

Computers, Security, and Individual Rights

A Friendly Introduction for the Concerned Citizen

©June, 2001 Andrew D. Hwang, ahwang@mathcs.holycross.edu

Contents

1	Introduction	1
2	How Computers Work—An Analogy	4
3	Networking and the Internet	9
4	Security	18
A	Computers and Information	27
B	Web Sites for Further Information	37

1 Introduction

Computer security is an issue of importance to all citizens of a democratic society. Unfortunately, many people believe the issues are too complicated to be understood except by “experts”. This abdication of judgment is perilous, because computers, in conjunction with legislation promoting corporate interests, can threaten the privacy of individuals. Laws are currently being drafted and promoted by parties that value profits above civil liberties.

There are several reasons to know something of computer security. You may wish to safeguard information such as personal email, web browsing habits, or medical and credit information. You may want to understand legislation that is being promoted by the government or software manufacturers, to be reassured that no one is going to take away your freedom of speech or claim ownership of your

information. You may be interested in why or how people break into computer systems, and what they do once they've broken in. This document is short, and does not answer all of these questions; the aim is to provide a foundation for continued self-education by outlining the basic aspects of computer security in a non-technical way, and providing links to additional resources on the world-wide web.

If you come across any terms you do not understand, or find topics that interest you but which are not discussed in detail here, try doing a web search. The search engine google is top-notch for computer-related topics; the URL is in Appendix B, along with several other security-relevant URLs.

Business Models and Legal Issues

The motivation for writing this article is to educate users. There is danger of erosion of civil liberties by restrictive computer security laws. If the public is not informed of the technical, social, and political issues surrounding computer use, then legislators will have no incentive to pass laws that uphold the spirit of the Constitution. Lobbying for software and media vendors already provides substantial pressure against laws that uphold the spirit of the Constitution. Even if you are not American, I hope you share the beliefs that people should control their destinies, and that a well-informed public is capable of governing itself better than a government that dictates regulations as seen fit by a small, inaccessible minority (often in the guise of protecting "competitiveness").

Even in a world of honest, well-intentioned people, there are conflicting interests. A recording company may have legal ownership of a popular song, and of course profits financially from the sale of copies of this song. With the advent of home computers, the majority of listeners have the capability to mass-produce high quality copies of the song more cheaply than can be bought from the legal owner.

The world's largest software vendors have been tireless advocates against giving control of computer programs to users. To make an analogy, think of a computer program as a cooking recipe. It is a piece of *information* that has no intrinsic physical existence (though it can be stored and sold in various forms), and it constitutes *instructions* for performing a task. The vendors' desired business model is this. Everyone who wants to use the recipe must hire a butler (buy a computer and operating system). These people may then request the recipe (license the software), which *only the butler is allowed to read* (the internal details of the program are hidden under legal penalty). You may not share the recipe with neighbors

(copy the software) nor modify the recipe to suit your own needs or tastes (alter the source code to add or remove features).

If recipes were distributed under modern software licenses, there would be general outrage. Who would willingly deny the cook's right to modify a recipe as needed, to substitute oregano for savory, leave out the salt, or use margarine instead of butter? Who would pay money for the right to use a recipe, without actually owning a copy, or without even being allowed to *review* the actual recipe? Who would be satisfied that the recipe did not contain unadvertised ingredients? Who would feel good about telling their friends, "Yes, it *is* delicious, isn't it? But you'll have to pay \$400 for your own copy. I don't want to go to jail."

There are intrinsic problems with this business model, because of the nature of the commodity. If a recipe is stolen and distributed widely, it can *never* be recovered, because information, unlike a physical object, can be freely copied without damaging the original. Since users can easily make copies of the recipe, *every sale is a potential theft!* The obvious way to prevent *de facto* "theft" is to pass and enforce Draconian laws against copying and sharing recipes. But how is one to enforce such laws? One way, already being implemented for software, is to monitor cooks by assigning every butler and every copy of every recipe a unique serial number. Each butler records the serial numbers of the recipes he is asked to make, and periodically reports these numbers back to the vendor. If the same serial number on a recipe is reported by two different butlers, a crime has occurred. Another way is to change the entire model, so that the recipe never leaves the vendor's hands. In software terms, users who want to use a program pay a license fee authorizing them to connect to a computer (called an *application server*) operated by the vendor; they are free to use the software on the remote computer. Users have no means of acquiring a copy of the program, so there is no danger to the vendor of "theft". Whatever data users create is returned to them, and (according to the vendor) the copy of the data on the vendor's machine is destroyed.

I am neither anti-business nor a communist, but am wary of conflicts of interest that arise when legal entities (especially wealthy corporations) can and do influence the passage of laws that affect the conditions under which they are allowed to do business. I am particularly concerned when the commodity being sold—personal computers and software—can literally be used like a radio collar, to track individuals' movements on the Internet, or like the telescreens in Orwell's *1984* to see and hear everything that goes on in a room, *without the owner's knowledge or permission*. This is not alarmism, but technical reality; anyone whose computer is attached to a web cam, a microphone, and the Internet is vulnerable to clandestine

surveillance while their computer is turned on. I do not believe such snooping currently occurs, but the infrastructure to implement it is in place. Other, more subtle, invasions of privacy (some of them described above) are already shipping and being eagerly bought and installed by millions of people worldwide.

It is true that an individual can simply decide never to use a computer (or telephones, electricity, and running water), but realistically it is impossible to avoid computer use without cutting oneself off from the mainstream of commerce and social life. As individuals, we are increasingly faced with an unacceptable choice: To drop out of the cultural mainstream, or to compromise our privacy. This is not an inevitable result of the march of technology, but is promoted by interests that wish (not necessarily maliciously) to control individual behavior in a way that would have seemed totalitarian in the 1940s.

There *are* alternatives. The Free Software Foundation has developed a professional-quality PC operating system called *GNU* (known to most people as “Linux”) whose distribution license, the *GNU General Public License* (GPL for short) is based on the traditional model of recipe trading: You can copy, modify, and distribute recipes in any way you see fit, so long as you do not restrict the rights of others to do the same. Some object to this enforced cooperativeness, and prefer *Open Source* licenses instead, which guarantee that everyone can read and modify recipes to their own needs, but limit the ways recipes can be distributed.

If you use a personal computer and are concerned with privacy and democracy, you have the right (and the responsibility) to become informed of technical and legal issues that stem from the expensive appliance on your desktop. I hope this article is useful to you as a starting point in an ongoing process of self-education.

2 How Computers Work—An Analogy

A modern personal computer is a device that can store and process enormous amounts of data rapidly and accurately. The familiar tasks you do with your computer—sending and receiving email, browsing the world-wide web, writing and printing documents, doing your household finances, scanning and editing photographic images and audio clips, and playing games—are abstractly nothing but creation and manipulation of data. When you send email or browse the web, your computer exchanges data with other computers. When your computer is connected to the Internet, it may be possible for other computers to request data from your machine. This capability of data sharing is a proverbial double-edged sword; it makes possible the Global Village as implemented by the Internet, but

brings the risk of intrusion by individuals, corporations, and governments, which can result in loss of privacy or even identity theft.

Running a Household

The internal workings of a computer are a little removed from everyday experience, so we will proceed by analogy with the process of running a large, complicated household. The metaphor is imperfect and sometimes whimsical, but you will find that many things a computer does have familiar analogues.

A computer consists of *hardware*: the case (really, the electronics inside), monitor, keyboard, and mouse. In the analogy, the hardware is the mansion occupied by the household. An empty building by itself is not much use as living space; you need furniture and decorations to make the space pleasant and useful, and you need many servants to run the day-to-day affairs of the household. Similarly, a “naked” computer is little more than an expensive paperweight. You need to be able to create, manipulate, store, and retrieve data in order that your computer be useful.

The instructions that tell a computer how to run, and that allow you to interact with your computer, are generally called *software*, and are grouped into *programs*, which are specialized units that perform (usually) one specific task. The software in a computer is analogous to servants who help you, the head of the household, run things smoothly. Different programs correspond to different kinds of servants; there are many servants because the household is very large and complicated.

Just as the butler manages the servants in a large household, every modern computer has one very special program that manages all the other programs. This program, called the *operating system kernel*, or “kernel” for short, is the first program to start when you turn on your computer, and it handles all the operation of the computer until you shut down. The kernel does not really keep house all by itself; it comes with a host of specialized servants (plumbers, electricians, builders, gardeners, and librarians) called *system utilities* that together with the kernel comprise the *operating system*. In some households (such as Windows or Macintosh), you never have to give instructions to the system utilities directly; other servants pass your requests to the operating system. Some households allow you to address the operating system directly; GNU/Linux and FreeBSD are the most common PC operating systems that do not shield you from the system utilities.

