
A distributed general-purpose computing system that enforces a multilevel security policy can be created by properly linking standard Unix systems and small trustworthy security mechanisms.

A Distributed Secure System

John Rushby and Brian Randell
University of Newcastle upon Tyne

A secure system is one that can be trusted to keep secrets, and the important word here is “trusted.” Individuals, governments, and institutions such as banks, hospitals, and other commercial enterprises will only consign their secrets to a computer system if they can be absolutely certain of confidentiality.

The problems of maintaining security are compounded because the sharing of secrets is generally desired but only in a tightly controlled manner. In the simplest case, an individual can choose other individuals or groups with whom he wishes to share his private information. This type of controlled sharing is called discretionary security because it is permitted at the discretion of the individual.

When the individuals concerned are members of an organization, however, that organization may circumscribe their discretionary power to grant access to information by imposing a mandatory security policy to safeguard the interests of the organization as a whole. The most widely used scheme of this type is the multilevel security, or MLS, policy employed in military and government environments[1]. Here, each individual is assigned a clearance chosen from the four hierarchically ordered levels, Unclassified, Confidential, Secret, and Top Secret, and each item of information is assigned a classification chosen from the same four levels. The fundamental requirement is that no individual should see information classified above his clearance.

The fewer the people who share a secret, the less the risk of its disclosure through accident or betrayal to unauthorized persons. Consequently, the basic MLS policy is enhanced by the use of compartments or categories designed to enforce “need-to-know” controls on the sharing of sensitive information. Each individual’s clearance includes the set of compartments of information to which he is permitted access, and the classification of information is similarly extended to include the set of compartments to which it belongs. The combination of a set of compartments and a clearance or classification is called a security partition. An individual is permitted access to information only if his clearance level equals or exceeds the classification of the information and if his set of compartments includes that of the information. Thus an individual with a Secret-level clearance for the NATO and Atomic compartments, abbreviated as a Secret(NATO, Atomic) clearance, may see information classified as Secret(NATO) or Confidential(NATO, Atomic), but not that classified as Top Secret(NATO) or Confidential(NATO, Crypto).

A multilevel secure system should enforce the policy outlined above; unfortunately, conventional computer systems are quite incapable of doing so. In the first place, they generally have no cognizance of the policy and therefore make no provision for enforcing it; there is usually no way of marking the security classification to which a file, for example, belongs. In the second place, experience shows that conventional systems are vulnerable to outside penetration. Their protection mechanisms can always be broken by sufficiently skilled and determined adversaries. Finally, and most worrisome of all, there is no assurance that the system itself cannot be subverted by the insertion of “trap doors” into its own code or by the infiltration of “Trojan horse” programs. In these cases, the enemy is located “inside the walls”

© 1983 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

and the system's protection mechanisms may be rendered worthless. This type of attack is particularly insidious and hard to detect or counter because it can compromise security without doing anything so flagrant as directly copying a Top Secret file into an Unclassified one. A Trojan horse program with legitimate access to a Top Secret file can convey the information therein to an Unclassified collaborator by "tapping it out" over clandestine communication channels that depend on the modulation of some apparently innocuous but visible component of the system state, such as the amount of disk space available.

Drastic measures have been adopted to overcome these deficiencies in the security mechanisms of conventional systems. One approach is to dedicate the entire system to a single security partition. Thus a system dedicated to Secret(NATO) operations would support only information and users belonging to that single security partition. The principal objection to this method of operation is that it fails to provide one of the main functions required of a secure system – the controlled sharing of information between different security partitions. Another drawback is the cost of providing separate systems for each security partition. This problem can be mitigated to some extent by employing *periods processing* in which a single system is dedicated to different security partitions at different times and is cleared of all information belonging to one partition before it is reallocated to a different one.

Another crude method for coping with the security problems of ordinary systems is to require all users to be cleared to the level of the most highly classified information that the system contains. This is called "system high" operation. The rationale is that even if the system has been subverted, it can reveal information only to those who can be trusted with it. The disadvantage to this scheme is that it is very expensive (and counter to normal security doctrines) to clear large numbers of people for highly classified information that they have no real need to know. Furthermore, many excellent people may be unable or unwilling to obtain the necessary clearances. This approach can also lead to the overclassification of information, thereby reducing its availability unnecessarily.

Acronym Definitions

CBC: Cipher block chaining
DES: Data Encryption Standard
FARM: File access reference monitor
FIG: File integrity guarantor
IFS: Isolated file store
LAN: Local area network
MARI: Microelectronics Applications Research Institute
MLS: Multilevel security
RPC: Remote procedure call
RSRS: Royal Signals and Radar Establishment
SFM: Secure file manager
SFS: Secure file store
TNIU: Trustworthy network interface unit
TTIU: Trustworthy terminal interface unit

Several attempts have been made to construct truly secure systems for use in classified and other sensitive environments. However, the builders of such systems face a new problem: They must not only *make* their systems secure, but also convince those who will rely on them that they *are* secure. A full general-purpose operating system is far too complex for anyone to be able to guarantee this security. Accordingly, most efforts have focused on partitioning the system into a small and simple *trusted* portion and a much larger and more complex *untrusted* one. The system should be structured so that all securityrelevant decisions and operations are performed by the trusted portion in a way that makes the untrusted portion irrelevant to the security of the overall system. It is then necessary to rigorously establish the properties required of the trusted portion and prove that it does indeed possess them. Such proofs constitute security verification; they use the techniques of formal program verification to show that the system implementation (usually its formal specification) is consistent with a mathematical model of the security properties required[1,2].

The trusted portion of a secure system is generally identified with a small operating system nucleus known as a security kernel; the rest of the operating system and all applications and user programs belong to the untrusted component. Certain difficulties attend the use of such kernelized systems, however.

Because it provides an additional level of interpretation beneath the main operating system, a security kernel necessarily imposes some performance degradation. This can be minor when specialized applications are concerned, since the kernel can be tuned to the application, but general-purpose kernelized operating systems are three to ten times slower than their insecure

counterparts. Also, the division of a conventional operating system into trusted and untrusted components is a complex and expensive task that cannot easily accommodate changes and enhancements to its base operating system. Consequently, kernelized systems often lag many versions behind the conventional operating systems from which they are derived.

