UNIX Security in a Supercomputing Environment

*Matt Bishop**

Department of Mathematics and Computer Science
Dartmouth College
Hanover, NH 03755

ABSTRACT

The UNIX† operating system is designed for collaborative work and not for security. Vendors have modified this operating system (in some cases, radically) to provide levels of security acceptable to their customers, but the versions used in supercomputing environments would benefit from enhancements present in so-called secure versions. This paper discusses the need for security in a supercomputing environment and suggests modifications to the UNIX operating system that would decrease the vulnerability of those sites to attacks. Among the issues are additional auditing controls, changes to network programs, improved user authentication, and better application of the principle of least privilege.

1. Introduction

Because supercomputing facilities are used primarily for research and development, security considerations seem out of place or unwelcome additions that serve to hinder, not advance, the effectiveness of such sites. However, consideration of the nature of these facilities provides ample reasons for incorporating security mechanisms into the systems used. First are the many forms of theft. Supercomputer time is valuable, and running unauthorized programs is a tempting goal for unauthorized users. Because commercial, scientific, and governmental organizations use supercomputers for their most time-consuming problems, gaining illicit access to a supercomputer would provide a competitor, spy (industrial or otherwise), or other agent the ability to see what others are doing, steal their data and/or programs, and embarrass the users by prematurely exposing results. Second is the issue of a non-hostile working environment; malicious people may have less-than-useful goals in mind, such as the interruption of work by forcing the supercomputer to be shut down or removed from production mode for some period of time, or the prevention of users from accessing the supercomputer. In the worst case, they could alter data on the supercomputer, thereby causing valuable work to be delayed, misdirected, or ruined, or alter programs to corrupt data or other programs, as viruses do.

Since computer vulnerabilities are often transitive in that penetrating one system allows penetration of connected systems, should a malicious person (called an *attacker* or *cracker*†) break into a computer on which jobs are prepared for the supercomputer, that attacker could alter the job to give him or her access to the supercomputer; similarly, he or she could modify results obtained from jobs run on the supercomputer to hinder development work or research. The "supercomputing environment," includes the supercomputer and those hosts on which jobs (including research, development, and administrative functions) are either prepared for execution on a supercomputer, or on which the output of such a job is analyzed. Of course, not all such machines may be under the control of the supercomputing staff.

This brief survey paper critiques some security mechanisms in most versions of the UNIX operating system and suggests more effective tools that either have working prototypes or have been implemented, for example in secure UNIX systems. Although no computer (not even a secure one) is impenetrable, breaking into systems with these alternate mechanisms will cost more, require more skill, and be more easily detected, than penetrations of systems without these mechanisms.

The mechanisms described fall into four classes (with considerable overlap). User authentication at the local host affirms the identity of the person using the computer. The principle of least privilege dictates that properly authenticated users should have rights precisely sufficient to perform their tasks, and system administration functions should be compartmentalized; to this end, access control lists or capabilities should either replace or augment the default UNIX protection system, and mandatory access controls implementing multilevel security models and integrity mechanisms should be available. Since most users access supercomputing environments using networks, the third class of mechanisms augment authentication and access over a network, and provide applications layer encryption services and authentication (where feasible). As no security is perfect, the fourth class of mechanisms logs events that may indicate possible security violations; this will allow the reconstruction of a successful penetration (if discovered), or possibly the detection of an attempted penetration.

* The support of grant NAG 2-480 from the National Aeronautics and Space Administration to Dartmouth College is gratefully acknowledged.

† UNIX is a trademark of AT&T Bell Laboratories.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.
© 1989 ACM 089791-341-8/89/0011/0693 $1.50

† See [39] for a discussion of "cracker" versus "hacker."

2. Local User Authentication

Threat: Some responsible administrator must authorize a user to access facilities or resources within the supercomputing environment. When such a user accesses the facility or resource, his or her identity determines what he or she may do. Hence accurate identification or authentication of the user is vital.

Standard UNIX Authentication Mechanisms: The usual authentication mechanism is the password, which most UNIX systems allow users to select subject to (usually mild) constraints. To encourage users to change passwords periodically, some versions of the UNIX operating system provide a mechanism to expire passwords; others do not. However, should a password expire, the user is forced to change it at the next successful login because the system does not allow any command other than the password changing command to be used.

