On Paradigms for Security Policies in Multipolicy Environments\textsuperscript{1}

Winfried E. Kühnhauser
German National Research Centre For Computer Science (GMD)
Schloss Birlinghoven
D-53754 Sankt Augustin, Germany
email: winfried.kuehnhauser@gmd.de

Abstract

For many years the traditional concept of the reference monitor has proven to be a sound architectural foundation for secure computer systems. However, with the advent of distributed systems and multiple, user-defined application-specific security policies the limitations of reference monitors become more and more obvious.

Recently, two concepts have been proposed that aim at supporting user-defined security policies in a distributed multipolicy environment. This paper is a comparative study of these concepts. It discusses their major conceptual differences with respect to supporting discretionary and mandatory security policies, policy encapsulation and separation, multiple security policy and metapolicy support and the amount of support required of an underlying policy neutral reference monitor.

Keywords: Security, Privacy, Integrity, Separation, Multiple Application - Dependent Security Policies, Metapolicies, Access Control Programs, Custodians.

1 Introduction

In the last decade, the ever increasing amount of software application systems has confronted us with higher and more individual security requirements. Security policies have been developed that support the needs of individual applications, with the Bell/LaPadula model (providing confidentiality) being only the first example in an ever-increasing multitude [BL76, Bib77, CW87, BN89, San92]. In the next decade, the integration of application dependent security policies in a distributed computer system will become a major challenge in computer security.

Several security policies in modern application systems address a wide variety of aspects of system security, such as access control, authentication, availability, and auditing. This variety requires concepts in the underlying system platform

\textsuperscript{1}Proceedings of the 11th International Information Security Conference (IFIP/SEC '95), Cape Town, South Africa 1995. Chapman & Hall.
that go beyond traditional access control mechanisms. Although the traditional architectural concept of the reference monitor as defined in the Trusted Computer System Evaluation Criteria (TCSEC) [Dep83] has been very successful within its intended scope, its role in supporting a wide range of application-specific policies is limited.

As a consequence, new paradigms for supporting user-defined security policies in a distributed multipolicy environment are currently emerging. This paper is a comparative study of two new paradigms: the custodian paradigm [Küh94, KvKO95] and advanced access control programs, a paradigm based on the idea of access control programs presented in [TNT92]. Both paradigms deal with a fundamental difficulty in current secure computer systems: the integration of application-dependent security policies in a distributed computer system requires a computing base whose very basic properties include the concurrent support of more than a single built-in security policy. This especially requires an evolution of traditional architectural principles like reference monitors, trusted computing base, and security kernels as defined in the TCSEC. However, the basic reference monitor properties tamperproofness, complete access mediation and verifiability are successful principles that should not be easily dismissed. While both paradigms thus support the dynamic integration of user-defined security policies, they also provide the reference monitor properties of tamperproofness, total communication control and verifiability for user-defined security policies.

While custodians have been developed with the goal of supporting multiple security policies, the original access control program (ACP) concept aimed at fine-grained delegation control in distributed systems. However, both concepts have in common that unlike traditional access control mechanisms they use algorithms for controlling security that provide much more flexibility in the types of security policies that can be implemented.

Algorithm-based security policies are found in many areas. In hospital information systems, several access decisions depend on the notion of time: nurses usually have different rights for accessing patient record during day and night hours. In a finance consulting company or a bank, the separation-of-duty policy is often applied before issuing a report or transferring large values. An example in [TNT92] describes a scenario where students in a university course are given access to a sample solution of their homework only after they have submitted their own solution. This is an example for access decisions depending on the state of other objects within a system. All these small example policies are difficult to implement using current, non-algorithmic access control mechanisms.

While the comparison of both concepts will generally result in a deeper understanding of the problem of policy integration in a multipolicy environment, contrasting custodians and access control programs is also promising because of the affinity of custodians to access control lists (ACLs) on the one hand and the affinity of ACPs to capabilities on the other. ACLs and capabilities are representations of a global access control matrix [Lam74]. An access control matrix can be viewed either from the point of view of an accessing subject (the rows of the matrix) or from the point
of view of an accessed object (the columns of the matrix). Capabilities implement
the subject view [CJ75, MT86, WN85]. ACLs implement the object view [Org85].
While formally both matrix representations are equivalent, the implementations of
both concepts nevertheless have quite different properties with respect to delegation
or revocation.

