NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

SUBVERSION:

THE NEGLECTED ASPECT OF COMPUTER SECURITY

by

Philip A. Myers

June, 1980

Thesis Advisor: Roger R. Schell

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Subversion: The Neglected Aspect of Computer Security		5. TYPE OF REPORT & PERIOD COVERED Master Thesis; June 1980
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Lt. Philip Alan Myers, USN		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey California, 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California, 93940		12. REPORT DATE June 1980
		13. NUMBER OF PAGES 113
14. MONITORING AGENCY NAME & ADDRESS(If different from Controlling Office) Naval Postgraduate School Monterey, California, 93940		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for Public Release; Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

subversion, protection policy, trap doors, Trojan horses, penetration, computer security, access control, evaluation criteria protection systems, leakage of data, security kernel

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This thesis distinguishes three methods of attacking internal protection mechanisms of computers: inadvertent disclosure, penetration, and subversion. Subversion is shown to be the most attractive to the serious attacker. Subversion is characterized by three phases of operations: the inserting of trap doors and Trojan horses, the exercising of them, and the retrieval of the resultant unauthorized information. Insertiion occurs over the

entire life cycle of the system from the system design phase to to the to the production phase. This thesis clarifies the high risk of using computer systems, particularly so-called 'trusted' subsystems for the protection of sensitive information. This leads to a basis for countermeasures based on the lifetime protection of security related system components combined with the application of adequate technology as exemplified in the security kernel concept.

Subversion:
The Neglected Aspect of Computer Security

by
Philip A. Myers
Lieutenant, United States Navy
B.S., North Carolina State University, 1973

Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June, 1980

Author: _____Philip G. Myers LT. USN_____

Approved by:

_____Roger R. Schell_____

Thesis Advisor

_____Douglas R. Smith_____

Second Reader

_____

Chairman, Department of Computer Science

_____

Dean of Information and Policy Sciences

ABSTRACT

This thesis distinguishes three methods of attacking
internal protection mechanisms of computers: inadvertent
disclosure, penetration, and subversion. Subversion is shown
to be the most attractive to the serious attacker.
Subversion is characterized by three phases of operations:
the inserting of trap doors and Trojan horses, the
exercising of them, and the retrieval of the resultant
unauthorized information. Insertion occurs over the entire
life cycle of the system from the system design phase to the
production phase. This thesis clarifies the high risk of
using computer systems, particularly so-called 'trusted'
subsystems for the protection of sensitive information. This
leads to a basis for countermeasures based on the lifetime
protection of security related system components combined
with the application of adequate technology as exemplified
in the security kernel concept.

TABLE OF CONTENTS

ACKNOWLEDGEMENT

I would like to thank my thesis advisor Lt. Col. Roger R. Schell, USAF, for the many hours he has spent in helping me to understand the relevent issues involved in computer security.

I. INTRODUCTION

To use internal mechanisms within a computer system to protect sensitive information without demonstrable assurances as to the origins and effectiveness of the system components is contrary to a sound security practice. Use of allegedly 'secure' or 'trusted' subsystems based on operating systems that are fundamentally unsecurable is likewise unsound. Yet these two conditions can, and do, exist within the constraints of current ADP security policy and practice. As a result, supposely 'secure' computer systems present a major risk of compromise for sensitive information.

These conditions can exist because there is a basic lack of understanding as to the possible vulnerabilities of computer systems. In particular, subversion is one area that is widely neglected. The subversion of a computer system is the covert and methodical undermining of internal and external controls over a systems lifetime to allow unauthorized and undetected access to system resources and/or information.

This thesis details the methodologies involved in subversion, and how they can be used to attack a computer system. It is essential that all personnel involved in ADP

security understand subversion and how it works. Without this understanding, effective policies and countermeasures cannot be devised and implemented.

The increased use of 'off the shelf' ADP systems and programs can help realize significant economies in procurement costs, but there are significant dangers as well. These dangers come about because there is a pressing need for computer systems to 'securely support multiple users of differing degrees of trustworthiness simultaneously handling data of differing degrees of sensitivity'. This is known as the classical computer security problem [1]. It is a problem because no known commercially available system can be proven to offer the secure support required.

Present technology such as that found in the Security Kernel [2] concept point the way to a solution to the computer security problem. But no technology will assure secure computer systems unless proper safeguards are implemented to protect this technology from subversion.

To understand what is involved in the subversion of computer systems one must first be aquainted with the background of the computer security problem (Chapter II). The problem is not merely a historical one. There is currently no clear policy as to what role computer systems are to play in the protection of information. As a result, systems are plagued with inadequate internal protection mechanisms whose effectiveness cannot be assured. Chapters

11

III and IV deal with how these inadequacies can be exploited
through subversion. Finally Chapter V discusses how the risk
of subversion can be minimized.

II. UNDERSTANDING THE COMPUTER SECURITY PROBLEM

The computer security problem has grown with the computer industry. When the entire system was dedicated to a single user, protection consisted of the user simply picking up his tapes and cards and clearing CPU core when the job was finished. Basically the user had complete control over his processing environment, including his data and programs. After a few years users began demanding better utilization of the resources. The response to this demand for more efficiency gave birth to multiplexing techniques, resource sharing operating systems, multiprogramming and various other techniques of the age. The user suddenly found not only a lack of control over the processing environment but a lack of control over the protection of his data and programs as well. Gat [3] indicates:

> With the appearance of multiplexing techniques there arose the problem of defending independent software structures from each other, as these were often implemented on the same physical resource. Thus, multiprogramming operating systems enforce some sort of isolation between simultaneously executing processes.

Since efficiency was the main consideration in computer systems design, criteria limited the ´defending´ and ´isolation´ to the containment of accidents and errors [2].

13

Organizations desiring to utilize the increased capacities of resource sharing systems demanded assurances that sensitive and nonsensitive information could be processed concurrently. Bisbey [25] comments:

> Responding to customer pressure, the systems manufacturers at first claimed that hardware and software mechanisms supporting resource sharing would also (with perhaps minor alterations) provide sufficient protection and isolation to permit multiprogramming of sensitive and nonsensitive programs and data.

This claim was soon discounted in the early 1970's with the introduction of several penetration tiger teams that were specifically tasked to test the protection offered by several major operating systems. Even those systems that underwent 'retrofitting' to correct known implementation errors and design oversights were penetrated with only moderate amounts of energy [1]. Evidence as recent as 1978 indicates that current operating systems for which the major vendors have 'conscientiously and competently attempted to improve security' have been successfully penetrated [1].

Finally, as a crowning blow to the state of current computer systems, a Consensus Report published in the proceedings of the 1979 National Computer Conference [1] states:

> It is a fact, demonstrable by any of several studies, that no existing commerically-produced computer system can be counted upon to protect any of its moderately knowledgable users from having complete and undetectable access to any information in the system, no matter what kinds of

so-called security features or mechanisms have been built into the system.

Harrison, Ruzzo, and Ullman in their paper 'Protection in Operating Systems' [4] provide conclusive proof that there is no algorithm that can prove an arbitrary protection system (such as an operating system) safe. This means it cannot be proven that an arbitrary operating system can withhold unauthorized information from malicious users. This is because a system may not be (and usually is not) designed in a manner that its safety can be precisely determined. However, for a properly designed system the safety question could be decided. But, the constraints placed on these 'model' systems are too severe to prove practical for the evaluation of current operating systems. In particular, systems designed using the security kernel technology [3] can be definitively evaluated for security. This technology will be briefly discussed in Chapter V.

It has been said that understanding the computer security problem requires close attention to three subjects: policy, mechanisms, and assurance [1]. It is essential to understand all aspects of the problem. Therefore, a brief discussion of each area is offered.

A. LACK OF COHERENT POLICY

In general, a security policy defines what is meant by 'secure'[5]. The sources of this policy are laws and

regulations that outline how information is to be handled.
The computer industry in general, both users and vendors,
have not reached a consensus as to what would constitute a
coherent approach to computer security policy. The Consensus
Report [1] indicates:

> This passive attitude on both sides tends to mask the
> general nature of the security problem because the more
> knowledgeable security users demand solutions to their
> unique problems, solutions that might not become standard
> parts of a product line.

DOD fairs better in having a more specific policy as to
the handling of sensitive information in general. This
policy involves a non-discretionary (or mandatory) access
control and within these contraints a discretionary control.

> When information is given a formal security
> classification, it is forbidden without explicit
> administrative declassification or downgrading to allow
> someone to have access to information of higher
> classification than he is cleared for, i.e., the holder of
> classified information has no discretionary authority in
> this respect concerning who he can share it with. This
> rule is an example of a mandatory access control policy
> [1].

Within the mandatory constraints there exists a
discretionary policy that allows the creator of the
information discretion over access to the information by
other cleared personnel. This is the concept of 'need to
know'. A person must have the clearance (mandatory) and a
need to know (discretionary) before access to information is
granted.

However in the area of sensitive information as it relates to the computer, guidelines, such as those outlined above, are less clear. Policy does not clearly discriminate between a computer providing only computation and one providing both computation and protection [6].

In a simple computation environment, protection or security is enforced by physical means external to the computer (fences, guards, etc.) as in a 'dedicated' mode of operation. In this mode, all users allowed access to the system are cleared for the highest level of information contained in the system (i.e. it is dedicated to processing at a given security level). All users, equipment, and information reside within this protective boundary or 'security perimeter'. Everything within the security perimeter is considered benign. The computer system is not expected to seriously 'defend' information from any of its users because they are considered non-malicious by virtue of their security clearances.

In the other environment (called the multilevel security mode) the computer not only provides computation but must internally provide mechanisms that distinguish levels of information and user authorization [6]. This is because not all users of the system are cleared for the highest level of information contained in the system. Here, the computer system must protect the information from the uncleared (and possibly malicious) user. In effect, the computer system

must become part of the security perimeter. The internal protection mechanisms (whatever they may be) must 'assume the role' of the guards, fences, etc. that are indicitive of the external security perimeter. Policy (which defines what is meant by 'secure') must be clearly translated into terms that can be implemented on a computer. Unless a specific policy is required to be implemented on a computer system in a VERIFIABLE manner, there would be no way one could determine if the computer system was EFFECTIVE in enforcing the given policy.

B. INADEQUATE INTERNAL MECHANISMS

The baseline documents within DOD for ADP security are DOD Directive 5200.28 'Security Requirements for ADP Systems' [7] and its associated Manual DOD 5200.28M 'The ADP Security Manual' [8]. The Directive states that 'techniques and procedures which can be used to secure and evaluate resource-sharing ADP systems' are contained in the ADP Security Manual. Therefore, it is instructive to specifically address the Manual.

Since the central issue of a multilevel security system concerns the use of internal protection mechanisms to enforce protection of information, it is important to understand what these mechanisms are.

The following are selected excerpts from the Manual that illustrate the offically annunciated role of internal software mechanisms:

4-300 General

The user and master modes of ADP Systems operation shall be separated so that a program operating in a user mode is prevented from performing control functions.

4-301 O/S Controls

The O/S shall contain controls which provide the user with all material to which he is authorized access, but no more.

4-305 Other Fundamental Features

.... Unauthorized attempts to change, circumvent, or otherwise violate these features should be detectable and reported.... In addition the incident shall be recorded in the audit log....

a. Memory/Storage protection - The operating system shall protect the security of the ADP system by controlling:

    1. Resource allocation (including primary and auxiliary memory);

    2. Memory access outside of assigned areas; and

    3. The execution of master (supervisory) mode instructions which could adversely affect the security of the O/S.

b. ...

c. Access Controls - Access to material stored within the ADP System shall be controlled by the ADP system security officer, ..., or by automatic processes operating under separate and specific controls within the O/S established through hardware, software, and procedural safeguards approved by the ADP System security officer.

d. ...
e. ...

f. User identification — Where needed to assure control of access and individual accountability, each user or specific group of users shall be identified to the ADP system by appropriate administrative or hardware/ software measures. Such identification measures must be in sufficient detail to enable the ADP system to provide the user only that material which he is authorized.

These seem to be reasonable requirements to ask of a multilevel security system. The problem is that there is no way that these requirements can be proven effective. They can only be proven ineffective. This is evident in the ADP Security Manual's ad-hoc method of Security Testing and Evaluation (ST&E). An evaluation is defined in paragraph 1-213 of the manual:

> The evaluator's report to the Designated Approving Authority describing the investigative and test procedures used in the analysis of the ADP System security features with a description and results of tests used to support or refute specific system weaknesses that would permit the acquisition of identifiable .classified material from secure or protected data files.