All standard versions of the UNIX operating system encrypt user passwords using a one-way function [6,27] and store the encrypted form in a world-readable file called the *password* file. It is not possible to determine who reads (or copies) this file.

An account may have no password; in this case, anyone may access that account; no password will be requested. On most systems, this requires a system administrator to reset a field in the password file.

Problems With the Standard Mechanisms: A good password is unguessable, easy to remember, not in a dictionary, and chosen from a set of possible passwords large enough to discourage exhaustive search of the set; it should also be changed periodically. Experiments run at both MIT and Dartmouth College [2,7,8] indicate that the current UNIX scheme is not adequate, since approximately 30% of users' passwords were guessed. Having the computer generate random passwords is not much better, since users will then write the passwords down, especially when they must remember passwords for many different hosts. Generating passwords according to a set of rules works reasonably well (a common method is to make the passwords pronounceable but meaningless), but if users are assigned different passwords for a large number of hosts they will still often write them down.

Implementations of a password aging mechanism should not require the user to think of a new password instantly; as Grampp and Morris [15] point out, "the most incredibly silly passwords tend to be found on systems equipped with password aging." Further, if a user may change passwords at any time, one need only change them to satisfy the password aging mechanism, and then can change them back. If the mechanism prevents a user from changing passwords more than once within a specified interval of time, the user may not be able to change a poorly-chosen or compromised password. So implementation of password aging schemes must avoid these risks.

Since the password encryption function is one-way, it appears that obtaining the encrypted form reveals nothing about the password. Since the password encryption routines are available in the system library, an attacker can test possible passwords by encrypting them and comparing the results to those stored in the file. Such "password cracking" is actually quite common, far simpler and more difficult to detect than repeatedly trying to log in.

Accounts with no passwords usually run programs with limited capabilities (such as printing the date or listing users of the system) rather than the command interpreter. Unless there is a very specific administrative reason for such an account, accounts without passwords should not exist, especially in an installation where resources are as precious as in a supercomputing center.

Alternatives to the Standard UNIX Mechanisms: Except where it affects the quality of the password, the source of the password (computer or user) is not relevant†. Simple modifications to the standard UNIX password changing program can check the proposed password against the user's name, account name, and various dictionaries and lists of common character combinations (such as acronyms and names not in any dictionaries); this satisfies criterion 2, and makes meeting criterion 1 more likely than if the checks were omitted. As an alternative, if the computer is to generate passwords, the passwords can be combinations of pronounceable (but meaningless) syllables separated by non-letter characters; users may be presented with a set of potential passwords from which they may select one, or request another set. If the syllables were generated from a rich enough set, this would meet criteria 1 and 2. Both schemes are used in various secure versions of the UNIX operating system (see for example [14,20]).

Secure systems which generate passwords for users sometimes use password aging mechanisms [11]. However, when users are to supply passwords, the aging mechanism should warn them before their password expires to allow time to think of another one.

Many secure versions of the UNIX operating system use a "shadow password file" [14,18,26] in which the encrypted password is placed in an alternate unreadable file, prevents password cracking without attempting to log in; since all systems should record failed login attempts, this increases visibility of password crackers. Modifications to the login program enable the security officer to allow the login but to a restricted account (to allow security officers to trace the connection [16,39]), or to disable the attacked account.

Other authentication methods may be used in place of, or in conjunction with, passwords. The *challenge-response protocol* [38] requires the computer to generate some number or string (the challenge), and the user to perform some operation on it (the response). The challenge varies from instance to instance, so even if an attacker obtains one challenge and its corresponding response, subsequent challenges will be different; to prevent the relationship between challenges and responses from being deduced, the response is usually the output of some cryptographic function of the challenge. This usually requires some physical hardware (such as a calculator-like device) to obtain the response, so both knowing another user's password and access to the appropriate hardware are necessary for impersonation; further, the challenge is issued from a satellite computer that is both physically secure and inaccessible to users of the main computer. Pass algorithms [17] work similarly, but implementations do not use external devices, so if the algorithm is not changed frequently it may be possible to deduce the relationship between the challenge and the response. Although in their infancy, biometric techniques may be used to authenticate

† The impact may be very subtle. In [27], Robert Morris and Ken Thompson describe a random password generator that appeared to choose passwords from a set of 36^8 possible passwords. Trying each possible password would take 112 years. Unfortunately, it did so by generating characters sequentially using a pseudorandom number generator, so the actual number of potential passwords was 2^{15} – and trying all of those would have taken only 42 minutes!

individuals [25,28] but some of these techniques are subject to replay attacks.

