Abstract:

We describe, in tutoriual detail, the design of a distributed general-purpose computing system that enforces a multilevel 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 total system which is not only demonstrably secure, but also highly efficient and cost effective.

Each UNIX system provides services to a single security partition and operates at full speed; security-critical tasks are performed by separate, specialised processors. These security processors control access to the different security partitions and mediate information flow between them. They also provide a multilevel secure file system and a facility for dynamically changing the security partition to which each UNIX system is assigned. Extensions to support controlled downgrading and multilevel objects are described as well.

Despite the sophistication of the overall system, individual security processors employ only very simple, straightforward mechanisms; their construction and verification requires no more than already established technology. And despite the heterogeneity of its compoents, the system as a whole appears to be a single multilevel secure UNIX system, since the fact that it is actually a distributed system is completely hidden from its users and their programs. This is achieved through the use of th "Newcastle Connection", a software subsystem that links together multiple UNIX or UNIX-look-alike systems, without requiring any changes to the source code of either the operating system or any user programs.

A first prototype system, providing multiple security partitions, and a multilevel secure file system, has already been successfully demonstrated - construction of a much more complete prototype is now planned.

A Distributed Secure System

By

J.M. Rushby and B. Randell

TECHNICAL REPORT SERIES

Editor: Mr. M.J. Elphick

Number 182
May, 1983

Printed and published by the University of Newcastle upon Tyne,
Computing Laboratory, Claremont Tower, Claremont Road,
Newcastle upon Tyne, NE1 7RU, England.

bibliographical details

RUSHBY, John Martin

A distributed secure system [By] J.M. Rushby and B. Randell

Newcastle upon Tyne: University of Newcastle upon Tyne, Computing Laboratory, 1983.

(University of Newcastle upon Tyne, Computing Laboratory, Technical Report Series, no. 182.)

Added entries

RANDELL, B.
UNIVERSITY OF NEWCASTLE UPON TYNE.
Computing Laboratory. Technical Report Series. 182.

Suggested classmarks (primary classmark underlined)

Dewey (18th): 001.64404 658.47
U.D.C. 519.687 519.718

Suggested keywords

DISTRIBUTED SYSTEMS
MULTILEVEL SECURE FILE SYSTEM
NEWCASTLE CONNECTION
SECURITY PARTITIONS
UNIX

Abstract

We describe, in tutorial detail, the design of a distributed general-purpose computing system that enforces a multilevel 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 total system which is not only demonstrably secure, but also highly efficient and cost effective.

Each UNIX system provides services to a single security partition and operates at full speed; security-critical tasks are performed by separate, specialised processors. These security processors control access to the different security partitions and mediate information flow between them. They also provide a multilevel secure file system and a facility for dynamically changing the security partition to which each UNIX system is assigned. Extensions to support controlled downgrading and multilevel objects are described as well.

Despite the sophistication of the overall system, individual security processors employ only very simple, straightforward mechanisms; their construction and verification requires no more than already established technology. And despite the heterogeneity of its components, the system as a whole appears to be a single multilevel secure UNIX system, since the fact that

Continued.

About the author

Dr. Rushby was a Research Associate in the Computing Laboratory from 1979-1982. He is now a Computer Scientist with SRI International, Menlo Park, California.

Professor Randell has been professor of Computing Science in the Computing Laboratory since 1969.

it is actually a distributed system is completely hidden from its users and their programs. This is achieved through the use of th "Newcastle Connection", a software subsystem that links together multiple UNIX or UNIX-look-alike systems, without requiring any changes to the source code of either the operating system or any user programs.

A first prototype system, providing multiple security partitions, and a multilevel secure file system, has already been successfully demonstrated - construction of a much more complete prototype is now planned.

TABLE OF CONTENTS

A Distributed Secure System

J.M. Rushby
B. Randell

Computing Laboratory
University of Newcastle upon Tyne
Newcastle upon Tyne NE1 7RU
England

Tel.: Newcastle (0632) 329233
Arpanet mail: NUMAC at SRI-CSL

1. Introduction

Attempts to construct secure general-purpose computing systems have not, so far, been notably successful. A study conducted by IBM to examine the state and prospects of current technology reported [Lomet82, p32]:

> "As a result of our study, we are forced to conclude that the 'secure' operating systems extant or nearing completion all appear to have fallen short of their intended goals. None appear to meet even the minimum performance goals that have been specified, none have successfully been verified to the extent desired or originally intended, and whether these systems satisfy their original security goals is an open question.
>
> "... It is our belief that the causes of the difficulty lie in inadequate tools, methods, processor hardware, and design ideas; inadequate to the extent of precluding a fully successful effort to produce a provably secure, adequately performing operating system in the near future."

In this paper we propose a system design which we hope will refute the pessimistic conclusion of this IBM study. Our technique is to finesse the problems that have caused difficulty in the past by aiming to build, not a secure operating system, but a distributed secure system.

A sound principle for structuring secure systems is to keep entities of different security classifications completely separate from one another - except when performing operations that require access to entities from more than one level. These latter operations must be performed by, or under the control of, trustworthy 'reference monitors' which ensure compliance with some externally imposed security policy.

By 'separation' we mean absence of information flow, and it seems clear that the fewer the resources that are shared across security levels, the simpler it should be to achieve separation between those levels. Unfortunately, the structure of conventional, centralised systems is antithetical to this very natural requirement: they comprise a single resource which must be shared between a number of users and functions. For secure operation, trustworthy components within their operating systems must synthesise separate 'virtual' resources from the shared resources actually available. The mechanisms that perform this 'synthetic separation' are not only inimical to the efficiency of the system, but are generally complex - making it difficult to guarantee their own correctness.

In contrast to traditional, centralised systems, modern distributed systems seem rather well matched to at least one of the requirements for secure operation: they necessarily comprise a number of physically separated components, each of which can, potentially, be dedicated to a single security level or to a single function. In order to achieve security, it is then only necessary to control communications between the distributed components and to provide trustworthy reference monitors for security-critical operations. The real challenge here is to structure the system so that the separation naturally provided by physical distribution is fully exploited to simplify the mechanisms of security enforcement without destroying the coherence of the overall system.

Our proposal for a Distributed Secure System involves connecting some small, specialised, provably trustworthy systems and a number of larger, untrusted 'host' machines in such a way as to provide a total system which is not only demonstrably secure, but also highly efficient, cost-effective, and convenient to use. The untrusted host machines will each provide services to a single security partition and will continue to run at their full speed. The trusted components will mediate communications between the untrusted hosts and will also provide specialised services such as multilevel secure file storage.

Any mention of security in the context of distributed systems immediately invites proposals based on cryptographic techniques and a number of such designs have already appeared [Berson82, Davida81a, Davida81b, Gasser82]. Our approach also makes use of cryptography, but differs from others in two important respects.

Firstly, whereas most other proposals largely eschew the use of trusted software (indeed, this is the primary motivation for some [Davida81b]), we embrace its use wherever it is the most appropriate mechanism. It is important to stress here that when we speak of 'trusted software', we mean software that is provably trustworthy; it is trusted by virtue of properties it has been shown to possess, not by virtue of the use to which it is put. Use of trusted software enables us to provide far greater functionality than systems based solely on encryption, but despite the great functionality provided in our total system, its trusted components <u>individually</u> provide only single and relatively simple functions; none of our trusted software demands more, either by way of efficiency or of verification, than has already been achieved in stand-alone applications of a similar nature. Indeed, several of the trusted mechanisms that we propose to use are directly derived from, or are (we belatedly find) re-inventions of, those already incorporated in such applications.

Secondly, and unlike, for example, those of [Berson82, Gasser82], our proposal confronts its users not with a network of systems, but with a system: all issues of networking and distributed processing are transparent to the user, who is given the impression of using a single multilevel secure system. This is achieved by placing our mechanisms within the environment provided by the `Newcastle Connection' [Brownbridge82]. This scheme enables a number of inter-linked, standard UNIX* systems to be composed into a coherent, distributed system which is functionally indistinguishable from a single UNIX. Within the present paper, we will refer to this distributed system as UNIX UNITED. Of course, neither UNIX UNITED nor even UNIX are necessary to the security enforcement mechanisms described in the following sections, but we consider them to provide a uniquely attractive environment in which to implement and use these mechanisms.

The remainder of this report is organised as follows. Section 2 describes the structuring principles for secure and distributed systems that underlie our work, and the basic mechanisms which we employ. Sound design principles and careful structuring are crucial to the success of any complex system design - and especially to those that claim such an elusive property as security. Section 3 describes the cryptographic techniques we propose for controlling communications between the components of our distributed system. The ability to control and protect these communications is crucial, not only in order to provide separation between security levels, but also in order to provide secure access to the special, trustworthy mechanisms that perform the movement of information across those levels. Section 4 outlines the design of one such special mechanism: a Secure File System that provides for the multilevel secure storage and accessing of files across different security levels. The construction of this system, which has the appearance of a multilevel secure UNIX file system, is achieved with a surprisingly small quantity of trusted mechanism. Section 5 proposes mechanisms for providing convenient terminal access to different security levels, and for rapidly and automatically changing the security level at which component computer systems (including personal workstations) operate. Once again, very little additional trusted mechanism is needed in order to provide these functions, especially the latter one. Section 6 explains how systems providing highly specialised secure services, such as downgrading and support for multilevel objects, can be smoothly integrated into the overall system, and also touches upon the problem of inter-networking. Finally, in Section 7, we present our conclusions and describe the current status of the project and our plans for the future. Readers who wish to skim should note that each of sections 3 to 6 ends with a brief summary.

2. Principles and Mechanisms for Secure and Distributed Systems

We have based our work on two orthogonal design principles. Our principle for designing secure systems is summed up in the phrase `separate and mediate', while that for distributed systems is called, rather grandly, the `recursive structuring principle'. These are described in the following sub-sections, together with mechanisms for realising them.

---

*UNIX is a Trademark of Bell Laboratories.

2.1. Secure Systems

We wish to to develop a system capable of processing information of many different sensitivity levels, while enforcing a mandatory security policy concerning disclosure of information to users of different clearances. Precise details of the security policy to be enforced are unimportant, but it is the military 'multilevel' scheme with levels and compartments that we have generally held in mind [Landwehr81]. (Karger [Karger78] and Lipner [Lipner82] discuss interpretations of this policy which may be suitable for civilian environments.) We are concerned only with the threat of simple disclosure and do not address problems of aggregation and inference, nor of integrity and denial of service. We assume that the environment provides adequate physical security to protect the system from direct external attack and that measures for procedural and personnel security safeguard the handling of information outside, and at the interfaces to, the system.

Our primary concern is with threats internal to the system. In particular, we assume, unless it has been proved otherwise, that each of its own components, whether hardware or software, may be a 'Trojan Horse': that is, a component supplied by an adversary with the intent of subverting any security provided by the rest of the system. We assume that such Trojan Horses may exploit subtle techniques involving the use of clandestine communication channels [Lampson73] if such are present. Readers unfamiliar with these topics may wish to consult the previously cited references, and also the excellent book by D.E. Denning [Denning82].

Principles

In order that its users may be assured that a system is able to maintain security in the face of threats such as those described above, it is necessary to structure that system very carefully - so that all the factors affecting its security may be exposed to scrutiny. All mechanisms critical to security must be readily identifiable and their precise role, and consequently the properties required of them, ascertained. Furthermore, the internal structure of those mechanisms must be sufficiently simple that it is possible to adduce compelling evidence that they do, indeed, possess the properties required of them.

The structure of all secure systems constructed or designed recently has been influenced by the idea of a reference monitor - a concept first described in the Report of the Anderson Panel [Anderson72, vol. 2, p. 22]:

> "... the reference monitor mediates each reference made by a program in execution by checking the proposed access against a list of accesses authorised for that user."

In other words, whenever a program requires access to data or, more generally, whenever it wishes to communicate outside its own 'domain', the proposed access or communication must be mediated and checked by a trustworthy reference monitor in order to ensure that it complies with the appropriate security policy.

It is implicit in this idea, but utterly fundamental to its appreciation and application, that information and users (and the programs

operating on behalf of users) belonging to different security classifications should be kept totally separate from one another. That is to say, there must be no channels for the flow of information between, or among, users and data of different security classifications, except those mediated by reference monitors.

Our approach to the design of secure systems is based on the twin key notions identified above - separation and mediation:

> It is necessary to separate entities of different security classification, and to mediate and control the communication channels between entities of different classifications.

The nature of the security policy that can be enforced will depend on the granularity of the separation and on the reference monitors that are provided to mediate the movement of information.

There is another form of 'separation' that has influenced our work: it is the software engineering principle of 'separation of logical concerns'. Separation and mediation reflect separate logical concerns and, in the interests of intellectual manageability, not to mention ease of development and verification, the mechanisms which realise them are best kept separate also. Such separation of concerns also accords well with with the principles of 'least privilege' and 'least common mechanism' which Saltzer and Schroeder [Saltzer75] have identified as useful guidelines for the design of security and protection mechanisms.

Mechanisms

Examination of several exisiting secure systems reveals that they fail to distinguish between the concerns of separation and mediation - indeed they fail to recognise separation as an explicit issue at all. Despite making obeisance to the reference monitor concept, the most obvious aspect of the structure of UCLA Data Secure UNIX (DSU) [Popek79], KVM/370 [Gold79], and KSOS [McCauley79] is that they are kernelized systems: in each case, their operating system is built around a (not so) small nucleus (called a 'security kernel') into which all security-relevant operations are concentrated. Or not quite all - for these systems contain 'trusted processes' which also perform security-critical operations, but which are outside the kernel. Furthermore, the role and the security properties required of trusted processes are different from those of the kernel and the basis for their verification has not been clearly established. Accordingly, the structure of these systems has been criticised by ourselves and others [Ames81, Rushby81] - principally on the grounds that it does not match that of the models which serve as the basis for their verification [Bell76, Feiertag77, Landwehr81, Popek78] and because the division into kernel and trusted processes is not a reflection of separate logical concerns.

