SECURITY AND PROTECTION OF DATA IN THE IBM SYSTEM/38

Viktors Berstis
IBM General Systems Division
Rochester, Minnesota 55901

ABSTRACT

The IBM System/38 machine architecture provides data security through addressability control and protection domains while minimizing overhead for this service. This paper describes the addressing mechanism, basic data organization and process structure of the System/38. These constructs are fundamental to the operation of the security mechanism. A description is given of security in the architecture and how it can be extended.

INTRODUCTION

Security as used in this paper refers to the control of access to data by multiple users of a computing system. With the increased concern in the data processing community for security, new computing systems are being developed which address this subject. In the IBM System/38, the security mechanism is part of the basic machine architecture as opposed to the operating system. This paper describes the System/38 machine architecture and its implementation of security.

The primary goal of the security mechanism in the System/38 is to provide control of access to data while minimizing overhead. Among the wide range of potential System/38 users are many who will have no immediate need for the full range of security functions. These users will experience little overhead in a minimum security environment. Secondarily, the security mechanism helps contain programming errors, thereby increasing the reliability and integrity of the system. It is with these goals in mind that the System/38 security aspects of the machine architecture were defined.

SECURITY PRINCIPLES

Three basic principles underlie the security architecture of the System/38. The first is the implementation of capability based addressing. The second is that objects called User Profiles determine the protection domain of a process. A user profile contains a set of access rights to system objects. The third is that processes are isolated from each other. The implementation of these principles is shown in light of the addressing structure and the way the instructions are supported.

STORAGE AND OBJECTS

The System/38 storage consists of semiconductor memory and secondary disk storage with up to 2.6 billion bytes in a maximum configuration. This storage is organized as a single virtual address space divided into 16-megabyte segments, further subdivided into 512-byte pages. The 64-bit virtual address is composed of a 40-bit segment identifier and a 24-bit offset.* A single microcode component manages the virtual storage, its allocation and paging. The remaining components of the system, including the channel, all use the same virtual addressing mechanism for referencing storage. Segment identifiers of destroyed objects are never reused. (It is assumed that all of the segment identifiers will never be consumed in the life of the system.)

Objects, which are high-level constructs such as programs, queues and indexes are stored in one or more segments. The segments have header information, seen only by the microcode, which identifies the object type contained in the segment and contains all of the attributes pertinent to that object. The header information also identifies additional segments which may be required for a given object type. The segment identifier of the base segment is used internally to uniquely identify an object.

Table 1 itemizes the various types of objects defined in the architecture. High-level machine instructions are used to operate on objects. These instructions maintain the integrity of the object's internal data structures and permit different implementations of their functions without impact to their users. Except for one type of object called a space, no provision is made to address bytes within objects. The contents of a space are directly addressable and operations at the bit level are allowed.

---

* The hardware only supports 64K-byte segments and a 48-bit address. The microcode extends the size of the virtual address to 64-bits at the cost of not allowing more than 16 million segments to exist at any given time.

CH1494-4/80/0000-0245 $00.75 © 1980 IEEE

Program variables are typically stored in a space. A space may be optionally associated with any other object type and is then referred to as an associated space. These spaces are used as a convenient place to store user defined data pertinent to that object. They are usually allocated in a separate segment since their allocated size may vary.

ADDRESSING

Addressability to objects is supported using tagged pointers. Pointers are capabilities in that they represent a unique identifier for objects and bytes in the system. Pointers, however, do not always contain access rights, as will be described later.

