



Architecture of the CORBA Component Model

CORBA Security

Network Security

- ✶ **Security** (Sicherheit): Guarantee of protection (Schutz) and integrity (Unverletzlichkeit) of data
 - technical security: devices and algorithms
 - privacy (Datenschutz) also includes definition of organizational procedures (e.g. roles and rights)
 - assumption of threat by deliberate manipulation
- ✶ cf. **Safety** (Zuverlässigkeit):
 - protection against failure of components in the system (by means of quality control, fault tolerance, etc)
- ✶ open systems are insecure by nature

Threats

- ✖ Breakage of control mechanisms:
 - an authenticated user obtains information not that were not mean for her
 - a user masquerades as somebody else
- ✖ Eavesdropping
 - recording of communication between sender and receiver
- ✖ Falsification (tampering)
 - Unauthorized alteration of data
- ✖ Missing accountability
 - The user performing an action cannot be identified

Goals of CORBA Security

- ✱ **Confidentiality** of data (Vertraulichkeit)
- ✱ **Integrity** of data (Unversehrtheit)
- ✱ **Accountability** of access (Nachvollziehbarkeit)
 - Non-repudation (Nicht-Abstreitbarkeit)
- ✱ **Availability** (Verfügbarkeit): Not a goal of CORBA Security, covered by other specs (e.g. fault tolerance)



Key Features

- ✱ Identification and Authentication
- ✱ Authorization and Access Control
- ✱ Security Auditing
- ✱ Security of Communication
 - over insecure lower-layer channels
- ✱ Non-repudiation
- ✱ Administration

Architectural Goals

- ✦ Simplicity
- ✦ Consistency (with existing infrastructures)
- ✦ Scalability
- ✦ Usability for end users
- ✦ Usability for administrators
- ✦ Usability for implementors
- ✦ Flexibility of security policy
- ✦ Independence of security technology
- ✦ Application portability
- ✦ Interoperability
- ✦ Performance
- ✦ Object Orientation



Service Conformance Levels (Basic Package)

✶ Level 1: Applications are unaware of CORBA Security

- user authentication
- identity of authenticated users accessible
- Application of policies for ORB domain
- Auditing
- optionally: non-repudation (optional package)

✶ Level 2: Applications are aware of roles and privileges

- API to enforce policies within the application



Security Replacability Packages

- ✿ ORB services replacability
 - through portable interceptors
- ✿ Security service replacability
 - through implementer interfaces



Common Secure Interoperability (CSI) Packages

- ✱ CSI Level 0: Identity based policies without delegation
- ✱ CSI Level 1: Identity based policies with unrestricted delegation
- ✱ CSI Level 3: Identity & privilege based policies with restricted delegation



SECIOP Interoperability Package

- ✱ Interoperability based on enhanced GIOP/IIOP, provided the same underlying security mechanism is supported

Security Mechanisms Package

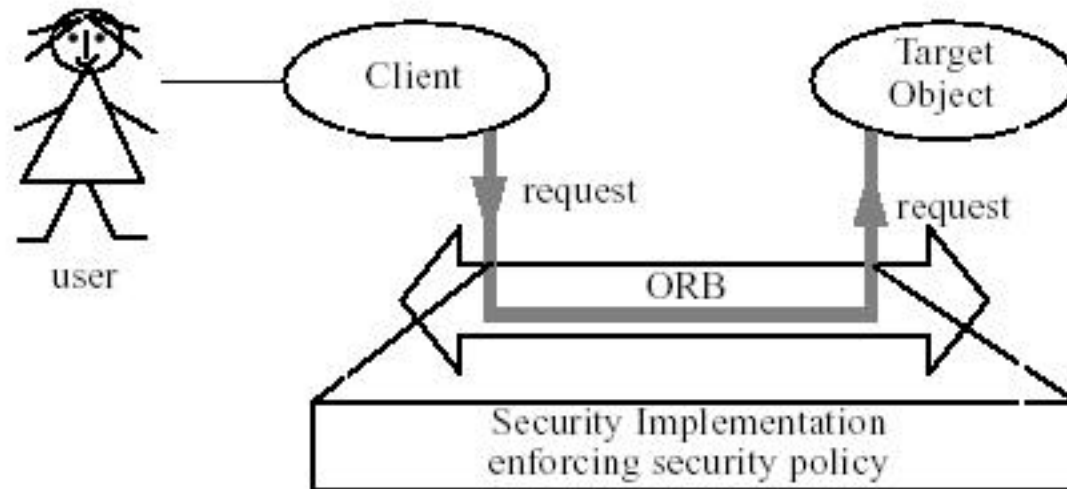
- ✧ SPKM (Simple Public Key Mechanism):
 - supports CSI level 0
 - uses SECIOP extensions
- ✧ GSS Kerberos:
 - supports CSI level 1
 - uses SECIOP extensions
- ✧ CSI-ECMA protocol:
 - CSI level 2
 - based on SECIOP
 - administrator can restrict use of delegation
 - can use either public or private key technology
- ✧ SSL:
 - CSI level 0
 - not based on SECIOP
- ✧ SECIOP + DCE CIOP