We have so far likened a computer to a house whose affairs are run by a butler; the butler slavishly follows instructions, but works *very* quickly, and does not tire from repetitive tasks. Most computers ship with an operating system installed. But

like a staffed house that has no furniture, a computer with nothing but an operating system can do very little, though it is more nearly capable of communicating in a language humans can speak and understand. The servants you talk to regularly are *application programs*, roughly analogous to secretaries, accountants, artists and craftspeople, jesters, and possibly others. With their help, you can stock the kitchen, bathroom, library, and game room, build your own furniture, hang artwork, and generally make the house livable. The physical objects in your house are analogous to the data on your computer.

Operating systems

Not all operating systems are equal. The most common PC operating systems have two substantial technical drawbacks and one unfortunate feature:

- The butler is capable of handling only one task at a time; rather than allowing the servants to go off and perform their tasks in their own time, returning when they have finished—the butler works sequentially with one servant at a time. In technical terms, the operating system is not *multi-tasking*. In such a household, the butler spends most of the time waiting for other servants to finish their tasks.
- The servants do not go off to separate rooms to do their work; consequently, their tasks sometimes cross paths, as when the cook sets down a cauldron of soup on the clean laundry waiting to be ironed. When this happens, one or both servants get confused and stop what they were doing, waiting for further instructions. If one of these servants happens to be the butler, the entire household stops (the familiar “blue screen of death” or “bomb icon”, depending on the operating system), requiring you to “reboot”. The soup, the partially cleaned sheets, and the secretary’s notes are all lost. In technical terms, the operating system is not running in *protected memory mode*, so different processes can write onto each others’ *address space*. This is not a bug, but a *design decision*.

Sometimes servants need to talk to each other about their respective tasks; in computer science, this is called *interprocess communication*. Rather than putting servants in the same room, it is safer to put them in different rooms and let them speak by intercom. A good operating system has intercoms, often called *pipes* and *sockets*.

- Finally, as already mentioned, the servants' instructions come in sealed envelopes that you are forbidden to read *by law*. You can only trust the company that built your house to have provided instructions that really do what they claim. The butler (or another servant) may be performing tasks that you did not ask for. We will return to this possibility later, because there are several more-or-less sinister ways in which the butler can be subverted.

In other words, the *source code* for the software on a PC is not usually available in human-readable form. There are two distinct movements in opposition to this state of affairs:

- The **Free Software** movement, which is built on the philosophy that the programs you run should be yours with specific rights, similar to the rights that protect speech or your right to trade recipes with neighbors. The intent is to make computer use a neighborly activity, in which everyone is able to share and improve existing computer programs for the betterment of all.
- The **Open Source** movement, whose philosophy is more purely pragmatic. The idea (which works exceedingly well in practice) is that if everyone can read and check the source code for important programs, then errors can be found and repaired quickly, and other improvements can be distributed easily.

Neither movement forbids the selling of software, and both have shown themselves to be viable and efficient means of creating powerful, useful software of unparalleled quality. The plurality of web servers in use at this writing are Free or Open Source, and run on Free or Open Source operating systems. Both movements embody “competition” in the best sense: A program is evaluated by how well it does its task, and how useful the task is, not by meaningless criteria of popularity or superficial attractiveness.

Application Programs

A computer program is created by writing a detailed set of instructions for a certain task, then using a special computer program called a *compiler* to translate the human-readable instructions into machine-readable ones. Even a simple task requires a lengthy set of instructions, because a computer cannot do anything unless it is explicitly told to.

A cooking recipe is a human analogue of a computer program. But even the simplest recipe leaves out common-sense details that a computer requires. For example, consider the recipe for brewed coffee: “For each 6 oz. of water, use 2 Tbsp. of ground coffee.” A human would know without asking to count how many people are expected to want coffee, to search the kitchen for a coffee pot and check that it is in working order, to add the water and coffee in the proper places, then to turn on the power and wait. A human could also deal with contingencies: There is no ground coffee in the house, the carafe is cracked, there is already water in the reservoir. A computer knows none of these things, and must be given explicit instructions for every eventuality. If a computer program finds itself in a situation for which it has no instructions, it will simply stop. If the instructions tell the computer to re-try a step in the event of failure, the machine will futilely try forever unless told to stop.

When several programs work together, the chances of one program passing improper information to another program multiply. Maybe there is another program that counts the number of people at home and passes this information to the coffee brewing program. If there are many coffee non-drinkers, the program will cause too much coffee to be made. Worse, if there are too many people home, the control program will overflow the coffee maker by trying to make more cups than the carafe can hold. This could cause further errors to occur, such as an electrical fire in the kitchen.

All but the simplest software has “bugs”, unforeseen contingencies or behavior different from what the programmer intended. Publically distributed software usually does not have obvious bugs, but may be veritably riddled with subtle, hidden bugs. Most of these bugs are harmless, but as we shall see some of them are as treacherous as frayed electrical wiring or defective door locks in a high-crime neighborhood. Software may also have arcane, undocumented “features” that range from the quirky to the potentially sinister. All you can do as a user is to be careful about the programs run on your computer. It is easier to “be careful” if you use Free Software or Open Source Software. Free and Open Source programs are not bug-free, but their source code is continually and thoroughly audited by a concerned community of able volunteers, and the bugs—once found—are documented openly and clearly, and are fixed with extreme rapidity. It is also possible to “be careful” with proprietary software, but bugs are usually not publicized openly, nor are they fixed quickly. Further, you must trust the software vendor not to have ulterior motives. Computers offer an unparalleled means of collecting and cataloging marketing profiles of individuals. It is naive to believe that a software vendor who owns a marketing agency will not use its software products to further

its marketing goals, regardless of the impact on individuals' privacy.

3 Networking and the Internet

Bugs aside, computers are so fast and reliable that we (as individuals, and as a society) rely on them increasingly for all manner of information management. In particular, more and more people conduct financial and other legal transactions over the "Internet" and the "world-wide web".

Just as information can be sent from one house to another via the postal service, computers worldwide are linked by the *Internet*. If your computer is connected to the Internet (usually through a modem), you can share data with any other computer that is also connected to the Internet. This allows you to send and receive email and browse the web, for example.

When a package arrives at your house, the butler receives it and acts accordingly upon it, either opening it immediately, passing it to another household servant, or putting it where you will find it and then waiting for instructions from you (as with an email message). The postal analogy is fairly good, but not perfect, because there are crucial differences between data and the contents of postal packages:

- First, **data can be transmitted rapidly**, almost anywhere in the world in minutes or seconds.
- Second, **data can be copied easily**, while physical objects cannot be. If you send your sister a Venetian blown-glass vase, you cannot keep a copy for yourself. If the vase were, instead, digital data, you could send a copy to as many people as you wanted, as many times as you wanted. You could make a spare copy for safekeeping, in case the cat knocked the original off the mantel; this is known as *backing up* data.
- Third, **data may contain instructions** that will be performed when the package is received. The butler does not consult you on each individual package (there may be thousands of packages per hour arriving at your house), so often these instructions are followed without your direct intervention.

In an ideal world, all of these attributes of data are beneficial. You can share your data quickly and as much as you like without losing it yourself. You can

email family photos to friends without the originals leaving your possession. Sharing recipes is a pre-computer example of data exchange; no matter how many people you give your recipes to, you do not lose the original. The copies can be used to make more copies, all identical to the original. (By contrast, the *ingredients* used in a recipe cannot be duplicated and shared; if you give a neighbor a cup of sugar, you have one cup less sugar yourself.) In the form of instructions, data can be a labor-saving device, allowing you to automate lengthy or tedious tasks (such as alphabetizing a long list of names or balancing the household budget with a spreadsheet). With an Internet connection you can telecommute, working without physically going to your office, even printing documents remotely for your co-workers. These are conveniences that we increasingly take for granted. Each of the three properties of data is a fundamental difference between physical commodities and information, and each plays a part in the power and utility of the computer.

It does not require much imagination to see that these properties of data carry risks. Your household servants are permitted to perform any actions requested (cleaning, moving furniture, cooking, taking out the garbage), but are slavishly obedient and by default childishly trusting. If a stranger sends a package asking the butler to chop the Hepplewhite into kindling and stuff the pieces into the toilet, the butler will do as asked unless you have cautioned the butler to ignore destructive requests from strangers. The outcome would be damaging in a different way if the butler honored a request to duplicate and mail off a copy of your financial records, including your bank account, credit card, and social security numbers. The speed and transmissibility of data mean that such a request can be initiated half a world away, and be honored in a very short time, perhaps under a minute, without your being the wiser. Naturally, you want to protect yourself against damage to the possessions in your house, and to control the divulging of your personal data. Unfortunately, it is not simple to achieve these goals, short of refusing all packages, which is tantamount to disconnecting yourself from the Internet.

