THE ACM CODE OF ETHICS

The Association for Computing Machinery (ACM) is the flagship organization for computing professionals. The ACM supports publications of research results and new trends in computer science, sponsors conferences and professional meetings, and provides standards for computer scientists as professionals. The standards concerning the conduct and professional responsibility of computer scientists have been published in the ACM Code of Ethics. The code is intended as a basis for ethical decision making and for judging the merits of complaints about violations of professional ethical standards.

The code lists several general moral imperatives for computer professionals:

- Contribute to society and human well-being.
- Avoid harm to others.
- Be honest and trustworthy.
- Be fair and take action not to discriminate.
- Honor property rights, including copyrights and patents.
- Give proper credit for intellectual property.
- Respect the privacy of others.
- Honor confidentiality.

The code also lists several more specific professional responsibilities:

- Strive to achieve the highest quality, effectiveness, and dignity in both the process and products of professional work.
- Acquire and maintain professional competence.
- Know and respect existing laws pertaining to professional work.
- Accept and provide appropriate professional review.
- Give comprehensive and thorough evaluations of computer systems and their impacts, including analysis of possible risks.
- Honor contracts, agreements, and assigned responsibilities.
- Improve public understanding of computing and its consequences.
- Access computing and communication resources only when authorized to do so.

In addition to these principles, the code offers a set of guidelines that provide professionals with explanations of various issues contained in the principles. The complete text of the ACM Code of Ethics is available at the ACM's Web site, http://www.acm.org.

Computer Ethics

COPYRIGHT, INTELLECTUAL PROPERTY, AND DIGITAL INFORMATION

For hundreds of years, copyright law has regulated the use of intellectual property. At stake are the rights of authors and publishers to a return on their investment in works of the intellect, which include printed matter (books, articles, etc.), recorded music, film, and video. More recently, copyright law has been extended to include software and other forms of digital information. For example, copyright law protects the software used with this book. This prohibits the purchaser from reproducing the software for sale or free distribution to others. If the software is stolen or "pirated" in this way, the perpetrator can be prosecuted and punished by law. However, copyright law also allows for "fair use"—the purchaser may make backup copies of the software for personal use. When the purchaser sells the software to another user, the seller thereby relinquishes the right to use it, and the new purchaser acquires this right.

When governments design copyright legislation, they try to balance the rights of authors and publishers to a return on their work against the rights of the public to fair use. In the case of printed matter and other works that have a physical embodiment, the meaning of fair use is usually clear. Without fair use, borrowing a book from a library or playing a CD at a high school dance would be unlawful.

With the rapid rise of digital information and its easy transmission on networks, different interest groups—authors, publishers, users, and computer professionals—are beginning to question the traditional balance of ownership rights and fair use. For example, is browsing a copyrighted manuscript on a network service an instance of fair use? Or does it involve a reproduction of the manuscript that violates the rights of the author or publisher? Is the manuscript a physical piece of intellectual property when browsed or just a temporary pattern of bits in a computer's memory? When you listen to an audio clip on a network, are you violating copyright, or only when you download the clip to your hard drive? Users and technical experts tend to favor free access to any information placed on a network. Publishers and, to a lesser extent, authors tend to worry that their work, when placed on a network, will be resold for profit.

Legislators struggling with the adjustment of copyright law to a digital environment face many of these questions and concerns. Providers and users of digital information should also be aware of the issues. For more information about these topics, visit the Creative Commons Web site at http://creativecommons.org/.

Computer Ethics

INTRUSIVE HACKING

Hacking is a term whose use goes back to the early days of computing. In its original sense, a "hacker" is a programmer who exhibits rare problem-solving ability and commands the respect of other programmers. A "hack" commonly refers to a programming maneuver that exhibits technical brilliance and elegance. The culture of hackers began in the late 1950s at the MIT computer science labs. These programmers, many of them students and later professionals and teachers in the field, regarded hacking as an accomplishment along the lines of Olympic gymnastics. These programmers even advocated a "hacker ethic," which stated, among other things, that hackers should respect the privacy of others and distribute their software for free. For a narrative of the early tradition of hacking, see Steven Levy, *Hackers: Heroes of the Computer Revolution* (Garden City, New York: Anchor Press/Doubleday, 1984).

Unfortunately, the practice of hacking has changed over the years, and the term has acquired darker connotations. Programmers who break into computer systems in an unauthorized way are called hackers, whether their intent is just to impress their peers or to cause actual harm. Students and professionals who lack a disciplined approach to programming are also called hackers. An excellent account of the most famous case of intrusive hacking can be found in Clifford Stoll, *The Cuckoo's Egg: Tracking Through the Maze of Computer Espionage* (New York: Doubleday, 1989).