IDL Module Names

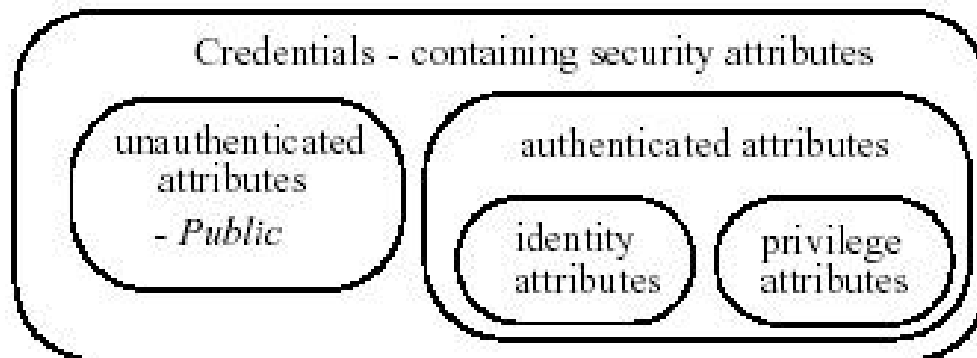
- ✱ SecurityLevel1
- ✱ SecurityLevel2
- ✱ NRservice (non-repudation)
- ✱ SecurityReplaceable
- ✱ Interceptor
- ✱ SECIOP
- ✱ SSL
- ✱ DCE_CIOPSecurity

Secure Method Call

- ✶ **Principal:** global user identity
- requires authentication
 - is associated with privileges



- ✱ **Credentials**: describe identity and privilege attributes
- ✱ **Security Association**: binding between caller and callee
 - accessible through "**SecurityCurrent**"
 - requires trusted channel
- ✱ Target object uses Current object to obtain identity and privileges of the caller
 - authorization through policies or access control lists
 - potentially auditing (communication through **audit channel**)





If target object invokes further operations:

- Call with identity and privileges of target object
- Call with identity and privileges of caller (**delegation**)
- privileges of caller and callee are combined



Non-repudiation:

- generation of **receipts** for request/response
- integration of a **delivery authority**



Message Protection

✂ Integrity:

- prevents undetected, unauthorized modification of messages
- may detect message addition, deletion, or change of order

