A DISTRIBUTED SECURE SYSTEM*

(Extended Abstract)

J.M. Rushby and B. Randell**

Computing Laboratory
University of Newcastle upon Tyne
England

Summary

We describe 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. Despite the heterogeneity of its components, 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 the "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. Construction of a prototype implementation is in progress.

1. Introduction

Attempts to construct secure general-purpose operating systems have not been notably successful so far. The performance of those systems that have been built is poor, they often lag many versions behind the conventional operating systems from which they are derived, and doubts have been expressed concerning the extent to which their verification really does provide compelling evidence for their security [9].

We believe that these problems are mainly due to reliance on a security kernel as the primary mechanism for enforcing security. Because it provides an additional level of interpretation, a security kernel necessarily imposes some performance degradation - and this degradation is likely to be greater when general-purpose, rather than specific, applications must be supported. Also, the division of a conventional operating system into untrusted and trusted (security kernel) components is a complex and expensive task which cannot easily accommodate changes and enhancements to its base operating system. Finally, and as we have argued elsewhere [10], security kernels for general-purpose operating systems tend to be complex in themselves, and to have complex interactions with non-kernel "trusted processes" - with the result that the verification of their security properties is neither as complete, nor as convincing, as might be desired. None of these difficulties are arguments against security kernels *per se*; they are arguments against using a security kernel as the sole security mechanism in a general-purpose system.

In this paper we propose a system design which uses a number of different mechanisms in order to provide a secure, general-purpose, distributed computing system. Our proposal involves interconnecting some small, specialized, provably trustworthy systems with a number of larger, untrusted "host" machines in a way that provides 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 specialized services such as multilevel secure file storage.

In short, we propose to finesse the problems that have caused difficulty in the past by aiming to build a distributed secure *system*, rather than a secure *operating system*.

This paper is derived from one which will appear in a forthcoming special issue of IEEE Computer. A much more extensive treatment of this material is available as a technical report [12].

2. Principles and Mechanisms for Secure and Distributed Systems

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 [1]:

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

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

* Research sponsored by the Royal Signals and Radar Establishment, Malvern, England.

** Present address: Computer Science Laboratory, SRI International, Menlo Park, CA. 94025.

execution, and 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. For their own protection, reference monitors must also be kept separate from untrusted system components.

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.*

Separation and mediation represent distinct logical concerns. In the interests of intellectual manageability, not to mention ease of development and verification, the mechanisms which realize them are best kept distinct also. We consider it a weakness of many previous secure system designs that they have confused these two issues and have used a single mechanism - a security kernel - to handle them both.

When separation is recognized as a distinct issue, it becomes possible to consider a number of alternative mechanisms for providing it. It seems clear that the fewer the physical resources that are shared between security levels, the simpler it should be to achieve separation between those levels. Unfortunately, the structure of conventional, centralized 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, a security kernel is needed to synthesize 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, centralized 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 find ways of structuring 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.

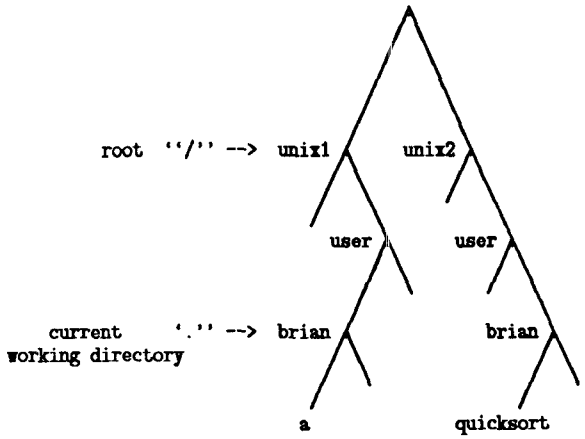
It is costly to provide physically separate systems for each security partition and reference monitor - consequently we use physical separation only for the main computing resources ("hosts") of the system and for the "security processors". Hosts may be used for activities in different security partitions provided those activities are separated in *time*, and are memoryless, while *cryptographic* techniques can be used to separate different uses of shared communications and storage media. Each security processor may support a number of different separation and reference monitor functions, and also some untrusted support functions, by using a *separation kernel* [10] to provide rugged separation between those functions. Experience indicates that separation kernels (simple security kernels whose only function is to provide separation) can be relatively small, simple, and fast [2], and their verification seems simpler and more complete than that for general-purpose security kernels [11].

