More patterns for operating systems access control

Eduardo B. Fernandez and John C. Sinibaldi
Dept. of Computer Science and Eng.
Florida Atlantic University
Boca Raton, FL, USA
ed@cse.fau.edu , John.Sinibaldi@radisys.com

Abstract

We present architectural patterns for access control in operating systems. These complement the patterns that we introduced in a previous paper. The patterns control access to resources represented as objects and include patterns for authentication, process creation, object creation, and object access.

Introduction

We present architectural patterns for access control in operating systems. These complement the patterns that we introduced in [Fer02]. That paper presented the following patterns:

- **File access control**. How do you control access to files in an operating system? Apply the Authorization pattern to describe access to files by subjects. The protection object is now a file component that may be a directory or a file.
- **Controlled Virtual Address Space**. How to control access by processes to specific areas of their virtual address space (VAS) according to a set of predefined access types? Divide the VAS into segments that correspond to logical units in the programs. Use special words (*descriptors*) to represent access rights for these segments.
- **Reference Monitor**. How to enforce authorizations when a process requests access to an object? Define an abstract process that intercepts all requests for resources and checks them for compliance with authorizations.
- **Controlled Execution Environment**. How to define an execution environment for processes? Attach to each process a set of descriptors that represent the rights of the process. Use the Reference Monitor to enforce access.

Here we add the following patterns:

- **Authenticator.** How to verify that a subject is who it says it is? Use a single point of access to receive the interactions of a subject with the system and apply a protocol to verify the identity of the subject.
- **Controlled-Process Creator**. How to define the rights to be given to a new process? Define their rights as part of their creation.

- **Controlled-Object Factory**. How to specify rights of processes with respect to a new object? When a process creates a new object through a Factory, the request includes the features of the new object. Among these features include a list of rights to access the object.
- **Controlled-Object Monitor**. How to control access by a subject to an object? Use a reference monitor to intercept access requests from processes. The reference monitor checks if the process has the requested type of access to the object.

Assume here that resources are represented as objects, as it is common in modern operating systems. Figure 1 shows how these patterns are organized into a pattern language. For example, Authentication is needed for file access and for controlled object access, a subject must be authorized to access some object in a specific way and we need to make sure that the requestor is not an impostor. The other three patterns complete the definition of the Controlled Execution Environment, where now the creation and access to objects are controlled. The language also shows that access to files is controlled by a Reference Monitor.

Background

Operating systems are fundamental to provide security to computing systems. The operating system supports the execution of applications and any security constraints defined at that level must be enforced by the operating system. The operating system must also protect itself because compromise would give access to all the user accounts and all the data in their files. A weak operating system would allow hackers access not only to data in the operating system files but data in database systems that use the services of the operating system. The operating system isolates processes from each other, protects the permanent data stored in its files, and provides controlled access to shared resources. Most operating systems use the access matrix as security model. An access matrix defines which processes (subjects in general) have what types of access to specific resources (resources are represented as objects in modern operating systems). To apply this model we need to make sure that subjects are authenticated before they perform any access (using the Authenticator). Processes are the active units that perform computational work and use resources, we need to control the rights given to each process when created (Controlled-Process Creator), and to let processes execute in a controlled environment where they cannot exceed their rights (Controlled Execution Environment). We also need to define access rights to access new objects (Controlled-Object Factory), and to control access to objects at execution time (Controlled-Object Monitor). This latter performs access control by intercepting requests and checking them for authorization. All these functions are the purpose of the patterns presented in these two papers.

