Project MAC
Computer Systems Research Division

July 11, 1974
Request for Comments No. 56

A Security Kernel Model for Page Control

by R. J. Feiertag

This paper describes a gedanken experiment in isolating those elements of page control which are necessary to assure system security. These elements can then be included in a system security kernel whose correctness is certified to assure that page control will operate securely. The remainder of page control may then be implemented as an uncertified part of the system without loss of security.

The work described here is the preliminary part of a larger effort intended to define a security kernel which contains capabilities sufficient to build a Multics like operating system, yet which can be proven to meet some useful security criteria.

For the purposes of this paper a security kernel is that part of the system which assures secure operation of those facilities provided by the system. There are many other possible definitions of kernel, but this one serves well for this experiment. The choice of which facilities are provided by the system itself is crucial to the design of a kernel by this definition, however, that problem is beyond the scope of this paper. Page control was chosen as the object of this experiment because it is likely to be a facility provided by many systems and because it can be easily and well defined.

Definition of security

A system operates securely if there can be:

1. no unauthorized release of information,
2. no unauthorized modification of information,
3. and no unauthorized denial of service.

In systems in which resources are multiplexed among processes in a manner observable in some form by the processes (e.g. a user can observe the time delay caused by a page fault which results from multiplexing primary memory) there is no known method of attaining (1) above. This is because a process having authorized access to certain information can pass that information to an unauthorized process by encoding the information in its use of the multiplexed resource in a manner observable by the unauthorized process. Although such information paths may be very noisy and difficult to exploit, there is no known technique for assuring that no such paths exist. For this reason the proposal presented below will not attempt to fully realize security objective (1) above, but will realize the more limited objective of preventing unauthorized direct access to

---

information. Since most of page control will not be in the certified kernel it will be assumed that page control is unauthorized and will, therefore, not be allowed access to the information contained in pages. The implications of the above restriction on security objective (1) in the case of this experiment will be discussed later.

In this paper, paging is considered to be a facility provided by the system and page control is the part of the system that provides the paging service. Since page control provides the paging service it is not possible to prevent page control from denying the paging service. Therefore, in order to assure security objective (3) it is necessary to include all of page control in the certified kernel. This increases the size of the kernel and makes certification of the kernel more difficult. For this experiment it was decided to leave page control outside the kernel and sacrifice security objective (3). It will be shown that this permits a considerable reduction in the size of the kernel. It is hoped that allowing page control to deny service will not cause more serious security compromises in other parts of the kernel that use page control. These and other implications of sacrificing security objective (3) will be the object of further work. The model for a kernel described below will, therefore, address the restricted security objective (1) and security objective (2).

A model for paging

The model described below is in a preliminary state of development and is, therefore, not yet complete. This is primarily because I wish to include the paging model in a larger model which will include other system functions and, in that context, it is not yet clear how to complete the model. However, the model as presented below does yield some interesting results.

The basic unit of reference in the model in the PAGE. The PAGE is the element that holds information or data. A PAGE consists of several COPYs. Each COPY contains the data that was associated with the PAGE at some point in time. There is always at least one COPY that has the data associated with the PAGE for the current time. A COPY has a set of attributes, described later, and has a FRAME. At any given point in time the memory of the system (this includes all forms of storage used in paging) is divided into disjoint blocks called frames. A FRAME has a set of attributes and a place to hold the DATA of a COPY. In summary the basic units of the model and their constituents are:

PAGE		definition of a page object
NUMBER_OF_COPIES	integer	
COPY(NUMBER_OF_COPIES)	COPY	array of COPYs
COPY		definition of a copy object
COPY_ATTRIBUTES	bit	binary attributes
FRAME	FRAME	frame defined below

FRAME		definition of a frame
FRAME_ATTRIBUTES	bit	binary attributes
DATA	bit	array of bits

One of the COPY_ATTRIBUTEs is named CURRENT. If a COPY is CURRENT then the DATA of the FRAME associated with this COPY contains current data for the PAGE to which the COPY belongs, i.e. this COPY is up to date. For the remainder of this paper, when I refer to the DATA of a PAGE, I mean the DATA of a FRAME of a CURRENT COPY of the PAGE.

