COMPUTER SECURITY IN AN EDUCATIONAL ENVIRONMENT

By

DONALD O'NEAL HEWITT

S.B., Massachusetts Institute of Technology
(1972)

SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF

SCIENCE

at the

MASSACHUSETTS INSTITUTE OF

TECHNOLOGY

June, 1973

Signature of Author .....
Alfred P. Sloan School of Management, May 11, 1973

Certified by .....
Thesis Supervisor

Accepted by .....
Chairman, Departmental Committee on Graduate Students

COMPUTER SECURITY IN AN EDUCATIONAL ENVIRONMENT

By

Donald O'Neal Hewitt

Submitted to the Alfred P. Sloan School of Management on
May 11, 1973, in partial fulfillment of the
requirements for the degree of Master of Science

ABSTRACT

The thesis begins with a qualitative description of the educational computing environment. The environment is described as consisting of three areas -- Administration, Research, and Classroom Support. After discussing these three areas, their security requirements are examined in terms of a simple framework. An attempt is made to separate physical and operating system security requirements.

The various security requirements are compared, and it is asserted that the solution of the problems of the Classroom Support Environment effectively alleviates the problems of the other areas.

The MIT Class Monitor System, in conjunction with the IBM Resource Security System (RSS), is used as an example of a trial solution to these security requirements. In conclusion, some problems of the adaptability of current operating systems to the Classroom Support environment are discussed.

Thesis Advisor: David N. Ness
Title: Associate Professor of Management

ACKNOWLEDGEMENTS

I would first like to thank David Ness and Stuart Madnick for their immense patience and expert quidance in my work on computer systems. It is edifying to know that men so swamped with their own work can care so much about someone else's.

My thanks also to George Dixon and Bob Daley, who were instrumental in the creation of the Class Monitor. Robert Scott and Joseph Patten provided valuable insight from their perspective as heads of their departments at MIT. Finally, I am grateful for the assistance of Kathy Bradley, my typist, Sandy Viglione, my secretary, and all my personal friends who put up with me during the time this thesis was produced.

This thesis is dedicated to Bob Rebello, who, at a time when I did not know a System 360 from a garbage compacter (a distinction with which I still have difficulty...), trusted me enough to give me my first job with computers -- and subsequently the opportunity to obtain my Bachelor's and Master's Degrees. For your trust, friendship, and good advice, I thank you.

TABLE OF CONTENTS

TABLE OF FIGURES

CHAPTER 1 - THE EDUCATIONAL COMPUTING ENVIRONMENT

COMPUTERS AND EDUCATION

The explosive growth of the computer industry since the completion of the ENIAC Computer in 1946 need not be described here. Virtually every professional and institutional activity has been affected in some way by this powerful and flexible tool. The business of education is no exception. As more and more of the information processing activities of modern educational institutions move to computer systems, the problem of security of programs and data becomes increasingly important. This makes the educational environment a good one in which to observe security problems and their implications.

Education comes, of course, at many different levels -- primary, secondary, college, and graduate. If we divide the basic functions of educational computing into administration, research, and classroom support (this chapter discusses these areas in some detail), we can compare the different levels of education in Figure 1. As can be seen from this simple comparison, when the level of student ability increases more types of processing become feasible, including individual research and special programs for classroom support. Because the applications found at the university level seem to include those of the lower levels, it will be assumed here that higher education is a reasonably representative segment of the educational environment.

For those who wish an extremely detailed and authoritative discussion of the educational environment, there is a publication of the Rand

Corporation[1] which gives an excellent summary. Most of this first chapter is based upon that document, referred to here as the Rand Report. The goal of this chapter is the presentation of the small portion of the Rand Report which is necessary to provide the proper background for our discussion.

LEVEL	PROCESSING: ADMINISTRATION	RESEARCH	CLASSROOM SUPPORT
PRIMARY	YES	NO	NO
SECONDARY	YES	VERY LITTLE	NO
UNDER-GRADUATE	YES	YES	YES
GRADUATE	YES	YES	YES

FIGURE 1: COMPARISON OF PROCESSING AT DIFFERENT EDUCATIONAL LEVELS

ADMINISTRATION

Within the educational environment, administrative data processing can be subdivided into three categories for discussion. It is important to realize that these categories represent only one of several breakdowns of this area. The Rand Study, for example, uses a slightly different one. The analysis in this paper is done from an information security point of view, and thus has a functionally different set of criteria for subdivision.

The first category is Corporate Processing, so called because it is virtually identical to the central administrative data processing of large corporations. The financial and accounting applications of the institution are found here, as well as the personnel processing common to all employers, such as employee payroll and benefits. In addition, programs for resource allocation are often included, among these being programs for scheduling classrooms and physical plant operations (analagous to production scheduling in the business world). In many ways this is the most "standard" of the various campus applications.

The second category is Student Processing. Here we find the set of data processing functions which are unique to students. In something of a chronological order, we can name Admissions, Registrar, Bursar, and Alumni Offices as groups which engage in Student Processing. Although it may seem somewhat arbitrary to separate Admissions from, say, the processing of applicants for jobs, it is in fact done almost without exception in universities. Even payrolls are kept separate, with

students and staff in separate runs. Further, from a security standpoint, the school often has more confidential information on its students than on its employees. Therefore, we keep Corporate and Student Processing separate.

The final category is Departmental Processing. This subset of administrative data processing is a direct outgrowth of the decentralized departmental organization of most universities, and it includes many of the same functions as Corporate and Student Processing. Graduate admissions, for example, are commonly handled both at the departmental and central level. Resource allocation is also handled as a cooperative effort between the master schedulers in a central office, and the departments who staff the various courses. Payroll, on the other hand, is an example of a function which is generally not duplicated. However, the department for which a faculty members teaches is likely to have to worry about which account or research fund should be tapped for salary and expenses, so that there is still some interaction. A great deal of aggregate planning is done at the department level, in the design of curricula and research programs. In summary, Departmental Processing is a decentralized subset of administration which has a more microcosmic view, and which requires interfacing with the central programs and data bases in order to be effective.

Having examined the components of administrative data processing, let us review some general characteristics of this area. This kind of processing operates mostly on fixed short-term deadlines - registration day, budget submission day, not to mention payday. This is often in

sharp contrast to the demands of Research or Classroom Support Processing, which, as we shall see, are quite flexible. Such a need for accurate timing dictates a high degree of system continuity and reliability. The manager of such an operation is likely to be quite conservative, a fact which underscores the similarity of this type of processing to its industrial counterpart.

Another facet of this area which resembles commercial enterprise is the accountability of managers for funds used. Because of the relatively standard nature of the tasks being performed, techniques have been established for their valuation and management. Further, since the users of administrative systems are employees rather than students, the resources of these facilities can be more closely (and more formally) managed. An interesting sidelight here is the absence of the "interesting area of research" syndrome, which often allows feasibility studies to be bypassed in research when considering the acquisition of new hardware or software. Thus innovation is more difficult in the administrative area.

A final note, and one which is certainly important to this analysis, is the need for security. It is clearly necessary to protest confidential data on both students and staff from unauthorized disclosure. Further, as in most corporate applications, it is essential to protect programs and data bases from accidental or deliberate modification or deletion, as well as to protect historical data files on tape or other media. A wide spectrum of physical and programming measures are needed to insure adequate protection.

The combination of differences in programming needs (e.g. FORTRAN versus COBOL), the need for conservative, reliable, short-term service, and the need for system security has led almost every university to physically separate their administrative data processing facilities from all others. This has automatically solved the majority of their security problems so far. There is, however, a terrific price for this security. First, the physical separation has necessitated duplication of hardware and programming staffs. In an era of budget throttling and cost cutting, this tends to be very unpopular. Second, and perhaps more subtle, is the fact that their deliberate isolation from the facilities utilized by Research and Classroom Support systems has meant that the administrators have been isolated from the technological progress being made in these areas. Although one would not expect this type of processing to keep up with the forefront of research (nor would this be desired from the standpoint of continuity and reliability, as mentioned earlier) the gap between the two technologies has been allowed to widen to such an extent that even if all security problems were solved tomorrow, it would be several years before most administrative facilities would be able to consider a merger of any sort with existing research facilities. The M.I.T. Office of Administrative Information Systems is a good example of this problem.[2]

One interesting exception to this general notion is Departmental Processing. Because this processing is at the department level where a large supply of cheap, high-quality labor exists (thesis students), more and more work is being done on these applications. Oddly enough,

because the students who work on such projects do not have access to the administrative facilities, the work has been forced onto the newer research facilities, with the expected benefit of the newer technology. Thus we see information systems for management and aggregate planning being developed, for example, in the Sloan School of Management at M.I.T.[3] which far outstrip the capacity of the central administrative facility in the sophistication of information handling techniques. However, these applications are in desperate need of solutions to the security problems which the central facility solved by physical separation. Thus we find the departments pioneering applications for the central facility by pushing into research facilities with innovative and complex planning systems which will force the security problems to be faced, probably well in advance of the needs of the central facility.

It should be note, in conclusion, that there are two possible ways for the administrators to proceed. First, they may find it advantageous to combine facilities with the administrative organizations of other universities. An example of this type of venture is WICHE (the Western Interstate Commission on Higher Education), which combines many universities in the design of standard administrative programs.[4] This would provide a stable system, and also promote innovation through common development of systems. However, the problem of security still exists, since many schools would be sharing common hardware if this cooperation were carried to its logical conclusion. The other alternative seems to be merger into a central campus computing facility either as a complete move, or as a remote "satellite" facility. This brings

more risks in terms of system instability, but has the advantage of allowing a much smoother interface among the various kinds of central and departmental programs. In any case, it seems clear that over the next five years, security problems will come to the point where they can no longer be avoided.