Finally, and as we have argued elsewhere[3], security kernels for general-purpose operating systems tend to be complex, and their interactions with nonkernel trusted processes are also complex. The result is that the verification of their security properties is neither as complete nor as convincing as might be desired. None of these problems are arguments against security kernels per se, which have proved very successful for certain limited and specialized applications such as cryptographic processors and message systems[4]; but they do indicate that security kernels are unlikely to prove satisfactory as the primary security mechanism for general-purpose systems[5].

Our approach is to finesse the problems that have caused difficulty in the past by constructing a distributed secure *system* instead of a secure *operating system*. Our system combines a number of different security mechanisms to provide a general-purpose distributed computing system that is not only demonstrably secure but also highly efficient, cost-effective, and convenient to use. The approach involves interconnecting small, specialized, provably trustworthy systems and a number of larger, untrusted host machines. The latter each provide services to a single security partition and continue to run at full speed. The trusted components mediate access to and communications between the untrusted hosts; they also provide specialized services such as a multilevel secure file store and a means for changing the security partition to which a given host belongs.

The most significant benefits of our approach to secure computing are that it requires no modifications to the untrusted host machines and it allows them 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 this approach makes it possible to construct secure systems whose verification is more compelling and whose performance, cost, and functionality are more attractive than in previous approaches.

Principles and mechanisms for secure and distributed systems

The structure of all secure systems constructed or designed recently has been influenced by the concept of a reference monitor. A reference monitor is a small, isolated, trustworthy mechanism that controls the behavior of untrusted system components by mediating their references to such external entities as data and other untrusted components. Each proposed access is checked against a record of the accesses that the security policy authorizes for that component.

It is implicit in the idea of a reference monitor, and utterly fundamental to its appreciation and application, that information, programs in execution, users, and all other entities belonging to different security classifications be kept totally separate from one another. All channels for the flow of information between or among users and data of different security classifications must be 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 these key notions of *separation* and *mediation*. These are distinct logical concerns, and for ease of development and verification, the mechanisms that realize them are best kept distinct also. We consider it a weakness that many previous secure system designs confused these two issues and used a single mechanism – a security kernel – to provide both. Once we recognize that separation is distinct from mediation, we can consider a number of different mechanisms for providing it and use each wherever it is most appropriate. In fact, our system uses four different separation mechanisms: physical, temporal, logical, and cryptographical.

Physical separation is achieved by allocating physically different resources to each security partition and function. Unfortunately, the structure of conventional centralized systems is antithetical to this approach; centralized systems constitute a single resource that must be shared by 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. This is not only inimical to the efficiency of the system, but it requires complex mechanisms whose own correctness is difficult to guarantee.

In contrast with traditional centralized systems, modern distributed systems are well suited to the provision of physical separation. They necessarily comprise a number of physically separated components, each with the potential for dedication to a single security level or a

single function. To achieve security, then, we must provide trustworthy reference monitors to control communications between the distributed components and to perform other security-critical operations. The real challenge is to find ways of structuring the system so that the separation provided by physical distribution is fully exploited to simplify the mechanisms of security enforcement without destroying the coherence of the overall system.

Because it is costly to provide physically separate systems for each security partition and reference monitor, we use physical separation only for the untrusted computing resources (hosts) of our system and for the security processors that house its trusted components. *Temporal* separation allows the untrusted host machines to be used for activities in different security partitions by separating those activities in time. The system state is reinitialized between activities belonging to different security partitions.

The real challenge is to find ways of structuring the system so that the separation provided by physical distribution is fully exploited to simplify the mechanisms of security enforcement without destroying the coherence of the overall system.

The security processors can each support a number of different separation and reference monitor functions, and also some untrusted support functions, by using a *separation kernel* to provide *logical* separation between those functions. Experience indicates that separation kernels (simple security kernels whose only function is to provide separation) can be relatively small, uncomplicated, and fast, and verification seems simpler and potentially more complete for them than it does for general-purpose security kernels[3].

Our fourth technique, *cryptographic* separation, uses encryption and related (checksum) techniques to separate different uses of shared communications and storage media.

The four separation techniques provide the basis for our distributed secure system. This is a heterogeneous system comprising both untrusted general-purpose systems and trusted specialized components, and to be useful it must operate as a coherent whole. To this end, our mechanisms for providing security are built on a distributed system called Unix United, developed in the Computing Laboratory at the University of Newcastle upon Tyne[6]. 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, and system administrator. The naming structures (for files, devices, commands, and directories) of each component Unix system are joined 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 directory naming structure of a Unix United system is set up to reflect the desired logical relationships between its various machines and is quite independent of the routing of their physical interconnections.

The simplest possible case of such a structure, incorporating just two Unix systems, named `unix1` and `unix2`, is shown in Figure 1. From `unix1`, and 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 Unix 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 at the current working directory, and the “..” symbol is used to indicate a parent directory.)

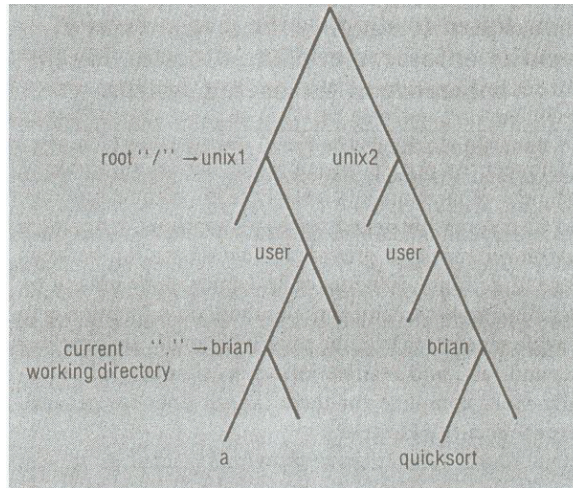


Figure 1: The naming structure of a simple Unix United system.