Real-Life Attacks

Anyone who follows the nightly news has heard stories of computer exploits. Many home users have been personally affected, perhaps unknowingly. These attacks range from cruelly ironic to merely cruel. To avoid being the victim, you need to know how the attacks work, how to secure yourself and your computer from vulnerability, and how to detect signs of an attack.

Email Worms

In the real world, where people with less-than-honorable intentions reside, connecting your computer to the Internet is a risk. Imagine a world, very much like our own, in which 90% of the population lives in houses built from the same set of blueprints, and employing the same servants who perform the same tasks. One morning, a package arrives at your house; because it is an email message, the butler puts it into your inbox. You see that it is from a friend, and is labelled “A special gift for you”, so you tell the butler to go ahead and open it. Inside the package, the butler finds the message, “Go to the desk, find the address book in the top left drawer, and copy the first fifty names. Make fifty copies of this package, and send them to those fifty people. Then go through all the picture books in the library and write ‘I love you’ in permanent red marker on each page.” The butler, who robotically follows directions, trashes your library and sends the package (technically, a *worm*) to fifty of your friends, very likely trashing *their* libraries and causing each of their butlers dutifully to copy the package fifty times, spreading the joy to 2500 people, whose butlers unwittingly attack yet more people. The entire process, from your telling the butler to open the package to the destruction of your books, takes a few seconds at most. There may be a delay until your friends open the special gift, but after perhaps half a day the postal service will be in danger of collapse from the sheer volume of mail, and millions of people will be left angry and violated. Luckily, your house can be rebuilt in its original condition in about an hour, though any personal touches you added may be damaged or lost, especially if you did not make spare copies of all your photo albums before the attack.

Of course, such an attack *did* happen, and cost people worldwide uncounted hours of cleanup and repair. The attack originated in a country with no laws against email worms, and there was little evidence tying the release to an individual, so prosecution was essentially impossible. The prime suspect claimed, probably truthfully, to have been trying to steal a few users’ passwords, not to bring down email servers and damage millions of users’ computer data worldwide. Why did such an attack work? One reason is that so many houses were set up the same, so it was easy to give the generic butler simple instructions to carry out the attack. Second, in the vulnerable houses, the butler was allowed to follow instructions from postal packages, without asking for confirmation. Sadly, this was the default behavior written into the butler’s contract when the house was built, and the owner either didn’t know the danger or felt the convenience was worth the risk.

The event was reminiscent of a child playing with matches who accidentally burns down a large part of a city. Of course, the child should not have been playing with matches, but had good fire-fighting infrastructure been in place the damage would have been far less. There is no safe way to run a program of unknown origin, and it is arguably negligent of a vendor to ship email software that is by default configured to run programs received in a mail message. Rather than laying blame, it is better to secure your own computer so that it is less vulnerable to email attacks.

Non-Windows users should not feel superior to those affected by the “I love you” worm. In 1988, a similar event, perpetrated by a computer science graduate student named Robert Morris, crashed hundreds of Unix and VMS mainframes. The problem was a bug in the *de facto* standard mail handling program, `sendmail`, that again allowed a piece of software to duplicate and send itself automatically.

Virus Hoaxes

A curious social phenomenon has emerged that illustrates the danger of having only a little knowledge. Most home computer users are aware that “viruses” exist, but few know how real exploits work, where they come from, or how news of them is officially distributed during an actual event. Several times a year, I receive dire-sounding email claiming to reveal a newly-discovered virus. There are always vague references to authoritative sources (IBM, Microsoft, Intel, and the television networks), and instructions on avoiding the virus, which usually arrives in an email message with a distinctive subject line. These messages come from friends, and uniformly implore me to forward the important warning on to everyone I know.

Without exception, the warning is a hoax. Ironically, the warning *itself* is a worm, whose purpose in life is to clog Internet connections and email servers with bogus messages. The act of passing the warning on spreads the worm, less rapidly but just as surely as allowing an email program to run executable code in an email message might. The irony is that users’ concern for other users (combined with ignorance and a bit of credulity) causes them to perpetrate the very act they are trying to prevent.

Human-propagated worms are usually nothing but annoyances, but recent ones can damage your operating system. In May, 2001, I received a message to the effect that a nasty virus was spreading, and would install itself on the hard drive under a certain name. If not deleted immediately, this virus would erupt on a certain date in the near future, erasing all the data on the hard drive. In reality, the

name of the “virus” referred to a real (but obscure) Windows 98 system file. At least some people who received the warning dutifully deleted the file; the bogus message correctly advised that Windows would try to stop the deletion of the offending file, but that one should continue anyway. Luckily, the file was easily restored from the installation disk, so the outcome was little more than public embarrassment—this time.

In response to the Morris Internet Worm of 1988, the U.S. Government’s Defense Advanced Research Projects Agency formed the “computer emergency response team”. The organization, now called CERT/CC (CERT Coordination Center), is housed at Carnegie-Mellon University in Pennsylvania. Their web URL is in Appendix B. (CERT is a trademark, not an acronym.) All major computer exploits are officially announced and documented by CERT/CC. If an exploit does not appear on the CERT/CC web site, there is a very good chance the “exploit” is a hoax. Do not forward “virus warnings” without checking their authenticity, regardless of how much you trust the sincerity of the friend who sent you the warning.

Remote Own3rship

You are in a chat room; a newcomer enters and puts some enticing offer on the screen, available from a web URL. (If the chat room is for children or teens, the offer may be for a cool prize related to a new movie. In an adult room, the link might promise unsavory video pleasures, or offer a fantastic deal “as seen on TV”.) You, or your child, click the link, and are taken to a page that seems to deliver on the promise. However, you have unwittingly done something risky; a web URL causes data to be sent to your computer, and in most popular PC operating systems, your butler’s default behavior (if the data contains instructions) is to regard your clicking the link as permission to follow the instructions that are received. Unbeknownst to you, your butler receives a package that says, “Put this receiver in your ear. Be sure not to let anyone know you’ve got it. Any time you hear something through the receiver, do it but act natural like you’re just going about your business. Destroy this message, and don’t tell anyone you received it.” Your butler, who is naive and obedient, complies to the letter.

The worst, most violating computer attack is when someone gains complete control of your machine without your knowledge. Uniquely computer-ish attributes make such attacks possible, and users’ desire for convenience makes them increasingly widespread. The details differ, but the result is the same: Your butler ends up obeying orders from a third party without your permission, and often

without your knowledge. In system cracker lingo, your machine has been 0wn3d. As long as your machine is connected to the Internet, the attacker can do anything with your computer that can be achieved by software alone. The attacker can view your keystrokes, capturing your passwords; copy or delete your data, and use your machine as a storage depot for stolen programs, or as a springboard to attack other computers; turn on your web cam and stream the output to a remote machine, capturing video of the real room in which your computer sits. Hanging up the modem or rebooting have no lasting effect; the next time you connect to the Internet, your butler will send the attacker a message, saying how to reach the house. Currently these hijinks appeal mostly to voyeurs and maladjusted youths, but identity theft is a real possibility if you keep financial records on your computer. Obviously, it can be very difficult to determine if your machine has been attacked, precisely because a good attacker will force your computer to lie to you about its own internal state.

The Windows family of home operating systems facilitates these attacks by permitting your computer to run *invisible processes*, programs that do not make their presence known to you, the user. Further, there are many ways to trick either the butler or you into giving the attacker a foot in the door. The GNU/Linux operating system is often vulnerable in the default configuration, but for different reasons. We will explore this issue in detail later. For now, suffice it to say that the operating system offers limited access to the computer (which can be a good thing), and can be tricked into offering complete access (a very bad thing). Most GNU/Linux users do not know how to give restrictive instructions to the servants to repel these attacks, nor do they know how to detect signs of an exploit.

Freedom and the Internet

Many powerful legal entities (mostly software and media corporations) see the Internet as a potential Marketers' Paradise, and they are absolutely correct. Computers can automatically tabulate, organize, and transmit data about where users have been on the web and what they have done at individual sites. If you enter some seemingly non-identifying data at one web site, it can be merged with a composite profile of you when you go to another site. How? First, distinct sites may have the same parent company, or may otherwise have a legal agreement to share marketing data. Second, your web browser *must* identify itself to the web sites you visit, for how else are the web pages to arrive at your computer? Third, in order to "personalize" your browsing experience, many web sites put small pieces of identifying data, called *cookies*, on your hard drive as you browse. These cook-

ies identify *your specific computer* to the web site if you leave and return. You can disable cookies, or tell your browser to notify you each time it is offered a cookie, but you might as well forget about web browsing if you do. Try it and see why if you're curious. Again, this is not a technological necessity, but a fact of life dictated by marketing interests. About the best you can do is to delete your cookies periodically, and certainly after each browsing session. You'll find that web sites are not as friendly or personalized when you first arrive, but that's what you'd expect. And of course, you can't uncollect all the information that's been accumulated from past browsing, because there is no disclosure of who stores the data, what data is stored, or who may access the data.

