Internet Insecurities Are Not Diseases

Paul N. Hilfinger*
University of California at Berkeley

13 July 2000†

As the Internet grows in size and importance, so too do malicious or mischievous attacks on it and on the computers it connects. After each such attack, the public is treated to the opinions of experts conveying a consistent set of messages. We are told that (1) computer criminals will always be able to kick holes in computer security at will; (2) these incursions will always cause some initial damage, which can be limited only by vigilant and repeated installation of the latest virus-scanning software; and (3) although there are methods by which users can avoid such damage, they all involve giving up significant useful functionality or convenience (such as the ability to click freely on e-mail attachments). In fact, these statements are all false. Their constant repetition has led the public to accept a significantly lower level of security than is possible. It is entirely feasible for software vendors to radically and continuously improve computer security with virtually no impact on the convenience of their customers, and it is the responsibility of the public to insist that vendors do so.

In the wake of a well-publicized attack, experts typically advise the public to take various draconian protective measures: don't click on attachments; turn your machine off when not in use; don't open spreadsheets or Word documents unless you are absolutely sure of their provenance. Likewise, manufacturers, in lieu of building the secure infrastructure they could, will instead simply disable or otherwise increase the inconvenience of the problematic features, as Microsoft did to its Outlook e-mail program in response

*Author's address: 387 Soda Hall M.C. 1776, Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA 94720. Telephone: (510) 642-8401. E-mail: Hilfinger@cs.berkeley.edu.

†Fixed typos 12 January 2003. Thanks to Bradford Wilson.

to the recent Love Bug. Of course if e-mail attachment really is a valuable feature and not a mere frill, one can't reasonably avoid clicking on attachments; active attacks on your machine can happen even as you use it, so turning it off provides limited protection (and is impossible if your machine acts as some kind of server); and if one has to deal with the general public, it is rather difficult in practice to vet every single sender or application. None of these measures are inherently necessary. The technology exists to allow safe and uninhibited use of most of the features of modern mailers, browsers, and other office software.

The prime culprit in the public's current state of misinformation is what we might call the *disease metaphor* for computer break-ins. This pictures a malicious program as an active agent (hence the term *virus*) that, like a biological disease, is contracted by contact with the outside environment and is often infectious. A perfectly healthy human, free of disease or genetic defect, will nevertheless fall victim to a new infection different from what his immune system has previously encountered. This difference, furthermore, needn't be fundamental—a relatively small mutation will do. In other words, it is perfectly normal and expected that humans will inevitably fall victim to an endless succession of diseases during their lifetimes. Although protective vaccines are possible, they are in general imperfect, are developed only after some members of the population get seriously ill from a new strain of disease, and cannot be manufactured quickly enough to deal with all infections. These properties are very familiar to us, which is what makes the disease metaphor useful as an explanatory device, and also powerfully misleading.

A more accurate metaphor is that of the computer (or operating system) as a fortress armored with an impregnable sheathing in which, due to hasty construction, there are numerous chinks. According to this metaphor, there are finitely many chinks and as they are repaired, the fortress more and more closely approaches invulnerability. Furthermore, no simple variation (read, mutation) of an attack that once succeeded in getting through a chink will ever succeed again, once that chink has been repaired; the would-be attacker must find an entirely different chink. In other words, the task of breaking into the fortress becomes increasingly difficult, and the information that led one attacker to a chink in the armor is useless to his successors. Furthermore, it is not the attacker's prerogative to blast a new chink, but rather he must find previously existing chinks, an enterprise that becomes effectively impossible to larger and larger classes of would-be criminal. It is this metaphor that should inform public thinking about the security of their systems, and the

properties they may reasonably demand of their vendors.

1 What you are entitled to expect

Given the state of technology I will describe below, you might legitimately expect a great deal more in the way of convenient security than current software provides.

- It can and should be safe to open any attachment or document—be it text document, spreadsheet, or executable program—without fear that your data or programs will be damaged, or any worms inserted in your computer.
- Executable attachments and documents from sources that you have previously arranged to trust can be reliably verified to come from those sources and be executed without your further intervention.
- Executable attachments and documents from sources in which you have not indicated trust can be run without intervention, and reliably restricted by your system so as not to perform potentially damaging acts.
- In those cases where such untrusted executables actually require operations that change parts of your data, your system can automatically ask permission for the specific operations to be performed and reliably restrict access to what you have explicitly permitted.
- You can arrange to partially trust certain sources of executable objects, and to pre-arrange permission for exactly the operations these objects require.
- All of this can be engineered to be convenient.

