Providing Policy Control Over Object Operations in a Mach Based System

Spencer E. Minear
Secure Computing Corporation
Roseville, Minnesota

Providing Policy Control Over Object Operations in a Mach Based System

Spencer E. Minear
Secure Computing Corporation
2675 Long Lake Road,
Roseville, Minnesota 55113-2536

Email: *minear@sctc.com*

28 April 1995

Abstract

In both secure and safety-critical systems it is desirable to have a very clear relationship between the system's mandatory security policy and its proven operational semantics. This relationship is made clearer if the system architecture provides strong separation between the enforcement mechanisms and the policy decisions, and if the policy decision software is clearly identifiable in the system's architecture.

This paper describes a prototype Unix system based on Mach which provides mandatory control over all kernel-supported operations. The prototype work modified the Mach kernel by extending its limited control mechanisms based on the Mach port right. The control extensions allow a mandatory control policy to specify control over not only access to an object via a port right, but over the individual services supported by the object. The mandatory security policy is implemented in an external Security Server which provides very strong separation between policy enforcement and policy decision software. This makes it possible to support a wide range of security policies with no change to the kernel or applications.

1 Introduction

The fundamental tenet on which this work is based is that high-integrity secure and safety-critical systems benefit from an architecture which possesses the following characteristics:

1. The system must have a clear, mandatory security policy which defines its desired operation,
2. The system's enforcement mechanisms must provide control over *all* system operations,
3. The decision logic implementing the system's mandatory security policy must be encapsulated in a very limited number of system elements,
4. The system's enforcement mechanism must be simple and easy to locate, and
5. The system should consist of distinct elements which provide well-defined simple services.

For both secure and safety-critical systems the first three characteristics embody the key requirements. Without a clearly defined security policy[1] it is generally not possible to know ex-

[1] Traditionally, secure systems have focused on privacy aspects of security while safety-critical systems have focused on integrity aspects. More generally, the security policy can address both privacy and integrity. However, both application areas require a high degree of assurance that the system operates as specified at all times.

actly how the system is supposed to operate. If the enforcement mechanisms fail to provide control over all system operations or lack sufficient granularity in their control capabilities, it becomes impossible to assure many aspects of the system's operation. Also, if the enforcement mechanisms are complicated or hard to identify in the system, it again becomes difficult to assure that the system operates in agreement with the stated policy at all times.

Client-server and object-oriented systems built on a microkernel, provide a structure which addresses many aspects of the characteristics listed above. These types of systems are made of distinct elements which provide well-defined simple services operating over well-defined interfaces. The microkernel, in such a system, is the base system element. It provides a small number of well-defined fundamental objects and services which provide the building blocks on which operating systems, like Unix®, and other applications can be built. Most microkernels focus on providing an Inter-Process Communication (IPC) facility that can be used as the foundation for object-oriented or client-server systems. Objects are associated with the system's IPC-provided communication connections. Control of objects is supported by the IPC control facilities provided by the microkernel.

OSF's Mach-based Unix, Sun's Spring, and Chorus Systèmes Chorus®-based Unix are three examples of microkernel-based systems providing a Unix programming interface. They all support communication-based systems on which it is easy to build object-oriented or client-server type systems. In Mach and Chorus the base communication facility is called a *port* and in Spring it is called a *door*. Each has different operational semantics and control mechanisms and characteristics, but all allow the use of their IPC mechanism to provide a "handle" through which clients can obtain access to object services. In addition to implementing the basic communication mechanisms, each implements a number of other kernel objects, normally only accessible via the IPC mechanism. In the case of Mach, the list of these kernel objects includes tasks, threads, and memory cache objects.

The work discussed in this paper has been done by Secure Computing Corporation[2] and researchers at the Information Security Computer Science Research Division of the Department of Defense. The work is directed at providing a prototype secure system that addresses the desired characteristics in the numbered list above. The focus is on having a system in which all operations are controlled by a mandatory security policy. The security policy is a replaceable system element distinct from the system's enforcement mechanisms. The mandatory security policy ensures that the system's runtime operation is, at all times, bound to the system specification and not to the decisions made by the system users. The usual example of a mandatory policy is one that defines the rules of control over classified documents within the Department of Defense. A number of variations on this type of policy are discussed in [12].

The following sections focus on the changes made to the Mach kernel to provide fine-grained enforcement mechanisms over all system operations at the direction of a well-defined system security policy. In particular, Section 2 provides a summary of the baseline Mach system, its basic structure and facilities. Section 3 describes the existing control mechanisms present in Mach and their limitations in the context of the desired characteristics for secure systems discussed above. Section 4 presents an overview of the changes made to the Mach kernel to provide the required fine-grained control mechanisms and mandatory policy's control interface. Section 5 gives a brief summary of the ongoing work to integrate the additional control capabilities into the Unix operating system personality. Section 6 presents a summary of the current status of the work.