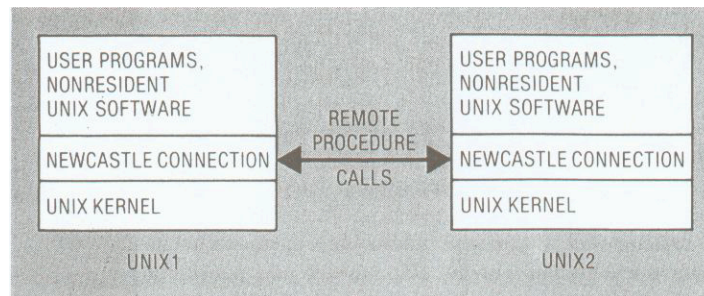


Figure 2: The Newcastle Connection.

This command is in fact a perfectly conventional use of the standard Unix shell command interpreter and would have exactly the same effect if the naming structure shown had been set up on a single machine and unix 1 and unix2 had been conventional directories.

All the standard Unix facilities, whether invoked by shell commands or by system calls within user programs, apply unchanged to Unix United, causing intermachine communication as necessary. A user can therefore specify a directory on a remote machine as his current working directory, request execution of a program held in a file on a remote machine, redirect input and/or output, use files and peripheral devices on a remote machine, and set up pipelines that cause parallel execution of communicating processes on different machines. Since these are completely standard Unix facilities, a user need not be concerned that several machines are involved.

Unix United conforms to a design principle for distributed systems that we call the “recursive structuring principle”. This requires that each component of a distributed system be functionally equivalent to the entire system. Applying this principle results in a system that automatically provides network transparency and can be extended (or contracted) without requiring any change to its user interface or to its external or internal program interfaces. The principle may seem to preclude systems containing specialized 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 an ordinary host machine can return an exception when asked to operate on a nonexistent file, so a specialized server that provides no file storage can always return exceptions when asked to perform file operations.

Unix United has been implemented without changing the standard Unix software in any way; neither the Unix kernel nor any of its utility programs – not even the shell command interpreter – have been reprogrammed. This has been accomplished by incorporating an additional layer of software called 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 looks like a normal user process. Its role is to filter out system calls that have to be redirected 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 and is shown schematically in Figure 2.

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, the local procedure call is simply intercepted by the Newcastle Connection and replaced with a remote one. This substitution is completely invisible at the user or program level, providing a powerful yet simple way of putting systems together. Equally important, it provides a means of partitioning what appears to be a single system into a number of distributed components. From our perspective, this partitioning is the crucial property of Unix United, since it enables a large, insecure Unix system to be broken into a number of physically separate components with no visible change at the user level. The following sections will explain how we exploit this physical separation to construct a secure system. We begin with a very simple system that merely isolates different security partitions from one another.

A securely partitioned distributed system

We will describe a secure Unix United system composed of standard Unix systems (and possibly some specialized servers that can masquerade as Unix) interconnected by a local area network, or LAN. We assume that all the component Unix systems are untrustworthy and that the security of the overall system must not depend on assumptions concerning their behavior – except that the LAN provides their only means of intercommunication.

The consequence of not trusting the individual systems is that the unit of protection must be those systems themselves; thus, we will dedicate each to a fixed security partition. We might allocate two systems to the Secret level, one to the Top Secret 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 Management. Users are assigned to hosts with the knowledge that no security is guaranteed within those individual systems. Note also that since the hosts are not trusted, they cannot be relied upon to authenticate their users correctly. Therefore, access to each system must be controlled by physical or other external mechanisms.

Although there is no security *within* an individual Unix 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 call it a trustworthy network interface unit, or TNIU (see Figure 3).

The initial and very restrictive purpose of TNIUs is to permit communication only between machines belonging to the same security partition. 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 between partitions securely, thereby providing true multilevel security.

Controlling which hosts can communicate with one another is a reference monitor function, but because the LAN can be subverted or tapped, the TNIUs must also provide a separation function to isolate and protect the legitimate host-to-host communications channels. This separation function is provided cryptographically, with TNIUs encrypting all communications sent over the LAN. Encryption is traditionally used to protect communications between parties who share a common interest in preserving the secrecy of that communication, but this is not the case here. Host machines are untrusted and may attempt to thwart the cryptographic protection provided by their TNIUs. For this reason, the encryption must be managed very carefully to prevent clandestine communication between host machines, or between a host machine and a wiretapping accomplice.

Although the basic principles of encryption management are well established[8], a tutorial outline of the issues and techniques as they affect our system may benefit readers to whom this material is new.

Encryption and the protection of communications. Trustworthy network interface units use the Data Encryption Standard, or DES[8] to protect information sent over the LAN. However, since host machines are untrusted and the LAN can be tapped, the simplest form of DES encryption – the so-called electronic code book mode – is ruled out. In this mode, each 64-bit block of data is encrypted as a separate unit, and even a very powerful encryption algorithm such as the DES cannot prevent the leakage of information from a corrupt host machine under these circumstances. For example, suppose that a corrupt host wishes to communicate the bit pattern 01101 to a wiretapping accomplice. The host constructs a

message XYYXY, where X and Y are arbitrary but distinct bit patterns of the same length and alignment as the units of block encryption, and sends it to its TNIU for transmission. The TNIU will encrypt the message to yield, say, PQQPQ before transmitting it over the LAN, but the bit pattern 01101 remains visible in this encrypted message and can easily be extracted by a wiretapper. Notice that the threat here is not due to any weakness in the encryption algorithm employed, but to the way in which it is used; one need not be able to decrypt messages to extract information planted by a corrupt machine,

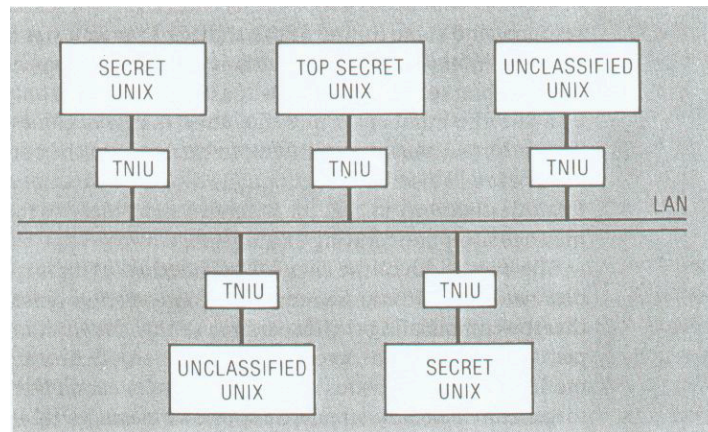


Figure 3: A securely partitioned system.