Verification is defined in paragraph 1-225:

> The successful testing and documentation of actual on-line system penetration or attempts to penetrate the system in support or in contradiction of assumptions developed during system review and analysis which are to be included in the Evaluation report.

The above methodology is fundamentally flawed. Recall from mathematics that it is sufficient to disprove a proposition (e.g., that a system is secure) by showing only one example where the proposition is false (e.g., a successful penetration). It is not sufficient to prove the proposition by offering an example where the proposition

20

appears to hold (e.g., unsuccessful penetration attempt).
The best position to take concerning these methods is stated
by Schell [6]:

> Do not trust security to technology unless that technology
> is demonstrably trustworthy, and the absence of
> demonstrated compromise is NOT a demonstration of
> security.

It is imperative that any mechanism that will be
required to aid in the securing of a computer system be
constructed in such a way that it can, in fact, be verified
effective.

C. FALSE ASSURANCES

False assurances concerning the reliability of computer
systems to effectively protect information come about
because people in positions of responsibility do not
understand that a 'technical computer security' problem
exists.

> .....government agencies, as well as private industry,
> continue to issue purchase requests containing sections
> labeled 'security requirements', which are mostly lists of
> features and mechanisms, in the apparent belief they will
> obtain something useful [1].

The previous section's discussion on policy illustrated
how the reliance on 'features and mechanisms' without
demanding demonstrable effectiveness can lead to false
assurances.

No self respecting computer system salesman is going to admit that his products cannot provide the effective protection that an application demands. No malicuous intent is implied by this statement, but the salesman is no more aware of the true nature of the computer security problem than the customer who unknowingly demands the ineffective 'features and mechanisms' in a procurement specification. The Consensus Report [1] demonstrates this lack of understanding:

> .....even if government procurement specifications were tightened to ask for the kind of security we believe possible with the current state of the art, fewer than fifty people in the country would understand the true implications of what is being asked for, and those fifty are concentrated in less than a half-dozen organizations, none of them in the main stream development organizations of the major mainframe vendors. This is partly because at the moment most efforts of vendors relating to security are concentrating on the 'mechanisms' part of the security problem, with very little attention to the 'assurance' part.

1. Reliance on 'Trusted' Subsystems

A subsystem can be viewed as any computing environment that restricts the users functions to a subset of the host computer's functional capabilities. An example of this is a transaction data management system. The user is bound to a restricted 'menu' of functions that allow him to carry out only his required tasks. For instance, a data entry clerk in such a subsystem has no need to write programs, so this capability is not part of the clerk's

menu. The general feeling about subsystems is that by restricting the users capabilities, he will be denied the 'tools' he needs to perform malicious activities.

Alleged 'secure' or 'trusted' subsystems are presently being developed within DOD as a means of coping with the computer security problem:

> Given an untrusted operating system, this approach employs the use of a trusted transaction data management system or other trusted special-purpose subsystem in concert with facility and procedural constraints that limit the population of users to the trusted subsystem. (Only trusted users are allowed access to any parts of the system outside of the trusted subsystem.) This solution combines trusted software (but not the operating system itself) and trusted procedures, and is an expedient until completely trusted operating systems are more widely available. Secure subsystems development for the DOD in limited transaction applications is currently underway [1].

Unfortunately one cannot exclude the operating system from the 'solution' as proposed in the above. All subsystems are 'built' upon an underlying operating system. The operating system must therefore be considered as an integral part of the trusted subsystem.

Ample discussion has already been offered as to the unreliability of current operating systems. A subsystem, when viewed from the aspect of the underlying operating system, is nothing more than another application program. If there are exploitable flaws in the underlying operating system that can be used to exploit the system without the subsystem, then these same flaws can be used to exploit it

with the subsystem. Chapter IV demonstrates how this can be done. Reliance must not be put on a 'trusted' subsystem unless the foundation on which it is built is solid and trustworthy.

2. No Lifetime Protection

There is no explicit Security Testing and Evaluation (ST&E) criteria in DOD guidlines that takes into account the history of system components. Using computer systems with uncertifiable backgrounds, particularly in multilevel security mode applications, can prove particularly disasterous. The main thrust of this thesis is concerned with just such issues. The lifetime of a computer system is not just the operational lifetime, i.e., when it comes under the control of an ADP security officer, but is from 'conception until death'. This includes the design, implementation, distribution, installation, and production phases of a computer system.

It is not sufficient to know that a given computer system and its associated software are standard 'off the shelf' versions of company XYZ's product line. Without specific assurances concerning the protective measures that have been afforded system components or the trustworthiness of development personnel, there is no way that an effective evaluation can occur. If at some time prior to the user taking control of a system, malicious elements have access

to system components, it would be virtually impossible to determine what modifications to invalidate security controls were made. This lack of protection is one of the fundamental reasons why the subversion of computer systems can be so effective. Later chapters will amplify this concept.

It has been proposed [1,9] that current operating systems be evaluated as to their security attributes. The result of this evaluation would yield an 'approved products list'. The resulting 'grade' that a system would receive would supposedly determine its relative ability to protect information. There is a problem in that this criteria does not substantivally address whether or not the security related components (hardware and software) have received the proper lifetime protection from malicious elements. Unless this vital factor has been taken into account, any 'approved products list' would prove meaningless.

D. CHAPTER SUMMARY

It has been the purpose of this chapter to aquaint the reader with the background of the computer security problem. This problem has been aggravated by a general lack of understanding as to the true nature of the computer security problem by those responsible for its solution. This has led to a reliance on inadequate internal mechanisms, and false assurances as to their effectivenss. It is important to

understand this background because it serves as a backdrop with which to view the subject of computer subversion, the principal topic of this thesis.

III. METHODS OF ATTACKING INTERNAL SYSTEM CONTROLS

There are three methods of attacking internal system controls in computers. They are are by inadvertent disclosure, penetration, and subversion. Each method is briefly discussed. Later chapters will develop the details involved in penetration and subversion. Distinctions are made between the current concept of penetration and the concept of subversion.

A. INADVERTENT DISCLOSURE

Inadvertent or accidental disclosures are basically probabilistic in nature. They may involve a combination of human, hardware, and timing factors that when combined could allow a disclosure of information to an unauthorized user. Simple examples of this method are a computer operator inadvertently mounting the wrong tape, or the hardware failure of memory bounds checking mechanisms. Users receiving information from this kind of disclosure are often victims of circumstances and may not be malicious in their intent. However, even though the success of this method relies on probabilistic events that one cannot control, it can be utilized by the determined attacker.

The basic approach used by an attacker in this method is to sit and wait for the proper set of circumstances to occur. Upon detection of a breach in the protection mechanism, the attacker would take appropriate actions to exploit the breach.

This method was addressed in the Multics Security Evaluation [10]. A program called the 'subverter' was written to run in the background of an unprivileged interactive process. Once each minute the subverter program received a timer interrupt and performed one test from a group of functions that would sample the integrity of the security sensitive hardware. These tests included:

1. Testing master mode instructions.

2. Attempting to violate read and write permission on segment access control lists.

3. testing of all instructions marked illegal.

4. Taking out-of-bounds faults on zero length segments.

Methods similar to those above could prove profitable to a malicious user, particularly if the system under attack had a history of questionable hardware reliability. Although this method is a viable attack method, other methods will be discussed that do not rely on these probabilistic circumstances.

B. PENETRATION

There are three major characteristics to penetration:

1. The penetrator is deliberate in his attempts.

2. The penetrator uses system foibles to circumvent system controls.

3. The methods are repeatable under the control of the penetrator.

It is important to realize that the penetrator is deliberate in his attempts. This is because it introduces a class of 'user' that contemporary computer system designers had not seriously considered. Designs reflect that the systems are expected to operate in a 'benign environment' where violations of the system controls are presumed to be accidential [2]. Because systems are presumed to be in a benign environment, the attacker does not have to exert much effort in his penetration attempts.

The second characteristic involves the utilization of system 'foibles'. Lackey [11] defines the term:

A foible is an accidental or unintentional opening that permits unauthorized control of the system or unauthorized access to information. It can occur in either hardware or software, but software penetrations are more common. A system programmer may inadvertently allow an obscure condition to occur for which no check is made, or accept parameters without adequate checking. Often the programs pass acceptance tests that don't expose these anomalies, and the program will work properly when used as intended.

Foibles that can be used by a penetrator to circumvent system controls come about because most computer designs for both software and hardware consider efficiency and convenience as primary factors rather than security.

The method is repeatable because the foible is a part of the system design or implementation. The penetrator can use it as though it were a 'special feature' of the system.

1. Penetration Environment

The penetrator carries out his malicious activities by using the computing (or rather the penetration) environment 'as is'. That is, he is content to exploit the system using those foibles that the designers and implementors inadvertently provided. But since deliberate penetration utilizes system weaknesses or foibles, the penetrator may have his 'access' routes cut off if the fallibility is discovered by a legitimate user or system maintenance personnel. However as indicated by Lackey, since the error was not detected during testing and the system works properly when used properly, this appears to be an effective method for gaining unauthorized information.

This is supported by reviewing the literature concerning computer crimes. Many of the criminals were not caught by the discovery of their penetration method or even in the actual act, but by some foolish action on the part of the criminal after the fact (e.g., high living on embezzled

funds). Only through subsequent investigations did the foibles become known to the victims.

But this environment, although lucrative, is not under the 'control' of the penetrator. Foibles could be discovered and corrected or procedural deficiencies revised. The determined penetrator would undoubtedly desire an environment that is more under his control and not as susceptable to change and possible detection by external forces.

2. The Penetrator

Current conceptions of computer system penetrators as glamorized by the newspapers and other popular literature would have one believe the the penetrator is a highly technical individual such as a programmer or computer scientist. This is a misconception. Several studies have shown that the a more accurate conception of the average penetrator is that:

1. He possesses only a limited technical knowledge of the computer system [12].

2. He is a 'white collar amateur' [13].

3. He is a user of the system, not the professional that supports the system [12].

4. He lacks the ability to think big [14].

But all these conceptions of the known penetrator reflect the same thing: that these conclusions are based on

on the amateur that got caught. They say nothing about the malicious elements that were sophisticated enough to avoid detection. It is this group that poses the greatest danger to the security of computer systems. What is the nature of the penetrator that was not caught, and how might he proceed in his malicious endeavors? It is imperative that these questions be addressed.

C. SUBVERSION

Recall from chapter I that subversion of a computer system involves the covert and methodical undermining of internal and external computer system controls to allow unauthorized and undetected access to computer system resources and/or information. But to understand the real implications of this definition, further amplification is required.

Subversion is characterized by the following:

1. It can occur at any time in the life cycle of a computer system.

2. It is under the control of highly skilled individuals.

3. It utilizes clandestine mechanisms called artifices deliberately constructed and inserted into a computer system to circumvent normal control or protection features.

Each of these characteristics will be introduced in the following sections. The detailed methodologies of subversion are discussed in the next chapter.

1. Subversion Over a System Life Cycle

Subversion is not limited to on-site operations, as in the case of deliberate penetration. It includes activities that spread over the entire life cycle of a computer system. This life cycle includes several phases:

1. Design- The beginnings of a system. All key decisions concerning the software and hardware specifications are made during this phase.

2. Implementation- The conversion of the design into a usable product. This includes manufacturing and testing of hardware components, and the coding and testing of software components.

3. Distribution- After all system components have been produced and tested, they are distributed to the various operational sites.

4. Installation- Upon receipt of new system components, these components must be installed and made operational. These components might be new software on old equipment, or old software on new equipment, or any combination of the above.

5. Production- This is the operational phase of the computer system and is the phase that has traditionally

received the most security considerations. This consideration is because of the presence of the sensitive information that is the object of the subverters efforts.

The legitimate activities that are carried on during the various life cycle phases offer ample opportunities for the subverter to undermine system components. The activities in the first four phases are basically not sensitive in nature and are carried out at relatively open facilities. Therefore, the subverter would have little difficulty in subverting the system components under development. Later in the production phase, these same components would be involved in the protection of information. By this phase the subverter would have an 'environment' purposefully constructed for the unauthorized and undetected exploitation of a system and the information it contains. The next chapter will outline possible activities that can be carried on by a subverter during each of these life cycle phases.

2. Skills Required

The subverter, unlike the penetrator, is not an amateur. To be able to carry out subversive operations, the subverter must understand the activities that are performed during the various phases of a computer system's life cycle. But none of these activities are beyond the skill range of the average undergraduate computer science major. In fact,

much of the activity involved with subversion can be carried out by individuals of much less technical knowledge. Subversion can be particularly effective as an organized effort that need only be CONTROLLED by the technically qualified.

