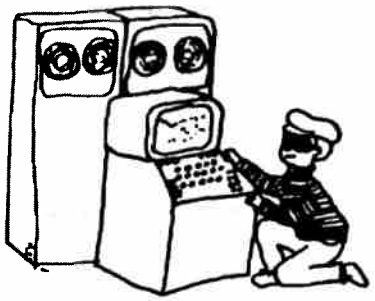
COMPUTER OPERATING SYSTEM VULNERABILITIES

[redacted] S86



P.L. 86-36

Can my system really be penetrated?" This is the question so often asked by computer system managers. The inevitable answer is "Yes. Any computer system can be penetrated by a knowledgeable user." Large computer systems, in particular, by their size and complexity, leave themselves open to attacks by unauthorized users. Let us examine some of the vulnerabilities of computer systems, as well as some of the possible defensive measures.

COMMON OPERATING SYSTEM VULNERABILITIES

Operating system vulnerabilities generally fall into one or more of the following seven classes: [1,2]

- Incomplete parameter validation
- Inconsistent parameter validation
- Implied sharing of privileged confidential data
- Asynchronous validation and inadequate serialization
- Inadequate identification, authentication or authorization
- Violable limits
- Exploitable logic error

Let us look in detail at each class of flaws and see how they affect the system operation.

Incomplete parameter validation. Whenever a user requests any type of service, the operating system must verify that the user is authorized to make that request and that a proper parameter string has been provided by the user. This verification is done to prevent the user from compromising a control program which is performing services for all users. Flaws in some operating systems may allow a user to "fool" a control program into:

providing him access to data which he would not otherwise be allowed access rights,

placing the user program into privileged or executive mode, or

severely degrading the operation of the ADP system.

The following is a good example of incomplete parameter validation:

Using a file dump routine, User X requests a dump of 300 records from File A, but File A contains only 200 records. The system honors the user request, and User X is allowed access to not only File A, but also to whatever data is stored beyond the address area of File A.

Security requirements should make the control routine validate the parameters and either reject the user request or dump only those records which apply to File A.

Inconsistent parameter validation. Inconsistent parameter validation occurs whenever there are multiple definitions for the same construct within the operating system. For example, a system control program may validate a user program's parameters but trusts another system routine's parameters as valid without verification. Therefore, a user who can fool the system into believing his code is system routine code can obtain unauthorized privileges. System routines should verify all input parameter strings, even those from another system routine.

Implied sharing of privileged or confidential data. In a multiprogramming environment, the computer's facilities are shared by many users. The operating system must have the built-in capability to isolate each user from all other users. Failure to provide this segregation can result in a possible compromise of privileged information. In modern operating systems two problems are generally noted in this area.[3] The first is the matter of sensitive residue. This involves information left behind in memory or other storage media after a run has terminated. An unauthorized

user can enter the system and obtain access to these "leftovers." This technique is commonly known as *scavenging*. The second problem involves the system sharing user space for its own storage. To save space, the operating system frequently shares the user's buffers to store temporary working tables. This may allow the user unauthorized access to the system tables, i.e., password tables, etc. This is frequently known as the *unerased blackboard problem.*[1]

Asynchronous validation and inadequate serialization. System integrity is guaranteed only if information passed between program sequences is protected. If the operating system allows asynchronous operations and the operations are not performed in a timely sequence, the information may be modified or compromised. An example of this would be permitting the user to perform I/O into a checkpoint or restart file so that his restarted program is given unauthorized or supervisory privileges.[4] To be secure, an operating system must be able to enforce timing constraints to a controlled state.

Inadequate identification, authorization or authentication. Most operating systems maintain some type of job initiation procedures which monitor authorized vs. unauthorized access. A system flaw exists whenever a system permits a user to bypass these security mechanisms. A user who finds a way to obtain executive operation mode can "walk" through the system without being questioned by the system monitor. Operating systems must require proof of access rights for all user requests. Security mechanisms must be protected from user tampering. For example, password files should be encrypted or protected from common access and must be unusual enough to void any guessing or permutation attempts.

Violable limits. Because of architectural limitations, the operating system has to limit the resources a user can control. These limits or "hands off" policies are usually described in the system documentation. Whenever an advertised limit is not enforced, a security flaw exists. For example, a user may be limited to operate within an assigned partition of storage; but a flaw in the system allows him access to another partition on an overflow condition. Because the operating system did not enforce the "rules of the road," a user could accidentally or deliberately cause a system overload, resulting in system degradation or crash.

