The Newcastle Connection as a Mechansim for providing a Multi-Level Secure File Store

J.E. Dobson

MARI Advanced Microelectronics Ltd

Abstract

This paper describes the overall architecture of a distributed computer system that is designed to enforce a multi-level security policy. The system is composed of standard UNIX* systems and small trustworthy security mechanisms linked together in such a way as to provide a highly cost effective secure system. This system appears from the point of view of its users to be a single multi-level secure UNIX system, since the fact that it is a distributed system is completely hidden from the users and their programs. This is achieved though the use of the "Newcastle Connection", a software subsystem that links together multiple UNIX systems into a single virtual UNIX system in which each component behaves as a directory in the overall UNIX United name tree.

* UNIX is a Trademark of Bell Laboratories

The main principle for the constuction of secure systems is to keep entities of different security classifications apart from one another, except when performing operations that require access to more than one level. These latter operations must be performed under the surveillance of trustworthy 'reference monitors' that encapsulate and ensure compliance with some externally imposed 'security policy'. Hence our approach to the design of a secure system is based on the two key notions of separation and mediation : the separation of entities of different security classification and the mediation of communication channels between entities of different classifications.

Modern distributed computing systems have in principle a structure which matches these twin principles, since they comprise a number of physically separated components each of which can potentially be dedicated to a single security level or specialised function. In order to achieve security it is then necessary to control communications between the distributed components and to provide trustworthy reference monitors for multi-level operations. Our design involves connecting a number of large untrusted 'host' systems together with some small specialised processors which are placed between the hosts and the underlying communications medium. The host systems will provide services to a single security partition and also provide file storage facilities; since it will be physically impossible for them to communicate without the messages passing through the interface units, the latter will act as the reference monitors in the overall system.

Although the system we wish to construct is distributed, we wish to hide the exact details of the distribution from the user and present a single system image. This transparency is most easily achieved if all user-visible system components have a common interface. And because it is desirable to admit the possibility that the components may themselves be distributed systems in their own right, we have constructed our system according to the Recursive Structuring Principle [1]:

> Each component of a distributed system should be functionally equivalent to the entire system of which it is a part.

The value of this principle is that it permits a system structured in this way to be indefinitely extended. It requires, however, that the component systems possess external characteristics that are appropriate for the system as a whole. The design of a single system interface that is equally suitable for a whole and for its components is a non-trivial task, and we do not feel that it is worth the effort to improve significantly on UNIX. We have found that the Newcastle Connection [2], which extends a uniprocessor UNIX to a single UNIX system spanning any number of component processors, to be a most suitable vehicle for the construction of a distributed secure system.

Other designers [3] have also identified UNIX as a suitable base from which to start, and recognising its untrustworthiness have attempted to patch in those features thought necessary to bring it up to the status of a secure operating system. We have chosen not to do this, firstly because we do not feel capable of the work involved, and secondly because we do not believe it to be necessary since an alternative and simpler approach can achieve the same end. We start from the premiss that all the component systems are untrustworthy and therefore the overall system may make no assumptions about their behaviour, except that the LAN provides the only means of inter-communication. We then investigate what mechanisms need to be interposed between the components in order to provide suitable reference monitors that can then guarantee the behaviour of the system as a whole. In order to do this we need first to examine the kinds of mechanism available, which we do by means of a simple classification.

In order to develop the security features of our design in more detail, we need to discuss the various mechanisms available for enforcing the separation required in a secure system. We have identified four different means by which this can be achieved. The four methods can be categorised into physical, temporal, cryptographical, and logical mechanisms. We shall discuss each of them separately.

As its name implies, physical separation keeps objects of different classifications physically separate. For example, CONFIDENTIAL and SECRET items will use dedicated, physically separate memory boards, disks, and machines. The advantage of this approach is that separation is manifest; its disadvantages are cost and inflexibility. For example the introduction of a new security compartment will require the introduction of new hardware components.

The temporal approach does allow common hardware to be used for different security compartments, but not simultaneously. Hardware components are time-shared between compartments, and are therefore required to be memoryless (i.e. none of their system state may persist across activations at different security levels). The temporal approach is often applied manually, and to entire systems. However, it can also be applied to individual system components and can be automated.

The cryptographical approach achieves separation by encrypting information of different security partitions under different cryptographic keys. There are only a very few security-related operations that can be performed which require the application of this technique (basically it is only applicable to bulk movement or storage operations) but in general it is the only mechanism suitable for those applications.

The fourth approach, the logical one, is one in which a higher-level mechanism (often implemented in software) manages simpler mechanisms of one or more of the three basic types previously mentioned -either in order to control their behaviour or in order to synthesize separate logical entities out of lower-level components. This approach is very powerful and flexible, but guaranteeing the separation can be non-trivial, involving for example the application of formal techniques.

The purpose of the taxonomy we have just introduced is to show that there are several types of mechanism available, and that each has its own advantages and disadvantages and area of application. We have done this because our system will incorporate mechanisms of all four types and use each type wherever it is most appropriate.

As previously mentioned, the unit of protection in our distributed system is the component system, since we are not prepared to trust a UNIX. Although there is no security within a system, we can enforce security on the communication of information between systems by placing a trustworthy mediation device or reference monitor between each system and its network connection. These 'Trustworthy Network Interface Units' or TNIUs (at least in the first instance) permit communication only between systems belonging to the same security partition. In its simplest form, it merely monitors the address fields of each incoming or outgoing packet, passing only those packets which are not attempting to cross partition boundaries. In addition, in order to provide security of packets on the network against eavesdroppers, the TNIUs are also responsible for the cryptographic function.