2 Mach Kernel Summary

Mach is a microkernel providing a set of basic facilities for use by operating systems and other applications. It is designed around

[2]This work was supported in part by the Maryland Procurement Office, contract MDA904-93-C-4209.

an Inter-Process Communication (IPC) facility based on a port. Mach and systems built on Mach utilize Object Oriented Design concepts by building on Mach's port abstraction. A port can be used as an object handle, through which object methods are invoked or object service requests are made. In addition to ports, Mach provides several other types of kernel objects including tasks, threads, and memory cache objects[11]. Tasks provide an environment in which all processing is done. All processing is done by a specific thread, and each thread is bound to a single task. Memory cache objects provide memory in which to store and manipulate data. In addition to these basic objects, the Mach kernel supports other objects such as devices, processors, and the kernel itself. Each of the non-port objects has one or more associated ports that are used to represent the object and through which all operations on the object are initiated.

An examination of the basic Mach structure shows that it is made up, primarily, of two parts; the IPC services provider and the set of object servers implementing the services of the non-port kernel objects. Mach uses its own IPC facilities to provide tasks access to its other types of objects. To request an operation on a non-port kernel object, a task sends a request to that object via its associated port. The IPC send operation is processed by the kernel the same as any send operation. When the send processing recognizes that the target object is a kernel object, control is transferred to that object's kernel server for processing. If the target object is managed by a task external to the kernel, the request is provided to that task via its use of the IPC receive operation on the same port. Figure 1 shows the relationship between the kernel's IPC services, the kernel's object servers, and the object servers that operate as tasks external to the kernel. Communication between clients and servers is provided by communication connections implemented in the kernel IPC services.

3 Mach Control Mechanism

As discussed in preceding sections, Mach's use of its IPC facility as the focus of all system operations means it supports the desired structural aspects of both secure and safety-critical systems. An important question, then, is the extent of control provided through the IPC facilities and how well the characteristics listed at the beginning of Section 1 are addressed.

Mach provides one primary control mechanism which is based on a *capability* concept. From the viewpoint of a task, a port is a task specific name called a port right. The task-specific port right embodies the capability (rights) that the task has to the port named by the port right. In Mach however, the range of capabilities embodied in a port right is limited. Each Mach port right represents the capability to access one or more of the following kernel-supported IPC-related operations:[3]

- Send,
- Send-Once, and
- Receive

The following related operations deal with the transfer of port rights between tasks:

- The holder of a send right can, through the use of a send operation on any send or send-once right, have the kernel either move or duplicate the send right to the receiver of the message.
- The holder of a send-once right can, through the use of a send operation on any send or other send-once right, have the kernel move the send-once right to the receiver of the message.
- The holder of a receive right can, through the use of a send operation on any send or send-once right, have the kernel create a send or send-once right for the receiver of the message, or move the receive right to the receiver.

There are two problems, each discussed in more detail below, with these existing control

[3] A single port right can define either a send and/or a receive right, or a send-once right.

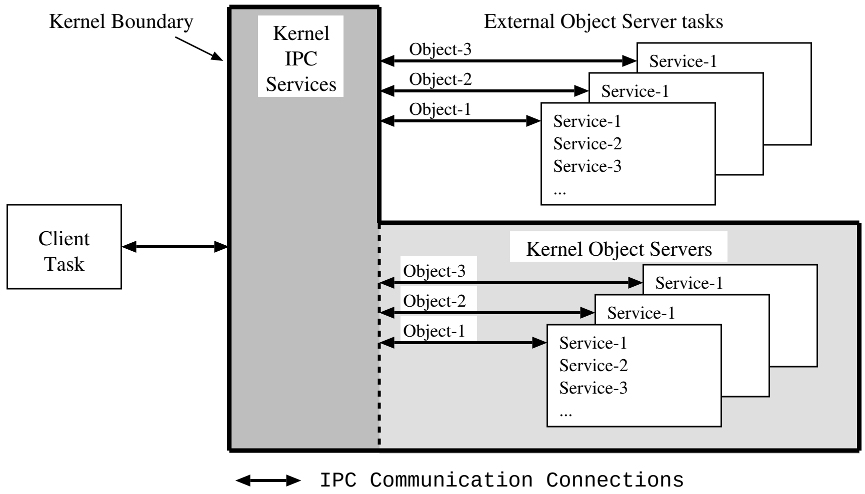

Figure 1: Mach Kernel Structure

mechanisms. The first is the limited control over the transfer of port rights, and the second is the complete lack of control over the object-related services. These problems are not associated with capabilities in general. The required characteristics of capabilities for use in secure systems were outlined very clearly by Karger and Herbert[7]. Previous works have provided designs for capability machines that do deal with the limitations present in the Mach implementation. Examples are provided by HYDRA[17], SCAP[8] and ICAP[5]. A common element in other systems is that a capability is necessary, but not sufficient to gain access to an object. The fundamental fault in Mach is that possession of a capability is sufficient for *full* access to *all* operations on the associated object. This makes it more difficult to build a secure system on Mach or on any other microkernel whose IPC lacks these control capabilities.

The intent of this work is to integrate the concepts present in these other capability machines into Mach. The desired result is a system that addresses the architecture characteristics identified in Section 1. The system utilizes the improved Mach kernel as the base for a Unix operating system controlled by an underlying mandatory control policy.