The paper shows that there is in much the same way a similar duality between
their active counterparts, ACPs and custodians. Our goal is to clarify this duality
and compare both the expressive power of both concepts and to discuss the maj-
or conceptual differences with respect to supporting discretionary and mandatory
security policies, policy encapsulation and separation, multiple security policy and
metapolicy support, and the amount of support required of an underlying policy
neutral reference monitor.

We apply our ideas to distributed systems consisting of connected, autonomous
and mutually suspicious computer systems and we do not assume any central man-
agement. We also assume that within such a system several independent applica-
tion systems coexist. Application systems consist of interacting entities, and any
entity interaction is mediated by local and autonomous reference monitors. The
individual security requirements of these applications are reflected in applica-
tion-specific security policies associated with the applications. Whenever applica-
tions with different security policies interact, metapolicies coordinate the involved se-
curity policies [Hos92, KvKO95].

The TCSEC glossary defines the term security policy as a "set of laws, rules,
and practices that regulate how an organization manages, protects, and distributes
sensitive information". In the following sections we will use a more technical inter-
pretation and use the term "security policy" to denote a software component that
implements and enforces these laws, rules, and practices within a computer system.

2 Access Control Programs

Access control programs were introduced by Marvin Theimer et. al. in [TNT92]. In
a distributed environment with a client/server - based cooperation scheme, ACPs
permit a fine-grained delegation of access rights to intermediates that act on behalf
of a client. The concept is based on the observation that existing delegation proto-
cols have only very limited possibilities to restrict delegated rights to precisely the
minimum the intermediate needs, which in general results in too much rights given
to an intermediate.

ACPs are programs that a client passes together with a request to a server. An
ACP precisely describes the rights a client is willing to delegate to an intermediate
for each single request. A digital signature prevents intermediates from tampering
with it. As a part of the permission check within the server, the server executes the
ACP and grants the intermediate's access only if the ACP approves.

\footnote{This view corresponds to the "multiple interconnected systems view" in the Trusted Network Interpretation (TNI) [Nat87] of the TCSEC}
Theimers paper focuses on delegation, ACP transmission and protection, ACP execution and on the ACP language. Nevertheless, the concept of ACPs has another very interesting facet. It withdraws from the traditional scheme of passive, database descriptions of access rights to algorithm-based access decisions, and thus is well in tune with the increasing need for ambitious security models and also with the direction the current standardization work is pointing to, both stressing the need for paradigms and mechanisms that provide the formal power of a Turing machine. Based on Theimers original idea of shipping algorithms for access control, we will in this paper discuss the paradigm of advanced access control programs (aACPs) that constitute a container for a complete security policy.

In this new view, every entity in a system that is submitted to a security policy is glued to a corresponding aACP. Whenever the entity communicates, the aACP is shipped together with the communication message to the target object where, then, the security policy within the aACP is executed to check the conformance of the entity’s operation with the security policy. If more than a single entity is submitted to the same security policy (the regular case), all these entities are glued to the same aACP. All entities submitted in this way to one security policy constitute the security policy’s domain.

While discretionary policies may allow the owner of an entity to select or modify the associated aACP, mandatory policies are enforced by aACP encapsulation and mandatory entity association. For enforcing encapsulation and the association to application entities, the aACP paradigm requires the support of the reference monitor of the underlying system platform. AACPs thus transport the reference monitor properties tamperproofness and total access mediation to user-defined security policies.