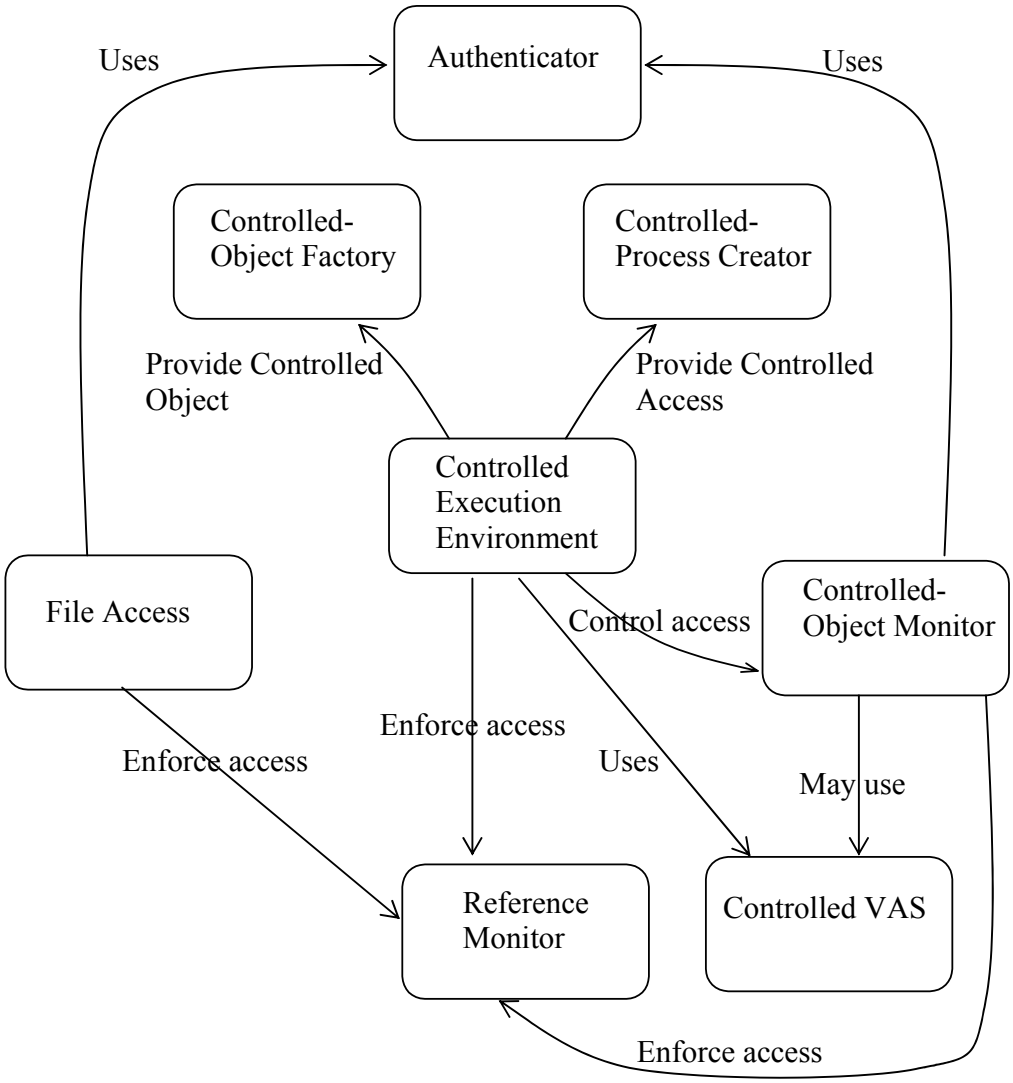

Figure 1. O.S. Access Control Pattern Language

Operating systems authenticate users when they first login and maybe again when they access specific resources. A user then executes an application composed of several concurrent processes. Processes are usually created through system calls to the operating system [Sil03]. A process that needs to create a new process gets the operating system to create a child process that is given access to some resources. Executing applications need to create objects for their work. Some objects are created at program initialization while others are created dynamically during execution. The access rights of processes with respect to objects must be defined when these processes are created. Applications also

need resources such as I/O devices and others that may come from resource pools; when these resources are allocated the application must be given rights for them. These rights are defined by authorization rules or policies that must be enforced when a process attempts to access an object. This means that we need to intercept every access request; this is done by the Reference Monitor.

Authenticator

Intent

How to verify that a user (subject) is who it says it is?

Context

The operating system controls the creation of a session in response to the request by a subject, typically a user. The authenticated user (represented by processes running on its behalf) is then allowed to access resources according to her rights. Sensitive resource access may require additional process authentication. Processes in distributed operating systems also need to be authenticated when they attempt to access resources in external nodes.

Problem

How to prevent impostors from accessing our system? A malicious attacker could try to impersonate a legitimate user to have access to her resources. This could be particularly serious if the impersonated user has a high level of privilege.

Forces

We need to apply these forces:

- There is a variety of users that may require different ways to authenticate them. We need to be able to handle all this variety or we risk security exposures.
- We need to authenticate users in a reliable way. This means a robust protocol and a way to protect the results of authentication. Otherwise, users may skip authentication or illegally modify its results, exposing the system to security violations.
- There are tradeoffs between security and cost, more secure systems are usually more expensive.
- If authentication needs to be performed frequently, performance may become an issue.

Solution

Use a single point of access to receive the interactions of a subject with the system and apply a protocol to verify the identity of the subject. The protocol used may imply that the user inputs some known values or may be more elaborated. Figure 2 shows the class diagram for this pattern. A **Subject**, typically a user, requests access to system resources. The **Authenticator** receives this request and applies a protocol using some

Authentication Information. If the authentication is successful, the Authenticator creates a **Proof of Identity** (this can be explicit, e.g., a Token, or implicit).

Dynamics

Figure 3 shows the dynamics of the authentication process. A user requests access to the Authenticator. The Authenticator applies some authentication protocol, verifies the information presented by the user, and as a result a proof of identity is created. The user is returned a **Handle** for the Proof of identity.

Variants.

Single Sign-On

Single Sign-On (SSO) is a process whereby a subject verifies its identity and the results of this verification can be used across several domains and for a given amount of time. [Kin01]. The result of the authentication is the Authentication Token used to qualify all future accesses by the user.

PKI Authenticator

Public Key Cryptography is a common way to verify identity. This authentication can be described with a slight modification of the pattern in Figure 2 (Figure 4). An Authenticator class performs the authentication using a certificate that contains a public key from a Certificate Authority that is used to sign the certificate. The result of the authentication could be an Authentication Token used to qualify all future accesses by this user (in this case this is also a variant of SSO).

Known Uses

- Most commercial operating systems use passwords to authenticate their users.
- RADIUS provides a centralized authentication service for network and distributed systems [Gar 02, Has02].
- The SSL authentication protocol uses a PKI arrangement for authentication.
- SAML, a web services standard for security, defines one of its main uses as a way to implement a SSO architecture [sam].

Consequences

This pattern provides the following benefits:

- Depending on the protocol and the authentication information used, we can handle any types of users and we can authenticate them in diverse ways.
- Since the authentication information is separated, we can store it in a protected area, where all subjects may have at most read-only access.