The subverter, unlike the penetrator, does not lack the ability to think big. He can utilize a diverse group of individuals that may or may not be aware of the subversive activities they are performing. One need only imagine the vast number of people that will have access to the various computer system components prior to their being delivered to the control of an unsuspecting ADP security officer.

3. The Artifice

The subverter could, and undoubtedly would, use various methods to circumvent the control features of a computer system, including the foible that is indicitive of the penetrators environment. But the subverter is concerned with the long term return on his subversive efforts. To rely on a design oversight or an implementation flaw that might be eventually corrected would not be a sound 'business' practice. Rather the subverter constructs his own mechanisms that are inserted into the hardware or software during one of the various phases of a computer systems life cycle. Any clandestine mechanism that is used in subversion is called an 'artifice' [11]. These mechanisms can be implemented in

35

either hardware or software. The most common forms of artifices are known as trap doors and Trojan horses. A hardware artifice is a particular instance of a trap door.

a. Trap Doors

The key characteristics of a trap door are:

1. It is exercised under the direct control of an activation stimulus.

2. It circumvents the normal control features of a system.

As the name implies, trap doors have a means of activation (like the latch on a door). This activation key is under the direct control of the attacker. A simple example of an activation key is a special sequence of characters that is typed into a terminal. A software trap door program, imbedded in the operating system code, can recognize this key and allow the user of the terminal special privledges. This is done by the software circumventing the normal control features of the system. It is important to realize that the only purpose of a trap door is to 'bypass' internal controls. It is up to the attacker to determine how this circumvention of control can be utilized.

The attacker can construct the trap door in such a manner as to make it virtually undetectable to even suspecting investigators. A penetration tiger team, organized by the Air Force to test the security features of

a computer manufacturers operating system, installed a small trap door that was so undetectable that the manufacturers personnel could not find the clandestine code, even when they were told it existed and how it worked [6].

b. Trojan Horses

A Trojan horse is different from a trap door in several ways. Whereas the trap door is generally constructed to circumvent normal system controls, the Trojan horse can accomplish its malicious tasks without circumventing these controls. Trojan horses are artifices, generally programs, that have two functions:

1. An overt function- This function serves as a lure to attract the program into use by an unsuspecting user.

2. A covert function- This function performs clandestine activities unknown to the user of the Trojan horse.

The overt or 'lure' function of a Trojan horse can, for example, be mathematical library routines, word processing programs, compilers or any program that might be widely used at an installation. Because these programs are executing on behalf of the user they assume all access privileges that the user has. This allows the covert function access to any information that is available to the user.

The covert function is exercised concurrently with the lure function. An example of this kind of artifice might be a text editor program that legitimately performs

editing functions for the unsuspecting user while browsing through his directories looking for interesting files to copy. This is a particularly effective option for the attacker due to the fact that as far as any internal protection mechanism of the computer system is concerned there is no 'illegal' actions in progress. The Trojan horse (e.g., text editor) is simply a user program, executing in user address space, accessing user files, performing perfectly legitimate system service requests such as giving another user (e.g., the subverter) a copy of his files.

D. CHAPTER SUMMARY

This chapter has offered a brief discussion of the three methods that can be used to attack a computer system. They are: indadvertant disclosure, penetration, subversion. There have been important distinctions made between the present conception of the known penetrator and his methods, and that of the subverter and his methods. The known penetrator is basically an amateur that is content to operate within the computing environment as it exists. The penetrators environment is one made of unintentional imperfections that can be used to exploit a system. The subverter, on the other hand, is a professional that actively constructs his subversion environment by the methodical undermining of a computer system throughout its life cycle by the use of

artifices. The next chapter will discuss in greater detail
the methodologies of this subversion.

IV. METHODOLOGIES OF SUBVERSION

To reiterate the definition of subversion, it is the covert and methodical undermining of internal and external security controls over a computer systems lifetime to allow unauthorized and undetected access to system resources and/or information. This chapter describes the methodologies involved in subversion.

It has been the purpose of the previous chapters to 'set the stage' for the discussion that follows. It is obvious that there is not a clear understanding in the computer security arena as to exactly what should be done to insure that computer systems can reliably protect information. As long as this confusion persists subversion will be a threat to the security of computerized information. It should be kept in mind that those who might be involved in subversive activities would not be confused as to what their goals are or how they would accomplish them.

A. GENERAL CONSIDERATIONS

The majority of this chapter is concerned with the activities that an subverter might consider as 'field operations'. These operations involve activities that are required to insert artifices, exercise them, and retrieve

the resultant information. But there are several general considerations that should be kept in mind when reading about the various phases of subversion. Principal among these is that any reference to the subverter is meant as a reference to the subversive organization. Individuals who might perform subversive acts would do so with the guidance of all the expertise that might be available in this organization.

1. Safe Computing Sites

Like any effective field operation, the subverter needs to insure that any techniques and mechanisms used in the field have been perfected at a safe computing site. This might seem difficult if a new system is the subversive target. However, there are machines available today that are micro-programmable emulators such as the Burroughs D Machine or the Nondata QM-1. A Feasibility Study [15] has demonstrated that a very sophiphisticated, large scale computer system (Multics) could be emulated on such a device. Because these machines are micro-programmable, one machine can be used to support several field operations.

Once a basic architecture is emulated, existing operating systems and subsystems could be installed. These systems could then be analyzed for exploitable foibles, and artifices could be designed and tested. The basic algorithms for software artifices can be refined in a safe atmosphere

to insure that there are no unwanted side effects. Sound software engineering practices would be employed to analyze the best approach to the subversion process.

2. Scope of Operations

The scope of subversion is completely under the control of the subverter. It can be as focused as one computing site or as widespread as several hundred installations, all with roughly the same expenditure of effort. This is accomplished by selecting the phase of a computer systems life cycle in which to start subversion operations [10]. The earlier in the life cycle a system has been subverted, the more global the opportunities for exploitation.

By installing artifices at the beginning phases of the life cycle (design or implementation) they will then become an integral part of the computer system. Anyone who subsequently procures one of these systems will become a potential target for exploitation. Identification of the victims need not occur until later. Should the subverter not have the opportunity to begin his operations in these first life cycle phases, he would have ample opportunities in the later phases.

The subverter can narrow the scope of his operations by performing his malicious activities during the distribution of system components to the selected sites. He can select

which sites are the most profitable and then intercept system components as necessary to accomplish his goals.

Finally, by initiating subversion operations during the installation or production phase of a computer system, he restricts his activities to that particular site.

3. Desirable Traits in Artifices

The following discussion will center on the three major types of artifices; software trap doors, Trojan horses, and hardware mechanisms. Not only are the below listed traits desirable, but they are qualities that can be easily incorporated into artifice construction.

a. Software Trap Doors

Recall that the principal function of a trap door is to circumvent internal system controls under the control of an activation key. With this in mind, the following are several desirable traits that the subverter would incorporate in the implementation of this type of artifice.

(1) <u>Compactness</u>. To give the user of the trap door unauthorized privileges may involve only enough code to recognize the activation trigger and the one or two instructions required to change the machine state to master mode. The fewer the instructions the better. Once this is accomplished, other programs can be invoked to perform the desired clandestine activities.

(2) Revision independence. To insure that a trap door remains in the system for years, perhaps its entire life, it is necessary to install it in an area of code that will not be liable to revision. Operating system software, as pointed out earlier, is often riddled with design errors or subject to planned changes. Placement of the trap door should be in an area that is not likely to undergo review. For example, I/O routines that are used to control hardware devices are not generally changed in software revisions. These are generally written in lower level languages for efficiency and offer an excellent 'refuge' for artifices.

(3) Installation independence. Many 'off the shelf' general purpose computer systems come with a wide range of options. But for a given family of systems, there is usually a 'core' operating system that will be common to any installation within the system family. By installing the trap door in this 'core' of code the subverter is assured that his artifice will be present in the system regardless of the particular configuration that would be generated at the installation.

(4) Untracable. The operation of the trap door should not in itself leave any trace of its operation. This implies that either its operation does not encounter system traps or audit trails, or it has the ability to erase any evidence of its activities. Frequently, the very 'primitive' or basic functions of an operating system, such as a

44

teletype stream handler, are at too low a level to be audited in system logs. These routines are also relatively 'stable' in that they are generally not subject to frequent revision.

(5) Uniquely Triggerable. The means by which the trap door is activated should be unique enough to insure that accidental activation is unlikely. One example is a trap door that is triggered by a unique sequence of characters in a teletype stream. Too short a sequence or too common a sequence might accidentally activate the artifice by someone other than the subverter or his agent. On the other hand, to long a sequence might require to much code to check against and make the trap door code too long.

(6) Adaptibility. The trap door should have a degree of generality or even programability. Since the trap door might have been installed during the early phases of the systems life cycle, the subverter cannot always predict the particularities of the installation or application. For instance, since trap doors circumvent normal controls, it could be designed to modify operating system code online. By circumventing the write protection of the operating system code area the trap door can allow the subverter to adapt the operating system to his needs.

b. Trojan Horses

As previously stated, a Trojan horse is a program that is invoked by an unsuspecting user. It will perform a

legitimate function (the lure) and a covert function. The following are a few desirable traits for this artifice.

(1) <u>Directed Lure</u>. The lure (or overt) function of the Trojan horse will determine what kind of information will come under the scrutinization of the covert function. If the desired information is scientific in nature then it might seem plausible to construct a Trojan horse that offers a lure of some sort of mathematical computation. If personnel records are the target then the lure might be a sort routine. It should be noted that the information available to the Trojan horse is any information that would be normally be available to the unsuspecting user. Not just the information needed to perform the lure function. This is because most operating systems consider any program executed by a user to be 'owned' by that user for the duration of the program execution. Any access rights that the user might have are imparted to programs run on his behalf.

(2) <u>Compatibility of Functions</u>. The covert and overt functions of a Trojan horse should perform 'expected' actions. It is not expected that a mathematical library routine would access the users file space (e.g., the covert function browsing through files) when it is computing the roots of a polynomial. System audit logs may record this activity and suspicions be aroused. This could be disasterous if the covert function was to inadvertently cause the user process to be interrupted by a disk error.

However it is expected that a sort file routine will access the users file space. Subsequent disk errors might be overlooked as merely a fluke. This can be viewed as way to 'functionally disguise' the Trojan horse.

c. Hardware Mechanisms

A Hardware mechanism is a special instance of a trap door. It performs the same function of circumventing normal system controls as its software counterpart. Its capabilities and traits are essentially the same. The method of activation may vary due to the unique hardware capabilities such as the ability to transceive radio signals. There are two cases of hardware mechanisms, programmable and non-programmable. Examples of each of these types are presented later in the chapter.

4. Obscuring Artifices

Proper obscuring can make artifices virtually undetectable. One must realize that once code or hardware is operational in a computer system there would be no reason to review it unless something failed. Think of how hard it is to find a difficult bug that is being purposefully searched for in a program. One can imagine how difficult a small trap door would be to find if the author of the trap door takes special pains to obscure it. Furthermore, even if found, the well-designed artifice will appear to be just another bug. Obscuring artifices is considered essential to the

subversion process. Obscuring techniques are limited only by the ability and understanding of the subverter installing the artifice.

Listed below are a few techniques that the subverter might use in this process.

a. Modifying Object code

Binary machine code is the most obscure medium in which a software artifice can reside. The Multics Security Evaluation [10] amplifies this point:

Clearly when a trap door is inserted, it must be well hidden to avoid detection by system maintenance personnel. Trap doors can best be hidden in changes to the binary code of a compiled routine. Such a change is completely invisible on system listings and can be detected only by comparing bit by bit the object code and the compiler listing.

Disadvantages of this obscuring method come about because object modules may be periodically recompiled for various reasons [10]. This, of course, may not be under the control of the subverter and methods must be devised to insure periodic reinsertion. It has been informally reported [10] that a compiler could be 'enhanced' to always reinsert an artifice in the object code when a particular system module was recompiled. Compilers themselves are rarely recompiled by the user. So the clandestine code that was located in the compiler would be quite safe.

Obscuring in object code is particularly suited for Trojan horses. Software that is procured from vendors as

'off the shelf' computing aids often do not provide source code listings. This is to protect proprietary rights. The subverter (perhaps a legitimate vendor) can use this fact to his advantage. He could offer software products to unsuspecting computer installations much as any other software vendor might. In fact, the subverter could anticipate the installations needs if he had agents on the premises that knew the particular situation. Since the subverter is not primarily in the business of making money by selling software, he can undercut competitive bids.

