Flexible Access Control Using IPC Redirection

Trent Jaeger Kevin Elphinstone Jochen Liedtke Vsevolod Panteleenko
Yoonho Park
IBM T. J. Watson Research Center
Hawthorne, NY 10532
Emails: {jaegert|kevine|jochen|vvp|yoonho@watson.ibm.com}

Abstract

We present a mechanism for inter-process communication (IPC) redirection that enables efficient and flexible access control for micro-kernel systems. In such systems, services are implemented at user-level, so IPC is the only means of communication between them. Thus, the system must be able to mediate IPCs to enforce its access control policy. Such mediation must enable enforcement of security policy with as little performance overhead as possible, but current mechanisms either: (1) place significant access control functionality in the kernel which increases IPC cost or (2) are static and require more IPCs than necessary to enforce access control. We define an IPC redirection mechanism that makes two improvements: (1) it removes the management of redirection policy from the kernel, so access control enforcement can be implemented outside the kernel and (2) it separates the notion of who controls the redirection policy from the redirections themselves, so redirections can be configured arbitrarily and dynamically. In this paper, we define our redirection mechanism, demonstrate its use, and examine possible, efficient implementations.

1. Introduction

In this paper, we present a flexible mechanism that supplements inter-process communication (IPC) so that we can enforce access control policies efficiently. In systems where address-space separation is used to restrict processes, all security-sensitive actions are implemented via IPCs. Therefore, an IPC mechanism must enable proper enforcement of access control. However, the overhead necessary for IPC to support access control should be as low as possible. Toward this end, we propose a mechanism that enables a trusted process to configure IPC paths between processes dynamically such that a system security policy can be enforced with low overhead.

The conflict between IPC performance and security-policy enforcement is fundamental to the development of micro-kernel systems. In such systems, many system services may be implemented at user-level, so many processes may need to communicate to complete a task. In early micro-kernel systems, such as Mach, the kernel only authorizes whether a process can send or receive an IPC from another process [9]. Actual authorization of the operations on server objects is performed by the servers themselves. While the server must be trusted to control access to its own objects, it may not be designed to either interpret or enforce the system's global security policy. In these cases, an additional trusted process, often called a *reference monitor*, is assigned the task of enforcing the system's access control policy properly.

Security researchers have made extensions to such systems to enable the enforcement of security policy, such as the Distributed Trusted Operating System (DTOS) [8], but Mach's IPC mechanism limits flexibility which, in turn, limits the potential optimizations. Mach ports enable a principal with receive rights to that port to receive IPCs from those principals with send rights to the port. In order to interpose a task to intercept IPCs, the current receiver must delegate its port receive right to another principal. This means that interposition requires the cooperation of the receiver which is a problem if either: (1) the receiver is incapable of the cooperation or (2) the receiver resists the interposition. Thus, the DTOS implementation which uses Mach's IPC mechanism depends on the server processes (or the kernel for kernel resources, such as ports) to call security servers to authorize operations using the system security policy (a round-trip IPC). Since the servers cannot tell when security policy has changed, they must continue to request authorizations using this mechanism. Even though the designers of DTOS smartly cache capabilities in the kernel to improve performance (to a single system call, about one IPC), in general, these IPCs would not be necessary if the IPCs could be redirected to reference monitors on demand (e.g., only when the security policy changes).

In more recent micro-kernel systems, Liedtke pushed authorization completely out of the kernel to optimize IPC performance [7]. In the L4 system, the kernel does not authorize communication, but provides unforgeable IPC source identification and permits any IPC to be redirected to another process instead. L4 uses a model called the *Clans & Chiefs* to define these redirections [6]. In this model, a chief is a special process in a clan (i.e., set of processes). Assignments of clan members to chiefs is static in L4. Any IPC between two processes in the same clan is forwarded directly to the destination by the kernel. However, any IPC either to a process outside the clan or from a process outside the clan is automatically redirected by the kernel to the clan's chief. To monitor individual processes, we found it necessary to assign each to their own clan, so three IPCs (client-chief, chief-chief, chief-server) are required for an inter-process request. While we showed that the base cost could be as low as 4 μs and that we measured 9 μs [5], in many cases, the chief-chief IPC is unnecessary [10].

