CHAPTER 11

Future Work: an HCI-SEC Research Agenda

It is widely acknowledged that one of the most important research areas for computer security today is the development of techniques that will make security systems easier to use—and correspondingly make easy-to-use systems more secure. For example, the February 2005 report of the President's Information Technology Advisory Committee stated that work on "holistic system security" including "interfaces that promote security and user awareness of its importance" should be one of the President's top 10 "priority areas for increased emphasis" in computer security research.[Pre05]

This chapter maps out an agenda for work in the field of HCI-SEC, with goals for both the short and long terms.

11.1 Short Term

Zurko and Simon famously argued that many usability problems in today's security systems can be addressed through the use of user-centered design techniques such as task analysis, user interviews, usability testing, and iterative design, a process that they termed "user-centered security" design techniques. [ZS96] Adams and Sasse have made have made similar claims. [AS99]

Unfortunately, most discussions of how to move forward on user-centered security rarely move beyond the problem of authenticating computer users. Substantial work has been done on a broad range of authentication technologies including passwords, tokens, PKI, and biometrics. But while authentication is certainly both a challenging and important problem, it's important that other problems not be ignored—especially when there are other HCI-SEC areas where significant progress can be readily made. This is low-hanging fruit that the HCI-SEC community should be aggressively harvesting.

11.1.1 Aggressively promote a culture of usability among developers

One of the reasons there is so much low-hanging fruit available is that user-centered design strategies are rarely employed in the design of today's security systems. This is not a reflection on the nature of security systems, but on the nature of programmers in general.

Most programmers do not create software that is very usable. That's because most programmers create software for themselves, and programmers approach computers very differently than mainstream computer users. This is not meant to be a rebuke of programmers, but a statement of fact: a programmer who uses his or her own application program will invariably use that program differently than everyone else, because that person has a unique relationship with their creation.

Thus, it may be possible to make great strides in HCI-SEC simply by promoting the values of usability within academia and the commercial computer industry much in the way that other qualities such as efficiency, modularity and correctness have been traditionally promoted.

11.1.2 Design for security goals, not tasks

Today most security-related interfaces provide low-level control over specific functions that the operating system or application program might accomplish; they do not provide controls for higher level desires of the computer's user.

To use the example from Chapter 4, both Internet Explorer and Mozilla Firefox have individual controls for clearing the browser's page history, it's cookie repository, and its page cache. If a user wishes to erase evidence of browsing history, it is necessary to clear all three of these databases—and then figure out some way to sanitize the deleted disk files! This is an attention to security *tasks*, not *goals*.

Cooper argues that programmers inherently employ this "Task-Directed Design" because that is the way that software is created. [Coo99, p.151]. This is especially true of security software features in programs of the Whitten/Tygar "Abstraction property" [WT99]: security properties are abstract and hard to understand, and as a result it is frequently easier for programmers to provide tools for controlling specific security tasks, rather than helping users to achieve broader security goals.

This thesis has shown that tasks can be brought into alignment with goals simply be re-evaluating the underlying behaviors and assumptions that today's systems are based upon—for example, having the Windows `FORMAT.EXE` command automatically overwrite all of the blocks on a hard drive would have prevented many of the data leakage problems observed in the "Remembrance of Data Passed" study. Many of the design patterns presented in the previous chapter were derived by trying to understand what those goals are, and then creating patterns to accomplish them.

11.1.3 Provide information in context

It's well known that humans find it easier to understand and act upon information when it is shown in context. Frequently the "context" is provided simply by interpreting information for the user's particular situation.

For example, in his user trial of a home-banking system, Nielsen discovered that error rates for fund transfers significantly decreased when the confirmation dialogue replaced the current date

with the word "today" if the transfer happened to be scheduled for the day the request was being made.[Nie93a, p.38]

In Nielsen's case, the "context" used to display the information was the same day on that the program was running. By presenting the user's day of transfer in context, it eliminated a step that the user otherwise had to go through—*i.e.,* asking oneself, "Now let's see, is that today's date?"

Presenting information in context does not require solving the AI problem: today's computers have a tremendous amount of information that they can use to provide historical context to the user for his or her current tasks.

For example, it's frequently the case that our computer systems know significantly more information about the task at hand than they display. Today's mail programs visually distinguish between mail that is new and mail that has been seen, but they don't distinguish between senders that are in the user's address book and senders that aren't. Likewise, they don't distinguish between senders that are using their normal SMTP server and those that are using unusual SMTP servers. Mail readers that made such distinctions might have provided users with more defenses against "phishing" than the mail readers that people use today.

During the course of the *Johnny 2* experiment, an effective attack was discovered on users of Microsoft's Outlook and Outlook Express products. If Alice and Bob are both account managers at IBM, and Eve is an attacker, Eve craft an email message such that Bob thinks that his reply goes to Alice, when in fact it goes to Eve. An example of such a message appears in Figure 11-1.

This attack works because Outlook and Outlook Express do not display RFC822 mailbox names (*i.e.,* email addresses) when they are presented with full names. In this case the user is tricked because the "full name" of the `Reply-To:` field is in fact an email address. To make matters worse, the `Reply-To:` field is not displayed, as shown in Figure 11-2. Apple Mail prior to version 10.3.9 was also susceptible to this attack, although version 10.3.9 and above displays both the full name and the email address, as shown in Figure 11-3. (Another way to circumvent this attack is through the use of persistent digital signatures, such that Bob realizes that the email from Alice is not digitally signed with her customary key.)

What's needed, then, is for interfaces to make judicious use of the information that the computer has. They (or their programmers) need to be able to determine when information should be displayed, when it should not, and they need to provide easy-to-use controls that let the user find out more.

11.1.4 Using time as a proxy for trust; incorporate practical limits

The sensible use of elapsed time may be one of the areas in which great strides in usability can be made. This is because different security policies may be appropriate after short delays than after long ones.