One of the greatest strengths of the Internet is its lack of global organization. The Internet is a powerful medium of communication because so many people have access to it, and can publish their views for all to consider. Not everyone can broadcast their own radio program or television show, or publish their own mass-distribution newspaper or magazine, but anyone with access to a computer can set up and maintain a web site. Attempts to regulate the content of Internet traffic are certain to benefit the already wealthy and powerful, and to marginalize those with views (even just, rational views) that happen to be unpopular. On the flip side, it is a sad fact of life that there are malign people in the world. Some of these people twist Free Speech to their own ends by promoting hate and prejudice, advocating violence, or simply by distorting or fabricating the truth. Others use their freedom to attack others, either for mischief (such as vandals) or for profit (hucksters and thieves).

A strong democracy depends on the ability of the average citizen to evaluate ideas they encounter, and to filter out uncomfortable truths from flim-flam, propaganda, lies, and hate-mongering. When Internet content is regulated by governments (or voluntarily, by Internet Service Providers), it is a sad day for civil liberties. One can take only small comfort in the international nature of the Internet, which acts in opposition to attempts to legislate the Internet by individual countries. The law has not yet caught up with technology regarding jurisdiction of actions on the Internet; an Australian attacker can break into an American computer by using an account on a machine in the Netherlands. (In reality, attacks are often far more convoluted.) If two of the countries involved do not have an appropriate treaty, or if their laws are in direct conflict, prosecution may be legally impossible.

I believe there is no legal solution to the essentially social problem of malign human nature, and that attempts to legislate behavior on the Internet are dangerous to individuals' freedoms. Consequently, I believe it is the right and responsibility

of each person using the Internet to be aware of the risks and to take appropriate steps to protect themselves and their data. The Internet may be a frontier society (with all the good and bad aspects entailed by benign anarchy), but people willing to become informed can defend themselves against individual attackers. By contrast, no one is safe from an institutionalized information police, be it governmental or corporate.

Regardless of what is or is not legal on the Internet, there will always be risks associated with computer networking, because the nature of computers and the complexity of their software makes them vulnerable to remote subversion by malicious attackers. Security is a technical and social problem, and cannot be legislated away by restricting Internet content. While you cannot do much about malign human nature, you *can* avoid many technological pitfalls, and set up your computer in a way that is difficult to subvert.

The structure of the Internet

The Internet is a network of computers located in countries throughout the world. Any two computers connected to the Internet can communicate by exchanging data with each other. To give a more familiar analogy, think of the Internet as the postal service, and of individual personal computers as houses. Sharing information between computers is roughly akin to sending a package from one house to another; indeed, the units of information on the Internet are called *packets*! The postal analogy is not perfect, and we will successively refine it as we go, but for now it will do.

When you send a package to a friend in another country, you do not need to know the details of how the package will be transported; you simply write your friend's address on the box and take the box to the post office. Exchange of data on the Internet is, to the user, similar; each packet of data has a *destination address* that gets the packet to the proper computer. The packet also carries a *source address* (analogous to the return address on a package) so that any data sent in reply to the packet can be returned to you.

The Internet is abstractly laid out much like a system of roads and highways. There are a few broad highways that connect large, distant points. Branching off from the highways are streets of varying size; a single street might connect the backbone to a few large universities, businesses, or clusters of Internet service providers. Connected to the streets are multitudes of alleys that are finally connected to the driveways of individual houses. Each branch point (or "post office") is a computer called a *router*; the router that connects your sub-network to the

Internet is sometimes called your *gateway*, and acts as your local post office, to which you take packages to be mailed and where you receive replies.

The Internet has an enormous number of connections; like roads, connections on the Internet carry varying amounts of traffic (depending on their location, the time, and the day of the week), which can cause the speed of transmission to slow down noticeably. Unlike roads, these connections come and go, as machines start up or shut down. A router acts in part like a traffic helicopter, attempting to send data through connections that are relatively unburdened.

Two computers on the Internet can typically exchange data by a path passing through no more than two dozen intermediate routers. Implicit in this assertion is a sobering thought: All the data you send over the Internet passes through several machines on the way to its destination; the sole exception is traffic between you and your gateway, or between you and your ISP's dial-in server. You have no control over who can read the data in transit; the danger is not that the postal service will snoop, but that the postal service has been wiretapped by an attacker. Home computers are not the only machines vulnerable to being *own3d*, and programs called *network sniffers* exist that run on routers, scanning all the traffic that passes through, usually sifting for passwords and credit card numbers.

The behavior of the Internet is local and cooperative. Traffic is routed over the Internet "dynamically": You specify where a packet is to be sent, but the path the data actually follows depends on the existing connections and the amount of traffic during the actual trip. A single message is usually broken into pieces by the sender, and the pieces of a single message will often take different paths to their destination.

Consider how a real postal package gets delivered: You take the package to the post office. The clerk does not know the location of your friend's house (or even how to get the package to your friend's country), but *does* know where the city's main post office is. At the main branch, the clerk sees that the destination is not within the city, and sends the package up to a higher level. At last your package arrives at a national depot where all mail to Switzerland goes. Once the package arrives in Switzerland, the Swiss post sends it to the appropriate city post, who pass it to a neighborhood office, and finally the package is delivered to your friend by her neighborhood mail carrier. The process is remarkable because it does not require any one post office to know everything about the entire postal system, but only about its immediate neighbors (in the postal system hierarchy). Routing of Internet traffic is similar, as it must be, because no central entity can maintain a list of computers attached to the Internet at a given moment.

Most Internet traffic is routed using the Domain Name System (DNS), a global,

distributed, dynamically updated database. The adjectives simply mean that “the list” of all computers on the Internet is scattered among computers worldwide, and is updated in real time as computers come online or shut down. Unless you run GNU/Linux or FreeBSD on your PC, you will probably never need to know more about DNS.

The Internet was designed as an environment where users were cooperating, responsible academics and government scientists. The protocols on which the Internet runs were not designed for security, and the programs that computers use to communicate were not designed to prevent dishonest use. Not until the explosion in popularity due to the inventions of hypertext, multimedia content, and web browsers did the Internet become a public arena. As more personal information (medical and credit information, purchasing patterns and browsing habits) is stored and transmitted, and as more business is conducted electronically, the information flowing through potentially insecure channels becomes of greater value to hucksters, thieves, voyeurs, and mass-marketers.

4 Security

In order to understand which risks are and are not within your control, you must first know what general risks are. When you mail a package, you assume implicitly that the package will not be opened except by the intended recipient (and customs agents). The contents of a package you send are private, and tampering is easily detected. A data packet, by contrast, is justly likened to a postcard; anyone who handles the packet can read it—even make a photographic copy of it—without your knowing. Preventing others from reading your data while it is in transit is the broad area of *network security*.

Individual computers which address on the Internet are called *Internet hosts*. Protecting the data in your own computer is the purvey of *host security*. Several people may have physical access to your computer, and there are good reasons why you’d want to keep different users’ data separate and private. Host security is also an issue because a computer can reply to requests from strangers on remote machines, and is childlike in its naiveté of who it trusts, unless you tell it otherwise.

Encryption

During wartime, when messages must be kept from enemy hands during transmission, the content of a message is *encrypted*, or converted into a form that cannot be read unless one has a secret piece of information possessed only by the intended recipient. In the same way, you can protect your data as it travels over the Internet by encrypting it, provided there is a software infrastructure in place. There is no other way of protecting transmitted data, because of the way the Internet works: Unless you set up a private network, your data will travel through machines that are not under your control.

Recent versions of Netscape and Internet Explorer support fairly strong encryption, though not all sites offer encrypted browsing. Most web sites' URLs begin `http://`. A site whose name starts `https://` is encrypted; the "s" stands for "secure". It is a bad idea to purchase goods or services from a site that does not encrypt the transaction. *It is not necessarily safe to purchase goods and services from encrypted sites!* As you will easily surmise, the host security of the machine storing your personal data is crucially important to you, and is something over which you have no control. Email can also be encrypted, but becomes substantially less easy to use. Each individual must currently weigh their concern for privacy with the added burden of encrypting and decrypting email.

In the cryptography world, transactions are described between hypothetical participants, Alice and Bob. Their nemesis, Charlie, is the evil man-in-the-middle who is trying to eavesdrop.

The most common type of encryption in use today is called *public key encryption*. Public key encryption does not refer to a single method of encoding data, but to a whole class of algorithms that share the following description. Using a computer program, Alice and Bob each generate two pieces of information, their *public key* and their *private key*. As the names suggest, the public key may be freely distributed, while the private key must be known only to the user. In order to communicate securely, Alice and Bob must each possess three keys: Alice has her private key, and both her and Bob's public keys, while Bob has his private key and both public keys.

