



# Lawrence Berkeley Laboratory

UNIVERSITY OF CALIFORNIA

## Information and Computing Sciences Division

To be presented at the Fourteenth Department of Energy,  
Computer Security Group Conference, Concord, CA,  
May 6-9, 1991, and to be published in the Proceedings

### **Why Computer Security Systems Don't Work**

D.F. Stevens

February 1991



Bldg. 50 Library.  
Copy 1

LBL-30352

### **DISCLAIMER**

This document was prepared as an account of work sponsored by the United States Government. Neither the United States Government nor any agency thereof, nor The Regents of the University of California, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial products process, or service by its trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or The Regents of the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof or The Regents of the University of California and shall not be used for advertising or product endorsement purposes.

Lawrence Berkeley Laboratory is an equal opportunity employer.

## **DISCLAIMER**

This document was prepared as an account of work sponsored by the United States Government. While this document is believed to contain correct information, neither the United States Government nor any agency thereof, nor the Regents of the University of California, nor any of their employees, makes any warranty, express or implied, or assumes any legal responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by its trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or the Regents of the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof or the Regents of the University of California.

# **Why Computer Security Systems Don't Work\***

David F. Stevens  
Information and Computing Sciences Division  
Lawrence Berkeley Laboratory  
University of California  
Berkeley, California 94720

**February, 1991**

Prepared for the  
Fourteenth  
Department of Energy Computer Security Group Conference  
Concord, California, 6-9 May, 1991

---

\* This work was supported by the U. S. Department of Energy under contract No. DE-AC03-76SF00098.

## Why Computer Security Systems Don't Work\*

D. F. Stevens  
Lawrence Berkeley Laboratory  
Berkeley, CA 94720

There are few endeavors in which so many sets of opposing forces operate as the development and operation of computer security systems. The particular sets that might be considered depend upon one's point of view. I tend to be a user advocate, and so would include all of the following:

- computers *vs* people;
- systems *vs* people;
- customers *vs* vendors (and occasionally vendors *vs* truth);
- security *vs* freedom;
- security *vs* productivity;
- interfaces *vs* people; and
- documentation *vs* proper design.

I doubt that there are any real surprises here other than the last item (a justification for which will follow in due course), and the list is probably incomplete, but it is long enough to provide ample food for computer-security-system designers' thought.

We are fortunate in that the difficulties we face when we attempt to balance these forces have been anticipated by many thoughtful people who have expressed their observations concisely, memorably, and, at least occasionally, wittily. I have drawn heavily upon their insight, casting this note as a list of aphorisms, loosely connected by exegetical narrative. I do not apologize for basing so much of this work on the wisdom of others (for I am as willing as Newton to stand on the shoulders of giants), but rather express dismay that this wisdom so often languishes unheeded: We readily accept the truth of the nuggets to be presented here, yet we continue to recreate the kinds of situations to which they apply; as Franklin put it [1],

*Experience keeps a dear school, yet system designers will learn in no other.*

(Actually, he seems to be rather optimistic. There are times when we are certain that there are some things they are unable to learn in *any* school.)

We shall begin by looking at the problems between people, on the one hand, and—in the order of the title—computers, security, and systems, on the other. Then we justify the inclusion of documentation in the list, and conclude with a couple of observations about vendors and consultants.

*Computer people are like Frenchmen: Whatever you say to them, they translate into their own language and it immediately becomes something different. [2]*

---

\* This work was supported by the U. S. Department of Energy under contract No. DE-AC03-76SF00098.

One of the major problems that real people have with computers is that computers are surrounded by computer people. Speaking in tongues and acronyms is only one of the many attributes that distinguishes computer people from real people. Other obvious characteristics include printing instead of writing (we used to print all in capital letters, but then *unix* came along, and now we print totally *without* capitals), slashing our zeros, and counting from zero (real people start from one). These are small differences, but in the aggregate they set us somewhat apart.