The following example illustrates that aACPs provide significant progress already in existing systems. The Kerberos system [SNS88, MNS88] administers client rights within the Kerberos ticket server. Requesting a file service requires first to get a ticket from the ticket server and then presenting the ticket to the file server, together with a proof of authentication. While currently all protection in Kerberos is based on the authentication of a client, allowing the ticket server to include aACPs within

---

\[3\]we will use the notion of aACP to distinguish between the extended view discussed in this paper and Theimer’s original ACP semantics
Kerberos tickets open the door for several algorithm-based authorization policies. As an example, as an alternative to the standard Kerberos authorization scheme, an electronic cash management application can apply an authorization policy that protects the anonymity of a client. Additionally, all of the examples in the introduction can be implemented with a ACPs in Kerberos.

To illustrate the aACP paradigm, we will discuss a simple multilevel security policy consisting of a set of application entities, a labelling of these entities and a rule defining read access rights based on the labelling.

For defining syntax and semantic of the policy, we use a small algebraic model consisting of types, operations, and axioms. The types in our example are the set \( \mathcal{E} \) of application entities, the set \( \text{BOOL} = \{ true, false \} \), and the set \( \mathcal{N} \) of natural numbers. The operations are a less-or-equal function returning \( true \) if the first argument is less than or equal to the second, a function returning the \( label \) of an entity, and the \( \text{ReadCheck} \) function with its semantics defined by the axiom.

\[
\begin{align*}
\text{types} & \quad \mathcal{E}, \mathcal{N}, \text{BOOL} \\
\text{operations} & \quad \leq : \quad \mathcal{N} \times \mathcal{N} \to \text{BOOL} \\
& \quad \text{label} : \quad \mathcal{E} \to \mathcal{N} \\
& \quad \text{ReadCheck} : \quad \mathcal{E} \times \mathcal{E} \to \text{BOOL} \\
\text{axioms} & \quad \text{ReadCheck}(s, o) \quad == \quad \leq (\text{label}(s), \text{label}(o))
\end{align*}
\]

The transformation of the algebra into an access control program is straightforward: the aACP contains a persistent representation of the label operation (an array indexed by entity IDs) and an implementation of the \( \text{ReadCheck} \) operation by means of an exported procedure.

Submitting an application to our small multilevel policy then consists of associating the aACP with all subjects of the application. The association guarantees that whenever a subject communicates, the associated aACP is shipped together with the request and executed by the target object by means of calling the exported \( \text{ReadCheck} \) operation (see figure 2).

Note that in this simple example a single common aACP for all application entities is sufficient. This is due to the centralized nature of our mandatory MLS policy. Influences from the environment like performance or availability requirements may cause several physical copies of the aACP.

A prototype implementation of aACPs has been done in the Distributed Computing Environment (DCE) [Fou92] of the Open Software Foundation [Lux95]. An aACP is implemented as a DCE server; the DCE server model provides persistency for the \( label \) information (by persistent files) and encapsulation for protecting aACP privacy and integrity. The following C++ class is an implementation of our above
Subject Request Object

reference monitor support: policy encapsulation, access mediation and binding

Figure 2: An MLS Access Control Program Associated with an Application Entity

MLS algebra as a DCE aACP. The class is an abstraction from the real aACP class: we omitted the constructor and destructor functions, and the 128-bit DCE entity identifiers are collapsed to natural numbers (to allow for simple indexing of the label data base).

class mlsACP: public DCEMetaACP
{
public:
    ReadCheck(int s, int o)
    { return (label(s) <= label(o)); }

private:
    int labelDB[1000];

    label(int e)
    { return (labelDB[e]); }
};

All definitions and operations that refer to embedding an aACP class instance into the DCE environment are inherited from and implemented by the superclass DCEMetaACP. DCEMetaACP provides the IDL interface, the name server enrollment, the DCE remote procedure call communication scheme and shared access synchronization. Further details of the DCE implementation are discussed in [Lux95].

The association of an aACP to its subject (here: a DCE client) is achieved by a modification of the remote procedure call implementation within the DCE
runtime system. The runtime system contains a reference to the associated aACP class instance and ships the instance with every outgoing remote procedure call. Shipping an aACP instance to a server can be done in three ways: by value, by reference or by name.