Computer Ethics

COMPUTER VIRUSES

A ***virus*** is a computer program that can replicate itself and move from computer to computer. Some programmers of viruses intend no harm; they just want to demonstrate their prowess by creating viruses that go undetected. Other programmers of viruses intend harm by causing system crashes, corruption of data, or hardware failures.

Viruses migrate by attaching themselves to normal programs, and then become active again when these programs are launched. Early viruses were easily detected if one had detection software. This software examined portions of each program on the suspect computer and could repair infected programs.

Viruses and virus detectors have coevolved through the years, however, and both kinds of software have become very sophisticated. Viruses now hide themselves better than they used to; virus detectors can no longer just examine pieces of data stored in memory to reveal the presence or absence of a virus. Researchers have recently developed a method of running a program that might contain a virus to see whether or not the virus becomes active. The suspect program runs in a "safe" environment that protects the computer from any potential harm. As you can imagine, this process takes time and costs money. For an overview of the history of viruses and the new detection technology, see Carey Nachenberg, "Computer Virus-Antivirus Coevolution," *Communications of the ACM*, Volume 40, No. 1 (January 1997): 46–51.

Electronic Voting Machines

In the 2000 presidential elections in the United States, votes were tallied by a variety of machines. Some machines processed cardboard ballots into which voters punched holes to indicate their choices (see Punch Card Ballot figure). When voters were not careful, remains of paper—the now infamous "chads"—were partially stuck in the punch cards, causing votes to be miscounted. A manual recount was necessary, but it was not carried out everywhere due to time constraints and procedural wrangling. The election was very close, and there remain doubts in the minds of many people whether the election outcome would have been different if the voting machines had accurately counted the intent of the voters.

Subsequently, voting machine manufacturers have argued that electronic voting machines would avoid the problems caused by punch cards or optically scanned forms. In an electronic voting machine, voters indicate their preferences by pressing buttons or touching icons on a computer screen. Typically, each voter is presented with a summary screen for review before casting the ballot. The process is very similar to using an automatic bank teller machine (see Touch Screen Voting Machine figure).

It seems plausible that these machines make it more likely that a vote is counted in the same way that the voter intends. However, there has been significant controversy

Punch Card Ballot

surrounding some types of electronic voting machines. If a machine simply records the votes and prints out the totals after the election has been completed, then how do you know that the machine worked correctly? Inside the machine is a computer that executes a program, and, as you may know from your own experience, programs can have bugs.

In fact, some electronic voting machines do have bugs. There have been isolated cases where machines reported tallies that were impossible. When a machine reports far more or far fewer votes than voters, then it is clear that it malfunctioned. Unfortunately, it is then impossible to find out the actual votes. Over time, one would expect these bugs to be fixed in the software. More insidiously, if the results are plausible, nobody may ever investigate.

Many computer scientists have spoken out on this issue and confirmed that it is impossible, with today's technology, to tell that software is error free and has not been tampered with. Many of them recommend that electronic voting machines should be complemented by a *voter verifiable audit trail.* (A good source of information is [1].) Typically, a voter-verifiable machine prints out the choices that are being tallied. Each voter has a chance to review the printout, and then deposits it in an old-fashioned ballot box. If there is a problem with the electronic equipment, the printouts can be counted by hand.

As this book is written, this concept is strongly resisted both by manufacturers of electronic voting machines and by their customers, the cities and counties that run elections. Manufacturers are reluctant to increase the cost of the machines because they may not be able to pass the cost increase on to their customers, who tend to have tight budgets. Election officials fear problems with malfunctioning printers, and some of them have publicly stated that they actually prefer equipment that eliminates bothersome recounts.

What do you think? You probably use an automatic bank teller machine to get cash from your bank account. Do you review the paper record that the machine issues? Do you check your bank statement? Even if you don't, do you put your faith in other people who double-check their balances, so that the bank won't get away with widespread cheating?

At any rate, is the integrity of banking equipment more important or less important than that of voting machines? Won't every voting process have some room for error and fraud anyway? Is the added cost for equipment, paper, and staff time reasonable to combat a potentially slight risk of malfunction and fraud? Computer scientists cannot answer these questions—an informed society must make these tradeoffs. But, like all professionals, they have an obligation to speak out and give accurate testimony about the capabilities and limitations of computing equipment.

An Early Internet Worm

In November 1988, a graduate student at Cornell University launched a virus program that infected about 6,000 computers connected to the Internet across the United States. Tens of thousands of computer users were unable to read their e-mail or otherwise use their computers. All major universities and many high-tech companies were affected. (The Internet was much smaller then than it is now.)

