Hidden Capabilities: Towards a Flexible Protection Utility for the Internet

D. Hagimont, J. Mossière. X. Rousset de Pina
Sirac project, Imag-Inria,
Inria Rhône-Alpes, 655, avenue de l'Europe, F-38330 Montbonnot Saint Martin
Internet: Daniel.Hagimont@imag.fr

1 Introduction

This paper presents a protection model based on software capabilities. Its main advantage is that capabilities are hidden from the application programmer, allowing the protection policy of an application to be defined independently from the application code. This is very interesting for protecting an already developed application.

Our claim is that this protection model is well suited for a wide range of environments, from clusters of tightly coupled servers to large loosely coupled servers on the Internet. It is particularly interesting when applications are built by assembling many existing pieces of code into a single application.

We argue our claim by describing our model and its integration in three different environments: a DSM (Distributed Shared Memory)system [Pérez 95], CORBA (Common Request Broker Architecture) [OMG 91] and the Java programming environment [Java 95].

The rest of the paper is structured as follows. In section 2, we provide an overview of our protection model based on hidden software capabilities. Section 3 discusses the integration of this model in a DSM system, an ORB and the Java environment. We conclude in section 4.

2 Protection Model

2.1 Motivations

Protection is a crucial aspect of distributed computing, in particular when untrusted users co-operate using shared objects. Several protection models have been proposed [Organick 72, Wulf 74, Kowalski 90, Cabrera 92, Hagimont 94], among these are capability-based models [Levy 84]. Capabilities are very convenient because of their flexibility, allowing the implementation of various execution and protection policies.

However, in all the proposed approaches (e.g. [Tanenbaum 86, Chase 94]), capabilities are made available at the programming language level through capability variables that are used explicitly for:

- accessing objects (mapping an object to a capability)
- changing protection domains (with a capability on the called domain's entry point)
- transferring access rights between protection domains (passing a capability as a parameter).

Therefore, in a protected application, the definition of protection is wired in the application code, which necessitates a different implementation for each distinct set of access rights associated with its execution. Moreover, protecting an already developed application or reusing an unprotected module implies at least a partial re-implementation.

We claim instead that capabilities should be invisible to the programmer, in the sense that it should be possible to define the protection policy of an application independently from its code. This allows a single piece of a code to be executed with different protection policies, which ensures both economy and separation of concerns. We have proposed a new protection model [Hagimont 96] which is based on capabilities that are hidden from the application programmer, and which allows the protection policy of an application to be defined only in terms of the application's interface.

2.2 Overall Design

The protection model is based on the concepts of capability, protection domain (a set of capabilities which defines the current protection environment of a process) and cross-domain capability (which allows controlling protection domain changes).

As explained in 2.1, the main idea is to hide capabilities from the programmer when they are used for access rights control, protection domain crossing and access rights transfer. It is implemented by managing capabilities in a separate protected layer (the operating system kernel or a run-time environment), and by providing administrators with an interface with which the management of access rights can be expressed. This results in a system in which application code does not depend on protection; protection-related actions (access rights checking, domain-crossing and rights transfers) are only triggered when the application traps into the protected layer.

2.3 Managing access to protected objects

Protected objects are supposed to be shared and persistent. Very often in such a case, the first time an application tries to access an object, it traps into the **object manager** which is supposed to bind the object to the language level reference (an oid or a pointer) used to access the object.

Thus, this object manager is the protected layer mentionned above. The application being executed in one protection domain, the object manager can check if a capability (for accessing the object) exists in the current protection domain. If this is the case, the object manager completes the binding and resumes the execution. If not, the object manager delivers a protection violation exception to the application.

Therefore, the application code only includes language references and does not have to include capability operations.

2.4 Managing protection domain crossings

We want to allow protection domain crossing by implicit use of a cross-domain capability. Therefore, in our model, a protection domain crossing capability is associated with an object and is used only when the object cannot be accessed locally.

When an application in one protection domain traps in the object manager (for an object binding) and if the protection domain doesn't contain a capability for the target operation, but contains a cross-domain capability (associated with the target operation), then the same access will trigger a domain crossing, allowing the operation to be executed in the remote domain (specified by the capability).