Passing an aACP by value includes a copy of the instance in the rpc request, while reference and name passing ships only a pointer. A reference is a 128 bit unique DCE identifier, while a name is a symbolic DCE name. On the implementation level, the difference between reference and name passing is small. However, passing by name requires the assistance of the name servers for executing an aACP, and thus causes additional lookup overhead and encloses the name servers in the trusted computing base.

3 Custodians

Custodians were first suggested 1993 in the BirliX Security Architecture (BSA) [HKK93] and discussed in detail in [Küh95]. The role of custodians within a framework for supporting multiple security policies is discussed in [KvKO95].

The custodian paradigm is a concept to support multiple, application-dependent and user-defined security policies for distributed applications. Similar to an aACP, a custodian is a shell for a security policy that constitutes an annex to a policy neutral reference monitor. While maintaining the reference monitor properties tamperproofness, complete access mediation and verifiability, custodians prepare the reference monitor to assimilate user-defined security policies. Custodians are glued to arbitrary system entities, hereby submitting the entities to the custodian’s security policy. No semantical restriction is imposed on the contained security policy. A custodian may emulate a simple traditional access control list or contain a sophisticated rule based security policy.

The custodian paradigm is based on a tamperproof reference monitor providing total communication control and separation. Additionally, the reference monitor provides the glue between application and security policy. The glue is a binding mechanism that associates a custodian to an arbitrary collection of application objects. Any such association results in a detour of any object communication to its associated custodian. Depending on the security policy, the communication can be suppressed, modified, or forwarded to the original recipient. The binding mechanism thus expands the total communication control property of the reference monitor to include custodians.

With respect to the first and third reference monitor properties (tamperproofness and verifiability), custodians make use of the object model of the underlying system platform to provide policy privacy, separation, persistency, and communication. The custodian paradigm does not dictate a separate object model. Instead, the object model is imported from the underlying system platform that itself has its roots in the separation and communication schemes of the underlying reference monitor. Thus the precise separation, persistency and communication models are
defined "lazily" when the paradigm is brought into a concrete environment. While this may result in restrictions caused by the shortcomings of the object model of a specific environment, we gain an easy integration of the concept into a wide variety of platforms. On every platform, custodians are ordinary system entities containing the security policy for their associated objects. Within the BirliX Security Architecture, custodians are ordinary user-defined abstract data types [HKK93]. On a Unix platform, custodians are ordinary program files (with limited data persistency), and in the OSF Distributed Computing Environment custodians are servers or access control list managers.

![Diagram of custodians guarding two communicating objects.](image)

**Figure 3:** Custodians Guarding Two Communicating Objects

To illustrate the custodian paradigm, we again use our small multilevel policy. In the example, we again confine ourselves to a single custodian instance for the entire domain of the MLS policy. Several impacts from the environment like performance and availability requirements, mutual distrust between the subject’s and the object’s site, or different security policies for the subject and object can cause different custodian instances (of the same class or of different classes) for the subject and object [Küh95] (see figure 4).

![Diagram of a single MLS custodian guarding application entities.](image)

**Figure 4:** A Single MLS Custodian Guarding Application Entities

Like a ACPs, custodians have been implemented in the DCE, and the DCE object model provides persistency (by persistent files), privacy and integrity [Lux95].
The transformation of the algebra into the program body of a custodian is nearly the same as in the case of aACPs. The only difference is that the embedding of a custodian class into the DCE environment is inherited from a different superclass \textit{DCE.MetaCustodian}. Even within the superclass, the differences to the aACP implementation are small. They deal with request-oriented policy selection vs. entity-oriented policy selection and with the additional cryptographic mechanisms needed for aACP sealing during shipment (see section 4.1).

\section{Blood Brothers}

This section is a comparative study of the custodian and aACPs paradigms with respect to their support of discretionary and mandatory security policies, policy encapsulation and separation, multiple security policies and metapolicies, and the amount of support required of an underlying policy neutral reference monitor.