Computer people become computer people because they like computers better than they like people. They like the challenge of getting computers to do their will, and often succeed, but are less successful with other people. Perhaps it is because they know that when they deal with computers, their instructions will be followed to the letter; with people, of course, one never knows what will happen. In other words,

*Machines follow rules much better than people do. [3]*

In fact, real people have trouble dealing with the insistence of computers on doing what they are told; they would much rather have computers that do what they (the users) *mean*. We will return to this general topic in the discussion of people and systems, but in the meantime we can pursue the implications of the instructability of computers with respect to people and security. More precisely, we shall pursue the implications of the relative *non-instructability* of people, for if people followed instructions as successfully and completely as computers do, then computer security systems might work. Unfortunately, many people consider that

*Computer security is about as useful as bones in a potato. [4]*

And although that gives us problems in our rôle as computer security professionals, when we allow ourselves to think as generic computer professionals, we recognize its fundamental truth. Security measures do not help the users get their jobs done, and we err if we fault them for their attitude. We, of all people, should recognize that

*Security is the antithesis of programmer convenience... [5]*

...for we have experienced both sides of the dilemma. As programmers, we have been fettered by the need to maintain secure procedures; as security practitioners, we have been victimized by those who left "convenience" gates in their code, gates that were later exploited for entry by those we would have preferred to keep out. It is as programmers that we most strongly realize that

*A procedure is a guide for those who cannot manage themselves, but a straightjacket for those who can. [6]*

This feeling is shared by modern users who are groping towards computer maturity. They are discovering that

*One cannot exercise maturity in a choice-free environment. [7]*

If all our users were mature, our life would be much simpler, but, alas, they are not. At least, not as we would like to define "mature". Comfortable in our own maturity, I suppose, we have trouble understanding the attraction of immature behavior. It seems to be an ever-increasing motto of our society that vice pays better than virtue, and there is a growing number of people who test our systems for personal gain. There are also those who see computers, or the ability to break into them, as a source of power. We are already familiar with the notion that power corrupts; we are now learning a fundamental extension of that thought, i.e., that

*Power attracts the corruptible. [8]*

Nevertheless, it is probably still the case that more of our current troubles stem from the mountain-climbing fraternity. They diddle our systems for fun, because they are there. Their motivation, if, indeed they have any, is obscure. Perhaps they are driven by an unconscious recognition of the Davies Diagnosis:

*People who mind their own business die of boredom at thirty,... [9]*

...and wish to avoid a boring demise.

For whatever reasons, the attacks persist, with the result that

*Security decays... [10]*

...and we cannot be sure that a system that was secure yesterday will be secure tomorrow. This is not due solely to active penetration attempts, either. For many systems the number of keys that have been issued is constantly growing, and not all the keys that are no longer needed are returned. Some are lost, some forgotten, some retained as status symbols, and many are inadequately protected. It is also the case that security measures are being overtaken by technology. Two examples of this are the contribution of desktop power to the cracking of dictionary passwords (of which you will hear more at this conference), and the contribution of enhanced communications capability to the expansion of our potential "user" base: a security procedure that was adequate for a local user population numbered in the hundreds may be quite ill-suited to a global user population numbered in the millions.

In the end it is clear that the only secure system is one with no users; in fact, since any running system is subject to failure, the only truly secure system is one that is turned off. That does seem to limit its utility somewhat. Somewhat more picturesquely,

*A ship in a harbour is safe, but that's not what ships are built for. [11]*

Computers are not installed to be secure, they are installed to do work.

While it is certainly true that computer security systems have special problems, they are also subject to the generic problems that exist between people and all kinds of systems. These have been

discussed at great length by a number of perspicacious observers, resulting in such well-known formulations as the Peter Principle and Parkinson's Law. But there are many lesser-known precepts that shed light on other aspects of the pervasive antagonism between people and systems. (Many of these have seen succinct expression in Gall's *Systemantics* [12], a book that no manager of computer security should be without.)