Detection risks for this obscuring method are considered relatively low. Even if the Trojan horse were to malfunction and lead system maintenance personnel to suspect it of 'performing strangley', without source code documentation the first order of business would be to contact the vendor for another copy of the program.

b. Abusing of Software Engineering Practices

When using source code as a means of inserting artifices, means must be devised to obscure the true purpose of the clandestine code. Program documentation could prove invaluable in this effort. Good program documentation is essential to the understanding of complex programs such as operating system software. Most higher level languages allow variable names of ample length. Yet many programmers are content to follow archaic FORTRAN or assembler-like practices that tend toward short, abreviated variable names

that have meaning only to the programmer at the time he wrote the code. Inadequate commenting of source code is another common abuse.

Writing programs that are unstructured or non-modular in organization can prove quite effective for obscuring. This is commonly refered to as 'spaghetti bowl' logic. By using non-local 'goto' statements that seem to jump around the program arbitrarily, it is virtually impossible to follow the program logic.

Allegedly 'good' documentation practices can also be utilized in the obscuring process. This technique can simply be labeled as lying. Plenty of apparently good comments can lure the reader away from scrutinizing the code too closely. Mislabeled variables can also steer the reader away from the actual purpose of the clandestine code.

The use of source code as a mean of inserting artifices has the dual distinction of offering the subverter the greatest returns as well as the greatest risk of detection. Source code artifices will not be destroyed by recompilation of the code as some other methods of insertion. However because it is in human readable form, artifices are more visible and therefore more vulnerable to possible detection [10].

c. Using Assembler languages

Most assembler language traits both good and bad are benifical from the subversion standpoint. Some of these traits are:

1. Most 'powerful' language available.

2. Most efficient in execution time and core requirements.

3. Least comprehensible of all the human interpretable computer languages.

Assembler languages are the most 'powerful' because they allow greater control over the programming environment than any other language. Assembler languages are not constrained to the addressing restrictions that are imposed by the structured environments of the higher level languages. There is no distinction between data and code areas. This allows the subverter to either write self modifying code or obscure clandestine code as data. Assembler programs are noted for their 'spagetti bowl' logic because it is difficult to write assembler programs that do not use goto statements. Since goto statements are expected in assembler code, it is easy for a subverter to write a program that has a goto statement whose operand is a variable label rather than a statement label. The variable label could define the begining of a series of hexadecimal or binary constants that are nothing more than the equivilent binary opcodes of the clandestine routine. Close

scrutiny is rarely given to these 'tables' of constants, particularly if the program is functioning properly.

Assembler language source code is assembled to machine code instructions on an almost one-to-one basis. Therefore the subverter can exactly predict the amount of 'overhead' the artifice will impart to the subverted system.

d. Strategic Placement

Obscuring software artifices, particularly trap doors can be greatly enhanced by strategically placing the clandestine code away from areas that might be subject to investigation. For example, consider a trap door that is triggered by an activation key from a teletype. Perhaps security investigators suspect that a trap door exists and that it is activated by a teletype stream. Naturally the investigation would inspect all code that handles the teletype stream. The subverter can foil these efforts by placing the trap door in an area totally unrelated to the teletype routines, such as the disk I/O driver. Since the trap door resides in a routine that executes in the master mode, addressing restrictions do not apply, and the teletype buffer is addressable from the trap door's vantage point.

The subverter can either wait for normal disk useage or execute a 'do nothing' program that uses the disk. This will insure that the trap door that resides in the disk driver routine will be exercised at the same time the activiation key is present in the teletype buffer area. Upon

recognizing the activation key the trap door will perform the necessary task required to circumvent the normal controls.

e. Using Confinement Channels

Confinement channel is the general term applied to information paths that can exist between a program (called a service) and its owner. The information is gained when another program (called a customer) invokes the service and the service subsequently extracts unauthorized information from the customer and passes it to the owner of the service [16].

Much of the computer security evaluation criteria [8] mentioned in Chapter II is concerned with what is called the simple security condition. This condition states that a subject (user or his program) cannot have read access to objects for which he is not cleared. Confinement channels generally meet this condition. However they do not meet what is called the confinement property (also known as the *-property). The confinement property states that if one program has read access to data at one security level it cannot have write access to another file at a lower security level [21]. Thus the program is 'confined' to not, in effect 'declassify' information, but it is confined to write into a file of the same security level or higher.

Most systems do not even consider the issues of confinement. If an artifice was to introduce such a channel

it would probably not be recognized for what it was. One type of this channel is sometimes called a covert channel. This channel is called covert because the method by which the information is passed is particularly difficult to detect. An example is offered by Denning [14]:

> One type of flow cannot be controlled easily, if at all. A program can convey information to an observer by encoding it into some physical phenomenon without storing it into the memory of the computer. These are called flows on covert channels... A simple covert channel is the running time of a program.

Because these channels for information flow are not the 'normal' paths that information are thought to flow on (i.e., variable parameters, files and other 'storage channels') they are easily overlooked by investigators. In the simple example above Denning [14] explains how the running time of the program can be used to convey information:

> A program might read a confidental value, then enter a loop that repeatedly subtracts 1 from the value until it reaches zero. The owner can determine the confidental value by simply observing the running time.

Confinement channels will be discussed again in later sections of the chapter.

f. Hardware Obscuring

Today integrated circuit technology offers a near perfect medium in which to obscure hardware mechanisms. Equipments that have medium scale integration (MSI) chips can be replaced with enhanced large scale integration (LSI)

chips. The enhanced chips would perform the required functions of the original chips, but also perform functions under the control of the subverter. Detection of these devices, once installed in target equipment is virtually impossible, since the subverter would undoubtedly insure that all external appearances such as physical appearance, logical operation, power comsumption, etc., would be the same. There is no non-destructive way to thoroughly examine these devices.

B. INSERTING ARTIFICES OVER THE LIFE CYCLE OF A COMPUTER SYSTEM

The subverter by inserting artifices into a computer system is, in effect, 'creating' a subversion environment on the targeted computer system. He is inserting the 'tools' which he will use to undermine the security of a computer system. Once this security is subverted, he can then extract the information he desires. But the timeframe between when the artifice is inserted and when information is retrieved may be years.

He can be very successful in his insertion efforts because the places in which the subversion occurs, are relatively open environments that are not hardened against his efforts. This is because there maybe no classified operations being conducted at many of the places the subversion occurs.

There is an interesting property in the insertion activity that differs from most other forms of criminal activity. The subverter is not removing or stealing anything from the premises, on the contrary, he is introducing 'a little something extra'.

1. Design Phase

The subversion of a computer system design is a subtle process. As in any design process there are hundreds of alternatives to consider. Among the many choices on any given issue, several may prove acceptable. It is the job of the subverter to be the 'standard bearer' of those alternatives that will aid him in his subversion efforts.

Inadequate design choices have been used in the past to exploit a system. In 1974 the Naval Research Laboratory conducted a penetration exercise on a Univac 1108 system running under Exec VIII. The author of the resulting report [17] comments:

> However, even if an MLS (multilevel security system) is completely bug-free, in the sense that its response to user requests is completely specified by its design, this does not imply that the MLS will not permit dissemination of data to unauthorized users. Our penetration of Exec VIII is not based on bugs in the implementation, though they certainly exist. Instead, we exploit several aspects of the Exec VIII design philosophy which, when taken together, make penetration possible.

Details of this particular penetration exercise are outlined later in the chapter.

The following is a brief discussion of how the subverter might make seemingly sound design choices and still subvert a systems design.

a. Operating System Software

(1) Password procedures. There are several ways to design password login procedures. Three viable choices that the subverter might propose are:

1. encrypt the passwords with a seemingly non-invertable algorithm

2. allow the user to choose his own passwords

3. allow multiple login attempts for the 'forgetful' user.

The first case was used on the Multics system at the time of the USAF security evaluation [18]. The designers of the system hoped that the algorithm they were using was non-invertable, the evaluation demonstrated that it was not.

In the second case, user chosen passwords are often easy to guess [10]. One such system allowed the user to choose his own password. The system administrators would enter a new user into the password file and as a convenience, would enter the users name as his password until the users first session, at which time the user was supposed to change the password to one of his own choosing. Due to a design choice, the password file was readable by

57

all users. This in itself was not a cause for alarm, as the password field is encrypted. But the first entry in the file is the user's name in plain text. A malicious user, knowing the administrators procedure, attempted the login sequence using the names in the password file until there was a sucessful login (presumably from a new user). Subsequent investigations revealed that many of the users had not ever bothered to change their passwords. This also points out the problem of allowing too many login attempts.

(2) Audit Procedures. Two design suggestions that a subverter might recommend are:

1. audit all actions that might be security related (the more the better), or

2. audit only user mode actions.

The subverter by recommending excessive auditing will, in effect, render the auditing process ineffective. Those that are tasked with the manual reviewing of audit logs will be quickly buried by the sheer volume of it all. The listings will quickly fall into disuse in the corner of some storeroom.

By auditing only user actions the subverter is given free 'license' to implant his artifices in master mode routines that are 'trusted'. The subverter need not worry about any actions carried out by artifices that exist in master mode routines because their actions will not be traced by any audit mechanism. If a trap door circumvents

control of the system by placing the subverter in master mode then any subsequent actions of the subverter will not be audited.

(3) <u>Confinement Channels.</u> Some areas of the computer system could be designed to pass information via a confinement channel. Should the subverter find himself working in one of these areas he would undoubtedly take advantage of the opportunity. The concept can be best illustrated using an example.

Many operating system designs are process oriented. Each time a new process is required by the system, a unique identifier is assigned to this process so the system can keep track of all the different processes. There appears to be nothing significant about the process-id. Therefore it would seem irrelevant as to how this unique identifier is selected. Logically the easiest choice would seem to be to assign process-id numbers sequentially as they are needed. By making this design choice the subverter has constructed a confinement channel.

Assume there are two processes, 'A' and 'B', active in a system at the same time. Process 'A' is a clandestine service routine (with a Trojan horse) that has access to sensitive information. Process 'A' desires to communicate some of this sensitive information to process 'B', that is not authorized access to the information. They will communicate by using the process-id number as a binary

communication channel. Because process-id mumbers are assigned sequentially, process 'B' can deduce information from the id number based on the previous values. If 'A' desires to send a binary '1', 'A' will create two new dummy processes (and immediately destroy them). This will increase the Process-id number by two. If 'A' desires to send a binary '0', it will create and destroy one process.

On the receiving end, 'B' will create one process and save the id-number and then destroy the process. 'B' will compare the new process-id with the one saved from its last active period and compare the two. If it is three greater than the previous process-id the information sent was a '1', if it was two greater it was a '0'. Because both 'A' and 'B' are executing on the same machine, these activities are not occuring at the same exact time and they are synchronized (in a crude sense). Because there will be other processes in the system creating new process-id numbers, the channel will be 'noisy'. But modern information theory can be applied to detect transmission errors and reliable results can be obtained [16].

(4) Backward compatible features. Manufacturers must insure that new product lines are backward compatible if they wish to upgrade old customers. The subverter can capitalize on these design requirements by insuring that older system foibles are carried along to the new systems design. The IBM Systems Journal [19] offers an example:

60

Two VM/370 features were discovered that permitted a total penetration, and others were discovered that could cause the system to fail. The first case concerned the OS/360 use of self modifying channel programs in its ISAM access method. To support this feature in a virtual machine, VM/370 had been modified to examine channel programs for the pattern associated with the use of self modifying code by OS/360. the VM/370 method of handling such channel programs was to execute some commands out of the users virtual storage, that is, not in VM/370 virtual storage space. As a consequence, a penetrator, mimicking the OS/360 channel program, could modify the commands in his storage before they were executed by the channel, and, thereby, overwrite arbitrary portions of VM/370.

b. Other Software Design Choices

Most computer systems are offered with a suit of supporting software such as compilers, text editors, service routines, etc. These can provide the subverter opportunities to incorporate Trojan horses into the overall system design. Software that is supplied as part of a package deal is financially attractive to customers that would have to otherwise procure these items from other sources. Many times for efficiency or convienence, a service like a compiler will have special privileges (like executing in master mode for some fuctions). Thus a trap door in this program is as effective as one in the operating system itself.

Service routines that are designed for benign purposes can be used by the subverter to insert artifices. IBM/360 offered one such service [20]:

The means for inserting a penetration mechanism into an existing program (either system or user) stored on a direct access device is provided by one of the Operating System/360's own Service Aid programs, IMASZAP.

This program is designed to modify data and instructions at any given location on a direct access file, which is to say, one can modify information anywhere on a disk pack.

c. Hardware Alternatives

The selection of hardware for computer systems will also offer the subverter many opportunities to aid his cause. The subverter can concentrate on central processors, peripheral equipments, or both.