3.1 Port Right Transfer

The primary problem associated with the rules for controlling the transfer of port rights is the complete lack of kernel-provided mechanisms or facilities to verify that once a transfer is complete, the operational state of the system is still in agreement with the system's security policy. This is particularly dangerous to both secure and safety-critical systems. It means that the kernel is unable to identify or stop a task from accessing a port via a port right it obtained as a result of an error or via malicious action. Applications are left to resolve this problem themselves without assistance from the kernel. One design technique that can be used is to inject a layer of indirection in the use of all IPC operations. For example, in a normal Mach environment two tasks that are allowed to communicate may do so directly with the use of IPC. This means, however, that the sending task can transfer any right it holds, either intentionally or by accident, to the receiving task. If an application needs to assure that rights cannot flow from the sender to the receiver, then it is necessary to have an intermediary task to filter messages to stop of the transfer of port rights that violate the system's security policy. This approach can lead to sufficient control for many applications but may result in undesirable performance penalties and increased complexity in the application.

3.2 Service Control

Because the Mach port right control facilities have no association with object services accessible via a port right, Mach provides no direct control over object services. If an application needs to provide control over individual object services, it must address the problem by binding groups of object services to ports, essentially subdividing an object. The application can then attempt to control access to the services by controlling the distribution of port rights to the various groups of services.

An example of the use of this approach can be seen in the design of the Mach kernel itself. One of the kernel objects is the kernel itself, referred to as the *host* object. The range of operations available for the manipulation of the host object, however, are split into two groups: the privileged operations, like **host_reboot** and **host_set_time**, and generally available operations, like **host_info** and **host_get_time**. The designers of the kernel recognized that it would be necessary to control access to the kernel's privileged operations independently from the general operations. Thus, the host operations were split into the two groups with the privileged operations bound to the *host-privilege* port and others to the *host port*.

There are two undesirable aspects of this approach for controlling services. The first is the lack of flexibility. A grouping that is correct for one application and security policy might be incorrect for another application or security policy. The lack of flexibility of the grouping approach is particularly evident in the grouping of task object services. In total, there are about 45 different task services available on a task port and there is no ability to control access to these services individually. Thus, a holder of a send right to a task port has implicit permission to all 45 task related services. It is an all or none situation.

The second undesirable aspect of this approach is that it does not scale well. If it were possible to assign the operations to different ports, the result might be a larger number of ports, especially in the case of objects with many services such as the kernel's task object. This leads to complexity of the control aspects of the design. Unnecessary complexity of any type in any system is undesirable. In the case of secure and safety-critical systems, unnecessary complexity is especially undesirable and must be avoided wherever possible.

4 The Prototype

The prototype being developed by Secure Computing Corporation consists of a modified Mach kernel and an external Security Server. The separation of policy decisions done in the Security Server from enforcement done in the kernel has proven successful in the LOCK system[13] and was discussed in the context of a Unix system by Walker, Kemmerer and Popek in [16]. The prototype attempts to resolve the limitations in the base Mach control mechanisms were outlined in the previous section. To accomplish this, the prototype has added two new control mechanisms not available in the base Mach kernel and added a new interface to the kernel. The additions are, respectively:

IPC Control — The prototype provides expanded control over all aspects of port right manipulations. This allows the prototype's kernel to enforce policy-directed control over the transfer of port rights as well as over the use of the basic IPC operations.

Object Service Control — The prototype extends the port right capabilities to define policy directed control over the individual object services. The prototype kernel provides control over the individual services related to all kernel objects.

Security Server — The prototype implements a new interface between the Mach kernel and an external Security Server. This allows very strong separation between the enforcement mechanisms and the security-policy decisions. It allows the prototype system to ensure that all port right usage is in agreement with the current state of the security policy at the time of each usage. It also allows the system to localize the security policy in a single system element.

These additions address the control limitations discussed in Section 3. They also provide the system features necessary to support the desired architectural characteristics listed in Section 1. The two additional control mechanisms ensure that all system operations are subject to control. The new interface provides for the flow of control information from a mandatory security policy implemented in the Security Server to the enforcement mechanisms in the kernel and non-kernel object servers.

The general approach used in the prototype to add these new control mechanism is based on the concept of a *security fault*. The security fault concept and its implementation within the prototype are very similar to that of Mach's page fault processing and the use of external pagers to implement memory objects. A security fault occurs when a task attempts to use a port right for which there is no readily available access-permission information. In response to the security fault, the kernel interacts with the Security Server to obtain the relevant permission information. To minimize the costly interactions between the kernel and the Security Server, the kernel caches the permission information, in the form of *access vectors*, for future reference, just as the kernel caches data to minimize interactions with pagers.

To implement the new control mechanisms following the ideas laid out by the security fault concept, five specific types of changes were made to the Mach kernel:

1. The addition of identification information on kernel objects to support the policy-based access decisions,
2. The addition of permission checks and security-fault detection in the kernel's IPC processing software,
3. The addition of permission checks and security-fault detection in the kernel's object service processing software,
4. The addition of an access vector cache to minimize interactions between the kernel and Security Server, and
5. The extension of the kernel interface:
    - The addition of a new interface for the Security Server. It allows the kernel to obtain object access-permission information from the Security Server and allows the Security Server to invalidate previously granted permissions.
    - The extension of the existing IPC facilities to provide identification and permission information to external object servers along with a service request. The identification and permission information are available to the kernel IPC services from the kernel's access vector cache.