RESEARCH

Computer Science students (and faculty) tend sometimes to describe the campus as composed of two types of people -- those who understand computers, and those who wish they did. Although this is a distinction which is sometimes difficult to make, it does have some relevance in describing research on today's campus. A more reasonable way of making the distinction for our purposes is to describe Research Processing as composed of research on computers, and research with computers. The former category is composed mostly of computer scientists, electrical engineers, and a few computer-oriented information systems specialists. The latter includes more of the campus every day, and even now, on the MIT campus, there is not a single department (including the Music Department) where the impact of the computer has not been felt heavily.

There are, of course, other distinctions which could be made here, for example, thesis versus non-thesis research, and sponsored versus non-sponsored. While we may make these distinctions here as examples of different accounting methods, our primary emphasis will be upon the differences between research on and with computers. This emphasis allows us to focus on the security aspects of research processing.

By nature, Research Processing is highly decentralized. This is primarily due to thesis projects, which are generally pursued individually under the supervision of an advisor or committee. In terms of the number of persons participating, this constitutes more than half of the research effort of most universities. In addition to extreme decentralization, Research Processing can be characterized by a lack

of definite deadlines. This is quite intuitive -- one can hardly set deadlines on pure research and development. The kinds of time constraints generally encountered are in the form of arbitrary limits such as project expiration dates and thesis deadlines, which are generally quite flexible. A related facet of managing research efforts is the idea of funds allocation. When examining administrative processing, we observed the formal structure for financial accountability. A different problem is encountered with the thesis user. Generally, the thesis user is not spending personal funds on computing. The department will often have a set procedure for computer resource allocation. The problem occurrs when the user consumes the full amount, and is not finished. It seems absurd to think that the student must quit. Somehow, more funding must be obtained. Control of not only the allocation of funds, but the use of funds thus becomes a problem.

A related problem is the expertise of the individual user. The researcher who is a computer specialist is likely to have computer expertise which permits efficient use of the computer resources, while the person who merely wants the computer as a tool for other research may be totally unfamiliar with the proper methodology, thus wasting vast amounts of money. Further, it is possible that the non-computer researcher may be unwilling to invest time in learning efficient techniques. From this perspective, we must realize that these users have their own priorities, and it is quite realistic to assume that the computer should not be allowed to detract appreciably from their primary goals of research in their selected field.

The last aspect of Research Processing to be discussed here is a mixed blessing. Certainly, the campus has been the scene of the most significant advances in computer technology -- the computer itself, core memory, software systems such as the MIT Compatible Time Sharing System (CTSS) and others. In addition, the research on computers within a given campus community often results in "tuning" improvements within existing equipment, such as improved device management, scheduling algorithms, etc. Sometimes, entire new systems such as the MULTICS system at MIT, become available to the community as a direct outgrowth of campus research. However, this technological fallout has two bad effects. First, educational systems tend even more than others to become "tailored" with a series of modifications which render them incompatible with all other systems. Second, when new resources are introduced, there ensues a great deal of contention about system stability and documentation for general use. The classic example of this problem is the development of the MULTICS System at MIT from 1968-1971. During this time, the system underwent constant "tuning" changes, as well as several major revisions. The stability of the system was very poor, and thus many non-computer users who wanted the machine for service were in direct contention with the systems development people, who wanted constant improvement.

For purposes of this discussion, the Research Processing area can be regarded as an extremely decentralized group of users, whose expertise ranges from superlative to negligible. It is characterized by a lack of structure both from time and budget perspectives. Finally,

we find the research on computer sometimes requires experimentation at a machine level which can seriously conflict with the standard service needs of the rest of the research community.

CLASSROOM SUPPORT

The most recent of the three areas is the emerging use of the computer in support of classroom activity. Although this type of processing won early attention in the form of the much heralded "interactive teaching program", this particular use proved disappointing. Early enthusiasm toward the replacement of the more mechanical aspects of teaching with time-sharing systems waned in the face of severe problems of human interface design and language processing problems both at the syntactic and semantic levels.

In recent years, the computer has begun an important and many faceted role in the classroom environment. It comes to the classroom both as subject and tool. We shall divide Classroom Support Processing along these lines, into support for computer courses and support for non-computer courses. The reason for this breakdown is a fundamental difference in the teaching objectives of these two areas, which is reflected in the type of activity produced.

There exist within the curricula of most large universities a set of courses designed to teach computer programming and other aspects of systems design and utilization. Whether the computer is being taught in its own right or in the context of some application such as mechanical engineering, the orientation of such courses is toward the exploration of the capabilities and limitations of the computer system. This focus upon the computer results in a great deal of student use, and, as we shall see, in a serious security problem from imaginative and mischevious student programmers.

In contrast to the computer courses, a rapidly-growing segment of the computer resources on campus are being consumed by students who have no interest whatever in computers or programming. These students are using pre-packaged programs which perform simulations, linear programming algorithms, and other computations which aid the students in their work. A good example of this type of program is the area of financial management, where a student might have access to small utility programs for present value analysis and discounting, as well as large packages for linear programming and modeling. The important distinction here is that the non-computer student is neither trained in nor (in most cases) interested in the computer. Thus the kind of in-depth exploration of the computer system characteristic of computer courses is not found here. However, from the security point of view, we find a different problem -- the control of large numbers of inexperienced and sometimes indifferent users. As we shall see, there is an implied problem of usage control by the instructor, who must see that the class budget is efficiently and equitably distributed.

In summary, we have divided the educational computing environment into three areas: Administration, Research, and Classroom Support. A schematic diagram (Figure 2) shows the overall breakdown, along with some rough figures on each area as a percent of total activity. One other dimension along which computation will be viewed here is that of batch versus interactive. All of the categories in our schematic diagram can be regarded as existing in a continuum which has batch monoprogramming at one end, followed by multiprogramming, limited inquiry

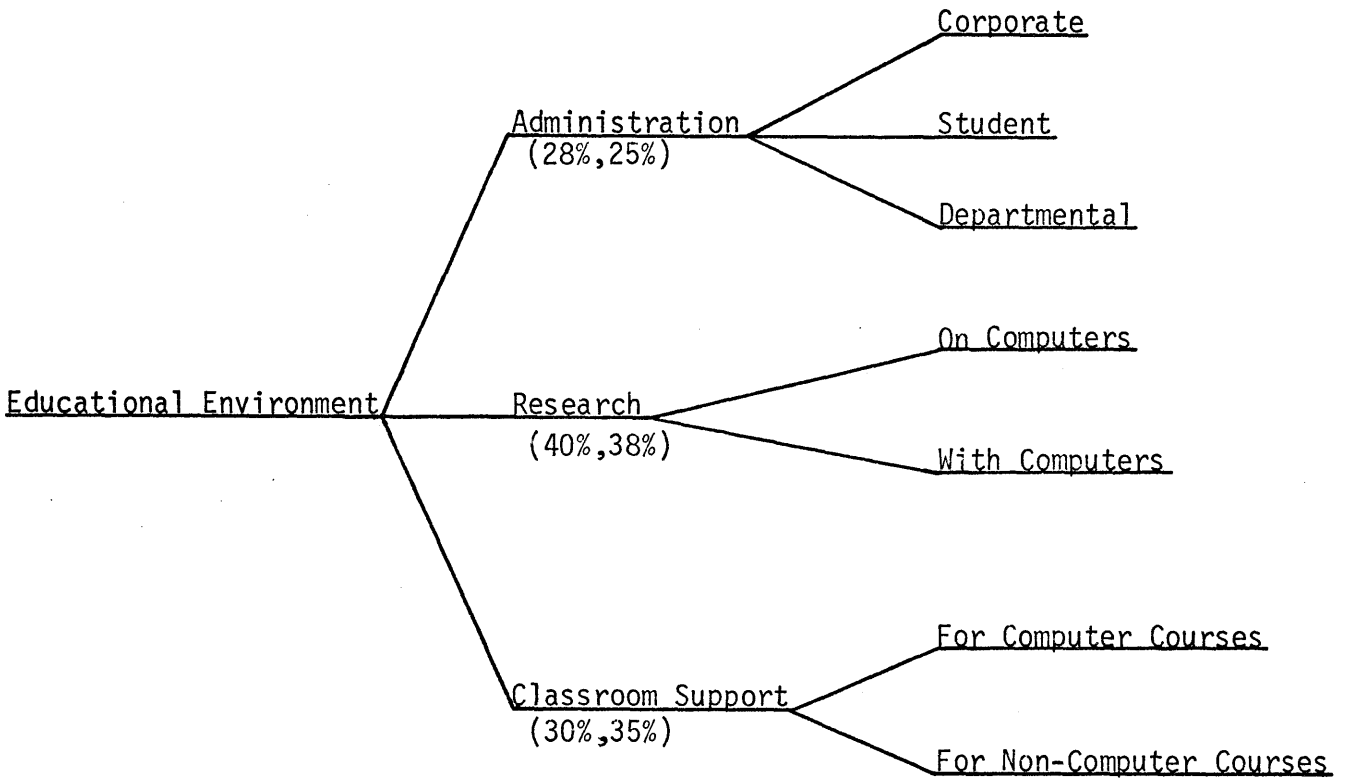


FIGURE 2: A SCHEMATIC OF THE EDUCATIONAL ENVIRONMENT

NOTE: An estimate of the three areas as a percentage of total university data processing expenditure is included. (A,B) indicates the percentage in 1967[5], and the author's estimate of the percentage in 1973, reflecting a relative increase in Classroom Support Processing. 2% is allowed for other uses.

systems, and finally interactive systems at the opposite end. The reason for this view is the difference in security requirements, for example, the problem of collection and distribution of decks in a batch environment versus terminal access control in timesharing systems.

