# A Secure Distributed Operating System

Thomas A. Casey, Jr., Stephen T. Vinter
BBN Laboratories
10 Moulton St.
Cambridge, MA 02238

D.G. Weber, Rammohan Varadarajan, David Rosenthal
Odyssey Research Associates
301A Harris B Dates Drive
Ithaca, NY 14850-1313

## Abstract

This paper[1] discusses some issues in distributed system security, in the context of the design of a secure distributed operating system. The design is targeted for an A1 rating. Some new developments in formal verification methods are reported. Distributed system security is contrasted with single-host and network security, and described in the context of the TNI. Problems unique to distributed system security are discussed. An argument is made for implementing security features in higher layers, corresponding roughly to the session thru application layers of the OSI model. A new security policy, based on message-passing rather than reads and writes, is described. The SDOS design is summarized.

## 1 Project Summary

The objective of this project was to investigate multilevel security issues as they relate to distributed operating system design. The required deliverables included a security policy, a formal model, and a formal top level specification for an A1 class secure distributed operating system, and documentation of the issues and possible solutions that were discovered in the course of the project. In support of these objectives, we devoted some effort to producing a high-level design for a secure distributed operating system (SDOS). (The project report [BBN88] contains more details on all of the topics discussed in this paper.)

Working entirely in the abstract is difficult and often unproductive. Therefore we chose to investigate multilevel security issues in the context of an existing, operational distributed operating system. The existing system that we chose is Cronus [Schan86]. Cronus is an object-oriented distributed operating system, that can operate across a heterogeneous set of networks, hosts, and operating systems. Cronus is more than a distributed file system. It provides such features as authentication, object name to address binding, and dynamic object locating. It has a set of runtime features that provide good support for the writing of distributed applications, and it has an internal architecture that allows those features to be implemented reliably and efficiently. By basing our SDOS design on Cronus, and preserving as much as possible of its feature set and internal architecture, we felt assured that the resulting design would be usable and implementable as well as secure.

A distributed operating system is built on top of a set of single hosts, which are connected by a network. There has been a great deal of research in the areas of single host security and network security. It is our conclusion that, while distributed system security is related to these other two areas, and in fact depends on them for its success, it is a distinct area with a set of problems unique to it. Exploring the distinctions between the three areas has helped increase our understanding of distributed system security.

Our research has uncovered a number of problems unique to distributed system security, and identified some possible solutions to them. They are mentioned briefly here, without any supporting arguments, in order to define the scope of the discussion. They are treated in more detail in the subsections that follow.

A distributed system security policy must be defined in terms of message passing between active entities, rather than the traditional (Bell and LaPadula) [Bell76] read and write operations of an active entity (process) on a passive entity (file). The concept of a distributed TCB, running in the higher layers (above the communications and host operating system layers), must be supported by the security services provided by the communications and host operating system layers; that is, the distributed TCB must be implementable within the security restrictions imposed by those lower layers. The distributed nature of the TCB, particularly the fact that parts of the TCB can drop out of and later rejoin the system, as hosts go down and come up, presents some security problems, especially in the area of object replication. The access class range (system low to system high) can, in general, be different for each host and each inter-host path; further, within each host it becomes useful to talk separately about the access class ranges for active entities (processes), passive entities (objects), and messages sent to, and received by, the host. This has implications for the multilevel security policy. There are a number of covert channels that are brought into existence by the distributed operating system's attempts to make its distributed nature transparent to application programs and users, and its attempts to operate efficiently, minimizing delays due to inter-host communication. The desirable objective of having the SDOS operate across a heterogeneous set of networks and host operating systems presents some security problems, especially when the various networks and hosts vary in the degree of assurance of their security features.

In the area of formal specification and verification several significant results were achieved. Our basic goal was to write the Formal Top-Level Specification of the design of SDOS, and to prove formally that it satisfied multi-level security. To a large extent, this goal was met.

We based our definition of multi-level security on an emerging theory of information flow security being developed at ORA [Ulysse87]. This theory defines information flow in terms of the deductions that can be made about unseen (higher security level)

[1]The work reported in this paper was supported by the Air Force Systems Command at Rome Air Development Center, under contract F30602-85-C-0056.

events in a system's history. A basic result of that theory is the discovery of a *composable* security property: two subsystems having the property can be hooked together to form a larger system also having the property. This fact can be used to add MLS services and MLS hosts to a distributed system in a secure manner.

Our work in this project has extended the theory of information flow security in two main ways:

1. We have proved theorems that enable one to break the primary security property into simpler sub-properties;
2. We have developed a technique for demonstrating the simpler sub-properties using the Gypsy Verification Environment.

Some of the results of this project's research into formal methods are discussed in [Weber87].

### 1.1 Design Summary

SDOS, like Cronus, is an object-oriented [Jones78] system. Objects are instances of abstract data types. The definition of a type includes the set of operations that are possible for objects of that type. There is a hierarchy of types: types inherit operations from their parents. Clients (processes acting on behalf of users) access objects by invoking operations on them. The invocation of an operation is the only way to access an object. Operations are implemented by object managers. A manager hides the internal representation of an object from a client, and provides a precisely defined high level interface to the object. All resources in the system are represented by objects, and all operations are carried out as described above.

The SDOS TCB consists of the kernel, and a set of trusted managers that provide system services. The trusted managers operate according to the object-oriented abstraction described above. Since the kernel is the entity that implements the object-oriented message delivery service, that service is not available for use within the kernel.

The kernel consists of the message switch, the locator, the process manager, the security database, the object database, and the process table.

The trusted managers include the file manager, catalog manager, authentication manager, and trusted interface process.

A more detailed description of the SDOS design can be found in Section 3.

### 1.2 Distinction Between Network and DOS Security

In our view, a distributed system is different from a network, and thus distributed system security is different from network security. The difference is that a distributed system is built on top of a network. It is implemented in and above the higher layers of the OSI model. A distributed operating system (DOS) provides runtime services that support the running of distributed applications. It attempts to identify those functions that are common to most distributed applications, and implements them in the operating system, relieving application programmers of the task of implementing them in each application. Some of the services provided by Cronus (the DOS on which this project was focused) would be identified with the three highest layers of OSI (session, presentation, and application); others would be located in still higher layers. The DOS attempts to hide the distributed nature of the environment from application programs. For example, it provides the same interface for accessing local and remote data objects. A security policy for an SDOS must be stated in terms of the subjects, objects, and operations implemented by the higher layers in which the SDOS exists, and not in terms of the entities of lower layers.

It is useful to put this discussion into the context of the terminology and concepts in the TNI [TNI87]. Section I.3.2 of the TNI describes two network views: the interconnected accredited AIS view and the single trusted system view. An SDOS is more closely related to the latter than to the former. However the collection of underlying hosts and networks upon which the SDOS is built may appear to conform more closely to the interconnected accredited AIS view. The nature of this hybrid view will be made clearer in the sections that follow.