Figure 2 shows the structure of the extended Mach kernel and its interaction with the Security Server. It shows that the permission checks are done in the IPC processing to control the use of all IPC related services. It also shows that permission checking is done in the kernel's service processing software to provide control over individual object services. Before a kernel object's server initiates a requested service, both of these permission checks must be passed successfully.

Searches in the kernel's access vector cache are based on a pair of identifiers, bound to the relevant kernel objects. The first identifier is the *Source Security ID* (SSID) which embodies the control-relevant identity of the task making the request. The second identifier is the *Target Security ID* (TSID) which embodies the control identity of the object being accessed. When no entry is found in the cache, the current thread takes a security fault and the kernel makes a permission information request to the Security Server task. The kernel provides the (SSID,TSID) pair of identifiers and the permission being checked to the Security Server. The Security Server responds with the required access vector information that reflects the permissions based on the current state of the system's security policy.

Figure 3 shows the flow of identify and permission information from the kernel's access vector cache to a receiver of a request. The IPC processing binds the requester's SSID and

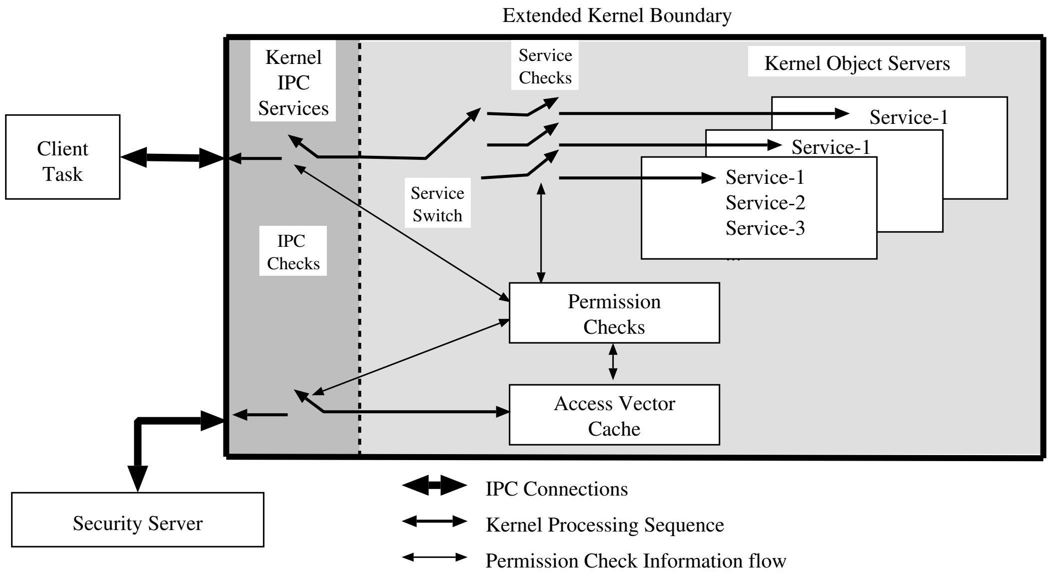

Figure 2: Kernel Control Mechanisms

access vector to the request message. Because the message receive operation is a direct communication between the kernel and a server, the object server can rely on the integrity of this identity and permission information and make object-specific policy-enforcement decisions as required. With the assumed proper operation of the kernel, the information is correct and was provided by the system's Security Server. This results in a system that naturally meets the desired design characteristics stated in Section 1. Each object server, whether in or out of the kernel, has a very simple enforcement operation that is easy to test and verify. Other enforcement related processing in the kernel is straightforward processing which binds information to relevant structures and reports the bound information correctly.

The Security Server is the central point in the system where all policy decisions, the most complicated and critical part of any secure or safety-critical system, are made. If the system's security policy cannot be assessed for correctness in the context of the single Security Server, it is highly unlikely that the security policy could be assured correct in any other implementation.[4]

The following sections discuss various aspects of the specific changes that were made to the Mach kernel.

4.1 Additional Identifiers

To support the split of enforcement from policy decision, it is necessary to bind identifiers to all kernel objects. Within the Security Server, the identifiers are bound to policy-specific attributes such as user name, data type, security level, etc. In the kernel's policy-enforcement operations, the identifiers are simply numbers associated with objects that are to be passed as parameters to permission checks. This makes the split between enforcement and policy very clean. The kernel and other server enforcement software is completely independent of the security policy. This makes it possible to use the same kernel and applications in sys-

[4] We recognize that there are multiple aspects of many system control policies and that not all of them should be centralized in all systems. What we are referring to here is the basic security policy which defines the fundamental operation of a system. Specific servers are free to extend this base policy. For example, a file system server is the proper place for a Discretionary Access Control (DAC) policy such as Access Control Lists (ACL).