That the security mechanisms of these systems are not structured as simply as one would like is probably due to the fact that they are big systems: they each aim to provide a secure version of a full, general-purpose, multi-user operating system. (UNIX in the case of UCLA DSU and KSOS, and VM/370 in the case of KVM/370.) The problems of supporting such powerful systems fetter the pursuit of a simple security structure - for the need to achieve acceptable performance causes certain operating system functions to be brought inside the kernel interface when pure

security considerations would dictate that they should remain outside. Despite this compromise, the efficiency of these systems is not high. KSOS and UCLA DSU seem at least an order of magnitude slower than standard UNIX, while KVM/370, in many ways the most successful of these projects, is currently reported to be three to four times slower than VM/370 (and seven to eight times slower for I/O operations) [Hinke82]. Big kernelized systems suffer from other disadvantages as well as poor performance. Because their internal structure differs considerably from the 'base' system from which they are derived, and because even small changes to a kernel may invalidate its entire verification, these systems tend to lag many 'releases' behind the current version of their base operating system. KVM/370, for example, is based on a much earlier version of VM/370 than that currently available.

It is possible, however, despite the examples cited above, to construct kernelized systems which support certain limited applications with acceptable efficiency, and in which the distinction between kernel and trusted processes does correspond to a separation of logical concerns. The SUE kernel [Barnes80, Barnes81], for example, has no other function but separation: it synthesises a number of isolated 'regimes' within a shared PDP-11/34. The trusted processes of this system are reference monitors which run in certain regimes in order to mediate and control the communications between other, untrusted regimes. We have dubbed this type of kernel a separation kernel [Rushby81] and have argued that it admits a more complete and convincing verification technique [Rushby82] than other types of kernel. We have also argued that by leading to a clean separation of concerns, this approach allows a secure system to be decomposed into smaller components, with a more transparent structure, than found in a monolith such as KSOS. It is encouraging to see that the designers of certain other recent systems have arrived, independently, at a broadly similar approach [Golber81, Grossman82].

These relatively 'pure' separation kernels have proved acceptably efficient in supporting the specialised applications for which they were intended. In each case, the application has been sufficiently well matched to the functionality of its separation kernel that it can be supported on the kernel interface directly, without the interposition of an operating system layer.

Experience therefore indicates that security kernels can provide a sound and efficient separation mechanism for certain limited applications but they have proved, and are likely to remain, inadequate to the task of supporting general-purpose applications.

Fortunately, a kernel is not the only mechanism available for enforcing the separation required in a secure system. In fact, it exploits just one of the four different ways by which this end can be achieved. The four methods depend, respectively, upon physical, temporal, cryptographical, and logical mechanisms.

As its name suggests, the first of these approaches keeps material of different security partitions apart physically. Thus CONFIDENTIAL and SECRET items will use dedicated, physically separate memory boards, disks - even (depending on the granularity of the separation) separate machines. The advantage of this approach is that its separation is utterly manifest; its disadvantages are cost (physical devices must be

replicated) and inflexibility (new security partitions are not easily introduced).

The temporal approach does allow common hardware to be used for different security classifications, but not simultaneously. System components are time-shared between different security partitions and are required to be memoryless: none of their system state may persist across activations at different security levels. This approach is the basis of 'periods processing', which is the current practice in many classified environments. In this form, separation is achieved through exchange of all demountable storage and initialisation of all other storage to fixed values. As exemplified by periods processing, the temporal approach is usually applied manually, and to entire systems. However, it is also applicable to individual system components and may be automated. As we will show, it then has interesting applications in the context of distributed systems.

The cryptographical approach achieves separation by encrypting information of different security partitions under different cryptographic keys. The quality of the separation achieved depends upon the cryptographic strength of the encryption technique employed, the care with which the encryption is managed, and the protection of its keys. There are few operations that can be performed on encrypted information (basically, it can be moved from one place to another or it can be stored for later retrieval) and, in consequence, the cryptographical approach is restricted in its applications. However, it is generally the only approach that is available for those applications.

The final approach, the logical one, is not really a distinct technique in the sense that the others are. Rather, we use the term to describe the approach in which a higher-level mechanism, usually implemented in software, manages simpler mechanisms of the more basic types - either in order to control their behaviour, or in order to synthesise separate logical entities out of more basic resources. A separation kernel, for example, may synthesise separate 'virtual machines' by controlling a hardware memory management unit in order to provide separate storage areas, and by using temporal separation to provide the appearance of multiple CPUs. The advantages of the logical approach are that it is very flexible (new security partitions can be created and destroyed dynamically), it can provide separation at a very fine level of granularity, and it can be economical: it is sometimes possible to share common hardware quite efficiently in this way. On the other hand, this is very definitely a high-technology approach: guaranteeing the separation provided by a logical mechanism requires application of the techniques of formal program specification and verification.

We do not claim that our taxonomy of separation mechanisms is at all profound, but we have found it helpful to recognise that there are several mechanisms available, and that each of them has its own particular advantages and disadvantages. Existing secure system designs tend to exploit only one of the four methods and must therefore accept the disadvantages of the particular approach chosen. In contrast, we will propose a system structure which (potentially) incorporates all four approaches and which uses each wherever it is the most appropriate.

The introduction of systems composed of individual computers connected by a local-area network (LAN) now makes such a 'combined'

approach attractive, if not inevitable. By the very nature of such a distributed system, its component computers are physically isolated from one another, and cryptography provides a natural method for securing communications over the network. The logical (kernel-based) approach to separation is appropriate to the internal organisation of the components that manage cryptographic and other specialised functions.

The most attractive feature of this approach to the provision of secure computing is that it enables the mechanisms of security enforcement to be isolated, single-purpose, and simple. Another benefit is that allows its (untrusted) component computer systems to provide their full functionality and performance. We therefore believe that with this approach it possible to construct secure systems whose verification is more compelling, and whose performance, cost, and functionality are more attractive, than is the case at present.

To be truly useful, such a heterogeneous network (comprising both untrusted general-purpose systems and trusted specialised components) must be made to operate as a single coherent system. We do not, therefore, consider the following discussion of principles and mechanisms for constructing distributed systems to be in any way a digression from our main concern.

2.2. Distributed Systems

Principles

Some designers of distributed computing systems have taken as their main structuring principle the identification of functions that can be allocated to separate computers and implemented as so-called 'servers' - name servers, file servers, boot servers, and so on [Needham82]. In this case, the fact that the system is actually constructed out of a number of autonomous computers, interacting via some sort of communications network, may be clearly evident. How a program accesses a file, for example, may depend on whether or not the program and the file are held in the same computer. Other distributed systems have been designed to provide what is sometimes called 'network transparency', so that this difference is hidden [Popek81]. Both approaches have their merits: the first enables physical resources to be used efficiently and optimised for certain functions, while the second is clearly a more attractive interface to present to the users of the system. We would like to retain the advantages of both.

Network transparency is most easily achieved if all system components have a common interface. We therefore propose that all those components of a distributed system which contain user-visible state information should appear to all the other components with which they interact to be complete and functionally equivalent computing systems. In fact, because it is desirable to admit the possibility that system components may, themselves, be distributed systems, we should demand more than this, and require that:

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

This is our 'recursive structuring principle' for the design of distributed systems. It may seem to preclude systems composed of heterogeneous components (such as servers), but this is not so. Any system interface must contain provisions for 'exception conditions' to be returned when a requested operation cannot be carried out. Just as the operating system of a host machine may return an exception condition when asked to operate on a non-existent file, so a specialised server that provides no file storage may return exceptions when asked to perform any file operations whatsoever.

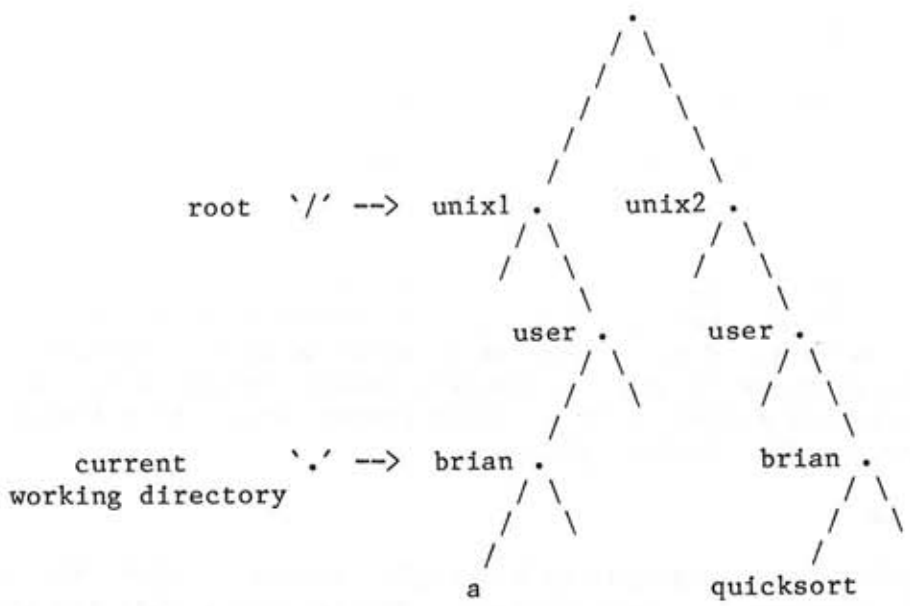
The value of the recursive structuring principle is that, by definition, a distributed system that is structured in this way is indefinitely extensible. Moreover, users interact with the system in the same way, and run their programs unchanged, independently of whether the system is actually supported by a single processor, or by a homogeneous or heterogeneous distributed system.

Mechanisms

The Recursive Structuring Principle requires that the component computer systems possess external characteristics that are appropriate for the distributed system as a whole. As discussed in [Randell82] they must possess at least the appearance of providing parallel processing facilities, and all the objects within a system (computers, data items, devices, programs, etc.) must be accessible by means which are independent of whether the system is in fact a complete one, or merely a component of a larger one.

The design of a single system interface that is equally appropriate to a distributed system as a whole and to its components, whether complete individual systems or specialised servers, would seem to be a task requiring exceptional care, judgement, and good taste. And so it proved! It was performed in the early 70's by the designers of UNIX [Ritchie74].

We have found the UNIX kernel interface almost perfectly suited to serve as the basis for the distributed system which we call UNIX UNITED. A UNIX UNITED system is composed of a (possibly large) set of interlinked standard UNIX systems (or systems that can masquerade as UNIX at the kernel interface level), each with its own storage and peripheral devices, accredited set of users, system administrator, etc. The naming structures (for files, devices, commands and directories) of each component UNIX system are joined together into a single naming structure, in which each UNIX system is, to all intents and purposes, just a directory. The result is that, subject to proper accreditation and appropriate access control, each user, on each UNIX system, can read or write any file, use any device, execute any command, or inspect any directory, regardless of which system it belongs to. The simplest possible case of such a structure, incorporating just two UNIX systems, named as 'unix1' and 'unix2', is shown below.



From unix1, with the root ('/') and current working directory ('.') as shown, one could copy the file 'a' into the corresponding directory on the other machine with the shell command

```
cp    a  /../unix2/user/brian/a
```

(For those unfamiliar with UNIX, the initial '/' symbol indicates that a path name starts at the root directory, rather than the current working directory, and the '..' symbol is used to indicate a parent directory.)

This command is in fact a perfectly conventional use of the standard 'shell' command interpreter, and would have exactly the same effect if the naming structure shown had been set up on a single machine, with 'unix1' and 'unix2' actually being conventional directories.

All the various standard UNIX facilities (whether invoked via shell commands, or by system calls within user programs) concerned with the naming structure carry over unchanged in form and meaning to UNIX UNITED, causing inter-machine communication to take place as necessary. It is therefore possible for a user to specify a directory on a remote machine as being his current working directory, to request execution of a program held in a file on a remote machine, to redirect input and/or output, and to use files and peripheral devices on a remote machine.

The convention in UNIX UNITED is that a program is executed on the machine where its code is held. Thus, using the same naming structure as before, the further commands

```
cd  /../unix2/user/brian

quicksort  < a  > /../unix1/user/brian/b
```

have the effect of changing the current working directory, applying the quicksort program on unix2 to the file 'a' which was previously copied across to it, and sending the resulting sorted file back to a newly created file 'b' on unix1. Alternatively, the command line

```
/../unix2/user/brian/quicksort  < /../unix2/user/brian/a  > b
```

would have had the same effect, without changing the current working directory. If the user had set the shell variable `other' as follows:

```
other = /../unix2/user/brian
```

then the previous command could have been abbreviated, using the shell's `$' convention, as

```
$other/quicksort  < $other/a  > b
```

The standard UNIX mechanisms for constructing process `pipelines' also carry over unchanged to UNIX UNITED. Thus, with a more extended directory tree than that shown above (and with the shell variables u1, u2, etc. set to name the directories containing utility programs in machines unix1, unix2, ... , respectively), a command line such as

```
$u2/neqn /../unix1/user/brian/paper | $u3/tbl | $u4/nroff
```

could be entered at a terminal logged in to unix5, say. This would cause the preprocessor for mathematical text, `neqn', to be run on machine unix2, with its data being fetched from unix1 and the results being piped to the table preprocessor `tbl' and the text formatter `nroff' running in parallel on machines unix3 and unix4, respectively. If unix1, unix2, etc. had merely been directories on a single UNIX machine, then the same command would have caused `neqn', `tbl', and `nroff' to be run in an interleaved fashion.

It is worth reiterating that these are completely standard UNIX facilities, and so can be used without conscious concern for the fact that several machines are involved. Furthermore, the directory naming structure of a UNIX UNITED system is set up to reflect the desired logical relationships between its various machines; it is quite independent of the routing of their physical interconnections [Black82].

UNIX UNITED has been implemented without changing the standard UNIX software in any way; we have not reprogrammed the UNIX kernel, nor any of its utility programs - not even the shell command interpreter. This has been achieved by incorporating an additional layer of software - the Newcastle Connection - in each of the component UNIX systems. This layer of software sits on top of the resident UNIX kernel; from above, it is functionally indistinguishable from the kernel, while from below, 2t appears to be a normal user process. Its role is to filter out system calls that have to be re-directed to another UNIX system, and to accept system calls that have been directed to it from other systems. Communication between the Connection layers on the various systems is based on the use of a remote procedure call protocol [Panzieri82, Shrivastava82], and is shown schematically below.