Recommendations: UNIX systems in supercomputing environment should use a two-tier method of authentication that cannot be disabled. The first is the password; either users should be allowed to select their own passwords, which are checked before being accepted by the computer, or the computer should generate a set of possible passwords and allow the user to select one (or request another set). The encrypted passwords should be kept in a shadow password file, and password aging should be used. The second tier uses another authentication method to require the user to demonstrate that he or she is not an intruder who knows the real user's password, such as a challenge-response protocol requiring an external device to obtain the reponse from the challenge. Users who fail to give a correct account name and password at the first level should still be forced to complete the second, so they cannot determine whether the failure was a bad account name, an invalid password, or an erroneous response. Further, some action deemed appropriate by the site security officer should be taken for repeated failures to log in to a specific account.

3. Access Control, Integrity, and Least Privilege

Threats: Access control mechanisms regulate the ability to access resources such as files, devices, and processes. If the access control mechanisms are inadequate, users may be able to modify or alter data they should not have access to, or may not be able to access data they are authorized to.

Standard UNIX Access Control Mechanisms: Standard UNIX systems use a simplification of the traditional access control list. Each object is assigned a *user* and a *group* identification (typically, the user identification is that of the owner of the object, and the group identification is either that of the owner or of some related object). Three sets of permissions are associated with each object, one corresponding to the user (owner), one to the group, and one to all others; each set consists of distinct read, write, and execute permissions. When someone tries to access the object, if that person has the same user identification as the object, the user permissions are checked; if not, and if that person has the same group identification as the object, the group permissions are checked; otherwise, the set of permissions for all others is checked. No access control checking of any kind is done to the privileged user *root* (also called the *superuser*).

The *setuid* mechanism enables processes to assume privileges of another user. For example, the program to read electronic mail may write a log record into a file in one of two ways. Either the file must be writable by all users (meaning they can change the log), or the log file must be writable only by its owner, and the mail reading program must be setuid to the owner of the log file. Whenever that program is executed, the UNIX operating system will change its user identification to that of the owner of the log file. A similar mechanism (called *setgid*) works for altering the group identification of a running program (and actually would be used in the above example).

Many automatic tools ensure consistency of the UNIX file system and repair inconsistencies when found.

Problems with the Standard Mechanism: The first problem is the lack of granularity in access control. To grant access to a file only to some subset of users of the system is either to make that subset a group (which requires a system administrator) and set the file's group identification appropriately, or to give the users in that subset access to an account by which the file is owned. Neither method is acceptable in practise. Hence when a subset of users of the system must access a file, its owner usually gives that access permission to all users. Thus the mechanism fails to support the principle of least privilege [36].

The existence of an omnipotent user also violates this principle, because once an attacker has gained access to that account he or she can do anything. The setuid mechanism makes access to other accounts all too easy if not programmed with care, because programs using that mechanism often fail to do adequate checking.

The only mechanism to check the integrity of files is a checksumming program that is not cryptographic and hence easy to evade. Further, the checksum program must be run manually.

Alternatives to the Standard UNIX Mechanisms: Some secure versions of the UNIX operating system implement standard access control lists, either providing them as an alternative to the simplification used by standard UNIX systems [14] or replacing the standard mechanism entirely [20]. They also provide mandatory access controls along the lines of the Bell-LaPadula model [4]: each object is assigned a *security level* and a set of *compartments*. A subject may read (or execute) an object at a lower or equal security level if the object's set of compartments is a subset of the subject's set of compartments, and the discretionary access controls on the object allow the action. A subject may write (or delete) an object at a higher or equal security level if the object's set of compartments is a subset of the subject's set of compartments, and the discretionary access controls on the object allow the action. (How this is done varies from system to system.) By judicious selection of security levels, sets of compartments, and setting of the access control lists, users (and the system administrator) can control access to any granularity required, thus these mechanisms enable enforcement of the principle of least privilege.