2 The good old days

The technology I refer to is not new or experimental, nor is the (perhaps even more important) attitude that security loopholes are repairable security bugs. Unfortunately, computer software went through a kind of "historical knothole" with the advent of the personal computer. Previous systems, being

very expensive, had to be shared, and the need to share brought with it a need to protect users from each other. As a result, in the 1960s and 1970s, computer security was an important issue, and an active research topic. The early designers of the personal computer, unfortunately, suffered from a collective failure of the imagination, and as the name implied, saw the new systems exclusively as isolated, private computers. Thus was over a decade of computer security technology discarded. It has gradually been restored in more recent versions of Windows and MacOS, but a legacy of careless security practice and culture still remains to be corrected (even, to some extent, on UNIX systems, which largely retained the older attitudes).

In the abstract, the problem of computer security is to somehow monitor the operations performed on a system—whether it be a single computer or a network of them—so as to enforce some *security policy* about which users (or *agents*) may perform what operations on what data. The challenges are to formulate policies that correctly reflect our needs, and then to find mechanisms that enforce these policies reliably and quickly. I'm not going to address the first challenge here, since I don't think this is the most pressing problem, and will instead concentrate on enforcement.

The single computer. Before the personal computer, security problems involved single computers shared by mutually distrustful users. Proper security boiled down to preventing one user's programs from writing to or reading from another's files without authorization to preventing them from reading or writing the primary memory of any other user's program. Implementing security in this world is, in principle, easy. Computer processors typically provide ways to restrict the memory available to a running program, and to allow input and output operations or changes to memory restrictions only to programs that run in a "privileged mode." Any program can enter privileged mode, but only by transferring control to a pre-arranged program, which is typically part of the operating system (and located in memory that the program cannot change). These arrangements are enforced by the computer hardware; by design there is no way to change them or to trick the hardware into breaking the rules. The effect is to give all control over damaging operations to the operating system. Significantly—and as evidence of the failure of imagination I mentioned earlier—the hardware of early PCs lacked a privileged mode. The legacy of this gap still persists in widely used operating systems.

Attacking the single system. There are three ways for a security breach to occur in single-computer systems as I've described them. First, there can be an error—a bug—in the operating system so that it fails to enforce the restrictions it was designed to (possibly allowing itself to be corrupted). Second, the method of identifying a user to the operating system (as needed by security policies that allow only certain users to touch certain data) can be subverted, which typically boils down to guessing a password. While this means of break-in is significant, it is also controllable: passwords can be made longer so as to reduce the probability of guessing them to acceptably infinitesimal levels; likewise, systems can limit the rate at which they accept passwords when incorrect ones are proffered; and users can be educated in proper "password hygiene." The system side of the password problem, in other words, is readily solved and hygiene is legitimately the responsibility of the user. For the rest of this paper, therefore, I will concentrate on what I see as the more difficult problem: closing security holes due to software bugs.

Third, it is also possible for a malicious user to do as he pleases by modifying the hardware. This is one reason that expensive, shared machines were physically secured in locked rooms. But of course, physical security is not a new problem, and not peculiar to computers; it applies just as well to one's wallet. The security of individual computers is not really the source of current problems, and in any case, the same old-fashioned methods that apply to banks and personal possessions apply to it. We will have to return to the issue of physical security when we get to the Internet.

Sandboxes. There is no particular reason why the security policy applied to a particular program has to be "allow unlimited access to all of Smith's files." In fact, one could design a system to allow any programmable test, including "Allow reading only of all Smith's files in directories (folders) X, Y, and Z" or "Allow creation only of up to 20 files, and only with names that have the form F," as well as resource limitations such as "Allow the program only 10 seconds of CPU time" or "Allow the program to write a total of at most 100K bytes." Such restrictions can be quite general, can in principle be programmed quite easily, and in fact have been implemented in some operating systems. The Java system provides the notion of a "security manager," which it consults to know if a user program's requested operation is legal. The security manager can be changed only once, so a monitor program can set up a desired policy and then call an untrusted application

that must then abide by it. Such restricted environments for programs go under the general title of *sandbox.* The concept of a security manager is certainly not new; the established (decades-old) term is "reference monitor."