We term the four separation mechanisms identified above the **physical**, **temporal**, **cryptographical**, and **logical** techniques, respectively. Whereas existing secure system designs tend to exploit only one technique (the logical - kernel based - one), our proposal incorporates all four and uses each wherever it is the most appropriate.

The most significant feature of this approach to the provision of secure computing is that it allows the (untrusted) host machines to provide their full functionality and performance. Another benefit is that it enables the mechanisms of security enforcement to be isolated, single-purpose, and simple. 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 specialized components) must operate as a single coherent system rather than as a network of systems. We have chosen to locate our mechanisms for providing security within the context of a distributed system called "UNIX UNITED" that has been developed in the Computing Laboratory at the University of Newcastle upon Tyne [3]. A UNIX UNITED system is composed of a (possibly large) set of inter-linked 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.

*** UNIX is a Trademark of Bell Laboratories.



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

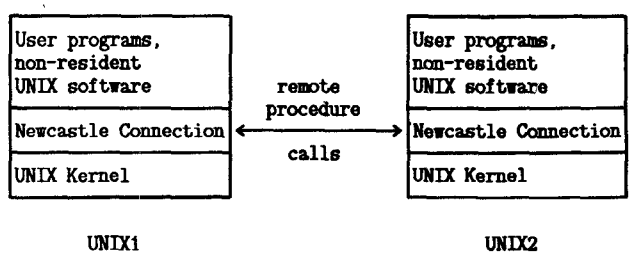
cp a /../unix2/user/brian/a

(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, to use files and peripheral devices on a remote machine, and to set up "pipelines" which cause parallel execution of communicating processes on different machines. 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.

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, it 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 Newcastle Connection layers on the various systems is based on the use of a remote procedure call protocol [13], 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.

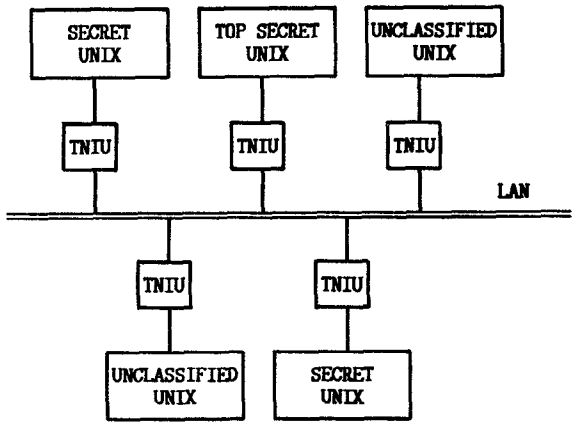
This is the crucial property of UNIX UNITED from our perspective, since it enables a large insecure system to be broken into a number of physically separate components with no visible change at the user level. In the following sections we will explain how we exploit this physical separation in order 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 specialized servers that can masquerade as UNIX) interconnected by a local area network (LAN) and we suppose that all these component systems are untrustworthy: the security of the overall system may make no assumptions about their behavior - 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. Limited 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 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.



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; we will describe later how our system can be extended to move information across partitions securely - thereby providing true multilevel security.

Controlling which hosts may communicate with each other is a reference monitor function. But because the LAN may be subject to both passive and active wire-tapping, the TNIUs must also provide a separation function in order to isolate and protect the legitimate host to host communications channels. This separation function will be provided cryptographically: TNIUs will encrypt all communications sent over the LAN. Since the host machines are untrusted, it is necessary to manage the encryption very carefully in order to prevent clandestine communication between a host machine and a wire-tapping accomplice. Some details of the cryptographic techniques which we employ are given in the full version of this paper [12]. Since the basic principles are well known [6], we only summarize them here.

We use the Cipher Block Chaining (CBC) mode of DES encryption [4] to prevent patterns planted in the plaintext being visible in the ciphertext and we prepend a time-stamp or sequence number to each message prior to encryption in order to cause plaintexts that share a common prefix to yield dissimilar ciphertext. The time-stamps or sequence numbers are also used to detect message replays ("spoofs"). Checksums (protected by encryption) are used to prevent message modification and forgery and care is taken to reduce the bandwidth of clandestine communication channels that modulate message lengths and destinations.