The DATA in a PAGE is broken up into a contiguous array of bits. The bits are identified by sequential positive integers with the first bit being numbered 0.

The two basic operations on the DATA of a PAGE are read and write. The operation

read (PAGE , offset)

returns the contents of the bit of DATA specified by the integer, offset, in the given PAGE. Since it is possible for more than one COPY of a PAGE to be CURRENT and since the read operation should be deterministic, it is necessary that all CURRENT COPYs of a PAGE must have identical DATA. That this property always holds will be shown later. Since the DATA of all CURRENT COPYs must be identical the read operation can read the DATA of any CURRENT COPY.

The operation

write (PAGE , offset , value)

modifies the bit of DATA specified by offset in the given PAGE. In order to maintain the DATA of all CURRENT COPYs of a PAGE identical, the write operation modifies all CURRENT COPYs.

The read and write operations are part of the system kernel. They will be used by other areas of the kernel such as parts of segment control and access control to produce the final read and write operations visible to the user. They serve here to illustrate the restrictions that page control must place upon the read and write operations. The only way to access or modify information in a PAGE is to use either read or write.

In order to permit page control to perform its function properly, it must be able to manipulate COPYs of PAGEs. At various times, COPYs of PAGEs will reside in primary memory, cache memory, secondary memory, etc, and page control must see that the COPYs are in the appropriate places at appropriate times to try to attain maximum efficiency. In order to permit page control to perform these functions easily two new kernel functions are provided.

make copy (PAGE.COPY(n) , FRAME)

causes the creation of a new COPY for PAGE at the given FRAME with the DATA and CURRENT attribute of the current nth COPY of

PAGE. That is, make copy creates a new COPY identical to an already existing COPY except that the DATA is located in a different place in physical memory.

delete copy (PAGE.COPY(n))

causes the nth COPY of a PAGE to be deleted.

In addition there are two operations to establish those COPYs which are current:

set current (PAGE.COPY(n))
remove current (PAGE.COPY(n))

These above operations on PAGEs, FRAMEs, and COPYs are sufficient to allow page control to implement a wide variety of possible paging schemes. I will show later how these operations can be used to implement the Multics paging scheme.

Security of the model

Before demonstrating the applicability of the model, I will demonstrate its security. At this writing, no formal proof of security has been obtained, however, the following discussion should convince the reader that a formal proof is possible without undue difficulty.

I wish to demonstrate that the kernel operations defined above do not violate the modified definition of security defined above. The first modified security objective is that there be no unauthorized direct access to information. The only operation of the six defined above that accessses information contained in pages is read. However, as stated earlier, the read operation defined above is not directly accessible to page control, but instead is filtered by access control to permit only authorized read operations. Therefore, non-kernel programs, including page control, do not have access to arbitrary pages, but only to pages which are authorized to them.

The other security objective we wish the kernel to assure is that there be no unauthorized modification of information. The only operation of the six above which permits arbitrary modification of information contained in pages is write. For the same reasons as with the read operation, no unauthorized calls to write can occur. Unfortunately, there are other means of modifying information in a limited way. For example, a program could use the make copy operation to make a COPY of a PAGE in a FRAME already belonging to a COPY of a different PAGE thereby modifying the latter. Or a program could set current a COPY of a PAGE when the DATA in that COPY is not the most recent version of the PAGE, thereby modifying the PAGE to a previous form. It is clear that, like read and write, the other four kernel operations must be more carefully controlled.