Clandestine communications channels based on plaintext patterns that persist into the ciphertext can be thwarted by employing a more elaborate mode of encryption called cipher block chaining, or CBC, which uses a feedback technique to mask such patterns by causing the encrypted value of each block to be a complex function of all previous blocks[8]. Of course, identical messages will yield identical ciphertexts, even when CBC-mode encryption is used. More importantly, messages that begin with the same prefix will yield ciphertexts that also share a common prefix. A corrupt host can therefore signal to a wiretapping accomplice by modulating the length of the prefix that successive messages have in common. This channel for clandestine information flow must be closed, and this will be achieved if TNIUs attach a random block of data, different in each case, to the front of each message before encrypting it,

The careful use of CBC-mode encryption prevents information from leaking through channels that modulate message *contents*, but significant channels for information leakage still remain. These are pattern-of-use channels whereby a corrupt host modulates the visible parameters of messages in a way that can be decoded by a wiretapping accomplice. The properties that can be modulated are the lengths of individual messages, their time and frequency of transmission, and their destination.

All techniques for introducing noise inevitably reduce the bandwidth available for legitimate communications and may increase the latency of message delivery.

(Presumably the source is fixed at the location of the corrupt host.) These properties, of which length and destination are by far the most important, can be modulated to yield clandestine communication channels of surprisingly high bandwidth. Unless link encryption is used to reinforce the end-to-end encryption described here, it will not be possible to completely sever these channels. Since link encryption is infeasible with most LAN technologies, the best approach is to reduce the bandwidth of these pattern-of-use channels to a tolerable level, either directly or through the introduction of noise,

The length channel is the easiest to deal with. TNIUs process message units of a fairly large, fixed size – say 1024 bytes. Long messages must be broken into a number of separate message units; short ones, and the residue of long ones, must be padded to fill a whole unit. (If this technique causes great numbers of largely empty message units to be generated, 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 the unit size is possible in any case.) When this is done, a

wiretapper cannot observe the exact length of a message but can only estimate the number of message units that it occupies. This information will be difficult to extract, and the corrupt host will also have to modulate a second parameter (e.g., destination) so that the wiretapper can identify the message units constituting each message.

The bandwidth of the channel that modulates message destinations can only be reduced by introducing noise, thereby complicating traffic patterns so that the wiretapper 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 messages to all other TNIUs in its own security partition. Spurious messages are marked as such (under encryption, of course) and are discarded by TNIUs that receive them. More refined strategies, such as routing messages 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 must choose its priorities in such a trade-off.

The techniques described so far enforce separation between the outside world and communications internal to the distributed secure system. They do not, however, provide separation between the different *internal* communications channels of the system. Thus, the reference monitor component of a Secret-level TNIU can determine that its host is attempting to communicate with another Secret-level host and that this communication accords with the security policy and may therefore proceed; however, it cannot prevent the LAN messages that constitute the communication from being delivered, either through error or malice, to the TNIU of, say, an Unclassified host. Furthermore, unless additional mechanisms are introduced, the receiving TNIU will not necessarily be able to detect that the messages have been sent to it in error.

Incorrect delivery can occur because the LAN hardware, by accident or intent, misinterprets message destination fields, or because those fields are modified by an active wiretapper. (Remember that these fields must be in the clear so that the LAN hardware can interpret them.) TNIUs may attempt to overcome this threat by embedding the true source, destination, and security partition of each message unit inside the data portion of the message unit itself, where it will be protected by encryption. However, this technique can be defeated by an active wiretapper who splices the identification portion of a genuinely Unclassified message onto the body of a Secret one.

It might appear that CBC-mode encryption automatically protects against this type of attack and that because the encrypted value of each block within a message unit is a complex function of all previous blocks, messages formed by splicing parts of different messages together *will* decrypt unintelligibly. In fact, this is not so. Although the encrypted value of each block produced by CBC-mode encryption depends *implicitly* on all prior plaintext blocks, it depends *explicitly* on only the *immediately preceding* ciphertext block[8]. Thus, damage to the contents or sequencing of ciphertext blocks affects only the decryption of the block immediately following the damaged or misplaced block; in other words, CBC-mode decryption is “self-healing.”

Two methods are available for securely separating the communications channels belonging to different security partitions. The first uses a high-quality checksum to guarantee the integrity of each message unit, including its identification fields. TNIUs must calculate the checksum of each message unit before they encrypt it, and they must encrypt the message unit and its checksum as a single unit so that the checksum will be protected by encryption. Whenever a TNIU receives a message unit, it must first decrypt it and recompute its checksum. Only if the recomputed checksum matches the one sent with the message unit should the unit be accepted by the TNIU for further processing. The integrity of all message units accepted is thereby guaranteed because they cannot be forged, modified, or formed by splicing parts of different units together during transmission over the LAN. Consequently, TNIUs can trust the value of the security partition identifier embedded in each message unit, then they can (and must) reject those bearing a different identifier.

The second method for distinguishing the communications belonging to different security partitions is to use a different encryption key for each partition. (Until now, we have implicitly assumed that the same key is used for all communications.) Each TNIU will be provided with only the single key associated with its own security partition and will therefore have no way of communicating with TNIUs belonging to different partitions. If a message unit is delivered to a TNIU belonging to a different security partition from its sender, it will be encrypted using one key and decrypted using another, making it unintelligible to the host attached to the receiving TNIU. It is unwise, however, to allow the untrusted host machines to see even such unintelligible transmissions from another security partition, so we propose to combine the use of different encryption keys with the checksum technique described earlier. A

message delivered to a TNIU in a different security partition from its sender, and therefore encrypted and decrypted with different keys, will certainly fail to checksum correctly.