The particular kind of virus used in this attack is called a worm. The virus program crawled from one computer on the Internet to the next. The entire program is quite complex; its major parts are explained in [2]. However, one of the methods used in the attack is of interest here. The worm would attempt to connect to `finger`, a program in the UNIX operating system for finding information on a user who has an account on a particular computer on the network. Like many programs in UNIX, `finger` was written in the C language. C does not have array lists, only arrays, and when you construct an array in C, as in Java, you have to make up your mind how many elements you need. To store the user name to be looked up (say, `walters@cs.sjsu.edu`), the finger program allocated an array of 512 characters, under the assumption that nobody would ever provide such a long input. Unfortunately, C, unlike Java, does not check that an array index is less than the length of the array. If you write into an array, using an index that is too large, you simply overwrite memory locations that belong to some other objects. In some versions of the `finger` program, the programmer had been lazy and had not checked whether the array holding the input characters was large enough to hold the input. So the worm program purposefully filled the 512-character array with 536 bytes. The excess 24 bytes would overwrite a return address, which the attacker knew was stored just after the line buffer. When that function was finished, it didn't return to its caller but to code supplied by the worm (see A "Buffer Overrun" Attack). That code ran under the same super-user privileges as `finger`, allowing the worm to gain entry into the remote system.

Had the programmer who wrote `finger` been more conscientious, this particular attack would not be possible. In C++ and C, all programmers must be especially careful not to overrun array boundaries.

One may well wonder what would possess a skilled programmer to spend many weeks or months to plan the antisocial act of breaking into thousands of computers and disabling them. It appears that the break-in was fully intended by the author, but the disabling of the computers was a side effect of continuous reinfection and efforts by the worm to avoid being killed. It is not clear whether the author was aware that these moves would cripple the attacked machines.

In recent years, the novelty of vandalizing other people's computers has worn off somewhat, and there are fewer jerks with programming skills who write new viruses. Other attacks by individuals with more criminal energy, whose intent has been to steal information or money, have surfaced. See [3] for a very readable account of the discovery and apprehension of one such person.

Software Piracy

As you read this, you have written a few computer programs, and you have experienced firsthand how much effort it takes to write even the humblest of programs. Writing a real software product, such as a financial application or a computer game, takes a lot of time and money. Few people, and fewer companies, are going to spend that kind of time and money if they don't have a reasonable chance to make more money from their effort. (Actually, some companies give away their software in the hope that users will upgrade to more elaborate paid versions. Other companies give away the software that enables users to read and use files but sell the software needed to create those files. Finally, there are individuals who donate their time, out of enthusiasm, and produce programs that you can copy freely.)

When selling software, a company must rely on the honesty of its customers. It is an easy matter for an unscrupulous person to make copies of computer programs without paying for them. In most countries that is illegal. Most governments provide legal protection, such as copyright laws and patents, to encourage the development of new products. Countries that tolerate widespread piracy have found that they have an ample cheap supply of foreign software, but no local manufacturers willing to design good software for their own citizens, such as word processors in the local script or financial programs adapted to the local tax laws.

When a mass market for software first appeared, vendors were enraged by the money they lost through piracy. They tried to fight back by various schemes to ensure that only the legitimate owner could use the software. Some manufacturers used *key disks:* disks with special patterns of holes burned in by a laser, which couldn't be copied. Others used *dongles:* devices that are attached to a printer port. Legitimate users hated these measures. They paid for the software, but they had to suffer through the inconvenience of inserting a key disk every time they started the software or having multiple dongles stick out from their computer. In the United States, market pressures forced most vendors to give up on these copy protection schemes, but they are still commonplace in other parts of the world.

Because it is so easy and inexpensive to pirate software, and the chance of being found out is minimal, you have to make a moral choice for yourself. If a package that you would really like to have is too expensive for your budget, do you steal it, or do you stay honest and get by with a more affordable product?

Of course, piracy is not limited to software. The same issues arise for other digital products as well. You may have had the opportunity to obtain copies of songs or movies without payment. Or you may have been frustrated by a copy protection device on your music player that made it difficult for you to listen to songs that you paid for. Admittedly, it can be difficult to have a lot of sympathy for a musical ensemble whose publisher charges a lot of money for what seems to have been very little effort on their part, at least when compared to the effort that goes into designing and implementing a software package. Nevertheless, it seems only fair that artists and authors receive some compensation for their efforts. How to pay artists, authors, and programmers fairly, without burdening honest customers, is an unsolved problem at the time of this writing, and many computer scientists are engaged in research in this area.