Both paradigms have several properties in common. They both constitute a container for user-defined security policies and provide traditional reference monitor properties. They both safeguard policy privacy and integrity, although some of the enforcing mechanisms are different. Last not least, both paradigms submit an application to a security policy by associating an aACP respectively a custodian to every application entity.

The fundamental difference between the custodian and the aACP paradigm is that

- in the aACP paradigm, a common security policy floats from the subject to the object for every subject/object communication
- in the custodian paradigm, security policies are bound to application entities, and different policies can be bound to the subject and the object.

\subsection{Encapsulation, Integrity, Privacy}

The goal of separation is rigorous isolation to gain integrity or confidentiality or other properties such as self-protection \cite{AHJ92}. In a multipolicy environment, separation between the individual security policies is obligatory to safeguard policy privacy, integrity, and verifiability.

To allow for a wide variety of discretionary and mandatory security policies, a policy must itself be protectable from access by arbitrary system entities. This principle can already be observed in the \textit{reference monitor concept} defined by the TCSEC. The TCSEC defines a reference monitor to be a component of the trusted computing base that enforces the access control policy, and the integrity of the reference monitor is the first of three major design requirements. While the TCSEC does not explicitly mention it, the \textit{privacy} of the security policy itself is similarly vital: several security policies maintain state information (e.g. the labels of subjects
and objects in a multilevel policy, company conflicts in the Chinese Wall policy [BN89, San92], or keys for issuing certificates in a X.509 certification authority [CC188] that itself is confidential information.

However, while mandatory security policies require a strict enforcement of the separation between an object and its controlling security policy, several discretionary security policies allow for a weaker implementation that sacrifices security in favor of performance. Two examples are the ticket checkers of Kerberos servers and the OSF DCE access control managers that are both directly linked to a server and run within the same address space [SNS88, Fou92]. While providing the better performance of procedure calls instead of remote procedure calls to the security-checking code, a malicious server may easily circumvent the security check. Thus this approach works only with trustworthy servers and includes the servers in the TCB.

So the point is that a general paradigm for supporting a wide variety of user-defined security policies must safeguard both, policy privacy and integrity, and to that end, must provide encapsulation.

Custodians as well as aACPs provide a policy encapsulation concept. Although in Theimer’s original ACP paper, a shipped ACP is executed within an execution environment provided by the target object, separation of an ACP and object can easily be enforced by a separate execution environment provided by the reference monitor of the object’s site. However, while the abstract concept of policy encapsulation is honored by both paradigms, the mechanisms to enforce encapsulation are quite different. While custodians are ordinary system entities using the encapsulation mechanisms of the underlying system platform that themselves have their roots in the separation mechanisms implemented within the underlying reference monitor, an ACP additionally is shipped via the network to the target object of a communication operation. Policy privacy and integrity thus has to be enforced by additional cryptographic operations that require a common agreement on the encryption/decryption functions as well as a common key management.

4.2 Metapolicies

The multiple interconnected systems view of the TNI covers large and complex computer networks connecting a multitude of independent research, education, governmental and commercial organizations. Cooperation within such a network generally requires a definition of the relationships between the involved security policies, a concept often referred to as metapolicies [Hos92].

Metapolicies are policies about policies. They create a framework for complex coexistence of several security policies, containing rules for interfacing, cooperation, and conflict resolution [KvKO95]. Such a framework basically needs two principles: policy separation (see 4.1) and policy communication [MS88].

Policy conflicts generally occur at the perimeters of policy domains, either when two policy domains overlap or when entities belonging to two different domains interact.
As an example, consider an estate agency operating a multilevel policy on its document handling systems. The same agency uses the services of a finance consulting company. The consultant company is British, so it is legally obliged to submit its own document handling systems to the Chinese Wall policy. A market analyst from the consulting company thus (as a subject) is submitted to Chinese Wall, while any document of the estate agency (as an object) is submitted to MLS.