3 On to the Internet.

Sun Microsystems uses the slogan "the network is the computer." I've seen this sentiment called cryptic, and it probably was to those who saw personal computers as isolated computers. By now, though, the enormous value of interconnection is obvious. In effect, the power and memory of one's computer is extended to those it connects to, making the computers and the network that connects them one entity. This introduces two classes of problem.

First, if an operating-system implementer is not careful, the effective operating system of this machine is now distributed among many computers. Security bugs in operating systems can be quite subtle, and as my armored-fortress metaphor suggests, a chink at one point is all it takes for a successful attack. Therefore, when the operating system becomes distributed onto machines outside one's control, including to machines under enemy (or at least, miscreant) control, it again seems possible for adversaries to attack at will on their own terms. If another computer tells yours "here's a request from user Smith" or "here's a system program to run," then unquestioning belief on the part of your computer allows many obvious possible abuses. Clearly, therefore, foreign computers must in general be treated with suspicion.

Nevertheless, we will always have good reason to trust some select set of remote computers in particular ways. The basis for this trust is not really a technical matter: we might trust upgrades to our system software from Microsoft, for example, on the grounds of the company's reputation and of its assumed desire to avoid the legal repercussions of liability. This part of the security problem therefore reduces to reliably identifying (authenticating) the sources of data and commands that come over the network (that is, the agents supplying them) and enforcing the same sorts of security policy as previously.

Second, even if you believe that you have secured a trusted set of machines, the network connecting them is (and probably always will be) subject to outside interference from untrusted machines, and from physical attack. Internet messages arrive containing the addresses of their purported senders, but messages with forged addresses can be inserted in a number of ways, even

without splicing physical wires. Likewise, intruders can listen to messages on the internet in some of the same ways as on telephones and if they use the same local network, they again don't need to modify any hardware.

The solutions to this problem involve cryptography, which allows us to hide data from a would-be intruder, and to recognize data that comes from an intruder rather than its intended source. Modern techniques rely on *one-way functions:* computations that are easy to perform in one direction (such as encrypting a message), but extraordinarily difficult to reverse (to decrypt) without the right secret key, even for someone who knows exactly how to produce encrypted messages. With the aid of such functions, one can establish communication channels between computers without prior arrangement between the two. One can arrange also to reliably authenticate the source of any transmission. In effect, one can get the effect of having the network connecting your computer to another be as secure as if it were a wire housed on your premises (although considerably more prone to breakdown), reducing the security problem to something like the single-computer problem.

Of course, an attacker could simply guess the secret key protecting an encrypted transmission. Therefore, the cryptographic techniques I refer to will fail in the face of a phenomenally lucky attacker or one with enough computing power to try all possible keys in a reasonable length of time. However, this is not an insuperable objection, because the amount of luck or computing power required can be increased arbitrarily until break-ins are entirely out of reach of any desired class of attacker.

Denial of service. So far, I have dealt with attacks that damage or steal data. Recently, we've seen another kind of attack launched against such companies as Yahoo, Amazon.com, and AOL: the denial-of-service attack. Here, the network is flooded with legitimate-looking traffic that overwhelms its target. This kind of problem is not easily prevented altogether for a simple reason: the same effect could be produced if a large number of users were to simultaneously decide—perfectly innocently—to visit a certain site. There is no easy way to detect electronically the intent behind a flood of messages.

First, however, denial-of-service attacks, while harmful, do not do the sort of permanent damage to data and software wrought by typical viral attacks. Secondly, and more to the point of this paper, it should be possible to sharply reduce the largest subclass of these attacks: those carried out by individuals who want to remain unidentified. For example, the attacks

in February 2000 involved breaking into servers and planting programs that issued requests on behalf of the cracked servers. Tracing the origins of the flood of messages, therefore, would lead only to other unwitting victims of the attack. Obviously, this approach was intended to disguise the real villains. Had it not been possible to break into the servers—or in general to run in the guise of another user—the attack would have been considerably more limited at worst and probably would not have occurred at all.

4 Obstacles

It will not be easy to achieve the security improvements I advocate here. First, the industry's passage through the historical knothole I referred to earlier scraped off traditional security considerations, so that current software has evolved from systems that did not properly address the subject. Security measures are particularly hard to retrofit. Second, it is not at all clear how to motivate software vendors to address security properly, especially given certain unfortunate trends in the law.

Messy systems mean many chinks

One recurring comment on drafts of this paper was that the current state of personal-computer software makes the kinds of improvements I am suggesting extremely difficult. I have tried to keep my remarks relatively neutral in treating operating systems, but this is one area where Windows has come in for particular criticism.