One characteristic in particular seems to typify the usual organizational approach to computer security: rigidity. This often starts out as "mere" formality, but

*Systems convert means to ends. [13]*

Thus what began as a formal mechanism to achieve some human (or bureaucratic—not necessarily the same thing) goal eventually becomes a goal in its own right. We stop measuring lines of code, for example, as a means to assess programmer productivity, and start trying to set new LOC records, regardless of the value of the lines produced. The system goal has replaced the human goal. This is not to be unexpected, for, as Gall [*op. cit.*] has noted

*Systems develop goals of their own the instant they come into being.*

If only we were better at detecting the generation of these systematical goals and uprooting them before they became embedded in the established culture, we would be much better off.

Because computer security systems are often implemented on the computers they "serve", they tend to be rather rigidly procedural. And therein lies another of the friction points between people and systems, for in the real world

*Loose systems last longer and work better. [12]*

We have seen some awesome demonstrations of the eventual fragility of rigid systems in Eastern Europe in the recent past. A large, strong, rigid system can stand against the shifts of the underlying world for a long time, but you can't bend an inflexible rod, and eventually the strain becomes too great and the system shatters. Loose systems allow for the fuzziness of human activity, and provide us all an opportunity to recover from error. Loose systems employ a benign inefficiency;

*Efficient systems are dangerous to themselves and to others. [12]*

Many of us have fallen victim to a classic recent example of ruthless system efficiency: the *unix* command *rm \**. This command requires only five keystrokes (including the carriage-return) to eliminate all of one's files. Looseness, of course, is not the only attribute of a well-running system. Another is illustrated by a story once told of the Lawrence Livermore National Laboratory. It seems that someone had the bright idea of providing free bicycles for on-site use, in the hopes of reducing the strain on the intra-site bus and taxi services, saving fuel, reducing air pollution, improving muscle tone, and all those good things. So bicycles were provided, in sufficient numbers that one could just pick up the closest one, ride it to wherever one was going, leave it there for someone else, pick up another for the next journey, etc. Well, this sort of worked. The trouble was, the bicycles

didn't stay randomly distributed around the Lab; a great preponderance seemed to come to rest at a certain building, that we shall call "Building N". They would be re-scattered periodically, but like the monarch butterflies to Pacific Grove, they would eventually find their way back to Building N. This remained a great mystery until someone happened to be looking at a topographical map of the site, and noticed that the entrance to Building N was located at the lowest spot in the Laboratory. They had just discovered that

*Systems run better when they run downhill. That is, they work better when they are aligned with human motivational vectors. [12]*

Ay, there's the rub. Security systems are rarely aligned with human motivational vectors. Not because people are malicious, but because, by and large, they don't care for bony potatoes. Yet another reason why security systems fail to align well with human motivation is that people, especially people in the sorts of environments we serve, like to experiment. From the point of view of computer systems, this translates into

*To err on purpose is human,...* [14]

...with its limiting case of "everything will be tried". Failure to recognize this tendency has led to some marvelous failures. One of my favorites concerns the old IBM 70xx series of computers, which had an *execute* instruction, XEC. XEC was useful for certain kinds of multiple-choice situations, especially when combined with the possibility of address indirection\*. This was acceptably complex for most programmers, but the fun really began when one combined this with the relative-addressing provision of the assembly-language program†. Still, all well and good...until one tried to perform an "execute indirect to self": XEC\* \*. This is the rough equivalent of handing the system a card that says

*Turn this card over,  
and follow the instructions found  
in the place specified  
on the other side of this card.*

on one side of the card, and

*The other side of this card*

---

\* The normal XEC zzz instruction would cause the system to execute the instruction contained in location zzz. Indirection was called for by inserting an asterisk in the instruction code, XEC\* zzz; in this case, the contents of location zzz were treated as a pointer, and the system executed the instruction contained in the location pointed to by zzz.

