1 Introduction

This chapter motivates why CORBA security in general — and the management of application–level access policies for CORBA objects in particular — deserve dedicated research. Section 1.1 briefly introduces CORBA and typical properties of CORBA environments. Section 1.2 states the problem addressed by this thesis, and section 1.3 outlines the contributions made in this work. Section 1.4 gives an overview of the contents of the thesis.

1.1 Middleware and Security

Distributed systems relying on middleware such as Java/RMI [Sun Microsystems, 1997] and the Object Management Group's (OMG) CORBA [OMG, 1999a], [Brose et al., 2001b] specification have seen widespread adoption. Both the Java platform and CORBA middleware are used in the industrial development of large–scale applications (e.g., [Koch and Murer, 1999]), and on a wide range of computing environments, including smart cards, PDAs [MICO, 1998], embedded systems [OMG, 1998a], standard PCs and workstations, high–performance application servers, and even parallel computers [Keahey and Gannon, 1997].

CORBA provides a unified data model that hides one main source of complexity in distributed systems, viz. the heterogeneity caused by diverse hardware platforms, operating systems, programming languages, and communication mechanisms. In particular, the abstraction of a location–transparent CORBA object hides all the details of accessing potentially remote data and functionality behind a well–defined interface that is declared in CORBA's Interface Definition Language (IDL). This interface language itself is entirely independent of implementation languages. Standardized IDL language mappings define how to generate code that represents IDL constructs in programming languages such as C, C++, Java, COBOL, LISP, Ada, Smalltalk, and Python. Interoperability between clients and servers written in different languages and running on different CORBA implementations on different operating systems is guaranteed through the use of a standardized protocol for request and response messages and a canonical transfer syntax. CORBA implementations from different vendors running on different operating systems thus allow programs written in different programming languages to interoperate by making object invocations.

1.1.1 Security and Complexity in Distributed Systems

> *[...] we all know that unmastered complexity is at the root of the misery.* [Dijkstra, 2001]

One of the main sources of security problems in distributed systems is their inherent complexity. While CORBA hides heterogeneity behind standardized interfaces, many other sources of complexity in distributed systems remain. In fact, these sources of complexity are often inherent in the motivation for using distributed systems in the first place.

One of the prime reasons for employing distributed rather than centralized systems is the potential for *combining resources* to build systems that are more powerful than any single centralized system [Tanenbaum, 1995]. The problem of scale, i.e., the sheer number of users, transactions, objects, and the potentially large number of messages exchanged between communicating entities is thus an inherent property of many distributed systems and contributes to their complexity. Another reason why distributed systems are used is the increased *availability of resources* that may be achieved if single points of failure are avoided and redundancy is provided in the design of a system. If one computing node in a system goes down, others may still be available. However, redundancy in the system design requires care to keep configurations consistent. Further, resources are often made remotely accessible to enable *sharing* by multiple applications. Sharing supports more efficient use of expensive resources, but it also necessitates coordination and protection efforts to eliminate unwanted interference between applications. Finally, many applications are not only logically but *physically distributed* because different parts of the applications must reside at different locations, for example, flight reservation and booking systems that can be accessed from travel agencies, airport offices, and perhaps even over the Internet.

An immediate consequence of all these uses of distributed systems is that multiple different nodes and resources have to be installed and managed. Due to the number and complexity of management tasks within distributed systems, different management tasks are usually separated from each other and performed by different people. This might be done simply to share the workload or because of logical or physical separations between subsystems, such as between different branches of an organization or because some tasks may require different skills than others. As a result, management becomes a cooperative task carried out by a number of different people, at potentially different locations, times, and levels.

Finally, complexity lurks not only in the computing environment and its management, but also in the development and deployment of applications that make adequate use of these environments. In general, these kinds of applications are concurrent if not parallel, and rely on communication. Not only is it technically challenging to develop these kinds of applications according to their specification, but development projects themselves are likely to be complex because the size of the task requires large development teams. Moreover, the technical skills and expertise required by this kind of development often motivate the hiring of external developers which are not part of the organisation that eventually uses the software, so in general deployment and management will not be carried out by the original developers of the software.

To summarize these issues, the kind of environments addressed here exhibit the following general characteristics, all of which contribute to their overall complexity:

- Scale — systems comprise large numbers of entities (sites, users, machines, transactions, network messages, application components),
- Redundancy — several instances of a functional component may exist in a system for load balancing or availability purposes,
- Sharing — multiple entities may access the same resource,
- Distributed management, development and deployment — tasks are carried out by different parties, at different times and locations.