An example of an application that interprets short time delays differently from long ones is the Windows XP screen saver system. By default, the XP screen saver displays when the keyboard and mouse have been idle for 5 minutes; when the mouse or keyboard are used again, the user's session is automatically suspended using the Windows "Switch User" facility and the user must

```
To: bob@ibm.com <bob@ibm.com>
From: alice@ibm.com <alice@ibm.com>
Subject: Need the annual report draft
Date: Tue, 19 Apr 2005 10:52:40 -0400
Reply-to: alice@ibm.com <attacker_eve@hotmail.com>


Bob, can you please send me a draft of the annual report? Thanks.

-Alice
```

Figure 11-1: An email message that, when delivered to `bob@ibm.com`, will appear in Microsoft Outlook Express has having come from `alice@ibm.com`. A reply to this message will actually be delivered to `attacker_eve@hotmail.com`.

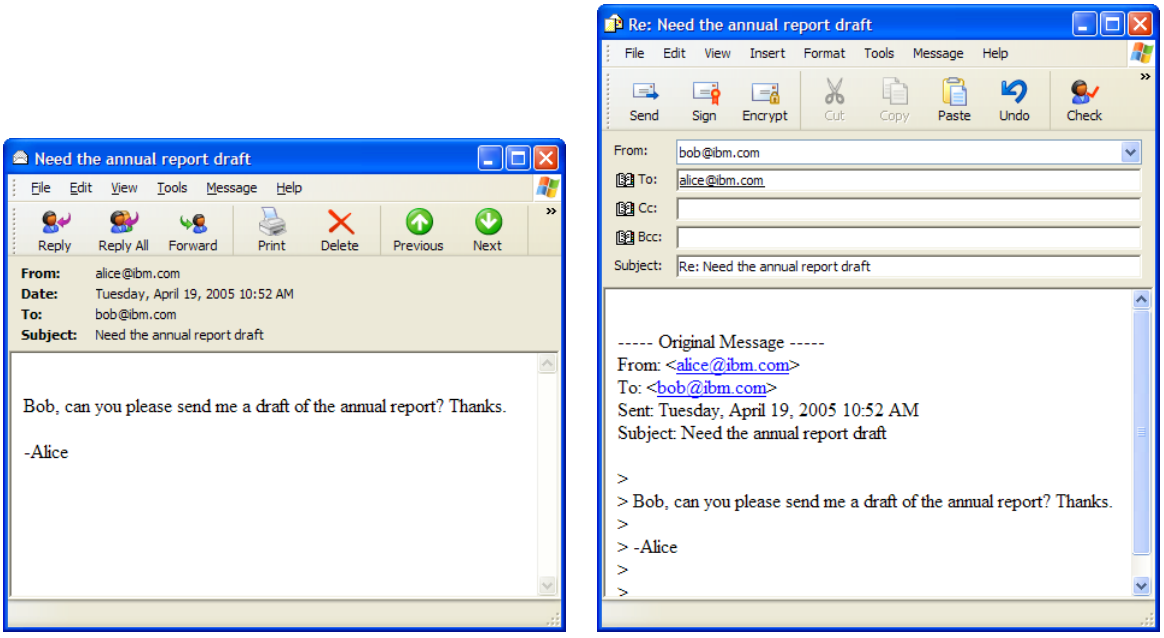

Figure 11-2: The message constructed in Figure 11-1, shown in Outlook Express (left), and the "reply" message that Outlook Express generates (right). Even though the email address `alice@ibm.com` is shown, the reply will go to `attacker_eve@hotmail.com`.

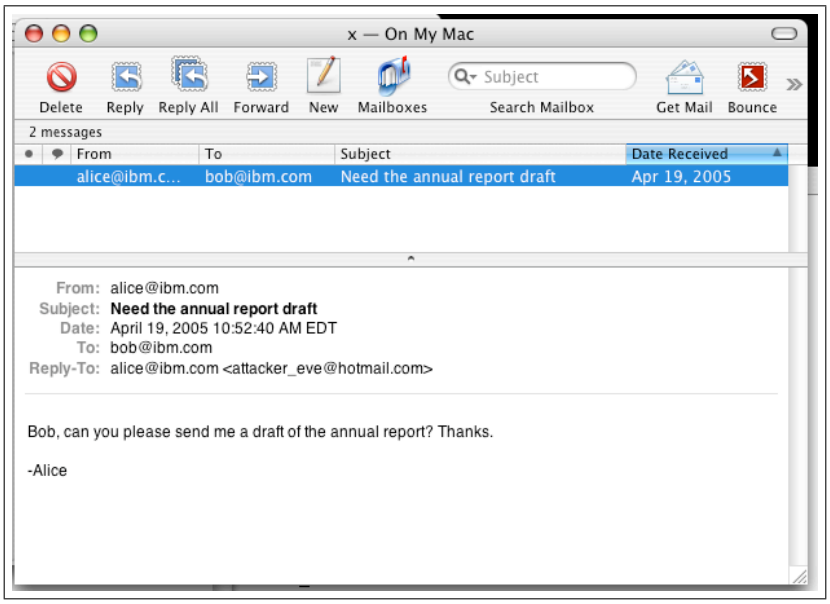

Figure 11-3: The same message constructed in Figure 11-1, this time displayed in Apple Mail version 10.3.9. Notice that both the "full name" *and* the RFC822 mailbox names are displayed, making spoofing considerably harder.

log in again. It would seem at first blush that this functionality forces users to choose between convenience and security: if the idle timeout period is set too low, the user will be inadvertently logged out whenever they take too much time to read a web page. But if the delay is set too high, then there will be a substantial window of vulnerability between the time that a user walks away from their computer and the time when the screen saver automatically locks. Indeed, the penalty for having the screen saver turn on may be perceived as being so high that users may disable the automatic logout feature entirely so that they do not need to be constantly typing a password.