Exploitable logic error. With four to five million lines of code, it is inevitable that there will be bugs in any major operating system.[4] A knowledgeable user may exploit these errors to his advantage to obtain access to information or programs to which he is not authorized. Logic errors can especially be created whenever the original design or coding has been changed. Logic modifications compromise any security measures designed into the original system. Examples of exploitable logic errors are frequently found in error-handling procedures. A user may request modifications or dumping of a file belonging to another user. Incorrect error handling may initiate the actions without first verifying that the user has access rights to that file. There is no way to avoid logic errors in large operating systems; however, these errors should be corrected when discovered to avoid prolonged compromise of sensitive information.

PENETRATION TECHNIQUES

Now that we know what some of the potential operating system flaws are, we need to know how a knowledgeable user, or *penetrator*, will exploit these flaws to obtain unauthorized access to the system. In planning his attack, the penetrator will have to answer the question, "What do I want—information or system degradation?"[5] The answer to this question will determine his method of attack. The penetrator's next step is to obtain all available system documentation. Valuable information which may point to vulnerabilities is available in the documentation. After reviewing the manuals, the penetrator can then decide on the techniques to be used in the penetration attempt. The penetrator's main objective is to attack one or more of the seven major flaw classes discussed earlier.

Probably one of the most available and easiest system penetration methods is the use of utility programs.[3] These service routines often execute user requests without requiring proof of access rights. Some types of utility routines are storage dump facilities, operations support programs and maintenance support programs.

Another widely used penetration technique is operator "spoofing." A penetrator can use trickery, such as giving his program the same name as a system routine, to make the operator think that his program is a privileged system routine. He may then request a load of privileged disc packs or magnetic tapes.

The penetrator can also obtain access to privileged information by creating a *Trojan horse.*[3] A Trojan horse is a program which, in addition to doing what it is advertised to do, does something else which its user doesn't know about and wouldn't want done. A Trojan horse is usually hidden in a utility program. An example would be a performance monitor which also dumps user information into a file somewhere (account numbers, passwords, etc.).

System penetration can also be obtained

by using any of several covert attacks.

Wire tapping. Also known as eavesdropping, this act involves the penetrator connecting some listening device to a communications line somewhere between a peripheral device and the computer central processing unit being penetrated. This is a passive operation.

Between lines entry. This is similar to wire tapping except that the process is active. The penetrator enters spurious commands onto the communication lines which were meant only for the legitimate users. This operation is usually done when the intended terminal is at an idle state.

Clandestine code. This operation involves the entering of changes, possibly a Trojan horse, into the coding of the computer operating system.

Masquerading. This involves logging into the computer system as a legitimate user whose account number and password have been acquired by begging, borrowing or stealing.

DEFENSIVE MEASURES (COUNTERMEASURES)

So, if our system is so susceptible to unauthorized access, how can we set up a defense against these measures? The best approach is to build security into the initial system design.[3] Patches to the design at a later time may create more flaws than they patch. The problem with most current operating systems lies in the fact that they were developed in the 1960s with no thought in mind for security requirements. Even with security in mind, we must remember that operating system security is not a binary yes-no condition. No large operating system currently in use can be completely certified as secure.[2]

Here are examples of measures which we can take to protect our system from attack.

Data encryption. Data encryption is becoming more widely used by both the government and private industry. Encryption should be performed whenever sensitive information, such as password files, payroll data, defense statistics, and the like, is stored or sent over data communication lines.

Using a minicomputer as a front-end security controller. This technique could be used to control access to the host computer from remote terminals. This would remove the security overhead from the host computer's operating system. The smaller operating system in the minicomputer would also be easier to certify as secure.

Mathematical models. Models allow systems analysts to study the complete operating system environment and pick each area apart for security analysis.

Kernels. Kernels are small portions of software blocked together to perform a single function. These small software modules could be certified secure.

Software verification tools. Many tools have been or are being developed to certify the security of computer software.

A LOOK AT FUTURE RESEARCH AREAS

Many areas in computer system security need to be explored in the future. Some of those areas are:

1. Development of better control structures (audit trails);[2]

2. Expansion of kernel theory to develop a "secure" operating system;[3]

3. Cost analysis studies (Where do we draw the line between cost of computer security and need? How do we measure security?)[3]

4. Development of strong consistent management policies to govern the use of computer facilities;[4]

5. Development of software verification tools to certify computer software;[3]

6. Development of some type of virtual machine monitor (an operating system which isolates each user into his own mini-operating system), which when properly designed and implemented is "spoof-proof";[3] and

7. Development of a security specification language which allows security requirements to be programmed into the operating system by the security officer.