Almost all versions of secure UNIX have modified or eliminated the notion of the superuser, especially by partitioning the superuser functions among several users [11,14,20]. In some cases these users are not permitted to log in, but must act in single-user mode; in others, they must access restricted command interpreters to perform their function. This eliminates the problem of system administrators logging in as the superuser and acting in ways that do not require privileges.

Some systems restrict the setuid privilege in various ways such as by removing it when a setuid file is written to, or requiring the use of a trusted path to activate the privilege (this prevents a Trojan horse from creating a setuid file and thereby steal privileges). This does not eliminate the danger of a careless programmer causing problems, but such danger is ameliorated by the restrictions the mandatory access controls place on the setuid program. Other versions simply replace setuid programs with trusted servers [26].

In the wake of computer viruses, integrity control has become prominent. Several systems implement aspects of Biba's integrity model [5], in which each object is assigned an *integrity level* and a set of *integrity compartments* (these may be different than the corresponding levels and compartments for the mandatory access control mechanism). Subjects may read (or execute) objects at a higher or equal integrity level, if the object's set of integrity compartments is a subset of the subject's set of integrity compartments, and the discretionary access controls on the object allow the action; a subject may alter (or delete) objects at a lower or equal integrity level if the object's set of integrity compartments is a subset of the

subject's set of integrity compartments, and the discretionary access controls on the object allow the action. The system administrators are free to use, or not use, any of the implemented aspects.

Recommendations: The discretionary access control mechanism should be access control lists, perhaps with the standard simplification being used should the user attempting access not be named in the list associated with the object. The superuser functions should be split over multiple accounts (each with less privilege), and the system should use a mandatory access control mechanism to limit the effects of penetrations. If setuid and setgid programs are to be present, they should lose their privileged setting when any process writes to them. The setuid and setgid privilege should be granted only through a trusted path, so no user would unknowingly surrender privileges. Finally, a multilevel integrity mechanism should be in place. However, the aspects of the multilevel mechanism to be used should be at the discretion of the system administrator, and he or she should be able to activate (or deactivate) them on a per-user basis†.

4. Network Privacy and Authentication

Threats: As most users communicate with supercomputing environments over networks, authenticating their identity and providing tools for private communication with the computer and with other users enables accurate identification of those entitled to use specific resources and allows work done on the supercomputer, and its results, to be private. The very general problem of authorization and secrecy in networks is discussed elsewhere [21,40]; the discussion here is confined to tools specific to the UNIX operating system and to the 4.2 and 4.3 Berkeley Software Distribution's user-level network tools.

Standard Berkeley UNIX Network Tools:

Network software based on the Berkeley 4.2 distribution allows users to designate specific users on specific remote hosts as "trusted;" when someone tries to log in remotely, if the user and remote host are trusted no password is requested. Similarly, the system administrator may also designate hosts as "trusted" for all users, so if a user has an account with the same name on a remote host which the system administrator has listed as trusted, no password will be requested. Once logged in, the remote user is treated just like a local user; that is, no additional constraints are applied.

The only encryption program on standard UNIX systems implements "a one-rotor machine along the lines of the German Enigma, but with a 256-element rotor" [9].

Electronic mail allows users to correspond with one another, and is available on all standard UNIX systems.

Problems With the Standard Mechanisms: Permitting users (or system administrators) to disable password checking is a useful convenience, but it can have catastrophic consequences unless site administration is very strict [35]. All hosts that trust a compromised computer are themselves compromised, because access to the latter implies access to all the former. Further, if different security and integrity levels and compartments are supported, the user's security clearance should be a function of the host from which he or she is connecting to the supercomputing environment.

Methods of breaking the standard UNIX encryption program are very widely known [34].

Electronic mail as supplied provides neither privacy nor authentication. While the body of letters may be encrypted using the program described above, the result must be expanded (to seven bit characters) to comply with the internet mail protocol [33]. Further, it is not possible to determine whether the named sender did in fact send the letter without an out-of-band mechanism (such as a telephone call.)

Alternatives to the Standard Berkeley UNIX Network Tools: Many sites only allow system administrators to designate hosts as trusted. Others allow users to designate those hosts under the administration's control as trusted, but no others (for example, at the NAS Project, the computer "prandtl.nas.nasa.gov" could be trusted, but "icefloe.dartmouth.edu" could not, since it is not under NAS control). Still others restrict this ability to unprivileged users; administrative or superuser accounts may not use this feature.