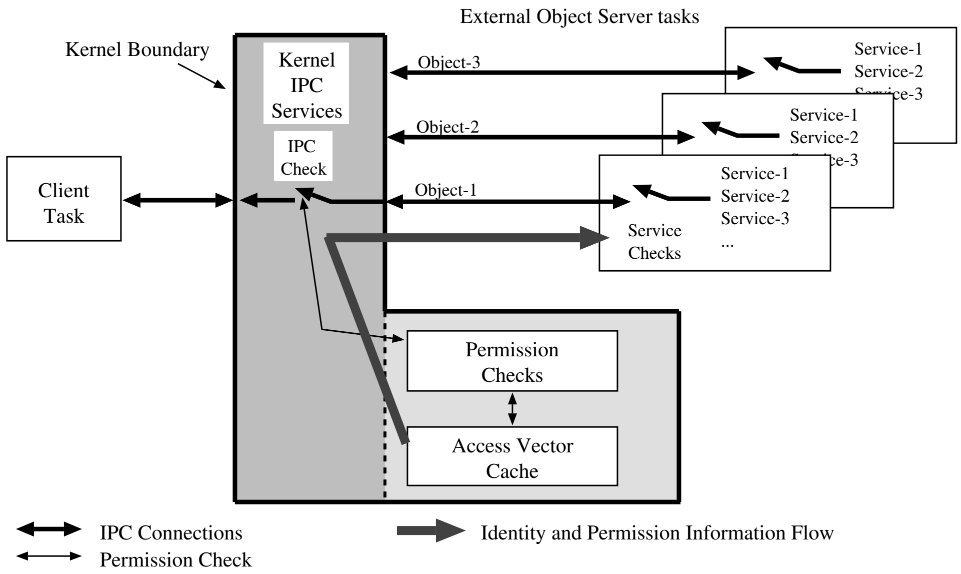

Figure 3: External Object Server Control

tems that must operate according to very different security policies.

The list of objects that were labeled with security identifiers includes:

- Tasks (SSID),
- Ports (TSID), and
- Memory Cache Objects (TSID).

The close relationship between Mach kernel objects as well as some practical performance issues lead to a decision to make some security identifiers derivable from the identifier of a closely related object. An initial approach, driven by a desire to keep policy issues out of the kernel, was to have the kernel provide the base object's identifier to the Security Server and have the Security Server provide the proper TSID value in return to the kernel. This was rejected primarily to avoid the obvious performance impact and simplify both the kernel and the Security Server. In addition, there is no loss in generality at the policy level if some of the identifiers are derived from other identifiers. The decision was made to use the high byte of each security identifier to indicate the class of object it is bound to. For example, one value indicates that the identifier is bound to a task's task port, while another indicates the identifier is bound to a thread port. The remaining bits of the security identifier are bound to the associated object's security attributes within the Security server. The security-identifier classification values are defined in the kernel-Security Server interface specification.

An example is the relationship between the identifier of a task and that of its associated task port. Each task is identified with a SSID since it is viewed as being the source of operation requests. A task's task port, like all ports, must be labeled with a TSID. The task port's identifier is produced by replacing the value in the high byte of the task's identifier with the task port-identifier classification value. This operation is very simple and very fast. Because these relationships must be common to the kernel and Security Server, the relationships are defined in their interface specification.

4.2 Access Vectors

For the kernel or any object server to actually enforce a policy decision, it is necessary for the enforcement software to have access to current permission information at each point where a security fault may occur. The permission information is provided in the form of an access vector which is computed based on the relevant (SSID, TSID) pair of identifiers. Each access vector defines the current state of permissions that the SSID has to all operations supported by the object bound to the TSID.

The structure of access vectors within the prototype is based on the two aspects of the control mechanisms: the IPC and object specific services. Figure 4 shows this basic two-part structure of an access vector. The fields in the IPC portion of the access vector are common to all access vectors because all services are accessed via IPC operations. The service part of the access vector, however, is viewed as a union of all possible object-specific access vectors. Within the prototype, service vectors for each of nine types of kernel objects supported by the Mach kernel have been defined. The addition of other service vectors has no impact on the kernel as service checks are always done in the context of the specific object.

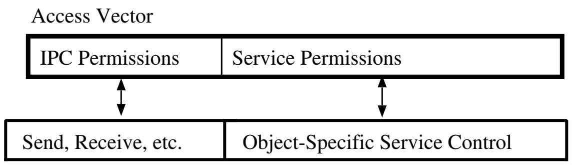

Figure 4: Access Vector Structure

This approach results in very simple, easy-to-assure enforcement software. It consists of a simple test of the appropriate field of an access vector. This approach is also very easy to extend to include the specification of control over application level objects as additions to the system's security policy.

Because all access vectors include the associated IPC permissions, the kernel continues to be the enforcer of the IPC permissions for all object accesses on the system. This means that the kernel—the unbypassable system element—is capable of enforcing the system security policy's definition of allowed task interactions.

4.3 Interface Extensions

Another aspect of the prototype kernel work is the modification of the Mach kernel interface. A key requirement levied on the prototype work was that all changes to the Mach kernel interface would maintain backward compatibility with the existing interface. To satisfy this requirement, all changes to the interface are in the form of one of two types of extensions.

1. Extensions to provide tasks with visibility of security relevant information, and
2. Extensions to support the kernel-Security Server interactions. These interactions resolve security faults and respond to policy state changes within the Security Server.