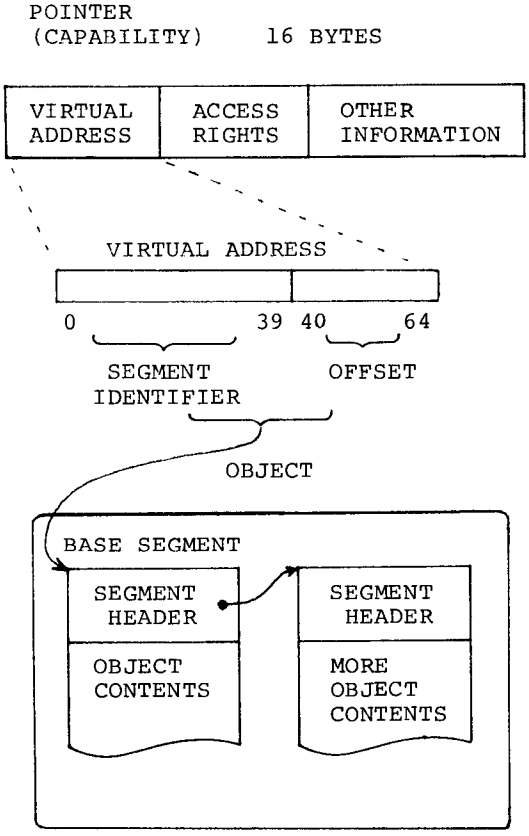


Figure 1. Addressing

Pointers are stored in spaces as 16-byte, quadword aligned entities which contain a virtual address and other information. The memory hardware supports tag bits which are used to authenticate pointers. The tag bits cannot be set on by the user so that addressability to arbitrary virtual addresses cannot be counterfeited. Any attempt to alter the contents of a pointer causes the associated tag to be removed and hence it is no longer recognized as a pointer. The microcode creates only valid pointers to addresses for which the user is authorized.

There are four types of pointers: System Pointers address objects; Space Pointers address a byte location in a space; Data Pointers are like space pointers except they also contain attributes of the data they locate; and Instruction Pointers, which act as label values for variable branches within a program.

Addressability to objects is initially obtained by using the object's name. An object called a Context contains names of other objects and their associated virtual addresses. When a pointer to an object is required, it can be obtained by using the Resolve System Pointer instruction, which searches a specified set of contexts for the desired name. A system pointer is returned providing addressability to the corresponding object, but not necessarily any access rights to it. The process must have Retrieve authority for any context it uses for obtaining system pointers to objects.

A further authority check must be passed when an instruction attempts to operate on the object itself. The checks governing access to objects are performed by a few common microcode components. The first component provides a fast way of fetching a pointer and verifying its type, validity and the existence of the addressed object. The second verifies that the process has authority to use the object, synchronizes multiple accesses to the object and performs various other checks. This second set of verifications is only used for object (as opposed to byte) oriented functions.