In this paper, we define a mechanism for controlled IPC redirection, in which trusted processes, called *redirection controllers*, can determine the redirections for their tasks. We show that we can use this mechanism to manage IPC paths such that system security policies can be enforced with few IPCs. For example, an IPC path may be monitored initially, but once approved, the reference monitor may be removed (if security policy allows). Only when the security policy changes should the reference monitor interpose the communication again.

A similar concept that is being developed concurrently to our IPC redirection mechanism is called a *portal* [2]. A portal is a reference to a customizable IPC mechanism. Similar to our IPC redirection entries, portals are defined by a privileged task and invoked by the kernel on an IPC request. The proposed implementation of portals uses a portal table in the nucleus to vector references to the portal code which the nucleus then executes (i.e., all portal code is trusted). The main difference between our IPC mechanism and portals is that portals enable trusted customizations to be run inside the nucleus whereas the IPC redirection mechanism aims to define a general nucleus mechanism for efficient IPC direction (i.e., no custom code is necessary for redirection). Customizations would be implemented outside the kernel (e.g., in the redirected destination). Further tests are necessary to compare the effectiveness of our IPC redirection mechanism to implementations of redirection using the portal mechanism.

The remainder of the paper is organized as follows. In Section 2, we define the roles of the system's trusted computing base (TCB), the kernel, and the servers in enforcing system security policies. In Section 3, we define our new mechanism. In Section 4, we describe how this mechanism can be used to enforce security policies, including Clans & Chiefs itself. In Section 5, we discuss how the mechanism can be implemented. In Section 6, we conclude and discuss future work.

2. Security Policies

In this section, we define the roles of the system's trusted computing base (TCB), the kernel, and the servers in enforcing system security policy. These roles determine the requirements of the mechanism.

The system TCB must be able to enforce the following security requirements:

- **Communication**: the system must be able to restrict the ability of a process to send an IPC to another process
- **Authentication**: the system must identify the source of an IPC
- **Authorization**: the system must be able to determine whether a particular operation on a particular object should be permitted
- **Delegation**: the system must be able to control the delegation of permissions from one process to another
- **Revocation**: the system must be able to revoke the ability of a process to communicate with another process or perform an operation on another process's objects
- **Denial**: the system must be able to prevent denial-of-service attacks on processes
- **Mechanism**: the system must be able to implement arbitrary access control mechanisms (e.g., optimized for the policy that they enforce) of its own choosing

The system TCB must be able to restrict communication between any two processes to prevent unauthorized information leaks. Once a communication is established, the system must provide the necessary authentication information to its monitors such that they can enforce system's access control policy effectively. Next, the system must be able to authorize whether the operation requested can be performed on the specified object. Since processes may be loaded dynamically and may not be aware of the system security policy, they may delegate permissions that the system security policy forbids the destination from obtaining. Therefore, the system TCB must be able to restrict delegation of any permission to perform any operation on a server object and revoke existing permissions. In addition, the system TCB must be able to prevent denial-of-service attacks on processes by permitting legitimate requests to be forwarded in preference to those deemed excessive. Lastly, the system

TCB should be able to support any conceivable system security policy without the need to modify the kernel.

The kernel need not actually perform any of these security checks itself. Rather, the kernel must enable its system's TCB to enforce these requirements. The goal is for the kernel to provide the system with the flexibility to enforce its security requirements while providing high-performance IPC services. Therefore, systems with no security requirements execute at the highest IPC efficiency, while those with some subset of the security requirements above may enforce those with a minimal number of IPCs and system calls.

An important issue is determining the security requirements that must be enforced by reference monitors and those which can be enforced by the servers themselves. An effective enforcement mechanism must mediate all IPCs, prevent tampering of security policy, and be sufficiently simple that it can be validated [1]. In general, reference monitors are necessary to control (i.e., block) any communication channel. However, once a communication channel between a client and server is authorized, a server may mediate IPCs on that channel. Since servers define the mapping of their data to their objects, they must be trusted to perform the accesses requested on their objects anyway. However, servers must also be able to protect the security policy from tampering to be entrusted with its enforcement. The degree of this trust may be tempered by limiting the policy managed by the server and the amount of time in which IPC is monitored solely by the server.