In making security relevant information visible to tasks, eight new entries were added to the kernel interface. Each is closely associated with an existing kernel interface and differs only in that extra parameters are accepted or provided. The additional entries are:

- Allow the creation of kernel entities, tasks, ports, and memory cache objects with specified identifiers, for example **task_create_secure** and **mach_port_allocate_secure**.
- Allow applications to obtain identifier and access information about kernel entities, for example **mach_msg_secure** and **mach_port_type_secure**.

All of the existing kernel interfaces remained syntactically the same, though their operational semantics may be affected by the policy denying the required permissions.

The extensions to support the kernel-Security Server interactions consist of one outcall from the kernel to the Security Server and four additional kernel services used by the Security Server. The single new outcall is used by the kernel when it needs to obtain an access vector to complete the processing for a security fault. The thread causing the security fault is forced to wait until the response is provided by the

Security Server. The kernel provides the Security Server with the appropriate (SSID,TSID) pair and indicates which permission is being checked. The Security Server responds with the same pair of identifiers, the current state of the associated access vector and cache control information. The response is sent on the thread's Remote Procedure Call (RPC) reply port which is controlled by the kernel.[5]

Two additional services were added to the kernel's host object, accessible on the generally available host port.[6] These additional services allow the Security Server to:

1. Register the port which the kernel uses to send permission requests to the Security Server, and
2. Tell the kernel to flush all or part of its access vector cache.

The Security Server uses the first new service to notify the kernel that it is operational and to identify the port to use for sending permission check requests. Prior to the point in time, during system startup, when the Security Server becomes operational, the kernel must be able to make permission decisions on its own. As part of the prototype development, a list of the permissions for the operations done during system startup was developed and integrated into the kernel as the initial state of the access vector cache. This means that the initial operation of the system is done in agreement with this limited security policy statement. This part of the design is important to help establish the integrity of the system's initial state which is a key issue in the operation of any secure or safety-critical system. The system could disable all permission checks until the Security Server is operational. It is better, however, to specify correct operation even during startup and ensure that permission checking is always enabled.

[5]The kernel-provided information of which permission is being checked and the Security Server's cache control information were added to the interface to support the use of policies which determine the current permissions based on the history of previous accesses to the associated objects.

[6]With the base Mach control concept these operations would have to be split between the host privilege port and the host port. The prototype relies instead on the policy-defined control to specify which tasks in the system are allowed to request the specific operations.

The Security Server uses the second new kernel service to control the state of the kernel access vector cache. This facility allows the prototype kernel to support Security Servers which implement a variety of dynamic security policies where access permissions change during the operation of the system for any number of policy-controlled reasons.

4.4 Performance Issues

Throughout the development of the prototype the resulting performance of the system was a critical concern. As stated above, the addition of a kernel resident access vector cache was done primarily to minimize the number of time consuming interactions between the kernel and the Security Server. Within the kernel this cache was implemented at two levels. The bottom level is a straightforward cache which stores access vectors so that they can be found based on the (SSID, TSID) pair. The second level consists of a change to the Mach kernel's port right structure to include a pointer to the relevant access vector in the cache storage area. Thus, once a port right is used by a task, subsequent references to that port right have direct access to the associated access vector. In this case, even the cost of the cache lookup is avoided. To avoid the problem of stale pointers, provisions were made to ensure that stale access vectors are detected and security fault processing is initiated.

5 Unix Issues

Though the focus of the prototype work discussed in this paper has been the Mach kernel, work is being done at the Information Security Computer Science Research Division of the Department of Defense and at Secure Computing Corporation. to define and develop a Unix-like Application Program Interface (API) over the prototype microkernel. The operating system design uses a multi-server model[6] in which operating system services such as the file system, process management and network management are implemented within separate Mach tasks. The initial prototype described in this paper makes use of single-server Unix operating systems while the multi-server work continues. The initial work is making use of both CMU's UX and the Lites operating systems. The plan calls for migration to the Hurd, being developed by the Free Software Foundation.

The security of this system architecture is increased by making use of both aspects of the prototype's extended control capability: control of the basic IPC operations and control of individual services associated with a specific object accessible via a port. In this system model nearly all operations depend on the use of the kernel's underlying IPC facilities. With the association of the proper security label to each process, the prototype kernel, under the direction of the mandatory security policy, provides assured separation of tasks. Thus, applications that should be separated are assured to have no access to each other. Also, specific types of applications can be isolated from operating system services to which they should have no access.

Control over object services allows the security policy to go beyond the simple issue of isolation to the question of how processes may interact. For example, because the kernel enforces all access to kernel object services, a process may be allowed to duplicate itself, through the use of **task_create**. But the same process is not allowed to create a task in a new security context by refusing it access to the **task_create_secure** service. The same concept is readily applicable to both operating system servers and other trusted applications that benefit from having control over object services.

The prototype work is providing a very simple demonstration database system which implements service-level control over operations on objects in the database. The demonstration system consists of a user interface application that is instantiated with security identifiers identifying the various user roles such as doctors, nurses and business management. Access to the database is controlled by a database-entry server. Application-specific control policy ensures that the database-entry server is the only task on the system with direct access to the database server. The application-specific security policy also specifies the services each user role is allowed. Security decisions for this policy are made in the Security Server and enforced by database entry server task.