(1) Central Processors. The selection of central processors from the subverter's point of view is straightforward. The simpler the architecture the less effort that will be required to subvert it. Optimally the best choice is an architecture with no hardware protection mechanisms. But this this choice is an impractical one for both the subverter as well as the customer. There would be little chance that such an architecture would be considered for use in a system handling sensitive information, and the subversion effort would be for naught. The subverter must work within at least minimum guidelines.

For example, one set of minimal guidelines can be found in The ADP Security Manual [8]. This list of mechanisms is extensive. One would think that such a complete list is sufficient to assure a secure system. However, many of the penetrated systems in chapter two had these features and penetrators were very successful in there efforts. It is important to realize that having these features is not sufficient for a secure condition, it is how

effectively they are employed. It is the job of the subverter to ensure that they are not effective even if they are present. The following is from the ADP Security Manual [8]:

4-200 Hardware Features.

a. The execution state of a processor sould include one or more variables, i.e., "protection state variables," which determine the interpretation of instructions executed by the processor......

b. The ability of a processor to access locations in memory (hereafter to include primary and auxiliary memory) should be controlled (e.g., in user mode, a memory access control register might allow access only to memory locations allocated to the user by the O/S).

c. The operation of certain instructions should depend on the protection state of the processor. For example, instructions which perform input or output operations would execute only when in master mode. Any attempt to execute an instruction which is not authorized should result in a hardware interrupt.....

d. All possible operation codes, with all possible tags or modifiers, whether legal or not, should produce known responses by the computer.

e. All registers should be capable of protecting their contents by error detection or redundancy checks......

f. Any register which can be loaded by the operating system should also be storable, so as to permit the O/S to check its current contents against its presumed contents....

g. Error detection should be performed on each fetch cycle of an instruction and its operant (e.g., parity check and address bounds check).

h. Error detection (e.g., parity checks) and memory bounds checking should be performed on transfers of data between memory and storage devices or terminals.

i. Automatic programmed interrupt should function to control system and operator malfunction.

j. The identity of remote terminals for input or output should be a feature of hardware in combination with the operating system.

k. Read, write, and execute access rights of the user should be verified on each fetch cycle of an instruction and its operant.

These requirements as outlined in the Security Manual are general enough so that viable arguments can be constructed to demonstrate most major vendor's processors 'acceptable'. A way in which the subverter could meet the letter of these requirements and still defeat the protection mechanisms was demonstrated in the Multics Security Evaluation [10].

The vulnerability involved violation of requirement 'k' listed above (access on each fetch). The Security Manual states that each instruction must produce known results (requirement 'd'), but this vulnerability involved a SEQUENCE of instructions. The Multics Security Evaluation [10] outines the method:

> This vulnerability occured when the execute instruction was in certain restricted locations of a segment with at least read-execute (re) permission. (see figure 1) The execute instruction then referenced an object instruction in word zero of a second segment with at least R (read) permission. The object instruction indirected through an ITS pointer in the first segment to access a word for reading or writing in a third segment. If all these conditions were met precisely, the access control fields in the SDW (segment descriptor word) of the third segment would be ignored and the object instruction permitted to complete without access checks.

This particular hardware 'bug' resulted from a field installed design change to the equipment that was installed

at all the computing sites. A subverter might well include such 'features' in the initial hardware design.

ENTER   RE ACCESS

          R ACCESS   NULL ACCESS
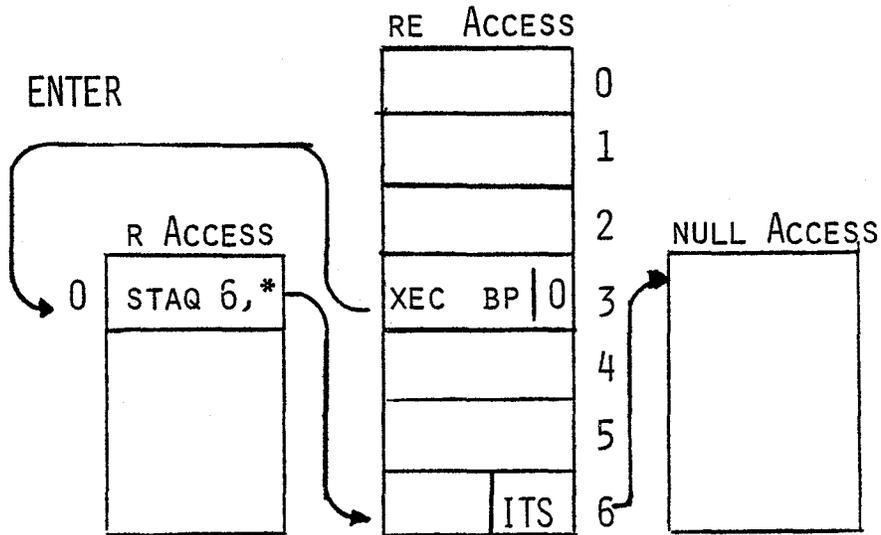
FIGURE 1. EXECUTE INSTRUCTION BYPASS

(2) Peripherals. Generally, peripherals do not have the stringent requirements placed on their internal behavior like central processors. They are generally thought of as being under the control of the central processor and if the CPU is 'contained' (in a security sense) then the peripherals will follow. This concept is rapidly changing in todays technology. Many devices such as direct memory access (DMA) I/O equipments are specialized processors in their own right.

Configuring a system so that 'specially modified' I/O devices can intercept (or directly access) sensitive information is totally within the realm of the subversive designer. Likewise, procurement policies that are based on the lowest bidder can (and have been known to) result in a composite system that comes from a variety of manufacturers. A subversive designer can specify equipments to such a degree that only one vendor (the subverter) will be able to meet the specification. By specifying in this manner or by competitive pricing these 'enhanced' equipments can find their way into a 'secure' computer system.

2. Implementation Phase

In this phase of a computer systems life cycle, there are two computer systems to consider. There is the computer system under development, and there is the computer system used for the development (i.e. the 'host' computer). The subverter would first penetrate the host computer. Once this is accomplished, he would have access to the new software under development. This technique was demonstrated during the Multics evaluation [10]. A trap door was inserted into a new version of the Multics software that was to be distributed to all Multics sites.

The target, of course, would be the new software (or hardware) under development. It would be these new products

that would be employed in the protection of information in the future.

Inserting artifices during the implementation phase can offer as many advantages as inserting during design. In fact, there are additional advantages because inserting artifices during implementation of a system does not require the subverter to be on the vendors payroll.

Often programmers can work from their homes on remote dialup terminals. Because these vendor development systems are not hardened against wiretapping or other possible penetration techniques, the subverter can infiltrate as desired. Private corporations would tend to shy away from particularly restrictive security practices when there is no classified activities present. The Multics Security Evaluation [10] which was written in 1974 pointed out such an environment:

> ... it should be noted that the software for WWMCCS (World Wide Military Command and Control System) is currently developed using uncleared personnel on a relatively open time sharing system at Honeywell's plant in Phoenix, Arizona. The software is monitored and distributed from an open time sharing system at the Joint Technical Support Agency (JTSA) at Reston, Virginia. Both of these sites are potentially vulnerable to penetration and trap door insertion.

Two areas of activity that might be subject to subversion in the implementation phase are, coding and testing, and hardware assembly and checkout.

a. Coding and Testing

Coding and testing of system software is concerned with one major goal: that the programs perform at least the required functions. This is a minimal requirement, not a maximal one. Testing criteria involves only insuring that a given module performs the required tasks correctly. It does not involve the concept of determining all the functions that it might be able to perform. In general, this characteristic cannot be determined for a program since this reduces to the unsolvable safety problem [4] discussed earlier.

If subversive activities are to be carried out by the actual programmers assigned to the project, there are a few general practices that the subverter might follow. One such practice is using global or external attributes for variables that might not otherwise require them. This can make data available to other covert routines that will be able to utilize them. This is common practice in operating system programming, particularly if the language used is assembler language.

Some languages, particularly higher level languages that are constructed for operating system use, do not perform run time bounds checking on data structures that use subscripting or pointers. This is not done because the extra code required cannot be afforded in an operating system environment. Effective use of such structures can

allow clandestine routines access to areas that would be otherwise inaccessable. For instance, a routine that has a trap door installed performs some processing on an array that is passed to it. The maximum expected size might be 100 elements. If there is no runtime subscript bounds checking, the routine could check the area just beyond the 100th element for a unique bit pattern that would activate the trap door. Specific features such as hardware bounds checking mechanisms will not help much because there would be no violation of the jobs total address space.

b. Hardware Assembly and Checkout

The safest time to carry out subversion activities on hardware is during the assembly of the equipment. Insertion costs and detection risks would be low during this period. Equipment could be assembled with specially enhanced integrated circuits that appear and function exactly like the normal circuits. This could be done by intercepting the suppliers shipment of parts to the assembly plant and replacing them with the subverted hardware. This way the subverter would be totally removed from the insertion process. Entire product lines can be equiped with these hardware trap doors. If shipments could not be intercepted, or if the assembly plant was the manufacturing facility as well, other arrangements could be made. Assembly line personnel could replace the normal chips in the assembly line parts bins with the enhanced chips.

Plant security is typically oriented toward individuals
taking products out of a plant, not bringing them into it.

3. Distribution Phase

The most significant advantage to inserting artifices
in system components (hardware and software) during the
distribution phase is that the subversion occurs after the
review process is completed. These components already carry
the 'seal of approval' and will, in all probability, not be
subjected to close scrutiny again.

Subversion activities carried out during the
distribution phase require significantly less investment in
technical talent than than other phases of the life cycle.
Activities involve the replacement or modification of valid
equipments and software with subverted copies. Personnel who
might be involved are delivery truck drivers, mailmen,
receiving or shipping clerks. Most of these personnel can
perform their aspect of the subversion and not be aware of
the 'big picture'. Even if apprehended and interrogated
their knowledge of the extent of the operation would be
minimal.

Suppose that the subversive organization
legitimately purchased several terminals from a company.
Upon receiving these terminals they are carefully unpacked
so as to not damage the orginal shipping containers.
Technicans could then modify the terminals with special

'enhancements' and insure that they perform as desired. The terminals are then carefully repacked so that nothing would appear disturbed. When the subverters received word that company XYZ had ordered some of these same terminals for a new multilevel security application, they could be replaced for the normal terminals. This way the subverters have a steady supply of terminals, with only the initial investment.

There are various methods that could be employed to substitute the enhanced terminals for the normal ones. It might require the services of a slightly dishonest truck driver or warehouse clerk.

The important point is that the terminals would not be suspected because they were not 'stolen' in the classical sense of the term, just replaced with 'enhanced' versions. The shipping papers could be changed to reflect the different numbers if serial numbers could not be changed.

In other areas, the process might even be easier. Companys often put out advance notice of upcoming software revisions, or hardware field changes. Subverters could be alert to these things and be ready with enhanced revisions or field changes. On a software revision the subverter could conceivably intercept a software revision tape and modify (or replace it) within hours. The delay would be negligible.

Another method that can be used is for the subverter to generate bogus software revisions or field changes to be

71

carried out by system maintenance personnel. These changes can be forwarded with forged stationary and customers would have no reason to suspect that the changes are bogus [10].

4. Installation Phase

The installation of any computer system is a rather chaotic period. The subverter can capitalize on this chaos and use it to his advantage.

There are several opportunities to install software artifices during the initial installation of a new system, particularly a new operating system. Several bugs are bound to surface and the system may require numerous regenerations of code to test out all the changes required by the tailoring of the system to the particular installation.

Systems programmers will be uncertain about the new systems behavior patterns. In such an uncertain environment security personnel will naturally not allow sensitve information to be processed, and in fact might allow the system to be run under less control than would otherwise be present. It is doubtful that a malicious systems programmer would be scrutinized very closely and he could insert many trap doors into the new system.

Many decisions are made during these initial break in periods concerning operational procedures that the subverter can offer his 'advice' on. Each installation is different and requires judgement calls on the particular

situation at hand. A highly technical subverter (such as the vendors representative) can prove suprisingly effective in this kind of situation.

An interesting method for inserting trap doors that can be implemented during the installation phase is suggested in the Multics Security Evaluation [10].

Here, the system initialization code is modified by the penetrator to insert other trap doors as the system is brought up. Such trap doors can be relatively invulnerable to detection and recompilation, because system initialization is usually a very complex and poorly understood procedure.