All requests for system-supported objects (such as files) ultimately result in procedure calls on the UNIX Kernel interface. If the service or object required is remote, rather than local, then the local procedure call is simply intercepted by the Newcastle Connection and replaced with a remote one. The substitution of remote for local procedure calls is completely invisible at the user or program level. This provides a powerful, yet simple way of putting systems together - but,

equally, it provides a means of partitioning a single system into a number of distributed components.

This is the crucial property of UNIX UNITED from our perspective, since it enables a large insecure system to be broken into a number of physically separate components with no visible change at the user level. In the following sections we will explain how we exploit this physical separation in order to construct a secure system. We will begin with a very simple system that merely isolates different security classifications from one another.

3. A Securely Partitioned Distributed System

We assume the environment of a UNIX UNITED system composed of standard UNIX systems (and possibly some specialised servers that can masquerade as UNIX) inter-connected by a local area network (LAN). We will use the terms 'host', 'machine', and even 'system' to refer ambiguously to both the (apparently) complete UNIX systems with which users interact directly, and to specialised servers which are always accessed remotely.

We start from the premiss that all these component systems are untrustworthy and that the security of the overall system may make no assumptions about their behaviour - except that the LAN provides their only means of inter-communication. The consequence of not trusting the individual systems is that the unit of protection must be these systems themselves: we will dedicate each one to a fixed security classification. Thus, we could allocate three systems to the SECRET level, two more to the CONFIDENTIAL level, and the rest to UNCLASSIFIED use. Need-to-know controls can be provided by dedicating individual machines to different compartments within a single security level: thus one of the SECRET systems could be dedicated to the ATOMIC compartment and another to NATO. In a commercial environment, some systems could be dedicated to FINANCE and others to PERSONNEL and to MANAGEMENT. Users are assigned to hosts with due regard to the fact that no security is guaranteed within those individual systems.* Notice that since the hosts

* Although we cannot guarantee the security of individual UNIX systems, penetration audits do show this system to be quite resistant to attack, provided it is set up and administered carefully [Bruce82, Ritchie79]. Also, its file protection facilities, while they cannot handle the hierarchical aspects of multilevel security, can provide compartmented access controls through use of the 'group' permissions. Thus it may be reasonable to allocate separate machines to each 'major' compartment

are not trusted, they cannot be relied upon to authenticate their users correctly. Thus, access to each system must be controlled by physical or other external mechanisms.

Although there is no security within an individual system, the key to our proposal is to enforce security on the communication of information between systems. To this end, we place a trustworthy mediation device between each system and its network connection. We will call these devices 'Trustworthy Network Interface Units', or 'TNIUs' for short. They are the 'reference monitors' in this design and must satisfy the usual requirements for such monitors:

- It must be impossible to bypass them. This must be guaranteed by the physical interconnections of each host, TNIU, and LAN station.
- They must be tamper-proof. This requires, as a minimum, that TNIUs and their inter-connections be physically protected.
- They must be correct.

The initial purpose of TNIUs is very restrictive: it is to permit communication only between machines belonging to the same security partition (machines are in the same partition if they have the same security level and belong to the same compartment). The single UNIX UNITED system is therefore divided into a number of disjoint subsystems. Of course, this could be achieved more easily with physically separate systems, each dedicated to a single partition. However, we will describe later how our system can be extended to move information across partitions securely - thereby providing true multilevel security. Before this can be done, though, it is necessary to establish the mechanisms whereby TNIUs can securely separate communications across the LAN.

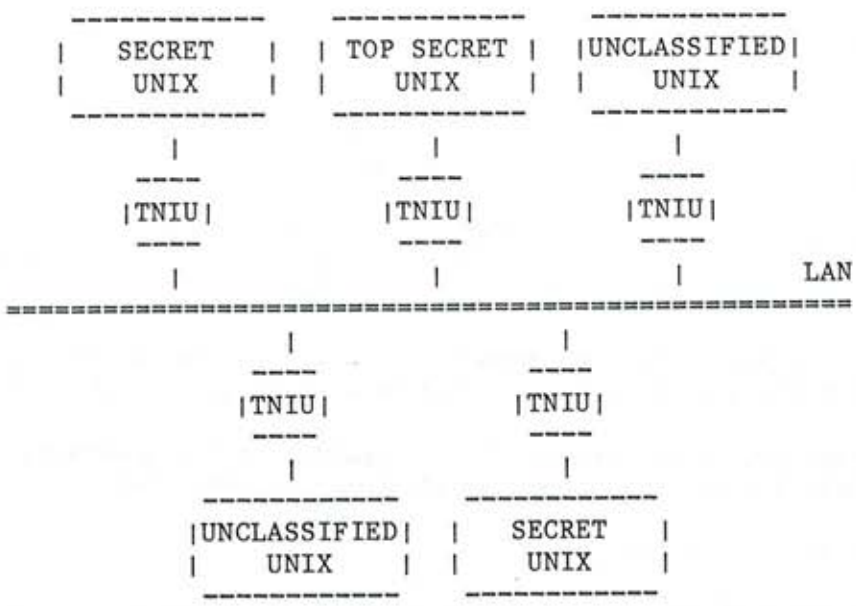
We expect our Distributed Secure System to be used in a variety of threat environments. Some installations may need to counter the most extreme and recondite forms of attack, others may face less determined adversaries. Wherever possible, therefore, we describe a spectrum of possible implementations for each of our security mechanisms. All are intended to be secure in the face of direct attack but differ in the extent to which they inhibit clandestine communications channels and, in the present case of TNIUs, in their vulnerability to wire-tapping threats.

The simplest form of TNIU merely monitors the source and/or destination fields of each incoming or outgoing LAN packet, passing only those which came from, or are going to, systems in the same security partition as their own. Monitoring both the source and destination fields is redundant if all machines are equipped with a TNIU - but it does permit the economical option of interfacing machines of one security partition (presumably UNCLASSIFIED) to the LAN directly.

Simple TNIUs of this type could be built directly in discrete logic and would be small, cheap, and fast - and their correctness could be established by informal design review and exhaustive testing. Each such

---

while relying on UNIX's internal mechanisms to provide discretionary access control on 'minor' sub-compartments.



(TNIU = Trustworthy Network Interface Unit)

A Securely Partitioned Distributed System

TNIU must contain a list of the LAN station numbers of the machines in its own security partition. This information could be built into the TNIUs when they are constructed, determined by a plug-in ROM, or inserted manually on switches or thumb-wheels.

Even such simple TNIUs do prevent machines belonging to different security partitions from communicating directly across the LAN and they ensure that the security of this partitioning can only be breached through attacks against the LAN itself (such as a wire-tap or subversion of a LAN station). If such attacks are mounted against the LAN, however, then these simple TNIUs provide no defence at all - additional or alternative mechanisms will be necessary.

Wire-tapping threats can be neutralised by locating the LAN in a physically secure environment or, for certain LAN technologies, by the use of link encryption or of pressurised or fibre-optic cables. These techniques are impracticable in many circumstances, however, and (with the possible exception of fibre-optics) are very expensive. Also, none of them gives any protection against subverted LAN stations. Where the possibility of attacks against the LAN is considered significant (and in many environments it is not), the use of enhanced TNIUs which perform end-to-end encryption offers an economical and attractive countermeasure. All data leaving a host machine will be encrypted by its attached TNIU, will travel the LAN in encrypted form, and will only be decrypted in the TNIU attached to its intended recipient.

The simplest method of encryption uses a single key for all communications within each security partition and operates at the LAN packet level. In this case, the TNIUs merely encrypt the data portions of

outgoing packets and decrypt those of incoming ones. (Control bits and the source and destination fields must, of course, be left in the clear so that the LAN hardware can interpret them correctly.) The advantage of this encryption scheme is that it is transparent to the host machines and can be managed by very simple TNIUs. Its disadvantages are that without an additional TNIU to TNIU protocol layer it may not be possible to prevent packets being delivered to a machine in a different security partition than their sender, and only very simple modes of encryption of the 'Electronic Code Book' (ECB) variety [Denning82, Price81] are likely to be feasible. Even very strong cryptographic algorithms such as the Data Encryption Standard (DES) [NBS77] may not prevent information leaking from a corrupt machine under these circumstances.

To see this, suppose malicious software in a SECRET machine desires to send the bit pattern 01101 to a machine at the UNCLASSIFIED level. The SECRET machine constructs a message XYYXY where X and Y are arbitrary but distinct bit patterns of the same size as the unit of block encryption (presumably, 64 bits) and then sends this message to its UNCLASSIFIED collaborator. (If TNIUs check the source and destination fields then the network must have been subverted for this to be possible - but the main reason for using encrypting TNIUs in the first place is to overcome network subversion.) The SECRET TNIU will encrypt the outgoing message with the SECRET key to yield, say, PQQPQ and this will be transmitted to the receiving UNCLASSIFIED TNIU which will 'decrypt' it with the UNCLASSIFIED key to yield, say, RSSRS. The UNCLASSIFIED machine now has a message which is quite different from the one originally transmitted, but in which the SECRET bit-pattern 01101 is plainly visible. Similarly, a wire-tapper would be able to see the same pattern in the encrypted message PQQPQ.

It should be realised that the threat here is not due to any weakness in the encryption algorithm employed, but to the way in which it is used: it is not necessary to be able to decrypt messages in order to extract information planted by a corrupt machine. The situation faced here is rather different to those in which encryption is ordinarily employed - where the end parties are assumed to be in legitimate communication and to have an interest in preserving the secrecy of their communication.

Clandestine communications channels based on cleartext patterns which persist into the ciphertext can be thwarted by the use of more elaborate modes of encryption such as 'Cipher Block Chaining' (CBC) which mask such patterns by causing the encrypted value of each block to be a complex function of all previous blocks [Denning82, Price81]. Encryption of this type is best performed over larger units than individual LAN packets and therefore needs to be invoked higher up the protocol layering hierarchy than the LAN packet level.* Measures to defeat active wire-tapping threats such as the modification of data in transit and 'spoofing' (the recording and replaying of genuine traffic) are also

---

* The LAN with which we are most familiar, and which was used for the original UNIX UNITED implementation, is the Cambridge Ring. This employs a 'minipacket' containing only 16 bits of user information. Some of the details, though not the general issues, concerning encryption management will differ in the case of LANs which provide larger packet sizes.

best applied at a fairly high level. Kent [Kent77] provides a good discussion of these issues.

Contemplation of the threats to which they are exposed, and of the necessary countermeasures, has convinced us that encrypting TNIUs really do need to be rather sophisticated and cannot operate at the very bottom of the protocol layering hierarchy. In fact, we propose that they should take over all but the highest level protocol functions and intend to use the Remote Procedure Call (RPC) protocol of the Newcastle Connection as the interface between host machines and their TNIUs. This approach has the additional benefit that the TNIUs act as 'network front-ends', relieving their hosts of the low-level network load and thereby boosting overall performance. We accept that TNIUs of this complexity present a considerable challenge in both construction and verification, but we argue that they are very similar in concept and function to the cryptographic front-ends of wide-area networks, examples of which have already been built and, in some cases, verified [Barnes80, Barnes81, Golber81, Grossman82].

The issue of assigning functions to layers in a protocol hierarchy can become quite complex in the presence of encryption. Fortunately, the protocol layering hierarchy used in UNIX UNITED is relatively simple and the integration of an encryption function poses few difficulties. The RPC protocol of the Newcastle Connection merely requires a 'fairly reliable' datagram service from the lower protocol layers [Panzieri82, Shrivastava82] and most RPCs and their results are encoded into a single datagram. Those concerned with file reads and writes, however, (which may transfer arbitrarily large amounts of data) are broken into as many separate datagrams as necessary by a sub-protocol of the RPC mechanism.

Individual datagrams will form the message units that are encrypted and protected by the TNIUs. Each datagram will be encrypted or decrypted in CBC mode using a key determined by the security partition to which the TNIU belongs. In order to prevent datagrams with a common prefix from yielding similar ciphertext, the TNIUs must prepend some unique material to the front of each one prior to encryption. This material would most usefully be a time-stamp or sequence number - since these are needed anyway in order to defeat spoofs (by enabling 'old' datagrams to be detected and discarded). Synchronising the sequence numbers or time-stamps used between each pair of TNIUs requires a special TNIU to TNIU protocol. This protocol must be resistant to spoofs, but obviously cannot itself use sequence numbers or time-stamps for this purpose. A challenge-response technique first proposed by Needham and Schroeder [Needham78] can be used instead.

We regard the sequencing or time-stamping of datagrams by TNIUs to be solely a countermeasure against spoofing. It is not the responsibility of TNIUs to associate RPCs with their replies, nor to ensure that the individual datagrams constituting long (file reading and writing) RPCs are correctly ordered. These issues are properly the responsibility of the RPC protocol managed by the hosts themselves. Notice, however, that the encryption performed within TNIUs will protect any sequencing information required and supplied by the RPC protocol layer.

In order to defeat an active wire-tapper who modifies encrypted datagrams, or who splices parts of different datagrams together, a checksum should be appended to each datagram prior to encryption. TNIUs

will then be able to detect receipt of damaged datagrams and will discard them. Since LANs are not error-free and can damage datagrams innocently as well as maliciously, it may also be desirable for cleartext checksums to be appended to encrypted datagrams at a lower level of the protocol hierarchy. Errors detected at this level may be assumed to be 'innocent', while those that pass through and are detected by the encrypted checksum must be assumed to be the work of an intelligent wire-tapper and should raise a security alarm.

Through the careful use of encryption, time-stamps, and checksums, we have made it impossible for machines in different security partitions to communicate with each other (even if the LAN is subverted) and have prevented wire-tappers from extracting information from message contents and from modifying or re-using genuine messages or constructing counterfeits. Significant channels for information disclosure still remain, however. They are 'pattern of use' channels whereby a malicious host modulates the visible parameters of legitimate messages in a way that can be decoded by a wire-tapping accomplice. The properties that can be modulated are the lengths of individual datagrams, their time and frequency of transmission, and their destination (presumably the source is fixed at the location of the malicious host). These properties, of which length and destination are much the most important, can be modulated to yield clandestine communication channels of surprisingly high bandwidth [Padlipsky78]. Since each datagram must, inescapably, have a length and a destination that can be observed by wire-tappers, it is not possible to sever these channels completely; the best that can be done is to reduce their bandwidth to a tolerable level, either directly, or through the introduction of noise.

