

# Conceptual overview of the EFSOC security service

Kees Leune

Infolab Technical Report Series, no. 15

February 2004

Infolab Technical Report Series, number 15.

Version 1.0

© 2004 Kees Leune <kees@uvt.nl>

All rights reserved.

## Table of Contents

1 Introduction.....	4
2 Authentication.....	5
2.1 Initial authentication.....	5
2.2 Acquiring additional roles.....	5
2.3 Relinquishing roles.....	5
3 Authorization.....	6
3.1 Events.....	7
3.2 Basic authorization rules.....	8
Authorized sender rule.....	8
Authorized receiver rule.....	8
Receive response rule.....	9
3.3 Advanced authorization rules.....	10
3.4 Active security.....	10
4 Confidentiality and Integrity.....	11
5 System Events for security.....	12
5.1 Authentication Events.....	12
authenticationRequestEvent.....	12
acquireRoleEvent.....	12
relinquishRoleEvent.....	12
authenticationResponseEvent.....	13
5.2 Authorization Events.....	13
5.3 PKI Events.....	13
getPublicKeyEvent.....	13
validatePublicKeyEvent.....	13
getPublicKeyResponseEvent.....	14
validatePublicKeyResponseEvent.....	14
6 Acknowledgements.....	15

# 1 Introduction

EFSOC is a secure framework for event-based service oriented computing. This document will describe the security service of the EFSOC framework. For more information about the EFSOC framework, please consult the public section of [www.pronir.nl](http://www.pronir.nl). This is a working document, which means that it will grow over time.

When designing a secure framework, attention must be given to a number of security aspects.

- **Authentication:** Authentication is the process in which a subject's credentials are verified and his identity is established.
- **Authorization:** Authorization is the process to decide whether or not a subject will be granted a certain permission.
- **Integrity:** Event integrity ensures that no part of the data contained in an event is modified while it travels between sender and receiver.
- **Non-repudiation:** Non-repudiation is the quality aspect that determines that when an event is generated, the generator cannot deny that he generated it.
- **Confidentiality:** Events may only be disclosed to authorized subjects.

The security service particularly addresses the issues of authentication, authorization and to a lesser extent, non-repudiation. In chapter 2, we discuss the authentication model of the framework. Chapter 3 will cover the concepts of the authorization model that is used. Finally, chapter four addresses briefly how integrity and confidentiality is achieved.

## **2 Authentication**

An authorization request is made when a subject introduces himself to the system for the first time, if he wants to elevate his permissions by taking on additional roles, or if he desires to relinquish permissions by discontinuing certain roles. Each of these three situations will be discussed below.

### ***2.1 Initial authentication***

When a subject introduces himself for the first time, the security service needs to determine if the subject really is who he claims to be. The most simple, and therefore fairly unreliable, method is to ask for a username and password. More advanced methods have been developed using chip cards or magnetic swipe cards containing physical or biometric information of the subject.

Whichever method is chosen, the essence is still that the user offers some credentials, i.e. some kind of proof, of his identity, which is subsequently validated against information that the authentication system assumes to be true.

Once the credentials that the subject presented have been validated, he is considered to be partially authenticated.

### ***2.2 Acquiring additional roles***

Partial authentication is generally not a sufficient base to grant a subject any authorizations. Usually, a subject will not be authorized unless the authorization operation decides that the subject is active in a particular role.

The authentication operation is aware of a number of predefined roles that subjects may play. Each role is associated with a number of role assignments. When a subject requests to be granted additional roles, the authentication operation will check if the request role exists, and if it can find a role assignment that explicitly grants the subject the right to play this role.

If both criteria are met, the subject is considered to be fully authenticated.

### ***2.3 Relinquishing roles***

The least-privilege assumption requires that subjects never have more permissions than they need to perform their activities. This implies that roles should be relinquished once the permissions that are granted by them are no longer required. A subject may relinquish roles at any time.

If a subject relinquishes his last role, he is considered to be partially authenticated again.

A subject issues an authentication request containing a username and a password, which are checked against a database containing authentication credentials. If that check is successful, each role that the subject requests is checked against the role assignment database. If that check is also successful, a new session is created and the subject is given a session token.



the form of authorization rules. Authorization rules are grouped in authorization policies, which are associated with events of a certain type.

Authorization rules may be evaluated to determine if an authorization request will be granted. The algorithm that is used to determine whether or not access will be granted is extremely simple. If a rule evaluates to an explicit denial of the authorization event, access will be refused. Second, if a rule evaluates to an explicit approval of the authorization event and there is no other rule evaluation results in an explicit denial, access will be granted. Third, when no rule explicitly grants access, the request will be refused. This algorithm will ensure that only duly authorized subjects may manipulate events, with explicit denial of permissions taking precedence over explicit acceptance.