The next chapter will discuss a framework for describing security requirements of our three areas in terms of this framework.

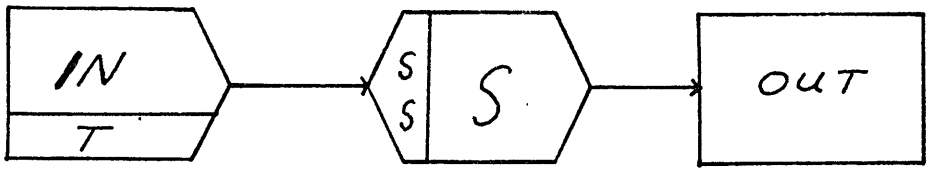
CHAPTER 2 - COMPUTER SYSTEM SECURITY

SECURITY -- THREATS AND PROMISES

The notion of security, whether in computer systems or in some other context, is one of those topics which eludes positive definition. People tend to describe security negatively, for example, security is not having your house broken into, or not having your medical records printed in the Boston Globe. It is not required here (thank goodness) that we arrive at a definition which will be accepted with great finality by all readers. However, it is essential that some working definition be developed which will facilitate the comparison of the security requirements of the different areas described in Chapter 1, as well as permit a general discussion of computer system security.

Let S denote an "ideal" computer system, one which provides uninterruptable and fully controlled service, and in which all data and programs are available only to authorized users, and only in a particular mode. Let OUT denote all of the effects which the system has on its environment, including work performed, output generated, and work generated for its staff (such as the modification of erroneous programs). Now let IN be the total input of the environment to the computer, including decks submitted, temperature of the room, etc. We now define T, (threats) a subset of IN which will cause "undesirable abberations" in the operations of S, and result in unacceptable results in OUT. Finally, we define SS as the security system, that part of S

designed specifically to screen T from IN, insuring the desired operation of S, and thus the desired OUT.



While the notion of an "ideal" computer system is admittedly a vague one, it does serve us well enough to establish the nature and function of computer security. Whatever we define as our desired system, computer system security is the ability to prevent it from being changed. One distinction which should be made here is the difference between programming errors and system errors. It is perfectly consistent with the notion of an "ideal" system to expect it to generate errors in the output of jobs whose input is incorrect. To execute exactly the program submitted is all we may ask (for the present) of any system. The system deviates from its desired performance when, for example, the accidental or intentional errors of some task are permitted to interfere with the execution of supposedly independent tasks. Thus we see security as the act of maintaining a set of system-wide relationships among programs, data, processes, users, and other system entities which define a computer system. A secure computer system, then is one which promises that its security mechanism SS is capable of dealing with every component of T without failure, insuring consistent performance of the system S.

THREATS - THEIR ORIGIN AND TARGETS

Having defined security in terms of coping with threats to the system, we now turn to the problem of classifying those threats. The first qualifier would logically seem to be the source, or origin of the threat. There are several different dimensions along which the source of a threat can be viewed. In a company, for example, threats could be classified as originating within the firm, or outside. For our purposes, however, we seek a dimension independent of the purpose of the machine, and one which is relevant to the organization of the computer system itself. Therefore, we will regard threats as originating in one of three modes:

1) EXTERNAL MODE
2) SUPERVISOR MODE
3) USER MODE

The first mode, EXTERNAL, reflects all threats which do not involve the execution of an instruction under control of the operating system. This includes most of the kinds of security problems referred to in the literature as "physical problems," such as fire, vandalism, theft of tapes, etc. It also includes the type of programs known as "background utilities" where the accidental mounting of a wrong disk pack can result in loss of good data through accidental initialization. In this case the threat is not under the control of the computer system, but is in direct control of an operator, who mounts a pack and pushes a button. Power surges are another common type of problem which

originates outside the system frame[illegible] se[illegible]al execution of instructions.

The two remaining modes, SUPERV[illegible]SER MODE, represent a distinction made in the design of [illegible]uter systems, such as the IBM 360/370 Series ("Supervis[illegible]te - [illegible]lem State"), and the GE 645 ("Master Mode - Slave Mod[illegible] Sinc[illegible] experience of the author is mainly on IBM equipment, m[illegible] f th[illegible]ples herein will be drawn from that environment. (See [illegible]]) T[illegible]st general description of the difference between Supervisor [illegible] and [illegible]Mode is that the entire instruction set of the machine [illegible]execu[illegible] from Supervisor Mode, while User Mode permits only a [illegible]rictc[illegible] set (in the 360/370 Series, all i/o instructions and sever[illegible] contr[illegible]structions require Supervisor State).[6]

It should be noted here that not [illegible]l prog[illegible] in the operating system run in Supervisor Mode. This [illegible] is g[illegible]lly reserved for important control routines, such as th[illegible]/O co[illegible]ler. If we think of Supervisor Mode as a mode of operatic[illegible]stric[illegible]o special system routines and User Mode as the mode of [illegible] prog[illegible]which execute under the control of these routines, then fo[illegible]ny no[illegible]ernal threat, the threat may be thought of as originati[illegible] <u>thin</u>[illegible]rvisor Mode or <u>outside</u> Supervisor Mode (User Mode). Assumin[illegible]at th[illegible]lity to switch from User to Supervisor Mode is closely co[illegible]lled [illegible]st machines (which it is), we say that there is a differ[illegible] in t[illegible]ture of the two threats, since one is a threat of int[illegible]l dis[illegible]on, while the other is a threat of system penetration.

One example of a Supervisor Mode threat is the IBM Attached Support Processor (ASP). In the current version of ASP being used at MIT in conjunction with the IBM Resource Security System (RSS), ASP runs in Supervisor State, with all protection disabled. In addition to being one of the less dependable parts of the operation system, ASP is generally modified by the local programming support staff to a large extent. Any bugs in ASP have an excellent chance of clobbering the operating system, since protection is disabled. Thus we see that there can be threats from within the Supervisor Mode. Threats may also originate, of course, from the programs which run in User Mode. Common threats of this type include the attempt to switch to Supervisor Mode without authorization, and the issuing of invalid requests to system service routines. (One of the classic examples of this in the System 360 is requesting the system clock to deposit the current time into the middle of the operating system programs, which it does without checking.)

In summary, we have categorized threats according to their origins, into those external to the computer itself, those internal to the computer and its privileged mode of operation, and those internal to the computer in normal operating mode. Having established where the threats are coming from, let us turn our attention to their targets.

For our purposes, it is convenient to divide threats into two major categories:

1) Threats against Operating System and Subsystem Integrity

2) Threats against dataset access control.

We shall not attempt the impossible task of enumerating all of the possible threats against a computer system. We will instead mention some of the salient types of threats, and note their places in our framework. Hopefully, this will enable the reader to insert particular cases which have been omitted here. The breakdown of our two major categories is outlined in Figure 3. As can be seen, Operating System and Subsystem Integrity includes a wide range of system functions, including accounting and control functions, as well as system stability. One important feature of this category is its recursive nature. All of the kinds of features inherent in system-wide operation, such as accounting, validation, and continuity also occur in subsystems, to an extent which depends upon the sophistication and design objectives of the subsystem. An example of system and subsystem integrity problems is the design of an interactive program for use under CP/CMS (Control Program / Cambridge Monitor System, a set of software used on the IBM 360 Model 67). The environment is shown in Figure 4. As can be seen, there are three levels of system, each with a separate set of control and accounting problems: first, the CP System, which provides a virtual machine environment which must be protected and isolated, second, the CMS System, which provides the command language and file system and must manage requests, and finally, the interactive program, which might have its own accounting and complete user environment to support and protect.

OPERATING SYSTEM AND SUBSYSTEM INTEGRITY

1) Accounting Mechanisms
2) User Validation
3) Priority and Process Scheduling
4) Integrity of Actual Code
5) Memory Access Control
6) Continuity of Operation

DATASET ACCESS CONTROL

1) Read
2) Write
3) Execute
4) Append
5) Delete
6) Restrict Access to specific programs
7) Control of access to offline files (tapes, cards, etc.)