2.5 Managing capability transfers

One of the main problems is then to allow capability parameters to be exchanged between protection domains while keeping code independent from protection. From now on, we assume that an operation on an object has a procedural form.

Our proposal is to express capability transfers using an Interface Definition Language (IDL). The key feature we are using here is that an interface is defined separately from the code of the application. Just like an interface is associated with an entry point in a remote server, an interface can be associated with a domain capability which is an entry point in the target protection domain. This interface would actually exist anyway if protection was implemented by a client-server system.

Therefore, our proposal is to specify capability transfer rules in an interface definition associated with each domain capability. The IDL is extended to specify protection rules that define the capabilities which must accompany object references transmitted as parameters. Note that this specification is associated with a domain capability and is completely separated from the application code.

3 Application

In order to defend our claim that this protection model is well suited for a wide range of environments, we discuss its integration into three environments, from tighly to lightly coupled: a DSM system, CORBA and Java.

3.1 Application to a DSM system

This protection model has first been implemented in a Distributed Shared Memory system in which a single address space is managed [Pérez 95][Hagimont 96].

Applications running in protection domains only know about virtual addresses. Objects (called segments in this context) are dynamically mapped in the address spaces of user level processes. At the first use of a segment in a process (i.e. at the first access to a virtual address in the segment), an addressing error event is delivered to the local kernel. In response to this event, a capability for the segment is searched for in the current domain. If the search succeeds, the segment is mapped and the execution is restarted. The segment can be either a data segment or a code segment.

Cross-domain calls are associated with procedure calls. In the case of a code segment, if the domain does not contain a capability for mapping the segment, but contains a domain capability associated with the segment, then a cross domain call is forced. We implement this call in the same way as an RPC: a stub is mapped instead of the code segment. This stub passes the effective parameters to the kernel which implements the RPC. The kernel is responsible for transfering capabilities together with pointer parameters according to the IDL specification associated with the domain capability.

3.2 Application to CORBA

In the framework of our project, we are also considering the application of hidden capabilities to CORBA. More precisely, we are currently designing an implementation of our protection model in an Orbix[1] environment [Iona 94] on PC-Intel/Windows based machines.

In Orbix, applications are composed of servers that manage C++ objects. A server can invoke a method on an object from a remote server by means of proxies. A proxy is an object that gives the illusion that the target object is local but, when invoked, implements an RPC to the remote server, thus ensuring the so-called distribution transparency. Proxies make use of stubs generated from IDL specifications associated with class definitions.

In this environment, we only check protection for cross-server invocations. A server executes in a protection domain in which capabilities define the remote objects that can be called. A capability is associated with an object and specifies the access rights in terms of authorized methods. When a cross-server invocation occurs, the executed

[1] Orbix is a Registered Trademark of Iona Technologies Ltd.

stub can check whether a capability exists for the call. This stub is also responsible for transfering capabilities to the remote server according to the protected IDL specification.

Two implementations are considered. The first one is to manage capabilities in a server which has to be consulted at each invocation, with the advantage to allow capability revocation. The second is to manage capabilities directly in the servers address spaces; in that case, the use of capabilities must be restricted, for example with password or encrypted capabilities. This choice is a trade-off between fonctionality and efficiency. For the integration in Orbix, we use the ability to define filters that are some kind of stub specialization.

3.3 Application to Java

In this section, we claim that hidden capabilities could also be used for the protection of large Internet applications such as those developed in the Java environment [Java 95].

Java is an object-oriented language and environment which allows the development of portable applications. The key idea is that any piece of code written with the Java language can be run on any machine on which the Java virtual machine has been installed. Thus, this is a means for building "dynamic" applications that can load (at run time) Java classes available on ftp or Web servers on the Internet. In such an environment, one of the key issues is the protection of the local file system from Java applications while allowing these applications to use local files if needed.

In the following, we provide some technical hints related to our file access control implementation in Java. A complete description is available in [Hagimont 96].

Java allows the management of some software protection domains that are used in order to isolate local classes from remote classes. We manage a separate protection domain associated with each remotely loaded class. This is implemented using the ability to manage our own class loader: a different class loader is associated with each protection domain.