† This used the asterisk to indicate the location containing the instruction. This was handy in such locutions as \*\*+3 to indicate the location three beyond the location of the executing instruction.

on the other. The system became so engrossed in turning the card over and over that it was unable to do anything else; the only way out was to (literally) turn off the computer! A simple example of "trying everything" that effectively caused the system to self-destruct. Another favorite example of mine is less dramatic, but I like it because it involved my then boss. During a computer-aided introductory course for an office system, the general procedure was to answer a question, then press the *ENTER* key. But the instructions were quite explicit: "After answering Question 24, *DO NOT PRESS ENTER.*" Being a programmer by training, my boss pressed *ENTER*; the system crashed. A co-worker of mine thought this was fascinating, so he, too, tried it. The system crashed. Not being a true programmer, I learned from the experience of others.

As is clear from the above remarks, cautionary advice is often ignored. While this says something about people, systems, and documentation, of which more anon, it also points out the fact that

*Time spent learning the system is perceived as time wasted. [6]*

This is not limited to the use of non-productive tools, such as security procedures. People don't install spreadsheet programs, for example, for the sheer pleasure of learning about spreadsheet programs; they install them because they have real work to do. Furthermore, they think they know what spreadsheets are, and so they think they know how they should work. Whatever time they have to spend unlearning their own style and learning the system developer's style is time stolen from the work they are trying to do. This can be a frustrating experience. If it is repeated often enough, they begin to feel that

*The system is to the user as the dog is to the tree. [15]*

It does not help to attempt to make the system "foolproof". To begin with, no one has yet built a foolproof system, and, secondly,

*Build a system that any fool can use, and only a fool will want to. [16]*

Well, we've avoided documentation long enough. The traditional complaint about documentation is that is too late, too obscure, and too inaccurate. While this may be true, the real truth is that even if it were on time, correct, and clear, it would not help matters a whole lot, because

*Nobody reads the manual to prevent errors. (This is because documents are better reminders than teachers.) [14]*

Even if the documentation is good, the necessity to have it at all is indicative of a more fundamental problem:

*You do not fix a poor interface by documenting its idiosyncracies. [17]*

(This is not limited to computer systems, of course; Doug Norman has noted more generally that

*When simple things need instructions, the design has failed. [18]*

People have tried to avoid the problem by making various forms of on-line and automated systems for providing assistance and advice. These, too, tend to fail, for a number of reasons:

*Unless you already know what there is, you can't find out what is there. [19]*

*The people who write indexes for computer system documentation are the same people who index the Yellow Pages and announce bus departures. [20]*

*Automated advice fails because it assumes the user knows what he did. [6]*

*Automated advice fails because it assumes the user will follow it. [21]*

The first two of these principles allude to the classic problem of most indexed assistance schemes: unless you know what's there before you look, and *know what words the indexer has chosen to use*, your search will generally be rather frustrating, and may well fail. The latter two are results of treating users as though they were computers: possessed of perfect recall and the ability to follow instructions without question. The third one, for example, doesn't address the user who was hitting keys randomly to see what might happen, or the user who meant to do one thing but actually did another, or the user who was in a totally unfamiliar context. And the last one just doesn't cope with people. The user might not be trying to do what the system thinks he had in mind, for instance, or may have decided to solve the problem a different way, or simply decided to do something else instead. One hopes the system will let him, but many systems do not.

Finally, we need to address what Stan Kelly-Bootle considers to be the fourth of the Seven Catastrophes of Computing [22]. So much has been said in other contexts that we need here list only a couple of the principles that one should bear in mind when dealing with vendors and consultants:

*Vendors are to vend. [23]*

*Never ask the barber if you could use a haircut. [24]*

*To a small boy with a hammer, everything looks like a nail. [24]*

It seems always to come as a surprise when people discover that vendors are not disinterested observers, and that their advice is sometimes self-serving. (Sigh.) So these are simply reminders, (a) that vendors and consultants are in business to sell their products and services, and you can often predict their answers to certain questions simply by remembering that fact; and (b) if they can find a way to redefine or contort your problem into the sort of nail for which their hammers are designed, you can bet they will.