3. Solution Proposal

In this section, we describe our redirection mechanism. In this mechanism, the kernel implements IPC given a redirection policy specified by a user-level process called *redirection controller*. A redirection controller is privileged to set redirection policy for the processes in its *redirection set*. Since a redirection controller is a user-level process, the system designers are free to implement redirection policy as desired for their system.

The kernel implements IPCs that may be redirected depending on the policy specified. Redirected IPC is described by the following formalism. We define a *redirection function* R that maps an IPC source process (s) and destination process (d) to an interim destination (i),

$$R(s,d) \mapsto i, \text{ for } s,d,i \in P$$

where P is the set of all processes.

Let $\rightarrow$ designate traditional IPC, thus $s \rightarrow d$ represents an IPC from process s to process d. Given this, we define a new IPC mechanism $s \Rightarrow d$ to be $s \rightarrow R(s,d)$.

The redirection controller has the ability to set R for each s in its redirection set, the set of processes for which R may set redirection policy. For example, setting R such that $R(s,d) = d$ allows direct, unrestricted IPC from s to d, setting $R(s,d) = i$ redirects IPC from s to an *interim destination* i. Arbitrary redirection can be implemented by the redirection controller within its redirection set, and the redirections can be established dynamically.

Interim destinations can block, revise, or forward IPCs between sources and destinations. In general, interim destinations can be used as reference monitors, IPC tracers, and debuggers to name just a few examples.

In order to authorize operations properly, the destination needs to know the identity of the original IPC source. The kernel specifies the identity of the direct source (i.e., the interim destination) of an IPC, but the interim destinations are permitted to specify the an original IPC source (called "deceiving" IPC in the Clans & Chiefs model). However, this specification must be limited to prevent interim destinations from claiming unauthorized sources. If i is acting as an interim destination for IPC from source s to destination d, then i is permitted to send IPC to d specifying s as the original source of the message ($i \overset{s}{\Rightarrow} d$). Permissible IPC deceit is specified by

$$i \overset{s}{\Rightarrow} d \text{ iff } R(s,d) = i$$

If multiple interim destinations are used, the destination can assume that all deceits are specified by

$$i \overset{s}{\Rightarrow} d \text{ iff } \begin{cases} R(s,d) = i \\ R(s,d) = x, i \overset{x}{\Rightarrow} d \end{cases}$$

In general, multiple interim destinations may forward an IPC and specify a original source. Therefore, trust in the validity of the original source specified in the IPC received by the destination is an issue. The kernel only provides: (1) a valid identity of the last interim destination and (2) a restriction on the set of sources that can be claimed by an interim destination. Thus, the number of possible sources for an interim destination should be commensurate with the system's trust in that interim destination. In addition, interim destinations can build a user-level chain of interim destinations to aid the final destination in authentication. In this case, each interim destination appends its identity to a chain of destinations. If there are any untrusted interim destinations, at least one will be discovered by a trusted interim destination or the final destination itself. If an untrusted interim destination appears in a chain, the destination may not believe the identity of the source. In which case, a secure channel may be established between the source and destination.

4. Examples

In this section, we aim to demonstrate the power and flexibility of the IPC redirection mechanism. In the first ex-

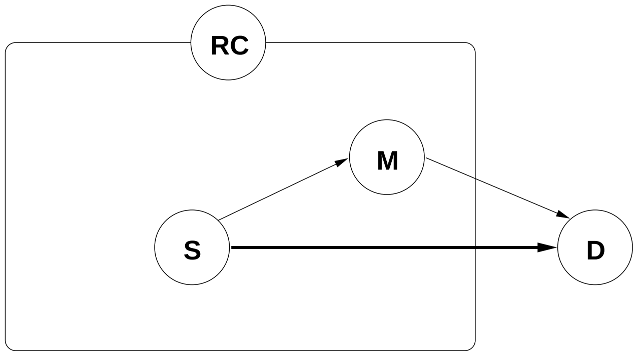

Figure 1. Process M monitors operations by its processes as specified by RC.

ample, we show how a reference monitor architecture can be constructed. In the second example, we show how the Clans & Chiefs model may be emulated.