The reference monitor function of the TNIUs is distinct from their separation function and need not interact with it: each TNIU could simply embed the identifier for its own security partition into each outgoing message and refuse to accept incoming messages bearing identifiers different to its own. However, since encryption is being used anyway, it is more attractive to associate the various security partitions with the use of different encryption keys. Each TNIU then needs only a single security-critical item of information: its key. Incoming messages from a different security partition than their receiver will fail to checksum (since they will have been encrypted and decrypted under different keys) and can be discarded.

Any system which uses encryption must contain mechanisms for generating and distributing keys securely. 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 (fixed) security partition to which its host belongs. This, combined with the fact that a LAN-based system is presumed to be geographically compact, makes manual key distribution perfectly viable. If the fear of cryptanalysis causes key changes to be desired more frequently than is convenient for manual distribution, then either a *set* of keys can be installed on each occasion, or else a single *master* key from which the TNIU can manufacture a whole set of communications keys. in either of these cases, the TNIUs must contain mechanisms for synchronizing their current encryption keys. Similar mechanisms are also needed for synchronizing their time-stamps or sequence numbers.

The Remote Procedure Call protocol of the Newcastle Connection requires the protocol layer immediately below it to provide a "fairly reliable" datagram service [13]. This datagram service forms the interface between host machines and and their TNIUs. This arrangement is desirable for the clean and secure integration of encryption into the protocol layering hierarchy, and provides the additional benefit of relieving the host machines of all the low-level network load - thereby improving their overall performance. TNIUs of the required 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 [2, 5]. A separation kernel will be used to enforce plaintext/ciphertext (so-called "red/black") separation within each TNIU. 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 information to flow from the SECRET to the TOP SECRET levels, for example, 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. The SECRET machine must therefore be able to receive information from the TOP SECRET machine as well as send to it - and this conflicts with the multilevel security policy. Notice, too, that this scheme would only provide for unsolicited communications: 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 information flow.

We consider that the best way to provide secure information flow across security boundaries is through 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.

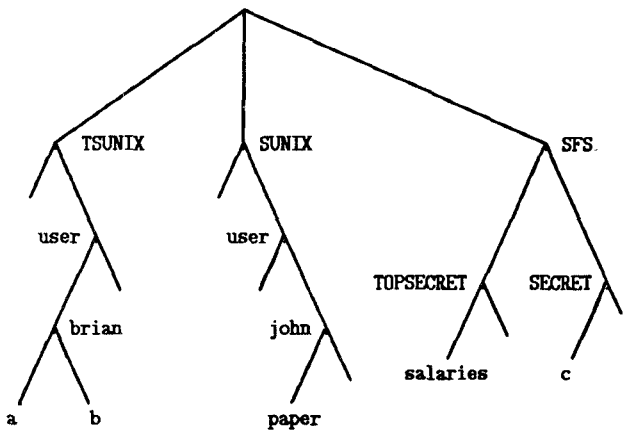
Before describing the mechanism of the Secure File Store, we need to outline its logical position and role within the overall UNIX UNITED system. Conceptually, the Secure File Store is just an ordinary UNIX system that returns exceptions to all system calls except certain ones concerned with files. As 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. 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:

publish paper /../SFS/SECRET/john/paper.

(We will explain later why this command uses "publish" - and



a later one uses "acquire" - instead of the standard UNIX command "cp".) This command will cause the Secure File Store machine to receive a remote procedure call 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 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 operation will be allowed to proceed and the copy of the file will be created. Similarly, when the TOP SECRET user "brian" attempts to obtain a copy of the paper by issuing the command

acquire /../SFS/SECRET/john/paper

the Secure File Store will receive a remote procedure call from the machine TSUNIX. Once again, it can apply the security policy and see that the request 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 the requirements of multilevel security [7]. Similarly, "john" would not be allowed to read the "salaries" file held under the TOPSECRET directory.

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 considerably.