FIGURE 3: TARGET CATEGORIES

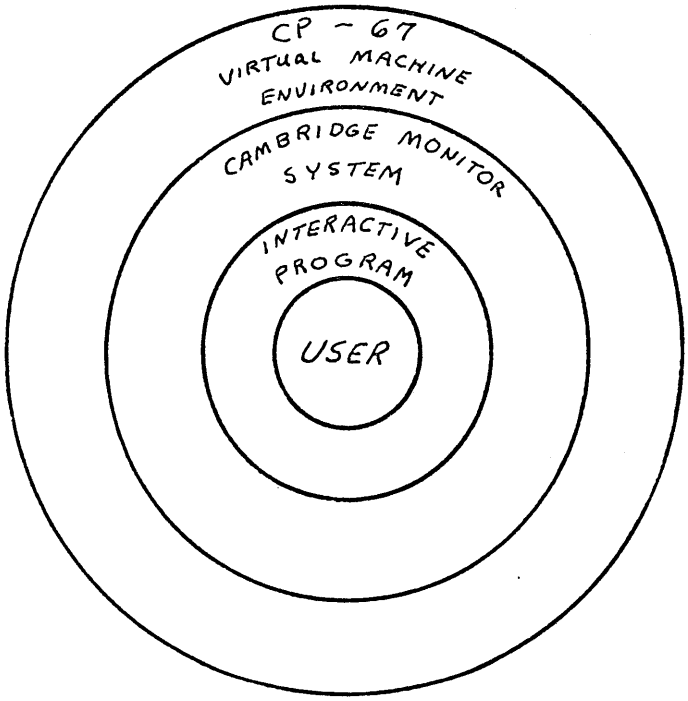


FIGURE 4: THREE-TIERED USER ENVIRONMENT

The other category of target is Dataset Access Control, where we define datasets in the most general sense as aggregations of data (or instructions), which may be either on-line, or off-line in the form of cards, tapes or printout. Here we find the conventional read, write, and other access categories for online datasets, as well as the full range of physical access controls necessary for offline files. All control of programs and data within the computer system (with the exception of those programs and data specifically designated as protected parts of the operating system) are thus placed in this category.

A schematic diagram showing the relationship between the origins and targets of several sample threats appears in Figures 5 and 6. Having outlined our security framework, we will now turn to a brief comparison of the security requirements of the educational environment.

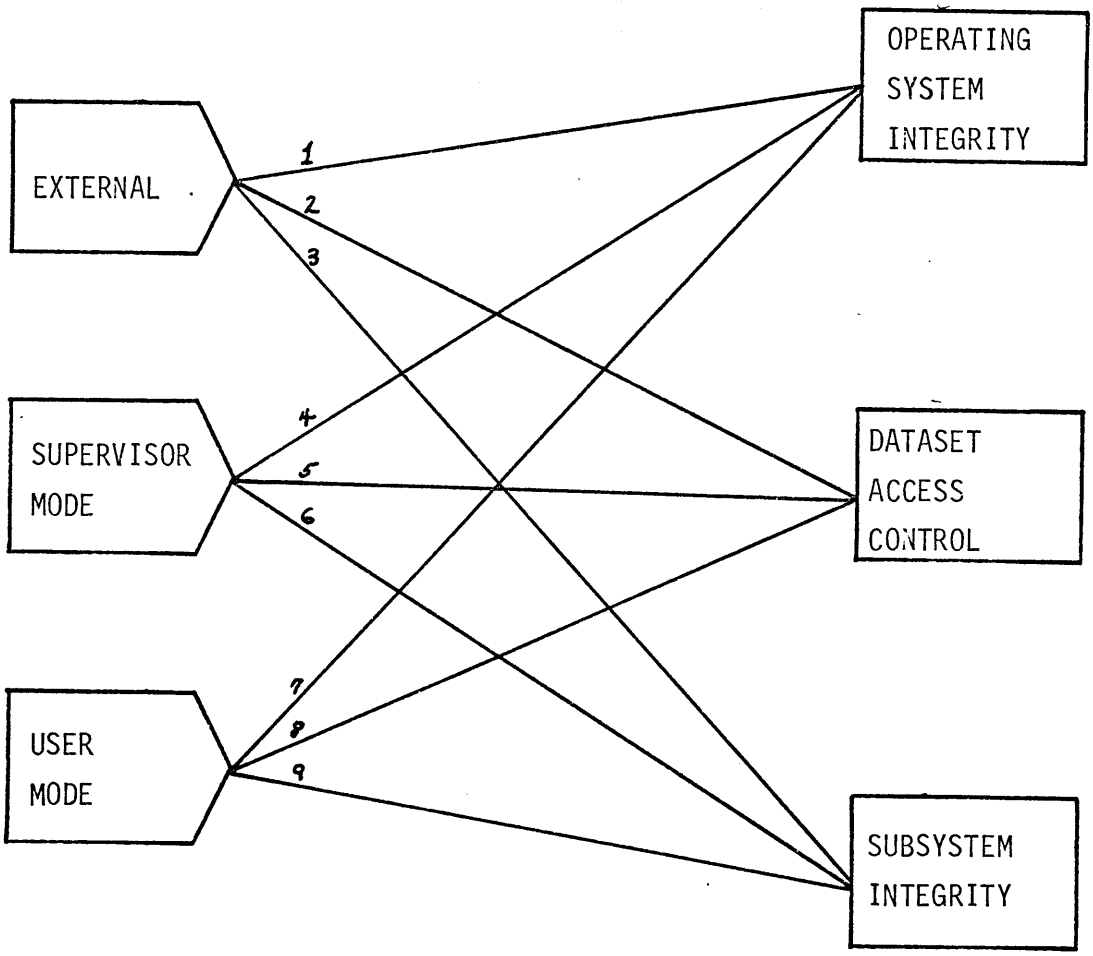


FIGURE 5: SCHEMATIC OF EXAMPLE THREATS
(SYSTEM AND SUBSYSTEM INTEGRITY SHOWN SEPARATELY)
(SEE FIGURE 6 FOR KEY)

EXAMPLES OF SECURITY THREATS

1 A power surge.

2 A tape is stolen from the library.

3 A user shuts off a dedicated terminal in a retrieval system.

4 A bug in the operating system causes a core lockup.

5 When no more space is available in the OS-360 job queue, new jobs are entered on top of jobs already in the queue, thus destroying spooled input before they can be processed.

6 The teleprocessing access method interprets a line error as an "attention" interrupt, and discontinues the program in progress.

7 User gives the time of day command a bad address, causing it to overwrite part of the OS.

8 In OS-360, users have almost unlimited access to delete datasets, even those belonging to others.

9 A user enters a character in a numeric input line and causes the program to end.

FIGURE 6: EXAMPLE THREATS

SECURITY IN EDUCATIONAL COMPUTING

Given our simple framework for classifying threats, the most convenient way of surveying the security requirements of the educational computing environment seems to be the use of the simple charts shown in Figures 7, 8, and 9. These charts outline the most important aspects of the three areas defined in Chapter 1. They will be supplemented by brief individual discussions here, followed by some general observations on the environment as a whole.

The Administrative area, as expected, has minimized many of the threat categories by its use of standard hardware and software, and its restriction of users (almost all are members of the programming staff). However, two areas are important here. First, External Dataset Access threats are extremely important, due to the large number of confidential and historical datasets, and the large amount of physical handling of these files. Also, threats from User Mode to Dataset Access come from the possibility of programming errors within the staff, as well as large scale possibilities if facilities were shared with other areas.

As we move into the Research area, we find the security requirements increasing, for two major reasons: first, the use of non-standard hardware and software vastly increases the potential threat from External and Supervisor Mode respectively, and second, the great increase in the number of users, and the corresponding decrease in control, makes User Mode Security much more important. While it is true that for most

	SYSTEM & SUBSYSTEM INTEGRITY	DATASET ACCESS
EXTERNAL	Relatively standard set of operator and environmental problems.	Relatively important standard set of problems concerning storage of confidential data and program files on cards and tape, as well as operator error.
SUPERVISOR	Minimal problems here due to use of very standard software designed to minimize operating system problems.	Minimal problems due to extremely standard and reliable software.
USER	Restricted user community and standard software minimize these problems.	Considerable problem of accidental modification since most users have full system access. Restricted user community mitigates these problems, but if facilities are shared, outside users are a problem.

FIGURE 7: ADMINISTRATIVE SECURITY PROFILE

	SYSTEM & SUBSYSTEM INTEGRITY	DATASET ACCESS
EXTERNAL	Standard set of operator and environmental problems, plus research problems if experimental equipment is interfaced with the system. Also problems of remote terminal control.	For classified research, input/output handling problems will occur. Also, standard risk of operator error.
SUPERVISOR	Experimental operating systems (and modifications to existing operating system softwate) may cause the system to be less reliable, increasing the problems in Supervisor Mode.	Operating system modifications increase the risk of dataset access problems while in supervisor mode.
USER	Most users are not interested in challenging the system. Experimental operating systems may allow accidental penetration. Computer science users may need to experiment at this level, and may cause problems if not isolated.	In an environment like OS, considerable problems with many inexperienced users having access to one another's files. Also, there is a problem of protecting users from their own errors, which is not possible under OS-360.

FIGURE 8: RESEARCH SECURITY PROFILE

	SYSTEM & SUBSYSTEM INTEGRITY	DATASET ACCESS
EXTERNAL	Full set of standard problems, plus problems of experimental equipment if facilities are shared with research users. Also, control problems with remote system and subsystem terminals.	Large scale problems in the collection and distribution of student decks and printouts, especially when these contain solutions to problem sets or examinations.
SUPERVISOR	Same as research area.	Same as research area.
USER	Extremely large scale problem of student exploration and challenge to system & subsystem control in almost every area from accounting, to validation, and continuity of service. This is particularly true of those students who are taking computer-oriented courses.	Equally large scale problem of students both deliberately and accidentally damaging or copying datasets to which they should not have access, or misusing datasets to which they should have limited access.

FIGURE 9: CLASSROOM SUPPORT SECURITY PROFILE

researchers, their preoccupation with their research makes tampering with the system unlikely, it is also true that for those who are researching computers at the most basic level, some simulation or virtual machine support is necessary if they are to use common facilities. This implies a great deal of subsystem support. Overall, we find our security problems expanding.

