A COMMENT ON THE CONFINEMENT PROBLEM

Steven B. Lipner
The MITRE Corporation
Bedford, Massachusetts

The confinement problem, as identified by Lampson, is the problem of assuring that a borrowed program does not steal for its author information that it processes for a borrower. An approach to proving that an operating system enforces confinement, by preventing borrowed programs from writing information in storage in violation of a formally stated security policy, is presented. The confinement problem presented by the possibility that a borrowed program will modulate its resource usage to transmit information to its author is also considered. This problem is manifest by covert channels associated with the perception of time by the program and its author; a scheme for closing such channels is suggested. The practical implications of the scheme are discussed.

Key Words and Phrases: protection, confinement, proprietary program, security, leakage of data

CR Categories: 2.11, 4.30, 4.35

1. INTRODUCTION

In [4] Lampson identified the confinement problem--the problem of assuring that a "borrowed" program (which Lampson called the service) was incapable of stealing information that it processed on behalf of a borrower (called the customer). Lampson illustrated the confinement problem by a series of examples and suggested a set of principles to apply to computer systems that would solve the problem.

The purpose of this note is to suggest that current research results in computer security allow a more precise characterization than Lampson's of the confinement problem and of principles for its solution in the context of a well-defined policy. To that end, this note discusses the methods that have been used to verify that confinement requirements are met by a real system. This paper also comments briefly on the covert channels identified by Lampson that are very difficult to block in a practical multiprogrammed computer system.

2. A REVIEW OF THE PROBLEM

In [4] Lampson identified six examples of mechanisms by which a service might attempt to violate the confinement constraints imposed by a customer. Of these, four were characterized as storage channels involving the writing by the service of storage locations that might be accessed by its (the service's) author to access the customer's data. The specific examples of storage suggested include own variables, temporary and permanent files, interprocess communication messages, and file interlocks (semaphores).

A second class of channel identified by Lampson was the legitimate one--the channel used by the service to pass billing information to its author. The third class of channel identified was the covert one, involving modulation of a computer's processor usage or paging rate (for example), by the service in a manner detectable by its author.

Section 3 below discusses the use of security policy models as a criterion for confinement. Section 4 describes a technique for closing the storage and legitimate channels while Section 5 considers the problem presented by the covert channels. Section 6 summarizes some experience with confinement in real systems and Section 7 presents some conclusions on the current status of confinement and the likely course of future developments.

3. CLOSING THE CHANNELS

Lampson suggested that the would-be developer of a secure system must enumerate each specific channel and then block it in order to solve the confinement problem. One purpose of this paper is to point out that orderly approaches are available for the achievement of that end.

A number of researchers [1] [15] [10] have developed models for the protection of sensitive information in computer systems. Each of the models referenced characterized a computer system (more or less formally) in terms of access by subjects (active system elements, processes) to objects (units of information). Each model also associated with each subject and object a "clearance" and "classification" respectively. In [1] and [15] clearance, classification, and the rules for comparing them were drawn from the security level and formal compartment rules of the military security system. In [10], a different descriptive language is used, but the model is essentially similar to the formal compartment rules of [1] and [15].

Given the basic identification of subjects, objects, clearances, classifications, and access rules, each model suggests a rule for keeping a dynamic record of the classification of each object. In [1] the rule, called the *-property, forbids a subject from writing in an object unless the classification of the object is greater than or equal to that of every object the subject has read since its creation and thus may "remember" as part of the state of its computation. In [15], the "high water mark" rule sets the classification of any new object that a subject creates to that of the highest classified object the subject has read. The rule of [10] is similar to the *-property in that information may only be written to objects bearing the same set of sensitivity compartments as the most sensitive of the objects read since subject creation.