Suppose Alice wants to send a message to Bob. After writing the text of her message, she uses a computer program together with Bob's *public key* to convert the text of her message into seeming gibberish. She then emails Bob the encrypted message. Bob receives the gibberish from Alice, but armed with a computer program and his *private key*, he is able to recover the text of Alice's message. Charlie may intercept Alice's message, but without Bob's private key Charlie cannot (in

practice) recover the encrypted text. (In principle, Charlie can crack the encryption, but it may take thousands of years and all the world's computing power.)

There is an additional wrinkle, though, because the address of the sender is easily forged in an email message. Alice can be certain that only Bob can read her message, but how is Bob to be certain that the message he received really came from Alice? To *authenticate* herself to Bob, Alice must somehow encode her message in a way that *only she* is capable of. Using her *private* key, Alice encrypts a short message and appends it to her original message as her signature, *then* she uses Bob's public key to encode the message. Now, when Bob decrypts Alice's message, there is some gibberish at the end. To check that the message really is from Alice, Bob attempts to decode the gibberish with Alice's *public* key. If the attempt is successful, Bob can be certain that Alice encoded (and presumably sent) the message. Otherwise, Bob suspects a forgery.

The beauty of public key encryption is that secret data need never be transmitted between Alice and Bob; before sending private messages, the only information they need exchange are their *public* keys. The security of the method depends only on the integrity of their *private* keys; even the details of the encryption procedure can be publicized! The substantial drawback of public key encryption is that email programs do not easily support it; Alice must sign and encrypt by hand every message she sends to Bob, and Bob must decrypt and authenticate every message he receives from Alice.

The U.S. Government opposes general encryption, under the belief that terrorists and international criminals will use it to communicate in secret; until the fall of 1998, the "RSA algorithm" for public key encryption was considered a military weapon by the State Department, and was prohibited from export. The FBI reluctantly supports citizens' right to encrypt data, but has proposed a *key escrow* system, in which everyone registers their private key with a neutral third party. That way, in theory, individuals can send private email, but law enforcement officers armed with a court order can obtain a suspect's private key and therefore read the suspect's email, just as they can tap a suspect's telephone. There are serious privacy issues and logistical difficulties with key escrow. How exactly are the keys to be stored, what sort of clearance would be required to have access to the key database, and how are keys to be transferred to "authorized law enforcement officials"? If the keys reside on a machine connected to the Internet, they are susceptible to being stolen, like any other data. But how else is the nationwide key database to be maintained?

Firewalls

A *firewall* is a program that controls the traffic that flows between the Internet and your computer(s). In our household analogy, a firewall is a servant the butler consults whenever a package arrives from the Internet. In “mission-critical” settings, an entire machine is often dedicated to the purpose, and the term “firewall” refers to the entire machine. There are two general “policies” a firewall can follow: Allow everything that is not expressly forbidden (easier to set up, but less safe), or reject everything that is not expressly permitted (requires more tweaking at first, but far safer in the long run). To explain how a firewall works, it is necessary to say more about networking.

Clients and Servers

Most computer interactions work on a *client-server model*. One machine has some useful information (perhaps data or multimedia files) or access to hardware (say, a printer, or to the computer itself); the other machine wants a copy of the information, or access to the hardware. For example, consider a database, a large collection of information that can be organized and grouped according to several criteria. The actual data resides in a file (or several files) on the first computer, which runs a program called a *database server*. The database server waits for certain types of packages to arrive, namely those that request information from the database. When a request arrives, the server program determines that the request came from an authorized computer, then replies to the request, usually by sending a package containing the requested data.

At the other end of the transaction is a computer being used by a person who wants information from the database. On this computer runs a program, the *client*, that sends appropriately formed requests for data to the server. If you are like most users, the client *is* the database; you ask it for information, and the data magically appears on your screen. Behind the scenes, however, is a slightly more complicated picture. You may be accustomed to speaking of a particular computer as a “server”; in many organizations, several kinds of service are handled by the same computer, and the only reason the average user interacts with the machine is as a client. However, it is more accurate to separate the physical hardware from the programs running on it. A single machine can be a client for one process and a server for another. In fact, every computer is a server for its own display; your computer monitor is a hardware resource, and a remote computer has to ask for permission before it can draw on your screen.

Negotiating a Network Connection

Data is sent over the Internet in relatively short pieces, analogous to small shipping boxes. The amount of data in a typical request is much too large to be shipped in one piece. If you use a web browser (a type of client, by the way) to download a web page, even a single web page may require dozens of packets. The server must therefore divide the page into chunks, and your computer must re-assemble these chunks before the page can be displayed.

The means by which computers arrange to share data is a little involved, but elegant. When two computers want to “speak”, they need to verify that they are speaking to whom they think. The usual procedure, called a *three-way handshake*, authenticates each computer to the other. The client initiates a transaction by sending a short packet to the server; the packet says, “I want to initiate communication. My sequence number is 36451.” (The number is chosen by the butler. The more random the number, the better for security.) The server receives this packet and sends a message, “I acknowledge receiving your synchronization request, and the next packet I expect to receive is 36452. My sequence number is 912.” The incremented sequence number proves that the server really did receive the original request. Finally, the client authenticates itself to the server by saying, “I expect to receive packet 913 from you. Here is the information I want...”. At this point, the two computers have established a connection, and each knows exactly where in the conversation it is.

Connection established, the server splits the information into pieces small enough to be transmitted over the Internet. If the reply is split into 156 packets, the server starts sending data numbered, “Page 1 of 156” and so forth. Your machine now knows to expect 156 packets in reply. Each time your machine receives a packet, it sends an acknowledgement to the server. It does not matter if the packets arrive out of order, because they are numbered. (Because the Internet routes traffic dynamically, there is a good chance the packets will *not* arrive in sequential order.) If the server has not received acknowledgement of receipt of a packet after a decent interval, it sends the packet again. Once all 156 packets have been transmitted successfully, the connection terminates.

The procedure outlined here is called the *transmission control protocol*, TCP. Email, web browsing, file and database access, and remote login sessions are conducted via TCP. The complete technical details can be found at <http://www.faqs.org/rfcs/rfc761.html> if you're curious.

Port Numbers

As already suggested, a single computer can act simultaneously as a server in many capacities. When a request comes in, how does the computer know which program should receive the request? The answer is that different kinds of service are associated with an address analogous to an apartment number or room number. There are dozens of common services, but there are plenty of available numbers, so it is no problem to assign each type of service a different number. The “apartment number” is technically known as a *port number*, and common network services are *bound to* port numbers. For example, web page requests go to port 80, requests to locate the address of a hostname (something like “directory information” for the Internet) go to port 53, file transfer requests arrive on port 21, and when a Windows computer boots, it asks on port 137 for an address.

Recall that at the start of a packet are a destination address and a source address that tell the Internet where the packet came from and where it is going. In addition to the destination address, a packet has a *source port number* that tells the butler what remote program sent the packet, and a *destination port number* that tells the butler the program for which the packet is intended. The butler sees the destination port number and hands the packet to the appropriate servant, er, server. You might think of the foyer of your house having several mail slots; not all the slots will be in use, but each server has its own mail slot.

Port Scanning

If you were a burglar, you would case a neighborhood, looking for houses that are easy to break into, testing door locks, telephoning to see if anyone is home. The network analogue is a *port scanner*, a program that sends packets to every allowable port at a given address and examines the replies. On the basis of the returned information, the scanner determines information about the host, such as its operating system and a list of servers waiting for incoming requests.

It is easy to portscan not just a single address, but hundreds of hosts who are on the same street (in the same *address block*). There are programs that do this automatically, and which sort and tabulate the data they receive according to your system resources and vulnerabilities. If you think no one will notice that your print sharing is turned on, or that you’re running a vulnerable file server, you’re mistaken. Expect to be port scanned about once an hour all the time you’re connected to the Internet. Because scans can be run from anywhere, they are equally likely to occur at any time of the night or day.

For obvious reasons, it is impolite to port scan machines at random, but it is not illegal in the United States. It *is* forbidden by many ISPs, but enforcement is lax. Even if scanning were illegal, prevention and enforcement are impossible. Attackers typically scan blocks of addresses from stolen computer accounts, and the electronic trail often leads back to a country with no laws against system cracking. All you can do is to ensure your machine gives away little information to a port scanner.

Effective port scanners are widely available, but far from wishing they were illegal you should embrace them as a valuable tool: By running a port scan on your own machine, you can find probable entry points for intruders, and close them up. A port scan is often the easiest way to determine if a machine has been attacked; any unauthorized services (such as the login server that gives an attacker remote access to your machine) will show up in a portscan, even if the operating system claims nothing is running on the port. Think of a port scanner as an external Tiger Team auditor who can tell you what's *really* going on with the servants.