The problem is that the boundary between operating system and application is blurred in the case of the Windows line. This complicates application of the sandbox strategy referred to earlier, which requires monitoring the set of possibly destructive requests to the operating system. It's essentially the difference between building a wall across a narrow isthmus separating operating system and applications (the "classical" notion) and building a wall around a complexly gerrymandered election district.

In their defense, Microsoft has claimed that their system structure is dictated by a desire to provide tightly integrated systems to their customers. The goal is laudable, but this oft-repeated explanation is nonetheless a *non sequitur:* the visible behavior of a system does not dictate its internal structure. Two icons may look radically different and seem geographically sepa-

rated on your screen, but the same program produces both of them, and the applications they denote make use of many shared components. Saying that an operating system will not function if its browser is replaced is like admitting that one has designed a car so that its transmission falls out when one changes the hub-caps. Tight integration as users see it comes from the careful design and documentation of the components available to programmers who write application software, and from widely published and clearly enunciated standards for interface design—not from mashing everything together into one vast, holistic mass of software.

Nevertheless, I feel some justification for guarded optimism. First, one can circumvent a messy operating system if necessary: Java implementations, for example, do not rely on the native operating system's security procedures to implement their own security. Secondly, every software company must periodically renew its software base to remain healthy. I have every reason to believe that Microsoft's technical staff is perfectly cognizant of basic principles of software engineering. After all, Steve McConnell's excellent book, *Code Complete,* treats of them, and is published by Microsoft Press. If the company eventually is broken into an applications company and an operating-system company, it will be an obvious opportunity to undertake a clean and well-documented definition of an isthmus to the operating system—to the benefit of the public and their own long-term profit.

Motivation and liability.

A big problem with really good security is its invisibility; you tend not to notice break-ins that don't happen. This property fits poorly with a culture geared to selling features. The automobile industry had a similar motivational problem. However, in the case of cars, there is a countervailing force at work: liability law. Litigation or the threat of it motivates manufacturers to address safety issues.

UCITA. The software industry, however, has had an entirely different culture. Typical end-user license agreements (EULAs) for software seem to disclaim responsibility even for basic functionality, let alone damage from attacks on security. It has not been entirely clear how effectively these shield software vendors, which, I would argue, is a good thing. Unfortunately, the part of the Uniform Commercial Code known as the Uniform Computer Information Technology Act (UCITA), which is now making its way through

the state legislatures, would strengthen the protection enjoyed by vendors from the disclaimers in their EULAs. One good way to further the cause of computer security, therefore, would be through modifications of this Act.

Inevitable error. Software vendors will claim that they need blanket disclaimers because software, being extremely complex, will always have errors, and that therefore, they need to be protected from their own mistakes to stay in business. Yet there are examples of manufacturers who appear quite willing to bet their companies on the safe, if not perfect, performance of some piece of software. Aviation companies are one example (high-performance aircraft can't remain stable without computer control, and civilian aircraft rely on computer-assisted navigation). Banks, with their computer-assisted funds transfer, are another. We know techniques of testing and internal checking for assuring an acceptable level of security (or any other correctness property) in software. It is just a matter of having the motivation to apply them.

Customer indifference. Again, software vendors will claim that their users don't care about security, and that in a free market, vendors' actions are dictated by customers' demands. First, this argument ignores our tendency to get fatalistic about situations we believe to be inevitable. That's why people are willing to live on earthquake faults. If, as I've argued, the disease metaphor has convinced the public that true security is impossible, then of course they will not insist on it. Second, the value of security in operating systems, like safety in cars, is not something that customers can easily detect, let alone evaluate. This value, after all, consists in *not* incurring the costs of break-ins, certainly a rather difficult figure to pin down. Without help, therefore, the free market will not necessarily arrive at the right mix of programming resources devoted to security vs. those devoted to functionality. Third, there is a collective component to security. Someone who breaks into my system gets another platform from which to attack yours. Your security therefore depends to some extent on mine. In other words, society as a whole has an interest in computer security that is not necessarily reflected in our preferences as individuals.

5 Answers to objections

There seem to be a number of standard responses to the criticisms and proposals I've made here that I'll try to rebut.