The motivation of each of the rules identified is precisely to address the confinement problem. Each model assures that, with respect to a characteristic of special interest (security level and compartment for example) no service operating on behalf of a subject can reduce the sensitivity of information by writing it into any object. The choice of the characteristic is based on its importance to the operators and users of the system. In the military context, for example, enforcement of classification and compartment restrictions is non-discretionary and confinement is of considerable importance. Within the classification/compartment restrictions, other access decisions are discretionary (viz."need-to-know")and confinement is not critical.

Either the *-property or high water mark, if applied on every access by a subject to an object, can solve the confinement problem for storage channels. No matter what the service attempts to do, it may not write an object less sensitive than its input from the customer. If the author of the service had inadequate clearance to see the objects input to it by the customer, he will not have the clearance required for access to any objects that the service can output.

4. IDENTIFYING THE OBJECTS

The rules of the *-property and high water mark simply formalize the confinement requirements of [4]. The problem still remains for the designer of a secure system to assure himself that he has enumerated every object--a similar requirement to enumerating and blocking every channel. Fortunately, available design approaches simplify and structure the required enumeration.

The design methodology of [9] decomposes the design of an operating system into the formal specification of a series of hierarchical levels. Each formal specification is proven internally consistent with respect to the assumptions of individual functions or modules, and correct with respect to a set of assertions. Assumptions of higher levels dictate assertions about lower levels -- thus, lower levels are proven to implement higher ones correctly. The formal specifications are similar to Parnas specifications [7] and are composed of o-functions that perform operations and v-functions that store and return state values. All information accessible to users of the operating system and hardware is represented by v-functions of the highest level specification and by the "error call" returns of the o- and v-functions; all possible calls by users on the operating system and hardware are represented as o-functions of the highest level specification.

Since the highest level specification represents the complete environment accessible to a subject, a proof that the *-property (for example) is met by that highest level specification is a proof that the confinement problem has been solved for the specific security policy represented by the *-property. It has been recognized by [12] [2] and [5] that a proof that the highest level specification meets the *-property can be constructed by characterizing every instance of a v-function (a v-function with the arguments specified) of the highest level by its classification and then proving that no o-function performs an operation that violates the *-property.

In practice the proof proceeds by examining the effects section of each o-function to determine what v-functions are read and then demonstrating that no effect that alters a v-function of a given classification can depend on (read or be conditional on) a v-function of a higher classification. In this proof, each instance of a v-function becomes an object and each invocation of an o-function a subject. The proof demonstrates that the *-property holds at the level of specification subjects and objects. The proof must also assure that no error call of a function can depend on a v-function that the process causing the error could not access.[1] No

[1] In the security kernel mentioned below, error calls were replaced by settings of a "return code" v-function that was, of course, proven to have a security level consistent with the *-property.

program can access information except that which is represented by a v-function of the highest level, and the specification identifies all members of the (finite) set of instances of v-functions. Thus the proof mechanizes the enumeration of all channels required by [4].

The proof that a given o-function satisfies the *-property may generate a relation that must hold among the values of several v-functions. Such a relation may be proven to be preserved by all other o-functions of the highest level using an inductive technique developed by Price [8].

The set of o-functions and v-functions of the specification addresses the entire range of information available to a process. This range covers not only memory locations in segments (whether in main memory or on secondary storage), but also a variety of objects to which the operating system provides interpretive access, including:

Interprocess communication channels;
Semaphores; and
Directories.

Even information "about information" is covered by the specification (and thus the *-property). Thus the directory entries containing access control lists and date-time-used (time of last read or write access) for segments and directories may only be updated consistent with the *-property. For some types of information such as date-time-used, what would appear to be a read access (to a segment) may also imply a write access (to the date-time-used). Such types of information may have to be stratified by classification, redefined (for example only date-time-written is recorded) or simply eliminated from a secure system.