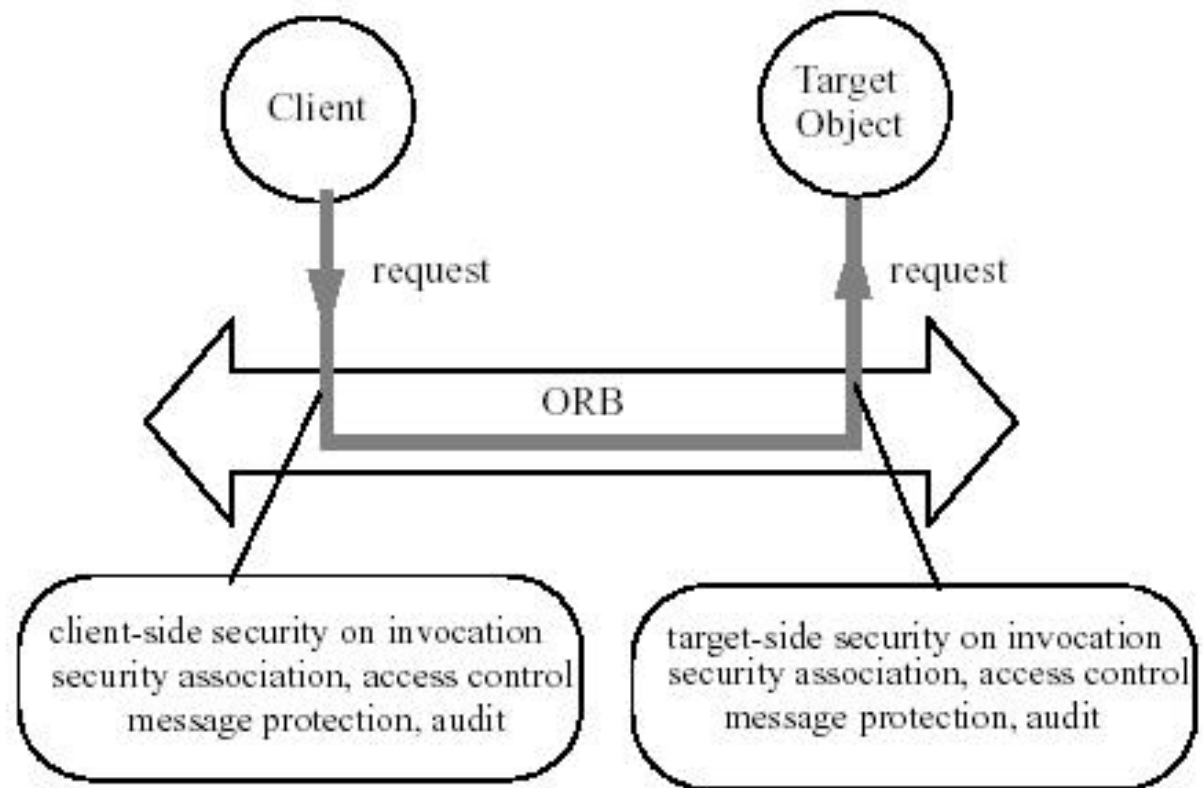
✂ Confidentiality:

- ensures that message is not read unauthorized

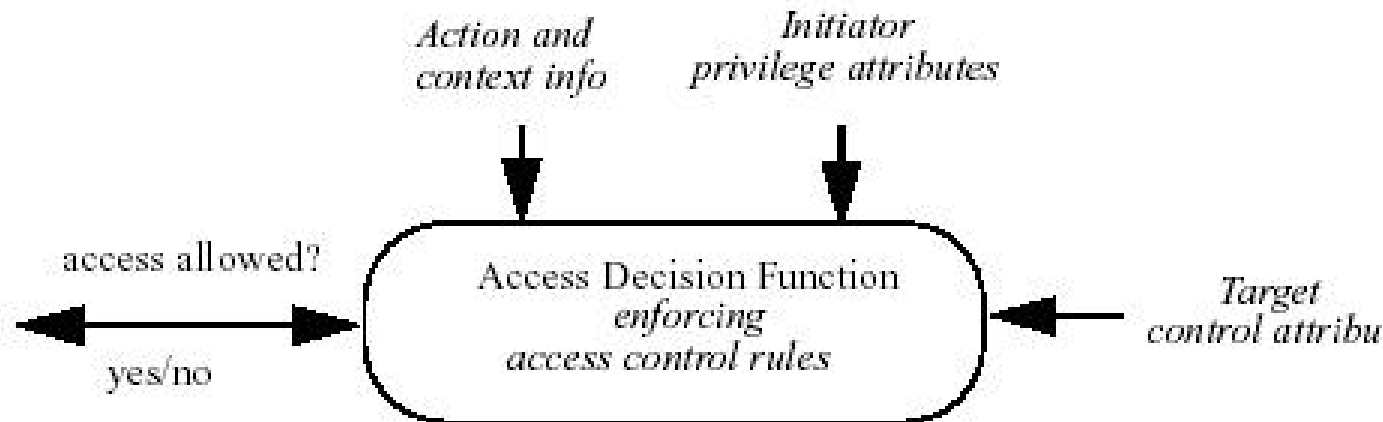
Access Control Model

- ✦ Object Invocation Access Policy
 - applied independent of application logic
 - client side restrictions, and target side restrictions
 - based on client privileges, operation (name), and object security attributes
- ✦ Application Access Policy
 - implemented in the application logic
- ✦ Privilege attributes:
 - principal's identity, roles, groups, capabilities
- ✦ Control attributes:
 - Access control lists, object classifications