The discussions of connection-oriented abstraction and subjects and objects in section I.3.2.2 are related to entities of layers lower than SDOS. (In the SDOS design, connections are owned by TCB partitions and are used for communication with their peers on other hosts.) Section I.4.3 mentions four types of security policies that may be supported by a network component: mandatory, discretionary, supportive, and application. It gives, as an example of an application security policy, a policy supported by a DBMS that is distinct from that supported by the underlying system. It goes on to say that application level policies will not be considered further in the TNI. We consider the relationship between the SDOS security policy and that of the underlying network to be similar to the relationship between the DBMS security policy and that of the underlying system: from the point of view of the network, the SDOS security policy is an application policy that is distinct from that supported by the underlying network.

Further, in some cases the SDOS security policy is in conflict with that of the underlying network (as outlined in the TNI), and therefore the SDOS layers must be privileged (i.e., they must be part of the TCB), so that they can enforce the SDOS security policy. The legitimate communications between these privileged layers must not be hampered by the network security policy. One example of such a conflict is the set of restrictions in B.4.1 of the TNI, two of which are that a subject is confined to a single network component and that it may directly access only objects within its own component. The SDOS deliberately tries to mask all distinctions between components (hosts). At the level of abstraction of the SDOS, a subject is logged in to, and authenticated for, the entire SDOS, and may access all objects in the SDOS, subject only to the restrictions of the SDOS security policy. This policy is one which prevents information flow from high secrecy to low secrecy entities and from low integrity to high integrity entities, where entities are the subjects and objects implemented by the SDOS. This discussion is continued in the section entitled Distributed TCB, below.

### 1.3 TCB Boundaries

In a traditional single-host secure operating system, the TCB begins at the top of some layer, and extends down through all intervening layers to (or into) the underlying firmware and hardware. Thus we think of the traditional TCB as having only an upper boundary The TCB must protect itself against untrusted (and possible hostile) software running above this boundary.

In contrast, the partitions of a distributed TCB (i.e., the parts

running in each host) each have both an upper and a lower boundary. In the general case, the partitions must exchange messages via some untrusted communications medium, and they must use some technique (most likely cryptographic) to protect the secrecy and integrity of the data that they transmit over the untrusted medium. Thus, by default, the lower boundary of each TCB partition is located at the beginning of the untrusted medium. The TCB must protect itself aginst possibly-hostile entities (either hardware or software) operating below this lower boundary.

The default location of the lower TCB boundary could be at a number of different places, depending on implementation details. For example, it could be at the beginning of an unprotected wire or an antenna. It could be at the host side of a hardware device driving the wire or antenna. If one or more of the lower layers of communication software run in a front end processor rather than in the host, the lower TCB boundary could be at the interface between the host and the front end (provided that no security-related functions are implemented in the lower layers).

It seems clear that the location of the lower TCB boundary should not be allowed to be determined by default, but rather that the best possible location should be chosen for it, taking into account a number of factors. These factors include the objective of minimizing TCB size, thereby maximizing assurance and minimizing implementation and verification cost; and choosing the architecturally best layer in which to locate each security feature. If the lower TCB boundary location is chosen such that some lower layers are outside the TCB but still within the host, some implementation technique (dependent on the host operating system) must be used that will protect the TCB from these lower layers even though the TCB calls them.

### 1.4 Object References

In a single-host system, object references (for example, file reads and writes) are typically handled by the TCB because the I/O system and file system are in the TCB. (Even if the full I/O and file systems are not in the TCB, some primitive object-management system, on which fully-functional I/O and file systems can be built, is in the TCB.) In a distributed system, on the other hand, object references are typically handled by untrusted processes on the hosts involved, which exchange messages with each other via the TCB. (In Cronus and SDOS, the process that issues the request on behalf of a user is called a client process, while the process that responds to the request, on the remote host where the object is located, is called a manager process.) The requirement for a two-way exchange of messages between the untrusted processes can sometimes cause security problems, because of mismatches between the access classes of the two processes and the object being referenced.

Consider, for example, the case of a high-secrecy process attempting to read a low-secrecy object. This is, on the face of it, a legal operation. In the single-host case, the read is implemented entirely within the TCB. In the distributed case, the read could involve the sending of a message from a client process on one host to a manager process on another host. Consider the constraints on the access class of the manager process. If it is of lower secrecy than the client, then the sending of the read request is in violation of the SDOS security policy (which forbids the flow of information from a high secrecy entity to a low secrecy entity). If the manager process is of higher secrecy than the client process, then the response to the read request would be in violation of the security policy. Clearly if there is to be a two-way exchange of messages between the single-level manager and client processes, they must have identical secrecy classes. But now consider the relationship between the secrecy classes of the manager process and the object(s) that it manages. If the manager has a higher secrecy class then the object, then it may read but not write the object. On the other hand if the manager has a lower secrecy class then the object, it may write but not read the object. Clearly if the manager is to be able to both read and write the object, the manager and object must have identical secrecy classes. (The analogous argument for integrity [Biba77] classes also applies, but it is omitted here for the sake of readability.) It seems that clients can communicate only with managers having access classes identical to their own, and that managers can both read and write only those objects having access classes identical to their own. Thus, in the absence of some solution to this problem, clients can access only those objects that have access classes identical to their own. This is inconvenient.

This problem arises from the replacement of object references via the TCB, in the single host case, by object references via unprivileged manager processes, in the distributed case. We see two possible solutions to this problem: move the manager process into the TCB, or provide a set of manager processes (potentially multiple instantiations of the same executable code) at and above the access class of the object. The former has the drawback that an object manager is inherently an application program, and it is an objective of the SDOS design (and a feature of Cronus) to allow users (or at least using organizations) to define new object types and implement managers for them. The latter has the drawback that it leads to an unmanageably-large number of manager processes: one for each of the possible client access classes. Our design allows a using organization to choose either of these solutions, according to their own requirements and resources. We neither require nor forbid the use of multilevel-secure (MLS) object managers to support multilevel object types. An organization that has the expertise and resources to implement MLS object managers may do so. The alternative of using single level managers, one for each client access class, is also available. The problem of host operating system limits on the number of active processes is addressed by providing for the dynamic activation of manager processes in response to requests from clients. The resulting performance problems are addressed by providing a least-recently-used manager deactivation algorithm similar to traditional page replacement algorithms, and by using various ad-hoc techniques to speed up the activation of manager processes.

### 1.5 Distributed TCB

We have identified a number of problems unique to distributed system security, and some possible solutions to them. We have chosen the term "Distributed TCB" (DTCB) to refer, collectively, to those solutions. The term was chosen deliberately to emphasize the difference between the DTCB and the Network TCB (NTCB) described in the TNI. The DTCB runs in higher layers than the NTCB. Conceptually, the DTCB could be implemented either on top of an NTCB, or on top of a collection of interconnected accredited AIS. (In practice, the implementation of the latter would be the more difficult.) The distributed oper-

ating system implements a number of abstract entities (subjects and objects); the DTCB enforces a mandatory security policy that controls the flow of information between those abstract entities.