An attempt to prove that the *-property holds in a top-level specification will only succeed if the environment includes only virtual (per-subject or per-process) resources (v-functions or objects) of the appropriate classification. The disk map for a virtual memory sytem, for example, is a system-wide resource and can be affected by any subject (for example when it creates a segment). Thus it has a "system high" classification (that of the most sensitive information in the system) and must be hidden from all but the most highly cleared (in practice, from all) subjects or processes. The required hiding of information is allowed by the specification technique of [9] as an aid to orderly well-structured design. In a secure system, a designer must write specifications so that all effects depending on disk address occur at lower levels, and are mapped into physical resource-independent effects at the highest level. If he does not, he will be unable to prove that the lower levels correctly implement the highest one.

5. A DIFFICULT PROBLEM

The approach outlined above addresses the storage channels of [4]. It also completely blocks all legitimate channels except in the case where the author of the service holds clearance for the information processed by the customer.[2] The only way to reacquire the legitimate channels is to provide an exception to the *-property (or high water mark) for a specific class of isolated, well-defined and certified programs. These programs may be the customer-determined billing programs of [4] or the restricted-granularity billing programs of [10]. The certification of these programs must assure that, as they are freed from the external restraints of the *-property by the operating system, they impose it on their own internal operations and do not leak sensitive information to users who are not allowed to access it. Alternatively, billing may be handled manually by trusted individuals operating outside the computer system.

The covert channels of Lampson are associated with the one system-wide resource, time, that can be observed in at least a coarse way by every user and every program. Each user (presumably) has a wristwatch and each program can tell how many instructions it executes before calling for input-output or touching a new page.

To close the "covert" channel, each subject must be constrained to see "virtual time" depending only on its own activity. A virtual clock can be associated with each process, assigned a classification in accordance with the *-property, and altered as needed by the effects section of each o-function. This requirement means, for example, that the virtual time required when a process touches a new page must be a constant and cannot depend on the possibility of another process having brought the page into main memory. (All page faults require the same virtual paging delay.) Similarly, virtual processor time must be made available to the virtual environment according to a criterion depending only on that virtual environment's activity.

The use of "virtual time" as outlined above seems to solve the covert channel problem. However, it is not clear that one can cause a process to operate in a completely virtual time environment and make this environment the one that is shown to users. Each user has a perception of real time that is independent of the virtual environment provided within the computer. For the confinement problem to be solved completely, the author of a service must not only see that his job requires a given number of virtual CPU-seconds, he must also be prevented from drawing any conclusion from the real response time of that job. In a batch processing system where jobs are returned at

[2] This case is relatively uninteresting for if the author of the service is cleared for the customer's data, he probably has more direct ways to access it -- for example, by asking the customer if he may see it.

fixed intervals, this restriction is simple to enforce. However, in a time-sharing system that performs adaptive scheduling and demand paging, it is not clear that the operating system can prevent the author of a service from correlating virtual and real time to at least some degree. Eliminating this correlation would seem to require eliminating any degree of adaptive resource-sharing -- for example by providing each user a CPU (or CPU time-slice) and fixed real memory resources. The price of enforcing this level of confinement is probably too great for any real system.

While the storage and legitimate channels of Lampson can be closed with a minimal impact on system efficiency, closing the covert channel seems to impose a direct and unreasonable performance penalty. It seems likely that some attempt at randomizing the relation of virtual and real time, as suggested by Lampson, will be made to make the covert channel noisy, instead of resorting to the measures required to close it as completely as possible.

6. EXPERIENCE

A prototype security kernel for the PDP-11/45 was initially designed using the model of [1] to specify a policy for the control of access to segments, but no controls at all on objects such as access control lists, semaphores and physical addresses [13]. When a variety of storage channels were discovered in this kernel, a redesign was undertaken that applied the model of [1] to all objects visible to the user, but failed to conceal completely the state of some physical resources. (In particular, users could detect the "disk full" condition by trying to create segments.) Experience with the design of a large system which, while not proven secure, did enforce the *-property on the activities of user programs [3] had by then shown that the environment had to be completely virtual, and that physical resource availability must be controlled by per-process (or per-classification) quotas. Subsequently the revised design [14] was formalized using the methodology of [9] and specification proofs. Two levels of specification were used to isolate from users' processes the system-wide resource data needed to implement the system. Since proofs of the *-property as applied to the kernel specification were completed, no new channels of these types have been found.[3] No attempt was made to close the covert channel in this effort.