The basic idea is to partition the Secure File Store into trusted and untrusted components housed in physically separate machines. The trusted component, called the "Secure File Manager" (SFM) will be concerned with enforcing the security policy, while its file storage will be provided by the untrusted components. These untrusted components will comprise (conceptually) a number of separate, standard UNIX systems

- each dedicated to a single security partition and identified with one of the sub-directories of the SFS directory. The TNIUs of the file storage machines will be provided with encryption keys that isolate them from each other and from the rest of the system and permit them to communicate only with the SFM. The TNIU of the SFM will have access to all encryption keys, so that the SFM may communicate with hosts in all security partitions as well as with the machines providing its file store.

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.

Host machines requiring access to "secure" files will send Remote Procedure Calls (RPCs) to the SFM. The TNIU of the SFM will know the security partition to which the sender of each RPC belongs (by virtue of the encryption key used in the communication) and will pass this information to the SFM along with the decrypted RPC. The SFM can then inspect the RPC in order to check that the requested operation complies with its security policy. If it does, then the SFM will simply forward the RPC to the appropriate file storage machine for processing and will relay the results back to the original caller.

There is an obvious flaw in this scheme: because the UNIX file storage machines cannot be trusted, they constitute a security weakness. A host machine in the TOP SECRET partition could modulate its (legitimate) requests for reading secure files belonging to the SECRET partition in order to convey TOP SECRET information to the SECRET level file storage machine. This machine could then encode the information received into a file that could subsequently be (legitimately) retrieved by a SECRET level host.

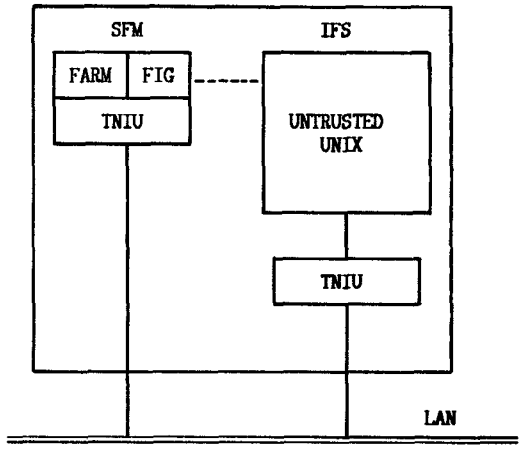
The solution to this problem is to recognize that although it is impossible to prevent TOP SECRET information getting *in* to the SECRET level file store, it is possible to prevent it getting back *out* again.

Since the only objects which leave file storage machines are the files which they retrieve in response to external requests, any clandestine information which is to reach the outside world must be encoded into those files. But all movement of files into and out of the file storage machines is mediated by the SFM - so security will be maintained if the SFM can prevent the file storage machines from encoding information into (i.e. modifying) outgoing files. In other words, security depends upon the SFM being able to guarantee the ***integrity*** of files stored by the file storage machines.

This can be achieved if an unforgeable crypto-checksum is added to each file by the SFM before it is stored in one of the untrusted file storage machines. Any attempt by a file storage machine 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.

Once clandestine information has been prevented from leaving a file storage machine, there is no longer any need to provide separate file storage machines for each security partition: the integrity checks performed by the SFM constitute a separation mechanism on their own. Accordingly, all the file storage machines can be replaced by a single UNIX system called the "Isolated File Store" (IFS).

The revised SFM 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 and checking the checksums on files sent to, or received from, the IFS.



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)

The FIG achieves its purpose by employing 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.

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.

The FIG checksum mechanism only allows files to be read or written in their *entirety*. This is different to the standard UNIX file system interface (which permits incremental reading and writing, and also the repositioning of the file "pointer"). For this reason, "secure" files cannot be accessed through the normal UNIX file system interface but must use a special extension to that interface provided by the Newcastle Connection. This extension adds new system calls to "publish", "acquire", and "delete" secure files, and to "list" the secure files belonging to a given security partition. We consider that the minor inconvenience caused to users by this non-standard interface (which is certainly no worse than that imposed by the "file transfer" programs used in conventional network architectures) is more than outweighed by the simplicity of the trusted mechanisms needed to implement it. Extensions to this scheme which do provide the full, standard UNIX file system interface are described in our report [12], but the difficulties of providing secure access to "i-node" information and to directories do rather compromise the attractive simplicity of the basic scheme. Completely different mechanisms are known and are probably to be preferred in this case.