Many sites have implemented alternate encryption programs, most of them based on the Data Encryption Standard [29,30]. These are believed to be considerably harder than the UNIX *crypt*(1) program to break; in any case, quick methods of breaking them are not widely known.

Currently several RFCs describe a proposed standard for privacy-enhanced electronic mail, and prototype implementations have successfully exchanged secure, authenticated mail messages [19,22-24]. Problems include a lack of available software and no existing supporting key management structure; Development of both is currently in progress.

Recommendations: The "trusted host" mechanism described above should be eliminated. Instead, all remote connections should require using the local user authentication mechanisms; further, if different security or integrity levels and compartments are supported, system administrators should be able to restrict the levels and compartments which users may access based on the identity of the remote computer.

The UNIX encryption program *crypt*(1) should be scrapped and implementations of better cryptosystems supplied, and when a suitable key management scheme is adopted, secure electronic mail should be made available to the user community.

5. Logging and Auditing

Threat: Since no system is completely secure, at some point a successful penetration may take place. Records of unsuccessful penetration attempts may provide information useful to system security or to determining if specific accounts have been compromised.

Standard UNIX Logging and Auditing Facilities: Most versions of the UNIX operating system log connect time and job execution for accounting purposes; specific programs (such as daemons) log changes of account, connections from other sites, and failed login attempts. There is no unified logging and auditing scheme, however, so the information logged varies from program to program.

A checksumming program exists on standard versions of the UNIX operating system.

Problems with the Standard Mechanisms: Although adequate for accounting purposes, the lack of unity in the design of logging and auditing mechanisms make them inadequate for secu-

† The rule against "reading down" would most likely not be applied to all users because, as Gasser points out, it "is probably more suited to containing errors" than threats to security [13, p. 70]. But the rule against "writing up" would be.

rity monitoring purposes, since critical files (such as the password file) can be altered without a trace, and information is kept on the computer where it may be altered or erased by nonexpert attackers.

There is no scheme to check the integrity of system files automatically. Further, since the checksum program is not cryptographic, changes to files can be made transparent to it.

Alternatives to the Standard UNIX Mechanisms: Secure versions of the UNIX operating system provide extensive logging and auditing facilities [1,11,14,18,20,26]. These facilities log events such as file accesses including, among other things, attempted file accesses, process creation and termination, and file creation and deletion. Each log entry should contain the user identification, relevant process or file identification information, the event being logged, and information about that event. Entries for file accesses should identify the user and process (command) requesting the access, how access is desired (read, write, execute, change something in the inode, etc.), the wall clock time of the attempt, and the disposition (was access granted or not). Entries for process creation and deletion should indicate the user, the wall clock time, the file executed, the disposition (was the execution begun), and the process from which execution was attempted. This information should be stored off-line on a computer not accessible to users of the monitored computer and not connected to a network; this *audit computer* should also have tools capable of analyzing the data and printing it in readable form.

A mechanism to mark files as critical to the system and not changeable should be available, and these files should be checksummed periodically (or randomly) using some cryptographic checksumming scheme. This can then be checked against precomputed checksums; again, this should be done on the audit machine mentioned above.

Monitoring all users and all file accesses may be quite expensive, so the system administrator should have the ability to enable the logging facilities for specific users, files, and events. For example, the password file should be monitored since it is so critical to the security of the system, but transient files produced by a compiler need not be monitored. If done on a selective basis, this type of monitoring should not adversely impact system performance [32].

More sophisticated monitoring tools may aid in the detection of successful or attempted penetrations. Intrusion detection systems [12,25] analyze statistical characteristics of users, or apply rules gleaned from earlier audit records, to detect anomalies in system use; since these may indicate an intruder, the intrusion detection system reports these to a systems administrator. Prototypes cause little or no change in the monitored system's response time [25,37], and at least one prototype has been developed for UNIX systems [3]. Preliminary results indicate that such a tool appears to be useful in detecting penetrations.