This conundrum was certainly a problem with the Windows 2000 screen saver. However, the Windows XP screen saver has a grace period between the time that the screen saver appears and the time that the user is forced to enter a password to log back in to their computer. If you are sitting at a Windows XP computer and notice that the screen saver has appeared, you can quickly tap the mouse or the keyboard and have the screen saver disengage without the need to type a password. Only after the screen saver has been engaged for more than a five seconds does the automatic logout take place. By eliminating the need to type a password in this common case—a case for which typing the password would not increase security—Windows XP makes it more likely that users will not disable the requirement to type a password in all cases. (Figure 11-5)

(As it turns out, the screen saver grace period can be modified with the Windows XP program "Tweak UI" from the Microsoft PowerToys web page. [Cor05d] However, Microsoft correctly chose *not* to expose this functionality to the average user, a good application of STANDARDIZED SECURITY POLICIES pattern.)

There are few other examples of using timeouts to gracefully migrate to higher security configurations. One current example is the "remember me" box on the Google GMail service. Whereas

many web applications have a "remember me" check-box that allows the computer to remember the user's name or even the name and password forever, the Google's GMail interface only "remembers" the user for two weeks (Figure 11-4). Two weeks is long enough so that the user won't be annoyed by continually re-entering the password, but short enough so that the user is unlikely to forget the password from disuse. (Renaud reports that 30 days is considered a threshold period after which non-meaningful items cannot be reasonably remembered unless there is some kind of memory "hook."[Ren05])

There are many other places in modern operating systems where sensible limitations can be built using elapsed time as a proxy for trust:

- A laptop could require that a password be provided when it wakes up after being asleep for longer than 5 minutes and when it is then reboot, but not otherwise. This would allow the user to close the laptop, carry it to another room and open it again without having to type a password.
- A PDA could require a password for accessing items in the calendar that are more than a few days in the past, but not otherwise. This would eliminate the hassle of using a password with the device most of the time, but would still prevent others who have temporary access to the device from snooping into the owner's past.
- A cell phone could only require a password when the total amount of money spent in one day exceeds a preset threshold—for example, a dollar.

In each of these cases, by eliminating the need of passwords for casual, frequent use, the cost to convenience of using the password is greatly reduced. As a result, the password can still provide protection against significant attacks—a stolen laptop, a snoop who goes through one's older calendar entries, or an individual who attempts to place many expensive international phone calls.

Figure 11-4: Google's GMail allows the password to be remembered for two weeks.

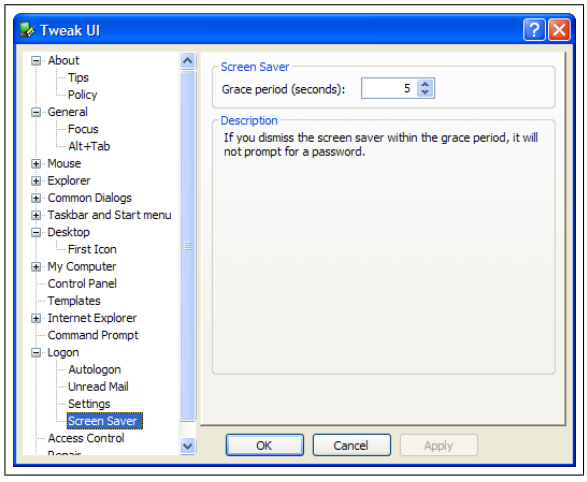

Figure 11-5: Microsoft's Windows XP "Power Toys" package allows the Windows XP screen saver grace period to be set by advanced users. This functionality is only exposed to users who download the add-on package from Microsoft's web site.

11.1.5 User perception surveys

The survey of Amazon.com merchants presented in this thesis just scratches the surface of the useful information that can be obtained using survey techniques. To date the results of user surveys have not been widely reported in the computer security literature. One reason may be the lack of training: Although survey work is not intrinsically difficult, neither the design of survey instruments nor the analysis of survey data is part of most computer science curriculums.

Instead of teaching computer science students to become survey practitioners, this could be viewed as an opportunity for cross-disciplinary work with between computer scientists and social scientists. As more bridges are built between these two communities, it is clear that there are many questions of security perception that could be profitably addressed through the use of survey instruments.

Some questions to investigate include:

- **SSL Lock.** What do typical web users think the "lock" icon means? Do they understand that it means that the web page was delivered using encryption, or do they think that it means that the web site will protect personal information once it is received? Can users distinguish between a "lock" icon that appears in the web browser's status bar and one that appears in the body of a web page that was delivered without the use of encryption?
- **Invalid Certificates.** The E-Soft survey found that roughly half of the SSL-enabled web site on the Internet have invalid certificates—for example, certificates that are expired, certificates that have name/site mismatch, or that are signed by an unknown CA. What do users think when they are alerted to these error conditions by their web browsers? If these messages are universally ignored, why have them at all?
- **How secure is email?** Weisband and Reinig wrote in 1995 "Employees must learn that passwords do not prevent others from accessing computer accounts and that backing up electronic files is standard practice. Employees should also understand the legal implications of email privacy so they are not surprised when messages they send or receive are used to document some undesirable behavior. Interface design issues could address this by reminding users that deleted messages are not sent to a trash can but to a filing cabinet." [WR95] Well, do employees know this? How about other computer users? How did they learn this information? How does the knowledge of users match and mesh existing practices and capabilities?

11.1.6 Throttles and governors

A *throttle* is a device that controls the rate at which fuel is delivered to an engine. It has an indirect control on the engine's speed. A *governor* is a device that steadily generates increased resistance as the speed of an engine increases; a governor directly controls an engine's top speed.