Access Control (2)

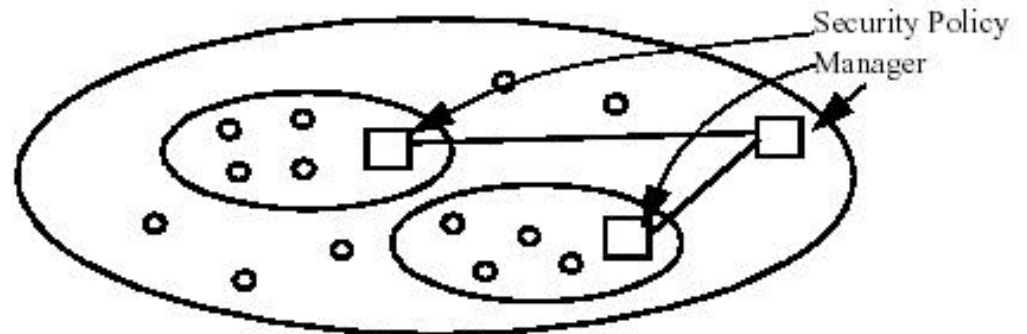


Access Decision Functions



Security Policy Domains

- ✦ Scope for which a certain set of security policies applies
- ✦ domain hierarchies
- ✦ domain federations
- ✦ management:
 - creation, deletion
 - membership of objects
 - policies associated with a domain



Authentication and Credentials

- ✶ through external **security logon**

- ✶ through PrincipalAuthenticator

- operations authenticate, continue_authentication
- produces Credentials

- ✶ Operations on Credentials: is_valid, refresh

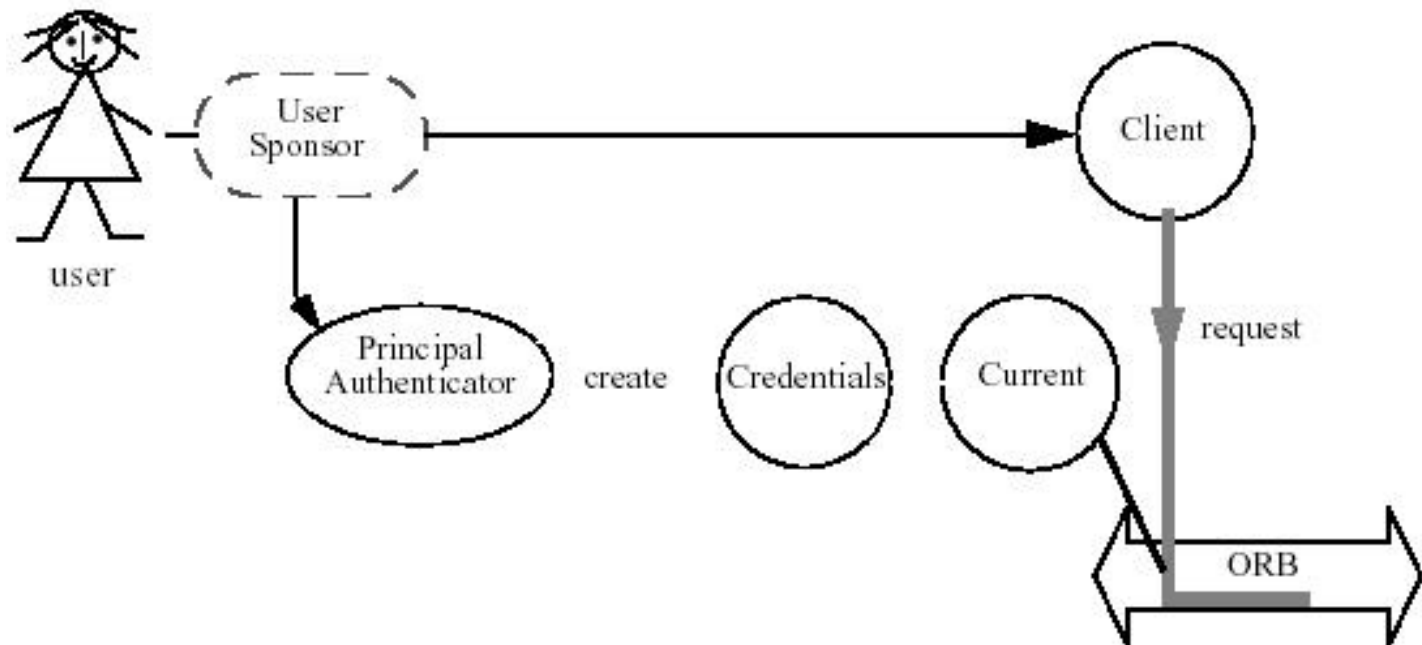
- ✶ Current object:

- either SecurityLevel1::Current or SecurityLevel2::Current
- access to credentials via get_credentials/set_credentials

- ✶ Target object computes Access_Decision object

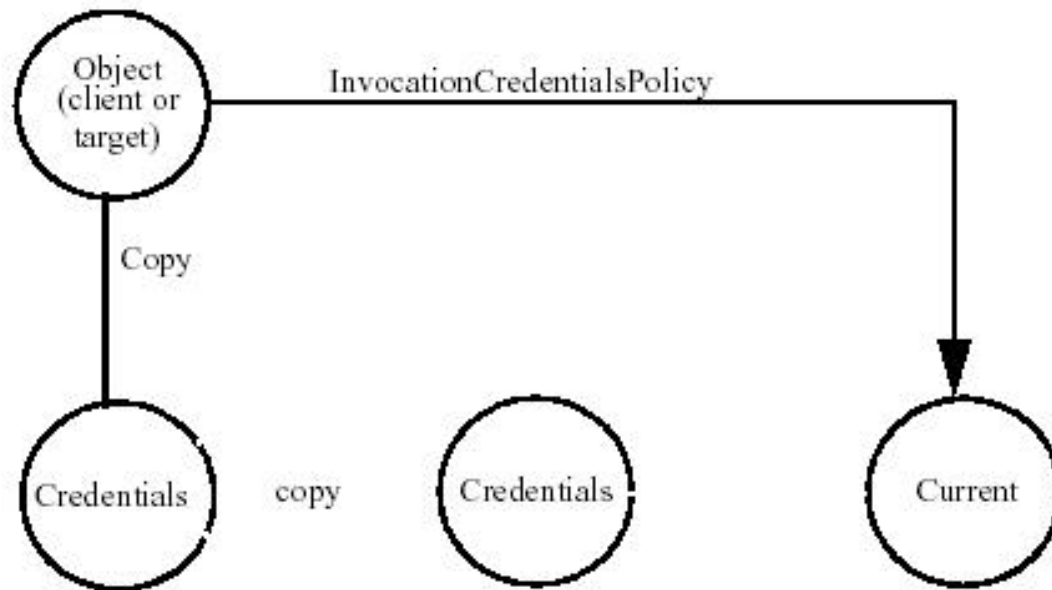
- access_allowed operation determines whether operation invocation should be rejected