The use of both checksums and different encryption keys is not strictly necessary, since either technique is sufficient to separate the communications channels belonging to different security partitions. The two techniques are complementary, however, and provide worthwhile redundancy. Checksums guarantee the integrity of message contents, a very desirable property in its own right, while the use of different encryption keys provides failsafe separation.

Any system using encryption must contain mechanisms for generating and distributing keys securely. But unlike connection-oriented (virtual circuit) schemes in which a unique key must be manufactured and distributed every time a new circuit is opened, 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 presumption that a LAN-based system is geographically compact, makes manual key distribution perfectly viable. Because of its evident simplicity and security, this is the mechanism we employ. If the fear of cryptanalysis calls for more frequent key changes than is convenient for manual distribution, a *set* of keys can be installed on each occasion or a single *master key* can be installed from which the TNIU can manufacture a whole set of communications keys. In either case, the TNIUs must contain mechanisms for synchronizing their current encryption keys.

Although not strictly necessary, it is highly desirable to be able to detect and counter the activity of an active wiretapper who attempts to “spoof” the system by replaying recordings of genuine LAN messages. (Consider, for example, a banking system that carries messages such as “move \$100 from account A to account B.”) Spoofs can be detected if sequence numbers or time stamps are embedded in each message unit. Of course, it is perfectly feasible for hosts to do this themselves, but it seems more appropriate for TNIUs to provide this function. The sequence number or time stamp of each message unit can constitute the unique material that should be attached to the front of each message prior to encryption to mask the similarity between messages that share a common prefix. Synchronizing 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 it obviously cannot use sequence numbers or time stamps itself for this purpose. A challenge-response technique first proposed by Needham and Schroeder[9] can be used instead.

The integration and construction of TNIUs. The interposition of a TNIU between a host and its LAN station raises interesting questions concerning the location of various protocol functions. The whole issue of assigning function to layers in a protocol hierarchy can become quite complex in the presence of encryption because standard functions such as checksums and sequence numbering, for example, are duplicated – though in a different and more sophisticated way – by the protection and security mechanisms. For this reason, TNIUs should not operate below the normal protocol layering hierarchy but must be integrated with it. In fact, we propose that TNIUs take over all protocol functions, except those at the highest level. The benefit of this approach is that the TNIUs act as network front ends, relieving their hosts of the low-level network load and thereby boosting overall performance.

The top-level protocol of the Newcastle Connection provides a remote procedure call, or RPC, service and requires a fairly reliable datagram service from the lower levels of its protocol hierarchy. We use this datagram service as the interface between host machines and their TNIUs; individual datagrams form the message units that are encrypted and protected by the TNIUs. Most RPCs and their results can be encoded into a single datagram, but those concerned with file reads and writes, which can transfer arbitrarily large amounts of data, are broken into as many separate datagrams as necessary by a subprotocol of the host machines’ RPC protocol.

The cryptographic techniques employed by TNIUs counter the threat of information disclosure. The remaining danger is 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, TNIUs must, as a correctness requirement, continue to provide reliable (though necessarily degraded) service in spite of such occurrences. It is also a correctness requirement of TNIUs that they recover from crashes safely. (Of course, verified software does not crash, but we must allow for the possibility of a power failure.)

TNIUs that perform all the functions described certainly present a challenge in both construction and verification. We argue, however, that they are very similar to the cryptographic front ends of wide-area networks, and examples of these have already been built and, in some cases, verified.⁴ Modern 16-bit microprocessors provide a suitable hardware base for the construction of TNIUs, and single-chip implementations of the DES algorithm are available that can perform CBC-mode encryption at LAN speeds. A separation kernel must be

used to enforce cleartext/ciphertext (so-called red/black) separation within each TNIU, with the basic physical protection provided by the memory management chips appropriate to the chosen processor. Since no disks are needed (the software can be held in ROM), a complete TNIU should fit on a single board and cost less than a thousand dollars.

Unclassified hosts can generally be considered to belong to the same security partition as the outside world. They need not be separated from it, and therefore their TNIUs need not use encryption to protect their communications. In this case, there is no need to provide TNIUs to Unclassified hosts, and this provides a worthwhile economy in systems where the majority of hosts are Unclassified. It also permits a standard, unpartitioned Unix United system to be smoothly upgraded to a securely partitioned one by the addition of a limited number of TNIUs.

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 will now explain how to extend this design to permit information to cross security partitions in a multilevel secure manner. This will allow information to flow from the Secret to the Top Secret levels, for example, but not vice versa.

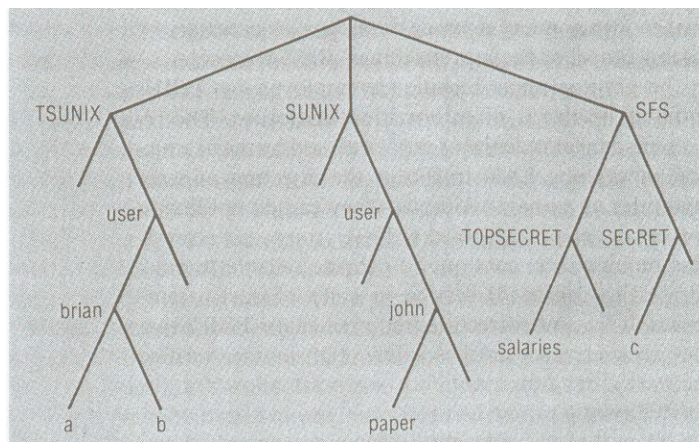


Figure 4. The naming structure of a simple Unix United system incorporating a secure file store.

It might seem that multilevel secure information flow could be provided by simply modifying the policy enforced at the TNIUs so that, for example, Top Secret machines could 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 could not 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. 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. If one host sends a file to another, the sender needs to know that the receiver has been able to store the file correctly, not merely that its TNIU received it correctly. 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 because its request would constitute an insecure information flow.