Although throttles and engines were developed for use in steam and gasoline engines, there is both an opportunity and a need for analogs of these devices in modern computer systems. Such constructs can limit the damage that can be done by malicious code or even user error. Properly designed and deployed, these benefits can be obtained without significant impact on usability.

Throttles and governors are different from user quotas and process limits, such as those found in the Unix operating system. While quotas and process limits restrict the *total amount* of disk space,

memory or CPU time that a single user or process can consume, throttles and governors control the *rate* at which a system resource can be consumed.

For example, a typical desktop computer is capable of initiating dozens to hundreds of outgoing network connections every second. Under most circumstances, this capacity is simply not needed. A typical web browser initiates a maximum of 4 connections per second, although frequently it initiates far fewer. In contrast, the Code Red worm initiated 200 connections per second, while the SQL Slammer worm could generate up to 30,000 packets per second using a very tight loop.

HP Labs has studied the this approach in detail and found that limiting user programs to one or two out-bound connections per second has no noticeable impact on users, other than dropping some ads from web pages that are filled with ads from third-party sites.[Wil03, TW03]

Microsoft rate-limiting technology was introduced in Windows XP SP2. Most users did not notice the technology, as the limits were set fairly high. But the technology did affect some programs that were initiating a high number of connections each second due to implementation errors—that is, the programs were buggy. But as the bugs had not previously been exposed, the user experience was that SP2 broke their software. One such program was eMule, a peer-to-peer file trading program. A user who was very disturbed by the throttling technology posted a binary patch to turn it off in an eMule support forum, concluding: "YOU ARE NOW FREE FROM MICROSOFT RULE AND OPPRESSION!"[Mxx04]

The posting demonstrates a point that Microsoft employees have been making for years: in fixing security holes or adding new pro-security technology, it is possible that existing programs may be damaged. for proper operation, and fixing these holes breaks those programs. On the other hand, the folks at eMule seem to have recovered.

There are other opportunities for throttles and governors, including:

- *File open rate or count.* Most programs rarely need to open new files. By limiting all but specially designated programs to a low file-open rate, or by triggering a warning if more than a certain number of discrete files are opened, programs that scavenge the hard drive for files that have particular contents could be identified.
- *File delete rate.* Likewise, programs rarely need to delete many files at once. Trapping and requiring additional confirmation for high file delete rates would catch not only some classes of hostile code but also occasional user mistakes (although a better approach would be to rework the computer's file system along the lines discussed in Chapter 3.)

11.2 Long Term

This section explores HCI-SEC issues that will probably take somewhat longer to address than those in the previous section.

11.2.1 Extended Key Continuity Management (EKCM)

A variant of Key Continuity Management that is not explored in this dissertation is to apply KCM to certifying keys—that is, keys belonging to Certification Authorities—rather than to keys themselves.

In the case of self-signed keys, EKCM degenerates to standard KCM.

Implementation of EKCM should be straightforward: When an unknown root key is seen for the second time, the user should have the option of giving it a unique name and accepting it into a database of EKCM certifiers. Identities that are certified by this root will be displayed as `name@ROOT` where `name` is the name on the certificate and `ROOT` is the user-provided name for the root. This is similar to the structure provided by Lotus Notes, except it would be performed automatically.

EKCM could further be designed to that the EKCM certifiers would only work for identities in the domain for which they were first encountered. That is, if a certifying key was found for the `@amazon.com` domain, it would not automatically be trusted for identities found in the `@ex.com` domain.

The advantage of EKCM is that it gracefully handles the case in which an organization has created its own certification authority for internal use. For example, if one routinely communicates with three employees from Amazon, and if Amazon has adopted a CA and issued all of its employees certificates, then one will only be warned about the new certificate once.

EKCM would allow users to slowly build trust in unknown CAs by watching the use of their signatures, rather than relying on out-of-channel means. It would make use of the independent PKIs that are being deployed by many businesses and universities without the need to formerly cross-certify or be certified by third-parties such as the VeriSign Certificate Interoperability Service.[Ver05a]

EKCM looks like a promising technique and it seems to build naturally on both the work of Lotus and the KCM work presented in Chapter 7. But EKCM needs to be validated with software and user studies to see if the model is superior to both KCM and today's CA model.

11.2.2 Risk communication

Although there has been significant research devoted to the subject of risk communication over the past two decades, including an excellent volume by the National Research Council in 1989[Nat89], there is little if any published research on the subject of risk communication applied to computer security risks.

Bostrom and Löfstedt suggest that although practical approaches to risk communication programs vary, the best follow a simple set of steps: "Know your audience—do formative research. Know the risk—know what it is and what can be done about it. Test your messages empirically. Iterate."[BL03, p.245] Although iterative user interface development is now accepted as the bedrock of usability development (see page 43 of this dissertation), there has been little attention to applying these techniques to the communication of computer risks. This represents an opportunity for future research.

Research has shown that the manner in which risks are communicated makes a significant difference to the way that both the general public and professionals judge risks. We may think that we base our opinions on a cold evaluation of the numbers, but even professionals frequently base their judgments on affect, feelings, and the way that statistics are framed. [SFPM04] In one study, Slovic found that 21% of clinicians would not discharge a patient from a hospital if they were told that "patients similar to Mr. Jones are estimated to have a 20% chance of committing an act of vio-

lence." But when the statistics were phrased "20 out of every 100 patients similar to Mr. Jones are estimated to commit an act of violence," the number of clinicians who refused to discharge *the same patient* rose to 41%! [SMM00] Slovic *et al.* report on a similar study by Yamagishi [Yam97] that found people judged a disease that "kills 1,286 people out of every 10,000" to be "more dangerous than one that kills 24.14% of the population."[SFPM04]

> "Follow-up studies showed that representations of risk in the form of individual probabilities of 10% or 20% led to relatively benign images of one person, unlikely to harm anyone, whereas the "equivalent" frequentistic representations created frightening images of violent patents (e.g., "some guy going crazy and killing someone"). These affect-laden images likely induced greater perceptions of risk in response to the relative-frequency frames." [SFPM04]

Risk communication is incredibly important for the future of computer security. Today's computer users continually make security-related decisions without much thought about their impact on security. Frequently it is because users simply do not understand the consequences and implications of their decisions—something that Cooper calls "uninformed consent." [Coo99, p.140] Deleting cookies is an example of uninformed consent: there's no practical way for today's users to tell the difference between a third-party tracking cookie and a cookie that's used to give access to a previously purchased electronic document.