1. *You can never make system security perfect.* Probably, but it has been said that the best is the enemy of the good. Houses burn down despite our best efforts, yet we do not use this as an excuse to stop developing fire-resistant building materials. I only argue that we can reduce gaping holes to at worst very slow leaks. There used to be a tendency among journalists (declining now, I like to think) to describe anyone who succeeded in breaking into a system as "brilliant." In fact, one scandalous aspect of the current state of affairs is that the vast majority of successful attacks come from people who are at best mediocre programmers, using relatively simple programs. Even if it is always possible for a mad genius to break into our systems, we can at least prevent the much more common average, disaffected idiot from breaking in. Indeed, we can do better: we can limit the class of successful criminals to extremely lucky and extremely wealthy evil geniuses, and require of them ever more luck and ever more expense.

2. *There are also holes in Sun/Linux/Microsoft/...software's security.* Also no doubt true, but quite irrelevant. My purpose is not to convince anyone to switch operating systems, but to insist on improvements to what they have. The fact that General Motors must occasionally recall vehicles does not insulate Ford from the need to improve its product. The fact that security holes are common does not indicate that they are inherent, but merely that system implementors in general have paid less attention to them than they could.

3. *It has always been easy to create Trojan horses, worms, and viruses for all modern operating systems, because all modern software systems make it easy for one program to call another.* This statement was part of a response from Microsoft's Security Response Center to a critical article by James Gleick on the "Love Bug" virus. It is, to say the least, misleading. It is true that the ability for one program (say a mail reader) to call another (say a Visual Basic interpreter) figures into many security breaches. But the real problem is that the actual functionality that's needed is that of calling another program with certain constraints

applied to its execution. Adding restrictions is often as much work as adding features, and does not result in snazzy new functionality, so implementors tend to get a bit lazy about security. Contrary to the Security Response Center's intended implication, there is no inherent reason that the ability of one program to call another must breach security.

4. *It's not the software vendors, but the virus writers who are at fault for the damage done by malicious documents.* The virus writers should incur penalties for the damage they do, but this does not diminish the responsibility of all those who could reasonably have prevented the damage. When a car runs a red light and collides with another, its driver is responsible for the damage done, but that is cold comfort to the victims if their air bags failed to deploy.

5. *Users are responsible for their own safety. In buying vendor X's software, you should be aware of the damage that it can be made to do.* It is certainly true that any operator of a machine is to some degree responsible for that machine's effects. However, most users are not equipped to distinguish, for example, safe executable attachments from unsafe ones. Even an expert would find such a task tedious and error prone.

6. *Our software warns users when they are about to perform a potentially unsafe operation, such as clicking on an attachment.* Such warnings, when they are given at all, typically are not terribly discriminating. A safe executable attachment is treated the same as a malicious one. This is no service to a user who lacks the expertise (or the leisure) to make an informed judgment. As I have argued, we can make our software much more finely discriminating, so that often even such warnings could safely be avoided.

7. *There is a trade-off between ease of use and security; we are doing the best we can to find the best compromise.* I regularly use security software that gives me authenticated, encrypted access to any of a number of remote systems—essentially as if I were sitting at their consoles. I establish one of these connections by clicking on the appropriate system from a menu, period. I could arrange that a single command would arrange for secure access to a new system. You may use your browser to buy things on the Web. Such connections are generally secure, but you

generally wouldn't know that unless the browser or the Web page made a point of telling you, usually for the sake of customer relations. These examples, I claim, are typical of the "inconvenience" that well-designed security software needs to involve, almost regardless of functionality.

6 Summary

Computer security problems are generally the result of bugs—errors in operating systems and application programs. They are within the power of software vendors to solve. Successful data-damaging attacks are not inevitable; the world's computer community need not be the permanent victim of malicious teenagers. To accomplish this increase in security, we must abandon the disease metaphor, which sees constant vulnerability as normal, and stop relying on virus-detection programs as a primary line of defense.

I have argued that the computer user community can reasonably insist on applications that provide security with essentially no decrease in their current functionality and convenience of use. Furthermore, they can reasonably insist on "low-maintenance" security that does not require their constant vigilance (and expenditure) to keep their system informed of the latest batch of disease agents. We must not forget, however, that from a vendor's point of view, it is difficult to sell software that is invisible and trouble-free, but provides no additional capabilities. Therefore we users do share a collective responsibility to sell ourselves on the idea, and then, using both the market and the law, to push vendors to provide it.

Acknowledgements. My thanks to L. Peter Deutsch, Robert B. K. Dewar, Michael Harrison, Josh MacDonald, Doug Tygar, and David Wagner for many very helpful comments, discussions, and disagreements about this paper. The opinions and errors in it are mine.