The last area, Classroom Support, carries the research problems one step further -- even greater numbers of students, some of whom are "hackers", and all of the problems inherited from attempting to gain from Research technology in the form of hardware or software modifications. We find here the full range of educational security problems. External, Supervisor, and User Modes seem very well filled out with standard and experimental equipment, and cooperative and malicious users.

There are clearly discrepancies in importance between compromising the solutions to a problem set and the medical records of students. If, however, one sets aside the intrinsic value of the information, and observes the security profile, it can be asserted that all of the security problems of the educational environment show up within the Classroom Support enviornment. It follows, then, that a computer system which satisfactorally solved those problems for the Classroom environment would be a satisfactory system for general use.

The remainder of this thesis deals with a specific aspect of the Classroom Support environment, namely, the creation of a monitor subsystem to allow controlled timesharing in a secure student environment.

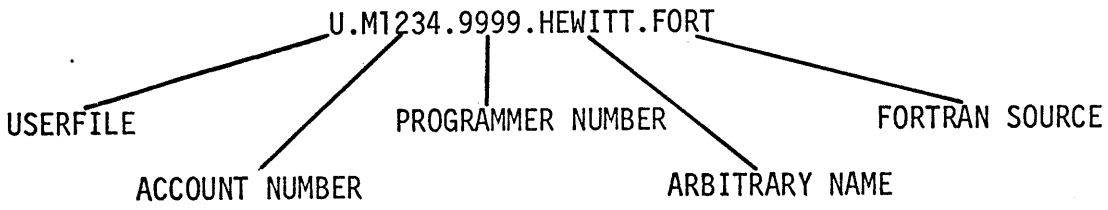
External and Supervisor Mode threats will be left for other research, and we will concentrate here on the implementation of a subsystem which must deal directly with User Mode threats in a timesharing environment.

CHAPTER 3 - THE CLASS MONITOR SYSTEM

INTRODUCTION

This chapter discusses one attempt to satisfy the need for controlled timesharing for Classroom Support Processing. The Class Monitor System for OS-360/TSO (Time Sharing Option) was implemented during the Summer of 1972 at MIT. In order to explain its design features and implementation strategy, it is first necessary to briefly describe the TSO environment at MIT.

The MIT Computation Center runs OS Release 21 with TSO on a System 370 Model 165 with 1.5 megabytes of core storage. Several 3330 Disk Drives are available, as well as two 2301 Drums for swapping and system datasets. A hybrid of ASP (Attached Support Processor) and LASP (Local Attached Support Processor) is used in support of the Main 165. Although several MIT modifications have been made to the system, the basic structure and user interface of TSO remain standard (see [8,9]). It should be noted here that MIT has a standard dataset naming convention for user datasets, which reflects the account number and programmer to be billed for the online starage. A standard MIT/TSO dataset might have the following name:



Each programmer on the system has a unique programmer number, and each project is assigned a unique account (or problem) number. Thus, only valid problem-programmer combinations are accepted within the system. Under TSO, a unique problem programmer number is associated with each USERID-PASSWORD combination, and the prefix "U.PROB.PROG" is automatically added to all dataset references (see [8] for details of TSO dataset conventions).

Under standard TSO, as might be expected, "all users are created equal." That is, each entry in the User Attribute Data Set (UADS) is assumed to be a "full" user, with all of the priviledges available to users of the system. The only exceptions to the notion of the standard user are users whose UADS entry contains either an administrative or operator flag. Thus all standard users have the same command repertoire and dataset access rights. Under OS-360, these dataset access rights are quite extensive, since they include all datasets in the system. In fact, there are currently only two ways to remove access to datasets in the system -- first, they may be password protected (a clumsy and not often used facility), and second, the physical device containing the dataset may be placed in read-only mode (this is done at MIT with the system residence volume). Note that the second method cannot protect against read access, while even the first cannot hide the existence and location (volume serial number) of the dataset in question. Thus we see that the standard TSO user has a quite potent facility at his command, which includes the ability to modify or delete more than 90% of the datasets on the system, either accidentally or deliberately.

Rounding out our discussion of standard users, consider the administration of USERID's within the system. The central user accounting office must handle both batch and TSO accounting. At any given time, there are several thousand active combinations of problem-programmer numbers, as well as hundreds of valid TSO USERID's. As classroom use of computers increases, the potential load on this office is obvious -- several thousand students, each involved in several courses, combine to produce many thousands of USERID-PASSWORD combinations. If we assume that the instructor in any given course would probably exercise fairly tight control over the course budget, the workload of the central office in closely overseeing all users becomes immense.

The Classroom Support Environment lends itself naturally to decentralized control of computer resources. Since the student users are already organized into classes, it seems logical to allow the person in charge of a particular class to allocate computer resources in any desired manner. Further, some controls should be provided for the class adminstrator to insure that the class budget is spent equitably. For example, in the current MIT accounting system (which, although primitive, is not unlike many university centers), if a class of ten students were each assigned standard TSO USERID's, all ten users would be billed against the class budget for the course. Now since they are "full" users, there is absolutely nothing to prevent a student from logging onto the system, and using it to create, compile, and run FORTRAN decks instead of executing the desired tutorial program. Moreover, since all users in the class are being billed against the master budget (a separate account for each user is unthinkable) a single enthusiastic student

can play until the entire budget is consumed, preventing the other none students from ever logging on at all. This rather disturbing scenario at least had the advantage of assuming non-hostile intentions on the part of the student. If a student who wished to cause trouble was released on the system, almost every user dataset on the system could be deleted without fear of discovery.

It is clear that there is a need for a subsystem which can achieve the dual purpose of relieving the central accounting office and providing control of student users. Figure 10 indicates an expanded set of design goals, which will be discussed individually. Following that discussion, the implementation of the MIT Class Monitor will be described.

DESIGN GOALS FOR CLASS MONITOR

1) Get immediate and uninterruptable control at logon.
2) Validate users and log invalid users completely off the system.
3) Give the valid user a pre-designated "subset" of full TSO.
4) Protect itself and its files.
5) Handle user file maintenance.
6) Get uninterruptable control at logoff or console shutoff.
7) Handle on-line accounting.
8) Provide a mechanism for maintenance of the master accounting file.
9) Have low operating overhead.
10) Be easy to use for inexperienced students.
11) Be easy to maintain for center personnel.

FIGURE 10: DESIGN GOALS FOR CLASS MONITOR

GOALS OF THE MONITOR

No matter how we implement our system, it is still necessary to use the concept of the TSO USERID. Now, however, we assign several "open" USERID's to our class, and install the Class Monitor in each one. The first goal of our system, then, is that the Class Monitor must intercept the standard logon in some manner, and gain control of the logon session, thus encapsulating the user in a new environment immediately, and without failure.

Since we have given out "open" USERID's, the process of validation and account balance checking must be undertaken by the Class Monitor logon processor, so that only valid users are admitted. If students fail to give the proper identification, they must be logged off the computer system. Note that the USERID had to have been logged completely onto the TSO system before the execution of the CMS processor could begin. Therefore it is necessary for CMS to fire a direct call to the system logoff routine to prevent the user from taking any action prior to being logged off the system.

Having admitted the student to CMS, we now face the problem of control. The Monitor must be able to allocate to the user only those commands which are necessary to do the assigned work, and in a manner which will make it difficult to perform any but the intended tasks. As will be seen later, extreme care must be taken to avoid giving the user commands with which he may bootstrap into a more powerful environment. For example, if we give the user a restricted command library,

and also the ability to copy command processors, the user will simply copy more powerful commands into the library and then execute them. This is a difficult area in the design of subsystems which attempt to contain student users.

If the Monitor is to have online accounting, as well as a command library of some sort, it is clear that these datasets must be protected from user tampering. Further, it is desirable that the existence of such files be hidden from the user, to eliminate the temptation of such tampering.

Perhaps the most difficult area for the monitor system is the handling of user file maintenance. First, we encounter the problem of giving the user the ability to create and delete datasets. This entails releasing commands which constitute a direct threat to other datasets on the system. Second, many interactive teaching programs use online datasets to save intermediate results between logon sessions. Thus we should like to provide some means of creating and deleting datasets from within a higher level language such as FORTRAN or PL1. Finally, we encounter the problem of billing the individual student for dataset space, since the entire file maintenance system of IPC TSO is built around the notion that all of the files under one USERID should be treated as a single group. Ideally, the Monitor should allow for program control of dataset allocation, automatic distinction between different students' files under the same USERID, and billing procedures for these files.

It is essential that the Monitor get uninterruptable control at logoff time, to insure a clean CMS termination, and proper billing at the student level. Further, it is essential that control be received in the event of an abnormal termination such as console shutoff or telephone disconnect, since these events could easily bypass the subsystem accounting mechanism.

The notion of online accounting is important for class use. This is primarily true due to the scarcity of computer resources at this level. The MIT TSO system does not, at this time, have online accounting, primarily for reasons of security. That system produces punched card records of each logon session, and performs a daily update Monday through Friday. With the limited resources available to classroom users, a student could easily log on several times during a weekend, and consume much more than the parcel of time alloted, knowing that the billing would not catch up until Monday. This is quite unsatisfactory for the classroom environment. In fact, there is some reason to believe that the accounting should be done periodically during the logon session, so that the session could be terminated when the balance reached zero instead of waiting for LOGOFF processing to update the balance. At the very least, the Monitor must compute a reasonably accurate cost figure for the session, update an online accounting file, and display both the session cost and balance to the user, so that the students may budget their time properly.