We have already reduced the bandwidth of these channels considerably by using the high level remote procedure call protocol as the interface between machines and their TNIUs. Malicious machines cannot modulate the destinations of individual packets (a very high bandwidth channel - up to half that of the 'official' channel in the case of the Cambridge Ring) but only those of complete datagrams. Similarly their control over the length of datagrams is imprecise because protocol material is added by the TNIUs. We propose to lower the bandwidth of these channels still further.

The length channel is the easiest to deal with. We propose to use datagrams of a fairly large, fixed size - say 1024 bytes. Long RPCs and replies will be broken into a number of separate datagrams by the host and short ones (and the residue of long ones) will be padded by the TNIU in order to fill a whole datagram. (If this technique causes great numbers of largely empty datagrams to be generated, then some of the legitimate bandwidth of the LAN will be wasted - but this is not usually a scarce resource, and some tuning of the choice of datagram size is possible in any case.) A wire-tapper will not then be able to observe the exact length of an RPC or its reply, only the number of datagrams that it requires. Even this information will be difficult to extract and the corrupt host will also have to modulate some other parameter (e.g. destination) in order for the wire-tapper to identify the datagrams belonging to each RPC.

The bandwidth of the channel that modulates message destinations can only be reduced by introducing 'noise' - complicating traffic patterns so that the wire-tapper finds it hard to detect and extract any

deliberate modulation. The obvious way to do this is for each TNIU to generate a steady stream of spurious datagrams to all the other TNIUs in its own security partition. Spurious datagrams are marked as such (under encryption, of course) and are discarded by TNIUs which receive them. More refined strategies (such as routing datagrams indirectly through a number of intermediate TNIUs before delivering them to their final destination) are clearly possible, but all techniques for introducing noise inevitably reduce the bandwidth available for legitimate communications (and may increase the latency of message delivery). Each installation will have to decide its priorities in the trade-off.

With TNIUs which employ all the countermeasures we have described, the channels available for the disclosure of information have, we believe, all been either closed down or made to yield only a low bandwidth while requiring penetration at multiple points. (A distributed Trojan Horse!) The remaining threats are those of 'denial of service' caused by the destruction of genuine LAN traffic, or the injection of large quantities of garbage. Although they can do nothing to prevent or defeat such attacks, it is a correctness requirement of TNIUs that they should continue to provide a reliable (though necessarily degraded) service in their presence. It is also a correctness requirement that they should be immune to attacks on their I/O interfaces and that they should recover from crashes safely. (Of course, verified software does not crash, but we must allow for the possibility of a power failure.)

Trustworthy Network Interface Units not only need to be correct and secure but, in order to be practical, they also need to be small, cheap and fast. This can be achieved by basing them on modern 16-bit microprocessors, and by using DES encryption. Single-chip implementations of the DES algorithm capable of operating in CBC mode at LAN speeds are already available. We intend to use a separation kernel to enforce cleartext/ciphertext (so-called 'red/black') separation, with the basic physical protection provided by the memory management (MMU) chips appropriate to the chosen processor. Since no disks are needed (the software can be held on ROM), the complete unit should fit on a single board and cost only a few hundred pounds.

Key Distribution

Any system which uses encryption must contain mechanisms for generating and distributing keys securely. These mechanisms are the Achilles heel of many proposals that depend on cryptographic techniques. However, and unlike connection-oriented (virtual circuit) schemes in which it is necessary to manufacture and distribute a unique key every time a new circuit is opened up, our system imposes no requirement for frequent or rapid key distribution: the key allocated to a TNIU is a function of the security partition to which its host belongs and this is assumed to be relatively static. This infrequency of security level change, combined with the fact that a LAN-based system is presumed to be geographically compact, makes manual key distribution perfectly viable.

As a logical minimum, it is only necessary to to distribute encryption keys when the system is first installed and whenever a host changes level, or a new one is introduced (both actions involving manual intervention in any case). Note that since TNIUs guarantee the integrity and legitimacy of all communications on the LAN, knowledge of the encryption key used by a security partition confers both the right and the

immediate ability to communicate with all other machines in that partition: there is not (and in the interests of simplicity and survivability there should not be) a requirement to 'register' the level of a TNIU with any central mechanism, nor with other TNIUs. Membership of a security partition is identified with possession of its encryption key. This means that only one piece of information - the key - needs to be distributed to each TNIU; there are no distributed partition-membership tables that need to kept consistent with each other.

Because cryptosystems can be broken or their keys revealed (either through capture of system components, or through compromise outside the computer system), it may not be considered desirable to use the same set of encryption keys over the entire lifetime of the system. Encryption keys are generally considered to have a finite life and to be in need of replacement once they have encrypted a certain volume of traffic (since the threat of a successful cryptanalytic attack increases with the amount of traffic available for analysis, as does the amount of information that would be compromised) or have been in use for a certain length of time (since the threat of betrayal outside the system increases with time). We therefore propose that a 'travelling key man' [Price81] should periodically install new keys into the TNIUs. Techniques for the generation and installation of cryptographic keys have been described by Matyas and Meyer [Matyas78].

If the fear of cryptanalysis causes key changes to be desired more frequently than it is convenient for the key man to call, then either he should deposit a set of keys at each visit, or else a single master key from which the TNIU can manufacture a whole set of keys [Ersham78]. When encryption keys change, it is important that all the TNIUs within each security partition perform the change at the same time. This poses an interesting problem in distributed algorithm design which we leave as a challenge to the reader.

Summary

In this first stage of our design for a Distributed Secure System, resources are partitioned by security level: information may not flow from one partition to another. This is achieved by allocating physically separate computer systems to each security partition. These untrusted component systems are connected to a local-area network via Trustworthy Network Interface Units (TNIUs) which ensure that only systems belonging to the same security partition are able to communicate.

TNIUs use encryption to enforce this separation: each security partition has its own, unique encryption key which is manually distributed to all and only the TNIUs belonging to that partition. The interface between the untrusted host machines and their TNIUs is the remote procedure call protocol of the Newcastle Connection. This high-level interface permits the use of encryption-based countermeasures to defeat the threats of message disclosure, modification, replay, and misrouting that may be posed by any combination of tapped or subverted network components and corrupted host machines. Also included are countermeasures which reduce the bandwidth available for clandestine communications based on modulated traffic patterns.

TNIUs of this sophistication are not simple, but their design and verification may be based on established techniques employed for the

'secure front-ends' of wide-area networks. Modern 16-bit microprocessors and DES encryption chips provide suitable hardware for the construction of TNIUs and should enable them to be manufactured quite cheaply.

4. A Multilevel Secure File Store

The design introduced so far imposes a very restrictive security policy: the security partitions are isolated from one another with no flow of information possible across different levels or compartments. We now show how to extend this design to permit information to cross security partitions in a controlled 'multilevel secure' (MLS) manner. This will allow, for example, information to flow from the SECRET to the TOP SECRET levels, but not vice-versa.

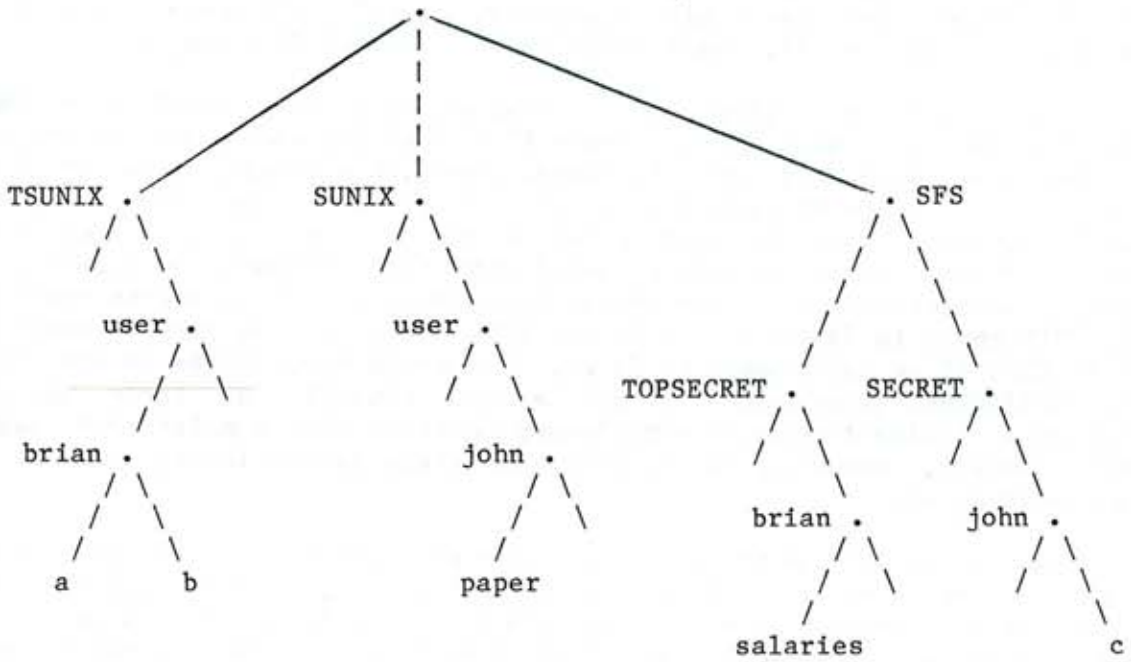
It might seem that multilevel secure information flow can be provided by simply modifying the policy enforced at the TNIUs so that, for example, TOP SECRET machines are able to receive communications from SECRET machines as well as TOP SECRET ones. TOP SECRET TNIUs would be provided with the SECRET as well as the TOP SECRET encryption keys and would permit incoming but not outgoing communications with SECRET level machines. The flaw in this scheme is that the communication cannot be truly one-way: a SECRET machine cannot reliably send information to a TOP SECRET one without first obtaining confirmation that the TOP SECRET machine is able to accept it and, later, that it has received it correctly. Thus the SECRET machine must be able to receive information from the TOP SECRET machine as well as send to it - and this conflicts with the multilevel security policy.

Certainly the trustworthy TNIUs in the system could be enhanced to undertake reliable delivery of data across security partitions, but this misses the point: it is end-to-end acknowledgement of the receipt and processing of information that is required, not the reliable delivery of uninterpreted bitstrings [Saltzer81]. Only the receiving host will understand the semantics of the communication and it alone will be able to confirm whether it has been able to process it correctly. If TNIUs are able to provide the acknowledgements, then they must be capable of processing the information concerned - in which case they are no longer simple communications processors but are taking on the character of secure hosts. Since the whole motivation for our design is the conviction that secure hosts are beyond the state of the art, we discard this approach. Notice, too, that it would only provide for unsolicited communications in any case: a SECRET machine could send information to a TOP SECRET machine of its own volition, but the TOP SECRET machine could not request that the information be sent - since the mere fact of its request would constitute an insecure flow.

The only secure way for information to flow across security boundaries is via a trustworthy intermediary that acts as a staging post. The complexity of such an intermediary will depend on the generality of the services which it provides. For simplicity, combined with the most useful functionality, we select files as the only objects that will be allowed to cross security boundaries and we choose the multilevel secure storage and retrieval of files as the service to be provided by the trustworthy intermediary. We do this by adding a (Multilevel) Secure File Store to the system with the ability to communicate with machines of all security classifications. The idea is that when a SECRET level machine wishes to make one of its files available to higher levels, it

'publishes' it by sending it to the Secure File Store. A TOP SECRET machine may then subsequently request a copy of this file from the Secure File Store.

Before describing the mechanism of the Secure File Store, we need to outline its logical position and role within the overall UNIX UNITED system. Conceptually, the Secure File Store is just an ordinary UNIX system that returns exceptions to all system calls except those concerned with files. As with any other component, it will be associated with a directory, say 'SFS', in the UNIX UNITED directory structure. The SFS directory will contain sub-directories for each security partition in the overall system and these will in turn contain further sub-directories of arbitrary structure (created by ordinary users). A simple UNIX UNITED directory structure containing just the Secure File Store and two ordinary hosts is shown below.



The ordinary hosts are associated with the directories 'TSUNIX' and 'SUNIX' and are allocated to the TOP SECRET and SECRET security partitions respectively. Of course, from within SUNIX, the TSUNIX branch of the directory tree is invisible (and vice-versa). Even if the Newcastle Connections within TSUNIX and SUNIX are aware of each others' existence, any attempted inter-communication will be stopped by their TNIUs.

If the SECRET level user 'john' of SUNIX wishes to make his 'paper' file available to the TOP SECRET user 'brian', he does so by simply copying it into a directory which is subordinate to the SFS directory. For example:

```
cp paper /../SFS/SECRET/john/paper
```

This will cause the Secure File Store machine to receive a series of remote procedure calls from SUNIX, requesting it to create and write a file called 'paper' located as a sibling of the file 'c'. The Secure

File Store will be assured by its TNIU that these requests do indeed come from a machine in the SECRET security partition and it can then consult its record of the security policy in order to determine whether such a machine is allowed to create SECRET level files. Since we may assume that it is, the requested file operations will be allowed to proceed and the copy of the file will be created. Similarly, when the TOP SECRET user 'brian' attempts to typeset the paper by issuing the command

```
nroff /../SFS/SECRET/john/paper
```

the Secure File Store will receive a series of read requests from the machine TSUNIX. Once again, it can apply the security policy and see that these requests may be allowed to proceed. The Secure File Store would, however, refuse requests from TSUNIX to write into this 'paper' file, or to delete it, since these contravene one of the requirements of multilevel security [Bell76]. Similarly, 'john' would not be allowed to read the 'salaries' file held under the TOP SECRET directory.

Notice how naturally these services are integrated into UNIX UNITED. They are invoked by ordinary UNIX commands and system calls, and differ from those that could be provided within a single standard UNIX only in that the hierarchical nature of the multilevel security policy is incorporated into the file access controls, and in the fact that those access controls are enforced with total rigour. In passing, it should be mentioned that the standard discretionary file access controls of UNIX apply to files in the Secure File Store just as they do normally - so that if he had chosen to do so, john could deny access to the file /../SFS/SECRET/john/paper to all except himself, or those in his 'group'. Unlike the non-discretionary controls of the multilevel security policy, however, the operation of these discretionary controls is not guaranteed.