The changes to the Unix system and to potential specialized secure applications are similar to the general change model discussed relative to the Mach kernel itself:

- Unix objects, such as files and processes, require security identifiers,
- The Unix interface requires extension to allow applications to specify and obtain identifiers of objects, and
- Object servers must reference the access information provided with each request and enforce the policy-defined permissions.

These changes are largely transparent and have a minimum impact on applications executing on the system.

6 Results

At this time the prototype is operational. It is being used to carry out further research into adaptive security policies and is being made available to other research organizations interested in this work.

The prototype kernel has demonstrated strong backward compatibility with the baseline Mach releases (MK83) from Carnegie Mellon University (CMU). The Unix server and Unix emulator operate on the prototype kernel with no change, as will the rest of the CMU-provided Unix daemons and Unix environment binaries. To facilitate further experimentation and assessment of the operation of the kernel, the Unix server in the prototype has been extended to support the creation of Unix processes that operate in tasks which are labeled with a SSID.

The prototype's Security Server implements a security policy with two fundamental control aspects. It provides control based on a non-hierarchical integrity policy, developed by Secure Computing Corporation, known as Type Enforcement[1][15], and provides a hierarchical Mandatory Access Control (MAC) policy. Work is continuing to investigate the prototype's policy flexibility.

Little change was required to add these features to Mach. Approximately 10% of the files have required some form of modification. The most typical changes are:

- Permission checks in the IPC and service processing, and
- Initialization and maintenance of security identity information.

Current lines-of-code counts indicate that the size of the baseline Mach kernel code increased by approximately 8%. The largest changes, in terms of lines of code, are related to the services that were added, with the access vector cache being the largest single addition. Other additions were generally duplications of an existing services, with minor changes to the logic or the input or output parameters. In many cases a duplicated routine could be merged with the existing one. We chose to use the duplicate approach to localize the changes and ensure that the existing services continued to operate.

6.1 Performance

System performance is being measured in two ways: with a Mach performance test suite developed at the Worcester Polytechnic Institute (WPI) Computer Science Department[3][4], and a simple kernel compilation test. The later measures system time to compile the IPC portion of the Mach kernel. All tests were being executed on a PC-clone with a 486DX2-66MHZ processor, 8 MB of memory and a 1GB SCSI disk.

The tests were run on the baseline Mach kernel and each of the incremental versions of the system during the development. Table 1 provides a summary of the results as run on the baseline Mach kernel and the first version of the completed prototype system. The test runs on the prototype system included a best case situation,(100% cache hits), and a worst case situation, (0% cache hits). Table 1 provides a summary of the test results.

In addition to gathering performance test data, we instrumented the system to count the number of permission checks and kernel-Security Server interactions made during a test. Each row, under Data Description, labeled *Permission Checks* in Table 1 indicates the number of permission checks made during the test. Each row labeled *Security Server Requests* indicates the number of times the kernel sent a permission check request to the Security Server. It should be noted that the difference between permission counts and Security Server interactions in the worst case reflects the fact that permission checks on Security Server operations are not allowed to result in a security fault. The Security Server is treated the same as all other tasks and thus all of its operations are subject to policy-defined permission checks. However, to avoid a deadlock, the cache is provided with wired access vectors that describe the allowed Security Server operations. Thus there is no special casing of permission checks within the kernel.

The three tests referenced in Table 1 are:

WPI Jigsaw: This test solves a mathematical model of a jigsaw puzzle. The test was designed to evaluate the performance of the memory management features of the system. The test was run with puzzle sizes ranging from 8x8 to 64x64.

WPI Sdbase: This test uses TCP/IP sockets to communicate between a single server and multiple clients. The test analyzes performance of a server client application. The test was run using both 5 clients and 25 clients.

IPC Compilation: This test measures time to compile the IPC portion of the baseline Mach kernel.

Due to the fact that we see variation in test results for run to run, it is probably dangerous,

Table 1: Performance Results

Test	Data Description	Baseline	Modified Kernel Best Case	Modified Kernel Worst Case
WPI Sdbase				
5 Clients	Avg. Client Total Time(ms)	39344	40084	202278
	Avg. Client Communication Time(ms)	16308	16670	23178
	Avg. Server Time(ms)	27564	28628	187500
	Permission Checks	NA	110799	415086
	Security Server Requests	NA	0	108692
WPI Sdbase				
25 Clients	Avg. Client Total Time(ms)	205168	235434	1231272
	Avg. Client Communication Time(ms)	20904	23469	81598
	Avg. Server Time(ms)	180422	209395	1116058
	Permission Checks	NA	692010	2647666
	Security Server Requests	NA	0	682160
WPI Jigsaw		Average Values Over 10 Runs		
8 x 8	Time(ms)	19	21	24
12 x 12	Time(ms)	83	74	78
24 x 24	Time(ms)	185	181	186
32 x 32	Time(ms)	1941	1996	1998
40 x 40	Time(ms)	4320	4382	4381
48 x 48	Time(ms)	8130	8190	8233
55 x 55	Time(ms)	13061	13130	13233
64 x 64	Time(ms)	22100	22459	22433
	Permission Checks	NA	32630	66869
	Security Server Requests	NA	0	16337
IPC Compile		Average Values Over 10 Runs		
	Real Time(sec)	987	1031	1787
	User+Sys(Sec)	749	767	842
	Percent Utilization	76	74	47
	Permission Checks	NA	436046	1457665
	Security Server Requests	NA	0	469294

at best, to try to draw narrow numeric conclusions from these early test results. We have seen time changes that can only be explained as resulting from a change in the page alignment of kernel code. We also believe that test results are influenced by the state of fragmentation on the disk.

