Brian Reid

viewpoint

Reflections on Some Recent Widespread Computer Break-Ins

In the first weeks of September 1986, some number of UNIX® systems in the San Francisco area, and elsewhere on the ARPANET, were systematically penetrated by talented intruder(s). We believe that it began at Stanford; it seems to have spread to many other computers around the ARPA Internet. The intruder often left behind recompiled login programs to simplify his return. His goal seems to have been purely numeric, to see how many computers he could crack. We are aware of no major destruction or theft of files—although the inconvenience to users and managers of these computers was significant. Most of these computers are operated without professional management by small groups of researchers; because no records were kept, it is difficult to tell exactly how many machines were penetrated. The number could be as high as 30 to 60 on the Stanford campus alone. Similar break-ins at other ARPA Internet sites have been reported to us. Virtually all of the other site managers have asked me not to reveal their identity, but I have first-hand knowledge of break-ins at 9 universities, 15 Silicon Valley companies, and 9 ARPANET sites, including 3 government labs.

An analysis of the break-ins shows that they were made possible by a combination of technology and human nature, creating an environment in which penetrations were easy and the consequences more dire than necessary. In reporting this event, I will review the technical nature of the break-ins, draw some conclusions, and make some recommendations based on what we have learned.

The user-level networking features in Berkeley UNIX are formulated around the concepts of remote execution and trusted host. For example, the command typed on computer A for remote copying from computer B to computer C is:

```
rcp B:file C:file
```

The decision of whether or not to permit these copy commands is based on permission files on computers B and C. If my account on computer B has a permission file that contains the entry "(`A,reid`)," the operating system on computer B will give access to someone who is logged on to A with identity "`reid`." The command will succeed if I have an account on all computers involved, and the permission file stored in my directory on those computers authorizes the attempted kind of access.

The break-ins started with an obscure Stanford computer that was used primarily as a mail gateway between UNIX and IBM computers on campus. This computer had a guest account with the user id "`guest`" and the password "`guest`." Although this is a silly mistake, it is a common one, and those managing this computer did not have much experience in setting up UNIX systems. Because this computer is used only as a mail gateway, there was no incentive to keep it constantly up-to-date or carefully patrolled. The intruder easily broke in to the guest account.

On most Berkeley UNIX systems, it is routine for an experienced programmer to be able to trick a privileged system program into executing nonstandard versions of system commands. If a user has write permission in a system directory, he or she can store a bogus version of a system command in that directory and then wait for some regularly scheduled system maintenance command to execute that version. The substitute command will thus be run as a certain privileged user, called a superuser. Because of the search path mechanism, by which several directories will be searched until the requested program is found, it is difficult to spot such security holes by casual inspection.

On the distribution tape of 4.2BSD (a version of Berkeley UNIX), it happens that the directory named "`/usr/spool/at`" is universally writeable by all users; it also happens that it is fairly easy to trick the operating system into executing privileged commands by storing them in that directory. When beginners install that distribution tape, they rarely realize that the directory protection is a security problem, and so do not change the protection from the default value. The intruder knew this and had no difficulty in becoming the superuser on that machine. We suspect that the intruder simply looked around the

The opinions of the author stated herein do not necessarily represent the position of ACM.

UNIX is a trademark of AT&T Bell Laboratories.

network until he found a machine with both a guest account and the writeable-directory bug.

Having become superuser, the intruder was able to assume the login identity of anybody who had an account on that computer. In particular, he was able to pretend to be user "x" or user "y," and in that guise ask for a remote login on other computers. This enabled him to get access to x's or y's privileges on various remote machines. One user, a systems programmer, had accounts on several machines, each of which had a permission file authorizing access from the mail gateway computer. Having logged in to the other computers, the intruder repeated the process of becoming the superuser, and worked his way through the entire network. The machine on which the initial break-in occurred was one I did not even know existed, and no one in my department had any control over it. Yet a leak on this seemingly inconsequential machine on a remote part of campus was able to spread to our machines because of the networking code.

The above description of how the intruder picked his way through the network illustrates the technological side of the break-ins. What made this whole affair so interesting is how the UNIX technology interacted with human nature to create a large number of permission files for users who had write permission to a system directory. Here is how that happened.

The networking mechanism in Berkeley UNIX is very, very convenient. To move a file from one place to another, the user just types "`rcp`" and it is there. Fast, efficient, and quite transparent. The alternative is the file transfer program (`ftp`), which is slower, less accurate, and harder to use. In my work I am often asked to help install software on remote machines. Like so many other systems people, I prefer `rcp`. To use `rcp`, I must create, however briefly, a permission file authorizing the transfer. If for some reason I am interrupted, and I forget to delete the permission file, it will remain behind, a silent aid to an intruder. After a few years of operation, a large network of Berkeley UNIX systems seems to grow extra permission files the way old oak trees grow mistletoe. A surprising number of these files are created by people like me, installing system software, and therefore having write permission to system directories.