Having described the services which the Secure File Store is to provide, we must now explain how it will be constructed. The services required are those of a multilevel secure UNIX file system and may seem to demand a substantial quantity of provably trustworthy mechanism - virtually a secure UNIX. With careful design, however, we can reduce the amount of trusted mechanism quite considerably. We begin by partitioning the Secure File Store into two components housed in physically separate machines. The first, called the 'Secure File Manager' (SFM), will be a small, trustworthy component concerned with enforcement of the security policy, while the second, called the 'Isolated File System' (IFS), will be a much larger, untrusted component whose task is to provide the actual file system. The SFS directory in the UNIX UNITED name structure will be identified with the machine that houses the SFM, but the entire UNIX file subsystem subordinate to the SFS directory will be held separately in the IFS. The IFS must therefore be a machine capable of maintaining a UNIX file system. The simple way to achieve this is to use a perfectly standard UNIX system for the IFS - and this is what we propose to do.

Because the IFS will contain files of all security classifications, and because it is untrusted, and possibly untrustworthy, it must obviously have no direct communications with the outside world; all its communications must be mediated by the SFM. Consequently, the IFS must either be connected directly to the SFM, or it must be attached to the

LAN via a Trustworthy Network Interface Unit which is provided with a special encryption key known only to itself and the TNIU of the SFM. The TNIU of the SFM must be a specially enhanced one which contains the encryption keys of all security partitions (and also the special key used for communications with the IFS if this is connected to the LAN). This is necessary so that machines in all security partitions can communicate with the SFM. (An alternative is for a second encryption key to be provided to all TNIUs for use when communicating with the SFM.) The internal structure of a TNIU with multiple encryption keys will be slightly more complex than one with just a single key, particularly if communications using different keys can be in progress simultaneously. Cleartext belonging to logically separate channels should be managed by separate regimes, and temporal separation must be provided for different uses of its single DES chip. These are not significant complications, however, and the responsibility for correctly managing more than one encryption key is a small additional burden to place on the trusted mechanism of a TNIU.

An over-simplified outline of the operation of the SFM/IFS combination is the following. Remote procedure calls requesting operations on 'secure' files will arrive at the SFM, where they will be checked against the security policy. If they are acceptable, they will simply be passed to the IFS - which will perform the action requested and pass the results back to the SFM for return to the original caller. The over-simplification in this account is that it has ignored the fact that the IFS cannot be trusted: the SFM must contain mechanisms for guaranteeing its 'good behaviour'.

The IFS has access to files of all security classifications and therefore has enormous opportunities for compromising security. Although its isolation prevents it from revealing the contents of files directly, it can easily do so indirectly by copying the contents of a TOP SECRET file, say, into an UNCLASSIFIED one - which can later be retrieved by UNCLASSIFIED users. Superficially, encryption might appear to offer a solution to this problem. If the SFM encrypted files before consigning them to the IFS, using different keys for each security level, then the previous threat would seem to have been thwarted: though the TOP SECRET file might still be copied into an UNCLASSIFIED one, its contents would be incomprehensible to the UNCLASSIFIED user who eventually acquired it, since they would have been encrypted using the TOP SECRET key and 'decrypted' using the UNCLASSIFIED key. Obviously, for the reasons explained in the previous section, the encryption technique used must be one, such as CBC, which masks cleartext patterns. This scheme is easily defeated, however.

Suppose it is the CBC mode of encryption that is employed. This mode of encryption has the useful property that it is 'self-healing': if a ciphertext block is corrupted, only the text corresponding to that block and the one following will decrypt incorrectly [Denning82, Konheim81, Price81]. The IFS can therefore communicate information to an UNCLASSIFIED user by selectively modifying one of his encrypted files. The mechanism is as follows. The IFS chooses to corrupt or not corrupt every fourth (say) block of an UNCLASSIFIED file. The decision whether to corrupt a particular block or not is determined by the TOP SECRET bit pattern to be communicated: 1 = corrupt, 0 = do not corrupt. The UNCLASSIFIED user who eventually recovers this file then simply examines every fourth block to see whether it has been changed during

its sojourn in the IFS - and can thereby extract the TOP SECRET bit pattern. The bandwidth of this channel can be increased by exploiting a further property of the CBC mode of encryption: although a corrupted ciphertext block decrypts unpredictably, the following block decrypts in a way that yields considerable information about the exact form of the corruption [Price81]. Even if the stream or block cipher used does not have the self-healing property (as when cleartext feedback is used in addition to ciphertext), it is still possible to encode several bits per file in the position where corruption starts.

This scenario naturally invites the question, "how did the IFS obtain TOP SECRET information in the first place if all files are encrypted?". This is easily answered. Suppose, for example, that a corrupt TOP SECRET host has 26 files, all of different lengths, stored in the IFS. If it then retrieves the 6 files whose lengths are the 18th, 21st, 19th, 8th, 2nd, and 25th shortest, in that order, the TOP SECRET string RUSHBY will have been communicated to the IFS. It is very important to appreciate that this channel for clandestine information flow would still be present even if there were a separate IFS for each security classification: if the 26 files were UNCLASSIFIED ones, held in an IFS dedicated to the UNCLASSIFIED security partition, then 'read' requests from a TOP SECRET machine would be perfectly legal and could still be modulated to convey TOP SECRET information into the UNCLASSIFIED IFS.

Since encryption and single level IFSs, and even a combination of the two, are unable to prevent clandestine information flow through the IFS mechanism, we appear to be in a difficult position. Perhaps the IFS has to be trustworthy after all.

In fact it does not and the analysis of the issue is quite instructive. In order for there to be a path for clandestine information to flow through the IFS, there must be a path into the IFS and another back out again. By virtue of the very function of the IFS, incoming information flow cannot be prevented: it is an essential property of the IFS that it operates on files of one security classification in response to requests from another, and there is no way to prevent a corrupt host from encoding information in the pattern of its requests. (This is an example of what Lampson [Lampson73] calls a 'legitimate channel' for information leakage.)

If we cannot prevent information coming into the IFS, perhaps we can prevent it escaping back out again. Since the only objects which leave the IFS are the files which it retrieves in response to external requests, any clandestine information which is to reach the outside world must be encoded into those files by the IFS. But all movement of files into and out of the IFS is mediated by the SFM - so security will be maintained if the SFM can prevent the IFS from encoding information into (i.e. modifying) outgoing files. In other words, and as first noted by Gligor and Lindsay [Gligor79], security depends upon the SFM being able to guarantee the integrity of files stored in the IFS.

This can be achieved by the use of a checksum added to each file by the SFM before it is stored in the IFS. Any attempt by the IFS to modify a file will be detected on its subsequent retrieval by the SFM when the recomputed checksum fails to match the one stored with the file. Of course, this only works so long as the IFS is unable to forge

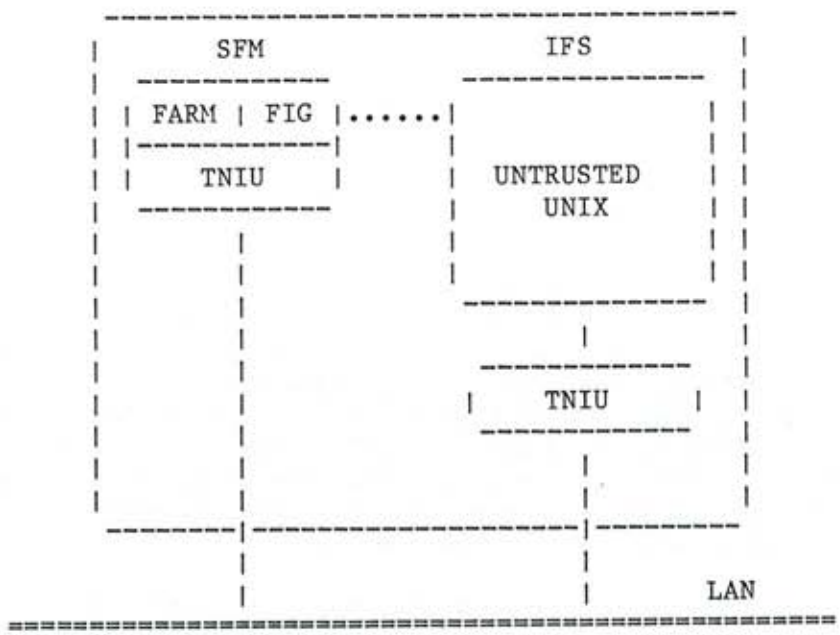
the checksums. There are two ways for making sure of this (other than by keeping the checksums in the SFM). The first is to use a conventional checksum (i.e. one computed by an algorithm that may be known to the IFS) but to protect it by encrypting the file and the checksum as a single unit. The second technique is to use a 'crypto-checksum'. This is one that depends upon a secret encryption key for its computation; an example is the final block of ciphertext produced at the end of CBC mode encryption. The advantage of crypto-checksums is that they are unforgeable by anyone who does not possess the encryption key and can therefore be used with information that is stored in the clear. This can be useful when information is retrieved by value, as it is in database applications, but is irrelevant in the case of a file system where retrieval is by name. Consequently, we propose to use conventional checksums protected by encryption. The use of encryption has the additional benefit that it renders most of the information stored in the IFS 'black', and may therefore simplify the task of handling and protecting its physical storage media. (Note, however, that the auxiliary information which UNIX associates with each file - e.g. file length, owner, date of creation, etc. - will be stored in the clear.)

The SFM as now described is required to perform two security-critical tasks and is therefore split into two logically separate components: the 'File Access Reference Monitor' (FARM) and the 'File Integrity Guarantor' (FIG). The task of the FARM is to ensure that all file access requests comply with the security policy; the FIG is responsible for computing the checksums and performing the encryption and decryption of files sent to, or received from, the IFS.

When a file is to be sent to the IFS, the FIG prepends some 'housekeeping' information to the front of the file, a checksum to the rear, and encrypts the whole lot using the DES algorithm in CBC mode. Intermediate checksums may be included at intervals within the file if the FIG has only limited buffer space. Should part of a file have already been delivered to a host when corruption of a later part is detected by the FIG, then some clandestine information may have been conveyed to that host through the position at which the corruption began and file transfer was aborted by the FIG. This channel has very limited bandwidth and, provided all checksum failures raise a security alarm and are logged by the SFM, it is not considered to constitute a serious security flaw.

Despite its attractive simplicity, there is a glaring difficulty with the FIG mechanism: it only allows files to be read and written in their entirety - whereas the UNIX file interface, which the Secure File Store is required to support, provides for incremental reading and writing and the repositioning of the file 'pointer'. It appears that although 'passive' files can be held in the IFS, 'active' files (those which are the target of current operations) must be held in the trustworthy SFM. This would probably require the SFM to have its own disks and to contain all the trusted mechanism necessary to manage them and to perform UNIX file operations securely. This complexity of trusted mechanism is precisely what we have been seeking to avoid. Fortunately, we can do without it.

All that is required is a real UNIX system in the right security partition to hold and manipulate each active file on behalf of the SFM. And it is easy to find such a system: the one which requested the file



SFM = Secure File Manager
IFS = Isolated File System
FARM = File Access Reference Monitor
FIG = File Integrity Guarantor
... = Logical channel between FIG and IFS
(physical channel is via TNIUs and LAN)

A Multilevel Secure File System

operations in the first place must, if it is allowed to have them performed at all, also be in the right partition to perform them itself.

The procedure for performing 'read' operations on a secure file then becomes the following (writing is similar). When a remote procedure call to 'open' the file for reading arrives at the SFM, its FARM will first check that the request complies with the security policy. If it does, the FIG will recover the file, in its entirety, from the IFS and, while checking it for integrity, write it into a file in the requesting host machine. (A special directory may be reserved in each host for this purpose.) All subsequent 'read' and 'seek' remote procedure calls can then simply be modified by the SFM so that they refer to the copy of the file already held by the host and then be passed straight back to the Newcastle Connection in that host. Obviously, optimisation is desirable, and this can be accomplished by extending the Newcastle Connection software so that it can be made aware the presence of a 'cached' copy of a remote file within its own machine. This extension is, in fact, already planned to occur so that file systems maintained by distinctly non-UNIX hosts can be accommodated within UNIX UNITED.

It should be admitted that there are some unattractive aspects to this 'caching' technique. Most importantly, two users operating on what is, ostensibly, the same file will, in fact, be operating on their own private copies. Neither will see changes made by the other and there

will be a consistency problem to resolve when the copies are written back into the IFS. In most environments, it is unlikely that users will be inconvenienced by (or even be aware of) this problem, but it cannot be considered a desirable feature. Unfortunately, it admits no easy solutions. The obvious technique of disallowing multiple 'open's on files in the Secure File System provides a direct signaling channel: a TOP SECRET host can deny an UNCLASSIFIED host access to an UNCLASSIFIED file (and thereby convey one bit of information) by opening the file itself.

If the consistency problem caused by the implicit 'caching' approach is considered unacceptable, then one alternative is make the 'caching' process explicit and manifest to users. This involves a retreat from our original desire to present a uniform interface and requires that 'secure' files should only be accessed through special system calls which copy an entire file in or out of the Secure File System.

As well as a consistency problem at its interface, the mechanism described so far contains security weaknesses. Most obviously, the IFS can substitute one complete file body for another. We propose to counter this threat by using different encryption keys for each file. In order to avoid the need to store large numbers of encryption keys for long periods of time, the key for each file can be generated by a cryptographic transformation of its security level and name. (Note that this excludes the possibility of 'links' from one file to another.) Using this technique, the SFM need only maintain a single master key. Its method of operation is as follows.

Whenever a file operation is requested from the SFM, it will encrypt the name and security level of the file in CBC mode using the master key and will generate the file encryption key from the final (checksum) block of the ciphertext. It is possible to prevent files of the same name and security level from using the same key if the full pathname of the file is used in the key generation process - but this will require either a restricted interface to secure files (the current working directory cannot be in the Secure File System), or more complexity in the SFM.

The IFS still has the ability to save old copies of legitimate files and can therefore signal a few bits of information to an outside collaborator by choosing which version of a file it will return. If this channel is to be blocked (and it may not be worth it), then the SFM must record a sequence number in a 'houskeeping' section of each file and must maintain a master file containing the sequence numbers of the most recent copies of all files. This master file can be stored on the IFS and protected by just the same mechanism as an ordinary file. The IFS could still attempt a 'master spoof' by retaining an old copy of the master file and this attack can be blocked if the SFM keeps a record, in trusted stable storage, of the sequence number of the current master file.