4.1. Reference Monitors

Consider Figure 1. Process RC is the redirection controller for the system. Process M is a monitor for processes s and d. Process RC redirects all IPCs sent from s to d to M. That is, $R(s,*) = M$. The monitor performs all access control checks for these processes, and it may send IPCs to the destinations directly (i.e., $R(M,x) = x$). Suppose that s wants to send an IPC to d. The kernel redirects the IPC to M which authorizes the IPC and forwards it to d. Any return IPC from d to s is automatically redirected to its reference monitor, which could be M or some other process.

In this application, the reference monitor M enforces the system's security policy on the two processes. The reference monitor maintains the capabilities for s to invoke operations on d. First, it can prevent s from making operation requests to objects on d. Second, it is capable of revoking any of s's capabilities for invoking operations on d when changes in access control policy occur. Third, the monitor can prevent s from using capabilities to perform operations on d that it may have been delegated by other processes. Fourth, it can prevent s from implementing a denial-of-service attack on d by limiting the number of requests that may be sent. Fifth, different monitors may implement different mechanisms to enforce the security requirements of their processes effectively and efficiently. Note in this scenario, the monitors are trusted to specify the source of the IPC to the destination.

The IPC redirection mechanism enables significant flexibility in access control enforcement by enabling the redirection controller to assign monitors to individual processes. Thus, the authorization mechanism used may be customized to the security requirements of the monitored process. Also, the redirection controller can change the current monitor of a process when security requirements demand it.

In addition to changing a process's monitor, the redirection controller may remove the monitor from individual IPC paths. Security requirements must be such that access control policy can still be enforced without the monitor. The requirements for permitting the redirection function mapping $R(s,d) = d$ are: (1) s can communicate with d; (2) d does not need protection from denial of service attacks from s; and (3) d is trusted to enforce authorization requirements on s. Delegation is controlled on accesses using transferred capabilities (by the monitor or server). The first condition can be determined from s's permissions. The second condition depends on the system's trust in s and d's ability to inform the monitor about a potential denial of service attack. The third condition requires either: (1) that s is trusted or (2) that d can obtain a representation of the system security policy for s to d's objects and d is trusted not to modify this policy. While d must be trusted to perform the operations requested on its objects, it may not be trusted to protect the access control policy from tampering. We limit the possibility of inadvertant tampering by providing a *system authorization library* that authorizes operations using a read-only access to system security policy. Servers manage the capabilities they grant using this library. The library enables a reference monitor to take over authorization using server capabilities on demand, so temporary control by servers is possible (e.g., until policy changes) [4].

The IPC mechanism permits re-establishment of control of a communication channel dynamically. If security policy changes for s's access to d's objects, then the redirection controller may reset the redirection function for $s \Rightarrow d$ to $R(s,d) = M$. This enables M to modify d's representation of s's security policy or shut off communication between s and d altogether.

4.2. Clans & Chiefs

We now demonstrate the flexibility of our mechanism by specifying a Clans & Chiefs policy using IPC redirection. The Clans & Chiefs mechanism has the following properties: (1) each member of a clan may send an IPC directly to any other member of the clan; (2) an IPC sent to a member of another clan is redirected to the clan's chief; (3) an IPC destined for a member of the clan that originated from a process in another clan is redirected to the destination's chief; and (4) clans may be nested such that a chief may belong to a clan of another chief. Clans & Chiefs enables the implementation of a wide variety of security policies. For example, mutually distrustful monitors may control IPC both into and out of their clan. Also, sub-clans can be created whose chiefs enforce different security policies.

Consider Figure 2. In this scenario, $M1$, $M2$, and $M3$

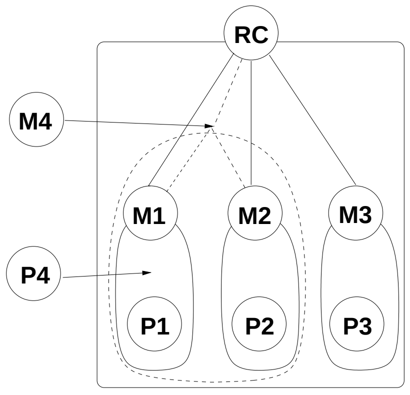

Figure 2. Simulation of Clans & Chiefs model with 3 clans. Also, $P4$ is inserted in $M1$'s clan and the $M4$ clan is created dynamically (not possible in the standard Clans & Chiefs model).