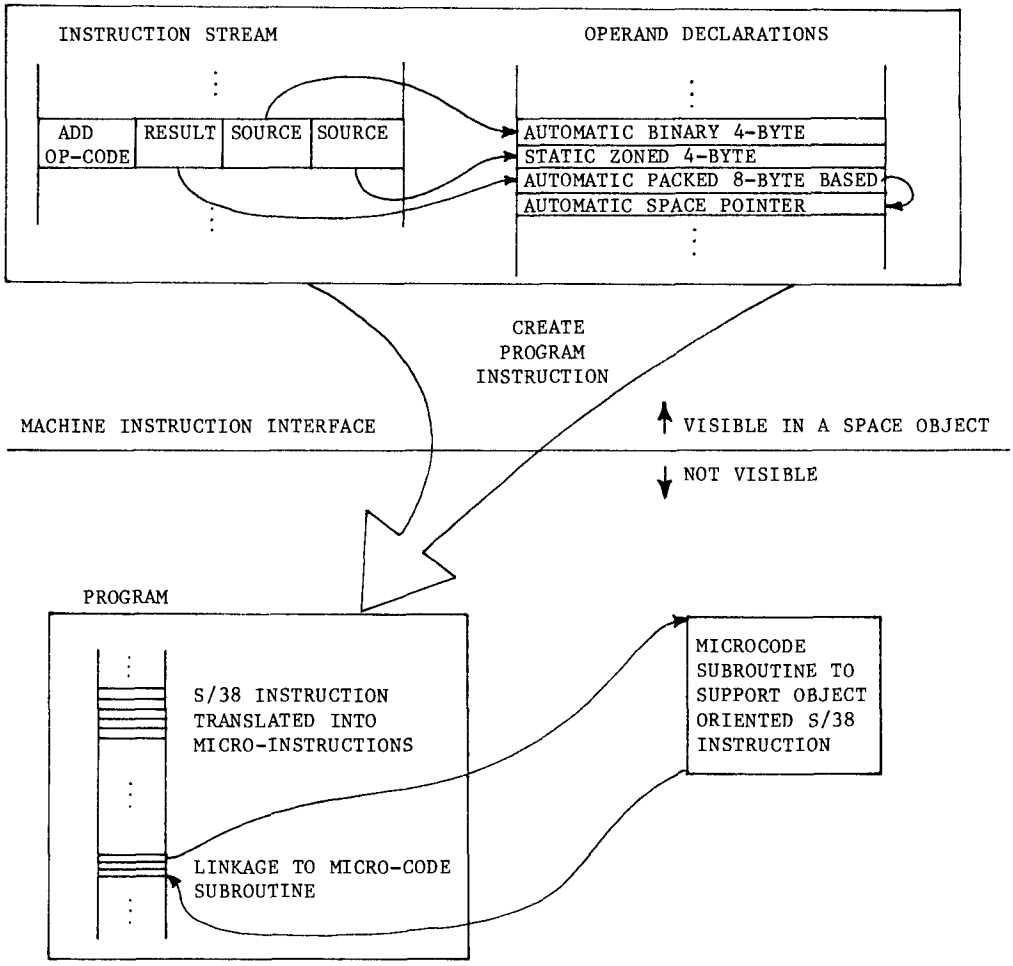
INSTRUCTIONS, PROGRAMS AND OPERANDS

The System/38 machine instructions fall into two classes. The first class consists of byte-oriented computation, and branching instructions. These are very similar to those found in other machines and include arithmetic, logical and byte manipulation instructions. The second class of instructions perform high-level functions on objects. Single machine instructions perform operations such as retrieving an entry from an index, enqueuing a message on a queue or invoking a program. The byte oriented instructions can only be used on the contents of spaces. Other objects cannot be accessed using the computation and branching instructions. Consequently, the data structures necessary to support these objects are protected from arbitrary modification.

Machine instructions are never directly executed in the System/38. Instead, they are translated into micro-instructions which either perform the operation "in-line" or invoke a microcode subroutine. The translation of a program is analogous to compiling a low level language program. The translation is done when the program is created using the Create Program instruction as illustrated in Figure 2. The input to the create program instruction is a string of byte data with the program's instructions encoded in the usual opcode-operand fashion and a coded description of all of the instruction operands (variables and constants) used in the program. The translated

Figure 2. Program Creation

INPUT TO THE CREATE PROGRAM INSTRUCTION DEFINING THE PROGRAM



program becomes an object in the system, and in this form it can be executed by issuing a call instruction. Programs cannot be modified when in the form of an object, nor can they be invoked at any point other than their proper entry point. This prevents unpredictable results if sections of a program were bypassed.

The create program instruction is one of the types of instructions which invokes a relatively large microcode subroutine. Other instructions may be translated into just a few micro-instructions. An example is the ADD instruction which can produce a large variety of micro-instruction expansions. The description of the program operands is used (during program creation) to both locate the data and generate appropriate micro-instructions depending on the data attributes.

The operands of a program are either constants stored with the program or automatic or static variables stored in a space. When a program is invoked for the first time in a process (task), the static variables are assigned to an available region of storage (in a space) and initialized to the specified values. Similarly, automatic variables are allocated and initialized (deallocated) on every call (return) of a program in a process. These variables and constants are directly addressed by the translated program. When instructions reference objects or other regions of storage, they are always indirectly addressed with pointers.

PROCESSES

A process is a machine construct that executes programs on behalf of a user. A process is started with an Initiate Process instruction that accepts a list of attributes as input to determine the components and initial state of a process. The first conponent is an object called a Process Control Space, which is used to hold all process related data which needs to be protected from user access. Data defining the current protection domain of the process is stored there, for

example. The second component is the initial program to be executed within the process. Next are two spaces specified for the allocation of program automatic and static storage and organized as stacks. As programs are invoked, their variables are allocated in these stacks. Finally, the most important component of a process, with respect to security, is the User Profile which is the basis for controlling the protection domain.

The principle of isolated processes is accomplished quite naturally with the addressing structure in the machine. No process can be affected by another without intentionally providing some sort of access to itself (e.g. addressability to its stack spaces). This leaves open the "Trojan horse" problem, but few if any systems have a completely effective solution, aside from total separation. Since it is relatively easy to share data in the System/38, little effort is required for a higher security level process to pass information to lower level security processes.