This is done by introducing two new attributes. First, we introduce FREE as a FRAME_ATTRIBUTE. If a FRAME is FREE then it can be used to make a new COPY of a PAGE, i.e. it can be used as the second argument to make copy. The make copy operation then

removes the FREE attribute from the FRAME so that it cannot be overwritten until the COPY is deleted. The delete copy operation requires that the FRAME associated with the COPY being deleted not be FREE. The delete copy operation makes the FRAME FREE after deleting the COPY.

The second new attribute is a COPY_ATTRIBUTE called MOST_RECENT. A COPY is MOST_RECENT if no other COPY of the same PAGE has been modified more recently than this COPY. As with CURRENT COPYs, all MOST_RECENT COPYs must have identical DATA. When a PAGE is created all of its COPYs are MOST_RECENT. When a new COPY of a PAGE is created make copy will make the new COPY MOST_RECENT only if the COPY being duplicated is MOST_RECENT. Only COPYs which are MOST_RECENT can be made CURRENT, and this is enforced by the set current operation. And finally, in order to assure that MOST_RECENT COPYs are truly the most recent, the write operation revokes the MOST_RECENT attribute on all COPYs that are not CURRENT, i.e. those COPYs that are not modified. These restrictions insure that read and write always access consistent and the most up-to-date COPYs.

At first glance one might believe that the CURRENT and MOST_RECENT attributes are the same. However, the MOST_RECENT attribute is maintained by the kernel and is needed to constrain the CURRENT attribute which is settable by non-kernel procedures. The set of CURRENT COPYs is a subset of the set of MOST_RECENT COPYs. This will be made clearer later when the model is applied to Multics.

More on security modifications

At the beginning of this paper I argued that the unmodified first security objective was unachievable and, therefore, I modified the objective to make it achievable. It is interesting to note now, that the four kernel operations directly accessible to page control (i.e. make copy, delete copy, set current, and remove current), do not release any information whatsoever. None of these four operations return any values. The model thus described does meet the unmodified first security objective. However, without any information, page control cannot function, so there must be some means by which page control can get information about PAGEs, COPYs, FRAMEs, etc. This information can be provided explicitly by an additional kernel operation. Once page control has this information there is no way the kernel can stop it from releasing the information. However, the information released has been strictly controlled by the kernel. Of course, the more sophisticated the page control algorithms, the more information they are likely to need. However, the trade-off is very explicit. The kernel will release to page control the information required by page control and only this information is subject to further release.

I also stated earleir that the third security objective, prevention of denial of service, is unachievable without a large amount of page control in the kernel. I went on to say that abandonment of that objective allows that part of page control in the kernel to be very small. It is now clear exactly how much of

page control must be in the kernel, namely the data about PAGEs, COPYs, and FRAMEs given in Appendix A and the operations given in Appendix B. These operations are all quite straightforward. What does not have to be included in the kernel are the strategy algorithms such as allocation of page frames to pages and the page removal algorithms. These strategy algorithms often tend to be complex requiring the manipulation of a lot of data and are, therefore, difficult to certify.

A question of interest arises when we observe that other parts of the kernel will wish to use page control to manage their pages. The question is how the modifications made to the security objectives for the page control part of the kernel will affect the other parts of the kernel using page control. Does the fact that the page control part of the kernel does not prevent denial of service simply mean that other parts of the kernel can be halted or can other security objectives be compromised in the kernel as well?

An example: Multics page control

In order to give the reader a better feel for this model I will now briefly describe how the model relates to Multics page control. To do so in detail would require much writing which the reader familiar with Multics page control can deduce for himself or herself. I will therefore simply sketch the more important points.

I begin by noting that the operations described above make no allowances for errors. What happens if a program attampts to make a new COPY in a FRAME that is not FREE or tries to make CURRENT a COPY that is not MOST_RECENT? The purist point of view would say that such errors sould not occur because the non-kernel page control should have obtaied enough information from the kernel so that it always takes correct action. Any incorrect call to the kernel should be ignored. A less pure approach would be to report errors either by error code return argument or condition type mechanism as in PL/1. One must be wary of these techniques because they can release information and one must certify that such release is proper.