5. Production Phase

Inserting artifices during the production phase of a systems life cycle may entail more risk than inserting during the other phases. All security measures will be in place due to the presence of sensitive information. But these risks are only high in comparision to inserting during the other life cycle phases, and in an absolute sense can be quite acceptable. Recall that the commom 'computer criminal' or penetrator works exclusively in this penetration environment and has had excellent results. Techniques used by the subverter to install artifices in the production phase of a system are the same techniques used by the penetrator to generally exploit a system, i.e., system foibles.

One could argue that it seems senseless to use an unintentional trap door (a foible) to install an intentional trap door (an artifice). But one must remember that the subverter is not out for the 'quick dollar'. He is a professional that is in the business of gathering information over a long period of time. The subverter will certainly use any device at his disposal, but the deliberate, well thought out, and tested artifice can insure results over the long haul, with a minimum of risk. The artifice will continue to work even if the orginal foible is found and corrected.

It is instructive to examine how one might insert clandestine code in a system when it is in an operational or production mode. The example choosen is the Univac 1108 penetration exercise. The success of the exercise was due to two design foibles [17]:

1. Inadequate error recovery. For any given job the user had the ability to request the control of error recovery. In general an error routine in the Exec VIII operating system had access to the same addressing environment as the routine causing the error. Exec VIII did not stack error handling routine requests, but deleted the previous request.

2. Unprotected reentrant routines. Shareable non-executive reentrant routines in Exec VIII are called reentrant processors (REP). Examples of these are

compilers, text editors, data management subsystems, etc. Each REP must have an associated data area that is writable. Due to a hardware design oversight, write protection is provided for BOTH instruction and data banks or for neither. For the REP to be able to modify its associated data bank the code area must run unprotected from modification.

Due to Exec VIII core allocation policies, there was usually a number of unused words at the end of the last core block allocated to the REP code area. The sequence of events was as follows [17]:

1. A legitimate program called BREAKER requests to handle its own error recovery.

2. The BREAKER program prepared an out-of-bounds data bank for the victim REP and linked to it.

3. BREAKER invoked the victim REP and the REP immediately caused a guard mode error while trying to access its data bank.

4. Control was immediately returned to the BREAKER routine via the error handling request. BREAKER then had write access to the victim REP.

5. BREAKER checked the end of the victim REP to see if there were enough free words in the code block to insert a calling sequence to a clandestine routine. If there was, the entry point of the REP was changed to a jump to the beginning of the free area and a calling sequence was inserted in the free area.

Using this method a subverter could essentially build a general purpose Trojan horse that could be used in various ways. Depending on the purpose of the clandestine program invoked by the calling sequence, the subverter could:

1. access information owned by any user who subsequently invokes the victim REP.

2. install trap doors in programs owned by users of the victim REP, such as the operating system.

6. Summary

The insertion phase is the most significant aspect of the subversion process. The efforts that go into this phase yield 'tools' that will give the subverter access to information almost as easily as the owner of the information. Whereas, the subverter has constructed a sound foundation from which to work, he has left the legitimate user one of sand.

C. EXERCISING ARTIFICES

The discussion up to this point has centered on the subverter creating the subversion environment. Attention will now turn to how the subverter can use this environment to exploit a computer system. There are several activities that can be carried out by the subverter after he has activated the artifice [11];

1. extraction-- the withdrawal or copying of data
2. alteration-- changing or modification of data, programs or hardware,
3. addition---- adding extraneous data
4. utilization- using the system resouces malicously.

All these activities are possible objectives of the subverter. Before these activities are discussed it is instructive to first understand how the artifices that will enable these activities are triggered.

1. Activating Artifices

   a. Software Activation

   (1) Trojan Horses. Trojan horses are usually activated by the victim program . Although the mechanism is considered activated that does not imply that the covert function of the Trojan horse will necessarily do anything malicious. Due the the possible wide usage that a Trojan horse can get, the subverter may desire to limit the information that it gathers.

A text editor can be enhanced to check the
file name of those files it is employed to edit and based on
a predetermined target the Trojan horse will respond
accordingly. The target might be the system password file.
When the editor senses this file it will copy the file to a
safe place, otherwise it will lay dormant. A safe place is
any area that is accessable to the subverter. This may be a
file in the subverters own directory or a system buffer area
that is accessable via a clandestine routine.

(2) Trap doors. Should the subverter require
close control over when an artifice is activated, it might
require an agent to input the trigger via a terminal or by
submission of a batch job. The activator need not be aware
of .what clandestine activities are in progress. For
instance, suppose a trap door was inserted in a system
during the implementation phase of the systems life cycle.
The subverter knew exactly what tasks needed to be performed
but not when. Remember that the insertion may have taken
place years prior to the time of its activation. Imagine the
following scenario.

A janitor is in the process of cleaning a
room that contains a terminal. Like many installations the
system runs 24 hours a day. The janitor has received
instructions to turn on the terminal and type in a given
string of characters. He then proceeds with his cleaning
chores. At the end of a predetermined time the janitor

switches off the terminal, and proceeds as though nothing had happened. The trap door was programmed to periodically check the teletype buffer for the predetermined pattern, perform its clandestine function and then erase all traces of its actions.

Another method of activation for trap doors is by timer. If a subverter is aware that some valuable information will be input into the system after a certain date, he can install a trap door that will periodically check the system clock for a certain date. Upon recognizing that the date has occurred the trap door will copy the information to a safe area for later retrieval. Variations on this theme have been informally reported within the Department of Defense. These artifices were implanted by disgruntled employees. The results of these implantations can be disasterous. It could mean the voiding of thousands of dollars worth of software because there is no way to find the malicious code and the risk could too great. If such a mechanism was installed in something like automated process control software, thousands of dollars worth of damage could result.

b. Hardware Activation

Methods for activating hardware artifices will vary with the sophistication of the mechanism. The following are a few examples:

1. An enhanced chip that is part of a teletype terminal is activated by the systems login sequence. Upon recognizing the sequence, the chip will store the users name and password in the chips own memory area.

2. An 'intelligent' chip such as a special purpose microprocessor that can be microprogrammed by the data stream that follows the trigger. This mechanism could reside in peripheral equipment and be used to selectively copy data to other storage devices on command.

3. A central processor that has been 'modified' to disable memory checking mechanisms or place the processor in master mode when a special sequence of unused opcodes is executed. The opcodes when executed in any other order will have no effect on the processor. There would be another special sequence of code that would restore the processor to normal operation.

2. Techniques of Exploitation

After the artifices have been activated there are several activities in which the subverter can engage. Below is a brief discussion of some of the possibilites.

a. Breaking Out of a Subsystem

As pointed out earlier, subsystems are built around an underlying operating system. This subsystem will use the primitive operations of the operating system to

construct the restricted environment that the user will see. To the operating system (and the subverter) the subsystem is nothing more than another program running concurrently on the system.

Assume a subsystem is designed to restrict the user to performing simple calcualator functions. That is, the user can type simple mathematical expressions at the terminal and the answer will be typed in reply. Any input other than a valid expression will result in the subsystem replying with the message 'invalid expression, try again'. This is clearly a restricted environment. The user does not have the ability to execute programs, or use any of the other sevices offered unrestricted users.

But if the underlying operating system had been subjected to subversion, the subsystem could be easily bypassed by the user. The method that can be used is similar to the trap door used by the janitor.

The user activates the trap door by typing in the trigger sequence. The trap door is periodically scanning the teletype buffer area for the trigger sequence. When the sequence is recognized by the trap door the terminal is removed from the subsystem environment and given whatever control the subverter that inserted the clandestine code desires.

b. NPS Penetration Case

During the time that this research was being carried out, one of the schools computer systems was subjected to 'attack' by a malicious individual. The system in question was a PDP-11/50 running under the UNIX operating system. This case is a simple example of breaking out of a subsystem.

The subsystem under consideration was the 'games' monitor. This system has several games programs that came with the system or were written by students as class projects. The subsystem is 'constructed' by having users (no password required) that log in under the games user-id restricted to executing only those programs and commands that reside in the games directory. The games option is only enabled during 'off' processing periods when the system use is low. The malicious user was familar enough with the system to know the dialup terminal phone number. It was apparent that he was familar with the UNIX system, because he wrote a program (the trap door) and inserted it into the games directory.

The program was called 'ZX' and it was a 'C' language program that executed one command language (called 'shell') statement. Since this program was in the games directory, the monitor environment did not prevent the execution of the command language statement. This trap door gave the individual all the privileges of an unrestricted

(non-super) user. He could (and did) read the password file for names of legitimate users. He found some users that had the same password as their name (this example was mentioned earlier). He was later discovered logged in under some of the legitimate users names, or would respond with one of these names when queried online.

Dialup capabilities were eventually restricted by a monitor to specially authorized personnel, and the mysterious 'attacker' did not make his presence known again. Several procedural errors where identified in the course of the 'investigation' and have since been corrected. Among these were the password assignment procedures (mentioned earlier) were no longer initialized as the users name, and the restriction of the dialup capabilities. This 'attacker' did not appear to be malevolent in his actions. He seemed as though he was looking for a little 'free' computer time. But there is no way to determine this for sure, nor is there a way to determine what other artifices might still be present in the system.

c. Using Emitters

Computer systems are electromagnetic emitters like any other piece of electrical equipment. Information can be gathered by monitoring these emanations. Communication lines and cathode ray tubes are particularly vulnerable to these techniques [11]. Security personnel are generally aware of this problem [8]. Computer sites can be

measured for the amount of emanations present. If they are sufficienctly low, a site could be certified as satisfactory in this area. However, if there were covert transceivers imbedded in the equipments at the factory this 'certification' could prove useless. A transceiver that is monitoring a data bus could sense a data stream trigger. Upon activation the transceiver would begin to broadcast the activity on the data bus at a higher power level than would be normally present. Since the transceiver was not active during the 'certification' its presence would not be detected. A similar sequence could act as the deactivation key to stop the transceiver from broadcasting. As one can see this is nothing more than a specialized hardware trap door.

d. Memory residue

In a resource shared system the allocation of memory could result in the exposure of sensitive information to unauthorized users. Unless specific actions are taken by the operating system or the previous user, memory assigned to a new user program will contain whatever was last placed in it.

The ADP Security Manual [8] addresses the problem:

The O/S shall ensure that classified material or critical elements of the system do not remain as accessable residue in memory or on on-line storage devices.

This means that the operating system must clear core before it is assigned to a program. This mechanism, if subverted, could be designed to 'turn off' by command.

This could prove valuable to the subverter who has agents that are legitimate users of a system. As a matter of standard procedure the agents could perform the following actions whenever they are processing jobs:

1. program begins execution and immediately turns off the clear core mechanism by activating an artifice.

2. program waits for sufficient residue to build up in the free core area, and requests additional core for the next processing step.

3. upon receiving the additional core the program dumps the contents of the core to a file in his directory for later review.

4. program turns residue mechanism back on and completes legitimate tasks.

Another problem with memory reside arises when a computer is involved in what is commonly called 'periods processing'. A periods processing environment is one that uses the same computer to process information of different security levels, but at different times.

After each processing period in one mode, special procedures are carried out to insure that all traces of information are removed from the system. This is known as 'color changing'. This includes removing all tapes, cards,

printouts, ribbons, etc., from the system. The next shift would bring all the necessary equipment with them to do the same. One of these procedures is, of course, clearing core. The program used to 'clear' core could be one that writes random patterns into core. This could be be repeated several times to ensure a good 'brainwashing'. Assuming the color change was from classified to unclassified, it would be possible to obtain information from the previous processing period. If the program that cleared core did not write random patterns into core, but just encrypted the information, it would be undectable by the operator. A clandestine process, that runs in the unclassified period could core dump the information to files for later decryption.

e. Using Confinement Channels

Confinement channels have traditionally been thought of as a slow means of extracting information. But in an environment where particular care has been taken to defend against subversion, this method may be the only way of gaining information. Channels on the order of a bit per second have been demonstrated and channels that can pass on the order of tens of bits per second have been hypothesized [22]. The following are a few examples of what form these channels might take:

1. If the system has interlocks which prevent files from being opened for writing and reading at the same time, the

86

the service can leak data if it is merely allowed to read files that have been written by its owner. The interlocks allow a file to simulate a shared boolean varible which one program can set and the other can test[16].

2. By varying its ratio of computing to input/output or its paging rate, the service can transmit imformation which a concurrently running process can receive by observing the performance of the system. The communication channel thus established is a noisy one, but the techniques of information theory can be used to devise an encoding which will allow the information to get through reliably no matter how small the effects of the service on system performance are, provided they are not zero. The data rate of this channel may be vary low, of course [16].