Authentication



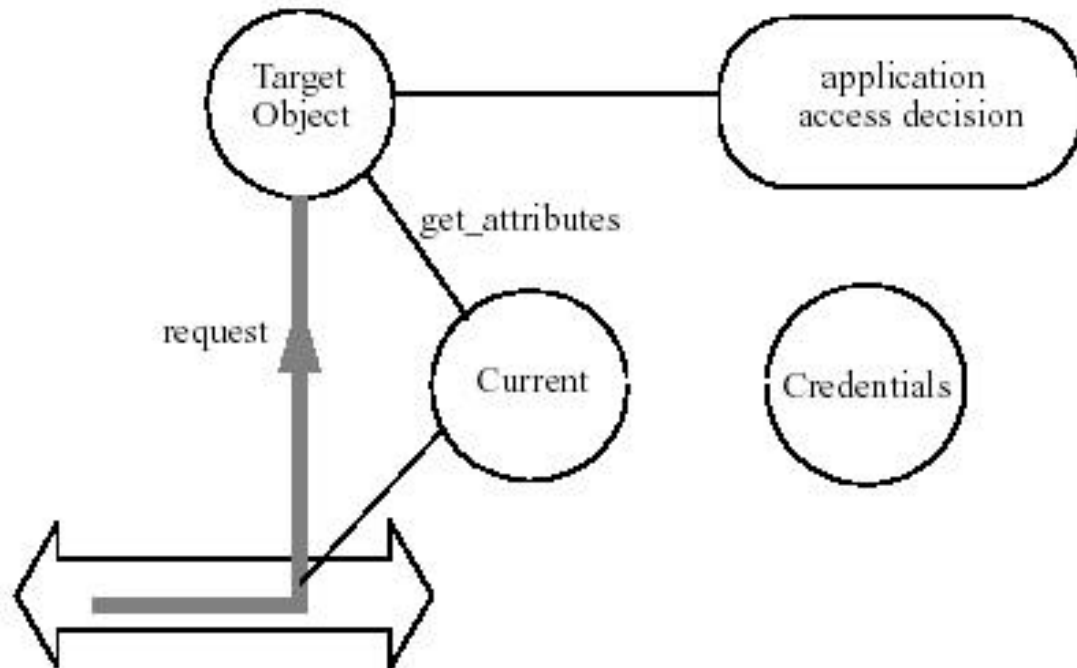
Multiple Credentials

- ✶ PolicyCurrent maintains InvocationCredentialsPolicy
- ✶ Multiple credentials can be copied, then modified



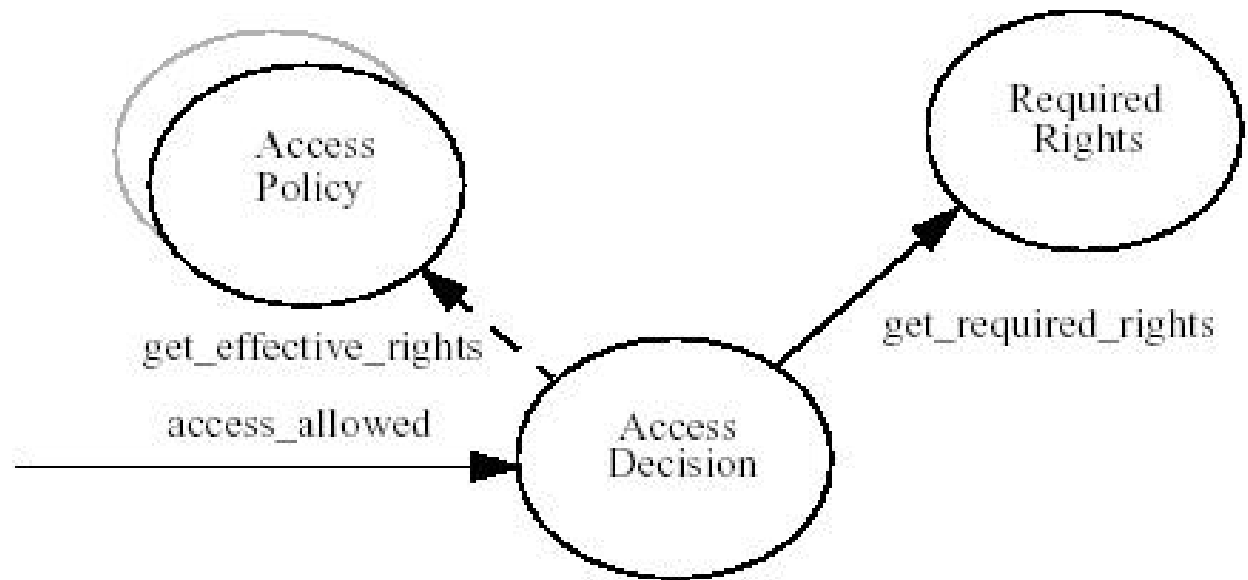
Security at the Target

- ✳ Target object uses SecurityCurrent to find security attributes



Access Decision Objects

✱ `access_allowed()` compare effective and required rights




SecurityCurrent

```
module SecurityLevel1 {  
  # pragma version SecurityLevel1 1.8  
  local interface Current : CORBA::Current {  
    # pragma version Current 1.8  
    // thread specific operations  
    Security::AttributeList get_attributes (  
      in Security::AttributeTypeList attributes  
    );  
  };  
};
```

SecurityCurrent, Level 2

```
module SecurityLevel2 {  
  local interface Current : SecurityLevel1::Current {  
    # pragma version Current 1.8  
    readonly attribute ReceivedCredentials  
    received_credentials;  
  };
```