Stateless and Stateful Firewalls

A firewall is a sort of door guard or mail censor. The firewall examines the address information and port numbers of each packet, and acts on instructions you give. The firewall can allow packets to pass silently, send them through but post a cautionary note in a log file, drop them in the trash without a word, or many other things. For example, some source addresses are, like 1234 Main St., Anytown USA, obviously bogus; you would probably tell your firewall to throw away any packet claiming to come from such an address.

The simplest kind of firewall determines the fate of a packet solely on the basis of information in the packet. In order to block incoming requests to use your printer, you could have your firewall block any packets that contain a synchronization request. Such a firewall is said to be *stateless*; it has no memory of an existing transaction, and makes decisions packet by packet. The Linux 2.2 kernel utility `ipchains` offers only stateless firewalling.

Stateless firewalls work well for home use, but are a little gullible. The address and port information in a network packet can be forged easily. If your firewall blocks only synchronization requests, an attacker can still conceivably get past the firewall by sending a packet that pretends to be an acknowledgement of a request from your machine. A stateless firewall may well admit such a packet. A *stateful firewall* keeps track of existing TCP connections; if an acknowledgement arrives but your machine has sent no synchronization request, the firewall can be

instructed to drop the packet. The Linux 2.4 kernel offers the stateful firewall utility iptables.

Firewalling capabilities are built into the Linux kernel. By contrast, some operating systems relegate firewalls to application programs. If you are truly concerned about remote exploits, you should not run your firewall on a single-user operating system.

Operating Systems and Security

Many popular PC operating systems cannot be made the basis of a secure Internet host; they were designed for user convenience rather than for security, and the foundations upon which they were written reflect their origins. If you run one of these, there is not much you can do but disable automatic execution of programs in email messages and web pages, turn off file and printer sharing, and be very careful about where you go on the Internet. Some features, such as enabling of Java and JavaScript for web pages, may turn themselves on automatically even if you disable them, so every time you start a web browser it is prudent to double-check that these scripting capabilities are turned off.

If you are concerned with security in your home computer(s), GNU/Linux is an excellent choice of operating system; it is a Free, low-cost, professional-quality operating system that runs on several common types of PC. Its main downside is the relative lack of user-friendliness, and the difference between its user interface model (often a command line and simple text editing) and that of “point-and-click” operating systems. GNU/Linux is not trivial to configure, but once properly configured will run stably and safely for months at a time.

Using Free software gives you good control over the use of your machine; when serious bugs are found, they are publicized in well-known places, and patches (corrections to the software) are often available within days—sometimes hours—of publication of the bug. Running GNU/Linux is something like designing and building your house from high-quality, pre-made parts. If you don’t like the way something runs, its inner workings are documented. You are free (hence the name “Free software”), even encouraged, to learn the inner workings of your system, and to improve them if you are able. Free software is also astonishingly cheap if you are accustomed to proprietary software. A complete “distribution” of GNU/Linux—the operating system and hundreds of utilities and application programs, including excellent tools for writing and compiling software—costs about US\$5 at this writing, and can be downloaded at no charge from many

sites around the world. Equivalent tools for other operating systems could easily cost \$2000 or more.

There are many resources on the web for installing, configuring, and using GNU/Linux. If the idea of using Free software appeals to you, please see the URLs at the end of this paper for places to look for more information on getting started.

Securing GNU/Linux

Linux, the kernel of the GNU/Linux operating system, is a multi-tasking, fully memory-protected clone of Unix, the mainframe operating system with 30 years of development behind it. GNU/Linux offers all the user security of Unix; different users cannot read or destroy each others' files, and only the *superuser*, root, has complete permission to modify or delete system files. If you, as a GNU/Linux user, fall prey to a URL attack, you cannot compromise more than your own account; it is not so easy to gain superuser access remotely.

That said, there are a handful of potentially serious risks in using certain distributions of GNU/Linux. Luckily, they are easily fixed. To explain them, we must first outline what happens when you install and boot GNU/Linux.

By default, the most popular distributions of GNU/Linux (Red Hat, Caldera, and Mandrake) set up your computer as a server for web documents, databases, files, electronic mail, and other common network facilities. After you install and boot for the first time, your computer will be ready and willing to answer requests from the wide world. To many converts from other operating systems, this plethora of abilities is incredibly nifty; for \$5, you've gotten hundreds of dollars' worth of software identical to what professionals at large businesses and universities use. To a system cracker, a newbie's GNU/Linux box is a gold mine.

By portscanning an address block, an attacker can easily find your machine, and with a few widely-available tools can tell what distribution and version you've installed. The companies that package distributions have to date been relatively lax about security, and several of the servers that are installed and enabled by default can be *remotely root compromised*. This means, in plain language, that if you install Red Hat 6.2 (or whatever) then leave your machine connected to the Internet, you have built a house capable of launching powerful attacks on other computers, but have left the door open for anyone on the Internet to walk in and take complete control.

The five most notorious servers are sendmail (electronic mail), bind (Internet name resolution service, or DNS), wu-ftpd (the Washington University file

transfer protocol daemon), `rpc-statd` (a server useful on networks of Sun machines), and `LPRng` (a print server). Learn how to disable these, and turn them off before bringing your machine onto the Internet. If you must run these services, get up-to-date patched versions (with no published vulnerabilities) and possibly firewall the ports so they are inaccessible from the Internet.

As of February, 2001, there are *fully automated* worms (Ramen and liOn, see Appendix B) traversing the Internet that scan for vulnerable machines, then attack one or more of the services listed above, gaining superuser access. They install copies of themselves and mail password and system configuration files to four email accounts (two in the U.S., two in China). They install a few backdoors, allowing a human to log on remotely as the superuser, and clean up traces of themselves, deleting lines in log files and replacing system utilities with versions that do not show existence of the worm. Then they begin to scan for more vulnerable boxes. In the forensic analysis of one Ramen attack, the elapsed time between the initial port scan and total remote ownership of the machine was 87 seconds. The Honeynet Project reports that the *mean* time until an unsecured Red Hat machine is root compromised on the Internet is 72 hours; one of their “honeypots” was rooted 15 minutes after being brought online. See Appendix B for the URL of the Honeynet Project.

A Computers and Information

Computers are electronic machines that store and manipulate data at incredible speed. Computers can be connected to each other (or *networked*) in a way that allows them to share data. When you send someone an email message, a chunk of data called a *file* is transferred from your computer to your friend’s computer. When you browse the world-wide web, files on a web server are copied and sent to your machine, where a web browser converts the information they contain into the text and images you see on your screen.

Text (email messages and web pages), proprietary format files (word processor documents or spreadsheets), and multimedia files (sounds and pictures) are common types of data on a typical home computer. The most important attributes of electronic data are that it can be copied quickly and repeatedly, and that the information can be transmitted to another machine anywhere in the world, as long as both machines are connected to the Internet.

Electronic Representations of Data

Abstractly, the information in a computer is stored as a pattern of *bits* (binary digits), often represented as a string of 0s and 1s. If you are already starting to feel lost, please keep reading; the details must be explained bit by bit (no pun). If computer internals are new to you, there will be short intervals of confusion as you read, followed by pleasant moments when you suddenly understand how a piece fits into the larger picture.

In modern computers, the smallest unit of data that can be manipulated directly by the hardware is a *byte*, consisting of eight bits. There are 256 bytes, namely 256 distinct sequences of 8 bits. These are listed in Table 1; the decimal numbers at left serve to enumerate the bytes, while the typewriter strings are the actual bytes.

0	00000000	64	01000000	128	10000000	192	11000000
1	00000001	65	01000001	129	10000001	193	11000001
2	00000010	66	01000010	130	10000010	194	11000010
3	00000011	67	01000011	131	10000011	195	11000011
4	00000100	68	01000100	132	10000100	196	11000100
	⋮		⋮		⋮		⋮
62	00111110	126	01111110	190	10111110	254	11111110
63	00111111	127	01111111	191	10111111	255	11111111

Table 1: Eight-bit bytes

Think of the eight bits in a byte as numbers on an odometer. When a wheel reaches the highest number, it rolls over to zero, and the next wheel to the left increments. The difference with an ordinary odometer is that here each wheel has *only two positions*, 0 and 1. If you are unfamiliar with binary arithmetic, you may enjoy writing down a few more odometer readings until you get a feel for the pattern.

The names 0 and 1 are conventional; they could just as well be called *false* and *true*, or *off* and *on*, or *black* and *white*. The significance of this will become clear shortly.

Character Coding

Suppose you wished to store a text document as a string of bits. One method would be to let letters of the alphabet correspond to counting numbers from 1 to 26: $a \leftrightarrow 1$, $b \leftrightarrow 2$, $c \leftrightarrow 3, \dots, z \leftrightarrow 26$. The word “cat” would correspond to the list of three numbers 3 1 20, and the list 4 15 7 would correspond to the word “dog”. If everyone in the world agreed on this coding scheme, then storing a text message would be tantamount to storing a (rather long!) list of numbers, one for each character of text. Computers are built for just such translation tasks; translating text into a string of numbers is simple and repetitive, but needs to be done quickly and accurately.