USER PROFILE AUTHORITIES

A user profile is an object used to establish the protection domain of a process. Each user of the system is identified with a user profile. When a user signs on, the operating system positively identifies the user and initiates a process with the appropriate user profile as its primary source of authority. There are four categories of authority available to a user profile, as shown in Table 2.

The first is storage resource authorization which limits the amount of storage allotted to the user for creating objects. Any objects the user creates are charged against this limit and are owned by that user profile. The ownership of an object implies certain additional rights to an object as described later.

The second category of authorization controls the use of privileged instructions. Various machine instructions are considered privileged and may be authorized to a user profile in any combination. Examples of privileged instructions are those which create user profiles, initiate processes and perform diagnostic functions.

The third category is the access rights to individual objects in the machine. There are eight object authorities which can be explicitly granted, in any combination, to specific user profiles for specific objects. For example, one user profile can be authorized to only retrieve entries in an index, while another can be authorized only to insert new entries. The object authorities represent the most important category for general data protection.

The fourth category consists of the special authorities. These represent authority to perform pervasive functions which are not necessarily associated with a particular object or type of instruction. Examples are the ability to modify machine attributes, perform service functions, and to implicitly have all of the object authorities to every object. This All Object special authority can be used for the system security officer's user profile, for example.

OBJECT AUTHORITIES

The eight object authorities may be conceptually placed in three groups. The first deals with object existence; the second with object access; the third with access to the contents of objects. The operations performed by the machine instructions are classified into the appropriate groups and the corresponding authority verification is included as part of their operation.

For example, the destroy object instructions change the existence of objects and therefore require Object Control authority for the object being destroyed. Similarly, the instruction which renames an object changes the name with which the object is accessed and therefore requires Object Management authority be available to the process. The object authorities, Retrieve, Insert, Delete and Update, are appropriate for operations on the contents of objects which contain entries of various kinds.

The Space Authority is required by the instruction which produces a space pointer for the associated space of an object (giving access to the contents of the space). Space pointers to bytes within a space can initially be obtained with only one machine instruction. An authority check is made to assure that the process executing this instruction is authorized to obtain a space pointer to the requested space. If not, an exception is signaled and the space pointer is not created. Once a space pointer is obtained for a space, the pointer may be adjusted to point to any byte within the space but not in any other object (segment), nor the header of the space. The instructions which add to, subtract from or set the space pointer address within the space, check for an overflow or underflow which would make the pointer address a byte outside the segment. In addition, there is a "low bound check" to keep the space pointer from addressing bytes in the segment header.

AUTHORITY VERIFICATION

The object authorities available to a particular user profile are stored internally as an entry in an index for each authorized object. There is one such index for each user profile. This index also contains entries for all of the objects owned by the user profile, and entries which indentify which user profiles have been granted object authorities for these owned objects. This indexing supports instructions which produce lists of user profiles authorized for a particular object and lists of objects explicitly authorized to a user profile. Figure 3 shows how these authority relationships are implemented. Object D owned by user profile A is authorized to user profiles B and C.

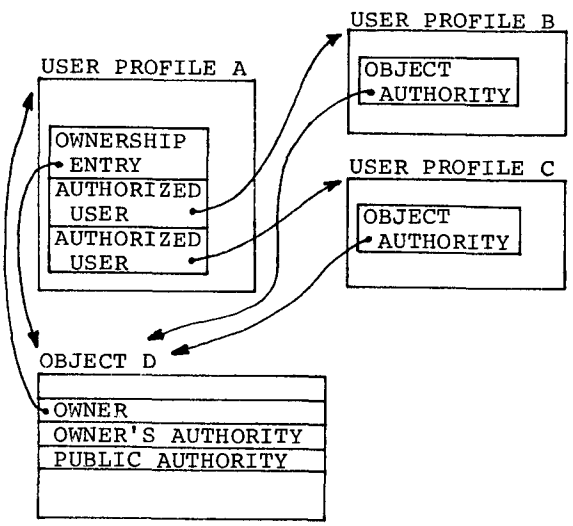


Figure 3. Object Authorization

If every reference to an object required a look-up in the user profile index, then this would imply a significant performance penalty. For this reason, additional measures have been taken both in the architecture and implementation to minimize the time required to verify authority.