Given that a partition of the DTCB resides in each host, it is useful to consider the meaning of the access class range of each host. If a host is assigned system-high and system-low access class limits, what do they mean? Do they delimit the range within which that host's DTCB partition is trusted? Or are they intended to place limits on the range within which unprivileged processes may run, or within which objects may be created? We will use an example to motivate the answer to this question.

Consider a very high speed computer that is dedicated to making weather predictions and continually updating a database containing world-wide weather information. Although accurate weather information could be of some value to an opponent, it is inherently of low secrecy. Further, it must be classified at least as low as the lowest clearance of any of its legitimate users. So in this example we will assign the weather information an access class containing a level of confidential, and an empty category set. The intention is that only the confidential weather information will reside on this host, but that clients of higher secrecy classes, on other hosts, will be able to request and receive weather information.

It is not possible to state this policy using a single access class range for each host. In fact, we need three ranges for each host, plus one for the system as a whole. System-high and system-low will refer to the SDOS as a whole, and will limit the range of information that can exist in the system, and also limit the values that the per-host ranges may have. On each host, there will be message-high, message-low, process-high, process-low, object-high, and object-low access class limits. Message-high is an upper limit on the messages that may be sent to the host, while message-low is a lower limit on the labels that may be put on messages leaving the host. Process-high and process-low limit the range of unprivileged processes that may run on the host, while object-high and object-low limit the range of objects that may reside in the permanent storage managed by the host.

In the case of the weather system, both object-high and object-low would be set to confidential. If the weather database has a MLS manager, then both process-high and process-low would also be confidential; however if there are single-level managers for the database, process-high would be set to the highest secrecy level from which client requests will be accepted. Message-high would be set to system-high of the SDOS, or to some lower value, if dictated by weak physical security at the weather host, or by "Yellow Book" [DoD-G85] restrictions.[2] Message-low would be set to the lower of the two values of process-low and object-low. (Message-high and message-low actually correspond quite closely, in their effect, to the system-high and system-low parameters of a single-host secure system.)

The effect of having three ranges per host is to allow the placing of separate constraints on unprivileged code and the DTCB partition in each host. The DTCB partition is trusted to process information of higher secrecy than any unprivileged program on the host. This allows the expression of complex policies, taking into consideration the different physical security and need-to-know characteristics of each host in a distributed system.

[2]Note that this would prevent very-high-secrecy clients from requesting weather information. This is not an unreasonable restriction, if the location for which weather is being requested could compromise a critical mission, if revealed to an opponent through weak physical security at the weather host.

## 1.6 SDOS Covert Channels

Any system will have covert channels in it as a side effect of its implementation. There are, however, a number of covert channels that occur in SDOS only as a result of its distributed nature. These channels, or similar ones, would probably be found in any distributed system. There are two broad classes of channels that are of interest here. The first class consists, in general, of the ability of a low secrecy process to detect the existence or recent use of an object, coupled with the ability of a high secrecy process to modify those pieces of information. The second class consists of ways in which an untrusted process could signal to a confederate who is listening to an insecure medium within the network underlying the SDOS. These covert channels, like most others, can be fully understood only in the context of a detailed description of the design in which they occur, and such a description is given in the final SDOS report [BBN88]. The nature of these channels will be briefly sketched here.

When a client invokes an operation on an object, the system must locate the object before carrying out the operation. In our design, this involves a broadcast to all hosts, requesting a response from the one on which the object resides. This causes a real-time delay for the requesting client, and possibly a throughput problem for the system as a whole if these broadcasts are done with unnecessary frequency. To solve this performance problem each host maintains a cache containing the locations of recently-used objects. This cache introduces two covert timing channels. (It is suspected that under ideal conditions the existence of the cache would allow the operation of sequencing channels.)

A high secrecy client can read a low secrecy object, causing its location to be placed in the cache; then a low secrecy client on the same host can invoke an operation on the object and detect, by measuring the time delay, whether or not the object's location was in the cache. This allows a covert communications protocol to be implemented, using a previously agreed upon group of objects to transmit the 1's and 0's of a message.

A high secrecy client can create and delete high secrecy objects at will. If the write-up operation is allowed by the SDOS security policy, a low secrecy client could detect the existence or nonexistence of a high secrecy object by repeatedly attempting to write it. An existing object would have its location cached after the first attempt, resulting in lower time delays for subsequent attempts. A nonexistent object would never have a cache entry and would always cause a long time delay before the operation completes (even when success or failure of the operation is not reported to the low secrecy client).

Various solutions suggest themselves: disallow all write-up operations; introduce random delays into the completion of selected operations; eliminate the cache completely; classify cache entries and allow only those processes who are cleared to read them to benefit from their contents. These solutions all have obvious functionality and performance disadvantages.

The second class of channels involves the ability of untrusted software to signal by modulating information visible on an untrusted medium. A distributed application, having a global view of its goals, and possibly some foresight into its future behavior, could make good use of the ability to influence the behavior of lower layers in ways that would globally optimize the use of

scarce resources such as communications bandwidth. (Lower layers can, at best, make local optimizations based on past history.) However, the ability of a higher layer to influence the behavior of a lower layer by passing detailed control information across the interface has the drawback that it allows untrusted software to deliberately modulate some of the information that is necessarily clearly visible on untrusted media—message destinations, routings, lengths—the sort of information that traffic flow analysis looks at. Current layered protocols provide some ways in which higher layers can direct the behavior of lower layers to achieve optimization. Current work in DOS research suggests that layer interface enhancements allowing the passing of even more control information to lower layers would be useful. However the closing of these covert channels would require that downward information flow be shut off by providing even less ability to pass control information than exists in current protocols. This conflict is unresolved, and it is one of the topics suggested below for future research. The work of Girling [Girl87] on Covert Channels in LAN's is applicable to this problem.

Solutions to these covert channel problems all involve tradeoffs of functionality and performance on the one hand against security on the other hand. The parameters involved in the tradeoff (the value to application programmers of a particular item of functionality, the performance implications of any design change, and the bandwidth of a particular covert channel) can only be estimated during early design stages. The choice of solutions should be put off until the prototype implementation stage.

## 1.7 Problems Arising from Heterogeneity

### 1.7.1 Heterogeneous Networks

It is a design objective of SDOS that its hosts be able to communicate with each other over heterogeneous networks, rather than being restricted to one particular set of network hardware and software technologies. The main benefit of the use of heterogeneous networks is to allow the use of existing networks. Many existing networks are not secure. Thus the objective of network heterogeneity becomes the objective of being able to maintain security within SDOS while using insecure networks for communication between SDOS hosts. Encryption can protect higher layer data, but some lower layer information must travel the insecure networks in the clear, for the simple reason that header information of the lower three layers of the OSI model (or the equivalent thereof) must be interpreted in the course of message processing within each insecure node of an insecure network (for example, ARPANet IMPs). The information handled by those layers is subject to traffic flow analysis, and also provides a potential covert channel, as outlined in an earlier section. These problems are addressed, to some extent, by the SDOS design, but further research would be beneficial.