Saltzer [11] has reported several attempts to build and measure covert channels on Multics [6]. These attempts involved processes "banging on the walls" of the confined environment via a combination of timing and paging rate. A channel of the order of a bit per second has been demonstrated, and channels of the order of tens of bits per second hypothesized.

[3] Of course, we recognize that not finding such channels does not prove their absence. Our belief that storage channels have been eliminated from the system derives from the formal design method and proofs, not from an ad hoc search for channels.

7. CONCLUSIONS

Current security design techniques are adequate to allow us to prove formally that storage and legitimate channels have been closed in a real system. The techniques that have been applied with the specific security policy of [1] can also be applied with at least some other policies. What is required is to provide an explicit model for the desired policy, and a proof that the requirements of the model are met by the specifications for all functions that are accessible to the confined programs. A remaining question is what set of policies can be the basis of such proof, and how to describe and characterize them.

Closing the covert channels seems at a minimum very difficult, and may very well be impossible in a system where physical resources are shared. Ad hoc measures can probably be of value here.

ACKNOWLEDGEMENTS

This paper reflects the suggestions of Edmund Burke and W. Lee Schiller of the MITRE Corporation and of Major Roger Schell and Lieutenants William Price and Paul Karger of the Air Force Electronic Systems Division. The preparation of the paper and much of the work reported were sponsored by the Electronic Systems Division under Contract F19628-75-C-0001, Projects 522B and 572B.

REFERENCES

1. Bell, D. Elliott and LaPadula, Leonard J. Secure computer systems. ESD-TR-73-278 (AD 770768, 771543, and 780528) The MITRE Corporation, Bedford, Massachusetts (November 1973).

2. Burke, Edmund L. Private communication--Burke and Schell seem to have devised the scheme of applying the *-property to variables inside a security kernel during late 1972 or early 1973.

3. Honeywell Information Systems. Design for Multics security enhancements. ESD-TR-74-176, Electronic Systems Division (AFSC), L. G. Hanscom AFB, Massachusetts (1974).

4. Lampson, Butler W. A note on the confinement problem. Communications of the ACM 16, 10 (October 1973), 613-615.

5. Millen, Jonathan K. Security kernel validation in practice. MTR-2932, Vol. 2, The MITRE Corporation, Bedford, Massachusetts (In preparation).

6. Organick, Elliott I. The Multics System; An Examination of its Structure. The MIT Press, Cambridge, Massachusetts, 1972.

7. Parnas, David L. A technique for software module specification with examples. Communications of the ACM 13, 5 (May 1972), 330-336.

8. Price, William R. Implications of a virtual memory mechanism for implementing protection in a family of operating systems. PhD Thesis, Carnegie-Mellon University (June 1973).

9. Robinson, L., Neumann, P. G., Levitt, K. N., and Saxena, A. On attaining reliable software for a secure operating system. Proceedings of the 1975 International Conference on Reliable Software., Los Angeles, California (April 1975) 267-284.

10. Rotenberg, Leo J. Making computers keep secrets. MAC-TR-115, Massachusetts Institute of Technology, Cambridge, Massachusetts (February 1974).

11. Saltzer, Jerome H. Private communication. (April 1975).

12. Schell, Roger R. See reference [2].

13. Schiller, W. Lee. Design of a security kernel for the PDP-11/45. ESD-TR-73-294 (AD 772808), The MITRE Corporation, Bedford, Massachusetts (December 1973).

14. Schiller, W. Lee. The design and specification of a security kernel for the PDP-11/45. ESD-TR-75-69 (AD A011712), The MITRE Corporation, Bedford, Massachusetts (March 1975).

15. Weissman, Clark. Security controls in the ADEPT-50 time-sharing system. AFIPS Conference Proceedings 35 (FJCC 1969) 119-133.