Recommendations: UNIX systems used in a supercomputing environment should have extensive logging capabilities of the nature outlined above built into their kernels. Ideally, all events should be logged; since the intrusion detection model postulates the availability of such information, they can be integrated into the system with a minimum of changes when in the future implementations become available. The ability to activate logging when an attack or penetration is suspected will enable system administrators to examine one specific area of risk without burying themselves (or the audit machine) in audit records; they can then determine what should be audited. A mechanism to check specific files for unauthorized alterations should also be present.

6. Conclusion

This paper presented a brief overview of some security mechanisms in conventional UNIX systems, discussed their deficiencies, described tools that improve or augment their effectiveness, and recommended changes to increase the difficulty of an attacker trying to penetrate undetectably a computer within the supercomputing environment. Supercomputer manufacturers offering versions of the UNIX operating system are aware of these needs, and at least one supplies a system enhanced with many of the features described above [10].

Many of the recommended tools are similar to, or identical to, parts of commercial secure versions of the UNIX operating system designed to meet criteria outlined in the *Trusted Computer System Evaluation Criteria* [31]. However, those criteria should be followed only if dictated by site (or administrative) policy. Instead, administrators of supercomputing environments should implement those recommendations they believe will be most effective within their specific environment. In this way they can achieve that delicate balance between security and the user-friendliness their user community requires.

References

1. Addison, K., Baron, L., Copple, M., Cragun, D., Hospers, K., Jordan, P., Lechner, M., Manley, M. and Schaufler, C., "Computer Security at Sun Microsystems, Inc.", *Tenth National Computer Security Conference Proceedings*, 216-219 (September 1987).
2. Baldwin, R., private communication, July 1987.
3. Bauer, D. S. and Koblentz, M. E., "NDIX – A Real-Time Intrusion Detection Expert System", *Proceedings of the Summer 1988 USENIX Conference*, 261-273 (June 1988).
4. Bell, D. and LaPadula, L., "Secure Computer Systems: Unified Exposition and MULTICS Interpretation", Technical Report MTR-2997, MITRE Corporation, Bedford, MA, July 1975.
5. Biba, K. J., "Integrity Considerations for Secure Computer Systems", ESD-TR-76-372 (NTIS AD-A0393324), Air Force Electronic Systems Division, Hanscom Air Force Base, MA.
6. Bishop, M., "An Application of a Fast Data Encryption Standard Implementation", *Computing Systems*, *1*, 3 (Summer 1988) 221-254.
7. Bishop, M. and Bryant, E., "Is Your Password Obvious?", *Kiewit Comments*, *19*, 4 Dartmouth College (February-March 1988) 1.
8. Bishop, M., "Bishop Continues UNIX Security Research", *Kiewit Comments*, *20*, 4 Dartmouth College (February-March 1989) 2.
9. *UNIX User's Reference Manual, 4.3 Berkeley Software Distribution, Virtual VAX-11 Version*, Computer Systems Research Group, Computer Science Division, EECS, University of California, Berkeley, Berkeley, CA 94720, November 1986. as reprinted by the USENIX Association.
10. "UNICOS Security Administration Reference Manual", SR-2062A, Cray Research, Inc., Mendota Heights, MH, March 1989.

