.2 CORBA Security Service

The CORBA Security Service specification [5] is one of the services surrounding the CORBA specification. The goal of the security service specification is to control different security aspects of complex distributed systems and separate those aspects as much as possible from the functionality of the application. Distributed method calls are transparently secured.

The specification deals with several security aspects. Message protection can be applied to guarantee the confidentiality and integrity of messages with critical information. The service provides access control, which can be used implicitly or when the application requests it. The service also offers the possibility to audit security relevant events, system or user defined. Non-repudiation features are included in the service.

The CORBA security service is well designed and most of the requirements from the previous section are met. However there are some disadvantages. First, the specification is based on CORBA and is as such not communication independent. Second, there is no real notion of anonymity. The specification states that *an identity may be revealed to some parties, but not to other*. This provides no certainty at all afterwards. Third, the policy granularity is defined at connection scope and not at message scope. This may be too restrictive for electronic commerce. And last, the specification provides little support for other security needs, like secure storage etc.

3.3 GSS-API

The General Security Service API [2] (GSS-API) is a security service that works on behalf of communication systems and provides authentication,

confidentiality and integrity. Communication systems must be made security aware to use this service.

GSS has several advantages over other security systems. GSS is communication system independent. Second, GSS provides support for anonymous communication. And last, GSS is not dependent of a specific trusted computing base.

Unfortunately there are several reasons why GSS is not really usable for electronic commerce applications. It is designed to secure communication only. As such it provides no authorisation, non-repudiation, nor any other supporting service. Moreover principals must be authenticated externally before they can use GSS. Adapting or extending the system to fit our needs would require a lot of work.

3.4 The Java family

Java [8] provides several security specifications that are relevant for distributed applications. JCA/JCE offers low level support for security actions, such as encryption/decryption, certificate handling, MAC's, etc. The JAAS specification adds support of authentication and authorisation to Java based on the principal who runs Java code. And the recently proposed Java RMI Security Extension protects communication providing confidentiality, integrity and distributed authentication.

The discussed specifications separately provide not enough functionality, but they can be combined to construct a powerful security system, which seems a promising approach. The only missing requirements are auditing and non-repudiation. However, these can be built using the discussed API's. Unfortunately JAAS and especially the RMI Security Extension are still in design phase and are as such not stable and usable at the time of this writing. This also makes it very difficult to look at the other non-functional requirements.

4. SECURITY SERVICE ARCHITECTURE

The systems presented in the previous section are not really good solutions for a general electronic commerce application. Either the security functionality is not satisfactory or there are other non-functional constraints that are too restrictive. Therefor we decided to design a system that would meet all the requirements that were discussed in section 2.

4.1 Architecture

The security architecture presented here is proposed in [3]. It is not our intention to discuss the details of the architecture in this section. Instead only the most important principles will be reviewed here.

The services of the security architecture can be used in two different ways. First security aware applications can invoke certain services by using specific interfaces. Generation of proof is a typical example since the security service cannot guess when this is required. And second the communication of security unaware applications can be protected transparently. Special interceptors are used to examine passing method calls and identify specific information about them. Objects that represent this information are instantiated by the interceptors. After this reification process the security architecture is triggered for further processing.

Naturally specification and application of security properties is an important part of the security service. The mechanism must be easy to use and still be general enough to cope with changes. For this purpose the system uses the three abstractions discussed next.

A *constraint* contains all relevant information about a security property. To give an idea the following code is a fictive example of a confidentiality constraint:

```
Confidentiality {
  Requires: authentication
  Method: RSA
  Key length: 512 bits
}
```

Every security property requires different information. For example an authorisation constraint will not need information about the key length. Therefor, different constraint types are introduced. The exact contents of a constraint type are defined by the next abstraction.

A *security module* executes a certain constraint. Since he must understand the contents of the constraint, he actually defines what options can be used. These option definitions act as a logical template for a particular constraint type. A fictive template for confidentiality constraints might look like the following:

```
Confidentiality template {
  Method: RSA | DES;
  Key length: 256 | 512 | 1024;
  Key Source: certificate | file;
}
```

Since security subsystems often require specific information, constraints can (and probably will) depend on the specific security subsystem used.

As a last abstraction, *appliance rules* indicate when constraints must be applied. These rules link a constraint with a criterion for application. The specification of this criterion is not imposed by the architecture. It is for example possible to use regular expressions based on a combination of the method name, parameters and the involved principals.

Besides these abstractions a secure communication channel with the peer security service is necessary, for example to exchange identity information and policy rules. Since the model provides fully anonymous communication, this secure channel must be anonymous when necessary. Note that the cost to set up a secure, anonymous channel is quite high. Therefor, the secure channel should be based on existing keys or other available information.

Dynamic adaptation of the security policies is one of the requirements of the security architecture. For this purpose the architecture provides a special interface. Applications that wish to dynamically add security constraints to a particular invocation can implement this interface. Registered interfaces are invoked for every invocation that passes through the security service.