In a thought-provoking master's thesis *Avoiding the Cyber Pandemic: A Public Health Approach to Preventing Malware Propagation*,[Zel04] Zelonis argues that the computer security community could come up with novel approaches for fighting self-reproducing computer programs through a careful consideration of the successes that some public health programs have had in fighting the AIDS virus. Although the terms like "worm" and "virus" are commonly used as linguistic analogies to describe malware, Zelonis argues that the analogies go much deeper:

> "Although monogamy decreases the risk of HIV/AIDS, a monogamous person whose partner is promiscuous is put at greater risk by connecting them indirectly with numerous partners. This is true, too, of malware, but the risk is difficult to avoid. The Internet is an inherently promiscuous partner. By placing a computer on-line, other computers can infect it even if no user action is taken to connect with those other machines."[Zel04, p.49]

Continuing the analogy, Zelonis argues that malware and HIV/AIDS have similar epidemic enablers, infection conduits, prophylactic measures, and even survivability characteristics. For example, there is a foolproof technique for avoiding infection through either AIDS or malware: abstinence. In the case of HIV/AIDS, this requires abstaining from sex, intravenous drugs, and blood transfusions. In the case of malware, one must abstain from networking and using third-party software. But neither prevention strategy seems particularly appealing. Zelonis' Figure 3 (reprinted in this thesis as Figure 11-6) lists other analogous attributes between HIV/AIDS and malware.

By examining the analogy in detail, Zelonis suggests a variety of mitigation strategies that have been successful in fighting HIV/AIDS which have not been applied to the problem of malware. These include:

Figure 3

Summary of Analogous Attributes

	HIV/AIDS	Malware
Geographic Scope	World Wide	World Wide
Epidemic Enabler	Sexual Revolution	Commercialization of the Internet
Infection Conduit	Bodily Fluids	Computer Code
Spread	Contact	Connection
Accelerant	Frequency of Contact	Frequency of Connection
Impact of behaviors	Certain behaviors increase risk	Certain behaviors increase risk
Basic Risk Behavior	Having sex (vaginal, anal, oral)	Connecting a computer to a network, particularly the Internet
Prophylactic Measures	Condom Topical Micobicide	Virus scanner Personal firewall Software patches Secure configurations
Promiscuity	Having many sexual partners (simultaneously or via serial monogamy)	Having an "always on" high speed Internet connection, opening unexpected email attachments, browsing unfamiliar web sites, using numerous protocols and software
Indirect Promiscuity	Sex with a promiscuous partner	The Internet is inherently promiscuous
Extreme Promiscuity	Having multiple or anonymous sexual partners	Peer-to-Peer (P2P) filesharing
Selection of partners	Almost exclusively by choice	A mixture of choice, randomness, and discrimination
Survivability	Slow destruction of host	Slow or minimal destruction of host
Network	Generally scale-free	Generally scale-free
Genealogy	Virus strains traceable	Code snippets traceable

Figure 11-6: Analogous Attributes between HIV/AIDS and Malware, from [Zel04, p.41]

- **Comprehensive information distribution.** In June 1988, a publication called "Understand AIDS" was sent to *all* US households. "It became the most widely read publication in the US at that time."
- **Infuse with popular culture.** Zelonis notes that there is no malware equivalent to the AIDS "red ribbon."
- **Testing for infection status.** HIV transmission goes down when people who are infected learn of their status; telling computer users the infection status of their computers should have a similar result. ISPs are in an ideal position to tell their customers if and when they become infected.
- **Use of real-life examples and true stories**. "Rather than teaching about malware risk through general descriptions and statistics, telling stories to which the audience can relate will allow them to better understand the potential impact of malware in their lives. This is similar to the idea of focusing on personal/group risk. An individual may not be able to relate

to news stories about major ecommerce sites being brought down by an attack, but he may respond to hearing how his neighbor's bank records were compromised."

- **Use of games and entertainment.** Noting on the success of a radio soap opera in Tanzania to teach the science and risks associated with HIV/AIDS, Zelonis suggests that there is an opportunity to use popular entertainment to convey important messages about cybersecurity. "Computers are already a popular platform for game-playing. Malware awareness could be incorporated into educational video games, particularly when targeting younger demographics. Computer security issues are periodically showing up in the plotlines of television programs and movies, but an educational value seems incidental. It would be interesting to see the impact of an entire program ... designed specifically to convey computer security messages... The key, as noted by Singhal and Rogers, is to use a high quality creative team so that the resultant feel is that of entertainment rather than education, while still delivering the necessary message.[SR03]"

Trying to solve the plague of malware through social, public health means rather than technical means is a novel approach which might yield surprising results.

11.2.3 Standardize the Vocabulary of Computer Security

Although standardization is clearly not leading-edge computer science research, this thesis has argued that standardizing the vocabulary of security terms and the placement of security controls could have significant benefits in terms of secure operation. Since Microsoft, Apple, and the Open Source community are unlikely to be able to settle their squabbles without outside mediation, there is an opportunity here for some neutral body to establish basic standards and then for the federal government to adopt those standards in its procurement regulations. A standardized vocabulary and the use of design patterns discussed in this thesis would be an excellent place to start.