The problem is not so simple if one carefully considers the read and write operations. One error that can arise in these operations is that an attempt is made to read or write a PAGE, but no COPY of that PAGE is CURRENT. This corresponds to a page not being in primary memory in Multics. One could require that all PAGEs have at least one CURRENT COPY or that a PAGE have a CURRENT COPY before it is accessed. The former of these possibilities is impractical in practice and defeats the purpose of paging and the latter solution precludes demand paging. It is therefore necessary to insist that some error mechanism exist in order to permit demand paging to occur. This error mechanism could be an error code returned by read and write indicating that no CURRENT COPY for the PAGE exists in which case the caller of read or write must then invoke page control to make a COPY CURRENT and then reinvoke read or write, or it could be some kind of mechanism in read and write which immediately calls page

control with an indication of the PAGE being referenced. Even though this call would come from within the kernel, non-kernel page control would still not have kernel privileges. This latter technique corresponds to the page fault mechanism of Multics. Both mechanisms do release information, however, the information is necessary for the proper functioning of page control and cannot be withheld.

Implementing the Multics page control algorithm in terms of these kernel operations is quite straightforward. When a page fault (no CURRENT COPY error) occurs, page control is called. A new COPY is made to a FRAME which is in primary memory from a COPY with a FRAME on the paging device, or if there is none on the paging device, from the disk. Since the paging device always has more up-to-date COPYs than the disk this will assure that the new COPY is MOST_RECENT. Once the COPY is in primary memory it can be set current. Note that in Multics each PAGE has only one CURRENT COPY (two COPYs would be CURRENT if Multics had a cache with store through on writes). Multics also imposes the further restriction that a copy must be in primary memory in order to be CURRENT. A CURRENT COPY on disk makes no sense because it cannot be directly accessed. When a COPY is to be removed from primary memory, a check is made to see if there is a MOST_RECENT COPY on the paging device or the disk. If so, the COPY in primary memory is simply deleted. This case corresponds to the modify bit off in the page table word. If there is no other MOST_RECENT COPY then a new MOST_RECENT COPY is made on the paging device or disk and the primary memory COPY is deleted. The removal algorithm for the paging device is similar (there is the added complication for the paging device that COPYs must be temporarily moved to primary memory and then to the disk, but this presents no difficulties to the model).

The implementation of the page control kernel for Multics can be accomplished without much trouble on the existing hardware. Unfortunately in the present implementation of Multics, the information about PAGEs, COPYs, and FRAMEs which must be kept by the kernel is quite spread out. The list of COPYs of a PAGE is scattered among the page tables, core map, and paging device map. The CURRENT attribute of a COPY is indicated by the lack of a directed fault in the page table word. The MOST_RECENT attribute of a COPY is encoded in the modified bit of the page table word and the modified_from_disk bit in the paging device map. The FREE attribute of a FRAME is in the core map for primary memory, the paging device map for the paging device, and in the File System Device Control Table for the disk. This information must be collected so that it can be protected by the kernel. For easier access some of the information might be represented differently. Of course, a significant redesign could greatly simplify the kernel. In any case, it is fairly clear that the kernel described above could be implemented on Multics in an efficient manner.

An anomaly

I have just shown that Multics page control can be easily implemented in terms of the kernel operations described. All the

paging schemes with which I am familiar can be implemented in terms of these operations. Multics does, however, have an interesting feature that does not fit well into the scheme. This feature is the discarding of pages of zeroes.

When Multics page control is about to remove a page from primary memory, and that page contains only zeroes, instead of writing the page to the paging device or disk, it is simply discarded. Any page that has no copy of its data is then assumed to contain all zeroes so that when it is paged into primary memory it is simply necessary to provide a frame of zeroes. This feature is provided to save the costs of transferring the copy to and from secondary memory and keeping the copy in secondary memory.