What does this have to do with bits and bytes? By Table 1, the numbers between 0 and 255 correspond to strings of eight bits. If we combine this with our earlier text-coding scheme, we would get, for example,

cat \leftrightarrow 3 1 20 \leftrightarrow 00000011 00000001 00010100.

Provided the computer parses the string 000000110000000100010100 correctly, it can store the word “cat” as a sequence of bits. Bits can be encoded by other equivalent schemes. Using our primitive character coding, “cat” might be stored on a bar code or compact disk as light and dark spots:

000000110000000100010100 \leftrightarrow ○○○○○○●●○○○○○○○●○○○●●○○○

The correspondence just described is not quite adequate, because actual text does not consist solely of lower-case letters, but also includes capitals, numerals, and punctuation. However, you have probably realized already that we had 256 numbers at our disposal for encoding, and have used only 26 of them. Allowing for upper- and lower-case letters, numerals, punctuation, and various other characters (such as space, tab, carriage return, and delete) requires 128 different symbols. Thus, all plain text can be encoded as a string of bytes (one byte per character), and can be decoded so long as the author and reader agree on the precise scheme of encoding and decoding. A common standard, published in 1968 by the United States American Standards Institute, is *ASCII*, the American Standard Code for Information Interchange. In ASCII, the phrase “A line of text.” (16 bytes including the carriage return) encodes, with spaces between bytes for clarity, to

```
01000001 00100000 01101100 01101001 01101110 01100101 00100000
01101111 01100110 00100000 01110100 01100101 01111000 01110100
00101110 00001010
```

Using ASCII, any two people in the world can communicate using the Roman alphabet, storing and transmitting text as a string of bytes. The International Standards Organization has published a host of character encodings to handle non-Roman alphabets. In view of the translated snippet above, ASCII may seem terribly inefficient, but computers are capable of storing unimaginable amounts of bytes. A page of text is roughly a thousand bytes, while a modern hard drive stores tens of *gigabytes*, tens of millions of pages of text. Moreover, if you examine the size of a word processor file, you'll find it is about 20 to 100 times the size of the raw text it contains. Such proprietary encodings are far less efficient than ASCII, and are much slower to transfer over slow connections such as modems.

A crucial point to remember is that the string of bytes itself has no intrinsic meaning; the original text cannot be recovered from the string of bytes unless the coding scheme is known. Sometimes this is desirable, as when one encrypts a message, and other times it is not, as when one wants information to be as widely available as possible. The free sharing of information between people depends upon an open standard of encoding that anyone may use. Proprietary encoding standards (word processor files, for example) are common, but have the unfortunate effect of excluding people who do not have access to programs that know the coding scheme.

Other Data Types

Almost any information that can be stored in physical form—a photograph or painting, a written work, an audio recording—can be stored as a stream of bytes. Naturally, the more complicated the information or the more detailed the description, the more bytes are required. (For a photograph, “more detail” means “higher resolution”, while for an audio recording it means “better fidelity”.) A string of about 10 million bytes is enough to encode an ordinary photograph with reasonably high resolution, or to store a 3 minute pop song, both using common (and not necessarily efficient) coding schemes. For illustration, here is roughly how the process works for photographs.

If you look closely at a color television screen or computer monitor, you will see a tiny honeycomb of phosphorescent spots grouped in threes. In each group is a dot capable of producing red, green, or blue. All the colors that a monitor can display are formed by a mixture of these (“subtractive”) primary colors in varying brightnesses.

Each trio of dots is so small as to appear a single spot, or *pixel*, to the human eye. The color state of a pixel can be described by three bytes: The first byte

(which, as above, represents a number between 0 and 255) encodes the brightness of red in the pixel (with 0 meaning “off” and 255 meaning “all the way on”), the second byte records green, and the third encodes blue. This particular coding scheme is called “24-bit RGB” for obvious reasons. In 24-bit RGB coding, a stream of bytes is interpreted not as a string of characters, but as a sequence of pixel states. Again, it is not the bytes themselves that are important, but *how they are interpreted*.

The number of colors that can be represented with 24 bits is $256 \times 256 \times 256 = 16,777,216$, or “millions of colors”. Five years ago this was state-of-the-art, but nowadays 24-bit color is considered somewhat inadequate; with 32 bits (four bytes), a computer can display over 4 billion colors. This approaches the limits of the human eye to distinguish. Good modern scanners, which are hardware devices that convert photographs into streams of bytes, use a 40 bit (five byte) data encoding.

Describing a 24-bit RGB image requires three bytes per pixel. The *number of pixels* required to describe a picture depends upon the resolution of the picture. A relatively low-resolution picture requires $600 \times 480 = 288,000$ pixels, or about one million bytes of 24-bit color, while a high-resolution rendering might require $1600 \times 1200 = 1,920,000$ pixels, or about ten million bytes of 40-bit color. The main point is that even a photograph can be stored as a long string of bytes.

Data Storage

As mentioned above, data of many types can be stored as a string of bytes. Bits themselves (eight to a byte) simply correspond to one of two possible alternatives, or “states”. In your computer, bits are represented physically in three primary ways:

- In “volatile storage”, or *RAM* (random-access memory). A computer memory chip is a lattice of millions of microscopic electronic switches that can be turned off and on in a few *billionths* of a second. If power is not supplied continually to the chip, the switches will begin to turn off spontaneously after a few million clock cycles (or about 1/1000 of a second), their states (whether they are “off” or “on”) will drift, and the stored information will be lost.

Perhaps nowhere else in a computer does one encounter the ephemerality of data; a photograph or audio recording can be encoded as a string of bytes and stored in memory. The stored information can be retrieved in its entirety

unless the power is interrupted, in which case the data will vanish almost instantly.

- In “permanent storage”, on the *hard drive*. The hard drive of a modern PC consists of one or more rigid magnetic disks, sealed inside a metal case about the size of a paperback novel. When the drive is operating, the disk spins rapidly (about a hundred times per second). Data is read from and written to the drive by means of a magnetic head that is mounted a small fraction of a millimeter from the surface of the disk. In a data-writing operation, the magnetic polarity of the head flips rapidly, magnetizing the disk as it passes the head. The pattern of the magnetic domains on the actual disk is abstractly equivalent to a stream of bits. Data is read from the disk in the reverse operation: When the disk spins past the head, the head detects whether adjacent domains on the disk have the same polarity (0) or opposite polarity (1).

Like an audio cassette (or a bar magnet!), a hard drive does not require a constant supply of electricity in order to store data; magnetic domains retain their polarity until they are re-written. A floppy disk is similar in principle to a hard disk, but can store far less information, and is slower. The main advantage is that a floppy disk can be removed physically from the machine.

- In “read-only” storage, such as a *CD-ROM*. A compact disk consists of a thin sheet (about 1 mm thick) of transparent acrylic, coated with a microscopically thin layer of reflective aluminum. As the disk spins in the drive, a laser shines on its surface, and is reflected into a photo-detector. To store data, small holes or pits are burned into the aluminum layer; as the disk spins past the laser, the beam is either reflected by the aluminum layer when there is no pit (0), or is not reflected if there is a pit (1). The stream of pits winds around the disk in a tight spiral reminiscent of the groove in a phonograph record; the entire line of bits on a full disk is about 1 kilometer long, and stores several hundred million bytes of information. Writable compact disks work on similar principles.

The advance of computer technology in the last two decades of the 20th Century is truly mind-boggling. The ingenuity required to develop the means of writing, storing, and retrieving huge amounts of data is awe-inspiring, and the success is phenomenal. At this writing, 128 megabytes of RAM costs about US\$50, 20 gigabytes of hard drive storage costs about US\$100, and a 650 megabyte compact

disk can be bought in bulk for less than US\$0.20! Good-quality computer hardware is incredibly reliable: A hard drive can be expected to mis-read or mis-write one byte in every million billion bytes. This is roughly equivalent to copying the entire Library of Congress and making a single typo.

Processing

A computer's other salient attribute (the first being data storage) is rapid *processing* of data. Abstractly, processing is the rapid execution of arithmetic operations on short strings of bytes, called *words*. Most modern processors use four-byte words, and are therefore "32-bit machines". Eight-byte words (64-bit architectures) are slowly being phased in, and will be the norm in a few years.

Decision questions of the the form, "If *A* is true, then do *B*; otherwise do *C*," can be phrased in terms of binary arithmetic. Binary arithmetic, in turn, can be represented physically by simple electronic circuits called *logic gates* built from two or three transistors. A modern PC processor consists of several million microscopic transistors etched into the surface of a silicon wafer. A clock on the chip emits pulses periodically, which serve to synchronize the activities of the processor and data coming in from main memory (RAM). The *clock speed* of a processor is measured in "ticks" per second; a 600 MHz processor's clock ticks *six hundred million* times per second. If time were scaled to one clock tick per second, one second of real time would last almost 20 years. To a computer, our familiar world is almost stationary.

