Security requirements for cooperative work: a model and its system implications

George Coulouris and Jean Dollimore
Department of Computer Science
Queen Mary and Westfield College
University of London
Mile End Road London E1 4NS
Email contact: George.Coulouris@dcs.qmw.ac.uk

(Work undertaken while the authors were on sabbatical at:
Bull-IMAG Systèmes, Grenoble, France)

Applications involving cooperation between several users are often expressed in terms of operations on shared objects. The challenge is to provide adequate access to shared objects for cooperating principals while maintaining the required level of integrity and privacy for objects. We assume an open system environment populated with distributed and potentially shareable and persistent objects. Security mechanisms should be independent of application programs, ensuring the security of data objects regardless of the programs that are used to access them. In general, we need the ability to control access at the level of each of the operations of an object.

We present a security model that reflects the structure of cooperative work, enabling users' security policies and other task requirements to be translated directly into access rights for those shared objects whose protection must be guaranteed for the successful outcome of cooperative tasks. Our model is derived from a study of some cooperative tasks in the real world [5]. The model is based on notions of group tasks, organisational roles and delegation. We consider briefly the implications of our security model for operating system design in the context of a shared object system based on a distributed shared memory model [1].

Elements of the model

A *principal* is an authority to which access rights can be attributed. Principals may be simple or compound. Simple principals are identifiers for active entities (e.g. *Frère Jacques*, *the chair of the examining board*, or *Corporate Finance Officer*). Compound principals are composed from simple ones using a few simple composition operations.

The notion of compound principal is due to Lampson *et al.* [3] who have defined several composition operations including role playing, delegation and conjunction. They have developed a logic that enables the authenticity of compound principals to be tested, and have shown how their logic may be used to construct an open distributed system in which the credentials for any operation can be checked [2]. Like them, we intend our model to be used in conjunction with an authentication system that constructs checkable credentials for compound principals of arbitrary complexity, with a minimum of *a priori* trusted components.

Our motivation is to produce a system in which the organisational authority for actions by users can be inferred and checked from the credentials that they hold. We have found the model of Lampson *et al.* an excellent starting point for the construction of a security model for cooperative work.

Our security model structures cooperative work in terms of *tasks* and *roles*. Tasks are work activities in which users perform actions on objects, with well-defined goals. The production of a directory of courses in a university, the preparation of a question paper for an examination, the joint preparation of a corporate financial report are examples of tasks.

Access to objects can be restricted to just those tasks for which they are needed. A task is a simple principal associated with a set of roles, and can be used in conjunction with one of those roles to obtain access rights to the objects associated with the task. For example, access to the versions of an examination paper would be restricted to principals holding credentials for roles in the task of preparing the paper.

Roles correspond to the administrative roles found in organisations. They are introduced because there is a need to restrict the rights of the individual users who hold credentials for a task. For example, all of the users with credentials for roles in the examination paper task may read the paper, but only two of the roles involved in the task may update it. A *role* is a simple principal that can be adopted (played) by one or more users. As in the DEC work, roles are associated with groups of principals and membership certificates are created and signed by an appropriate authority to certify the right of a user to adopt a role. Membership certificates are also used to certify the roles that are allowed to participate in each task.

In cooperative work most roles are played by a single user - for example *chair of the examination committee*. Some roles can be played by several users - for example, *member of the examination committee*. Roles may play a part in several tasks; for example, the chair of examiners in a university is involved in the tasks of preparing several examinations and the evaluation of their results.

We associate access rights for objects with roles and with the tasks in which there is a need to access the objects. In many types of cooperative work there is a requirement for a 'need to know' policy, which we support at two levels: a user adopting a role restricts her rights to just those of that role; and a role joining a task restricts the rights of the role to just those available in the task. We add an operation **in** to the authentication logic of DEC to denote participation in a task. User A has the rights associated with R **in** T if A is in role R and is participating in task T.

Each object has an access control list (ACL) that specifies the rights of principals to access its operations. The ACLs must contain entries that can be interpreted as giving rights that correspond to R **in** T for the various roles and tasks in which the object needs to be accessed.

In general, the ACL should contain an entry for each possible R **in** T that needs access to the object, but the complexity of ACLs can be substantially reduced by the following method in which role and task rights are separately specified. If task T has access rights A_T and role R has access rights A_R to some object, then role R performing task T has rights defined by the intersection: $A_T \cap A_R$ provided that credentials for R **in** T are available.