Closely related to the notion of online accounting is the ability to maintain this file by adding or deleting users, allocating funds, changing passwords, printing reports, etc. All of these standard maintenance functions must be provided in conversational form so that the class administrator can maintain the file without assistance from the central accounting office.

The final three design goals are common to all types of subsystems. Obviously, the Monitor must not consume large amounts of computer time in providing the student environment, or the benefits of this environment will be outweighed by its consumption of resources. Since the Monitor is to be used by students of all disciplines, it must provide a set of interactions which are extremely straightforward, so that there is little chance of confusing inexperienced students (who tend to be somewhat frightened of the computer). Finally, the Monitor must be easily maintained, since the programming support staffs of the university generally have a fairly high turnover rate.

We have now outlined a set of design goals for a Classroom Support subsystem. In the second half of this chapter, we will discuss the implementation of the MIT Class Monitor System, which attempts to achieve these goals.

THE MIT CLASS MONITOR SYSTEM

The Class Monitor System consists of three major components:

LOGON PROCESSING

SESSION CONTROL

LOGOFF PROCESSING

We will examine each of these components and the implementation strategy used, concluding with a summary of the strengths and weaknesses of the system as currently available.

When a TSO user logs onto the system, the Terminal Monitor Program (TMP) is invoked. This is the standard IBM control program, which supervises the console session. It is the TMP which accepts and executes all user commands. The TMP also has a very important feature which we use to get control. At the beginning of the console session, the TMP looks at the PARM field of the EXEC statement which invokes it. (see [10] for details on the operation of the invocation process and catalogued procedure used). If any non-blank characters are present in this field, they are placed into the Command Buffer, and taken as the first command line to be executed. Further, no information is accepted from the terminal until this first command is processed to completion. This includes processing of "attention" interruptions, which are suspended until the first command terminates normally, and until the TMP issues the first READY message to the console.

The availibility of the "first command" option means that the Monitor can specify a single load module which will be executed before any user intervention is permitted. This load module must, of course, contain the CMS user validation scheme, as well as some ability to assure that invalid users are logged off the system. As a matter of convenience, the Master File Update Routine is included as a subroutine in the LOGON module, thus allowing us to implement this version of the Monitor using only two load modules, one at logon, and one at logoff. The modules are both written in PL1. Figures 11, 12, and 13 give the basic flowcharts of the system, and each one will now be explained.

Figure 11 depicts the operation of the LOGON processor. A standard TSO "nopass" option is used for all TSO CMS USERID's which allows the invocation of the TMP without a TSO password, that is, TSO logon is accomplished by merely typing "LOGON USERID." At this time, when finished with initialization, the TMP passes control to the CMS LOGON routine. The first function performed is the basic user validation. CMS uses group numbers for user identification. This was done so that the class administrator would not have to cope with both USERID's and passwords. Under this scheme, the administrator may assign group numbers based on a signup list or class roll sheet, and either assign or collect unique passwords. This scheme also tends to assist students by giving them a number instead of a character string as their identifier. If they forget their password, they can always identify themselves to the administrator by group number and get their memory refreshed. The user is given two tries to enter a valid