The complexity inherent in distributed systems constitutes a major problem for overall security. Complex systems are hard to understand, build, verify, and manage: "Almost all security failures are in fact due to implementation and management errors" [Anderson, 1994]. The chances for human error in the stages of developing and using these systems are significant, and some of these errors are particularly critical for security: incomplete or inconsistent specifications that do not correctly state security requirements, vulnerabilities due to exploitable implementation errors or even undetected backdoors, management errors that lead to insecure system states, etc. Any comprehensive approach to securing distributed systems must take factors like these into account.

Abstracting from individual platforms by allocating security functions at the middleware level and not at the level of individual operating systems is an important first step to reduce the complexity incurred by heterogeneity. The CORBA Security Service takes this approach and provides mechanisms to address security requirements at the level of abstract CORBA objects, which hides much of the heterogeneity and provides a single, homogeneous layer. For the remainder of this thesis, it is assumed that security functionality is allocated at this level to enable building and operating non–trivial applications in a heterogeneous distributed system. The primary focus of this work are thus security policies for application objects that are expressed in terms of the CORBA object model.

1.1.2 Access Control

The security requirements that are predominant in CORBA environments are integrity requirements that can be addressed by employing access control. These are sometimes also referred to as "commercial" security policies [Clark and Wilson, 1987]. This is not to say that commercial environments do not also have confidentiality requirements, but for most non–military organizations (perhaps excluding organized crime) the damage that can be caused by leakage of information generally does not pose an existential threat, even if it may lead to severe competitive disadvantages. Loss of integrity of critical data, however, may put a company out of business altogether. For the purpose of this thesis, the discussion is therefore narrowed further

to access control. Other classical security requirements, such as confidentiality or availability of resources, are beyond the scope of this work when not enforced by access control.

Access control is concerned with preventing unauthorized accesses to shared resources. For any access to an object the caller's authorizations need to be checked before the access can be allowed. The mechanism that makes access decisions is called *access decision function* (ADF) [ISO/IEC, 1996a, ISO/IEC, 1996b]. This function takes *access control information* as an argument, which describes the initiator, the attempted access itself, and its context. This data is compared with an *access control policy*. In the most general sense, a security policy is a set of rules or constraints that defines those states of a given system that meet its security requirements. An access control policy describes which accesses in a system are authorized and which are not. The notion of the term policy that is used in this thesis implies that this representation is in terms of a formal access model and can be directly evaluated by an access control mechanism.[1] A more formal definition will be given in chapter 4.

The security requirements that are enforced by access control policies include the integrity or confidentiality of data. Access control policies can also be used to enforce general resource usage restrictions, e.g., to ensure availability or even coordination policies in CSCW systems [Edwards, 1996]. As an example, consider a set of product design documents that are critical assets in a company. A potential security requirement for these documents is to keep them private from competitors or protecting them from malicious modification by external saboteurs or even corrupt or disgruntled staff. Commonly, security requirements like these are initially specified in natural language at an abstract level and later refined and "implemented" by policies in terms of a particular access control model.

1.2 Problem Statement

> *The worst problem with access control policy, especially in object systems, is that there's so much of it.* [Blakley, 2000, p. 45]

Addressing security at the level of CORBA objects does not positively affect the complexity dimensions of scale and of division of labor. The number of objects that need to be protected and the number of accesses to these objects must be expected to be high, so *scalable* approaches are required. Also, it must generally be assumed that development, deployment, and management will be carried out by independent or only loosely–coupled entities, each of which may be further structured internally. In order to be *manageable* in such an environment, a security infrastructure must support means for modularization of tasks, e.g., hierarchical structures that support modeling of refinement. Finally, because of the wide–ranging applications of CORBA, the security infrastructure must be *flexible* in order not to restrict the general applicability of the middleware.

This thesis focuses on policies rather than mechanisms. A well–established set of security mechanisms is available for implementations of security services for CORBA, so the need

[1] This corresponds to the notion of an *Automated Security Policy* (ASP) in the terminology of [Sterne, 1991].

for new mechanisms is small. With regard to access control, for example, the allocation of a reference–monitor–like [Department of Defense, 1985] protection mechanism in CORBA is technically straightforward because every invocation of a remote object is mediated by the middleware, so accesses to objects can always be intercepted. The OMG's *Security Service Specification* [OMG, 2001b] defines a component called *access control interceptor* which is responsible for access control enforcement in CORBA.