### 1.7.2 Heterogeneous Hosts

It is a design objective of SDOS that the host-resident software be portable to various processors and operating systems. The benefits of this heterogeneity of hosts include the ability to provide varying amounts of processing power to meet the needs of various applications, the preservation of investments in existing OS and application software, and the ability to take advantage of new processors and operating systems as they become available from manufacturers. (Naturally, the hosts on which SDOS is built must have hardware and operating system features capable of supporting multilevel security, which implies a rating of B2 or above as defined by [DoD85].)

The problems arising from the heterogeneity objective fall into two areas: implementation costs and architectural questions. There are two kinds of implementation costs: the cost of actually porting SDOS to a new host, and the cost of designing an interface that allows it to be ported easily. It is our feeling that, while the former cost should be borne by the various using organizations that have a requirement to run SDOS on particular hosts, the latter cost (portable design) should be a part of the initial development of SDOS (in other words, we believe that portability to a heterogeneous set of hosts is an important requirement).

The architectural questions that are raised by heterogeneity could be viewed as merely higher level portability issues, although they do have a more fundamental impact on SDOS functionality. These questions have the common property that they involve decisions between using security features in the multilevel secure host operating system or ignoring those features and implementing analogous, but slightly different, features in the higher SDOS layers. The economic advantage to using existing features is obvious. The disadvantage is that various parts of the SDOS security feature set become constrained by the details of the host operating system security features. One example is the size of access class labels: the number of different levels and categories that can be expressed in labels. These architecture/cost tradeoffs must be made, even if SDOS is designed to run on only one type of host. The requirement for heterogeneous hosts makes these tradeoffs even more difficult.

The cost tradeoff problems discussed in this section are far from fundamental topics for computer security research, but they are important nonetheless. Users of secure systems have as their fundamental objective the doing of the job that is their organization's reason for existence. Security is merely one of the constraints within which they must operate; cost is another constraint. Design decisions involving cost must be given serious attention if a secure systems project is to succeed.

## 1.8 Problems Related to Object Replication

Data replication, or the maintenance of several copies of data on different hosts, is the primary technique for achieving data availability in the event of host or communication failures. Data replication is achieved by coordinating reads and writes across all copies of the data. Much research has been directed toward developing algorithms for supporting data replication. The algorithms vary with respect to the degree of availability, consistency and performance that they offer. The consistency, availability and performance requirements of distributed, highly available applications also vary. The replication scheme that satisfies best an application's data requirements is dependent on the application. As a result, replication mechanisms are commonly placed in the resource management software component that understands the semantics of the data. In object-oriented systems such as SDOS, these mechanisms are placed in object managers, and the object is the granularity at which replication is supported. If replication mechanisms were placed in the Object Database instead of object managers then it would be impossible to tailor the management of replicated objects in an application-specific way.

In addition to the data stored in an object, there is a significant amount of administrative information that needs to be associated with an object. For example, all objects in SDOS have access control lists and security labels. Additionally, the set of hosts where copies of a replicated object reside needs to be maintained. Objects that are replicated must have this administrative and security-related information replicated as well, both to avoid bottlenecks and to provide high availability.

Security labels are stored separately from objects in the Security Database within the TCB. Access control lists are maintained with objects in the Object Database. The design decisions that led to this placement reflected the considerations of the desired functionality and criticality of the information. We have come to understand that there are additional trade-offs to consider when the objects are replicated.

When the adminstrative information is maintained with the object by the object's manager (in the case of SDOS, the access control lists), the application developer may be given control over the policy that governs how copies of the information are maintained. In the case of access control lists, a variety of different policies are conceivable. For example, strict consistency would ensure that changes to an ACL are propagated to all copies atomically. However, this would prevent an administrators ability to remove a client from an access control list when some copies are unavailable. Regardless of the particular policy adopted, placing the information with the object provides the maximum amount of flexibility possible for setting the replication maintenance policy.

Security labels cannot be maintained by object managers because they are critical to the enforcement of the mandatory security policy and they are used by other kernel components. Placing them in the Security Database forces the security database to support their replication. This has several implications:

- The security database (i.e., the kernel) must implement algorithms to maintain consistency between the copies of an object's label. While it is necessary to implement a single policy for maintaining security label replicates, placing these algorithms in the kernel complicates it. It also requires the security database to maintain a list of where all copies of an object (or more specifically, its labels) reside.
- The policy setting the consistency and availability of objects may differ from the policy determining the consistency and availability of their labels.
- The replication of objects must be coordinated with the replication of their labels. As a result, both the Object Database and security database must support operations to replicate and dereplicate an object and its label, respectively. Furthermore, only the object database may invoke the security database replicate and dereplicate operations; otherwise, label and object copies could be created or removed in an uncoordinated fashion.

The policy on the replication of security labels is loose consistency: updates to security labels are propagated as permitted by the connectivity of hosts. Since only the System Manager may update security labels, we believe this policy is appropriate despite connectivity problems. This policy also simplifies the replication management performed by the Security Database.

Section 2.3 discusses the complications security features introduce for replication management based on our experience with the formal specification of the Catalog Manager.

# 2 SDOS Security Policy

The SDOS security policy was formulated in response to perceived threats to security. The resulting policy rules can be divided roughly into three groups:

- A discretionary policy, designed to control the use of SDOS' abstract operations on the basis of client identities;
- A mandatory policy, controlling the flow of information on the basis of DoD security levels;
- A configuration policy, defining the system's security "configuration" in terms of a set of "policy parameters", and controlling the modification of those parameters, both by system users and by changes to the network connectivity.

The mandatory policy, in turn, is composed of two parts:

1. A policy controlling message passing between entities. Each DoD security level indicates both secrecy and integrity access classes. Each system component has a set of levels, called its *label*, which records the levels of data the component is authorized to handle. A component may only send a message with a level from its label, and it may only receive a message if the message's level is in its label or can be upgraded to be.

   The system components are divided into two groups: those that are certified to handle multi-level data (MLS entities), and those that are not. Those that are not have singleton labels.

2. A policy controlling information flow within each MLS entity. The policy we chose is *composable*, which means that the controls on local information flow, taken together, imply controls on global flow for the entire system.

Several aspects of this policy are noteworthy. First, the mandatory and discretionary policies are cleanly separated. The discretionary policy is stated in terms of abstract operations of the object model that SDOS supports. The mandatory policy refers to the message passing operations which are used to implement the object model.

Second, it is a global policy, giving requirements for the entire system rather than for individual hosts.

In the following sections, we discuss some ways in which policy concerns in SDOS differ from those in a centralized system.