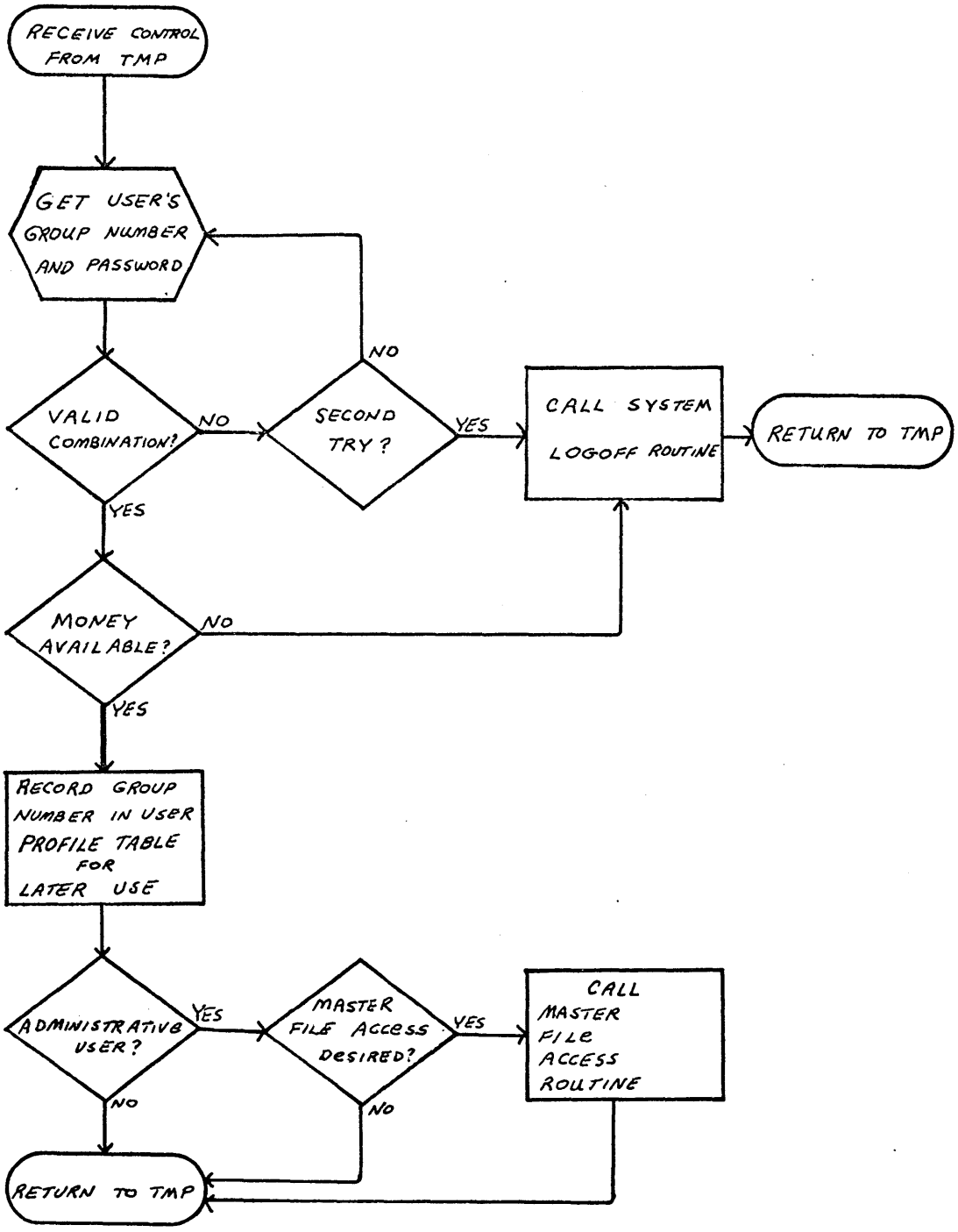


FIGURE 11: MACRO FLOWCHART OF CMS LOGON ROUTINE

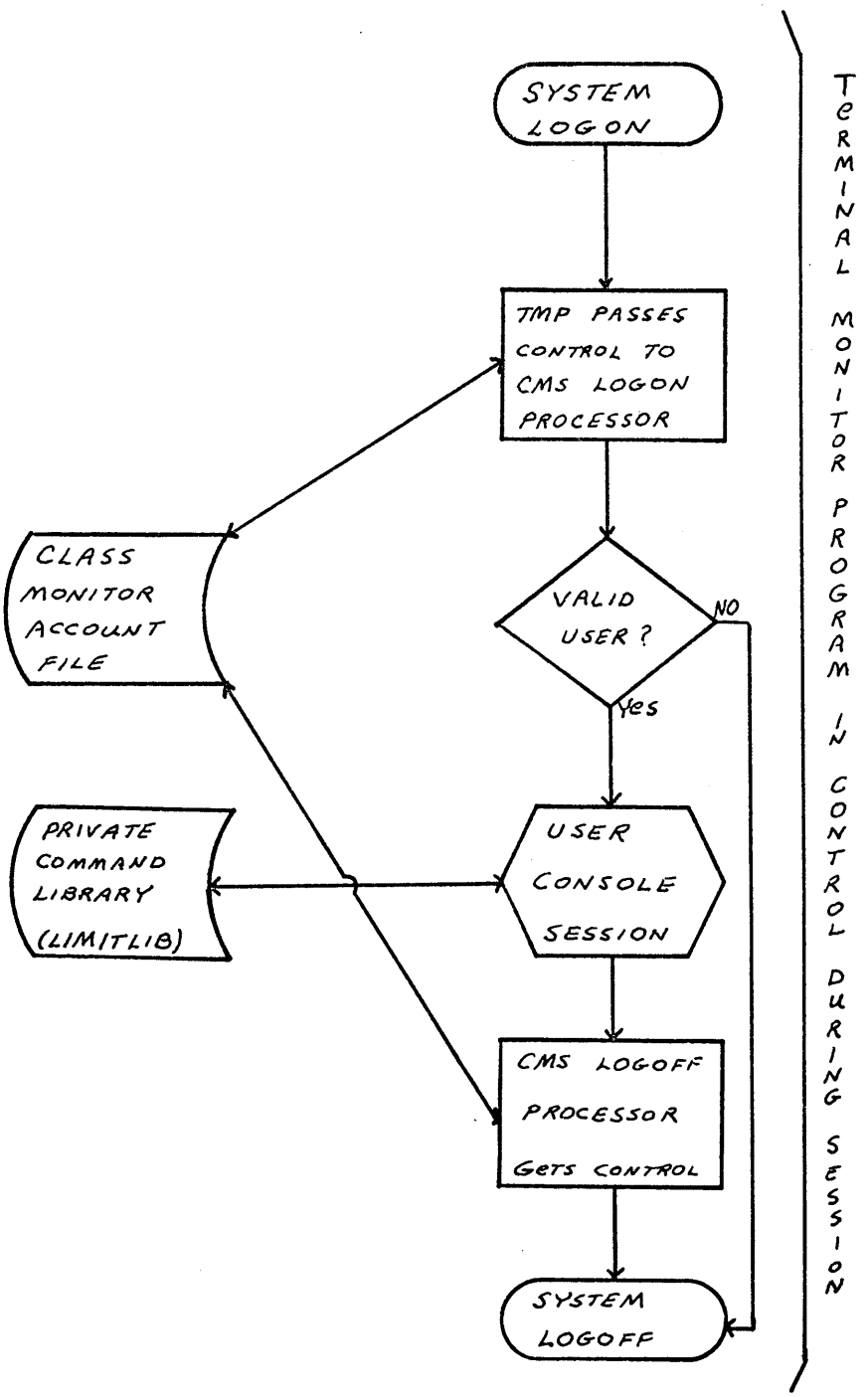


FIGURE 12: CLASS MONITOR CONSOLE SESSION

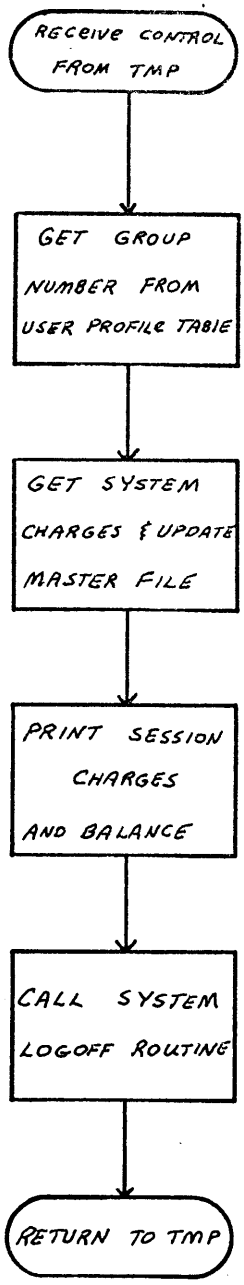


FIGURE 13: MACRO FLOWCHART OF CMS LOGOFF ROUTINE

group number/password combination. If unsuccessful, the user is logged off the system completely. This is made possible by the mechanism of the system logoff routine. It is called directly from within the LOGON processor, and simply turns on a bit in one of the system control blocks for the process. Whenever control is returned to the TMP, this bit is automatically checked, and if on, causes the logon session to be cancelled. Since control is returned directly from the LOGON processor to the TMP, this screening method cannot be subverted. The next task is to determine the account balance. If there is any money left in the account, the user continues processing. If not, an appropriate message is printed, and the user is logged off. At this time, we are certain that the user is a valid one, either a student user or an administrator. We must now solve an important problem caused by the modular nature of TSO.

Although we now know the valid user's group number, as soon as the LOGON processor is finished executing, all of the current information disappears as a new program is fetched into the TSO region for execution. How, then, does the LOGOFF processor know which group number to charge for the session? It is clearly undesirable to have the group number and password reentered at the end of the session. The problem is solved using the User Profile Table, one of the control tables which remain in use throughout the session, and in which, for example, the characters for character and line deletion are recorded. Several bytes of installation-usable space are reserved in this table, and one halfword is used to record the group number. Each time the TMP invokes a command, the

address of this table is passed in the argument list. This allows our LOGON and LOGOFF commands to communicate.

Having recorded the identity of the user, we check an administrative flag in the accounting record. If the user has administrative privileges, we give him an opportunity to access the Master File for maintenance. Otherwise, we simply return to the TMP, our work completed. If the user accesses the Master File, we return to the TMP when finished.

The Master File Access Routine is a PL1 procedure callable from the LOGON processor. It is simply a conversational file maintenance routine, which provides the obviously necessary functions of administration, such as adding and removing group numbers, changing passwords, allocating money, printing reports, etc. Most of the programming and design involved are relatively mundane, and available to the interested reader in the CMS Programmer's Guide.[11] One feature which is relevant here, however, is the hierarchical scheme currently employed for administration. Users are classified either as non-administrative (no access to Master File), administrators (access to the records of all student users in the file), and super-users (access to all records in the file). Without judging the merits of the scheme, it is interesting to note that almost all subsystems of this nature possess the "deity syndrome", that is, there is always one particular user (in this system called a 'super-user'), who has absolute accessing rights to all information within the system. This usually starts as a protective measure during the debugging stages of the development, when it is clearly advantageous to have an override mechanism for emergencies. But somehow,

as time goes on, this facility is never removed, so that the creator of the system always has "the power". It is likewise true of CMS that the current CMS administrator has absolute access to all Master Files for all classes. Since the class administrator is often an inexperienced student assistant, it has been found that the override facility has been very useful so far.

LOGON processing, then, includes user validation and screening, account checking, posting of group numbers, and Master File access for administrators. Control is then returned to the TMP.

When the TMP regains control from the CMS LOGON processor, the system logoff bit is immediately checked. If this bit is set, the session is terminated. If not, the user is presumed to be valid, and a READY message is issued to the terminal. (For a description of the CMS LOGON and LOGOFF user interface, see [12].) At this point, we are at the TSO READY State. This is often referred to in other timesharing systems as the Supervisor Level, or Command Level. When a user is in READY state, the TMP is ready to process commands from the terminal.

Having performed the necessary functions at LOGON, we are now faced with the problem of controlling the user console session, as depicted in Figure 12. As noted earlier, we would like to give the user the absolute minimum number of commands necessary to perform the assigned task. In doing so, we would like to use standard TSO commands as much as possible, so that when a user has a question about a command, all of the existant TSO documentation (such as the Command Language

Reference Manual) will still be relevant. The procedure then, is to start with the standard set of TSO commands, and simply remove those that are not needed. This turns out to be an extremely easy task, because of the straightforward way in which TSO processes commands. When a command line is sent to the Command Buffer for processing, the TMP has a standard search path for locating the proper program to fetch. The part that concerns us here is the location of the commands themselves. They all reside in a partitioned dataset called SYS1.CMDLIB. Each command is a separate member of the dataset, with its command name and abbreviations corresponding to a member name and aliai. In order to restrict the commands available to the user, we simply copy the desired subset of the TSO commands into a new partitioned dataset, and substitute that dataset into the search path in place of the standard command library. The DDNAME of this new dataset is LIMITLIB, and the TMP is modified to make the substitution. Thus we tailor the command set of a given class by varying the members of the LIMITLIB used for that class. When commands are requested which are not included in LIMITLIB, the standard TSO error message "COMMAND XXXX NOT FOUND" is received.

For some applications, the LIMITLIB alone is a satisfactory solution to our control problems. However, in many cases it is necessary to allocate and delete datasets in order to run instructional programs. If we release the ALLOCATE and DELETE commands to users for this purpose, then they will be able to allocate any datasets they wish (or accidentally allocate large datasets by incorrectly specifying the allocation para-

meters) as well as delete any dataset on the system. Therefore we use the TSO EXEC feature, which allows us to store commands in a dataset for execution. Then we rename the commands we need, placing the renamed members in LIMITLIB, and using the new commands in our EXEC file. Finally, we remove the option of the EXEC command which allows the command dataset to be listed, so that the user cannot learn the new command names (otherwise, the user could simply invoke them from LIMITLIB by their new names). This procedure allows us to include commands in LIMITLIB which are too powerful to give to the user, providing that we are careful to avoid giving out the ability to discover the new command names. The LIMITLIB dataset and the EXEC command, then, provide user control during the console session.

At the end of the console session, CMS must get control to perform accounting functions and print charges. This is accomplished by inserting a CMS module into LIMITLIB as LOGOFF. The flowchart for LOGOFF is shown in Figure 13. When the user types the command LOGOFF, the CMS module is fetched instead of the standard system program. After CMS performs its accounting functions, it calls the system logoff routine, assuring that when control is returned to the TMP, a normal session termination will occur.

Referring back to Figure 10, let us evaluate the current version of the Monitor in light of our design goals. Following that, we will discuss some of the security aspects of the Monitor.

We have seen that CMS LOGON gets the desired control at logon time, and properly validates all users. The combination of LIMITLIB and EXEC

provides the ability to tailor the TSO command language. Protection is accomplished mainly by hiding essential commands and datasets from the user. At the present time, no file maintenance is performed by CMS, due to the lack of an interface between higher level languages and DAIR (Dynamic Allocation Interface Routine). Several schemes are being studied to implement a file subsystem. The LOGOFF routine gets control when the user issues a logoff request. In Version 2 of the Monitor, abnormal termination such as console shutoff will also be handled. Online accounting and Master File maintenance are provided. The system has proven to be extremely easy to use and maintain, due to its modular design and close resemblance to normal TSO. Finally, the overhead has been measured at approximately one dollar per student session, and this figure will be reduced in the next version.