3. An exploitable path for information flow can be created between an uncleared individual accessing the system during one processing period and the classified information processed by the system during another processing period if, over time, the same software is employed in both processing periods. Such a 'covert leakage path' can effectively negate the necessary complete isolation between processing periods...[23].

Case 1 is very similar to the process-id binary channel discussed earlier. But in this case the binary channel is the interlock. The owner (subverter) knows the service program (which has access to the sensitive data) is sending a binary '1' if the service opens the given file for reading. This is because he would be prevented from writing into the file by the interlock. He would be receiving a '0' if he was permitted to write the file.

Case 2 is similar to the example that measured the runtime of a program. In this case low system performance means a '0' and higher system performance a '1'.

Case 3 is an example of passing information between processing periods. Assume that the machine in

question is one that supports memory paging. Also assume that the programs in question are reentrant routines. This means that they would not get swapped out during a page fault, just overwritten. Should the program be able to execute in the master mode, it could write sensitive information into unused portions of the code block (like the UNIVAC 1108 example). Since the code block was modified the page swapping routine would swap it out vice overwriting it. When the next unclassified processing period starts, the subverter merely reads the data from the code block of the program.

f. Affecting System Performance

Not all subversion activities would be concerned with gathering information. For some computer systems the subverter may only be interested in rendering these systems ineffective at key times. Tactical or strategic systems are examples of where this might be desirable.

A systems design or implementation could be subverted so that its performance may suffer during critical situations. It is often difficult to test such systems under critical real world conditions. These systems could meet performance specifications under simulated situations but prove ineffective in a real world situation.

Triggering of artifices in these systems can be by external events. Suppose there is a command and control

system that keeps track of potentially hostile ships. A trap door entered during the implementation phase of this particular system is designed to activate whenever it detects that a certain ship was reported at a certain position. When the opposing side decides to start hostile operations, it could sent the designated ship out to the predetermined position before the start of hostilities. The ship could remain at that position long enough to insure that the intelligence system had time to enter the ship into the system. When the trap door recognized the activation key (ship identification and position) it could cause the system to gradually degrade in performance until it was ineffective. The ship would have, in effect, 'sunk' the command and control system from thousands of miles away. Examples of what an artifice could cause to happen to this kind of system are:

1. cause the system to crash at random intervals,

2. slow down the system performance by randomly clearing core page usage data, thus causing the system to swap pages in and out of core excessively (thrashing),

3. randomly ignore or lock out the command console.

Activity such as this would render the system unreliable and create an unwillingness to use it. Furthermore, systems maintenance personnel would make the system unavailable for many long hours while looking for a bug that may never be

found. Since it was installed during the implementation phase it would exist in all copies of the system code.

D. Retrieving Information

Once information has been accessed by the methods outlined previously, the problem of removing the information from the confines of the security perimeter still remain. As one might expect, the difficulty of the retrieving process is directly related to the 'strength' of the security perimeter. In a relatively open system retrieval might be as easy as walking out the front door with listings under one's arm. In a more restrictive environment other methods can be devised. In a multilevel security mode, the unclassified user is frequently not scrutinized; in fact, he might by using a dialup terminal several miles from the computer installation.

This discussion will assume that the exercising phase of subversion has placed the desired information in a 'safe' place (i.e., any area that is accessable to the subverter).

1. Retrieving Files

If the internal protection mechanisms were used to enforce the security perimeter (as in a multilevel security system) then the subverter may have a simple job of retrieving the information. Since the security controls were

circumvented in obtaining the information, the security perimeter has been breached and retrieval may only involve dumping the information out in some transportable form. However, if this is not the case the information may be reviewed by someone before it is allowed to cross the security perimeter. In this case the information must be desguised or perhaps even encrypted.

Information can be hidden in the header pages or system job statistics areas of batch job printouts. These are often ignored areas of a listing. These areas could offer low bandwidth channels for the information.

Encrypting information into statistical tables or core dumps can significantly increase the volume of information that can be channeled through the security perimeter.

2. Retrieving with Hardware devices

Hardware transmitters can be used to pass informtion beyond the security perimeter. These devices can offer channels of very high bandwidth. A high speed printer that had a transmitter imbedded into it during the installation phase is an example. Again the activation key could be a sequence of characters in the data stream that turns on the transmitter and a similar sequence to turn it off.

An interesting method that could be used for a low bandwidth channel is the front panel of the computer

console. Some installations have big glass windows that define an external security perimeter. A subverter could submit an unclassified job to a system that could serve to activate a trap door. The subverter only need watch the register lights for the information to be flashed to him. Naturally the normal register lights would be flashing to rapidly for the subverter to understand them. However the parity light for the registers could be control in such a manner that they could send Morse code to the subverter. By having a program that repeatedly enters even parity or odd parity values in to a register an information channel could established. Furthermore, the flashing could be recorded photographically or using vidio tape.

E. CHAPTER SUMMARY

This chapter has outlined the methodologies of computer subversion. This subversion may involve the organized efforts of many individuals whose talents could range from a computer scientist to an unskilled laborer. Subversion is a three step process involving the insertion of artifices into computer system components, exercising them, and retrieving the resultant information. The insertion process could be carried out over the entire life cycle of a computer system, from the beginnings of its design through to, and including, the the production phase. Once installed these artifices can

be used to circumvent normal internal controls of the computer system for the purpose of accessing unauthorized information. Once unauthorized access is obtained, the subverter need only disguise this information into a form that will circumvent any external controls that may exist, thus effecting its retrieval.

Subversion is clearly a threat to the security of any information that relies on a computer system to protect it. In the next chapter ways of minimizing the risk of subversion are investigated.

V. MINIMIZING THE RISK OF SUBVERSION

Theoretically, there are three ways in which subversion can be minimized, and they relate directly to the three phases of subversion:

1. Prevent the the insertion of all mechanisms that can be utilized to defeat internal scurity controls, or

2. Prevent the malicious user from exercising these mechanisms, or

3. Prevent the retrieval of any information gained via exercising techniques.

Any one of the three methods mentioned above could prevent subversion. Each method will be briefly discussed as to its merits in helping to minimize the threat of subversion.

A. RESTRICTING INSERTION OPPORTUNITIES

Preventing the subverter from inserting artifices may not be a simple task, but it is essential to the ultimate solution to the problem of subversion. It has been demonstrated how subversion can occur over the entire life cycle of a computer system. To prevent the insertion of artifices implies that the subverter must be prevented the opportunity to access system components at any point during

this life cycle. Clearly, system components that affect the security of the system must be afforded lifetime protection.

1. Lifetime Protection

For lifetime protection to be effective it must involve such measures as:

1. Appropriate security clearances for any personnel involved in the various stages of the computer systems life cycle [2].

2. Sufficient 'hardening' of manufacturing and development programming sites to prevent subversion by external forces [2].

3. Proper protection of all system components from access by malicious elements for the entire systems life cycle.

Without the above measures, proper assurances would not exist concerning the safe history of system components. That is, whether or not malicious elements have had the opportunity to subvert the components. The only appropriate course of action would be to not allow these components to participate in the protection of information. This is because the very nature of subversion is covert, and it would be virtually impossible to detect if it had occured in a system after the fact. If any period during the lifetime

of a computer system has a lapse in protection it must be similarly assumed that these components are unreliable from that point forward.

2. Appropriate Protection Policies

The above measures should be viewed in the proper perspective. What is meant by 'sufficient hardening' of development sites, or 'proper protection' of system components?

Just because a computer system will be involved in the protection of classified information does not mean that the system components are themselves inherently classified. It would therefore not be appropriate (even counterproductive) to demand that these system components be protected in the same way as classified materials. For instance there would not be any reason to prevent copies of programs from being seen. The central issue is not the content of the programs, but restricting access (for modification) to the particular copies of those programs that will be used to enforce protection in the system.

A more appropriate protection policy is needed. In essence this policy should outline a strategy of 'look, but do not touch'. For instance, in the area of development or manufacturing sites, hardening does not have to be concerned with emanations where the is no sensitive information

contained in the operating system code or hardware equipment at this point in the life cycle.

Similarly, the proper protection of system components would dictate that they be protected from malicious elements having access. Previous chapters have outlined in detail that there are many ways that a subverter can access system components. Therefore, countermeasures to these access routes must be devised. But restricted access need only apply to those particular programs and equipments actually involved in the protection of information. Copies of the programs could conceivably be made available to anyone. However, those particular components (programs or hardware) that will actually be used in the protection of information need to be clearly distinguished and protected. Specifically, those particular components involved in the protection of information should be labeled and protected from access at the same level as the information they are expected to protect.

One of the basic principals of subversion involves the introduction of clandestine mechanisms into security related system components. However current DOD security program regulations and directives [7,8,24] are primarily concerned with the REMOVAL of sensitive materials from a secure environment. These directives must be changed to ensure that security not be compromised by the INTRODUCTION of materials as well.

B. RESTRICTING EXERCISING OPPORTUNITIES

To prevent the exercising of mechanisms that could defeat internal system controls, one could:

1. Find and eliminate all such mechanisms, or

2. Somehow guarantee that they could not be employed. Both these 'solutions' when applied to current operating systems are, in any practical sense, infeasible.

Both these 'solutions' assume that such mechanisms can be identified in the first place. To do do this would require a means of determining that every program executed on a machine is 'safe'. But chapter II brought out the fact that there is no general solution to the safety problem [4]. A simple example of this is a Trojan horse. As previously indicated, the user willingly invokes a malicious program and, in doing so, gives it 'permission' to perform its covert functions. Not only will most computer systems not prevent the employment of such a program, it will unknowingly aid in its endeavors.

Finally, one must consider the system foible (design and implementation errors). Recall that these are mechanisms that can also be of use to the subverter. To presume that all such foibles are identified and eliminated is to imply that the perfect design was flawlessly implemented. This is a highly unlikely prospect. Chapter II offered ample testimony to the fact that current technology is a long way

from the perfect implementation of something the size of a modern operating system. If 'accidents' such as system foibles are difficult to find, then the deliberately obscured artifice would be virtually impossible to detect. Attempting to prevent the exercising of artifices is a futile approach.

C. RESTRICTING THE RETRIEVAL OF INFORMATION

Restricting the retrieval of information must presently be considered the last defense against subversion. This is obvious because, as pointed out earlier:

1. No assurances exist as to the absence of past subversive activities on system components, therefore subversion of the components must be assumed.

2. There exists no general method that can prevent the exercising of clandestine mechanisms in a computer system.

Ultimately, preventing the retrieval of unauthorized information from a system will lie with the effectiveness of the security perimeter. If the subverter can cross this defensive barrier then he has, in effect, retrieved the information. One must clearly delineate where this perimeter lies. Unless it is clearly delineated, one cannot determine the effectiveness of those mechanisms designated to enforce it.

1. Delineating the Internal Security Perimeter

When the security perimeter of a computer system is enforced by strictly external means, the system is said to be operating in the dedicated security mode [21,22]. The security perimeter is clearly defined as those physical measures (such as guards, etc.) required to insure that no unauthorized information will leave the boundries of the perimeter. All users, equipment, and information reside within this perimeter. The effectiveness of this kind of security perimeter is easily determined as it is based on established practices that are not unique to computer security. The dedicated mode of operation is the result of the need to restrict retrieval of information. This is certainly a sound technique but it does not solve the classical computer security problem. That is, the need to reliably share information of varying degrees of sensitivity among users of varying degrees of trustworthiness.

In the case of the computer that is used in the multilevel security (MLS) mode, the security perimeter is less clear. In this mode of operation the security perimeter is enforced by the internal protection mechanisms of the computer system. This is because personnel that are not cleared for the highest level of information contained within the system are allowed some form of access to the system. The only barrier between the uncleared user and the

information that he is not authorized to access is the internal protection mechanisms of the computer system. Therefore it is imperative that this internal barrier (i.e., security perimeter) be well defined within the system.

The difficultly with contemporary computer systems is that control of these internal protection mechanisms is distributed throughout the entire operating system. There is no clear distinction as to which parts of the system enforce the security perimeter and which do not. As a result of this vagueness, any attempt to evaluate the effectiveness of a computer system to enforce a security perimeter is doomed to the ad-hoc approaches such as those outlined in Chapter II. And these are notoriously ineffective.

So called 'trusted' subsystems compound the problem by attempting to 'establish' a security perimeter with a special program. But ultimately a subsystem will use the very same protection mechanism that the underlying operating system uses. It should be clear by now, that in the face of subversion the subsystem is not the least bit more secure than the underlying operating system and other security related components that it embraces.

