UNIX and Security: The Influences of History*

Eugene H. Spafford
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907–1398
spaf@cs.purdue.edu

Revised January 1992
Revised April 1994

Abstract

UNIX has a reputation as an operating system that is difficult to secure. This reputation is largely unfounded. Instead, the blame lies partially with the traditional use of UNIX and partially with the poor security consciousness of its users. UNIX's reputation as a nonsecure operating system comes not from design flaws but from practice. For its first 15 years, UNIX was used primarily in academic and computer industrial environments — two places where computer security has not been a priority until recently. Users in these environments often configured their systems with lax security, and even developed philosophies that viewed security as something to avoid. Because they cater to this community, (and hire from it) many UNIX vendors have been slow to incorporate stringent security mechanisms into their systems.

This paper describes how the history and development of UNIX can be viewed as the source of many serious problems. Some suggestions are made of approaches to help increase the security of your system, and of the UNIX community.

1 Introduction

UNIX[1] security is often described a an oxymoron — like "instant classic" or "military intelligence." Contrary to prevailing opinion, however, UNIX can be a moderately secure operating system. Today, after two decades of development and modification — much of it motivated by a desire for improved security — UNIX is perhaps the best understood security-conscious operating system in general use.

One of the keys to understanding UNIX's bad reputation is to understand something of its history. UNIX was developed in an environment much different from most commercial operating systems, and this history is one of the chief sources of UNIX weaknesses.

* Earlier versions of this paper were presented as the keynote address at the 1991 Austrian UNIX Users Group annual conference, Vienna, Austria, and under the title *Making* UNIX *Secure* as [17].

[1] UNIX is a registered trademark of Novell. This week.

2 History[2]

2.1 The Beginning

The roots of UNIX go back to the mid-1960s, when American Telephone and Telegraph, Honeywell, General Electric, and the Massachusetts Institute of Technology embarked on a massive project to develop an information utility. The project, called Multics (standing for *Mult*iplexed *I*nformation and *C*omputing *S*ervice), was heavily funded by the Department of Defense Advanced Research Projects Agency (then known as ARPA, now known as DARPA). Most of the research took place in Cambridge, Massachusetts, at MIT.

Multics was a modular system built from banks of high-speed processors, memory, and communications equipment. By design, parts of the computer could be shut down for service without affecting other parts or the users. The goal was to provide computer service 24 hours a day, 365 days a year — a computer that could be made faster by adding more parts, much in the same way that a power plant can be made bigger by adding more furnaces, boilers, and turbines.

Multics was also designed with military security in mind. Multics was designed both to be resistant to external attacks and to protect the users on the system from each other. Multics was supposed to support the concept of *multi-level* security: Top Secret, Secret, Confidential, and Unclassified information could all coexist on the same computer. The Multics operating system was designed to prevent information that had been classified at one level from finding its way into the hands of someone who had not been cleared to see that information.

In 1969, Multics was far behind schedule; its creators had promised far more than they could deliver within the projected timeframe. Already at a disadvantage because of the distance between its New Jersey laboratories and MIT, AT&T decided to pull out of the Multics Project.

That year Ken Thompson, an AT&T researcher who had worked on the Multics Project, took an unused PDP–7 computer at Bell Labs to continue pursuing some of the Multics ideas on his own. Thompson was soon joined by others, including Dennis Ritchie, who had also worked on Multics. Peter Neumann gets the credit for suggesting the name UNIX for the new system — a pun on the name Multics and a backhanded slap at the project that was continuing in Cambridge (and indeed continued for another decade and a half). Whereas Multics tried to do many things, UNIX tried to do one thing well: run programs. The concept of high security was not part of this original goal.

The smaller scope was all the impetus that the researchers needed; an early version of UNIX was operational several months before Multics. Within two years Thompson, Ritchie and others rewrote UNIX for Digital's new PDP–11 computer.

As AT&T scientists added features to their system throughout the 1970s, UNIX evolved into a programmer's dream. The system was based on compact programs, called tools, each of which performed a single function. By putting tools together, programmers could do complicated things. UNIX mimicked the way programmers thought. In order to get the full functionality of the system, users needed access to all of these tools. Furthermore, the environment was such that as the system evolved, nearly everyone with access to the machines aided in the creation of new tools and in the debugging of existing tools.