The best way to provide secure information flow across security boundaries is through a trustworthy intermediary that provides an independent and useful service. The complexity of such an intermediary will depend on the generality of that service. Combining simplicity with the most useful function, we have selected files as the only objects allowed to cross security boundaries, and we have chosen the multilevel secure storage and retrieval of files as the service provided by the trustworthy intermediary. This is achieved by adding to the system a secure file store with the ability to communicate with hosts of all security classifications. The idea is that when a Secret-level host 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 host can subsequently acquire 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 subdirectories 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 in Figure 4.

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 other's existence, any attempted intercommunication 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 that is subordinate to the SFS directory. For example:

```
publish <paper/ ../SFS/SECRET/john/paper.
```

(We 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 to determine whether such a machine is allowed to create Secret-level files. Assuming 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 print a copy of the paper by issuing the command

```
acquire ../SFS/SECRET/john/paperllpr
```

the secure file store will receive a remote procedure call from the machine TSUnix requesting a copy of the file. Once again, it can consult the security policy, where it will see that the request should be allowed to proceed. The secure file store will, however, refuse requests from TSUnix to write into this "paper" file, or to delete it, since these contravene the requirements of multilevel security. Similarly, John will not be allowed to read the "salaries" file held under the TOPSECRET directory.

We now move from the services provided by the secure file store to its construction. A multilevel secure Unix file system might seem to demand a substantial number of provably trustworthy mechanisms – virtually a secure Unix. With careful design, however, we can reduce the number of trusted mechanisms 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, or SFM, is a reference monitor concerned with enforcing the security policy; its file storage is provided by the untrusted components. These untrusted components can be thought of as separate, standard Unix systems connected directly to the SFM. Each untrusted file storage machine is dedicated to a single security partition and is identified with one of the subdirectories of the SFS directory (see Figure 5).

Communicating with hosts in different security partitions requires an enhanced TNIU for the SFM, one that contains the encryption keys of all security partitions. The internal structure of a TNIU with multiple encryption keys will be slightly more complex than that of 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 virtual machines, 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 for the trusted mechanism of a TNIU.

Host machines requiring access to secure files send remote procedure calls, or RPCs, to the SFM. The TNIU of the SFM determines the sender's security partition and passes this information to the SFM along with the decrypted RPC. The SFM can then inspect the RPC to check if the requested operation complies with its security policy. If it does, the SFM simply forwards the RPC to the appropriate file storage machine for processing and relays the results (suitably encrypted) back to the original caller.

There is an obvious flaw in this scheme, however. Because the Unix file storage machines cannot be trusted, they constitute a security weakness – even though each holds files

belonging to only a single security partition. A host machine in the Top Secret partition could modulate its legitimate requests for reading secure files belonging to the Secret partition to convey Top Secret information to the Secret-level file storage machine. For example, suppose the secret file storage machine contains a group of 26 files, each a different length. If a corrupt Top Secret host requests, as it may legitimately do, copies of the fifth, fourteenth, ninth, seventh, thirteenth, and first shortest files in that order, then the Top Secret string E-N-I-G-M-A will have been communicated to the Secret file storage machine. This machine could then encode the information received into a file that could subsequently and legitimately be retrieved by a Secret-level host.

Although we cannot prevent Top Secret information from getting *into* the Secret-level file store, we *can* prevent it from getting back *out* again. Once we recognize this, the solution to the above problem is at hand.

The only objects that leave file storage machines are files retrieved in response to external requests. Consequently, any clandestine information that is to reach the outside world must be encoded into those files. Since all movement of files into and out of the file storage machines is mediated by the SFM, security will be maintained if the SFM prevents the file storage machines from encoding information into (i.e., modifying) outgoing files. In other words, security depends on the SFM being able to guarantee the integrity of files retrieved from the file storage machines.

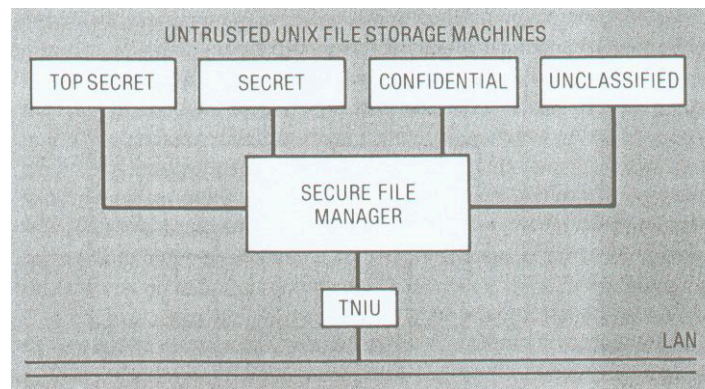


Figure 5. Conceptual structure of the secure file store.

This can be achieved if a 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. Of course, this only works so long as the file storage machines are unable to forge the checksums. This can be ensured in two ways (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 file storage machines) but to protect it by encrypting the file and the checksum as a single unit. The second technique is to use a crypto-checksum that depends on a secret key for its computation. An example of a crypto-checksum is the final block of ciphertext produced during CBC-mode encryption, an alternative is to simply encrypt a conventional checksum. The advantage of crypto-checksums is that they cannot be forged by those who do not possess the key; they can therefore be used with information stored in the clear.

Either technique can be used to guarantee the integrity of files retrieved from the untrusted file storage machines. We prefer the crypto-checksum approach because it requires only a single operation. Intermediate checksums can be included at intervals within the file if the SFM has insufficient space to buffer an entire file. If part of a file has already been delivered to a host when modification to a later part is detected by the SFM, then some clandestine information may have been conveyed to the host through the position at which the modification began and file transfer was aborted by the SFM. This channel has very limited bandwidth, and as long as all checksum failures raise a security alarm and are logged by the SFM, it is not considered a serious security risk.

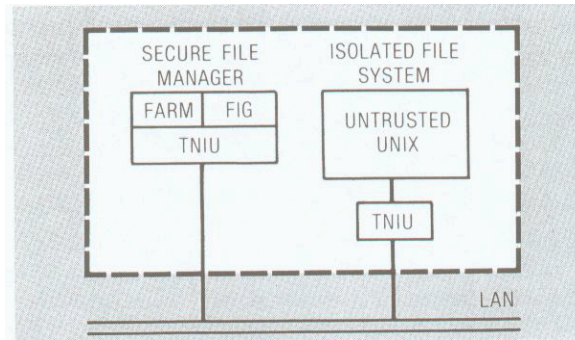


Figure 6. Actual structure of the secure file store.