Alas, right now there is a lack of organizations where interface standards can even be *discussed*. As others have noted, [HB05] the Internet Engineering Task Force has done little work in the field of user interfaces. (RFC768 used the term "user interface" to describe a programmer's API and not a lot has changed in the intervening 25 years.[Pos80a])

The World Wide Web Consortium (W3C) might be an appropriate forum. For example, the W3C's so-called "Interaction Domain"[Hos04] specifically addresses issue of web browsers interfaces, although to date the group has been concerned with presentation technologies such as CSS, MathML, and SMIL, rather than with the actual interface of the web browser. Clearly, for W3C to expand its mission to encompass user interfaces for security would be a considerable and almost certainly controversial undertaking.

11.2.4 Rethinking patch management

Patch management is a long-term issue because—in the short term, at least—the need to patch operating systems isn't going to go away. What's more, it seems likely that the industry will adopt standards regarding which systems are automatically patched and which are not. Specifically, end-user and client systems that are in more-or-less standardized configurations will automatically receive and install patches, while servers and other systems that are running custom configurations will not automatically be patched because of the higher cost of downtime on these systems.

In the coming years it will become increasingly difficult for end users to make reasonable choices regarding the administration of their computers. How could a 14-year-old decide whether or not to download the patch for her six-month-old cellphone? How could a 68-year-old retiree? Indeed, how could a 28-year-old with a Ph.D. in computer security make an informed decision without possession of proprietary information belonging to the cell phone manufacturer?

Not surprisingly, many security practitioners have argued that automated installation of patches is necessary to remove the "human factor" from today's practice of "patch-and-pray." But as Vicente rightfully points out, automation of this sort doesn't eliminate the human factor: it only concentrates the potential human factor impact among a small number of administrators.[Vic01] For example, in April 2005 the anti-virus company Trend Micro, Inc., released an update to its anti-virus system. The update was automatically downloaded and installed on many computers, including those belonging to East Japan Railway Co and several prominent media organizations in Japan. Unfortunately, the update wasn't properly tested, and those computers all became unusable after they were reboot with the update installed.[CK05, The05b] Humans run automated systems, and such systems frequently magnifies the impact of human error.

Patching is simply not an acceptable solution for the long term. One reason is that increasingly there will be systems deployed that simply cannot be patched: the systems may not have access to the Internet, or that may run applications that will break if the underlying operating system is upgraded, or the systems may simply not have sufficient memory to handle the upgrade. Already, Microsoft is not offering security updates for Windows 95 or Windows 98. Although it is tempting to refuse updates to these systems with the hope that they will become infected, die, and have to be replaced, this does not seem like a socially responsible approach for long-term security.

In the long term, we will need new approach. Perhaps this approach will be computers that are delivered with a set of applications that can never be upgraded or updated—hopefully protecting the computer against malware in the process.

Another possibility is the so-called "computer immune system."[KSS+97, SHF97] Substantial work is currently being done in this area. (*e.g.,* [DG02, Som02, FSA97].) See [dCZ00] and [For05] for a survey of current work.

11.2.5 Backup, rollback, restoration and recovery

With the exception of unauthorized disclosure and its consequences, most security woes can be undone with good systems in place for backup, restoration and recovery. Today's computer systems have ample storage capacities. Yet storage invariably goes unused while important information is insufficiently backed up.

Consider these examples:

- A person who types 100 words per minute, 8 hours each day, types roughly a quarter of a megabyte every day, or 90 megabytes a year. Even one of today's low-end computers could trivially devote one or two gigabytes of hard disk to capture every stroke and mouse click. Although such a simple-minded approach would no doubt have privacy problems, it would provide a new kind of backup with strengths and weaknesses very different from current

approaches.

- Likewise, a computer system with even a moderately sized hard drive could store dozens—or even hundreds—of versions of each document that the user edits. If storage becomes tight, old versions could be thrown away stochastically, rather than on a first-in, first-out basis. Pieces of such a system may now be appearing. For example, Spinellis has created an open source tool called `fileprune` that performs such housekeeping [Spi03]; Strunk *et al.* have presented S4 which uses a log-structured file system, journals, and an audit log to detect tampering and prevent data loss from accidental deletes. [SGS+00] Nevertheless, such technology is rarely present on systems belonging to the users who could benefit the most from massive backup: consumer PCs that are not centrally managed.
- Rekimoto has proposed an approach for backups called "Time Machine Computing:" [Rek99b]

> "Imagine that your computer has a dial for time-traveling. With such a computer, when you create a document you can simply leave it on the desktop. You can also remove documents at any time. If you later need to refer the previously created information, you can time-travel to the day when that document was on the desktop. You might also see other related information that were simultaneously placed on the computer screen, and these items would help you to recall the activity context at that time."[Rek99a]

These are all innovative approaches to the backup problem, but these ideas that are not making it into mainstream operating systems. Indeed, many organizations are moving in the other direction—trying to come up with new clever techniques for throwing away information so that it cannot be used in court battles.

Much of the research in Chapters 3 and 4 of this thesis were based on the premise that systems should provide facilities for COMPLETE DELETE. But while those chapters were being written, the author suffered from two hard drive crashes and several system wipeouts. Without exceedingly good backups and data replication, much of the work in this thesis would have been lost. Thus, there is a need not just for COMPLETE DELETE and for good backup systems—there is also a need for all of these systems to interoperate together.

One of the most interesting recovery systems built into Windows XP is its ability to take snapshots of configurations and revert if problems arise. More work needs to be done in this area—work not just on operating systems, but on application data as well. And we need to come up with both technical and legal frameworks so that people are not afraid to use the technology once it is created.