[2] Portions of this section are taken from [6] and [14].

In 1973, Thompson rewrote UNIX in Ritchie's newly invented C programming language. C was designed to be a simple, portable language. Programs written in C could be moved easily from one kind of computer to another — as was the case with programs written in other high-level languages like FORTRAN — yet they ran nearly as fast as programs coded directly in a computer's native machine language.

At least, that was the theory. In practice, every different kind of computer at Bell Labs had its own operating system. C programs written on the PDP-11 could be recompiled on the lab's other machines, but they didn't always run properly, because every operating system performed input and output in slightly different ways. Mike Lesk developed a "portable I/O library" to overcome some of the incompatibilities, but many remained. Then, in 1977, the group realized that it might be easier to port the UNIX operating system itself rather than trying to port all of the libraries. UNIX was first ported to the lab's Interdata 8/32, a micro-computer similar to the PDP-11. In 1978, the operating system was ported to Digital's new VAX minicomputer. However, it still remained very much an experimental operating system.

2.2 Outside Bell Labs

UNIX had become a popular operating system in many universities and was already being marketed by several companies. UNIX had become more than just a research curiosity. As early as 1973, there were over 16 different AT&T or Western Electric sites outside Bell Labs running the operating system. UNIX soon spread even further.

Thompson and Ritchie presented a paper on the operating system at the ACM Symposium on Operating System Principles in October, 1973. Within a matter of months, sites around the world had obtained and installed copies of the system. Even though AT&T was forbidden under the terms of its 1956 Consent Decree with the US Federal government from advertising, marketing, or supporting computer software, demand for UNIX steadily rose. By 1977, more than 500 sites were running the operating system; 125 of them were at universities, and more than 10 foreign countries. 1977 also saw the first commercial support for UNIX, then at Version 6.

In the universities, the typical UNIX environment was like that inside Bell Labs: the machines were in well-equipped labs with restricted physical access. The users who made extensive use of the machines were people who had on-going access, and who usually made significant modifications to the operating system and its utilities to provide additional functionality. They did not need to worry about security on the system because only authorized individuals had access to the machines. In fact, implementing security mechanisms often hindered the development of utilities and customization of the software. I worked in two such labs in the early 1980s, and in one location we viewed having a password on the root account as an annoyance because everyone who could get to the machine was authorized to use it as the super-user!

This environment was perhaps best typified by the development at the University of California at Berkeley. Like other schools, Berkeley had paid $400 for a tape that included the complete source code to the operating system. But instead of merely running UNIX, two of Berkeley's bright graduate students, Bill Joy and Chuck Haley, started making modifications. In 1978, Joy sent out 30 free copies of the "Berkeley Software Distribution," a collection of programs and modifications to the UNIX system.

Over the next six years, in an effort funded by DARPA, the so-called BSD UNIX grew into an operating system of its own that offered significant improvements over AT&T's. For example, a programmer using BSD UNIX could switch between multiple programs running at the same time. AT&T's UNIX allowed the names on files to be only 14 letters long, but Berkeley's allowed names of up to 255 characters. Berkeley also developed software to connect many UNIX computers together using high-speed networks. But perhaps the most important of the Berkeley improvements was the 4.2 UNIX networking software, which made it easy to connect UNIX computers to *local* area networks. (Note the stress on *local.*) For all of these reasons, the Berkeley version of UNIX became very popular with the research and academic communities.

About the same time, AT&T had been freed from its restrictions on developing and marketing source code as a result of the enforced divestiture of the phone company. Executives realized that they had a strong potential product in UNIX, and they set about developing it into a more polished commercial product.

2.3 Today

Today versions of UNIX are running on several million computers world-wide. Versions of UNIX run on nearly every computer in existence, from IBM PCs to Crays. Because it is so easily adapted to new kinds of computers, UNIX has been the operating system of choice for many of today's high-performance microprocessors. Because the operating system's source code is readily available to educational institutions, UNIX has also become the operating system of choice for educational computing at many universities and colleges. It is also popular in the research community because computer scientists like the ability to modify the tools they use to suit their own needs.