Many objects, common system programs for example, are intended for use by all users. For this purpose, each object also has a set of public object authorities defined, which are stored in the header of the object. A system program may be authorized with retrieve authority to the public but not Object Control. This allows all users to invoke the program but not destroy it. The public authority check is performed early in the authority verification sequence. Users not requiring individual authorization of objects would use public authorization for minimum overhead.

Objects which are owned by a user profile also require little time for authority verification since the owner's authority is stored in the object header rather than in the user profile's internal index. When an object is created, the process user profile becomes the owner and is granted all of the object authority rights. (In this way, the user can continue using the object without further concern for authority as long as other user profiles do not require access to the object.

Objects can be created as permanent or temporary. The difference between these is that handling temporary objects requires less overhead. Temporary objects do not provide authorization controls and no attempt is made to safeguard their contents in case of system failure. In fact, if a temporary object has not been explicitly destroyed when the machine is shut down, it is destroyed at the next IPL (initial program load).

Permanent objects are always owned by a user profile. The owning user profile always posesses the implicit ability to grant or retract authority to the object. Temporary objects are never owned by a user profile and their storage is charged against a limit set for the process when it was initiated.

AUTHORIZED POINTERS

When explicit object authorization is required, the authority can be stored in the system pointer used to address the object. This makes the pointer a capability containing access rights. An option on the resolve system pointer instruction permits the user to request particular object authorities to be stored in the pointer. Of course, the process is not permitted to store authorities (into the poiner) which are not available through some user profile. The object authority verification process is the fastest for these authorized pointers because the users profile's index need only be searched on the first reference to the object. Subsequent references suffer negligible overhead.

With authorized pointers, it is not possible to retract authority to a given object and guarantee that no user can subsequently access it. This is because authorized pointers can be copied and saved indefinitely in spaces for later use. If the ability to absolutely retract authority is necessary, then the object can be destroyed making all existing pointers to the object useless. The object can never be recreated with the same virtual address since segment identifiers are never reused. There may be cases where destruction is not a desirable alternative (eg. the object is frequently in use). In this situation, repeated index searches for authority verification may be necessary and the storing of authority in pointers would be prohibited. The Authorized Pointer authority is used to control which users may or may not store authority in the pointer.

Note that space pointers need no authority checks for their use. The access rights to a space are controlled only at the time the space pointer is initially obtained.

ADOPTED USER PROFILES

Thus far, the sources of object authority to a process have been the process user profile, public authority and authorized pointers. At times, the protection domain must be altered within a process. To accomplish this, the process user profile can be replaced with another, under appropriate authority controls. Also, it is possible to add the authority of another user profile to the one already available to the process when special programs are invoked. These programs are created with an attribute which says they "adopt" their owners user profile.

The Adopted User Profile attribute permits a program to have more authority than its caller. This kind of program is useful when providing system services through a program controlled interface (a system program to initiate another

process, for example). Another attribute of the program specifies if the adopted user profile's authority is to be propagated to programs it calls. If several programs both adopt and propagate their owning user profile authorities, then the currently invoked program has available to it the sum of the authority of all of those user profiles.

Since a program can be invoked to handle an asynchronous event or an exception, the propagation of adopted authorities is stopped and not available below that point in the program invocation stack. The event or exception handling programs may, however, adopt their owning user profiles and optionally propagate as can the programs they call. Breaking of propagation at exception and event handlers prevents a program earlier in the program invocation stack from gaining direct control in the protection domain of a more privileged program.

Although there are many sources of authority, the architecture does not reveal in which order the authority sources are examined. This permits changing the machine implementation as various hardware and microcode design improvements are made. In general, the faster checks are performed first while the slower index searches are performed last.

GRANTING AUTHORITY

There are various instructions for altering the authority available to user profiles as well as those which show what authority is available. The Grant instruction is used to authorize other user profiles access to an object. To grant authority for an object to another user profile, the process must possess the authority to be granted in addition to object management authority for the object. By not authorizing Object Management to other user profiles, the owner of the object is assured that further authorization does not take place. The other categories of authority are changed with a Modify User Profile instruction. Again, the issuing process cannot grant more authority than it itself has.

EXTENSIONS

The security architecture defined for the System/38 is presently more than adequate for the projected users of the system. The programming languages offered are RPG III and the command language in the Control Programming Facility (CPF). Since there is no assembler on System/38, it is not possible to manipulate pointers and issue machine instructions that would compromise any of the data structures supported by CPF or RPG III.