I was tempted at the beginning to entitle this note "Why computer security systems *can't* work", but upon reflection I realized that that isn't quite true. They can work—for a while, under certain

circumstances, with the right user population, and so on—but by and large they work as well as they do only because they are not severely tested.

As you have realized by now, I hope, my intention was not to provide a recipe for the creation of workable computer security systems, but to provide a sense of how the real world in which those systems must live can conflict with the ideal world we sometimes inhabit when we are in system-design mode. When our attention is narrowly focussed on our special task and discipline of computer security, it is easy to forget the general truths presented here. We must keep them in mind, however, if we are to have any hope of producing systems that are both acceptably secure and reasonably useful.

## References

- [1] Ben Franklin, *Poor Richard's Almanack*. He spoke of fools rather than systems designers, but sometimes it's hard to tell the difference.
- [2] Göthe, speaking of mathematicians, quoted in *St. Patrick's Almanack*, The "Open Channel" column in *Computer*, March, 1986. Had there been computer people when he was alive, he would undoubtedly have addressed this complaint to them.
- [3] J. Seymour, *Coping with computer egos*, an AMA Management Briefing, 1984.
- [4] J. S. Zimmerman, *PC Security: So what's new?*, *Datamation*, 11/1/85.
- [5] Brian Reid, *Viewpoint*, CACM, February, 1987, who stated the inverse, but I am taking antitheticity to be self-evidently commutative.
- [6] D. F. Stevens, *Why computer security systems don't work*, 1991.
- [7] A slight rewording of an observation in *Encouraging mature user behavior*, D. F. Stevens, 1980.
- [8] Frank Herbert, in an interview in *Psychology Today*, October, 1984.
- [9] Robertson Davies, *A mixture of frailties* (Vol 3 of the Salterton Trilogy, Penguin, 1985).
- [10] Anon.
- [11] John Parks, quoted in Rob Slade's e-mail sign-off block, February, 1991.
- [12] John Gall, *Systemantics*; Quadrangle, 1975 (I am told that a revised edition has recently appeared).
- [13] D. F. Stevens, *Towards user-oriented performance management*, EDP Performance Review, May, 1979. The original form—"if the goals don't determine the measures, then the measures will determine the goals"—arose in the context of computer performance measurement.
- [14] D. F. Stevens, *Cautionary aphorisms for user-oriented computer management*, 1979; revised, 1983.
- [15] Adapted from an oral comment at the 1979 University of California Management Institute.
- [16] Chris Shaw, *Shaw's Rule in The Official Rules* (Paul Dickson; Lacorte Press, 1978).
- [17] D. Verne Morland, *Human factors guidelines for terminal interface design*, CACM, July, 1983.
- [18] Douglas Norman, *The psychology of everyday things*, Basic Books, 1988. He includes many fascinating examples of what one would think were well-understood, self-explanatory objects, such as doors, failing utterly to convey the information necessary for proper operation.
- [19] D. F. Stevens, *Thoughts on usability*, 1982.
- [20] Evelyn C. McDonald, *You know what really bothers me...?*, *Government Computer News*, 5/22/87.
- [21] William C. Hill, *How some advice fails*, CHI '89 Proceedings.
- [22] Stan Kelly-Bootle, *The devil's DP dictionary*, McGraw-Hill Paperbacks, 1981. They are the user, the manufacturer, the model, the salesperson, the operating system, the language, and the application.
- [23] D. F. Stevens, *More cautionary aphorisms for user-oriented computer management*, 1986.
- [24] Paul Dickson, *The official rules*, Lacorte Press, 1978. The first of these is given as The First Law of Expert Advice, the second is a rephrasing of Kaplan's Law of the Instrument.

LAWRENCE BERKELEY LABORATORY  
UNIVERSITY OF CALIFORNIA  
INFORMATION RESOURCES DEPARTMENT  
BERKELEY, CALIFORNIA 94720