People often let convenience dominate caution. Search paths are almost universally misused. Many sites modify the root search path so that it will be convenient for systems programmers to use interactively as the superuser, forgetting that the same search path will be used by system maintenance scripts run automatically during the night. Essentially every UNIX computer I have ever explored has grievous security leaks caused by very general or overly long search paths for privileged users.

Systems programmers are in short supply, and they are almost always overworked. They often make heavy use of sophisticated software tools and shortcuts to make their jobs more tractable. In this instance, the convenience of the remote file access mechanism was so compelling that it may have lured some systems programmers into using it to lighten their workloads, sacrificing security for convenience.

UNIX was created as a laboratory research vehicle, not as a commercial operating system. As it has become more widely used commercially, many of the properties that made it attractive in the laboratory have created problems. For example, the permission file mechanism described above lets me easily give my colleagues full access to the files on my computer. When UNIX systems are installed in nonlaboratory applications by people who are not trained to think about operating system security, however, the same mechanism that is convenient in the laboratory becomes dangerous in the field. There is no way to assign fault or blame for these security problems, because if the UNIX system is used as its designers intended, security is not a problem.

A frustrating irony was that after we saw and understood the *modus operandi* of the intruder, we had great difficulty convincing the night shift authorities that this was a serious problem. We could not get the telephone calls traced that night. At one point an intruder spent two hours talking on the telephone with a Stanford system manager, bragging about his accomplishment, but we could not get the call traced. He must have known this, or he would not have called.

It is possible to err in the opposite direction by applying too much police power. A few years ago, Stanford and other universities suffered a series of break-ins by a high school student. He was ultimately convicted of felony charges. The conviction and the sentence were excessive relative to the true damage that he caused, but were probably typical in terms of the amount of police and FBI effort that was required to catch him.

In summary, the two technological entry points that made these intrusions possible were:

- The large number of permanent permission files, with too many permissions stored in them, found all over the campus computers (and, for that matter, all over the ARPANET).

- The presence of system directories in which users have write permission. This was caused in part because the people who prepared the 4.2BSD release tape did not anticipate the diversity of potential customers for their software. They were probably thinking that the customers were university computer science departments.

My conclusions from all of this are:

- Nobody, no matter how important, should have write permission into any directory on the system search path. Ever. One should not be able to install a new program without typing a password.
- It would be a worthwhile research venture to carefully rethink the user interface of the Berkeley networking mechanisms, to find ways to permit people to type passwords as they are needed, rather than requiring them to edit new permissions into their permissions files.
- The permission-file security access mechanism is fundamentally vulnerable. It would be quite reasonable for a system manager to forbid the use of them, or to drastically limit the use of them. Mechanized checking is easy.
- Programmer convenience is the antithesis of security, because it is going to become intruder convenience if the programmer's account is ever compromised. This is especially true in setting up the search path for the superuser.
- To catch intruders efficiently advance plans must be made. Making them will also help keep down the level of frenzy during the chase. If computer installations make some advance arrangements with the local police, and come to agreements about how the police can help solve electronic intrusions, it will increase the effectiveness of the deterrent while decreasing the likelihood of an overreaction. Computer break-ins are becoming routine, and we need to make the response to them equally routine.

Author's Present Address: Brian Reid, DEC WRL, 100 Hamilton Avenue, Palo Alto, CA 94301 (reid@wrl.dec.com.)

Authors (*continued from p. 99*)

the Department of Computer Science, Queen's University of Belfast, where he has been a staff member since 1969. His current research interests include programming languages for multiprocessor, distributed, and array and vector processor configurations; design and analysis of parallel algorithms.

JAN SANISLO

Heterogeneous Computing Environments

page 132

Jan Sanislo is a programmer working on the heterogeneous computing project at the University of Washington. His research interests include distributed systems.

MICHAEL SCHWARTZ

Heterogeneous Computing Environments

page 132

Michael Schwartz is a Ph.D. candidate at the University of Washington. His research interests include distributed systems, programming environments, and computer architecture, with particular emphasis on very large heterogeneous systems.

ZEN-CHUNG SHIH

Systolic Algorithms to Examine All Pairs of Elements

page 161

Zen-Chung Shih's research interests include algorithm design and analysis, parallel processing, computational geometry, computer graphics and combinatorics. He is a Ph.D. candidate of the Graduate Institute of Computer and Decision Sciences, National Tsing Hua University.

ALAN STEWART

An Improved Parallel Thinning Algorithm

page 156

Alan Stewart is a lecturer in the Computer Science Department at Queen's University of Belfast. He is interested in parallel algorithms and mathematical specification techniques.