11. Cummings, P. T., Fullam, D. A., Goldstein, M. J., Gosselin, M. J., Picciotto, J., Woodward, J. P. L. and Wynn, J., "Compartmented Mode Workstation: Results Through Prototyping", *Proceedings of the 1987 Symposium on Security and Privacy*, 2-12 (April 1987).
12. Denning, D. E., "An Intrusion-Detection Model", *IEEE Transactions on Software Engineering, SE13*, 2 (February 1987) 222.
13. Gasser, M., *Building a Secure Computer System*, Van Nostrand Reinhold Company Inc., New York City, NY, 1988.
14. Gligor, V. D., Chandersekaran, C. S., Chapman, R. S., Dotterer, L. J., Hecht, M. S., Jiang, W., Johri, A., Luckenbaugh, G. L. and Vasudevan, N., "Design and Implementation of Secure Xenix", *IEEE Transactions on Software Engineering, SE-13*, 2 (February 1987) 208-221.
15. Grampp, F. T. and Morris, R. H., "The UNIX System: UNIX Operating System Security", *AT&T Bell Laboratories Technical Journal, 63*, 8, part 2 (October 1984) 1649-1672.
16. Hanson, S. and Eldredge, M., "Intruder Isolation and Monitoring", *Proceedings of the UNIX Security Workshop*, Portland, OR, 63-64 (August 1988).
17. Haskett, J. A., "Pass-Algorithms: A User Validation Scheme Based on Knowledge of Secret Algorithms", *Communications of the ACM, 27*, 8 (August 1984) 777-781.
18. Hecht, M. S., Johri, A., Aditham, R. and Wei, T. J., "Experience Adding C2 Security Features to UNIX", *Proceedings of the Summer 1988 USENIX Conference*, 133-146 (June 1988).
19. Kent, S. and Linn, J., "Privacy Enhancement for Internet Electronic Mail: Part II -- Certificate-Based Key Management", RFC 1114, August 1989.
20. Kramer, S., "Linus IV – An Experiment in Computer Security", *Proceedings of the 1984 Symposium on Security and Privacy*, 24-32 (April 1984).
21. Leiner, B. M., "Policy Issues in Interconnecting Networks", Technical Report 89.25, Research Institute for Advanced Computer Science, Moffett Field, CA, June 1989.
22. Linn, J., "Privacy Enhancement for Internet Electronic Mail: Part I -- Message Encipherment and Authentication Procedures", RFC 1113, August 1989.
23. Linn, J. and Kent, S. T., "Privacy for DARPA-Internet Mail", *Twelfth National Computer Security Conference Proceedings*, (to appear) (1989).
24. Linn, J., "Privacy Enhancement for Internet Electronic Mail: Part III -- Algorithms, Modes, and Identifiers", RFC 1115, August 1989.
25. Lunt, T. F., "Automated Audit Trail Analysis and Intrusion Detection: A Survey", *Eleventh National Computer Security Conference Proceedings*, 65-73 (October 1988).
26. Miller, G., Sutton, S., Matthews, M., Yip, J. and Thomas, T., "Integrity Mechanisms in a Secure UNIX: Gould UTX/32S", *AIAA/ASIS/DODCI Second Aerospace Computer Security Conference*, 19-26 (December 1986).
27. Morris, R. and Thompson, K., "Password Security: A Case History", *Communications of the ACM, 22*, 11 (November 1979) 594-597.
28. "Guidelines on Evaluation of Techniques for Automated Personal Identification", FIPS PUB 48, National Bureau of Standards, Washington, DC, April 1977 .
29. "Data Encryption Standard", FIPS PUB 46, National Bureau of Standards, Washington, DC, January 1977.
30. "DES Modes of Operation", FIPS PUB 81, National Bureau of Standards, Washington, DC, 1981.
31. "Trusted Computer System Evaluation Criteria", DOD 5200.28-STD, National Computer Security Center, Fort Meade, MD, December 1985.
32. Picciotto, J., "The Design of an Effective Auditing Subsystem", *Proceedings of the 1987 Symposium on Security and Privacy*, 13-22 (April 1987).
33. Postel, J., "Simple Mail Transfer Protocol", RFC 821, August 1982.
34. Reeds, J. A. and Weinberger, P. J., "The UNIX System: File Security and the UNIX System Crypt Command", *AT&T Bell Laboratories Technical Journal, 63*, 8, part 2 (October 1984) 1673-1683.
35. Reid, B., "Reflections on Some Recent Widespread Computer Break-ins", *Communications of the ACM, 30*, 2 (February 1987) 103-105.
36. Saltzer, J. and Schroeder, M., "The Protection of Information in Computer Systems", *Proceedings of the IEEE, 63*, 9 (September 1975) 1278-1308.
37. Sebring, M. M., Shellhouse, E., Hanna, M. E. and Whitehurst, R. A., "Expert Systems in Intrusion Detection: A Case Study", *Eleventh National Computer Security Conference Proceedings*, 74-81 (October 1988).
38. Spender, J., "Computer Security and User Authentication: Old Problems, New Solutions", *AIAA/ASIS/DODCI Second Aerospace Computer Security Conference*, 126-132 (December 1986).
39. Stoll, C., "Stalking the Wily Hacker", *Communications of the ACM, 31*, 5 (May 1988) 484-497.
40. Voydock, V. L. and Kent, S. T., "Security Mechanisms in High-Level Network Protocols", *ACM Computing Surveys, 15*, 2 (June 1983) 135-171.