## 2.1 Restriction: Hook-up Security

The part of the mandatory policy that controls message passing eliminates direct downgrading of data. However, it is the other part, the policy for each multi-level secure entity, that prevents information compromise via covert channels. That policy must guarantee that the levels an MLS entity assigns to each message are not underestimates of the sensitivity of the message's content. We describe that policy now.

We have used the multi-level security policy of McCullough [McCull87]. That policy defines security in terms of information flow. Information flow is defined in terms of the deducibility of facts about the history of inputs received by a component. A

system component is secure if it does not allow information to flow from high security levels to lower ones.

The McCullough policy goes beyond this, however. It has the additional property of *composability*: two MLS components, when hooked together, form a larger component that is also MLS. A component with this property is sometimes called "hookup secure", but in this paper, we have called these components *restrictive*. (Other properties exist that limit deducibility and are also *composable* in the above sense.)

The SDOS policy requires that the collection of all MLS entities be restrictive. Since restriction is a composable property, it is sufficient to verify that each MLS component is restrictive. The fact that security verification can be decomposed in this fashion is a tremendous advantage when trying to verify security for a distributed system such as SDOS.

The fact that MLS components must be restrictive is also an advantage when a secure system is to be extended. Extensions may include either new hardware or new software. In SDOS, extensibility means adding new object managers to the system to define new classes of objects and new abstract operations on those objects. If a new component is added to SDOS, and if it is verified to be restrictive with the same degree of assurance as the original system, then adding the component will create a new system that is also restrictive. The information flow security of the new system can be guaranteed with the same degree of assurance, and without a re-verification.

Our work on SDOS is almost certainly the first attempt to verify the property of restriction for a secure operating system.

## 2.2 Read-down and Write-up

In a distributed system, a "read" operation will often not be considered a fundamental operation. A "read" may be composed of a pair of messages: a request-to-read, followed by a response. If reading data from a lower level entity is to be considered secure, as it is in Bell-LaPadula, it is because the request to read message causes no harm. However, the fact that a request to read has been sent is in general as sensitive as the reader itself, and in general it will downgrade information.

There are two approaches to this problem. One may demand assurance from the message's sender that the request is overclassified, and can securely be sent at the lower level. On the other hand, one may demand assurance from the message's receiver that the information in a high-level request to read will not be used for any purpose other than initiating the read operation itself. Either approach can be developed. For SDOS, we have used only the former approach: a request to read can be downgraded in some cases if a human being decides that it is secure to do so.

With regard to write-up's, we have taken the approach that the existence of an entity will in general be as sensitive as the entity itself. This was done to allow clients to delete entities at their own level. However, the approach has a drawback: for a "write-up" operation, in which the client's level has to be dominated by the object's, the location of the object cannot be found if the search is carried out at the client's level.

We provide a "write-up mode" in which write-up operations are treated as special. In this mode, a client chooses to upgrade an operation so that it can be carried out completely at the object's level. The client does not know what level that is, and gives up any hope of seeing a meaningful acknowledgement. Once the request is upgraded, though, the system can find the object's level and can carry out the operation. It would be insecure to give an acknowledgement indicating, for example, that the request had arrived at some manager, because it would convey to the client that the higher level object exists and has been located. Modulating an object's existence could then be used as a covert channel. As a result, it is not possible to build a reliable write-up operation.

## 2.3 Object Replication

Section 1.8 considered the impact of object replication on the management of security labels. In this section we consider the impact of security on the management of replicated objects. Our formal specification of the design of the Catalog Manager, which maintains replicated directories, provided an opportunity to experiment with a particular replication management (i.e., concurrency control) mechanism. We first consider this experience, and then generalize these results to more general replication management mechanisms.

The catalog manager locks out all but one concurrent update to a replicated directory. A significant fact emerged from the verification exercise: concurrency control as used in the catalog manager, will not interfere with multi-level security. The following two points explain this:

- When a client invokes an operation on a higher-level object, the SDOS message switch does not try to guarantee that the client can use manager services at the its own level. Instead, the client must choose to let the invocation be upgraded to the object's level, and surrenders any expectation of an acknowledgement from the object manager. Because such invocations either fail or are upgraded to their object's level, the catalog manager is guaranteed to receive only invocations at levels that dominate the level of their object.

- The only invocations that can succeed at levels strictly greater than their object's level are requests to read ("Lookup") a directory. In the catalog manager as specified, both read and modify operations are atomic when applied to a single directory replica. Therefore, a read (Lookup) will never be blocked because a distributed update is in progress.

The concurrency control mechanisms used by the Catalog Manager, then, only block operations which write directories at the same level as the concurrent invocations. Since the clients of these requests are blocked by the activity of clients at the same level, the concurrency control mechanism does not act as a covert channel.

In general, since accesses to an object may come from clients at many levels, a concurrency control mechanism that prevents access of one client because of accesses of other clients at higher levels will be insecure. The specific problem that arises is that clients reading down to objects may cause lower level (writing) clients to be blocked, and this execution delay is a covert channel. For example, read/write locking is insecure, because higher level clients can lock an object for read and cause lower level writing clients to be blocked. In contrast, voting algorithms do not block read-downs, and therefore are not insecure.

## 2.4 Configuration Policy

The need for a configuration policy follows naturally when considering security in a distributed system context. The system's

security is configured by a set of values called here *policy parameters*. These parameters include the security labels of system components, discretionary access control lists, and the results of specific choices such as whether audit records are kept, and whether the system can be extended with new trusted software. The system's security configuration may change with time. The configuration policy constrains who may cause these changes, and what consistency is required between security configurations on different SDOS hosts.

# 3 SDOS Design

## 3.1 Overview of Design

The SDOS TCB consists of the kernel, and a set of trusted managers that provide system services.

The kernel consists of the message switch, the locator, the process manager (so-named for historical reasons—it is an internal part of the kernel, and not an object manager), the security database, the object database, and the process table.

The trusted managers include the file manager, catalog manager, authentication manager, and trusted interface process.

The function of each of these TCB components is briefly described below. More detailed descriptions may be found in [BBN88].

- **Message Switch:** Routes messages between entities, both locally and remotely. Enforces the mandatory security policy governing the passing of messages between entities. Communicates with its peer message switches on other hosts, and cooperates with them in the passing of messages and the enforcement of the security policy.
- **Locator:** Locates objects that do not reside on the local host. Provides this service only to the local message switch. Maintains a cache containing the locations (remote host IDs) of recently used remote objects. Remote objects are located by broadcasting a request for the object to all hosts. If no positive response is received after a suitable interval, failure is reported to the message switch.
- **Process Manager:** Creates and destroys processes, and maintains (sets and shows) process bindings. Process bindings is the term for the set of information that includes a user's identity, and mandatory and discretionary access control attributes.
- **Security Database:** The collection of data needed for the enforcement of the mandatory security policy. This information includes, for each entity on a host: an access class label; a switch indicating single-class or multilevel-secure; for a replicated object, the number of replicas in the system; and for an object type, information about its manager, including: whether local managers exist, whether they are active, and the location of their executable code.
- **Object Database:** Provides storage for all objects that reside on the local host. Used by object managers.
- **Process Table:** Contains information about all active processes on the local host, including the process bindings. Maintained by the Process Manager. Consulted also by the Message Switch, when making mandatory access control decisions.
- **File Manager:** A multilevel secure manager, allowing the write-up and read-down operations. Also implements create, delete, open, and close operations.
- **Catalog Manager:** Provides an abstract space of symbolic names for objects. Translates from an object's symbolic name into its UID. (The UID is used to reference an object when invoking an operation.)
- **Authentication Manager:** Implements login and logout requests from interactive users. Sets appropriate process bindings for users logging in.
- **Trusted Interface Process:** Implements a trusted path between the system and an interactive user at a terminal. Maintains the state of the terminal, with respect to whether or not a user is logged in. Relays login requests to the authentication manager. Relays the requests of a logged-in user to other parts of the system. Could be called the Trusted Terminal Manager.