### 3.1 Events

Authorization policies are associated with events. An event has the form as shown below

```
Event
  Event Envelope
        Event Type
        Event ID
        Timestamp
        References
        Sender
  Event Body
```

Each event has an event envelope containing meta-information, such as an event type, which represents the semantic significance of the event and a timestamp, which can be used to determine historic relativity of the event. In addition to the fields in the event envelope, each event has a body which contains information that the sender deems necessary for the event receiver to accurately process the request. Assume an event that represents a loan request. The event would take the following form

```
Event
  Event Envelope
        EventType = loanRequestEvent
        EventID = 20040204193
        Timestamp = 1075890192
        Reference =
        Sender = john
  Event Body
        InterestRate = 0.05
        PaybackPeriod = 12
        LoanAmount = 10000
```

In this particular example, we use an event identifier that is generated using the date and a sequence number (February 4, 2004, sequence number 193) and a timestamp that is a standard UNIX timestamp, but any value that is meaningful to an application may be used.

The event header is determined written by the event service and used by the authorization service to determine whether or not access should be granted. The event body would typically contain information such as requested interest rate, payback period, loan amount, etc.

## 3.2 Basic authorization rules

The authorization service contains a number of basic authentication rules that may be used to compose authorization policies. This section described a number of these basic authorization rules.

### Authorized sender rule

An event may only be sent by a fully authenticated subject playing a role that is explicitly authorized to send the event. For example, assume that John is an actor who is known as a client. Furthermore, assume that clients are authorized to send loan requests. The following code fragment of Prolog code will illustrate the authorized sender rule.

```
% a subject is either a process, an actor or a service
is_subject(X) :-
    is_process(X);
    is_actor(X);
    is_service(X).

% a valid role assignment is one where the subject and the role
% are known and the role has been explicitly assigned to the
subject
valid_role_assignment(Subject, Role) :-
    is_subject(Subject),
    is_role(Role),
    subject_has_role(Subject, Role).

% a subject may send an event if he is assigned to a role that
may send
% the event.
check_authorized_sender(Subject, Event) :-
    is_subject(Subject),
    is_event(Event),
    is_role(Role),
    valid_role_assignment(Subject, Role),
    role_may_send_event(Role, Event).
```

Combine these rules with the following facts.

```
is_role(customerRole)
is_actor(john)
is_event(loanRequestEvent)
subject_has_role(john, loanRequestEvent)
role_may_send_event(customerRole, loanRequestEvent)
```

Asking the prolog compiler if john may send a loanRequestEvent will now return 'Yes'.

```
?- check_authorized_sender(john, loanRequestEvent).
Yes
```

### Authorized receiver rule

Similar to the authorized sender rule, we define the authorized receiver rule. This rule states that an event may only be received by a fully authorized subject playing a role that is explicitly authorized to receive the event. For example, assume that loanRequests may be received by a

business process called `loanProcess`. The following code fragment, combined with the fragment in the previous paragraph, illustrates the authorized receiver rule.

```
% a subject may receive an event if he is assigned to a role that
may
% receive the event.
check_authorized_receiver(Subject, Event) :-
    is_subject(Subject),
    is_event(Event),
    is_role(Role),
    valid_role_assignment(Subject, Role),
    role_may_receive_event(Role, Event).
```

Also introduce the following facts:

```
is_role(loanProcessRo).
is_process(loanProcess).
subject_has_role(loanProcess, loanProcessRole).
role_may_receive_event(loanProcessRole, loanRequestEvent).
```

Asking the Prolog compiler if the `loanProcess` may receive a `loanRequestEvent` will now return 'Yes'.

```
?- check_authorized_receiver(loanProcess, loanRequestEvent).
Yes
```

### Receive response rule

Often, a subject may only receive those events if that event is sent as a response to an event that was sent previously by the subject. This implies that the authorization service must somehow keep track of all events that are sent. As discussed previously, authorization decisions can be fully reached by inspecting the envelope of an event. In addition to a unique identifier and a timestamp, each event may also contain references to previous events. The authorization service needs to verify this information and if it proves to be correct, may use it to come to an access control decision.

Referring to the example, assume that clients may only receive `loanResponseEvents` that are sent in reaction to a `loanRequest` that was issued by them earlier. To check this, we assume that the authorization service asserts a fact each time when an event is sent. This assertion may take the following form:

```
subject_sent_event(Subject, EventID, timestamp).
```

For example

```
subject_sent_event(john, 20040204193, 1075890192).
```

Using these facts, we can implement a rule

```
check_valid_response(Subject, EventID) :-
    find_event_instance(Event, EventID),
    check_authorized_receiver(Subject, Event),
    get_event_reference(Event, Reference),
    subject_sent_event(Subject, Reference, _).
```

This assumes that two additional rules (`find_event_type` and `get_event_reference`) are available which may be used to find the instance of a certain event, resp. the reference of a subject.

### ***3.3 Advanced authorization rules***

In addition to the basic rules outlined in the previous paragraph, it is possible to add many more rules that are driven by business policies.

### ***3.4 Active security***

The EFSOC authorization model is a role-based access control model. As roles are a pivotal concept in our approach, it must be possible to dynamically create, adapt or remove roles without affecting the overall integrity of the model.