The core of the processor consists of the actual processing unit and several *registers*, small memory locations where data is held during the few clock ticks when it is needed. The processor chip also has a relatively small amount of memory on it, about 32 thousand bytes worth of *level one cache* that can be accessed at full clock speed, and a larger amount (a few hundred thousand bytes) of *level two cache* that be accessed at a slower speed, usually half the clock speed.

At the core of the processor, data queues up waiting to be operated upon. A secretary keeps track of the location of the data to be operated upon at the next clock tick, known as the *stack pointer*, and the location of the next operation to be performed, the *instruction pointer*. These pieces of information change from tick to tick according to the outcome of the operation itself. By performing hundreds of millions of simple decisions per second, a computer exhibits intelligent (or at least responsive) behavior.

Remote Exploits

Aside from the intrinsic interest of how a computer works, the description above is necessary to understand how a machine can be tricked into following rogue instructions. Imagine first that the computer is running a file server. Because the file server needs to have access to files owned by different users, the process has root privileges; anything the file server wants to do will be allowed by the operating system.

In order to accept and respond to requests for file transfer, the file server sets aside space in memory, called an *input buffer*, where information related to a request can be written temporarily. The buffer holds, say, 256 bytes, large enough to accommodate any reasonable transaction a user might request, and then some. The crucial error is that *this* file server does not check the size of an actual request as it fills the input buffer; no *reasonable* request will fill the buffer, but a program should still check users' input to ensure the length conforms to the available space. In our case, an attacker has found, either by carefully auditing the source code, or by accident, that there is no checking of the input buffer's size, and that the input buffer and the operating system's instruction pointer are in abstractly adjacent sections of memory. The attacker carefully crafts a fake client request that exactly fills the input buffer with a small program, and then writes a few bytes of data that must be placed with precision; a missing or extra byte of padding will spoil the attack. The small program asks the kernel to start a shell (command prompt) with root privileges on the file server's port; the extra data that overflows the input buffer overwrites the kernel's instruction pointer.

Now the stage is set. The file server sits waiting for incoming connections. The kernel sees a seemingly legitimate synchronization request arrive for the file server, makes a note of the next instruction it will perform when the file server has finished, and hands off control of the processor. The file server blindly copies the incoming data—the attacker's request for a root shell—to the input buffer, exactly filling it. If that were all, the file server would ask the kernel what to do next, the kernel would say, "Parse the input," the file server would discover that the input is not a legitimate request, and would harmlessly send the attacker an error message. However, a few bytes of data remain in the incoming signal; the file server, unaware of any problem, writes these into the adjacent memory. But as the attacker carefully noted, the next few bytes of memory in the machine are the post-it pad where the kernel wrote the next instruction to be performed. With the last few bytes, the attacker has overwritten the memo, replacing the next legitimate operation with the *the attacker's request for a root shell*. The file server

says to the kernel, “Input received. Now what?” and the kernel, who trusts the file server completely, follows the attacker’s pointer and starts a root shell on the file server’s port. The attacker, at the far end of a remote connection to the file server, now has complete control of the machine.

The attack just described is a typical *buffer overflow* exploit. Careful programmers do not accept user-supplied input without checking that it conforms to what the program expects, but potential buffer overflows can be difficult to find in existing programs. The five services listed earlier are periodically found to be vulnerable to buffer overflows. Single-user operating systems and their software are particularly vulnerable because there is no concept of a non-root user, so *every application* is potentially a source of attack.

Unfortunately, though patches to vulnerable programs (for GNU/Linux) are usually distributed almost simultaneously with the exploit report, many system administrators do not immediately patch their software. There is, consequently, a window of opportunity, sometimes lasting months or years, between the time an exploit is announced and the time when the last vulnerable machine is patched. If you are a system administrator (for example, if you run GNU/Linux on your PC), it’s a good idea to subscribe to CERT’s mailing list, and a good habit to patch exploitable services immediately.

Script Kiddies

Buffer overflows and other remote attacks may seem abstruse and of little cause for concern, but they are far more widespread than the difficulty of writing one might indicate. The real risk is that once a vulnerability is found, an exploit is usually published quickly, as proof of concept. The source code for exploits gets posted in well-known locations on the Internet (do a google search for “hacking” or “alt.2600” and see what you find), and maladjusted youths know where these sites are. The end result is that thousands of kids (usually male teenagers) who know almost nothing about computers are capable of successfully getting root access on vulnerable machines. It is darkly comic to view system logs of compromised machines; attackers often bumble around the exploited machine, trying to figure out how the operating system works.

In security circles, these attackers are known generically as *script kiddies*, and are an annoyance more than a danger. The script kiddie subculture is pathetic but tragic; many are intelligent children with unhappy family or school lives. Being a computer outlaw gives them a sense of worth and purpose, malign though it is. Lance Spitzner, a system security expert, has written a series of excellent papers

on attackers (mostly script kiddies) that can be downloaded from the web, see the URL for the HoneyNet Project. These papers range from a fascinating view into the psychology of the “typical” script kiddie, to the actions they perform once they gain control of your computer.

Availability and Legality of Attack Tools

Every attack tool has a generally beneficial use, namely to test systems for vulnerability. It is in legitimate users’ best interest that these tools be *easily available and freely usable*. To understand why, consider that—as computer programs—attack tools consist of data. They can be written by any competent programmer regardless of motivation, and once written can be freely copied and distributed. Short of turning the Internet into a virtual police state, there is no way to restrict access to attack tools, any more than the exchange of ideas or information can be curtailed. Because the Internet extends worldwide, there is currently no legal entity that has global jurisdiction anyway, so even Draconian laws in one country will not stop the authorship and distribution of attack programs. If these tools are made illegal, users who do not wish to break the law (or for whom the penalty of use is truly dire) will lose the potential benefits, while attackers will not.

Though this argument is superficially the same as the rationale for keeping firearms legal, the fundamental issues with software are completely different. First, a gunshot may kill or irreversibly injure someone. A computer attack may destroy data, but you can make an exact backup copy. If guns were truly analogous to attack software, one could resurrect gunshot victims. Second, the deterrent argument with guns is the threat of retaliation; a mugger is less likely to attack you if he thinks you may shoot him. With attack programs, the benefit is not to deter system crackers, but to obviate their attempts to attack your computer. By using portscanners and remote exploit programs against your own machine(s), you can search for and repair potential vulnerabilities, so that attacks against you are simply ineffective.

Restrictive laws on the use of attack software are seductive because they seem to protect innocent users, who cannot be expected to stay abreast of newly-discovered vulnerabilities, security patches, and other developments. But the government’s job is not to protect citizens as parents protect infants, nor can it be in the case of computer technology, without becoming totalitarian. The only way for citizens to protect themselves and their personal data in electronic form is to give all responsible citizens the means to do so, both in terms of education and in availability of appropriate software tools.

Computers and data are, in fundamental quantitative ways, different from any prior or existing technology, and the legal establishment is working diligently to legislate societal conduct with respect to electronic information. The coming years will be a time of rapid and revolutionary change in computer law. If we concerned citizens do not educate ourselves of the technological, legal, social, and economic issues surrounding computers and their use as vehicles for financial and legal transactions, we will bear the brunt of laws drafted in the interests of a few wealthy corporations. These entities have a right to do business for a profit, but their aims are often at odds with the (American) Constitutional guarantees of Freedom of Speech and the Right to Privacy. Technology is changing rapidly, and the process of self-education is daunting and ongoing, but as citizens of a Free Society, we would all do well to regard computer software issues as matters of Free Speech, rather than as trade secrets designed to protect corporate profits, or as the purvey of experts we cannot hope to understand.

B Web Sites for Further Information

I am not personally affiliated with any sites listed here.

- <http://www.google.org> (Search engine)
- <http://www.fsf.org> (The Free Software Foundation)
- <http://www.opensource.org> (The Open Source Project)
- <http://www.linuxdoc.org/links/index.html>
(The GNU/Linux Documentation Project)
- <http://www.cert.org> (CERT/CC)
- <http://project.honeynet.org> (The Honeynet Project)
- <http://www.whitehats.com> (Max Vision's security site)
- <http://www.whitehats.com/library/worms/lion/index.html>
(Analysis of the li0n worm)
- <http://www.linuxsecurity.com>
(A security portal for GNU/Linux users)

- <http://www.attrition.org> and <http://www.insecure.org>
(Hacker sites in the best sense of the word)
- http://www.cultdeadcow.com/cDc_files/cDc-351
(*The Tao of Windows Buffer Overflow*, a detailed but salty technical paper on writing buffer overflow exploits.)