Cooperative working patterns often involve the delegation of some or all of a person's work in a task. Delegation is not effective unless there is a transfer of some access rights. The 'need to know' policy is even more relevant for this situation. Ideally, the rights transferred should be just those needed to perform the delegated work, but in our model, we assume that the work delegated corresponds to a role in a task. That is, the authority of the principal R **in** T is delegated. We rely on the *delegation* construct of the DEC model to enable any principal that has authority to play a role in a task to transfer that authority to another suitable principal by supplying them with *delegate credentials*. Delegate credentials specify the task and role that the delegate may perform. A delegate need not hold any special role; a check of their suitability – for example, that they belong to a specified group of principals – may be made at the time of delegation.

System implications

Our model does not assume any particular execution environment. We have studied a design for its implementation in a distributed shared virtual memory (DSVM) environment, in which object sharing is modelled by the direct mapping of objects into the address spaces of processes. Such an execution environment is an attractive basis for the implementation of cooperative applications because it offers a very direct and efficient form of object sharing.

Associated with a group task is a set of application objects that are *secure for the task*. (We use the term application object to refer to individual objects or to collections of objects with similar access rights.) Only compound principals that have R **in** T credentials should be allowed to access the secure objects.

The notion of objects that are secure for a task can be modelled quite directly in a shared memory environment with hardware-supported protection domains by placing the objects in a single protection domain and allowing access to the domain only for the permitted principals. At this basic level, the DSVM/protection domain model matches our security model well, supporting the guarantee: *only principals that have access to a group task can obtain any access to the objects that are secure for the task.*

However, our model calls for finer-grained protection within a task than that offered by a mapping onto traditional protection domains. In particular, we require that access to objects is controlled at the level of the individual operations of objects. For example, in the examination preparation task some principals will require read access to the draft examination papers, others will require the right to append questions to the examination and still others will require the right to modify any part of the examination paper.

The fine-grained protection of objects required for cooperative work goes well beyond that provided by hardware-supported protection domains, even when a separate protection domain is constructed for each active principal. A mechanism is required to control access to the individual operations of objects within a protection domain.

A language-based model for the specification of access rights and their enforcement in the GUIDE object-oriented language has been described by Hagimont [6]. It is based on the compilation of operation dispatch tables restricted according to the rights of the current process. Objects are shared through DSVM but each protection domain requires a different dispatch table. The different dispatch tables should appear at the same location in all of the protection domains, which is a problem for single-address-space DSVM systems! A solution is discussed in [1].

The use of dispatch tables cannot be enforced in weakly-typed programming languages. We are therefore left with the problem of enforcing access rights at the level of individual operations on objects in programs written in languages that are not strongly typed, such as C++. Chase *et al*. have proposed a scheme based on the use of separate domains for groups of mutually trusting objects (called protected service objects) and user-level capabilities containing operation-level rights [4]. This solution offers adequate security at the cost of a domain change for each access to a secure object.

Acknowledgements

We wish to thank Sacha Krokowiak, Jacques Mossiere and Xavier Rousset de Pina for their participation in the work described. Thanks also to Sacha Krakowiak for arranging our visit to the Bull-IMAG laboratory, and to all of the staff in the laboratory for their hospitality and support.

References

1. Mossière, J. and Rousset de Pina, X., Single address space or private address spaces?, Bull-Imag Systèmes, April 1994. (Position paper to be presented at Sixth European SIGOPS Workshop).
2. Wobber, E., Abadi, M., Burrows, M. and Lampson, B.W., Authentication in the TAOS operating system, *ACM Operating Systems Review*, December 1993, pp. 256-269.
3. Lampson, B.W., Abadi, M., Burrows, M. and Wobber, E. Authentication in Distributed systems: Theory and Practice. *ACM Trans. on Computer Systems*, 10, 4, Nov. 1992, pp. 265-310.
4. Chase, J. S., Levy, H.M., Lazowska, E.D., Baker-Harvey, M., Lightweight shared objects in a 64-bit Operating System, Proc. OOPSLA ’92, pp. 397-413, ACM Press 1992.
5. Coulouris, G.F. and Dollimore, J., Requirements for security in cooperative work: two case studies, Technical Report 671, Department of Computer Science, Queen Mary and Westfield College, May 1994.
6. Hagimont, D. Protection in the Guide object-oriented distributed system. To appear in *Proceedings of ECOOP 94*.