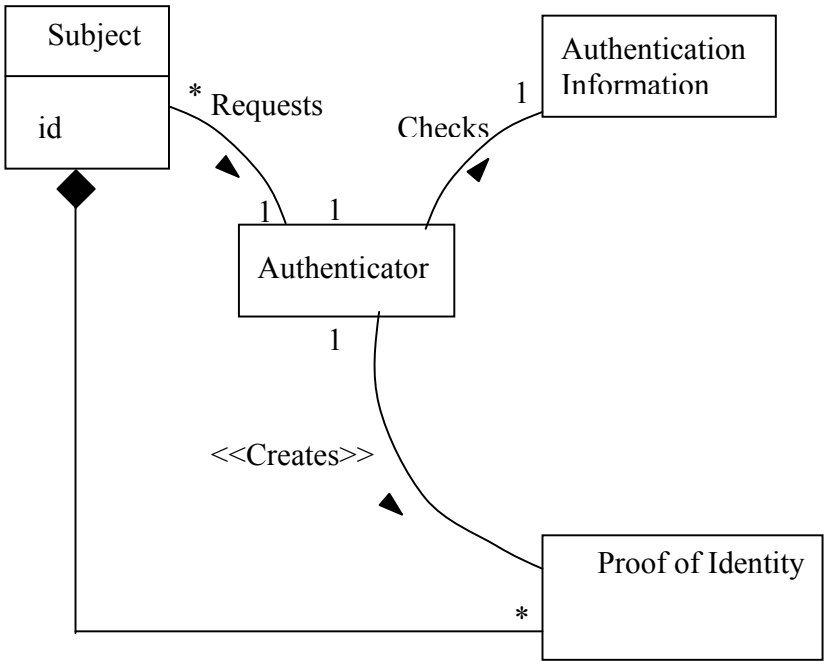

Figure 2 Authentication Pattern

- We can use a variety of algorithms and protocols of different strength for authentication. The selection depends on the security and cost tradeoffs. Three varieties include: something the user knows (passwords), something the user has (id cards), something the user is (biometrics), or where the user is (terminal, node).
- Authentication can be performed in centralized or distributed environments.
- We can produce a proof of identity to be used in lieu of further authentication. This improves performance.

Some liabilities are:

- The authentication process takes some time.
- The general complexity and cost of the system increase with the level of security.

Related patterns

The Distributed Authenticator [Bro99] discusses an approach to authentication in distributed systems.

The Distributed Filtering and Access Control framework includes authentication [Hay00].