## 3.2 Enforcing Security

### 3.2.1 Location of Security Mechanisms

One of the fundamental decisions in the design of a secure system is the choice of locations for the implementation of security mechanisms. In the case of the mandatory controls in SDOS, the choice was rather clear. The mandatory security policy is based on message passing; the message switch is the entity which is responsible for the passing of messages; therefore the message switch is the natural place to implement the mandatory controls when the identity of a message's sender is involved. In general, MLS controls on message passing are enforced by every component of the DTCB, including the message switch, but also by other kernel components and by every trusted MLS manager.

For discretionary controls the choice was more difficult. The SDOS discretionary control scheme, which is based on that of Cronus, allows, in general, for a different set of discretionary control rules for each object type. The SDOS discretionary control scheme is summarized here and in the following section and discussed in greater detail in a companion paper [Vinter88]. Briefly, an ACL consists of a list of entries, each of which consists of some client identification information (details omitted here), and a specification of the set of operations which the bearer of that identification is permitted to invoke on the object. The set of legal operations is different for each object type. Therefore the set of operations that can be specified in an ACL entry is different for each object type, and the code that searches and interprets an ACL must be different for each object type. This argues for placing ACL interpretation in the manager of each object type. However, managers are assumed to have low assurance, being writable by users. The idea that it is acceptable for DAC to have lower assurance than mandatory controls has gained some acceptance recently, but to suggest that DAC be implemented in user programs of no assurance whatsoever would be going too far. In fact, we do not suggest this. Elsewhere in this paper, we discuss the possibility that using organizations would in some cases have to develop the expertise required to write multilevel secure managers. We here suggest that all using organizations that wish to extend the system by defining new types and writing managers for them will need to have the capability of writing managers of at least C2 assurance, in order to provide acceptable

discretionary controls. We have developed a technique for making it easy to implement managers that provide C2 assurance for discretionary controls on hosts that provide hardware rings usable by application developers [Vinter88].

The choice of location for the encryption mechanism was fairly straightforward. Encryption of data being sent out over an untrusted network is done in or near the IP sublayer of the Network layer. This is the lowest place in the OSI model where end-to-end encryption can be done across the internet. Encryption at a lower point would interfere with the operation of the network layer in untrusted network nodes. Encryption at a higher point would result in the passing of more unencrypted information in message headers, and require individualized encryption mechanisms for each of the higher layer protocols.

#### 3.2.2 Discretionary Controls

The SDOS discretionary access control mechanisms are based on access control lists. However, several aspects of their design distinguish them from conventional approaches, including their support of roles, nondiscretionary rights, direct operations, intermodule connection control, and proxies.

Discretionary controls are type specific, as each type defines the privileges clients may have to invoke operations. Clients are identified by a *principal* (user) and *project* (task or group). A client may be associated with several different projects (though always acting on behalf of exactly one), and in a different capacity in each project. For example, a client may be an operator for one application but a developer for another. The different capacities, or *roles*, in which a client acts determine the operations that are available to the client to access objects. Roles tend to have similar meaning across many different types, thus providing an aspect of uniformity to the type independence of SDOS access control.

The operations in an access control list are divided into discretionary and nondiscretionary categories. The degree to which an operation is discretionary reflects the extent to which its entries in access control lists may be modified. Discretionary operation entries in an access control list may be modified by a group of users for a type having the *Controlling Group* role. Nondiscretionary operation entries in an access control list may only be modified by the System Manager. Since intervention by the System Manager is expected to be rare, the extent to which modifications to nondiscretionary operation ACL entries may change is highly constrained.

Direct operations are operations which only may be invoked by a trusted Terminal Interface Process. Thus, operations which should only be invoked by humans can be protected from inadvertent or malicious invocation by application software.

Object-oriented systems invariably create instances of nested object invocations, where client A invokes an operation handled by manager B, which in turns invokes an operation handled by manager C. For example, a process authenticating itself causes the Authentication Manager to invoke a nested call to set its process bindings for discretionary access control. In many instances, the nested call should be made on behalf of (i.e., using the identity of) the original client. Clients may send proxies, or a highly constrained part of their identity, to managers, which can then act on behalf of that client's limited identity. Proxies constrain the behavior of malicious managers and assure that managers cannot take on illegal or forged identities.

Module interconnection controls refers to the control of which modules (clients) can call other modules (managers). For example, only the Authentication Manager should be able to invoke the **SetProcessBindings** operation for a process. SDOS discretionary controls allow a manager to determine the identity of the client, even when the client is acting on behalf of another client by using a proxy.

See [Vinter88] for a more complete discussion of the SDOS discretionary control design.

### 3.3 Host Operating System Security

SDOS, like Cronus, is a collection of higher layer software that is implemented on top of an existing operating system in each host. Unlike Cronus, which is implemented on top of a variety of commercially available operating systems having ratings of D through C2, SDOS must be implemented on top of a multilevel-secure host operating system. This is necessary in order to provide the required assurance that the SDOS security features cannot be tampered with. The Criteria mandate at least B2 assurance for multilevel security. The design objective of SDOS is an A1 rating. Thus, the host operating system(s) on top of which SDOS is implemented must have a minimum of a B2 rating, and ratings of B3 or A1 are more desirable.

#### 3.3.1 Advantages and Disadvantages of MLS Support

Operating systems having B2 through A1 ratings will have multilevel security policies built into them. These policies will all be different from the one designed for SDOS. Some may be similar, while others may be completely incompatible. The reason for building on top of a secure system is to benefit from its assurance. The policy comes with it for free, and we must decide what to do with it—use it or ignore it. The temptation is strong to use it, for economic reasons. This may or may not be possible, depending on the policy of the particular system chosen. Any decision to use the policy of an existing system will involve some changes to the designed SDOS policy. These changes may range from changes so fundamental as to be unacceptable, to changes in unimportant details.