It is clearly essential that any internal protection mechanism be defined in such a way that it's effectiveness can be demonstrated. One such mechanism is the Security Kernel. Schell [6] states:

The chief distinguishing characteristic (from whence its
name) of the security kernel concept is that a kernel
represents a distinct internal security perimeter. In
particular, that portion of the system responsible for
maintaining internal security is reduced from essentially
the entire computer to principally the kernel.

It is instructive to see how this mechanism could be used to

prevent the subverter from retrieving unauthorized

information.

2. Security Kernel Concept

In a system that is based on a security kernel,

protection is realized within the computer system by the

verifiable implementation of a mathematical model of

information security. This model is based on an abstract

representation of security called the reference monitor [5].

The reference monitor describes a mechanism for controlling

the access privileges within the system (see references

[2,5] for further details on the monitor). The

implementation of this mechanism is the security kernel.

The security kernel is designed to be a verifiable

subset of security related operating system functions. These

functions form an interface (i.e., a security perimeter)

between the user and the information. If the security kernel

is implemented correctly, its use will guarantee that the

information in the system will be protected in accordance

with the security policy that is outlined in the security model. Essential design requirements of the security kernel are:

1. It must be tamper proof.

2. It must always be invoked.

3. It must be small enough to be subject to analysis and tests, the completeness of which can be assured.

The Multics Security Evaluation [10] points out how comtemporary systems have been unable to meet these criteria:

> The stated design goals of contemporary systems such as GCOS or OS/360 are to meet the first requirement (albeit unsuccessfully). The second requirement is generally not met by contemporary systems since they usually include 'bypasses' to permit special software to operate or must suspend the reference monitor to provide addressability for the operating system in exercising its service functions. The best known of these is the bypass in OS/360 for the IBM supplied service aid, IMASPZAP (SUPERZAP). Finally and most important, current operating systems are so large, so complex, and so monolithic that one cannot begin to attempt a formal proof of certification of their correct implementation.

Two basic precepts that are enforced in the security kernel are:

1. The simple security condition- This means that a user or his program is not allowed access to information for which he has no authorization.

2. Confinement property- if a user or his program has read access to information at one security level, say secret, then he cannot have simultaneous write access to

103

a file that exists at a lower security level (i.e., unclassified). This prevents what is called a 'write down'.

These simple precepts and other supporting strict rules of the security kernel are the basis by which the subverter is prevented from retrieving unauthorized information.

In the case of the Trojan horse, the simple security condition and the confinement property can render such a clandestine mechanism useless. The basic concept behind a Trojan horse presumes that it will be allowed into an environment that contains sensitive information. Once in this environment the covert function attempts to obtain sensitive information and place (write) it in area that will be accessible to a subverter. The security kernel, through the confinement property, will not permit a 'write down'. That is, it will prevent the covert function from 'declassifying' the information by not allowing it to be copied to anywhere but another classified file. Assuming the subverter is an unclassified user, the simple security condition will prevent him from accessing any files gained through this method because he will not have the proper clearance to to read the file provided by the Trojan horse.

D. CHAPTER SUMMARY

Security kernel technology directly addresses the problem of minimizing subversion. It offers a basic design

that can be proven effective. Through this verifiable protection mechanism a distinct internal security perimeter can be relied on to prevent the retrieval of unauthorized information by malicious elements.

But security kernel technology is not immune to to the subversive techniques outlined in this thesis. In fact, it might be more susceptible to subversion due to the high probability that such a system will be used in sensitive areas. Lifetime protection is essential to any mechanism that will be employed in the protection of information.

The security kernel clearly defines the security related mechanism of a computer system. Because of this it is the only part of a computer operating system that need be offered lifetime protection. Providing protection for the security kernel is a far more practical an idea than requiring the lifetime protection of an entire operating system and numerous privileged utilities. Its small size and clear boundaries offer a secure foundation from which to build any operating system. But without lifetime protection from malicious access, there would be no assurances as to the integrity of components involved in the protection of information and subversion must be assumed.

VI. CONCLUSIONS AND RECOMMENDATIONS

This thesis offers a detailed examination of an aspect of the computer security problem known as subversion. It is not the purpose of this document to provide a handbook of subversion for subverters; they do not need one! This thesis does offer awareness to those who must deal with the computer security problem. People like ADP administrators, ADP security officers, system designers, and others involved in the decision making process must understand subversion if they are to effectively combat it. It is difficult to make intelligent decisions concerning the security of information in computer systems unless one understands the possible extent of the vulnerabilities that could exist in them.

The first part of this thesis identified several problem areas in computer security. One of these areas involve a lack of a coherent policy concerning the exact role that computers should play in the protection of information. This in turn has led to a reliance on inadequate internal mechanisms, and false assurances as to their effectiveness. All these problem areas play a role in the success of subversion.

Important distinctions have been made between the current conception of computer penetration and that of subversion. The penetrator is basically an amateur that

exploits system design and implementation errors to gain control of a system. Subversion on the other hand involves the organized efforts of several individuals, some of whom are highly competent at the subversion process. The subversion process involves the use of clandestine mechanisms called artifices. Principal among these artifices are trap doors and trojan horses. By constructing and inserting these mechanisms into computer systems the subverter creates a safe environment which can be used to exploit a computer system at will.

The three phases of subversion are the inserting of artifices, the exercising of them, and the retrieval of the resultant unauthorized information. Central to the theme of subversion is the insertion of artifices over the entire lifecycle of a computer system. This can be done because computer system components that would be involved in the protection of information do not receive adequate protection against subversive activities during their lifetime.

Subversion is a clear threat to the security of any computer system involved in the protection of information. This threat must be minimized before computer systems can be relied on to adequately protect information. Until such a time, no computer system should be used as a means to protect information. So-called 'trusted' subsystems are no exception. They suffer from the same risk of subversion as any other system. The problem of 'trusted' subsystems is

107

compounded by the fact that they are built on an underlying operating system that is essentially unsecureable. These systems must be considered particularly dangerous to use because they lull the user into a false sense of security.

Minimizing the threat of subversion is a twofold process. First, adequate lifetime protection must be afforded to all security related components that will be involved in the protection of information. The integrity of security related components cannot be assured without this protection.

Second, the application of adequate technology as exemplified by the security kernel concept must be incorporated in the design of secure systems. Without this verifiable design, the effectiveness of the protection mechanism cannot be reliably determined. Unless these essential requirements are met, there will be no such thing as a secure computing system.

LIST OF REFERENCES

1. Consensus Report, Processors, Operating Systems and
   Nearby Peripherals, Theodore M. P. Lee (Chairman),
   AFIPS Conference Proceedings, 1979 National Computer
   Conference, June 4-7, 1979.

2. USAF Electronics Systems Division Report ESD-TR-73-51,
   Vol. I, Computer Security Technology Planning Study,
   October 1972.

3. Gat, Isreal, Security Aspects of Operating Systems,
   Second Jerusalem Conference on Information Technology,
   1974.

4. Harrison, M.A. Ruzzo, W.L., Ullman, J.D., "Protection
   in Operating Systems", Communications of the ACM,
   Vol. 19, no. 8, August 1976.

5. Schell, Roger R.,LtCol., USAF, "Security Kernels: A
   Methodical Design of System Security",USE Inc., Spring
   Conference, March 1979.

6. Schell, Roger R.,LtCol., USAF, " Computer Security, The
   Achilles Heel of the Electronic Air Force", Air
   University Review, Vol.XXX no. 2, January-Febuary 1979.

7. Department of Defense Directive 5200.28 "Security
   Requirements for Automatic Data Processing (ADP)
   Systems", 18 December 1972.

8. Department of Defense Manual 5200.28M "Techniques and
   Procedures for Implementing, Deactivating, Testing,and
   Evaluating Secure Resource-sharing ADP Systems",
   January 1973.

9. Nibaldi, G.H., "Proposed Technical Evaluation Criteria
   For Trusted Computer Systems", Mitre Corp., no.M79-225
   Bedford, Mass., 25 October 1979.

10. USAF Electonics Systems Division Report ESD-TR-74-193.
    Vol. II, Multics Security Evaluation: Vulnerability
    Analysis, by Paul A. Kruger, 2Lt., USAF and Roger R.
    Schell, Major, USAF, June 1974.

11. Lackey, R.D., "Penetration of Computer Systems, an Overview", Honeywell Computer Journal, Vol. 8, no. 2 1974.

12. Comptroller General of the United States, Report to the Congress. Computer Related Crimes in Federal Programs, General Accounting Office (GAO), April 27, 1976.

13. Stanford Research Institute Report PB-231 320, Computer Abuse, by Donn B. Parker, Susan B. Nycum, and Stephen S. Oura, November 1973.

14. Denning, Peter J. and Dorothy E. "Data Security" Computing Surveys, Vol. II no. 3 September 1979.

15. Rome Air Development Center Report RADC -TR-74-137, Emulating a Honeywell 6180 Computer System, Mitre Corporation June 1974.

16. Lampson, D.W., "A Note on the Confinement Problem", Communications of the ACM, Vol. 16 no. 10 October 1973.

17. Naval Research Laboratory Memorandum Report 2821 Subversion of a "Secure" Operating System, by David Stryker June 1974.

18. USAF Electronics Systems Division Report ESD-TR-74-193, Vol. III, Multics Security Evaluation: Password and File Encryption Techniques by Lt. Peter J. Downey, USAF, June 1977.

19. Attanasio, C.R., Markstien, P.W., and Phillips, R.J., Penetrating an Operating System: A Study of VM/370 Integrity", IBM Systems Journal, Vol. 15 no. 1, 1976.

20. Goheen, S.M., and Fiske, R.S., OS/360 Computer Security Penetration Exercise, Mitre Corp., Bedford Mass. October 1972.

21. USAF Electronic Systems Division Report ESD-TR-75-69, The Design and Specification of a Security Kernel for the PDP-11/45 By W.L. Schiller (Mitre Corp). May 1975.

22. Lipner, Steven , "A Comment on the Confinement Problem" Mitre Corporation, Bedford Mass.

23. Department of Defense Industrial Security Newsletter
    no. 80L-1, 28 March 1980.

24. Department of Defense Information Security Program
    Regulation 5200.1R, December 1978.

25. Information Sciences Institute Report ISI/SR-78-13,
    Protection Analysis: Final Report, by Richard Bisbey
    and Dennis Hollingworth, May 1978.

INITIAL DISTRIBUTION LIST

No. Copies

1. Defense Technical Information Center   2
   Cameron Station
   Alexandria, Virginia 22314

2. Library, Code 0142   2
   Naval Postgraduate School
   Monterey, California 93940

3. LtCol. Roger R. Schell Code 52Sj   5
   Department of Computer Science
   Naval Postgraduate School
   Monterey, California 93940

4. Asst. Prof. Douglas Smith Code 52Sc   1
   Department of Computer Science
   Naval Postgraduate School
   Moneterey, California 93940

5. Lt. Philip A. Myers, USN   5
   Naval Data Automation Command
   Washington Navy Yard
   Washington, D.C. 20374

6. LCDR. S.L. Reitz   1
   NAVAL SEA SYSTEMS COMMAND
   Technical Representive
   St. Paul, Minnesota 30845

7. Capt. L.A. Talmage   1
   Manpower Utilization Unit
   Building 2009
   MCDEC, Quantico, Va. 22134

8. Capt. John Ross, USAF   1
   552 AWACW/ADM
   Tinker AFB, Oklahoma 73145

9. Ms. Cheron Vail, Code 302   1
   NAVPERSRANDCEN
   San Diego, California 92152

10. Lt. M.L. Maurer, USN   1
    CARGRU Five
    F.P.O. San Francisco, California 96601

11. Lt. W.J. Wasson, USN   1
    Naval Electronics Systems Command
    Headquarters, PME 124
    Washington, D.C.   20360

12. Department Chairman, Code 52   2
    Department of Computer Science
    Naval Postgraduate School
    Monterey, California   93940

13.Lt. Alan Gary, USN   1
    Operations Department
    U.S.S. Nimitz (CVN 68)
    F.P.O. New York, New York 09542

14. LCDR. Edmund Moore, USN   1
    Naval Electronics Systems Command
    Headquarters, PME 107
    Washington, D.C.   20360

15. Lt. William C. Hess, USN   1
    Naval Electronics Systems Command
    Headquarters
    Washington, D.C.   20360

16. LCDR. B. Johnson, Code 0371   1
    Computer Technology Curricular Office
    Naval Postgraduate School
    Monterey, California   93940

17. Mr. Carl Landwehr   1
    Code 7522
    Naval Research Laboratory
    Washington, D.C.   20375