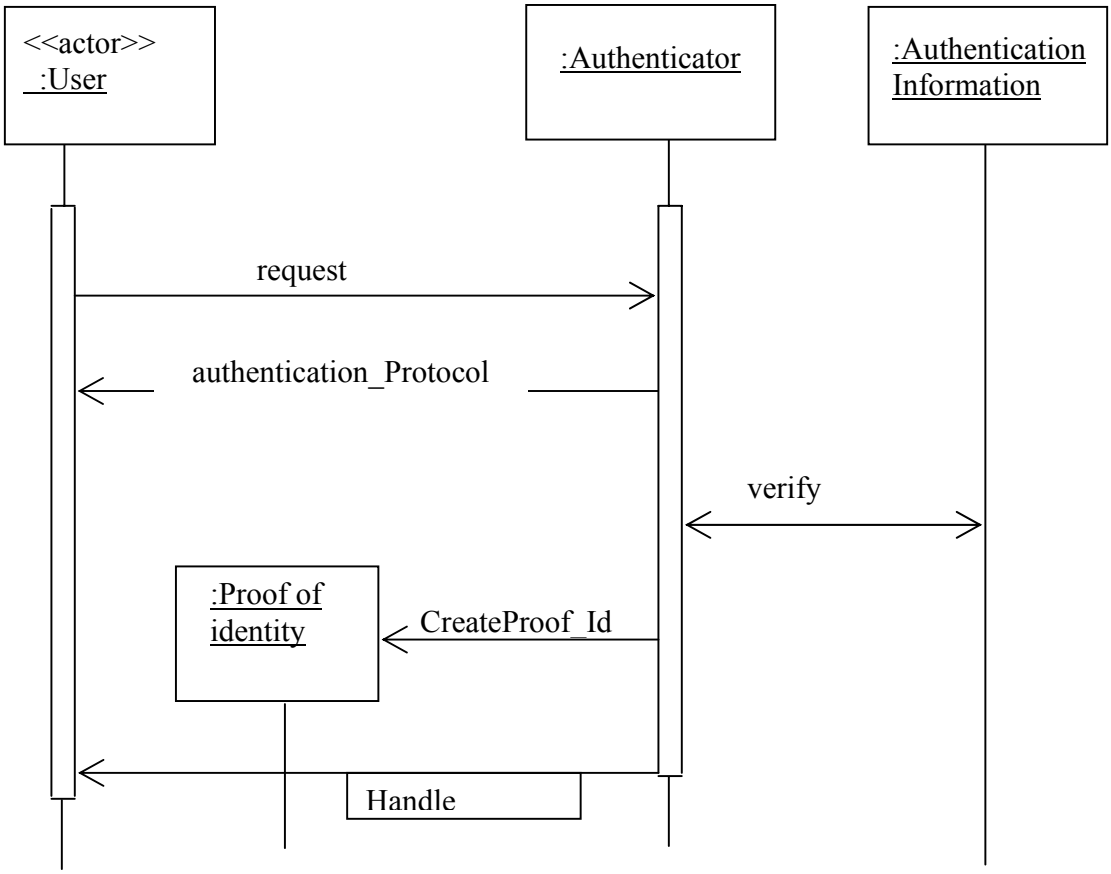

Figure 3 Authentication Dynamics

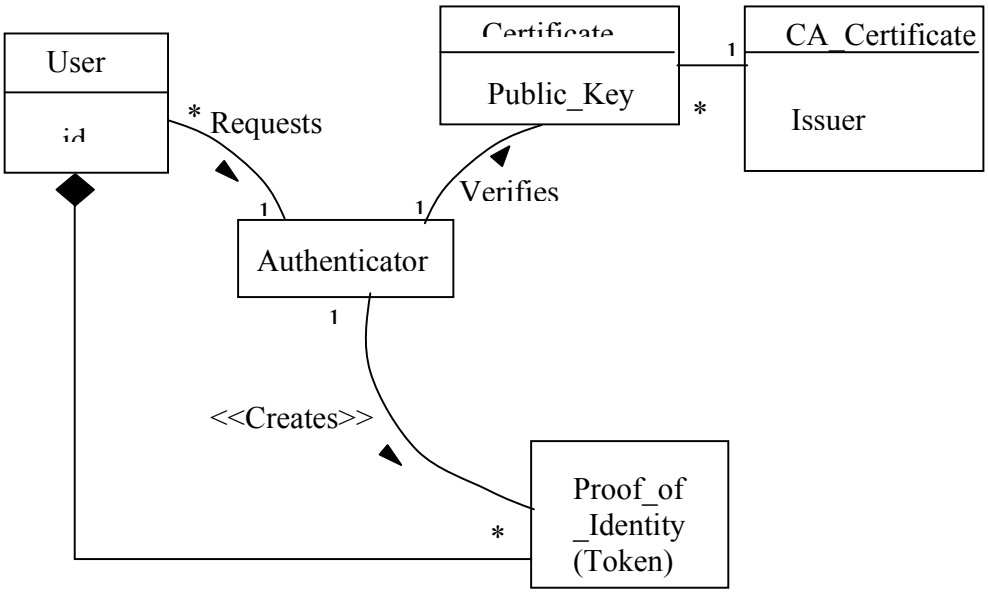

Figure 4. Class model for PKI authentication

Controlled-Process Creator

Intent

Define and grant appropriate access rights for a new process.

Context

An operating system where processes or threads need to be created according to application needs.

Problem

A computing system uses many processes or threads. Processes need to be created according to application needs and the operating system itself is composed of processes. If processes are not controlled they can interfere with each other and access data illegally. Their rights for resources should be carefully defined according to appropriate policies, e.g., need-to-know.

Forces

We need to apply the following forces:

- There should be a convenient way to select a policy to define process' rights. Defining rights without a policy brings contradictory and not systematic access restrictions, which can be easily circumvented.
- The child may need to impersonate its parent in specific actions, but this should be carefully controlled. Otherwise, a compromised child could leak information or destroy data.
- The number of children created by a process must be restricted or there could be denial-of-service attacks.
- There are situations where a process needs to act with more than its normal rights, e.g., to get data from a file to which it doesn't normally have access.

Solution

Since new processes are created through system calls or messages to the operating system, we have a chance to control the rights given to the new process. Typically, operating systems create a new process as a child process. There are several policies for granting rights to a child process. The child process can inherit all the rights or a subset of its parent's rights. Another policy is to let the parent assign a specific set of rights to its children (more secure because a more precise control of rights is possible).

Figure 5 shows the class diagram for this pattern. The **Controlled Process Creator** is a part of the operating system in charge of creating processes. The **Creation Request** contains the access rights that the parent defines for the created child. These access rights must be a subset of the parent's access rights.

Dynamics

Figure 6 shows the dynamics of process creation. A process will request the creation of a new process. The access rights passed in the creation request will be used to create the new access rights for the new process.