A decision to ignore the policy of an existing system also has its drawbacks. The security mechanisms in the existing system will continue to operate and contribute to overhead, even if SDOS is built on top of it in such a way that the existing policy does not restrict SDOS (for example, by labeling all SDOS entities with the same access class label in the existing system's label set). This decision will also tend to increase the SDOS implementation cost.

#### 3.3.2 Desirable Host Operating System Properties

A secure operating system that is a good candidate on which to implement SDOS will provide the following:

- assured process separation — the ability to prevent direct interprocess communication that is not controlled by the system;
- non-interference with process operation — SDOS processes responsible for security must not be tampered with;
- stable storage — data needed for enforcing security, such as user authentication data, must be stored in a fault-tolerant and protected manner.

In addition, the good candidate will have a multilevel security policy that lends itself to being used to implement the SDOS policy, rather than being ignored.

### 3.4 Network Security

#### 3.4.1 Open vs Closed Networks

A closed network is one whose nodes and inter-node communications media are under the physical control of the using organization, such that their security can be assured, making it practical to implement multilevel security in the nodes. An open network, on the other hand, uses public communications media (such as phone lines or radio signals) and public nodes, such as those in commercial packet switched networks or in the ARPANet. Physical security of the nodes and media of a public network can obviously not be assured, so multilevel security is impractical. It is a design objective of SDOS that the system be able to maintain its own security even when it is operating over an open network. The reason for this requirement is a practical one: open networks are prevalent, and becoming more so every day. Closed networks are relatively rare, usually unavailable, and costly and time-consuming to construct and maintain.

#### 3.4.2 Encryption

Encryption is the usual solution to the problem of maintaining security of communications across an open network. Two classes of encryption are of interest here: link encryption and end-to-end encryption.

Link encryption protects data on an insecure medium that is being used for communication between two secure network nodes. Messages being relayed through several nodes to an ultimate destination are decrypted and re-encrypted at each intermediate node.

End-to-end encryption is used by higher layer entities to protect data from untrusted lower layers or from untrusted nodes in an open network. Messages are encrypted by the sending higher layer entity, decrypted by the receiving higher layer entity, and remain encrypted while moving through lower layers and intermediate network nodes.

End-to-end encryption has the advantages that the encryption and decryption are only done once for each message, and that layers below the encrypting layer, and intermediate network nodes, do not have to be trusted. It has the disadvantage that message headers for the layers below the encrypting layer travel the network in the clear. The information in these headers is subject to traffic flow analysis. Further, as suggested earlier, there is a potential covert channel if untrusted software above the TCB is able to exercise control over lower layer operation in ways that would modulate the lower layer header information.

Link encryption has the advantage that all information carried on the insecure medium is protected. It has the disadvantage that all layers down to the physical layer, and all intermediate network nodes, must be secure. In other words, link encryption implies a closed network.

Since the use of open networks is a requirement for SDOS, we have chosen to use end-to-end encryption, in the IP sublayer of the Network layer. This is the lowest point at which end-to-end encryption can be done without interfering with the activities of the network layer in intermediate untrusted nodes of an open network. This allows layers below the point of encryption in each SDOS host to be untrusted. Layers at and above the point of encryption, up through the TCB/application interface, must be in the TCB.

## 4 Formal Methods

One of the goals of this project has been to formally verify that the SDOS design meets the requirements of the SDOS security policy. This would give high assurance that the design is "secure". Much of our work toward this goal has focussed on verifying the part of the mandatory policy requiring constraints on information flow.

We have formalized the design of SDOS as a program in Gypsy [Good78]. To prove, using the Gypsy methodology, that a program meets it requirements, one must express those requirements as assertions that are true at particular times during the execution of the program. We found, however, that the information flow constraints of the SDOS policy cannot be directly expressed in this way. Other approaches were needed.

### 4.1 A New Security Methodology

Our emphasis on mandatory information flow security is a result of the emergence of a new methodology for security verification. The aim of this methodology is security verification through analysis. In other words, large designs can be decomposed into smaller ones, and the security of the larger can be inferred once the properties of the smaller are known. This approach has obvious merits, but it is only recently that it has been applied to formal specifications for information flow security.

The work of McCullough [McCull87] is a particular case of this new methodology. In his work, system components are defined in terms of their possible behaviours; the approach he used simplified and slightly modified the approach of CSP [Brook84]. The hook-up, or composition, of two components is defined as in CSP. McCullough searched for, and found, a property that captures many desirable features of information flow security and is also a *composable* property: the hook-up of two components with the property is a new component with the property. We have called his security property *restrictiveness*, or *restriction*.

The verification work done for the SDOS project ties into the new methodology. The SDOS security policy requires that the entire trusted part of the system be restrictive. We have endeavored to show that the multi-level secure processes that comprise SDOS are each restrictive, so that the restrictiveness of the entire system can be inferred from composability. However, the problem of demonstrating the restrictiveness of each MLS process remains. One way in which we have handled this problem is to find other, simpler, properties, which when taken together, imply the restrictiveness property. These simpler properties are then proved, using Gypsy, for each component. We needed to develop special techniques for proving some of the simpler properties using Gypsy.

Other than restrictiveness, we formally defined several properties that are "security-like", in the sense that they also limit deducibility, and hence, information flow. Of primary importance is the property called *weak non-interference* (WNI). The WNI property limits deducibility in a way that is similar to the Goguen-Meseguer model [Goguen82]. However, WNI is both weaker than restriction, and not composable. By conjuncting

several other simple properties with WNI, however, we can infer restriction. A proof of this claim is given in the final SDOS report [BBN88], and is outlined in [Weber87].

As stated earlier, Gypsy is ill-suited to direct verification of properties such as restriction and WNI. Each is a property of the form: "Given any history $a$, there must be a history $b$ such that $P(a, b)$ holds." Essentially, one is required to show the existence of particular histories of a component. Gypsy embedded assertions, though, state requirements of individual histories, taken in isolation. They never directly imply the existence of any history. However, simply changing the specification language was unlikely to solve the problem: other popular specification methodologies used for proving invariant properties of state machines would fare no better.

We developed a technique for proving that WNI holds for a design expressed in Gypsy. The technique does not supply embedded assertions for the design itself, but rather, it first transforms the design into a new form, and then supplies assertions about the new form that imply WNI. The restrictiveness of a design is to be inferred from the fact that it satisfies WNI, plus other simpler properties. This "program transformation" technique essentially shows that an alternate history exists by constructing it from the actual history. The transformed Gypsy design contains the state variables and control structure of the original design, but replicated, so that the two histories can be constructed in parallel.

Decomposing restrictiveness into simpler properties can now be seen as an advantage, since these simpler properties turn out to be easier to handle in this program transformation technique.

In verifying the information flow security of various SDOS components, we found that the definition of security as restrictiveness may not always be appropriate. We needed generalizations of the property to permit the following:

- limited downgrading of information via covert channels;
- special protocols used in communication between components;
- assumptions about the boundary between the assured system components and process and users with limited or no assurance.

This project achieved some successes in these directions. However, the subject is far from closed.