Our initial assessment of the test results is that the best-case performance of the prototype system tends to be slightly slower than the baseline. Some best-case tests are faster while others are slower. The worst case tests show significant differences, but the range of difference depend on the nature of the tests. Over all the test result behavior is largely what we anticipated;

- Since the amount of code required to make a permission check is small in comparison to that involved in normal Mach kernel processing, and
- Context switching to the Security Server is clearly more expensive and should be avoided when possible,
- The performance will be influenced more by the effectiveness of the cache than the fact that the checks are being made.

Applications like the SDBase which make heavy use of IPC, results shows larger potential impact in the worst case, while the best case is comparable to the baseline. Applications like the memory intensive JIGSAW test shows little difference in the best case. And since it has fewer permission checks per execution time the worst case test shows a smaller amount of potential change.

Further testing using more operational scenarios is required before firm conclusions can be made. At this early stage of prototype system operation, the performance test results are encouraging and confirm our opinion that very fine grained security control can be done, in many application areas, with a minimum performance impact.

7 References

[1] W.E. Boebert and R.Y. Kain. A Practical Alternative to Hierarchical Integrity Policies. In *Proceedings of the 8th National Computer Security Conference*, September 1985.

[2] Ellis Cohen and David Jefferson. Protection in the Hydra Operating System. In *Proceedings of the Fifth Symposium on Operating Systems Principles, Operating Systems Review 9,5*, pages 141–160, Austin, TX, November 1975.

[3] David Finkel, Robert E Kinicki, Aju John, Bradford Nichols, and Somesh Rao. Developing Benchmarks to Measure the Performance of the Mach Operating System. In *Proceedings of the USENIX Mach Workshop*, pages 83–100, 1990.

[4] David Finkel, Robert E Kinicki, Jonas A. Lehmann, and Joseph CaraDonna. Comparisons of Distributed Operating System Performance Using the WPI Benchmark Suite. Technical Report WPI-CS-TR-92-2, Department of Computer Science, Worchester Polytechnic Institute, Worcester, MA 10609, 1992.

[5] Li Gong. A Secure Identity-Based Capability System. In *IEEE Symposium on Computer Security and Privacy*, pages 56–63. IEEE, 1989.

[6] Daniel P. Julin, Jonathan J. Chew, and J. Mark Stevenson. Generalized Emulation Services for Mach 3.0 — Overview, Experiences and Current Status. In *Proceedings of the USENIX Mach Symposium*, pages 13–27, Monterey, California, November 1991.

[7] P.A. Karger and A.J. Herbert. An Augmented Capability Architecture to Support Lattice Security and Traceability of Access. In *Proceedings of the 1984 IEEE Symposium on Security and Privacy*, pages 2–12, April 1984.

[8] Paul Ashley Karger. Improving Security and Performance for Capability Systems. Technical Report 149, Univeristy of Cambridge, Cambridge England, October 1988.

[9] John Knight and Bev Littlewood. Critical Task of Writing Dependable Software. *IEEE Software*, 11(1):16–20, January 1994.

[10] Roy Levin, Ellis Cohen, William Corwin, Fred Pollack, and William A. Wulf. Policy/Mechanism Separation in Hydra. In *Proceedings of the Fifth Symposium on Operating Systems Principles*, pages 132–140, Austin, TX, November 1975.

[11] Keith Loepere. *Mach 3 Kernel Interfaces*. Open Software Foundation and Carnegie Mellon University, November 1992.

[12] Ravi S. Sandhu. Lattice-Based Access Control Models. *Computer*, 26(11):9–19, November 1993.

[13] O. Saydjari, J. Beckman, and J. Leaman. LOCK Trek: Navigating Uncharted Space. In *IEEE Symposium on Computer Security and Privacy*, pages 167–175. IEEE, 1989.

[14] Daniel Jay Thomsen. Integrity Issues in Secure Systems. Master's thesis, University of Minnesota, May 1991.

[15] D.J. Thomsen and J.T. Haigh. A Comparison of Type Enforcement and Unix Setuid, Implementation of Well Formed Transactions. In *Proceedings of the 1990 Computer Security Applications Conference*, pages 304–312, December 1990.

[16] Bruce J. Walker, Richard A. Kemmerer, and Gerald J. Popek. Specification and Verification of the UCLA Unix Security Kernel. *Communications of the ACM*, 23(2):118–131, February 1980.

[17] William A. Wulf, Ellis Cohen, William Corwin, Anita K. Jones, Roy Levin, C. Pierson, and F. Pollack. HYDRA: The Kernel of a Multiprocessor Operating System. *Communications of the ACM*, 17(6):337–345, June 1974.