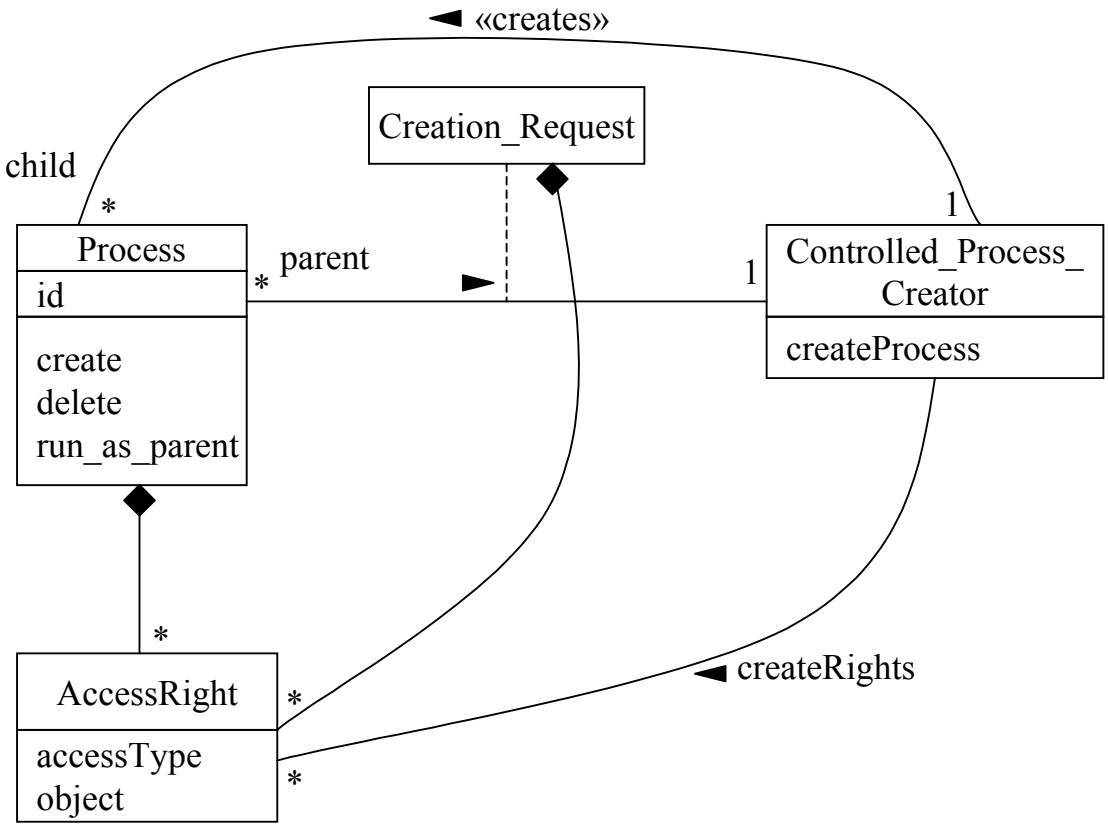

Figure 5. Class diagram of Controlled-Process Creator

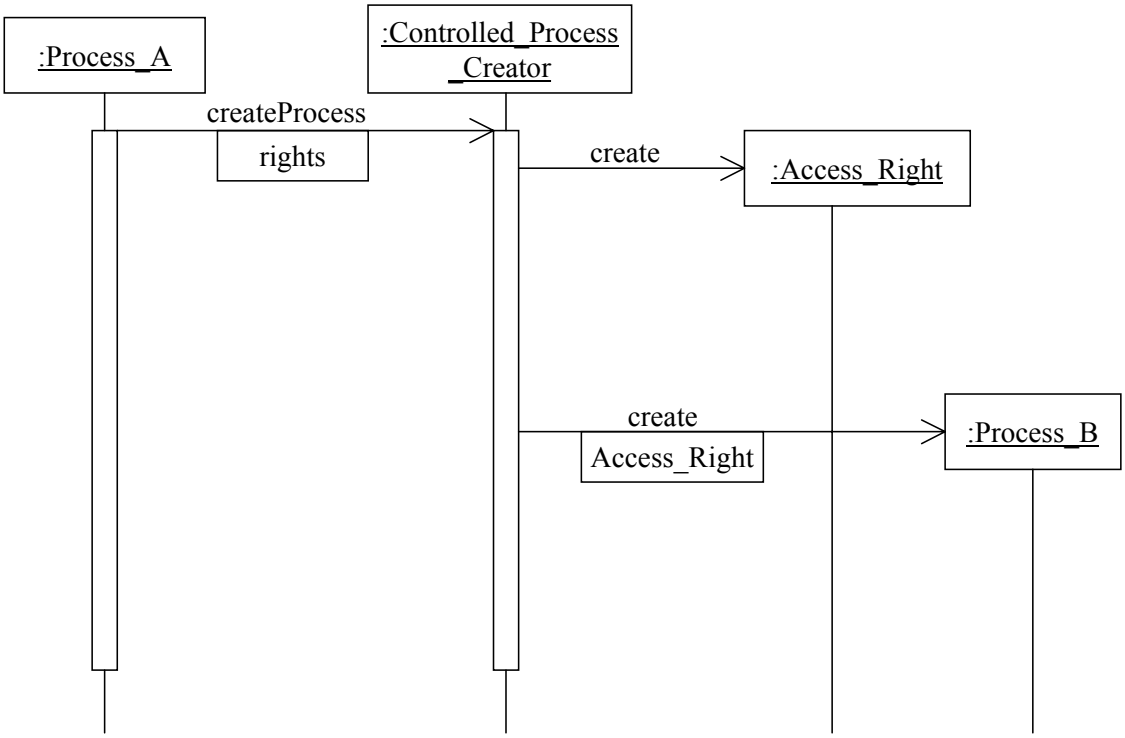

Figure 6. Process creation dynamics

Known Uses

In many operating systems, e.g., Unix, rights are inherited as a subset from the parent. Some hardened operating systems such as Hewlett Packard's Virtual Vault do not allow inheritance and a new set of rights must be defined for each child [HP].

Consequences

The advantages of this pattern are as follows:

- The created process can receive rights according to different security policies.
- The number of children produced by a process can be controlled. This is useful to control denial of service attacks.
- The rights may include the parent's id, allowing the child to run with the rights of its parent.

Controlled-Object Creator

Intent

Objects are created for specific purposes and the rights allowed to the other processes with respect to their access must be specified when they are created.

Context

A computing system that needs to control access to its created objects because of their different degrees of sensitivity.

Problem

In a computing environment, executing applications need to create objects for their work. Some objects are created at program initialization while others are created dynamically during execution. The access rights of processes with respect to objects must be defined when these processes are created or there may be opportunities for the processes to misuse them.

Forces

- Applications create objects of many different types but we need to handle them uniformly with respect to rights for their access. Otherwise, it would be difficult to apply standard security policies.
- We need to allow objects in a resource pool to be allocated and have their rights set dynamically. Not doing so would be too rigid.
- There may be specific policies that define who can access a new object, and we need to apply them when creating the rights for an object. This is a basic aspect of security.

Solution

When a **Process** creates a new object through a **Factory**, the **Creation_Request** includes the features of the new object. Among these features is a list of rights defining rights for a **Subject** to access the created **Object** (Figure 8).

Dynamics

Figure 9 shows the dynamics of object creation.

Consequences

The advantages of the above pattern are as follows:

- It is possible to define rights to use the object according to its sensitivity.
- Objects allocated from a resource pool can have rights dynamically attached.
- The operating system can apply ownership policies, e.g., the creator of an object may receive all possible rights for the objects it creates.

Known Uses

The Win32 API allows a process to create objects with various create system calls using a structure containing access control information (DACL) that is passed as a reference. When the object is created the access control information is associated with the object by the kernel. The kernel returns a handle to the caller to be used for access to the

object. Other operating systems apply predefined setups of rights; for example all the members of the owner's group in Unix may receive equal rights for a new file.