I hope I have been able to provide some insight into just how vulnerable modern computer operating systems are. Department of Defense studies have shown a need for protecting data relating to the nation's defense because of the many opportunities for fraud and embezzlement.[2] We must also realize that software security is only one aspect of the total security environment. We must also consider administrative, personnel, physical, communications, emanations and hardware security. As modern technological advances are made, with their applications for computers, we will have a continuing requirement for operating system security.

No matter what misuses take place, we must realize that people are still going to use that magnificent adding machine, the computer. It has been proven that there are people with

skills to crack safes, yet people still use safes. The same correlation can be made to computer usage. Our job as system managers is to attempt to protect against accidental or deliberate destruction, modification, or disclosure.[2] Security policy (administrative, personnel, physical, communications, emanations, hardware and software) and practices must be sufficient to make up for the computer's inability to protect itself.

---

1. Webb, D.A. and Frickel, W.G., "Handbook for Analyzing the Security of Operating Systems," Lawrence Livermore Laboratories, 1976.

2. Abbott, R.P. et al., "Security and Enhancements of Computer Operating Systems," National Bureau of Standards, Rept. NBSIR 76-1041, April 1976.

3. Hoffman, L.J., *Modern Methods for Computer Security and Privacy*, Prentice-Hall Inc., New Jersey, 1977.

4. Chin, J.S., "Analysis of Operating System Security," Lawrence Livermore Laboratories, December 2, 1975.

5. Linde, R.R., "Operating System Security," *Proceedings of National Computer Conference, 1975*, 1975, pp. 361—368. (U)

"DATA STANDARDS WITHOUT TEARS"

A COMMENT BY [redacted] P1

Much of what [redacted] says in "Data Standards Without Tears" has merit. The Data Dictionary concept can play a role in the standardization process, but not in the "magical" way he outlines. You can only have standards with *sweat* — without tears, perhaps, but certainly not without considerable labor. I am afraid that we have to indict [redacted] for not really giving due credit to the standardization process that the NDSC has long been pursuing, and also for presenting a few half-truths here and there along with the nuggets of wisdom.

~ "No one agrees that data standards should be enforced on his project at the expense of operational necessity."

Right. The NDSC has not tried to shut off anyone's job because of failure to observe standards. On paper we have the authority: both NSA Regulation 80-9 and USSID 414, "Standardization of Data Elements and Related Features for SIGINT Activities," Annex B ("Implementation of Standard Data Elements and Related Features in NSA/CSS Computer Projects") give us the *authority* to make life very unhappy for sponsors whose jobs ignore or conflict with published standards. In theory we can point to the concept of enforcement of data standards, even to the short-run disadvantage of a computer project. In actual practice, we sacrifice the long-term benefits to the Agency that would follow from a rigorous enforcement of the standards we already have.

~ "...we view standards as something which not only can be but must be imposed in an inflexible, hard-handed manner."

The Center never "imposes" standards in this way but issues them only after a long and rigorous process. This begins with a recognized need, research and discussion, drafting of a "proposal" etc., and continues with coordination through the Senior Data Representatives (SDR) of the DDO elements. There are draftings and redraftings to meet objections, suggestions, etc., and final approval comes, in many cases, only after a painfully long process. This is far from an "inflexible, hard-handed manner." A proposed standard always has wide circulation throughout the Agency.

~ "It goes without saying that [standards] cannot be achieved without some degree of magic. On the practical level the magic machine already exists for rendering coarse materials into fine standard gold..."

I guess a good name for this philosophy of standardization might be the "Rumplestiltskin Syndrome" — after the legendary gnome who was able to weave straw into gold to further his nefarious designs. Let us not accuse our good friends from the DED/D team of such plotting. Everyone would like to have the magic machine dispense usable and workable standards without going through the long and often painful process outlined above.

This philosophy is, I'm afraid, a naive one when viewed in the harsh light of the standardization process. I think I see what [redacted] is saying here, however. He is pointing out:

— the DED/D will expose people to the already-published standard data elements in the dictionary part of the system;

— the DED/D will show people, in the dictionary portion, what the current usage of data fields is along a wide spectrum of different Agency applications. Exposure to this usage will gradually lead us towards the necessary standardization. (The author of the essay does not explicitly state this, but this is my understanding of his concept.)

[redacted] goes on to separate the data features we deal with into two "domains" — Data Elements and Data Fields. I agree that this

---

*CRYPTOLOG, February 1979