Unfortunately this efficiency gain produces a security breach. In order for page control to know not to make a copy of the page on secondary storage it must know that the page contains zeroes. Giving the non-kernel portion of page control this information is a direct breach of the modified security objective (1). In order to preserve security objective (1), it would be necessary to place much more of page control within the kernel. This, if course, is counter to the desire that the kernel be small.

There appears to be a fundamental conflict here. One must trade efficiency for security. At least, in this case, the trade-off is fairly clear.

Completion of the model

This discussion has not considered the creation or deletion of PAGEs. This is because several different kernel operations can be used to provide these functions and I have not yet determined which alternative is most useful. For the sake of completeness I will briefly outline one set of alternatives, it is not intended as a permanent part of the model. The kernel operation create page creates a PAGE with one COPY with a new attribute NULL. This attribute indicates that there is no FRAME associated with this COPY. The COPY is MOST_RECENT, but is not CURRENT and cannot be made CURRENT. When make copy is invoked on this COPY it places zeroes in the new FRAME, makes the new COPY MOST_RECENT and deletes the NULL COPY automatically. create page returns the created PAGE to its caller. The operation delete page (PAGE) FREEs all FRAMEs of PAGE and deletes all COPYs and the PAGE itself. The operations create page and delete page are not directly callable by non-kernel procedures. As with read and write they are subject to constraints imposed by other parts of the kernel such as access control and segment control. In order to insure that page control can not, in effect, delete a PAGE, the delete copy operation will not delete the last MOST_RECENT COPY of a PAGE. The addition of the create page and delete page operations to the kernel do not affect any of the security arguments given earlier.

Finally I will say a few words about initialization and shutdown of the kernel. In the abstract model presented above, all that need be initialized are the kernel operations described

above (i.e. they have to be placed in directly addressable memory) and the list of FRAMEs. No PAGEs or COPYs need exist when page control is initialized. When the system is to be shut down, all PAGEs must be deleted.

Conclusion

This paper has presented a preliminary model for that part of an operating system security kernel dealing with page control. The significance of the model is that the author believes it both certifiably secure and easily implementable. It is not the only or optimum model, but simply serves to demonstrate the feasibility of this approach. Further research will broaden the model to include other operating system functions, improve the model (i.e. make it more easily certifiable and more easily and efficiently realizable), and explore the implications of the restrictions on security described at the beginning of this paper.

Appendix A - Data protected by the kernel

For each PAGE
1. A list of COPYs of that PAGE

For each COPY
1. Whether or not the COPY is CURRENT
2. Whether or not the COPY is MOST_RECENT
3 The FRAME associated with this COPY

For each FRAME
1. Whether or not the FRAME is FREE
2. The DATA of the FRAME

Appendix B - Required functions of kernel operations

read (PAGE , offset)
1. locate a CURRENT COPY
2. if no CURRENT COPY indicate error or call page control
3. return bit at indicated offset

write (PAGE , offset , value)
1. locate a CURRENT COPY
2. if no CURRENT COPY indicate error or call page control
3. remove MOST_RECENT attribute from all non-CURRENT COPYs
4. replace bit indicated by offset with value in all CURRENT COPYs

make copy (COPY , FRAME)
1. if FRAME not FREE return
2. copy DATA from COPY to new FRAME
3. remove FREE attribute from FRAME
4. create new COPY in PAGE of COPY
5. make new COPY not CURRENT
6. make new COPY MOST_RECENT if and only if COPY is MOST_RECENT
7. make FRAME be associated with new COPY

delete copy (COPY)
1. if COPY is only MOST_RECENT COPY of its PAGE return
2. delete COPY from PAGE
3. make FRAME of COPY FREE

set current (COPY)
1. if COPY not MOST_RECENT return
2. make COPY CURRENT

remove current (COPY)
1. remove CURRENT attribute from COPY