UNIX has become popular, too, in the business community. In large part this is because of the increasing numbers of people who have studied computing using a UNIX system, and they have sought to use UNIXin their business applications. Users who become very familiar with UNIX become very attached to the openness and flexibility of the system.

Furthermore, a standard for a UNIX-like operating system interface (POSIX) has emerged, although considerable variability remains. It is now possible to buy different machines from different vendors, and still have a common interface. UNIX is based on many accepted standards, and this greatly increases its attractiveness as a common platform base. It is arguable that UNIX is the root cause of the "open systems" movement: without UNIX, the very concept might not have been accepted by so many people as possible.

3 Problems

This evolution of UNIX has led to many problems. These can be classified into three categories:

- Problems of user expectation.
- Problems of software quality.
- Problems of add-on integration.

The following sections discuss each of these in some more detail.

3.1 User Expectation

Users have grown to expect UNIX to be configured in a particular way. Their experience with UNIX has always been that they have access to most of the directories on the system, and that they have access to most commands. Users are accustomed to making their files world-readable by default. Users are also often accustomed to being able to build and install their own software, often requiring system privileges to do so.

Unfortunately, all of these expectations are contrary to good security practice. To have stronger security, it is often necessary to curtail access to files and commands other than what are strictly needed for users to do their jobs. Thus, someone who needs e-mail and a text processor for his work should not also expect to be able to run the network diagnostic programs and the C compiler. Likewise, to heighten security, it is not wise to allow users to install software that has not been examined and approved by an authorized individual.

The tradition of open access is strong, and is one of the reasons that UNIX has been attractive to so many people. Some users argue that to restrict these kinds of access would somehow make the systems something other than UNIX. Perhaps this is so, but in instances where strong security is required, such measures may need to be taken.

At the same time, it is possible to strengthen security by applying some general principles, although not in the extreme possible. For instance, rather than removing all compilers and libraries from each machine, they can instead be protected so that only users in a certain user group may access them. Users with a need for such access, and who can be trusted to take due care, will be allowed in this group. Similar methods can be used with other classes of tools, too, such as network monitoring software.

Furthermore, it may be helpful to change the fundamental view of data on the system: from readable by default to nonreadable by default. For instance, user files and directories should be protected against read access instead of open by default. Setting `umask` values appropriately and using adjunct password files are just two examples of how this attitude can affect the system configuration.[6]

The most critical aspect here is that the users themselves participate in the alteration of their expectations. The best way to do this is not by decree, but through education — the users probably have not needed to use the system in an environment where threat is present and the value of lost information is great. By educating users in the dangers and how their cooperation can help thwart those dangers, the security of the system cannot help but be increased.

3.2 Software Quality

Many of the UNIX utilities people take for granted were written as student projects, or as quick "hacks" by software developers inside research labs. They were not formally designed and tested. Instead, they were put together and debugged on-the-fly. The result is a large collection of tools that usually work, but sometimes fail in spectacular manners. The utilities are not the only things written by students. Much of BSD UNIX, including the networking code, was written by students as research projects of one sort or another.

This is not intended to cast aspersions on the abilities of students, certainly, but to point out that UNIX was not created as a carefully-designed and tested system. Instead, many of the significant features and tools have been developed using ad-hoc techniques. It is no wonder that occasionally there will be bugs discovered that compromise the security of the system; it is perhaps a wonder that so few are evident!

Unfortunately, two things are not occuring as a result of the discovery of bugs. The first thing that does not seem to be happening is that people are not learning from past mistakes. For instance buffer overruns have been a known problem for some time, yet software continues to be written and discovered containing such bugs.

A second, and more serious problem is that few, if any, vendors are performing any organized program of testing on the utilities they provide. With over $\frac{1}{3}$ of the utilities on some machines being buggy ([9]), one might think that vendors would be eager to test their versions of the software and correct lurking bugs. However, as one vendor's software engineer told me, "The customers want their UNIX — including the bugs — exactly like every other implementation."