SECURITY AND THE MONITOR

Although the Class Monitor has proved a useful tool in non-computer courses at MIT, it is of limited value in the control of computer-oriented courses due to its vulnerability. It lacks protection in two major areas.

First, since the datasets used to implement the monitor are simply normal user datasets, they may be edited, listed, or deleted by other "full" TSO users. A specific example of this occurred last fall, when a staff member at the Computation Center, in an effort to assist a Class Monitor user, listed his EXEC files using a command not available within LIMITLIB. Thus the code names (and access) to all commands were given to this user. Further, students who are given full TSO access for thesis projects are free to alter or destroy all of the essential datasets for the Monitor. Therefore, the Monitor is open to sabotage from non-Monitor users.

A second area of vulnerability arises from the fact that all of the programs within the Monitor are written in PL1, and run in User Mode. If we give a PL1 programming class access to a version of the Monitor which includes the ability to edit, compile, run and delete PL1 programs, we face two problems: first, anything that the Monitor does, they can do (for example, gain access to the User Profile Table, and change the group nember to be billed for the session), and second, the logical structure of TSO provides no method of keeping different user's datasets apart in a single USERID, and releasing file maintenance

commands gives unlimited access to all system datasets. Therefore, we have relied upon the good nature of our students when using the Monitor for more sophisticated applications.

The final chapter discusses the IBM Resource Security System, and what additional security it will bring to the Class Monitor in the OS/TSO environment.

CHAPTER 4 - RSS AND CLASSROOM SUPPORT SECURITY

INTRODUCTION

In May of 1973, the IBM Resource Security System (RSS) was installed at the MIT Computation Center as part of a study on operating system security. RSS is designed as an addition to OS Release 21 which, when fully implemented (and debugged), provides the additional software necessary to "secure" the operating system. Since extremely detailed documentation is available for RSS [13, 14, 15], only a very brief description will be given here.

RSS is a system primarily concerned with data security, and one which clearly reflects its military ancestry (it was originally designed for use in the World-Wide Military Command and Control System). System resources (programs, datasets, and terminals) are accessed by users on the basis of security levels, access categories, and need-to-know. Security levels reflect the sensitivity of data, in a manner directly analagous to the military "confidential, secret, and top secret" classification. Datasets are assigned one of eight security levels, and each user has an attribute which sets a maximum permissable level of data access. Access categories provide a means of implementing a group need-to-know strategy by associating groups of users with groups of system resources. For example, all of the administrative users in a given department might be authorized to the set of confidential files for that department. Finally, in the extreme case, individual users can be authorized to specific datasets on a specific need-to-know basis.

Perhaps the most important reflection of the military design strategy in RSS is the notion of the Security Officer. The control of all security procedures within the system rests with the person (or persons) designated as Security Officer. This control includes the definition of access categories, maintenance of the security profiles of all users, and the control of the authorization procedure for all controlled datasets.

Through the authorization procedure described in [13], users are given rights to datasets on the basis of their codewords, which specify access categories, levels, and need-to-know. The RSS System then monitors the use of all controlled datasets, and attempts to prevent any access to the system which might subvert the control mechanism.

Since the MIT community has had little opportunity to observe the system, we will not critique its effectiveness here. For interested readers, some information on performance is available from Cornell University [16]. We will, however, take a brief look at the potential effectiveness of RSS in allieviating the security problems of the Class Monitor. The chapter will conclude with some comments on the design of operation systems for use in the Classroom Support Environment.

THE POTENTIAL OF RSS

When OS-360 was originally introduced, the designers were very proud of the ease with which data in the system was accessed. It was a very "open" system, and the most flexible available in terms of file system organization. As the need for data protection became clear, and TSO was added to the environment, RSS was developed to gain control of the system. For the Class Monitor, this means the ability to protect its control datasets from outside disturbances, and further, to authorize the contents of those datasets to specific programs. For example, the accounting file could be authorized only to LOGON, LOGOFF, and the Master File Access Routine, thus preventing access by student programs. However, it sould be noted here that even though this is a major improvement in OS-TSO, the result is no advance in the state-of-the-art. Indeed, there is some reason to believe that the design of a timesharing system in which one user can access another user's files (or even know that they exist) was a terrible mistake at best, and that dataset protection in TSO in fact brings the design of the system up to a level just below that of systems such as CP/CMS, since the user in TSO can still find out that controlled datasets exist from the system catalog.

In the Classroom Support Environment, as noted in Chapter 3, a very decentralized user community exists. Unfortunately, in RSS, only the Security Officer can protect datasets and assign privileges. This military notion of security centralization is in direct conflict with the needs of our environment. This and other problems of RSS in a

"service bureau" environment are discussed by Daley.[17]

RSS provides no assistance in the other major area of difficulty, that of preventing the user from accessing the User Profile Table and other sensitive control tables during execution. Although RSS in most cases can catch a user before the OS environment is affected, it offers no assistance in maintaining the subsystem environment needed in our application.

In summary, the addition of RSS to OS-360 provides some useful control of sensitive datasets, but fails to provide the mechanisms necessary for subsystem control, such as automatic user exits from various sections of the TMP, and authorization mechanisms which operate without the Security Officer. Much modification of TSO is needed before the Class Monitor can be secured, and RSS must be extended to include more specific authorizations, such as the execute-only access and program-program-file permission discussed by Daley. RSS, while solving many dataset access problems, falls short of the requirements of the Classroom Environment.

CONCLUSIONS

Experience in the Classroom Environment at MIT has shown that a decentralized approach such as the Class Monitor System provides the necessary simplicity and computing power for students, as well as the control required for administrators. Severe problems occur, however, in attempting to provide adequate subsystem integrity. It seems clear that any operating system which intends to service this environment must include mechanisms for subsystem implementation which include access (either by user exits or open entry points) to most major modules of the system. File systems, accounting systems, command processors, and many other areas must be available to provide adequate subsystem security.

In conclusion, we should note that there are some viable alternatives to the OS environment for Classroom Support. Madnick and Donovan[18] make a strong case for the use of virtual machine systems in areas where security is a problem. Certainly this idea is appealing in attempting to isolate computer research, for example, from other campus activities. Even with a class virtual machine, however, some mechanism will be required to protect members of that class from one another, making some subsystem necessary. The Multics design[19] is one which makes the implementation of subsystems somewhat easier.

The needs of the Classroom Support environment, then, are mainly in the area of subsystem security. Hopefully, the work of the RSS Study

Group at the Sloan School of Management will provide a framework for analyzing these needs which will assist designers in meeting them more completely.

REFERENCES

1. Levien, R.E., Blackwell, F.W., Comstock, G.A., Hawkins, M.L., Holland, W.B., Mosmann, C. The Emerging Technology: Instructional Uses of the Computer in Higher Education, Preliminary Edition, The Rand Corporation, Santa Monica, California, 1970.

2. Interview with Joseph Patten, Director, Office of Administrative Information Systems, MIT on April 26, 1973.

3. Lavigne, Jean C., Porges, Denis, S.M. Thesis, MIT Sloan School, A Management Information and Control System for a School and its Applications to the Sloan School of Management, 1971.

4. Chaney, John F., "Data Management and Interrelated Data Systems for Higher Education", Management Information Systems: Their Development and Use in the Administration of Higher Education, Western Interstate Commission for Higher Education, Boulder, Colorado., October 1969, pp. 17-27.

5. Rand Corporation (op. cit.) p. 168.

6. IBM System/360 Principles of Operation, IBM Corporation, Form GA22 - 6821.

7. IBM System/360 Operating System, Introduction, IBM Corporation, Form GC28 - 6534.

8. IBM System/360 Operating System, Time-Sharing Option - Command Language Reference, IBM Corporation, Form - GC28 - 6732.

9. IBM System/360 Operating System, Time-Sharing Option - Terminal User's Guide, IBM Corporation, Form GC28 - 6763.

10. IBM System/360 Operating System, Time-Sharing Option - Guide to Writing a Terminal Monitor Program or a Command Processor, IBM Corporation, Form GC28 - 6764.

11. Hewitt, D. "Class Monitor System Programmer's Guide", Multilithed Manual, East Campus Computer Facility, MIT, 1973.

12. Hewitt, D. "Class Monitor System User's Guide", Multilithed Manual East Campus Computer Facility, MIT, 1972.

13. IBM OS/MVT With Resource Security: General Information and Planning Manual, IBM Corporation, Form GH20 - 1058.

14. IBM OS/MVT with Resource Security: Installation and System Programmer Guide, IBM Corporation, Form GH20 - 1021.

15. IBM OS/MVT with Resource Security: Security Officer's Guide, IBM Corporation, Form GH20 - 1057.

16. Weiner, Herb. "An Analysis of Computer Software Security at Cornell," Working Paper, Cornell University, Ithaca, New York, February, 1973.

17. Daley, Robert C., "Authorization and Disseminated Control in a Resource Security System," Paper Presented at the IBM Quarterly Security Conference, MIT, Cambridge, Massachusetts, April, 1973.

18. Madnick, S.E., and Donovan, J.J. "Application and Analysis of the Virtual Machine Approach to Information System Security and Isolation", Paper Presented at the ACM Workshop on Virtual Computer Systems, Harvard University, Cambridge, Massachusetts, March, 1973.

19. Bensoussan, A., Clingen, C.T., Daley, R.C. "The Multics Virtual Memory: Concepts and Design", Communication of the ACM, Volume 15, Number 5, May, 1972.