# 5 Future Directions

The work described in this paper has only made a start toward the development of a secure distributed operating system. There is more work to be done, in a number of areas, ranging from the very theoretical to the very practical. Several of these areas are described below.

## 5.1 Prototype Implementation

The development of a prototype SDOS would have a number of useful results. It would allow the concepts developed in this project to be tested and proved in a practical setting. It would uncover any ideas that look good on paper but prove to be impractical during implementation. It would allow measurements to be made of performance-critical operations and of covert channel bandwidths, and experimental attempts to be made to speed up the former and slow down the latter. It would allow experimental applications to be developed using the facilities provided by SDOS, to determine their suitability for use in practical applications. It would be a first step toward the deployment of an operational secure distributed operating system.

## 5.2 Alternative Layering Schemes

Layering, abstraction, and data hiding are thought to be good software design methodologies, and their use in secure systems is mandated by the Criteria. In any layered software system that is undergoing evolution and change, there arise occasions where the addition of some function or the improvement of some existing function requires the passing of information and control across layer boundaries in ways that were not intended when the boundaries were originally defined. Often some compromise of the strict layering rules will be made to avoid the need to completely re-modularize the system and redefine the layers to accomodate the change being made. But it is usually thought that such a layer redefinition could always be done successfully if resources were available.

Layer violations in SDOS are especially troublesome, because we depend to some extent on the restriction of downward information flow between the layers to limit the bandwidth of some covert channels. For this reason, careful attention must be given to modularization and layering within SDOS. We have the layers of the SDOS kernel and the trusted managers, the layers of the multilevel secure operating system in the host, and the layers of the OSI model. SDOS development involves the evolution of the Cronus kernel into the SDOS kernel, the addition of security features to the communications layers, and the integration of these two sets of layers with those of the TCB of the host operating system. It has been difficult to create a layer diagram that clearly and correctly represents the relationships between all of these components of the system.

Further work in this area might result in a redistribution of functions between the layers, that resolves these problems. On the other hand, it might result in the conclusion that the relationships between the entities in a system of this size and complexity are too complex to be described by the one-dimensional ordering represented by a traditional layer diagram, and that some multi-dimensional representation is more appropriate for describing their relationships.

## 5.3 Research into Formal Methods

Our work on this project developed several results of formal security analysis. We proved several theorems showing that the interconnection of components with certain properties will yield systems with known properties. These theorems effectively extend the theory of "hook-up security" begun by McCullough [McCull87]. Clearly, the theory can be extended further with theorems about other kinds of hook-ups.

Several points should be noted concerning the limitations of the formal methods of verification discussed in previous sections.

- In practice, the Gypsy program transformation technique is clumsy. The practitioner not only needs to combat the difficulties of using Gypsy, but must also carry out many tedious transformation steps manually. When an error is discovered in the transformed program, not only that program, but the original source must be corrected. Complicated assertions

**must be** given at many places in the text of the transformed program. Many of the verification conditions have a repetitively similar form.

None of this is intrinsic to the method. Many of these difficulties could be relieved by automated support.

- In at least one place, the design of SDOS depends on nondeterminism for its security. (The security database uses random numbers to generate identifiers securely.) However, the methods we developed for proving non-interference using Gypsy are not applicable to non-deterministic designs. Not only are Gypsy procedures intended to model just deterministic algorithms, but showing the existence of traces in a non-deterministic system is a harder problem in general than in the deterministic case.

- The method of decomposition called "input-limited restriction" is only a simple example of a larger search: if two components agree to communicate using some protocol, and the security of each is made dependent on whether the other obeys the protocol, what component properties and interesting protocols can be used to hook together a restrictive system?

- In cases that a covert channel could not be eliminated completely from an SDOS component, the component obviously could not be proved restrictive. But leaving such a component unverified is not satisfactory. Generalized versions of restriction need to be found, such that limited violations of security are possible. The severity of the violation can be controlled and quantified. The work appearing at the end of chapter 4 of [BBN88] is an attempt at such a generalization. However, more powerful extensions can undoubtedly be found.

Each of these points represents an avenue of possible future research.

## References


[BBN88] "The Secure Distributed Operating System Project," *BBN Laboratories, Inc. Report No. 6144 Rev. 2.1*, January 1988.

[Bell76] Bell, D.E., LaPadula, L.J., "Secure Computer Systems: Unified Exposition and Multics Interpretation," *Mitre Corp. Technical Report MTR-2997*, Revision 2, March 1976.

[Biba77] Biba, K.J., "Integrity Considerations For Secure Computer Systems," *Mitre Corp. Technical Report MTR-3153*, April 1977.

[Brook84] Brookes, S.D., Hoare, C.A.R., Roscoe, A.W., "A Theory of Communicating Sequential Processes", *Journal of the ACM*, vol. 31, no. 3, 1984.

[DoD85] "Department of Defense Trusted Computer System Evaluation Criteria", DoD, National Computer Security Center, Standard DOD 5200.28-STD, December 1985.

[DoD-G85] "Guidance for Applying the DoD Trusted Computer System Evaluation Criteria in Specific Environments", DoD Computer Security Center CSC-STD-004-85, June 1985.

[Girl87] Girling, C.G., "Covert Channels in LAN's," *IEEE Transactions on Software Engineering*, Vol. SE-13, No. 2, February 1987, pp. 292-296.

[Goguen82] Goguen, J.A., Meseguer, J., "Security Policy and Security Models," *Proceedings of the IEEE Symposium on Security and Privacy*, 1982.

[Good78] Good, B.I., et al., "Report on the Gypsy Language, Version 2.0," *Technical Report TR ICSCA-CMP-10*, Institute of Computer Science, University of Texas, Austin, September 1978.

[Jones78] Jones, A.K., "The Object Model: A Conceptual Tool for Structuring Software," *Operating Systems, An Advanced Course; Lecture Notes in Computer Science*, Springer-Verlag, Editors Bayer, Graham, and Seegmuller, 1978.

[McCull87] McCullough, D. "Specifications for Multi-Level Security and a Hook-Up Property", *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, May 1987.

[Schan86] Schantz, R.E., Thomas, R.H., and Bono, G., "The Architecture of the Cronus Distributed Operating System," *Proc. 6th International Conference in Distributed Computing Systems*, IEEE, June 1986, pp. 250-259.

[TNI87] "Trusted Network Interpretation of the TCSEC", National Computer Security Center, NCSC-TG-005, Version-1, July 1987.

[Ulysse87] "Foundations of Ulysses: The Theory of Security", ORA Tech. Report for RADC contract F30602-85-C-0098, April 1987.

[Vinter88] Vinter, S.T., "Extended Discretionary Access Controls," *1988 Symposium On Security and Privacy*, IEEE, April 1988.

[Weber87] Weber, D.G., and Lubarsky, R., "The SDOS Project—Verifying Hook-up Security," *Proc. 3rd Aerospace Computer Security Conference*, December 1987.