A more significant channel for information leakage out of the IFS uses the contents of the 'i-nodes'. These contain supplementary information about a file (e.g. its owner, discretionary access permission bits, date of creation, etc.) and UNIX provides a system call ('stat') for reading this information and others for altering it. The initial

contents of an i-node are not all determined by information explicitly provided in the system call that creates the file, but are manufactured by the file system itself from information in the system tables. It is therefore not possible for the SFM to check on the authenticity of information returned by the IFS in response to a 'stat' call, and this provides the IFS with the opportunity to convey a couple of dozen bytes of unchecked information back to the caller. There are a number of possible measures that can be taken to counter this threat, though none of them is entirely satisfactory.

One possibility is for the SFM to refuse to respond to 'stat' calls. Though secure, this is a rather strong response to the threat and it means that a number of standard UNIX programs will fail to work on files held in the Secure File System. A better response would be for the SFM itself to construct replies to 'stat' calls using accurate information wherever this is known to the SFM (e.g. file length, time of creation) and arbitrary or fixed values wherever it is not (e.g. actual i-numbers). The amount of accurate i-node information maintained by the SFM will influence the quantity of trusted mechanism that it must contain: in order to maintain everything it would almost have to become a trusted UNIX file system and maintain a full system table. A compromise would be to allow the IFS to manufacture the initial i-node for each file (since this is the operation that requires a full UNIX), but for the SFM to monitor its contents thereafter. This could be done in the following way. As soon as a file is created in the IFS, the SFM would perform a 'stat' on it and save the result (in encrypted, checksummed form) back in the IFS. The SFM would interpret all subsequent system calls to change i-node information and would simulate their operation on its own copy of the i-node and check its result against that produced by the IFS.

Directories pose a similar threat to i-nodes since they also contain system-generated information (i-numbers) and can be read like ordinary files. We suggest that this danger can be countered using similar techniques to those proposed for i-nodes. Observe that it is not necessary to keep separate encrypted and checksummed copies of directories in order to detect modifications, since the SFM can keep a special file in each directory whose name is a crypto-checksum for the directory contents.

Because i-nodes and directories provide limited channels for information disclosure if they are made directly visible to outside callers, it is prudent that the channels for information leakage into the IFS should be reduced as much as possible. Accordingly, we recommend that file names, as well as file contents, should be encrypted before they are consigned to the IFS, and that file lengths should be padded to a multiple of some fairly large value (say 1024 bytes).

Because we have discussed the difficulties posed by our method for providing a Secure File System at some length, we may have obscured its basic merits. Essentially, the mechanisms we have proposed allow a trade-off between uniformity of interface, security, and quantity of trusted mechanism. If a special interface is constructed for the manipulation of secure files, then complete security can be provided with very little trusted mechanism. Alternatively, if the full UNIX file system interface is desired, then the trade-off reduces to one between security and the quantity of trusted mechanism. The SFM is a reference monitor

for an untrusted UNIX file system and the security which it provides is a function of the completeness of its mediation of that file system. Complete security requires complete mediation - which in turn requires that the trusted software in the SFM must simulate many of the characteristics of a full UNIX file system. We believe, however, that the techniques we have proposed make it possible for a relatively simple SFM to provide a partial mediation of the behaviour of the IFS which is of sufficient power that the channels for clandestine information flow through the IFS will yield only low bandwidth while requiring a massive Trojan Horse within the IFS.

The mechanisms proposed here protect against the threat of disclosure, but they are obviously vulnerable to denial of service attacks: the IFS can simply deny that it holds a certain file, or even claim to be completely broken. Though potentially serious, this threat is far less insidious than that of disclosure since it is always obvious when it is happening. Assuming that a standard UNIX of known pedigree is used for the IFS and that sensible back-up precautions are taken, we do not expect that malicious denial of service will be considered a serious threat in many environments. However, to guard against accidental failures, and perhaps reduce the threat of malicious ones, we suggest that it would be straightforward to employ multiple IFSs - either to replicate or to distribute vital records.*

Summary

We have described how the sophisticated functions of a multilevel secure UNIX file system can be provided by trusted mechanisms that are relatively small and uncomplicated. The file system is maintained by a perfectly standard, untrusted UNIX system whose potential for betrayal is thwarted by the fact that all its communications are mediated by a trustworthy Secure File Manager (SFM) housed in a separate machine. The SFM itself comprises two trusted sub-components: the File Integrity Guarantor (FIG) and the File Access Reference Monitor (FARM). The responsibility of the FIG is to prevent any undetected modification to files or other ('i-node') information held by the Isolated File System, while the FARM is responsible for enforcing the security policy concerning the reading and writing of files.

The FIG achieves its purpose by employing encryption and checksum techniques which are very similar to those used, for LAN messages, by the Trustworthy Network Interface Units. We therefore suggest that the FIG can be constructed by minor modifications and extensions to an ordinary TNIU. The FARM function of the SFM is also straightforward, requiring only the imposition of simple access control rules determined by a security policy. This function could be performed inside a separate regime provided by the separation kernel of the machine which supports the TNIU/SFM functions.

---

* As they are separate Logical Concerns, UNIX UNITED keeps issues concerning tolerance to hardware and software failure quite separate from those of distribution - as we have tried to keep both these issues separate from security. A prototype extension to UNIX UNITED which provides for the user-transparent replication of files and processors in order to mask hardware faults has already been demonstrated [Liu82], and could usefully be incorporated into the Secure File Store mechanism.

We therefore conclude that all the functions of a complete SFM can easily be integrated into the TNIU which connects it to the LAN. The development and verification costs of an integrated TNIU/SFM should be little more than those for a TNIU alone, and production costs should be about the same - just a few hundred pounds.

Because it is integrated into a UNIX UNITED environment, the special nature and construction of the Secure File System need not obtrude into the user's view of the system. In its most general (though also its most complex) form, the Secure File System can appear to users and their programs just like any other directory in the name structure of the UNIX UNITED file system. The single difference is that whereas files belonging to ordinary directories are only available within their own security partition, those held by the Secure File System are, subject to the security policy enforced by the FARM, available to many different partitions.

5. The Accessing and Allocation of Security Partitions

A system such as the present one in which terminals are attached to machines of fixed security level can be somewhat inconvenient to use. A SECRET level user can send mail to a TOP SECRET one via the Secure File System, but the recipient can only reply by leaving his TOP SECRET machine and logging in to one at the SECRET level or lower. We can avoid this inconvenience, and also make possible the provision of additional services, by connecting terminals to 'Trustworthy Terminal Interface Units' rather than to hosts directly. Moreover, we can then include provisions for dynamically changing the allocation of machines to security partitions.

5.1. Accessing Different Security Partitions

What we term a Trustworthy Terminal Interface Unit (TTIU) is basically a Trustworthy Network Interface Unit (TNIU) enhanced with some additional trusted functions. These comprise a terminal driver, some very limited Newcastle Connection software, and an authentication mechanism. These are all logically separate mechanisms and will each run in individual regimes provided by the separation kernel which supports the TTIU.

A TTIU in the 'idle' state simply ignores all characters reaching it from the LAN or from its terminal - until a special character sequence is typed at the keyboard. This will cause the TTIU to connect the terminal to its authentication mechanism, which will then interrogate the user in order to determine his identity.* Once the user has been authenticated, he can be asked for the security partition to which he wishes to be connected. If the requested partition is within his clearance and all other requirements of the security policy are satisfied (for example, a terminal located in a public place may not be

* Alternatively, badge readers or other recognition devices could be employed. Also, although we speak of the authenticator being located in the TTIU, it may be preferable for authentication and clearance checks to be centralised. In this case, the authenticator in the TTIU will merely act as a liaison between the user and a central authentication mechanism accessed over the LAN.

permitted a TOP SECRET connection, even if its user is authorised to that level), then the TTIU will load the encryption key of the partition concerned into its DES chip. The Newcastle Connection software in the TTIU will then be able to establish contact with its counterpart in a host machine belonging to the appropriate security partition and the user will thereafter interact with that remote machine exactly as if he were connected to it directly.

The Newcastle Connection component in the TTIU must be able to respond to remote procedure calls directed to it by the Newcastle Connection of the remote machine. The only calls that require a non-error response are those appropriate to terminals, namely 'read from the keyboard', 'write to the screen', and a couple more concerned with status information. Thus only a fraction of the full Newcastle Connection software is required for a TTIU and, just like the similar software in a conventional host, it need not be trusted.

The minimum special function required of the terminal driver component of a TTIU is the ability to recognise the character sequence which users type to indicate that they wish to communicate with the authenticator. If the terminal is an 'intelligent' one, then it may also be desirable for the driver to filter the character sequences sent to it for any invocation of 'dangerous' functions. (Even quite modest terminals possess capabilities that allow apparently innocuous messages to have serious side effects.) More ambitious services that could be provided by the terminal driver include a local scrolling buffer and the ability to drive a printer. (Hard copy must be marked with the security classification of the session in progress at the time it was produced.) Any information buffered within the TTIU or its terminal must, of course, be erased when the session terminates, at which point also the encryption key in the DES chip should be unloaded.

None of the additional trusted mechanisms which are required to upgrade a TNIU into a TTIU should present an undue challenge in either construction or verification. Nor, given that TNIUs are constructed on top of a separation kernel, should the presence of these additional mechanisms affect the construction or verification of the TNIU components themselves. In fact, the presence of a separation kernel makes it perfectly feasible to support multiple terminals, each with a separate set of TTIU and TNIU components, on a single processor. It should even be possible, without great complexity, to permit a single terminal to maintain simultaneous connections with two or more remote machines in different security partitions, each using a separate window on the screen.

5.2. Changing Security Partitions Dynamically

Trustworthy Terminal Interface Units enable users to connect to machines in different security partitions and therefore allow them to perform each of their activities at the most appropriate level within their clearance. However, if a security policy with a fine granularity of need-to-know compartmentation is supported, then the number of different security classifications may well exceed the number of physical hosts available. Even when the number of distinct security classifications is small, the demand for resources within each classification may vary with time. Furthermore, some users may possess personal workstations which they wish to use for all their activities at many different

security levels. In all these cases, some provision for reallocating host machines to different security partitions is needed.

With untrusted hosts, this can only be accomplished by 'temporal separation' which, in its simplest form, is 'periods processing'. This requires manual intervention to perform the exchange of all demountable storage and the re-initialisation of all fixed storage in order to remove every trace of information from the old security partition before the machine can be brought up again at its new level - either 'clean' or reloaded with the suspended state of some previous activation at that level.

Manual periods processing requires very rigid administrative controls and is slow and expensive to perform. We will therefore propose a mechanism for automating the process so that it becomes both rapid and secure. Readers familiar with the history of secure systems proposals will recognise ours as a reincarnation of the 'encapsulation' approach proposed nearly ten years ago [Bisbey74, Lipner74]. The idea was not put into practice at that time, although considerable design work was done by the System Development Corporation for an implementation called the 'Job Stream Separator' which was intended to control a Cray-1 system. We will refer to our implementation of the idea as an 'Automatic Partition Changer'. It will comprise a number of sub-mechanisms.

The first requirement is for a separation mechanism to ensure that, at all times, the host machine only has access to permanent storage (i.e. disks) in the same security partition as its own current activation. The simplest such mechanism would merely be a switch between separate banks of disks. This has the advantage of being manifestly secure and requires no change to the host operating system. Its disadvantage is cost: the switch might be cheap, but providing multiple banks of disks would certainly not be.

The next most direct realisation of the mechanism would be one which synthesised logically separate 'mini-disks' on shared physical disks, as in KVM/370 [Gold79]. Unlike that in KVM/370, however, this synthesis would have to be performed in a physically separate processor since the host machine is completely untrusted. These 'disk interface processors' would be rather complex and difficult to verify, since they would contain trustworthy disk drivers and would need to operate extremely rapidly.

A more attractive arrangement in the context of a distributed system is for host machines to be physically remote from their permanent storage, and to obtain it as a service over the LAN - mediated by their Trustworthy Network Interface Units in the usual way. There are several operational benefits to this approach. Most obviously, the small, expensive (in terms of cost per bit), and slow disks that are generally attached to personal and other small machines can be replaced by large, cheap and fast disks held at a central site under good environmental conditions. Maintenance and back-up procedures are also simpler to arrange and control with a centralised storage facility. Some distributed systems already provide the basic mechanisms needed to support this approach: the Cambridge System does so at a low level [Birrell80, Needham82], while UNIX UNITED can do so at the level of files. However, while it is perfectly feasible for a UNIX UNITED system to run with no local files, the disk storage needed by the UNIX kernel for swap space

presents difficulty - since swapping is an activity which occurs below the level of the Newcastle Connection interface.

Our proposal for a mechanism that is appropriate the present system is a modification of the last approach. Hosts will be configured without a local file system and all references to apparently local files will be intercepted by a 'Local File Relocation Process' in the Newcastle Connection. This will redirect them as remote procedure calls to a file system held in a remote machine that is permanently assigned as a file server to the security partition which the host currently occupies. For example, if the host is known as 'PW5' (Personal Workstation number 5) and is currently operating in the (SECRET, NATO) partition, then the remote procedure call sent out in response to a request for the local file /bin/shell might actually name the file /../SNSERVER/PW5/bin/shell, where 'SNSERVER' is the name of the machine that maintains the (SECRET, NATO) file system. This transformation is perfectly straightforward and does not need to be trusted - since an attempt to name a machine in the wrong security partition will be caught by the standard TNIU mechanisms (the local and remote machines will have incompatible encryption keys).

We conclude that this aspect of the Automatic Partition Changer's function is easily provided and now turn to the control of the swap device. The key to this problem is to recognise that although the swap device may be a disk, it is only used for transient, rather than permanent, data. Hosts may therefore be provided with a local disk for use as a swap device in a perfectly standard way, except that the contents of this disk must be erased whenever the host changes security partitions.* The host processor's local memory and registers must also be erased at this time.