It is symptomatic of the difficulty of security issues that even this simple architecture turned out on examination to be not nearly so simple as it might appear. This is not the place to discuss the problems in detail, but it can be said that they arise from the need for the protocols not only to be formally correct but also to be trustworthy and secure. The issue of assigning functions to layers in a protocol hierarchy can become quite complex in the presence of encryption.

So far, we have merely considered how resources can be separated by security level: information may not flow from one partition to another. This can be achieved by allocating physically separated computer systems to each partition with specialised monitor processors to provide cryptographic protection and to enforce the separation between component systems belonging to different compartments. We now wish to extend the system so that information can be moved across partition boundaries in a controlled secure manner to provide true multi-level security. This will allow, for example, information to flow upwards but not downwards in the security lattice.

Again we introduce the function of a reference monitor to mediate such information flow. The complexity of the monitor will depend on the generality of the services it provides and the granularity and complexity of the objects whose flow is controlled. For simplicity we have chosen files as the only objects that will be allowed to cross security boundaries, with the implication that the services to be provided by the reference monitor are the (multi-level secure) storage and retrieval of files. It is a consideration also that the choice of the file as the unit of granularity fits in well with the UNIX philosophy, which also uses the file as the unit of naming and external storage. The idea is that when a system wishes to make one of its files available to higher levels, it sends it to the filestore. Another machine can then, subject to security policy, request a copy of this file from the filestore.

Conceptually, the filestore is just an ordinary UNIX system that is associated with a directory (SFS, say) in the UNIX United directory structure. This directory contains sub-directories for each security partition in the overall system and these in turn will be structured according to the user's needs. However, the structure just described exists in name space; it does not reflect the functional requirements of the filestore, which has two separate concerns to address: the security of the system, and the archiving of the files.

Thus we need to partition the filestore into two components which according to the principle of physical separation are housed in distinct machines. The first component, called the Secure File Manager (SFM) will be a small trustworthy component concerned with the enforcement of security policy, while the second component, called the Isolated File Store (IFS) will be a larger component whose task is to provide the actual file system. The SFS directory in name space will be identified with the machine that houses the SFM, but the entire UNIX file subsystem below that directory node will be physically held separately on the IFS machine - which must therefore be a machine capable of maintaining a UNIX file system. One obvious way of satisfying that requirement is to use a perfectly standard UNIX system for the IFS.

Because the IFS will contain files of all security classifications, and because it is untrusted (and possibly untrustworthy), it must obviously be presented from communicating with the outside world and also from corrupting the files entrusted to its care. Thus all its communications must be mediated by the SFM, which must also be responsible for its good behaviour. These are two separate functions of the SFM, and we therefore partition the SFM into a File Access Reference Monitor (FARM) and a File Integrity Guarantor (FIG). The task of the FARM is to ensure that file access requests are granted only in accordance with the security policy, and that of the FIG is to guarantee that the IFS does not leak information or corrupt either the files or the file structure it is supposed to maintain.

Again, the details of the design and implementation of these components proved significantly more complex than might be expected (though this will surprise nobody who has already themselves attempted the task). What made our approach more tractable than previous attempts was the ease with which UNIX (and UNIX United in particular) permits the separation of naming issues from addressing issues, and both of these from access issues. Because UNIX provides access only through the system call interface, and because it is the system call that is encapsulated in an inter-processor message, it is possible to conceal from the caller whether or not it is a UNIX system that responds to the call - it only need appear to be a UNIX. The trusted mechanism through which the requests must be channelled can be made quite invisible, and this requirement for invisibility constrains and simplifies the design.

Acknowledgements

Brian Randell and John Rushby originated many of the ideas behind this approach (4), and I have enjoyed many hours of discussion with them. The work has been supported by RSRE Malvern, and my gratitude is due to Derek Barnes for his continuing help and encouragement.

References

1. B. Randell, Proc. 3rd Symp. on Reliability of Distributed Software and Database Systems, IEEE (Oct 1983).

2. D.R. Brownbridge, L.F. Marshall and B. Randell, "The Newcastle Connection or UNIXes of the World Unite", Software Practice and Experience, Vol 12 (December 1982) pp. 1147-1162.

3. B.J. Walker, R.A. Kemmerer and G.J. Popek, "Specification and Verification of the UCLA Unix Security Kernel", CACM Vol. 23(2) (February 1980) pp.118-131.

4. J.M. Rushby and B. Randell, "A Distributed Secure System", IEEE Computer Vol. 16 No.7 (July 1983).

DISCUSSION

Professor Goos addressed the problem of the network which was being used to provide the interconnection between the component systems. In particular, how did the DSS solve the problem of finding out about corruption of messages. Mr. Dobson agreed that this was a problem which the TNIU had to be aware of. Normally the corruption would be innocent, and would be attributable to the usual types of error to which networks are susceptible, but when dealing with a security system, it has to be assumed that all corruptions are malicious. This is recognised as being a difficult problem in the design of the TNIU.

Dr. Freeman referred to the remark in which Mr. Dobson had suggested that intelligence within terminals was an undesirable feature. Some part of the system design must surely take account of the possibility of intelligent terminals being used to penetrate the security of the system. Mr. Dobson agreed that the system must protect itself against wire-tapping, but since the data crossing the network would all be encrypted, it was the protection of the encryption keys which was the crucial factors. The TNIU must be responsible not only for checking the credentials of the (supposed) originator of the message, but also for decrypting the message according to an appropriate key to decide whether it is a valid message.

Dr. Lipner pointed out that the secure file system did implement the 'star-property' (or containment property) but would also allow the possibility of 'trojan horse' penetration. Mr. Dobson agreed that this was so, and that the designers were aware of the problem, but declined to give any further details of the method of solution.