So long as the customers demand strict conformance of behavior by existing versions of the programs, and so long as software quality is not made a fundamental issue by customers, it seems unlikely that many vendors will make any concerted effort to test and fix their software. The existence of even de-facto standards that include weaknesses or bugs further complicates the issue. Formal standards, like the ANSI C standard and POSIX standard help perpetuate and formalize bugs, too. For instance, the ANSI C standard[3] perpetuates the `gets` call that does no bound checking on input; this was one of the methods exploited by the Internet Worm program, among others.[16, 15]

3.3 Add-On Functionality

The final category involves the way new functionality has been added to UNIX over the years. UNIX is often cited for its flexibility and reuse characteristics, so it is no surprise that new functions are constantly built on top of UNIX platforms, and eventually integrated into released versions. Unfortunately, the addition of new features is often done without understanding the assumptions that were made with the underlying mechanisms.

For instance, the concept of user-ids and group-ids controlling access to files was developed at a time when the typical UNIX machine was in a physically secure environment. On top of this was added remote manipulation commands such as `rlogin` and `rcp` that tried to maintain the user-id/group-id paradigm by using the concept of "trusted ports" for network connections. Within the local network at Berkeley, using only VAX-class machines, this (usually) worked well. But now, with the proliferation of workstations and non-UNIX machines on international networks, such assumptions usually lead to major weaknesses in the security mechanism.[11, 18] This does not even begin to consider what happens when a higher-level system such as NFS is built upon the already existing, unsecure layers!

[3] ANSI X3J11

It is both a tribute to Unix, and a condemnation, that such difficulties arise. It is a tribute to the robust nature of Unix that it can accept and support new applications by building on the old. It is in a sense also a condemnation that the existing mechanisms are sometimes completely inappropriate for the tasks assigned to them. Actually, it is more an indictment of the developers for failing to consider the security ramifications of building on the existing foundation; it should be noted than not all of those foundations were laid by Unix developers, either — TCP/IP, for instance, was developed outside of Unix initially, and was primarily developed without a sufficient concern for authentication and privacy.

Here there is a conundrum: to rewrite large portions of Unix and the protocols underlying its environment, or to fundamentally change its structure would be to attack the very reasons Unix has become so widely used. Furthermore it would be contrary to the spirit of standardization that been a major factor in the recent wide acceptance of Unix. At the same time, without such a reevaluation and some restructuring, there is some serious doubt about the level of trust that can be placed in the system. And it was that same spirit of development and change that led Unix to its current niche.

4 Concluding Remarks

Many conclusions can be derived from the experience with Unix. Certainly one of the most obvious involves the evolution and design of code. 25 years ago, it would have seemed inconceivable that a simple operating system designed almost as a simple exercise would become such a major part of the world of computing. Had that knowledge been available then, it is almost certain that Dr. Thompson (and Kernighan, Ritchie, Morris, Ossanna, and the others) would have thought more about their design. He and the others might not have changed anything, but I suspect they certainly would have given serious thought to how to handle other matters (such as security).

Of course, it is easy to look back two decades and describe design decisions that should have been made differently. In a research environment, decisions are made all the time out of expediency, and often, ignorance of the future. There is certainly no dishonor in cutting some corners when one has limited goals and resources, but there is a problem when those mistakes become codified and accepted as standard practice by others.

The real heart of the problem, however, is the marketing of software that was largely experimental in nature, and without extensive testing. To further compound the problem, users and vendors have pushed for standards that lock many of the troublesome features in place. This may well mean that we are stuck with these weaknesses for some time, or else we must be willing to accept a certain amount of experimentation and "non-standard" code as vendors seek to come to better designs.

We have recently begin to see attempts to "standardize" features for security that do not address the fundamental problems with Unix.[7] For instance, drafts of the IEEE P1003.6 standard specifically exclude issues of authentication from its scope. These "standards" describe features that have not been widely-implemented and tested in real Unix systems, if they have been tested at all. This means there is no assurance that the mechanisms will work as expected, or that they are comprehensive or flexible enough for typical environments. We may see considerable effort expended on the implementation and development of standards-compliant software, only to see a new set of weaknesses and bugs as a result.