As a consequence of the highly dynamic nature of the authentication model, the security service must be able to rapidly adapt to changing security constraints. As soon as changes in the role hierarchy or in the authorization rules are implemented, active sessions that violate the new security constraints must become invalid and need to be terminated, or at least isolated. The ability to rapidly change to changing security constraints is called active security.

## 4 Confidentiality and Integrity

Confidentiality and integrity are two important aspects of any secure system. Confidentiality is the quality attribute that ensures that messages may only be viewed by duly authorized parties, generally the sender and/or the receiver. Integrity is the quality aspect which ensures that a message content does not change while it is in transit, i.e. it ensures that the message arrives in the same form as it was sent.

Confidentiality and Integrity may be achieved in a variety of ways. The most obvious and most widely available technologies are based on the concept of asymmetrical cyphers. An asymmetrical cypher contains of a public key which may be used to encrypt messages and a private key which may be used to digitally sign messages. Assume an infrastructure where a subject john wants to send an event. To do so, he contacts the event service and sends a message. John uses his private key to sign the message that he is sending, which will allow the event service to check whether the message has been altered during transport. In addition to the ability to check if the message was not altered, the security service can be queried to find out if the message was signed by the key that is known to the system and belonging to john.

In addition to digitally signing, john also encrypts the message using the public key of the event service. This ensures that only the event service can decrypt the message, thus ensuring message confidentiality. The event service will perform the same signing and encryption of messages it sends out to other subjects.

The security service provides a PKI infrastructure for public keys. This means that subjects can consult the security service to retrieve or validate the public key of a subject. This way, it is possible to ensure that only communication between fully authenticated subjects is allowed.

## 5 System Events for security

As the entire EFSOC framework is event-based, all interaction between the framework and subjects is achieved by exchanging messages. This chapter addresses which system events are supported by the security service.

Field names in [ ] are optional.

### 5.1 Authentication Events

#### **authenticationRequestEvent**

Fields:

- **username.** The username of the subject to authenticate.
- **password.** The password matching the username

Authorization policy:

- All subjects may send an authentication request
- The security service may receive an authentication request

#### **acquireRoleEvent**

Fields:

- **sessionid.** The sessionid returned by an authenticationResponseEvent when authentication was successful.
- **role.** The role that the subject requests to acquire.

Authorization policy:

- All partially authenticated subjects may send an acquireRoleEvent
- All fully authenticated subjects may send an acquireRoleEvent
- The security service may receive an acquireRoleEvent

#### **relinquishRoleEvent**

Fields:

- **sessionid:** The sessionid returned by an authenticationResponseEvent when authentication was successful.
- **role:** The role that the subject requests to relinquish.

Authorization policy:

- All partially authenticated subjects may send a relinquishRoleEvent

- All fully authenticated subjects may send a `relinquishRoleEvent`
- The security service may receive a `relinquishRoleEvent`

### **authenticationResponseEvent**

Fields:

- `result`: “success” or “failed”
- `reason`: if result is “failed”, a reason why the authentication failed.
- `[sessionid]`: if the authentication was successful,

Authorization policy:

- All subjects may receive an `authenticationResponseEvent`, providing the event is sent as a response to an `authenticationRequestEvent`.
- All subjects may receive an `authenticationResponseEvent`, providing the event is sent as a response to an `acquireRoleEvent`.
- All subjects may receive an `authenticationResponseEvent`, providing the event is sent as a response to an `relinquishRoleEvent`.
- The security service may send an `authenticationResponseEvent`.

## ***5.2 Authorization Events***

Authorization events are special events in the sense that they will mostly be used by the event service only.

## ***5.3 PKI Events***

### **getPublicKeyEvent**

Fields:

- `username`: the username of the owner of the public key that is searched.

Authorization policy:

- All subjects (also unauthenticated) may send `getPublicKeyEvents`
- The security service may receive `getPublicKeyEvents`

### **validatePublicKeyEvent**

Fields:

- `username`: the username of the owner of the public key that must be validated.
- `publickey`: the public key that must be validated.

Authorization policy:

- All subjects (also unauthenticated) may send `validatePublicKeyEvents`
- The security service may receive `validatePublicKeyEvents`

### **getPublicKeyResponseEvent**

Fields:

- `publickey`: the requested public key

Authorization policy:

- All subjects (also unauthenticated) may receive `getPublicKeyResponseEvents`, providing the event is sent as a response to a `getPublicKeyEvent`.
- The security service may send `getPublicKeyEvents`

### **validatePublicKeyResponseEvent**

Fields:

- `result`: “Success” or “Failure”
- `reason`: the reason for a failure, if any

Authorization policy:

- All subjects (also unauthenticated) may receive `validatePublicKeyResponseEvents`, providing the event is sent as a response to a `getPublicKeyEvent`.
- The security service may send `validatePublicKeyEvents`

## 6 Acknowledgements

This document could not have been written without the valuable feedback that was received from members of the Special Interest Group on Service-Oriented Computing<sup>1</sup> (SIGSOC), in particular Benedikt Kratz and Bart Orriens.

---

<sup>1</sup> <http://maximus.uvt.nl/research/sigsoc>