11.2.6 Logfiles with "undo"

One approach for extending recovery would be an approach called "logfiles with undo." This approach could be implemented with a new logging subsystem that records not just actions that happened on the computer, but also the necessary steps that would be required to undo the action. Ideally these log entries would be stored with some kind of dependency information, allowing them to be executed out-of-order. But Logfiles with undo cannot be implemented with today's logfiles as they simply do not record enough information.

Indeed, the whole question of what information belongs in logfiles needs to be seriously considered in any HCI-SEC agenda. Today's logfiles are not merely not designed for usability: they are often not even designed for *use*. Instead, they are typically created as a debugging tool by the original application programmers. One of the few exceptions to this general state of logfile malaise is change log that Lotus Notes maintains for Access Control Lists (Figure 11-8). Exposing logs directly within application interfaces, as Lotus has done, may push programmers to put more meaningful information into the logs and to add "undo" capabilities.

Finally, an operating system undo framework should provide "undo/undo-undo" support through the use of transactions (similar to the GNU Emacs undo facility), rather than the more simplistic "undo/redo" rollback facility built into Microsoft Word and the Apple Cocoa application framework. A common problem with the Microsoft/Apple approach is that some actions are undone, a character is typed, and then all of the "redo" information is lost. The Emacs approach avoids this usability problem, since the "undo" actions are themselves stored as transactions that can be "undone."

11.2.7 Virtualization

A growing number of researchers and practitioners think that the way to solve the malware problem is through the use of virtualization or other forms of system partitioning. This is a line of thinking that goes back to the IBM virtual machine operating systems of the 1960s, [Cre81] and has recently gained interest thanks to the success of the Virtual PC and VMWare [Wal02] virtual environments.

Today a common suggestion is that the computer system of the future will use virtualization to create multiple protection zones. One might be a "green" or safe zone for important operations such as banking and commerce, while there might be a "yellow" or cautious zone for operations like email, and a "red" or "crash and burn" zone for surfing the Internet and playing downloaded games.

One natural problem with this approach is the tendency for code to escalate into the regions of higher privilege. For example, screen savers should probably run in the "crash and burn" zone: after all, hostile code posing as screen savers are a common problem on the Internet today. [Ile04] Placing screensavers in the crash and burn zone prevents them from doing damage to other programs or data on the PC. On the other hand, a very popular Microsoft screensaver is the one that goes through all of the files on the computer, finds the photographs, and displays them in semi-random order. This screensaver, then, would have to be run in the "green" zone. Likewise, a web banking application would almost certainly want to communicate with a web browser running in the "green" zone—but if the "green" zone can run a web browser, then it will almost certainly be a target for phishing attacks.

In this context, Microsoft's Next-Generation Secure Computing Base (NGSCB, [Mic05] previously known as "Palladium") can be thought of as a kind of lightweight virtualization system that uses cryptography to provide isolation between applications.

Other potential problems with the isolation approach is whether or not the separations between protection domains can be presented to users in a manner such that the isolation capability and rules are understandable. Alternatively, is it possible to make the isolation automatic and invisible? These are long-term problems requiring considerable research, and unlikely to be resolved through

the application of a near-term fix.

11.2.8 Adaptive computing

There are many different kinds of people in the world, and increasingly they all use computer systems. Some people speak English; many don't. Some people have sight, others don't. Some people can read, while others cannot. And of course, this range in physical and mental abilities is matched by an equally broad wide range of computer skills.

Traditionally computer professionals have used the shorthand of "novice," "intermediate," and "expert" or "power user" to describe different skill levels among computer users. Some programs mimic these categories, with "novice menus" and "full menus."

Computer skills are not measured on a single scale, either. Some people are very good with application programs, but they have little or no knowledge of operating systems. Some people have knowledge of one part of an application program but not of another part. Some people are wed to a particular operating system and will use no other.

But as software becomes more complex and the security threat becomes even more pressing, software will need to become increasingly adaptive to user needs. The system will need to do this automatically—realizing when users need more help and offering it, while realizing when users understand the implications of what they are doing and getting out of their way.

Unfortunately, the most widely deployed example of adaptive interface technology on the desktop computer today is an utter failure—a failure realized by practically everyone other than the technology's promoter. Microsoft introduced a kind of adaptive menu in its Windows 2000 operating system; the technology was also incorporated in the company's Office application suite.[Mic03b] The Microsoft technology hides menu commands that are not generally used; once the command is used, however, the menu reconfigures itself for a time during with the command is shown. After a period of disuse the menu command hides itself again.

The problem with Microsoft's approach is that it does precisely the wrong thing: commands that are used infrequently are precisely the commands of which users need reminding! What Microsoft should have done was come up with a better way to structure its menu system, not remove the seldom-used commands from its application menus.

Raskin reports that users who experienced the Windows 2000 interface quickly found it disagreeable. "A typical remark was, 'Adaptive menus seemed like a cool idea, but the first time a menu changed on me, I found it upsetting. I don't like the idea any more.' " [Ras00]

Recently a study by Findlater and McGrener of static, adaptable, and adaptive menus found that adaptive menus were slower for users to use and produced more frustration. Despite the fact that a majority of the users in the study preferred the concept of adaptable menus, these menus did not confer a performance advantage. [FM04]

Can systems automatically detect the user's skill level and adapt? Can they provide help when needed but get out of the way at other times? Would it be better to have a single interface that everybody uses? Modern automobiles have certainly done well with standardized user interfaces.