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 partitions 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 permitted a TOP SECRET connection, even if its user is authorized 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.

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.

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-initialization 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.

As well as a means for causing the TNIU of the host concerned to load the encryption key of a new security partition, we require a temporal separation mechanism to ensure that the host machine is memoryless across its activations in different security partitions. The system state of a host machine is contained in its writable storage: CPU registers, RAM, and disks. The disks of a UNIX system are used for two purposes: they provide swap space, and they contain the local file system.

With the exception of its file system, all the local information available to a host may be considered "transient" and can simply be erased and re-initialized when the host changes security partitions. This can be arranged by causing the host to boot-load a trusted stand-alone "purge" program from ROM on power-up, or on command from its TNIU. This program will systematically clear and re-initialize all temporary storage available to the host processor. Unlike temporary storage, however, the local file system cannot simply be erased but must be retained (inaccessibly) for later activations of the host in the same security partition. Since UNIX UNITED provides transparent access to remote files, this requirement can be achieved by holding files remotely.

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).

Running host machines with absolutely no local file storage is likely to be inefficient and is infeasible if the host's file system is actually held by the Secure File Store (since this uses a non-standard interface). We therefore propose the following refinement to the basic scheme. The purge program will create a local file system on its host's disk and will initialize it to contain the standard utility programs. (These can be obtained from a local read-only floppy disk, or from a "boot-server" accessed over the LAN.) All references to local files will be intercepted by the Local File Relocation Process, which will check to see if they are already present in the local file system. If they are, then the access may be allowed to proceed normally. If they are not, then the Relocation Process must first obtain a copy of the file from the machine that maintains the permanent version of the host's file system for the security partition concerned. Files which are modified or created during a session must, of course, be written back to this permanent file system at the end of the session.

In outline, the complete scenario for automatically changing the security partition in which a host operates is therefore 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 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 host's Local File Relocation Process of the identity of that partition, and initiate the purging and re-initialization of its host machine.

6. Conclusions

We have described a distributed system that provides a limited, but useful form of multilevel secure operation. 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 in order to provide security without undue inefficiency and with a very limited quantity of trusted mechanism. Moreover, our trusted mechanisms are relatively 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. The full version of this paper [12] describes our mechanisms in more detail and discusses some enhancements to the basic system (e.g. access to "downgraders" or "guards" [14], and support for "multilevel objects" [8]).

A project to develop an implementation of the system described here is being sponsored by the Royal Signals and Radar Establishment (RSRE) of the UK Ministry of Defence, and 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.

7. Acknowledgements

We very much appreciate 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.

References

1. J.P. Anderson, "Computer Security Technology Planning Study," ESD-TR-73-51, Air Force Systems Command, USAF, October 1972, (Two volumes).

2. 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.

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

4. D.E. Denning, *Cryptography and Data Security,* Addison-Wesley, 1982.

5. 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.

6. 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.

7. C.E. Landwehr, "A Survey of Formal Models for Computer Security," *Computing Surveys,* Vol. 13, No. 3, pp. 247-278, September 1981.

8. C.E. Landwehr and C.L. Heitmeyer, "Military Message Systems: Requirements and Security Model," NRL Memorandum Report 4925, Naval Research Laboratory, September 1982.

9. D. Lomet et al., "A Study of Provably Secure Operating Systems," Research Report RC9239, IBM T.J. Watson Research Center, February 1982.

10. 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).

11. 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, M. Dezani-Cianaglini and U. Montanari, eds., Springer-Verlag Lecture Notes in Computer Science, Vol. 137, April 1982.

12. J.M. Rushby and B. Randell, "A Distributed Secure System," Technical Report 182, Computing Laboratory, University of Newcastle upon Tyne, England, February 1983, (Also available from the first author at SRI International).

13. S.K. Shrivastava and F. Panzieri, "The Design of a Reliable Remote Procedure Call Mechanism," *IEEE Trans. Computers,* Vol. C-31, No. 7, pp. 692-697, July 1982.

14. J.P.L. Woodward, "Applications for Multilevel Secure Operating Systems," *National Computer Conference,* pp. 319-328, AFIPS Conference Proc., Vol. 48, 1979.