The simplest way to accomplish this erasure, and the one which we propose to employ, is to cause the host processor to boot-load a trusted stand-alone 'purge' program from ROM. This program will systematically clear all storage available to the processor. If the number of distinct security partitions exceeds the number of physically dedicated file servers that can reasonably be provided, then the Secure File System may be used to provide logically separate file systems for each combination of workstation and security partition. In this case, the local disk of each workstation must contain a small file system for holding 'cached' copies of open files. Since this file system is only used for transient information, it may (and must) be purged and re-initialised during a security level change, just like all the other information on the local disk.

In outline, the complete scenario for automatically changing the security partition in which a host operates is the following. A user at a terminal attached to a TTIU is authenticated and asked for the security partition in which he wishes to work. If this partition is within his clearance, a signal will be sent to the TNIU of a vacant host

---

* Security procedures in certain high-threat environments forbid the re-use of magnetic recording media for information of different security levels, even after erasure. (It is held that magnetic media cannot be completely erased.) It may be possible to avoid these objections by using 'silicon disks' (i.e. semiconductor bulk RAM) or by placing an encryption device in the disk access channel.

machine instructing it to switch to the indicated security partition. This signal will be protected against forgery or spoofing by the standard encryption techniques employed between TNIUs. On receipt of the signal, the host's TNIU will load the encryption key appropriate to the new security partition, inform its Local File Relocation Process of the identity of that partition, and initiate the purging of its host machine. At this point UNIX can be re-booted on the host. The necessary copy of the operating system could either be obtained locally from a read-only floppy disk, say, or from a remote 'boot-server' accessed over the LAN [Needham82].

Summary

We have described two additional mechanisms which increase the flexibility and convenience of our Distributed Secure System. The first is a Trustworthy Terminal Interface Unit (TTIU) which enables users to connect to the security partition of their choice (subject to their clearance) without having to be physically present at the site of a host machine operating in that partition. The second mechanism is an Automatic Partition Changer which enables host machines to be assigned, rapidly and securely, to different security partitions. These facilities require new trusted mechanisms of a very modest character. A TTIU is basically no more than a TNIU enhanced with additional trustworthy software for performing user authentication, key loading, and terminal support.

For automatically changing the security partition in which a host machine operates, its TNIU must be able to purge all of the host's storage, and to change its own encryption key. One implementation of the purging mechanism simply requires the host to be caused to boot-load a trusted program from ROM. All the host's files must be held remotely and accessed through an untrusted redirection process in its TNIU.

These generalisations of the original, very inflexible, but highly secure, system based on static allocation of terminals, local file stores, and processors to individual security partitions are, we would claim, all well within the state of the art. They require the addition of a number of specialised mechanisms to the system, but all are relatively simple, and have established antecedents. Moreover, they all fit easily into the overall architecture we have described, without disturbing the security mechanisms provided earlier, and without intruding into the user's view of the system.

6. Further Topics

In this section we discuss some extensions to the Distributed Secure System presented so far. We begin by considering the integration into our Distributed Secure System of specialised components which have the ability to change or override the security policy enforced by the rest of the system. The most important and necessary of such components are those which provide for 'downgrading' the classification of information.

A true downgrade occurs when information is reassigned to a security classification lower than that which it had originally - either because it is found to have been over-classified in the first place (this may be the consequence of it coming from an insecure computer

system which operates 'system high') or because, due to the passage of time and events, it is no longer as sensitive as it once was. A second form of downgrading is 'sanitisation'. This is the deletion or modification of the most sensitive parts of a body of information in order to produce a new version which may be classified at a lower level than the original. In the world of security administration, sanitisation is quite distinct from downgrading since no information is actually lowered in classification but, in a computer system such as ours, sanitisation will require the same mechanism as downgrading. This is because preparation of the sanitised information must occur in a machine that has access to the highly classified original; the sanitised version of the information will therefore be locked into the security partition occupied by that machine and the downgrading mechanism must be invoked in order to allow it to escape to a lower level.

A number of computer systems have been designed and built to mechanise or assist the processes of sanitisation and downgrading. Examples include the ACCAT and LSI GUARDS [Ames80, Woodward79, Craigen82, Hathaway80, Stahl81], and the University of Texas 'Message Flow Modulator' [Good81]. Some of these systems are able to operate in an automatic mode in which information arriving at one security level is automatically sanitised and sent out again at a lower level, but in most cases they require a 'man in the loop'. This means that a trusted human operator is required to review all material passing through and, after optionally sanitising it, to decide whether or not it may be released at a lower level.

It is not the construction of these 'downgrader' systems that concerns us here, but their integration into our own system. Their physical integration seems a straightforward engineering problem: they need to be provided with a front end process (or processor) that couples them to the remote procedure call interface of a TNIU. The TNIU must be slightly modified so that it will change its encryption key (and, in effect, therefore, the security partition to which it belongs) at the behest of the trusted downgrader machine. This will make it possible for the downgrader to draw a file in from one security partition, process it in some way, and send it back out again in a different partition.

Our real concern, however, is not with these details of physical integration, but with how a user of the system can obtain the services of a downgrader from his own terminal. A crude approach would be to require him to explicitly indicate to his Trustworthy Terminal Interface Unit that he wishes to be connected to a downgrader, rather than to an ordinary host. Subject to authentication and his possession of the appropriate clearances, a secure path could then be set up between the user's terminal and the downgrader machine using the standard encryption techniques. This would enable him to invoke the functions of the downgrader as if he were connected to it directly.

Our objection to this approach is that it is contrary to our desire to present a completely uniform interface to our total system. We would like the user to be able to invoke the downgrade mechanism quite naturally while in normal communication with an untrusted host. Invoking the downgrader should be no different than invoking any other service. Thus, we would like the user to be able to type a 'shell' command such as

```
$GUARD/downgrade < $SFS/TOPSECRET/brian/salaries
                                    > $SFS/SECRET/brian/salaries
```

directly to an ordinary, untrusted host machine (where GUARD and SFS are shell variables that name the machines providing the downgrade and secure file storage facilities).

The solution to this problem of providing a uniform interface to special functions is to realise that there is no problem! An untrusted host confronted with a command such as that above can do one of two things: it can attempt to masquerade as a downgrader, or it can call the real one. The attempted masquerade only presents a denial of service threat - since the standard security mechanisms prevent downgrades by untrusted hosts.

If the host does invoke the real downgrader (by a remote procedure call) then the first task of the downgrader will be to ensure that the call is genuine. It will ask the untrusted host which issued the call for the identity and location of the user who is claimed to have initiated it. The downgrader will then enquire the true identity and clearance of that user from the authenticator in the TTIU to which his host claims he is attached. If the user is found to have the appropriate clearances, the downgrader will request the user's TTIU to provide it with a secure path to the user's terminal. Then the downgrader can ask him "did you really invoke this downgrade?" and, assuming that he confirms that he did, further dialogue may take place to establish the propriety of the operation before it is carried out. To the user, the fact that this dialogue is with a machine different in identity, function, and construction, (and of superior trustworthiness) than that which he used to invoke the service, will be completely transparent; from his terminal he has the (correct!) impression that he is using a single multilevel secure system.

Provided the path used for communication between the downgrader and the user is a secure one, it does not matter that it is set up in response to a request from an untrusted host. The path will only connect trustworthy entities which have authenticated each others' identities and which may be trusted not abuse their special privileges. A host which initiates spurious requests for such paths to be set up, or which fails to initiate genuine ones, is merely a nuisance; it cannot bring about a security compromise.

The mechanisms needed to establish a secure path are little more than a combination of those already present: our encryption mechanisms can establish a secure path from the downgrader's TNIU to the user's TTIU and the internal channels of the TTIU may be trusted to extend that path to the user's terminal. The downgrader and the authenticator of the user's TTIU must be able to authenticate each other before they set up the path between them and this, coupled with the fact that a unique encryption key may be desired for every such path, prompts us to suggest that this is the stage in the evolution of our system at which it is appropriate to add a component to each TNIU and TTIU for managing a cryptographic authentication and key distribution protocol [Denning81, Denning82, Ersham78, Needham78].

Once it becomes possible to access special security mechanisms, such as downgraders, conveniently and safely, it is natural to enquire

what further special mechanisms are both desirable and practicable. The most useful extension to the basic system which it seems feasible to provide with present technology would be the ability to manage 'multilevel objects'. The notion of a multilevel object has been proposed in order to formalise the procedures governing certain types of military messages and documents [Landwehr82]. Intuitively, the concern is to manage documents which not only carry an overall security classification, but whose individual paragraphs bear their own, local classifications. Although it derives from military practice, the idea of a multilevel object is of quite general utility. A mail message to a manager, for example, may have UNCLASSIFIED address and subject fields which may be read by his secretary for filing purposes, but a MANAGEMENT level message field which he alone should see.

In our Distributed Secure System, multilevel objects are most naturally represented by files: a multilevel file will have internal markings indicating the security levels of its constituent parts. A trustworthy 'Multilevel File Manager' will be responsible for assembling and marking the constituent parts of a multilevel file and for extracting the parts that may be seen by users who are not cleared to the level of its overall classification. Outside the multilevel file manager, however, a multilevel file will be treated as an ordinary file belonging to the security partition determined by its overall classification.

The simplest way of constructing the constituent parts of a multilevel file is for them to be prepared as separate files, each belonging to the security partition appropriate to its own contents, using the standard facilities of the system. A multilevel file can then simply comprise a list of the names of the files that contain its constituent parts. It is necessary that the preparation of this list should be performed in a completely trustworthy environment - since it must be performed from within a security partition that has access to the most highly classified information in the file (or to the overall classification of the file if that is higher), but will yield an object whose parts may eventually be seen by less highly cleared users. Although the standard security mechanisms ensure that none of the constituent files will ever be seen by users who are not cleared to view their contents, the selection and ordering of low-level constituents could, if performed in untrustworthy environments, be modulated to convey high-level information.

The identification of the files that constitute the parts of a multilevel file must therefore be performed by a suitably cleared user in secure communication with the trustworthy multilevel file manager. This will require the same mechanism as that described above for establishing secure communications between a user and a downgrader.

Once the multilevel file manager has obtained the list of file names that constitute a multilevel file, it can append a crypto-checksum to the list as a digital signature, and release the list and its signature as an ordinary file belonging to the security partition of its overall classification. Users cleared to see the whole file can recover the information which it represents by reading and assembling the files named within it, using the standard mechanisms for reading files from other (lower) security partitions.

Less highly cleared users require assistance from the multilevel file manager before they can access the parts of a multilevel file that are within their clearance. The multilevel file manager must recover the file of file names that represents the multilevel file and check, using the digital signature which it appended to the file when it was manufactured, that it has not been modified by untrustworthy software in the security partition that has been storing it. If all is well, the multilevel file manager can then prepare a modified list of file names, containing just those which the requesting user is cleared to view. (Although the standard security mechanisms ensure that users cannot read the contents of files which they are not cleared to view, it is probably desirable to suppress information about what information it is that is being suppressed, and in which parts of the file it occurs.) The modified file of file names can then be released to the user, who can then assemble the files named within it.

Numerous modifications and extensions are possible to this scheme for supporting multilevel objects. For example, the multilevel file manager need not sign the lists of file names used to represent multilevel files if these are kept in the Secure File Store and never directly released to untrusted components. Also, a trustworthy 'view' program could be provided by the multilevel file manager if it is desired to exclude the possibility that untrusted machines might attempt to mislead their users by incorrectly displaying the security markings of the constituent parts of multilevel files.

We have chosen only to sketch our proposal for the management of multilevel objects since we hope that the reader will be able to complete it for himself - or to design an alternative mechanism. And this is the main point that we hope to have conveyed here: our system provides a series of simple, easily understood building blocks, which can be combined, modified, and extended in order to provide additional capabilities. It is therefore perfectly reasonable for ourselves, and others, to incorporate additional specialised security mechanisms into the system without destroying its overall coherence, or the simplicity of its basic security mechanisms.

Independent of mechanisms which modify or extend the security properties of the system are those which extend its operational characteristics - most obviously by the provision of inter-networking. In fact, since our architecture depends on physical separation to provide the foundation for its security mechanisms, we have tackled the problems of distributed processing and, by logical extension, inter-networking right from the start: the principles underlying UNIX UNITED extend smoothly to the case where its constituent components are interconnected by a variety and multiplicity of both local and wide-area networks. The naming structure of the a UNIX UNITED system reflects the desired logical relationships among its constituents, and need have no correspondence to their physical inter-connections: close neighbours in the name tree may be situated on different sides of the Atlantic. All the unpleasant details of routing and inter-networking are taken care of by the Newcastle Connection [Black82].

Since inter-networking in UNIX UNITED is logically no different to the case where all communications are provided by a single LAN, the security mechanisms of the basic system apply to a larger, internetworked one as well; provided each component is connected to its

communications medium by a Trustworthy Network Interface Unit (and is otherwise isolated), all will be well. The single difference caused by wide geographical distribution will be in the area of key handling: while it is convenient for all the machines belonging to one security classification and attached to the same LAN to use a single encryption key, the same is not true when the machines may be widely dispersed. Mechanisms for automatic authentication and key distribution between remote TNIUs become necessary in this case.

Summary

We have sketched the means whereby specialised security mechanisms, such as a 'downgrader' and support for 'multilevel objects', can be incorporated into our system, and have indicated how it can be extended to encompass networks of systems.

There are two points which we wish to stress here. Firstly, that specialised mechanisms can be incorporated in a way that does not disturb the overall coherence of the system as perceived by its users, and secondly, that such specialised mechanisms can be added to the basic system as technology and resources permit, without disturbing or complicating the mechanisms already provided.

7. Conclusions

To reiterate, our approach to the design of a secure computing system involves the interconnection of various trustworthy and specialised security mechanisms together with a number of larger untrusted general purpose computers in such a way that users and their programs can treat the resulting somewhat heterogeneous system as a secure, but otherwise conventional, multiprogramming system. Our account has illustrated how each of four distinct methods for achieving separation (physical, temporal, cryptographical and logical) can be used appropriately within such a system, and has described in reasonable detail the functions of several specialised security mechanisms.