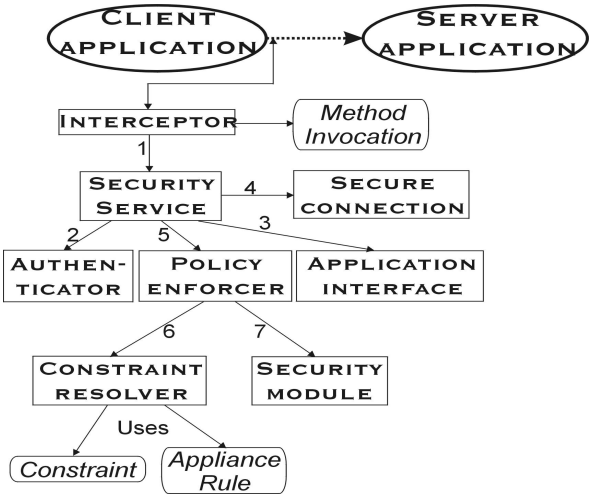

Figure 1. Scenario used to secure communication

We will now very briefly describe the dynamic behaviour of the architecture. *Figure 1* shows the execution of the security system. Suppose that a client object sends a message to a server object. An interceptor reifies this invocation, creates a method invocation object and calls the security service to process the invocation (1). The service first authenticates the principal that requested the invocation (2). After that he calls registered applications to add constraints if desired (3). The service uses the authentication information to open a secure connection to the peer security

service (4). It then passes the invocation to a policy enforcer class that co-ordinates further processing (5). The enforcer determines which constraints apply to the invocation (6) and orders respective security modules to process the invocation (7). At the end the resulting invocation is passed back to the interceptor who returns it to the communication subsystem.

4.2 Discussion

The architecture discussed in the previous section meets the requirements from section 2. It is possible to support all the security requirements that were enumerated there. The non-functional requirements are also met. The use of a general interceptor mechanism provides application and communication independence.

The architecture is open enough to allow the use of new security technologies. This is the result of the separation of security specification (constraints) and security implementation (modules). Introducing new security technology only requires a new implementation of a security module and possibly a new constraint type. This idea can be compared to the non-functional policies described in [6]. The main difference between the two techniques is an actual template generation phase in the case of non-functional policies. The templates discussed here are only logical and are part of the specification of a security module.

The security service is highly dynamic configurable for two reasons. First, constraints, modules and appliance rules are hot pluggable, which means that the service must not be restarted to extend its functionality. Second, dynamic adaptation of the security policies is possible through a specific interface discussed earlier. Today this interface only allows adding constraints. However this restriction can be removed when necessary.

This architecture combines the best of several worlds. It provides application independence as in the CORBA Security Service, which is a limiting factor in several security systems. Besides that it is also independent of the communication subsystem. And more important, the architecture is able to work with the best security systems because of its openness.

Until now some issues are not studied well enough, for example the use of regular expressions for the appliance rules. This still requires some work. Another aspect is the constraint specification for the most common security properties. This should be studied carefully to discover common features etc. A last issue is the development of a general event system to provide better support for auditing. The CORBA Security Service already contains such a system.

At the time of this writing we have not implemented this system completely, which means that some problems could show up. However, we

are convinced that the overall architecture is designed well enough to cope with most problems.

5. CONCLUSIONS

General electronic commerce applications (like electronic shops, electronic auctions) have specific security requirements that most modern security architectures are not able to provide. They lack both security and other non-functional requirements. The security architecture discussed in this paper provides a solution to this problem. We presented the most important principles of the architecture. Furthermore, we discussed its possibility to support the most important security requirements and its non-functional advantages.

We think there are two important tracks for further research. The first one is the further implementation of the proposed model in order to validate and improve the design. Second we would like to explore new techniques of adding security-related code to the application in a very simple way, for example by using Aspect Oriented Programming [2]. In combination with our security service this can turn out to be a very powerful mechanism.

References

1. Ashley P., Broom B., *A survey of secure multi-domain distributed architectures*, FIT technical report, 1997, FIT-TR-97-08
2. Aspect Oriented Programming, *Aspect Oriented Programming* Webserver, http://www.parc.xerox.com/csl/projects/aop/
3. De Win Bart, Van den Bergh Jan, Matthijs Frank, Joosen Wouter, *A security service for the electronic commerce framework*, 1999
4. Linn J., *Generic Security Service Application Program Interface (Version 2)*, RFC2078, Jan. 1997, http://www.it.kth.se/docs/rfc/rfcs/rfc2078.txt
5. OMG, *CORBAServices: Security Service Specification*, November 1996, http://www.omg.org/corba/sectrans.html
6. Robben B. et al., *Non-functional policies*, Proceedings of the Second International Conference on Metalevel Architectures and Reflection, July 1999
7. Sesame, Sesame Webserver, https://www.cosic.esat.kuleuven.ac.be/sesame/
8. Sun Microsystems, Inc., Java Webserver, http://java.sun.com/