Whenever an analyst from the Chinese Wall domain accesses a document within the MLS domain of the estate agency, this access will not run smoothly. Neither is the subject within the MLS domain nor is the object within the Chinese Wall domain. In this situation, a metapolicy kicks in and handles policy conflicts and cooperation by a set of rules that have been agreed upon by both the estate agency and the finance consultants. Rules for policy cooperation and conflict handling in metapolicies are maintained in cooperation and conflict matrices as introduced in [KvKO95].
a subject submitted to Chinese Wall accesses an object submitted to MLS, the metapolicy is consulted and cooperates in the access decision.

An aACP implementation of the same scenario reveals a major drawback of aACPs. As aACPS provide only a single access authority in a subject/object interaction, both policies have to be merged into a single aACP. Because neither organisation trusts the other, the merging has to be done by a trusted third party, the metapolicy. Even then,

- the subject site must trust the object site to execute the subject’s part of the sandwich policy correctly
- without additional cryptography, the object site must trust the subject site not to tamper with the object’s part of the sandwich policy.

The consequence is that because security policies are performed only at the object’s site, the aACP paradigm is incapable of implementing policies that deal with shaked trustworthiness of either site of an interaction. Examples for corresponding problems are mentioned in the Trusted Network Interpretation: the cascading problem and the propagation-of-risk problem.

With respect to the propagation-of-risk problem, the TNI states that in many cases operational needs have led to the accreditation of systems for multilevel operation that actually do not meet the requirements of the recommended class. While the increased risk may be accepted by the local users of a particular system, connection of such a system to a network propagates the additional risk to all users in the network. Systems that do not accept this risk run policies that restrict communication to accredited systems only, and these policies must be enforced on the trusted local system.

As an example of the cascading problem, consider a same small distributed system of two hosts, each of which is accredited to handle only two levels of information, while the system as a whole handles three levels. When both hosts are connected so that medium level information may be exchanged, it is sufficient for any intruder to defeat the protection mechanisms of the individual systems (designed to maintain only two levels) to be able to downgrade information over three levels. Again, systems that do not accept this risk run policies that restrict communication to systems accredited for the same levels only, and these policies must be enforced on the trusted local system.

Concluding, because in the aACP paradigm only the subject provides the security policy, the object’s site must trust the subject’s site to provide a correct common policy. Because a security policy is executed and enforced only at the object’s site, the subject’s site must trust the object’s site to correctly enforce the policy. Such mutual trust is not essential in the custodian paradigm that thus is more powerful in general environments with mutually distrusting systems.
5 Conclusions

The paper is a comparison of two recent paradigms for supporting user-defined application-specific security policies in distributed multipolicy environments.

On the conceptual level, custodians and aACPs have several common properties. Both paradigms constitute a container for user-defined security policies, provide policy encapsulation as a basic concept for safeguarding policy privacy and integrity, persistency for maintaining long-term policy states, and policy communication as a fundamental requirement for metapolicy interaction. The major conceptual difference is that while in a subject/object interaction custodians guard both parties at their site, an aACP floats from the subject to the object and is executed at the object site only. Without additional ways to protect their integrity, aACPs thus assume trust by the object that the subject provides a correct policy as well as trust by the subject that the object site correctly enforces the policy. AACPs thus are less suited for multiple interconnected and mutually distrusting systems.

With respect to the mechanisms required to support the paradigm, both paradigms commonly require the same persistency, communication, application association and encapsulation mechanisms. However, while custodians use only the corresponding mechanisms of an underlying reference monitor, aACPs additionally require cryptographical sealing for their shipment as well as a common key management between the subject and the object site.

With respect to performance, the conceptual superiority of custodians in distributed systems with mutual distrust must be paid for by an additional custodian invocation in every subject/object communication. However, this invocation requires an expensive mechanism only when a security policy must be enforced without the servers being part of the trusted computing base. In more trustworthy environments, ways have been shown for mitigating communication costs as well as the encryption costs for shipping aACPs.

6 Acknowledgments

The author would like to thank the systems security group at GMD, including Ciaran Bryce, Michael von Kopp Ostrowski, Wolfgang Lux and Mathias Reinholz, for innumerable valuable discussions.

References