It might be thought that we are postulating a rather large number of security-critical mechanisms. However, we believe that in each case the particular mechanism is very simple and within the current state of the art. Indeed, a number of them have previously been proposed (and some implemented) by others - though usually as stand-alone systems. Moreover, any secure system design must surely contain mechanisms for secure communications, terminal access and authentication, file storage and so on. If the system appears to contain only a single security-critical mechanism (such as a security kernel), then that single mechanism must perform all these different tasks and the appearance of simplicity will be illusory. One of the significant benefits of the overall architecture that we have developed is that it separates its security mechanisms, minimises their interaction, and allows them to be used and combined in very straightforward ways. Many additional such mechanisms (e.g. for monitored downgrading, processing of documents containing sections of differing security levels, automatic key distribution, etc.) could be constructed and incorporated into our Distributed Secure System in an equally straightforward and effective manner. This extensibility is another advantage of our approach: installations can provide themselves with just the amount of secure functionality that they require - and can later add more without disrupting what they have already.

In essence, our approach retrofits security mechanisms onto a well-accepted and widely used operating system in a way which should not compromise its overall performance: the computing resources of each security partition are provided by standard, untrusted UNIX systems running at full speed. Moreover this is done in a way that is orthogonal to the mechanisms for achieving reliability and availability that are likely to be called for in any system that is responsible for holding and protecting extremely valuable information. The simplicity of our approach is illustrated by the fact that it took literally just a few days to construct a prototype (which provides multiple security partitions and a multilevel secure file store) by adding crude software simulations of Trustworthy Network Interface Units and of a Secure File Manager to the original UNIX UNITED system running on a set of PDP-11 computers interconnected by a Cambridge Ring.

The construction and certification of a fully realistic implementation of our Distributed Secure System will of course take considerable effort, particularly to develop the detailed designs for the various trustworthy mechanisms to the point where they are of sufficient simplicity for their correctness to be either manifest, or formally proven. Nevertheless it already seems clear that the costs involved will be modest compared to many past attempts at building highly secure systems, and the prospects for achieving satisfactory performance are much greater.

The present report is in fact derived from an earlier one which is serving as the basis for a cooperative development effort aimed at constructing a fully practical, general purpose, distributed multilevel secure system. This project is being sponsored by the Royal Signals and Radar Establishment (RSRE) of the UK Ministry of Defence, and is being carried out by System Designers Ltd. of Camberley, in conjunction with the Microelectronics Applications Research Institute and the Computing Laboratory of the University of Newcastle upon Tyne. The first stage of this project calls for the delivery of a prototype in the Spring of 1983. The security mechanisms of the prototype will be provided by ordinary user processes in a standard UNIX UNITED system. This will not, of course, be secure, but it will allow the operation of the various mechanisms to be studied in practice, it will enable the overall performance of the system to be evaluated, and, most importantly, it will permit the impact of a mechanically enforced security policy to be observed in a realistic environment. If this stage is judged a success, it will be followed by a prototype implementation of the real system. We hope that it will not be long before it is possible to report on the progress of this project and, in due course, on how well it has achieved its security, useability and performance goals.

Acknowledgements

We would not have pursued this work so vigorously without the enthusiastic encouragement of Derek Barnes of RSRE and the stimulation of our many colleagues at Newcastle, particularly those involved with UNIX UNITED. The Newcastle Connection (which is fully operational and presently undergoing commercial evaluation) is the creation of Lindsay Marshall and Dave Brownbridge, while the Remote Procedure Call mechanism is the work of Fabio Panzieri and Santosh Shrivastava. Jay Black provided very helpful criticism of early drafts of this report and an anonymous referee drew our attention to a number of technical problems.

References

[Ames80] S.R. Ames Jr. and J.G. Keeton-Williams, "Demonstrating Security for Trusted Applications on a Security Kernel Base", Proc. 1980 Symposium on Security and Privacy, Oakland, CA., pp.145-156, IEEE Computer Society (April 1980).

[Ames81] S.R. Ames Jr., "Security Kernels: a Solution or a Problem?", Proc. 1981 Symposium on Security and Privacy, Oakland, CA., pp.141-150, IEEE Computer Society (April 1981).

[Anderson72] J.P. Anderson, "Computer Security Technology Planning Study", ESD-TR-73-51 (October 1972). (Two volumes).

[Barnes80] D.H. Barnes, "Computer Security in the RSRE PPSN", Networks '80, pp.605-620, Online Conferences (June 1980).

[Barnes81] D.H. Barnes, "The Provision of End To End Security for User Data on an Experimental Packet Switched Network", Proc. 4th International Conference on Software Engineering for Telecommunications Switching Systems, Warwick, England, pp.144-148, IEE (July 1981).

[Bell76] D.E. Bell and L.J. La Padula, "Secure Computer System: Unified Exposition and Multics Interpretation", ESD-TR-75-306, MITRE Corporation, Bedford, MA. (March 1976).

[Berson82] T.A. Berson and R.K. Bauer, "Local Network Cryptosystem Architecture", Proc. COMPCON, pp.138-143, IEEE Computer Society (Spring 1982).

[Birrell80] A.D. Birrell and R.M. Needham, "A Universal File Server", IEEE Transactions on Software Engineering Vol. **SE-6**(5), pp.450-453 (September 1980).

[Bisbey74] R.L. Bisbey II and G.J. Popek, "Encapsulation: an Approach to Operating System Security", Proc. ACM National Conference, pp.666-675 (1974).

[Black82] J.P. Black and F. Panzieri, "Unix United, Stage II: Internetworking", Internal Report, Computing Laboratory, University of Newcastle upon Tyne, England (November 1982).

[Brownbridge82] D.R. Brownbridge, L.F. Marshall, and B. Randell, "The Newcastle Connection, or UNIXes of the World Unite!", Software - Practice and Experience (To appear, 1982).

[Bruce82] S. Bruce, "Penetration Audit of the UNIX Operating System", M.Sc. Dissertation, Computing Laboratory, University of Newcastle upon Tyne, England (October 1982).

[Craigen82] D. Craigen, "A Formal Specification of the LSI Guard", TR-5031-82-2, I.P. Sharp Associates, Ottawa (August 1982).

[Davida81a] G.I. Davida and J. Livesey, "The Design of Secure CPU-Multiplexed Computer Systems: The Master/Slave Architecture", Proc. 1981 Symposium on Security and Privacy, Oakland, CA., pp.133-140,

IEEE Computer Society (April 1981).

[Davida81b] G.I. Davida, R.A. DeMillo, and R.J. Lipton, "Multilevel Secure Distributed Systems", Proc. 2nd International Conference on Distributed Computing Systems, Paris, pp.8-10, IEEE Computer Society (April 1981).

[Denning81] D.E. Denning and G.M. Sacco, "Timestamps in Key Distribution Protocols", CACM Vol. **24**(8), pp.533-536 (August 1981).

[Denning82] D.E. Denning, Cryptography and Data Security, Addison-Wesley (1982).

[Ersham78] W.F. Ersham et al., "A Cryptographic Key Management Scheme for Implementing the Data Encryption Standard", IBM Systems Journal Vol. **17**(2), pp.106-125 (1978).

[Feiertag77] R.J. Feiertag, K.N. Levitt, and L. Robinson, "Proving Multilevel Security of a System Design", Proc. 6th ACM Symposium on Operating System Principles, pp.57-65 (1977).

[Gasser82] M. Gasser and D.P. Sidhu, "A Multilevel Secure Local Area Network", Proc. 1982 Symposium on Security and Privacy, Oakland, CA., pp.137-143, IEEE Computer Society (April 1982).

[Gligor79] V.D. Gligor and B.G. Lindsay, "Object Migration and Authentication", IEEE Transactions on Software Engineering Vol. **SE-5**(6), pp.607-611 (November 1979).

[Golber81] D.L. Golber, "The SDC Communications Kernel", Presented at DoD Computer Security Industry Seminar, Washington, D.C. (August 1981).

[Gold79] B.D. Gold et al., "A Security Retrofit of VM/370", AFIPS Conference Proc. Vol. **48**, pp.335-344 (1979).

[Good81] D.I. Good, "The Message Flow Modulator", Internal Report, Institute for Computing Science & Computer Applications, University of Texas, Austin, TX. (August 1981).

[Grossman82] G. Grossman, "A Practical Executive for Secure Communications", Proc. 1982 Symposium on Security and Privacy, Oakland, CA., pp.144-155, IEEE Computer Society (April 1982).

[Hathaway80] A. Hathaway, "LSI Guard System Specification (type A)", Draft, MITRE Corporation, Bedford, MA. (July 1980).

[Hinke82] T. Hinke, Private Communication (1982).

[Karger78] P.A. Karger, "The Lattice Model in a Public Computing Network", Proc. ACM National Conference, Vol. 1, pp.453-459 (December 1978).

[Kent77] S.T. Kent, "Encryption-Based Protection for Interactive User/Computer Communication", 5th Data Communications Symposium, Snowbird, UT., pp.5-7 through 5-13, ACM and IEEE Computer Society (September 1977).

[Konheim81] A.G. Konheim, Cryptography - A Primer, Wiley (1981).

[Lampson73] B.W. Lampson, "A Note on the Confinement Problem", CACM Vol. **16**(10), pp.613-615 (October 1973).

[Landwehr81] C.E. Landwehr, "A Survey of Formal Models for Computer Security", Computing Surveys Vol. **13**(3), pp.247-278 (September 1981).

[Landwehr82] C.E. Landwehr and C.L. Heitmeyer, "Military Message Systems: Requirements and Security Model", NRL Memorandum Report 4925, Naval Research Laboratory, Washington, D.C. (September 1982).

[Lipner74] S.B. Lipner, "A Minicomputer Security Control System", Proc. COMPCON, IEEE Computer Society (February 1974).

[Lipner82] S.B. Lipner, "Non-discretionary Controls for Commercial Applications", Proc. 1982 Symposium on Security and Privacy, Oakland, CA., pp.2-10, IEEE Computer Society (April 1982).

[Liu82] L.Y. Liu and B. Randell, "A UNIX Based Triple Modular Hardware Redundancy Scheme", Internal Report, Computing Laboratory, University of Newcastle upon Tyne, England (October 1982).

[Lomet82] D. Lomet et al., "A Study of Provably Secure Operating Systems", RC9239, IBM T.J. Watson Research Center, Yorktown Heights, NY. (February 1982).

[Matyas78] S.M. Matyas and C.H. Meyer, "Generation, Distribution, and Installation of Cryptographic Keys", IBM Systems Journal Vol. **17**(2), pp.126-137 (1978).

[McCauley79] E.J. McCauley and P.J. Drongowski, "KSOS - The Design of a Secure Operating System", AFIPS Conference Proc. Vol. **48**, pp.345-353 (1979).

[NBS77] "Data Encryption Standard", FIPS PUB 46, National Bureau of Standards, Washington, D.C. (January 1977).

[Needham78] R.M. Needham and M. Schroeder, "Using Encryption for Authentication in Large Networks of Computers", CACM Vol. **21**(12), pp.993-999 (December 1978).

[Needham82] R.M. Needham and A.J. Herbert, The Cambridge Distributed Computing System, Addison-Wesley (1982).

[Padlipsky78] M.A. Padlipsky, D.W. Snow, and P.A. Karger, "Limitations of End-to-End Encryption in Secure Computer Networks", MTR-3592, MITRE Corporation, Bedford, MA. (August 1978).

[Panzieri82] F. Panzieri and S.K. Shrivastava, "Reliable Remote Calls for Distributed UNIX: an Implementation", Proc. 2nd Symposium on Reliability in Distributed Software and Database Systems, Pittsburgh, PA., IEEE Computer Society (July 1982).

[Popek78] G.J. Popek and D.A. Farber, "A Model for Verification of Data Security in Operating Systems", CACM Vol. **21**(9), pp.737-749 (September 1978).

[Popek79] G.J. Popek et al., "UCLA Secure UNIX", AFIPS Conference Proc. Vol. **48**, pp.355-364 (1979).

[Popek81] G.J. Popek et al., "LOCUS: A Network Transparent, High Reliabilty Distributed System", Proc. 8th ACM Symposium on Operating System Principles, Asilomar, CA., pp.169-177 (December 1981). (ACM Operating Systems Review, Vol. 15, No. 5).

[Price81] W.L. Price, "The Data Encryption Standard and its Modes of Use", pp. 293-310 in New Advances in Distribute Computing, ed. K.G. Beauchamp, D. Reidel Publ. Co. (1981). (Proc. Nato Advanced Study Institute, Bonas, France, June 15-26, 1981).

[Randell82] B. Randell, "The Structuring of Distributed Computing Systems", Internal Report, Computing Laboratory, University of Newcastle upon Tyne, England (November 1982).

[Ritchie74] D.M. Ritchie and K. Thompson, "The UNIX Time-Sharing System", CACM Vol. **17**(7), pp.365-375 (1974).

[Ritchie79] D.M. Ritchie, "On the Security of UNIX", in UNIX Programmer's Manual, Volume 2 - Supplementary Documents (Seventh Edition), Bell Laboratories (January 1979).

[Rushby81] J.M. Rushby, "The Design and Verification of Secure Systems", Proc. 8th ACM Symposium on Operating System Principles, Asilomar, CA., pp.12-21 (December 1981). (ACM Operating Systems Review, Vol. 15, No. 5).

[Rushby82] J.M. Rushby, "Proof of Separability - a Verification Technique for a Class of Security Kernels", Proc. 5th International Symposium on Programming, Turin, Italy, pp.352-367, Springer-Verlag Lecture Notes in Computer Science, Vol. 137 (April 1982).

[Saltzer75] J.H. Saltzer and M.D. Schroeder, "The Protection of Information in Computer Systems", Proc. IEEE Vol. **63**(9), pp.1278-1308 (September 1975).

[Saltzer81] J.H. Saltzer, D.P. Reed, and D.D. Clark, "End-To-End Arguments in System Design", Proc. 2nd International Conference on Distributed Computing Systems, Paris, pp.509-512, IEEE Computer Society (April 1981).

[Shrivastava82] S.K. Shrivastava and F. Panzieri, "The Design of a Reliable Remote Procedure Call Mechanism", IEEE Transactions on Computers Vol. **C-31**(7), pp.692-697 (July 1982).

[Stahl81] S. Stahl, "LSI Guard Security Specification", MTR 8451, MITRE Corporation, Bedford, MA. (September 1981).

[Woodward79] J.P.L. Woodward, "Applications for Multilevel Secure Operating Systems", AFIPS Conference Proc. Vol. **48**, pp.319-328 (1979).