A second conclusion is a bit more subtle, and is that there is perhaps a problem with our educational system. Many of our students have been taught using this experimental operating system, and they have entered the world of commerce seeking that same system. We failed to expose those students to other operating systems, and we certainly failed to educate most of them about how to make good decisions about security and risk analysis. This same narrowness of view is evidenced in more places than just security, but is a severe problem here.

Of course, the problem is not limited to just computer science-oriented programs. Other programs have been teaching using PCs for some time, and these machines are susceptible to a wider range of problems than most UNIX machines, including computer viruses. Even elementary schools are using personal computers and workstations to teach, and the educators there are encouraging students to purchase their own computers. Unfortunately, few programs at any level in the educational structure are teaching responsible behavior and risk awareness to accompany the other computer-oriented material they offer.

A third observation is that we need to reevaluate our view of how our computers should be used and how we view UNIX. For instance, not too long ago, computers were big and expensive, and it was necessary to do everything on a single system. Now, with cheaper hardware, we are still operating in the same mode. This is probably not a good idea. Instead, we should separate functionality out to machines best suited — by hardware, security, or other considerations — to handle them. Thus, it may be best to have all the program development tools on one system (or set of systems), but not have those systems equipped with electronic mail and news software. We would have another set of systems with the mail, and a third set with the production software in use. By partitioning our tools and our usage, it may help reduce the threat.

Whatever our conclusions, we must agree that UNIX is here to stay, and its presence will only become more widespread. We cannot change its history, nor can we easily (or quickly) change its fundamental nature. Thus, if we wish to have better security, we are going to have to alter our expectations of what a UNIX environment is and has to offer, or else we are going to have to settle for more than a modicum of risk.

References

[1] Bill Cheswick and Steve Bellovin. *Firewalls and Internet Security - Repelling the Wily Hacker.* Addison Wesley, 1994.

[2] David A. Curry. *UNIX System Security: A Guide for Users and System Administrators.* Addison-Wesley, Reading, MA, 1992.

[3] Daniel Farmer and Eugene H. Spafford. The COPS security checker system. In *Proceedings of the Summer Usenix Conference.* Usenix Association, June 1990.

[4] Rik Farrow. *Unix System Security.* Addison-Wesley, 1991.

[5] Æleen Frisch. *Essential System Administration.* Nutshell Handbook, Inc. O'Reilly & Associates, Petaluma, CA, 1991.

[6] Simson Garfinkel and Gene Spafford. *Practical Unix Security.* O'Reilly & Associates, Inc., Sebastapol, CA, 1991.

[7] IEEE P1003.6 Committee. Draft standard for information technology — portable operating system interface (posix) security interface. Ballot draft 12, September 1991.

[8] Gene H. Kim and Eugene H. Spafford. Experiences with tripwire: Using integrity checkers for intrusion detection. In *Systems Administration, Networking and Security Conference III.* Usenix, April 1994.

[9] Barton P. Miller, Lars Fredriksen, and Bryan So. An empirical study of the reliability of unix utilities. *Communications of the ACM*, 33(12):32–44, December 1990.

[10] E. Nemeth, G. Snyder, and S. Seebass. *Unix System Administration Handbook.* Prentice Hall, 1989.

[11] Brian Reid. Reflections on some recent computer break-ins. *Communications of the ACM*, 30(2):103–105, February 1987.

[12] Deborah Russell and G. T. Gangemi Sr. *Computer Security Basics.* O'Reilly and Associates, Inc., July 1991.

[13] David R. Safford, Douglas Lee Schales, and David K. Hess. The TAMU security package: An ongoing response to internet intruders in an academic environment. In Edward DeHart, editor, *Proceedings of the Security IV Conference*, pages 91–118, Berkeley, CA, 1993. USENIX Association.

[14] Peter H. Salus. *A Quarter Century of Unix.* Addison Wesley, 1994.

[15] Donn Seeley. Password cracking: A game of wits. *Communications of the ACM*, 32(6):700–703, June 1989. 1989.

[16] Eugene H. Spafford. The Internet Worm: Crisis and aftermath. *Communications of the ACM*, 32(6):678–687, June 1989.

[17] Eugene H. Spafford. Making UNIX secure. In *Proceedings of the Virus Bulletin Conference on Security*, pages 125–132, Oxon, England, September 1991. Virus Bulletin Ltd.