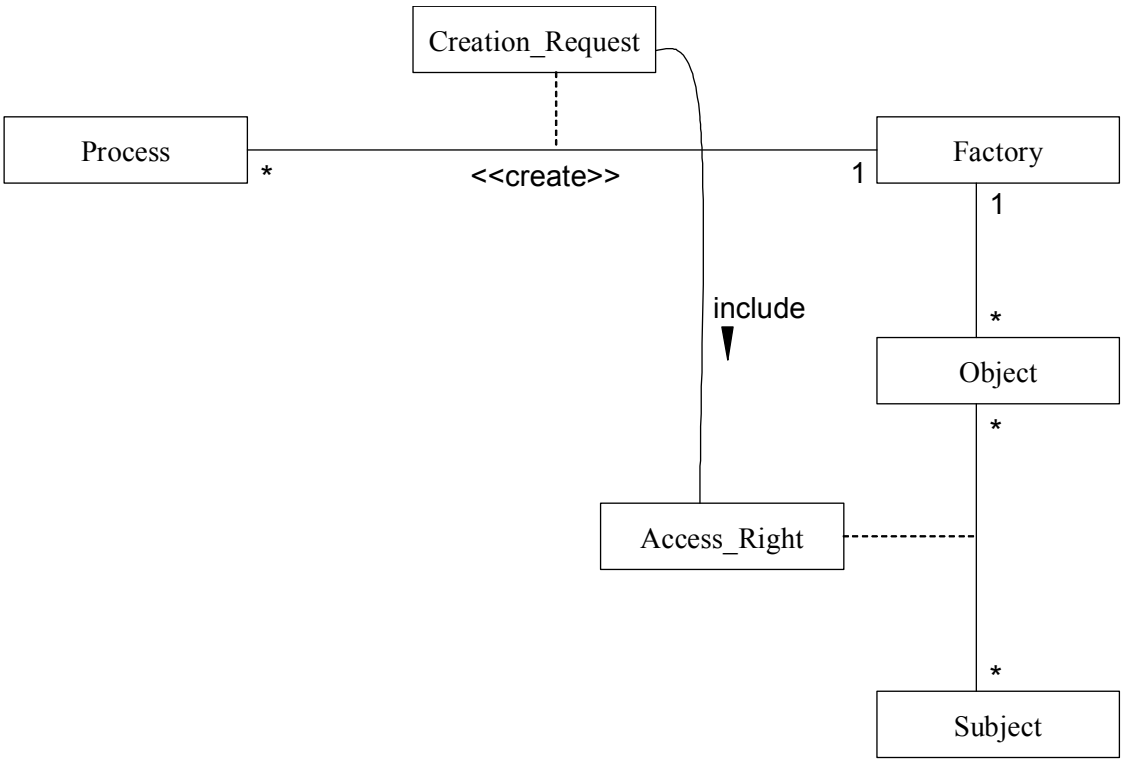

Figure 8. Class diagram of the Controlled Object Creator

Controlled-Object Monitor

Intent

How to control access by subjects to objects.

Context

An operating system consisting of objects that have controlled access.

Problem

When objects are created we define the rights of processes over them. These authorization rules or policies must be enforced when a process attempts to access an object.

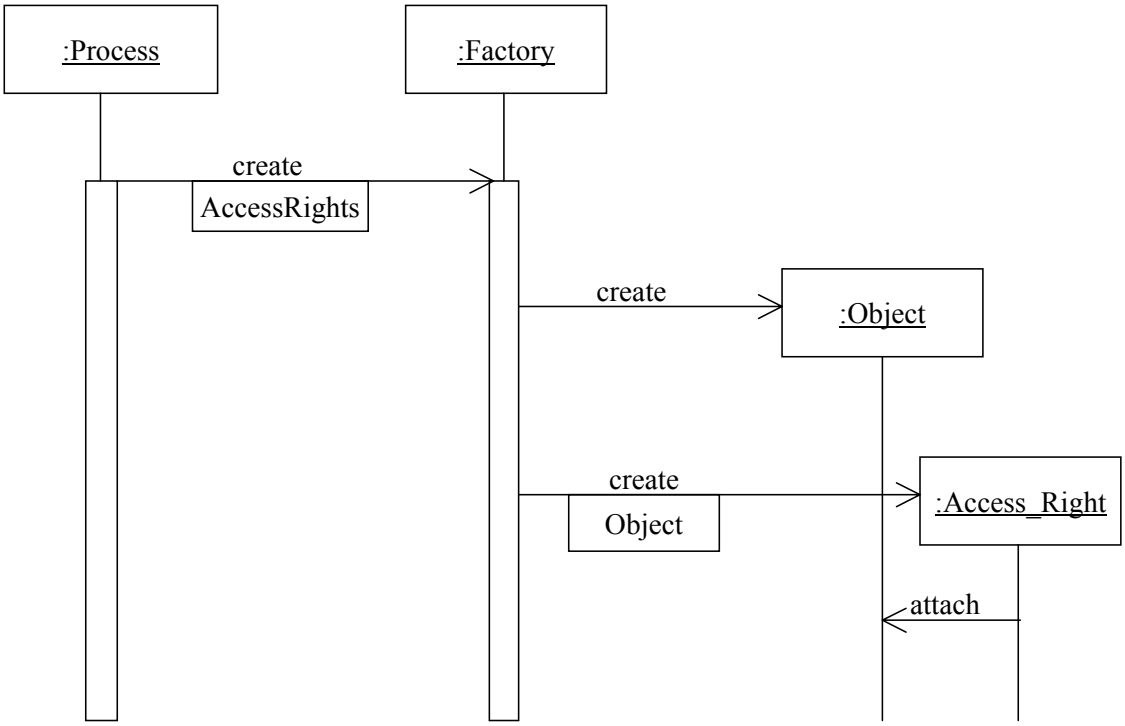

Figure 9. Object creation dynamics

Forces

The following forces apply:

- There may be many objects with different access restrictions defined by authorization rules; we need to enforce these restrictions when a process attempts to access an object.
- We need to control different types of access or the object may be misused.

Solution

Use a **Reference_Monitor** to intercept access requests from processes. The reference monitor checks if the process has the requested type of access to the object according to some **Access_Rule**.