Checksums prevent the untrusted file storage machines from modifying the files consigned to them and from manufacturing forgeries, but they do not prevent them from signaling to a collaborator by choosing *which* legitimate files they return in response to requests. For example, a Secret host could send a steady stream of requests for file X, but the files actually returned by the Secret file storage machine could be quite different. In particular, they could be files selected on the basis of length, say, to convey information in the manner of the E-N-I-G-M-A example given earlier. To close this channel, the SFM must be able to verify that the correct file is returned in response to each request. This is easily accomplished by including the name of each file in the checksum calculation.

A variation on this method of covert communication is not so easily countered, however. A file storage machine can keep several old copies of a legitimate file and signal to an outside collaborator by choosing which version of the file to return in response to each request. This type of attack can be countered by recording a time stamp with each file and keeping a separate record of the time stamp that identifies the current version of the file. The problem here is to find a safe place to keep the record of each file's current time stamp. It cannot be entrusted to the file storage machines without additional mechanisms for safeguarding its own integrity, and keeping it in the SFNI will impose a substantial storage requirement on a machine that is intended to have no disks of its own.

To keep the trusted mechanism of the SFM simple, we prefer to reduce the bandwidth of this channel rather than attempt to close it completely. The SFM embeds a time stamp into each file before calculating its checksum and consigning it to an untrusted file storage machine. In addition, the SFM maintains, in its own private storage, a cache of the names and time stamps of all files read from or written to an untrusted file storage machine during, say, the last five minutes. Any attempt to return different versions of the same file within a shorter period will be detected by the SFM and will raise a security alarm. Attacks that operate over a longer period will go undetected, but their bandwidth will then be so low that they can be discounted as serious threats.

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 the required separation mechanism. Accordingly, the file storage machines can all be replaced by a single Unix system called the isolated file store, or IFS. Rather than connect the IFS directly to the SFM, we prefer to connect it to the LAN via a TNIU in the standard way. For it to be truly isolated from the rest of the system, however, the TNIU of the IFS must be loaded with a special encryption key that is shared only with the TNIU of the SFM (see Figure 6).

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, or FARM, and the file integrity guarantor, or 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 and time stamps on files sent to or received from the IFS.

The FIG achieves its purpose by employing checksum techniques very similar to those used for LAN messages by the TNIUs. We therefore suggest constructing the FIG by making 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 can be performed inside a separate virtual machine provided by the separation kernel of the machine that supports the TNIU/SFM functions.

We therefore conclude that all the functions of a complete SFM can be easily integrated into the TNIU that 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 – approximately a thousand dollars.

The FIG checksum mechanism allows files to be read or written only in their entirety. This is different from the standard Unix file system interface, which permits incremental reading and writing, and 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 names of the secure files belonging to a given security partition, (The list operation must be implemented very carefully so as not to provide the IFS with a clandestine information channel.) The minor inconvenience caused for users by this nonstandard interface is certainly no worse than that imposed by the file transfer programs used in conventional network architectures and is more than outweighed by the simplicity of the trusted mechanisms needed to implement it. Extensions to this scheme that do provide the full, standard Unix file system interface are described in a technical report[10], but the difficulties of providing secure access to Unix i-node information and to directories tend to compromise the attractive simplicity of the basic scheme. Completely different mechanisms are known and are probably preferable in this case.

The accessing and allocation of security partitions

A system 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 user 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 make additional services possible by connecting terminals to trustworthy terminal interface units, or TTIUs, rather than to hosts directly. Moreover, we can then include provisions for dynamically changing the allocation of machines to security partitions.

Accessing different security partitions. What we term a trustworthy terminal interface unit is basically a TNIU enhanced with some additional trusted functions, including a terminal driver, some very limited Newcastle Connection software, and an authentication mechanism. Each of these logically separate mechanisms runs in an individual virtual machine provided by the separation kernel supporting 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 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 is not 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 contact 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 nonerror response are those appropriate to terminals, namely “read from the keyboard”, “write to the screen,” and a couple of others 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.

A system in which terminals are attached to machines of fixed security level can be somewhat inconvenient to use.

None of the additional trusted mechanisms required to upgrade a TNIU to a TTIU should present an undue challenge in either construction or verification, Nor should the presence of these additional mechanisms affect the construction or verification of the TNIU components themselves, since TNIUs are constructed on top of a separation kernel. 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.

Changing security partitions dynamically. TTIUs enable users to connect to machines in different security partitions, thereby allowing them to perform each of their activities at the most appropriate level within their clearance. If a security policy with a large number of need-to-know compartments is supported, however, the number of different security partitions can well exceed the number of physical hosts available. Even when the number of distinct security partitions is small, the demand for resources within each partition can vary with time. Furthermore, some users might prefer to use personal workstations for their activities in many different security partitions. All these cases require some provision for reallocating host machines to different security partitions.

With untrusted hosts, this can only be accomplished by temporal separation, which in its simplest form is periods processing. This requires manual intervention to exchange all demountable storage and the reinitialization of all fixed storage 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 it is slow and expensive. We therefore propose mechanisms for automating the process, making it both rapid and secure. The mechanisms required include one that causes a host's TNIU to load the encryption key of a new security partition, and another that provides temporal separation for different uses of the host machine,

A project to develop an implementation of the system described here has a sponsor and is being carried out.

The system state of a host machine is contained in its writable storage: CPU registers, RAM, and disks. The disks of a Unix system provide swap space and contain the local file system. With the exception of the file system, the local storage available to a host is all used for strictly temporary purposes and can simply be erased and reinitialized when the host changes security partitions. This is achieved 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 systematically clears and reinitializes all temporary storage available to its host processor.

Unlike temporary storage, the local file system cannot just be erased when the host changes security partitions; it must be retained (inaccessibly) for later activations of the host in the same partition. Since Unix United provides convenient access to remote files, this requirement can be satisfied by holding files remotely, either in file servers dedicated to particular security partitions, or in the secure file store.