[18] Cliff Stoll. *The Cuckoo's Egg.* Doubleday, NY, NY, October 1989.

A Suggested Readings

There is very little written about UNIX security. Some computer vendors offer a security guide along with their standard documentation, but that is usually just information on the technical details of how to use the tools provided with the system.

A few recent books on UNIX security are available, and recommended. Rik Farrow's book [4] is a nice introduction to UNIX security. It focuses on System V-derivations of UNIX and provides sound advice on how to use the available security tools and protection mechanisms. Dave Curry's book [2] presents a more technical view of UNIX security, and includes many helpful tips for all versions of UNIX. *Practical Unix Security* ([6]) is a 512 page in-depth look at practical methods for both System V and BSD-based versions of UNIX. It includes chapters on NFS, network firewalls, encryption, Kerberos, and handling a breakin along with information on the usual topics. It also has an extensive bibliography and source list for additional material and organizational support.

The administrator's guides by Nemeth, Snyder and Seebas ([10]) and [5] are excellent guides for UNIX system administrators, and both provide information on a wide variety of topics, including some related to security.

Computer Security Basics by Russell and Gangemi ([12]) is an excellent introduction to the basic terminology and literature of computer security. It explains such things as TEMPEST and the Orange Book, as well as providing a very extensive source book of information on other resources. It is highly oriented to US law and regulations, however.

B Specific Recommendations

The following fourteen suggestions, if applied, can make a significant improvement in your overall UNIX system security:

1. Set an enterprise-wide security and ethics policy — for everything, including the computing resources. Be sure that the policy is explicit, fair, and applies to everyone. Describe why it is important to follow the policy. Be certain everyone receives a copy of the policy and understands it.

2. Do regular backups of *everything* on your system. Test the backups regularly to be sure they can be used to restore the system. Protect the backup media from theft, snooping, and catastrophe (fire, flood, etc.).

3. Educate your users about good security practice. Do not provide a list of do's and don'ts without explanation; rather, provide some sensible instruction in good security practice. Included should be (at least) advice in setting good passwords, file protections and setting `umask` values, using group IDs to best effect, and physical security.

4. Establish contingency plans. Review and rehearse the plans on a regular basis – conduct "hacker drills" on a periodic basis. Important personnel should be familiar with the plan, and all computer users should be aware of how to activate the emergency provisions.

5. Designate specific personnel to be in charge of security. Provide them with authority concomitant with this responsibility. Provide them with sufficient support (including scheduled time and continuing education funding) to perform their job.

6. Do not assume that all the threats to your operation are from the outside. In most environments, the likely threat is from the inside: disgruntled or opportunistic employees, relatives and friends of employees, or ex-employees.

7. Carefully evaluate your computing needs and roles. Does everyone with access currently need that access? Do all the machines networked together need to be on the same network?

8. Stay current with bug fixes and announcements. See if your vendor has an update list that describes security fixes and problems. Watch various newsgroups and mailing lists that deal with UNIX security information.
9. Turn on whatever auditing capabilities your system may have, and regularly monitor the output. Investigate suspicious entries and activities.
10. Regularly scan your system for changes, odd protection modes, new or altered programs, strange mailer aliases, or altered configuration files. The checklists in [6] can help, as can use of static audit tools such as COPS[3] and Tiger[13]. Keep a paper listing of important configuration files and protection information for comparison purposes. Consider using an integrity monitor such as Tripwire [8].
11. Consider putting seldom-changed configuration files and commands on read-only media that can only be updated during single-user mode operation. This keeps those commands and files from being tampered with. Note that software read-only protection is not adequate in a UNIX environment.
12. Consider carefully removing commands and facilities, or protecting them so that they are not accessible by every user. On production machines, compilers and program development tools should be removed or protected to be unusable by regular users.
13. Consider curtailing or removing network services (especially NFS) when there are untrusted machines on your local network. Use subnetting to isolate untrusted machines from your protected hosts, or put them on isolated networks.
14. If your local network needs to connect to the outside world, put a *firewall machine* in place.[6, 1] Done properly, this will protect your systems inside from cracking activity, and protect against disclosure of information to the outside.