Figure 10 shows the class diagram for this pattern. This is a more specific implementation of the Reference Monitor pattern of [Fer02]. The modification shows how the system associates the Rules to the secure object in question.

Dynamics

Figure 11 shows the dynamics of secure subject access to a secure object. Here the request is sent to the Reference Monitor where it checks the Access Rules. If the access is allowed, it is performed and result returned to the subject. Note that here, a Handle or ticket is returned to the Subject so that future access to the secure object can be directly performed without additional checking.

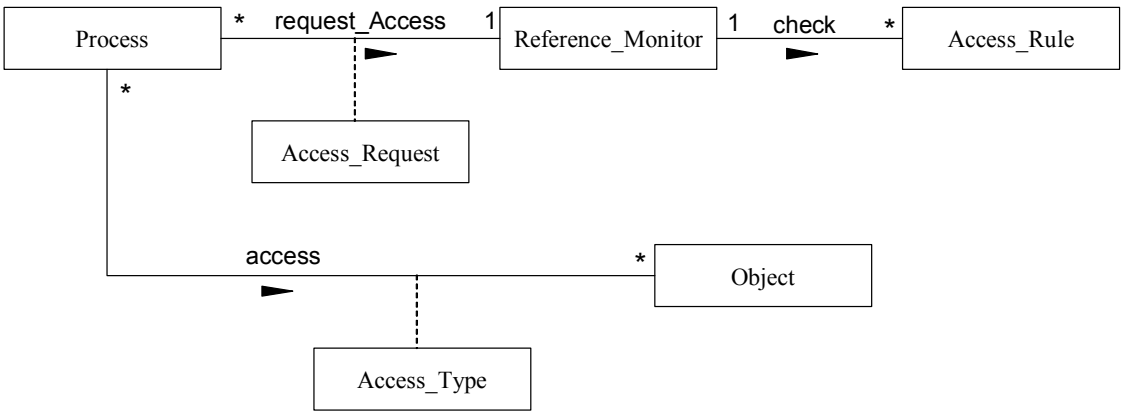

Figure 10. Class diagram of the Controlled Object Monitor

Consequences

The advantages of this pattern are as follows:

- The access rules can implement an access matrix defining different types of access for each subject.

- Each access request can be intercepted and accepted or rejected depending on the authorization rules.

The disadvantages are:

- Need to protect the authorization rules.
- There is an overhead in controlling each access.

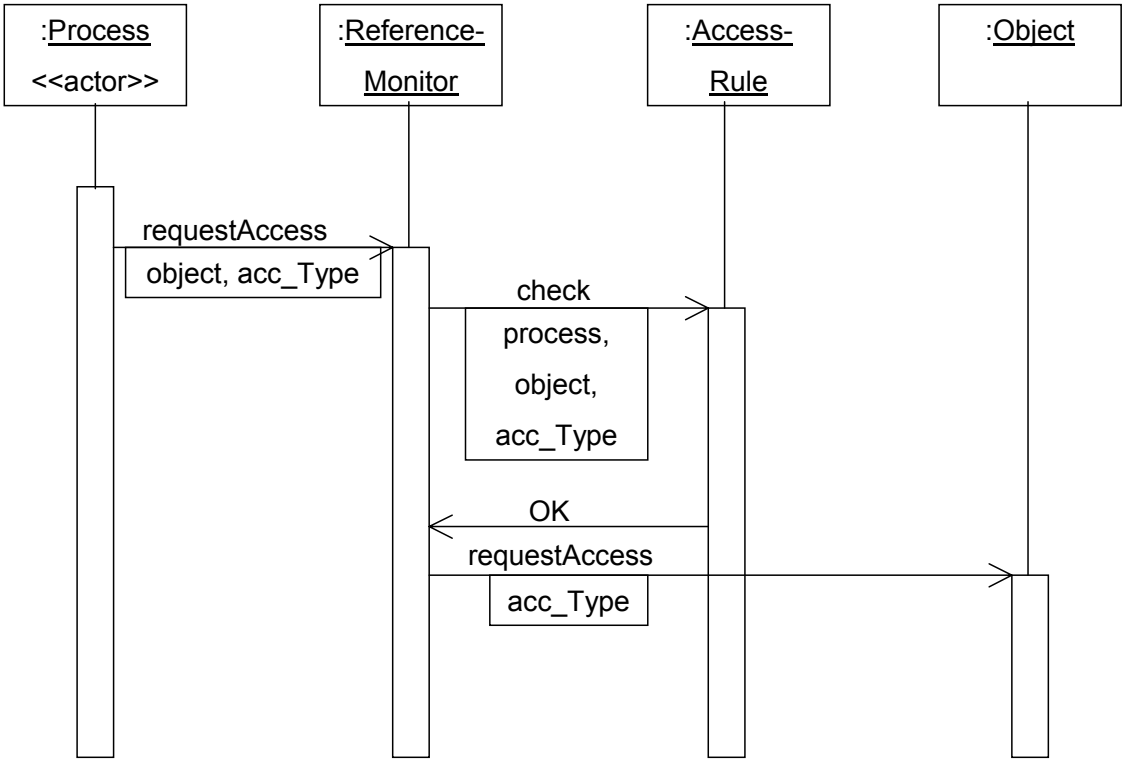

Figure 11. Sequence diagram for validating an access request

Implementation

A possible implementation would be: A user is authenticated when she logs on. Created objects inherit the original user's ID that is contained within a token. This token associates with the user process to be used by the Operating System to resolve access rights. Only those authorized may have the desired access to the secure object.

Each object that a user wishes to access may have an associated Access Control List (ACL). This will list what right each user has for the associated object. Each entry

specifies what right any other object within the system can have. In general, each right can be an "allow" or a "deny." These are also known as Access Control Entries (ACE) in the Windows environment [Har01, Mic00, Zac99]. The set of access rules is also known as the Access Control List (ACL) in Windows and most operating systems.