SecurityManager



```
✂ resolve_initial_references("SecurityManager")  
local interface SecurityManager {  
    readonly attribute Security::MechandOptionsList supported_mechanisms;  
    readonly attribute CredentialsList own_credentials;  
    readonly attribute RequiredRights required_rights_object;  
    readonly attribute PrincipalAuthenticator principal_authenticator;  
    readonly attribute AccessDecision access_decision;  
    readonly attribute AuditDecision audit_decision;  
    TargetCredentials get_target_credentials (in Object obj_ref);  
    void remove_own_credentials(in Credentials creds);  
    CORBA::Policy get_security_policy (in CORBA::PolicyType policy_type);  
};
```

AccessDecision

```
local interface AccessDecision {  
    boolean access_allowed (  
        in SecurityLevel2::CredentialsList cred_list,  
        in Object target,  
        in CORBA::Identifier operation_name,  
        in CORBA::Identifier target_interface_name  
    );  
};
```

Policy Management

```
module SecurityAdmin {  
  interface DomainAccessPolicy : AccessPolicy {  
    void grant_rights(   in Security::SecAttribute priv_attr,  
                        in Security::DelegationState del_state,  
                        in Security::RightsList rights);  
    void revoke_rights( in Security::SecAttribute priv_attr,  
                       in Security::DelegationState del_state,  
                       in Security::RightsList rights);  
    void replace_rights (in Security::SecAttribute priv_attr,  
                        in Security::DelegationState del_state,  
                        in Security::RightsList rights);  
    Security::RightsList get_rights (   in Security::SecAttribute priv_attr,  
                                       in Security::DelegationState del_state,  
                                       in Security::ExtensibleFamily rights_family);  
  }  
}
```

//...

Rights

```
module Security {  
    struct ExtensibleFamily {  
        unsigned short    family_definer;  
        unsigned short    family;  
    };  
    struct Right {  
        ExtensibleFamily    rights_family;  
        string              rights_list;  
    };  
    typedef sequence <Right> RightsList;  
};
```

Rights Families

- ✶ Family definer: 0 (OMG)

- ✶ Family 1: (CORBA)

- ✶ Rights:

- „get“: read object state
- „set“: write object state
- „manage“: modification of security attributes of the object
- „use“: access to general service interface of the object

Client Example

```
securitymanager = orb->resolve_initial_references ("SecurityManager");  
Security::AuthenticationMethod our_method =  
    (Security::AuthenticationMethod)SecurityLevel2::KeyCertCAPass;  
Security::SSLKeyCertCAPass *method_struct;  
method_struct = new Security::SSLKeyCertCAPass;
```

Client Example(2)

```
// Install client certificate for authenticator
CORBA::Any* any_struct = secman -> get_method_data(our_method);
*any_struct >>= *method_struct;
method_struct -> key = "ClientKey.pem";
method_struct -> cert = "ClientCert.pem";
method_struct -> CAfile = "";
method_struct -> CAdir = "";
method_struct -> pass = "";
//obtain authenticator
SecurityLevel2::PrincipalAuthenticator_ptr pa = secman -> principal_authenticator();
```

Client Example (3)

```
const char* security_name = "ssl";  
Security::AttributeList privileges;  
SecurityLevel2::Credentials_ptr creds;  
CORBA::Any* continuation_data;  
CORBA::Any* auth_specific_data;
```

```
// authenticate
```

```
pa -> authenticate( our_method, "", security_name, *out_any_struct, privileges,  
    creds, continuation_data, auth_specific_data);
```

```
// invoke operations
```

```
...
```

Server Example

- ✦ Authenticate to primary authenticator: likewise
- ✦ In the method implementation: perform access check

```
CORBA::Object_var securitycurrent;
```

```
SecurityLevel2::Current_var seccur;
```

```
securitycurrent = orb->resolve_initial_references ("SecurityCurrent");
```

```
seccur = SecurityLevel2::Current::_narrow(securitycurrent);
```

```
SecurityLevel2::ReceivedCredentials_var rc = seccur->received_credentials();
```

Server Example(2)

```
Security::ExtensibleFamily fam;  
fam.family_definer = 0;  
fam.family = 1;  
Security::AttributeType at;  
at.attribute_family = fam;  
at.attribute_type = Security::AccessId;  
Security::AttributeTypeList atl;  
atl.length(1);  
atl[0]=at;  
Security::AttributeList_var al = rc->get_attributes( atl );
```