are monitors that correspond to chiefs in a Clans & Chiefs implementation. Each monitor control the flow of IPCs to and from a single process. RC is the redirection controller as before (and in the Clans & Chiefs model, RC is the chief of $M1$, $M2$, and $M3$). To simulate the Clans & Chiefs semantics, RC sets the redirection functions of the individual processes as follows:

- **P1**: $R(P1, P2) = M1, R(P1, P3) = M1$
- **P2**: $R(P2, P1) = M2, R(P2, P3) = M2$
- **P3**: $R(P3, P1) = M3, R(P3, P2) = M3$

Similarly, the chief's IPCs are redirected:

- **M1**: $R(M1, P1) = P1, R(M1, P2) = M2, R(M1, P3) = M3$
- **M2**: $R(M2, P1) = M1, R(M2, P2) = P2, R(M2, P3) = M3$
- **M3**: $R(M3, P1) = M1, R(M3, P2) = M2, R(M3, P3) = P3$

The chiefs can all intercommunicate directly because they are in RC's clan.

If a new process is entered into a clan, then its redirection functions are set according to the clan that it is entered. Consider a $P4$ that is added to $M1$'s clan. Its redirection functions are

$$R(P4, P1) = P1, R(P4, P2) = M1, R(P4, P3) = M1$$

The redirection functions of other processes are analogous.

In addition, RC may introduce a new process $M4$ such that $M1$ and $M2$ are added to $M4$'s clan. Therefore, $M1$ and $M2$ can no longer communicate directly with $M3$, but instead,

$$R(M1, M3) = M4, R(M2, M3) = M4$$

and vice versa for $M3$ as the sender. Adding a new chief is not possible in the L4 implementation of Clans & Chiefs system without recreating the processes.

In addition, relaxations of the Clans & Chiefs semantics that enable better performance are possible. For example, it may not be necessary for both $M1$ and $M2$ to authorize IPC between $P1$ and $P2$. They can share read access to the security database, so $M1$ can authorize $P1 \Rightarrow P2$ and $M2$ can authorize $P2 \Rightarrow P1$. Because $P1$ and $P2$ are controlled via different security policies, it may be preferable to keep one monitor for each. Of course, the optimization of the previous section (enabling $R(P1, P2) = P2$) can also be implemented.

5. Implementation

Implementation of our redirection mechanism requires that the kernel be extended to: (1) store and act on the redirection information, (2) provide an interface for the redirection controller to modify redirection information. This section focuses on the first requirement rather than the second, as we expect the second requirement's contribution to the mechanism's overhead to be small. We envisage redirection policy to be infrequently modified compared to the frequency of IPC itself.

The efficiency of the implementation of redirection itself is of paramount importance as it is potentially applied to every IPC. The obvious naive implementation of redirection is to store R in a two-dimensional array. Each IPC would involve an array lookup based on source and intended destination to produce the redirected destination. This method has potential for fast lookup as it involves a simple calculation based on readily available information in the IPC code path, together with a single memory reference. However this method is space inefficient requiring $N^2 \log_2 N$ bits, where N is the maximum number of processes.

A better strategy for implementation is to consider R a translation function analogous to virtual to physical address translation in virtual memory implementations. Virtual to physical address translation via page tables is well-understood (see Jacob [3] for a list of references). R would be analogous to a sparse virtual address space which is best translated by an inverted page table (i.e., hash table). Inverted page tables also have the advantage of less impact on the data cache compared to hierarchical arrangements.

The protocol works as follows. Given the identity of the source and the destination processes, the kernel can attempt to retrieve the redirection data entry from the inverted "page table." If there is no redirection entry for the combination of source and destination, then the kernel has a *redirection fault*, upon which it forwards the IPC to the redirection controller of the source. The redirection controller can block, revise, or forward the IPC (i.e., act as an interim destination). Additionally, it can set one or more entries of redirection data to redirect future IPC to the appropriate destination. We expect that redirection data will change infrequently, so few redirection faults will occur.