An alternative to the ACL are capabilities. A capability corresponds to a row in an access matrix. This is in contrast to the ACL, which is associated with the object. The capability indicates to the secure object that the subject does indeed have the right to perform the operation. The capability may carry some authentication features in order to show that the object can trust the provided capability information. A global table can contain rows that represent capabilities for each authenticated user [And01]. Or the capability may be implemented as a list corresponding to each user indicating what object the each user has access to [Kin01].

Uses of the combined patterns

The following examples use combinations of the above patterns:

Windows NT

The Windows NT security subsystem provides security using the patterns described here. It has the following three components [Har01, Kel97, Mic00]:

- Local Security Authority (LSA)
- Security Account Manager (SAM)
- Security Reference Monitor (SRM)

The Local Security Authority (LSA) and Security Account Manager (SAM) work together to authenticate the user and create the user's access token. The security reference monitor runs in kernel mode and is responsible for the enforcement of access validation. When an access to an object is requested, a comparison is made between the file's security descriptor and the SID information stored in the user's access token. The security descriptor is made up of Access Control Entries (ACE's) included in the object's Access Control List (ACL). When an object has an ACL the SRM checks each ACE in the ACL to determine if access is to be granted. After the SRM grants access to the object, further access checks are not needed since a handle to that object is returned the first time, which allows further access.

Types of object permissions are: No access, Read, Change, Full Control, and Special Access. For directory access, the following are added: List, Add, and Read.

Further, Windows utilizes the concept of a Handle for access to protected objects within the system. Each object has a Security Descriptor (SD) which contains a Discretionary Access Control List (DACL) of the object. Also, each process had a security token in addition which contains an SID (Secure ID) which identifies the process. This is used by the kernel to determine whether access is allowed. The ACL contains Access Control

Entries (ACE's) that indicate what access is allowed for a particular process SID. The kernel scans the ACL for the rights corresponding to the requested access.

A process requests access to the object when it asks for a handle using, for example, a call to CreateFile(). CreateFile() is used to create a new file or open an existing file. When the file is created a pointer to a SD is passed as a parameter. When an existing file is opened, the request parameters in addition to the file handle, contains the desired access, such as GENERIC_READ. If the process has the desired rights for the access, the request succeeds and an access handle is returned. Thus different handles to the same object may have different accesses [Har01]. Once the Handle is obtained, additional access to read a file will not require authorization to be done again. Also, the handle may be passed to another trusted function for further processing.

Java 1.2 Security

The Java security subsystem provides security using the patterns described here. The Java Access Controller builds access permissions based on permission and policy. It has a checkPermission method that determines the codesource object of each calling method and uses the current Policy Object to determine the permission objects associated with it. Note that the checkPermission method will traverse the call stack to determine access of all calling methods in the stack. The java.policy file is used by the Security Manager that contains the grant statements for each codesource.

Acknowledgements

We thank our shepherd Wolfgang Keller for his valuable suggestions that improved this paper considerably. The workshop participants at EuroPLoP 2003 also gave valuable comments.

References

[And01] R. Anderson, *Security Engineering*, Wiley 2001

[Bro99] F.L. Brown and E.B. Fernandez, "The Authenticator pattern", *Procs. of Pattern Languages of Programs Conf. (PLoP99)*, http://jerry.cs.uiuc.edu/~plop/plop99

[Fer99] E.B.Fernandez, "Coordination of security levels for Internet architectures", *Procs. 10th Intl. Workshop on Database and Expert Systems Applications, DEXA99*

[Fer01] E B. Fernandez and R.Y. Pan, "A pattern language for security models", *Procs. of PLoP 2001*, http://jerry.cs.uiuc.edu/~plop/plop2001/accepted_submissions/accepted-papers.html

[Fer02] E.B.Fernandez, "Patterns for operating systems access control", *Procs. of PLoP 2002*, http://jerry.cs.uiuc.edu/~plop/plop2002/proceedings.html

[Gar02] S. Garfinkel, *Web Security, Privacy & Commerce*, 2nd Edition O'Reilly

2002.

[Har01] J. M. Hart, *Win32 System Programming*, Second Edition, Addison Wesley 2001

[Has02] J. Hassell, *RADIUS*, O'Reilly, 2002.

[Hay00] V. Hays, M. Loutrel, and E.B.Fernandez, "The Object Filter and Access Control Framework", *Procs. of PLoP 2000,* http://jerry.cs.uiuc.edu/~plop//plop2k/proceedings/proceedings.html

[HP] Hewlett Packard Corp., Virtual Vault, http://www.hp.com/security/products/virtualvault

[Kel97] M. Kelley, "*Windows NT Network Security, A Manager's Guide,"* Lawrence Livermore National Laboratory, 1997.

[Kin01] C. King, et.al., *Security Architecture*, Osborne McGraw Hill 2001.

[Lan99] C. R. Landau, "Security in a Secure Capability-Based System," *Operating Systems Review,* October 1999.

[Mic00] Microsoft, *Windows 2000 Security, Technical Reference*, 2000

[sam] SAML, http://www.saml.org and http://www.oasis-open.org/committees/security/

[Sch00] D. Schmidt, et.al. "Pattern-Oriented Software Architecture," Wiley 2000.

[Sil03] A. Silberschatz, P. Galvin, G. Gagne, *Operating System Concepts (6th Ed.),* John Wiley & Sons, 2003.

[Vis99] P. Viscarola, "*Windows NT Device Driver Development*," Macmillan Technical Publishing, 1999.

[Zac99] W. H. Zack, *Windows 2000 and Mainframe Integration*, Macmillan Technical Publishing 1999.