The machine architecture has been designed to be extendable so that new functions can be supported as needs and experience grow. For example, more protection between programs in a process may be required. This could be accomplished by providing another class of variables which would be allocated in a protected region of storage. Other programs in the process would not have access to this data. With such protected storage, authorized pointers and spaces could be used more freely for securing data. Since the existing automatic and static storage areas are contained in single spaces within the process, any program may gain addressability to other program's variables within the process. The protected variables would offer a safe place to store sensitive data within a program invocation.

Another possible extension would be to support read-only authorization of spaces. These could be used to share commonly used data among processes in a space without the possibility of accidental or deliberate modification of that data.

SUMMARY

The System/38 achieves security primarily through its capability based addressing structure, authority verification and its inherent isolation of processes. Memory tags assure that addressability is restricted to authorized areas of storage. A flexible authorization mechanism provides various levels of control over access to data and certain machine functions. The protection domains of processes can be altered using different user profiles as the sources of authority. The architectural support of multiprogramming permits sharing data as well as isolation between users. Finally, the architecture is extendable so that appropriate new functions could be added.

REFERENCES

1 Denning, D E, and Denning, P J, "Data Security," ACM Computing Surveys, Vol 11, NO. 3, September 1979, pp 227-249.

2 Fabry, R S, "Capability-Based Addressing," Communications of the ACM, Vol 17, No. 7, July 1974, pp 403-412.

3 IBM, "IBM System/38 Functional Concepts Manual," (available with machine shipment).

4 IBM, "IBM System/38 Functional Reference Manual," (available with machine shipment).

5 Linden, T A, "Operating System Structures to Support Security and Reliable Software," Computing Surveys, Vol 8, No. 4, December 1976, pp 409-445.

6 Utley, B G, et al, "IBM System/38 Technical Developments," IBM GS80-0237, 1978, pp 1-110.

Table 1: System/38 Object Types

Object Type	Description
SPACE	Byte addressable storage
PROGRAM	Internal form of a program consisting of an instruction stream and operand descriptions
USER PROFILE	Indentification of a user and the user's authority
DATA SPACE	Homogeneous file of data entries
DATA SPACE INDEX	Logical file of entries in one or more data spaces
CURSOR	Access path to data space entries
QUEUE	Message queue for interprocess communication and synchronization
INDEX	Index containing up to 120 byte data records in ordered sequence
CONTEXT	Index for gaining pointer addressability to objects by their names
PROCESS CONTROL SPACE	Identification of a process (task) and repository for internal control information
LOGICAL UNIT DESCRIPTION	Identification and attributes for an input/output device
CONTROLLER DESCRIPTION	Identification and attributes for a device controller
NETWORK DESCRIPTION	Identification and attributes of a communications line
ACCESS GROUP	Groups objects physically together to improve paging characteristics

Table 2. Authority Categories

Category Authority	Description
RESOURCE	
Storage allotment	Limit on storage space for permanent objects
PRIVILEGED INSTRUCTIONS	
Create user profile	Initiate a user in the machine
Initiate process	Start the execution of programs in a task
Terminate machine processing	Shut down machine operation
Create logical unit description	Configure a device
Create network description	Configure a communications line
Create controller description	Configure a device controller
Modify user profile	Change the authority of a user profile
Modify resource management control	Change various tuning parameters
Diagnose	Perform certain diagnostic functions
SPECIAL AUTHORITIES	
All object	Implicit object authority to all objects
Dump	Make offline backup copy of an object
Suspend	Truncate object to minimum size without destroying, cannot use until loaded
Load	Replace an existing object of create a new object using a backup dumped copy
Process control	Allows the control over the execution of other processes
Service	To use internal service functions
Modify machine attributes	Change system clock time, etc.
OBJECT AUTHORITIES - Authorized on a per object basis	
Existence:	
Object control	Control object existence: destroy, suspend
Access:	
Object management	Control object access: rename, grant authority, change addressability ...
Authorized pointer	Permits storing of authority in system pointer to object
Contents:	
Space	Required to obtain space pointer to space
Retrieve	Retrieve entries from an object: retrieve entry from data space ...
Insert	Insert new entry in object: add entry to data space ...
Delete	Remove an entry from object: delete entry from data space ...
Update	Modify an existing entry in object: update entry in data space ...