In Java, files are accessible via a FILE class which is a system class. The Java SECURITYMANAGER class provides security checks with the use of the system classes. Thus, this is a means to implement selective access control to file objects. The *checkRead()* and *checkWrite()* methods are exported by the SECURITYMANAGER class and can be overridden in order to set these security checks. From these checking methods, the current class loader is authenticated by the system, therefore authenticating the current protection domain.

When a remote class needs to be loaded, we create a new instance of our class loader (another implementation of the same abstract class). This class loader instance is used for loading the class and it corresponds to a new protection domain. As explained just above, the checking primitives of the security manager will be able to get the reference of this class loader in order to implement access control. In order to transfer capabilities to the new protection domain we install a stub class "in front of" the loaded class. This stub class implements the same interface as the class, except that, when the IDL specification indicates that a file capability must be passed along with a parameter, the stub adds the capability in the capability list associated with the loaded class.

4 Conclusion

In conclusion, we believe that hidden capabilities are well suited for many application areas. They are currently used as the base mechanism for protecting applications on a DSM system. We are experimenting with hidden capabilities on an ORB and exploring their use on top of the Java virtual machine. This latter investigation shows a high potential for protection in very large scale applications.

Bibliography

[Cabrera 92] L.-F. Cabrera, A. W. Luniewski, J. W. Stamos. Fine-Grained Access Control in a Transactional Object-Oriented System, *Computing Systems*, 5(3), Summer 1992.

[Chase 94] J. S. Chase, H. M. Levy, M. J. Feeley, E. D. Lazowska. Sharing and Protection in a Single-Address-Space Operating System, *ACM Transactions on Computer Systems,* 12(4), pp. 271-307, November 1994.

[Hagimont 94] D. Hagimont. Protection in the Guide Object-Oriented Distributed System*, Proc. of the 8th European Conference on Object-Oriented Programming*, pp. 280-298 , July 1994.

[Hagimont 96] D. Hagimont, J. Mossière, X. Rousset de Pina, F. Saunier. Hidden Software Capabilities*, 16th International Conference on Distributed Computing Systems* (to appear) , May 1996.

[Iona 94] Iona Technologies. Orbix distributed object technology - programmer's guide, Version 1.2, February 1994.

[Java 95] The Java Language Environment: A White Paper, http://java.sun.com/whitePaper/java-whitepaper-1.html, Sun Microsystems Inc., 1995.

[Kowalski 90] O. C. Kowalski, H. Härtig. Protection in the BirliX Operating System, *Proc. of the 10th International Conference on Distributed Computing Systems,* pp. 160-166, May 1990.

[Levy 84] H. M. Levy. *Capability-Based Computer Systems*, Digital Press, 1984.

[OMG 91] OMG. The Common Object Request Broker: Architecture and Specification, OMG Document Number 91.12.1, Revision 1.1, December 1991.

[Organick 72] E.I. Organick. *The Multics System: an Examination of its Structure,* MIT Press, 1972.

[Pérez 95] E. Pérez-Cortés, P. Dechamboux, J. Han, "Generic Support for Consistency in Arias", *5th International Workshop on Hot Topics in Operating Systems (HOTOS-V),* pp. 113-118, Rosario, Washington, May 1995.

[Hagimont 96]. D. Hagimont, S. Krakowiak, J. Mossière and X. Rousset de Pina. *A Selective Protection Scheme for the Java Environment*, Technical Report, RT-Sirac-96-12, http://sirac.inrialpes.fr

[Tanenbaum 86] A. S. Tanenbaum, S. J. Mullender, R. Van Renesse. Using Sparse Capabilities in a Distributed Operating System, *Proc. of the Sixth IEEE International Conference on Distributed Computing Systems*, pp. 558-563, 1986.

[Wulf 74] W.A. Wulf, E. Cohen, W. Corwin, A. Jones, R. Levin, C. Pierson, F. Pollack. Hydra: The Kernel of a Multiprocessor Operating System, *Communications of the ACM,* 17(6), June 1974.