While the complete mediation property of middleware makes interception conceptually simple, it also means that the middleware must now be trusted, which significantly extends the trusted computing base (TCB) [Department of Defense, 1985] and thus the effort required to reach some level of assurance. At the time of this writing, the number of interoperable security products for CORBA is still very small and high–assurance CORBA security mechanisms are virtually non–existent. None of the existing products is certified according to security criteria such as TCSEC [Department of Defense, 1985] or CC [Common Criteria, 1999]. How and how much assurance for complex middleware can be achieved is an important problem that requires more research, but these issues are beyond the scope of this work.

A basic premise of this thesis is that the correct design, specification and management of security policies is a central problem in distributed systems because these tasks are both error–prone and by their very nature security–critical. This observation corresponds directly to the design principle of psychological acceptability in [Saltzer and Schroeder, 1975]: "[...] to the extent that the user's mental image of his protection goals matches the mechanisms he must use, mistakes will be minimized. If he must translate his image of his protection needs into a radically different specification language, he will make errors."

The potential damage caused by inadequate handling of policies is significant and there are currently no standard methods or tools that provide appropriate support to application designers and security administrators. Moreover, it is not sufficient to support these roles in isolation. Because of the potentially large number of objects and complex interaction patterns in object–oriented applications, a security administrator cannot be expected to completely understand the logic of all applications running under his supervision. Thus, the security implications of, e.g., introducing a new right or object type into a running system might be unclear, so managers need security documentation at a suitable level of abstraction. This information can only be provided by application designers.

The problem that this thesis addresses is the following: Given these general characteristics of CORBA environments, how can the specification, deployment and management of application–oriented access control policies be supported in a way that increases overall security? In particular, how can the task of writing, deploying and managing access control policies be divided between application developers, deployers and managers of distributed object systems such that indeed only legitimate accesses will be allowed? What are the right abstractions and tools that help to increase the involved parties' understanding of their task and decrease the probability of error? What kind of security architecture is required to provide these abstractions and tools?

The main focus of this work is on security policies for application objects because policies

for application objects are typically both more interesting and more complex than those for system objects because application objects tend to be more dynamic and fine–grained, and because these objects exhibit shorter lifetimes and more complex interactions. It is expected, however, that any approach that adequately supports managing the security of application objects will also support managing system objects, i.e., system resources with CORBA interfaces.

1.3 Contributions

This dissertation takes a classical software engineering approach to the problem of specifying and managing access control policies. In essence, a formal language called VPL (*view policy language*) is devised, which allows designers and administrators to deal with abstractions that are adequate for their respective tasks. We claim that this approach reaps the general benefits of language support like documentation, structuring, support for static analysis, specification reuse and enhanced communication. It thus significantly reduces the potential for vulnerabilities introduced by human error and increases the overall system security.

The main contribution of this thesis is the new access control model underlying the specification language. The model is based on a new authorization concept called *view* and combines role–based and matrix–based access control concepts. It is shown how this model addresses the requirements of distributed object systems, and that it is better suited to express advanced policy concepts than the default access model specified in the OMG's *Security Service Specification* [OMG, 2001b]. A comprehensive, fully implemented case study is presented that shows the applicability of this model to realistic applications.

Another contribution of this thesis is the presentation of a general architecture of the security infrastructure required for this approach. Its feasibility is demonstrated by a Java implementation of an access control mechanism that uses the view–based access model for representing policies. Moreover, prototype implementations of the required management infrastructure are provided, in particular a VPL compiler, a role server, a policy server, and a management service for object domains.

1.4 Thesis Overview

This thesis is structured es follows:

- Chapter 2 introduces terminology and identifies the requirements for a manageable security architecture for distributed object systems.
- Chapter 3 gives an overview of the standard CORBA Security Framework and evaluates its access control concepts with respect to the management requirements established in chapter 2.

- Chapter 4 introduces a policy language and a new, view–based access control model that is designed to meet the requirements identified in chapter 2.
- Chapter 5 formalizes the access model presented in chapter 4.
- Chapter 6 presents an application case study that demonstrates the use of the policy language.
- Chapter 7 describes the architecture and implementation of a security infrastructure that supports the view–based access control model.
- Chapter 8 discusses related work.
- Chapter 9 summarizes this thesis and discusses options for future research.