Using the above mechanism, the cost of applying R to any IPC is approximately the cost of performing a TLB refill in software per IPC, without the TLB-specific overhead. This is approximately 10 instructions depending on the architecture, plus the cost of referencing the redirection data in memory (if it is not in the cache). This analysis assumes that the redirection hash table size is tuned so as to achieve a high ratio of first probe hits. This is not unreasonable if hits resolved via entries in collision resolution chains result in the entry being promoted to the front of the hash chain. Our expectation is that such a hash table will be significantly smaller than the naive implementation which always achieves a lookup in a single probe.

Currently, L4 has the best IPC performance. An analysis of the inherent cost of performing IPC [7] revealed that in a typical implementations, register-only IPC consists of 50-80 instructions contained in 6-14 instruction cache lines, which accesses 4-6 cache lines of data.

Our redirection mechanism, if added to the L4 IPC path, would increase the number of instructions by approximately 12-20%. In the absence of cache misses, we would expect a similar increase in the number of cycles needed to perform IPC.

If cache misses are prevalent, cache miss cost is the dominant contribution to IPC cost. Liedtke reports Pentium IPC times of 295 cycles assuming the maximum number of L1 cache misses during IPC, but no L2 cache misses. The ideal time (i.e., no cache misses) is 121 cycles. In this situation, one can consider the cache footprint as a rough estimator of IPC performance. Our redirection mechanism increases the instruction cache footprint by approximately 12-33% (2 cache lines), and data cache footprint by approximately 16-25% (1 cache line).

Based on this simple analysis, we expect our redirection mechanism to add less than 50% overhead to L4 register-based IPC. Indeed, we hope to achieve an average overhead of approximately 20%. For IPC involving messages in memory, which is much more expensive, we expect the overhead of our redirection mechanism to become insignificant.

6 Summary

In this paper, we presented a mechanism that uses kernel redirection to provide a basis for efficient access control. This mechanism enables the removal of access control management from the kernel, so a variety of system-specific representations and mechanisms can be implemented. We demonstrated how the mechanism can be used to implement a reference monitor architecture that satisfies our security requirements and an architecture that implements Clans & Chiefs' semantics. In general, we expect that an efficient implementation is possible that may add only 20% to L4 register-based IPC time and have a negligible performance effect on long (i.e., greater than 8 byte) messages. Future work includes the examination of the actual performance effects of the mechanism, the extension of the model to support hierarchical redirection controllers, and application of this mechanism to other services, such as debugging.

References

[1] J. P. Anderson. Computer security technology planning study. Technical Report ESD-TR-73-51, James P. Anderson and Co., Fort Washington, PA, USA, 1972.

[2] E. Gabber, C. Small, J. Bruno, J. Brustoloni, and A. Silberschatz. Building efficient operating systems from user-level components in Pebble. In *Proceedings of the 1999 USENIX Annual Technical Conference*, 1999. To appear.

[3] B. L. Jacobs and T. N. Mudge. A look at several memory management units, TLB-refill mechanisms, and page table organizations. In *ASPLOS-VIII*, 1998.

[4] T. Jaeger. *Secure Internet Programming: Security Issues for Distributed and Mobile Objects*. Springer-Verlag, 1999. To appear.

[5] T. Jaeger, J. Liedtke, and N. Islam. Operating system protection for fine-grained programs. In *Proceedings of the 7th USENIX Security Symposium*, pages 143–156, Jan. 1998.

[6] J. Liedtke. Clans & chiefs. In *Architektur von Rechensystemen*. Springer-Verlag, Mar. 1992. In English.

[7] J. Liedtke, K. Elphinstone, S. Schonberg, H. Hartig, G. Heiser, N. Islam, and T. Jaeger. Achieved IPC performance. In *Proceedings of the Sixth Workshop on Hot Topics in Operating Systems*, pages 28–31, May 1997.

[8] S. E. Minear. Providing policy control over object operations in a Mach-based system. In *Proceedings of the 5th USENIX Security Symposium*, 1995.

[9] R. Rashid, A. Tevanian Jr., M. Young, D. Golub, D. Baron, D. Black, W. J. Bolosky, and J. Chew. Machine-independent virtual memory management for paged uniprocessor and multiprocessor architectures. *IEEE Transactions on Computers*, 37(8):896–908, Aug. 1988.

[10] J. E. Tidswell and J. M. Potter. Domain and type enforcement in a micro-kernel. In *Proceedings of the 20th Australasian Computer Science Conference*, 1997.