Operating host machines without local file store is inefficient. Accordingly, the purge program creates a local file system on its host's disk and initializes 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.) Each reference to an apparently local file is intercepted by a local file relocation process added to the Newcastle Connection. This process checks to see if the requested file is already present in the local file system. If it is, the access is allowed to proceed normally. If it is not, the relocation process first obtains a local copy of the file from the machine that maintains the permanent version of the host's file system for the security partition concerned. For example, if the host is known as PW5 (personal workstation number 5) and is currently operating in the Secret(NATO) partition, then a reference to the local file /user /john /paper might cause the local file relocation process to obtain a copy of the file ../SNSERVER/PW5/user/john/paper, where SNSERVER is the name of the machine that maintains the Secret(NATO) file system. This process is perfectly straightforward and need not 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). Files modified or created during a session must, of course, be written back to their permanent file system by the local file relocation process at or before the end of the session,

In outline, the complete scenario for automatically changing the security partition in which a host operates is as follows. 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 is sent to the TNIU of a vacant host machine (or to the user's personal workstation) instructing it to switch to the indicated security partition. This signal is protected against forgery or spoofing by the standard encryption techniques employed between TNIUs. Upon

receipt of the signal, the host's TNIU loads the encryption key appropriate to the new security partition, initiates the purging and reinitialization of its host machine, and informs the host's local file relocation process of the identity of its new security partition,

We have described a distributed system that provides a limited but useful form of multilevel secure operation. Four distinct methods for achieving separation (physical, temporal, cryptographic, and logical) have been illustrated. When used judiciously, they can provide security without inefficiency and with only a limited number of trusted mechanisms. 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. A more extensive treatment of the topics covered in this article is available as a technical report[10]. It describes our mechanisms in more detail, relates them to their precursors, and discusses some enhancements to the basic system (the inclusion of downgraders or guards, and support for multilevel objects, for instance). Readers who wish to learn more about issues and techniques relating to computer security should consult the excellent book by D. E. Denning[8].

A project to develop an implementation of the system described here is being sponsored by the Royal Signals and Radar Establishment, or 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, or MARI, and the Computing Laboratory of the University of Newcastle upon Tyne. The first stage of this project will result in the delivery of a prototype to RSRE in mid-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, enable the overall performance of the system to be evaluated, and most importantly, permit the impact of a mechanically enforced security policy to be observed in a realistic environment. If this stage is judged a success, a prototype implementation of the real system will follow. We hope that before long we will be able to report on the progress of this project and, in due course, on how well it achieves its security, usability, and performance goals.

Acknowledgments

This research was sponsored by the Royal Signals and Radar Establishment, Malvern, England. 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, a commercial product licensed by MARI, is the creation of Lindsay Marshall and Dave Brownbridge, while the remote procedure call mechanism is the work of Fabio Panzieri and Santosh Shrivastava.

Two anonymous referees directed our attention to a number of vexatious technical problems with some of our mechanisms and led us to make several improvements. The final form of this article benefited considerably from the careful reading, criticism, and advice of Pete Tasker of the Mitre Corporation. Sarah Rolph, also of the Mitre Corporation, suggested many improvements in the presentation of this material.

References

1. C. E. Landwehr, "A Survey of Formal Models for Computer Security," *Computing Surveys*, Vol. 13, No. 3, Sept. 1981, pp. 247-278.
2. M. H. Cheheyli et al., "Verifying Security," *Computing Surveys*, Vol. 13, No. 3, Sept. 1981, pp. 279-339.
3. J. M. Rushby, "The Design and Verification of Secure Systems," *Proc. Eighth ACM Symp. Operating System Principles*, Dec. 1981, pp. 12-21, (*ACM Operating Systems Review*, Vol. 15, No. 5).
4. G. Grossman, "A Practical Executive for Secure Communications," *Proc. 1982 Symp. Security and Privacy*, IEEE Computer Society, Apr. 1982, pp. 144-155.
5. D. Lomet et al., "A Study of Provably Secure Operating Systems," research report RC9239, IBM T. J. Watson Research Center, Feb. 1982.
6. D. R. Brownbridge, L. F. Marshall, and B. Randell, "The Newcastle Connection, or UNIXes of the World Unite!" *Software - Practice and Experience*, Vol. 12, Wiley Interscience, Dec. 1982, pp. 1147-1162.

7. S. K. Shrivastava and F. Panzieri, "The Design of a Reliable Remote Procedure Call Mechanism," *IEEE Trans. Computers*, Vol. C-31, No. 7, July 1982, pp. 692-697.
8. D. E. Denning, *Cryptography and Data Security*, Addison-Wesley, Reading, Mass., 1982.
9. R. M. Needham and M. Schroeder, "Using Encryption for Authentication in Large Networks of Computers," *Comm. ACM*, Vol. 21, No. 12, Dec. 1978, pp. 993-999.
10. J. M. Rushby and B. Randell, "A Distributed Secure System," tech. report 182, Computing Laboratory, University of Newcastle upon Tyne, England, Feb. 1983. (Also available from first author at SRI International.)

John M. Rushby is a computer scientist with the Computer Science Laboratory of SRI International. His research interests include the design, specification, and verification of secure systems and other computer systems that must satisfy stringent requirements. From 1979 to 1982, he was a research associate with the Computing Laboratory of the University of Newcastle upon Tyne, England, and from 1975 to 1978 he was a lecturer in the Department of Computer Science at Manchester University, England. Rushby received BSc and PhD degrees in Computer Science, both from the University of Newcastle upon Tyne, in 1971 and 1977, respectively.

Brian Randell is a professor of computing science at the University of Newcastle upon Tyne, where in 1971 he initiated a program of research on computing systems that now encompasses several major projects. From 1964 to 1969, he was with IBM, primarily at the IBM Research Center in the US, working on operating systems, the design of ultra high speed computers, and system design methodology. Before that, he worked for the English Electric Company, where he led a team that implemented a number of compilers, including the Whetstone KDF9 Algol compiler. Randell graduated in mathematics from Imperial College, London, in 1957.