Standardized interfaces can be made more usable through the use of "agents" or "co-pilots" that look over our shoulder and offer help when necessary. (Certainly the readiness of Microsoft's "Clippy" office assistant to offer help was a failure that ultimately resulted in Clippy's banishment [Mic01]; but as Xiao *et al.* show, properly constructed agents can both increase the accuracy and enjoyment of computer users.[XSC04])

At the same time, consistency is not just at odds with adaptability; it's at odds with innovation. Witness the differences between Windows 95, Windows 98, Windows 2000, and Windows XP. On the surface level, each of these interfaces appears to be consistent, but there are deep changes between each of these operating systems in the placement of control panels, the layout of "properties" and "advanced" tabs, the underlying security models, and the like. A large number of the changes between these systems is the result of attempts to make each successive version of windows more secure. Few computer scientists would want to live in a world where interfaces did not change over time.

It's possible that the rate of change in interface design is going to slow down. The layout of controls in Windows XP is far more similar to Windows 2000 than 2000 was to Windows 95. That's a 10-year span. But the previous 10 years took the industry from DOS to Windows 3.1 to Windows 95—changes that are radical and extreme by any measure. Although systems are becoming smarter, the rate at which genuinely new interfaces are deployed seems to have decreased significantly. Even handheld devices today look more like Windows (or MacOS) running on a desktop than they look like, say, the interface of the Xerox Star or the Symbolics Lisp Machine.

11.3 A Call for New Patterns

Finally, there are many areas where usability and security can be simultaneously improved through the development and adoption of more HCI-SEC patterns. This section proposes several proto-patterns for which the need seems evident, but which have not yet been refined.

Programmers spend considerable time writing code that displays information to the user. But sometimes it seems that this code is designed for the *programmer's* benefit and sensibilities, and not for the end user. Consider the display of certificates in today's operating systems: these displays look as if they were designed for debugging the certificate management code! Similar problems can be found in most logfiles.

The good news, then, is that significant usability strides can be made through the use of simple tools for displaying or visualizing operating system data.

One of the reason that tools are so important is that some of the most basic concepts in the programmers' toolbox are quite alien to non-programmers. Consider containment. Cooper notes that many users have difficulty relying on the hierarchical file system used by Unix and Windows both to locate data and to enforce permissions.[Coo99, p.52]. The reason appears that the concepts of containment and recursion are simply not part of most people's day-to-day experience. Indeed, Good and Krekelberg observed that users of the Kazaa P2P file-sharing system were generally unaware that sharing a folder shared *all* of the sub-folders that the folder contained. Users were also unaware that sharing a folder shared *all* of the files inside a directory, rather than simply the music

files. One user in their study even exclaimed, "You mean it shares *all* files?"[GK03]

It's an untested belief that better displays can convey these concepts to the majority of computer users. Other alternatives are mandatory training (such as Whitten's *safe staging*), and simply removing the concepts from the underlying system. Of these three alternatives, better displays seems the most likely to succeed.

11.3.1 Display numeric information meaningfully

Many programs display binary blobs as a long string of hex digits separated by colons; adding spaces would make these strings much easier to understand. Numbers should be displayed in a manner that is optimized for their *human use*, rather than for the programmer's convenience.

Numbers that are displayed in hex by convention—for example, certificate fingerprints and Ethernet MAC addresses—should group numbers in sets of four digits separated by a space or colon to improve readable.

Consider these two alternative displays of a 128-bit binary value:

(1) `8ca3 b82d b8e2 720d ba64 aec9 f775 5964`
(2) `8c:a3:b8:2d:b8:e2:72:0d:ba:64:ae:c9:f7:75:59:64`

It is likely that humans can parse string (1) both faster and with with a lower error rate.[Nor05] The error rate may be reduced further still by displaying shorter groupings in Base64. Yet string (2) is the approach that is commonly used by today's operating systems and applications.

In many cases small hexadecimal numbers can be more meaningfully displayed in decimal than in hex. For example, instead of displaying a certificate number as `0d 04 d8`, display it as `853,208`. (This only works when certificates are assigned sequentially, as is the case with those issued by Thawte; see Chapter 6. It is an open question if Thawte would have continued to issue certificates with sequential numbers if the company's senior management had been able to easily convert the certificate serial numbers from hex to decimal.)

Where numbers have units, those units should be displayed if they are not obvious. For example, use "Certificate Lifetime: 365 days" rather than "Certificate Lifetime: 365."

This pattern was widely adopted prior to the coming of e-commerce. For example, credit card numbers are displayed with spaces between groups of digits to ease in their reading and transcription. Unfortunately, many web sites that require users to *type* credit card numbers do not allow the numbers to be entered with spaces. Instead, web site advise users to type credit card numbers with "no spaces or dashes." This is a profound barrier to usability.

11.3.2 Analyze user's "effective" access

Access control lists are hard to understand and made more complex by rules governing inheritance and special cases. Since the computer system ultimately makes the decision whether or not user U has access to object O, it makes sense for the system to make this decision making process visible. Lotus Notes provides a system for allowing administrators to easily calculate the "effective access"

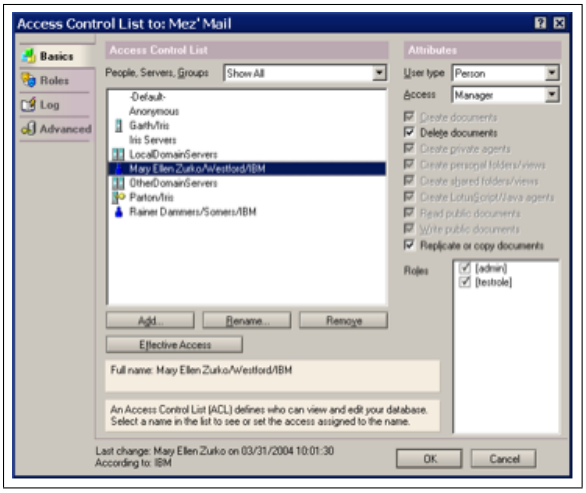

Figure 11-7: Lotus Notes provides administrators a tool for a window that allows an administrators to determine a user's effective access on a particular object. Used with permission.

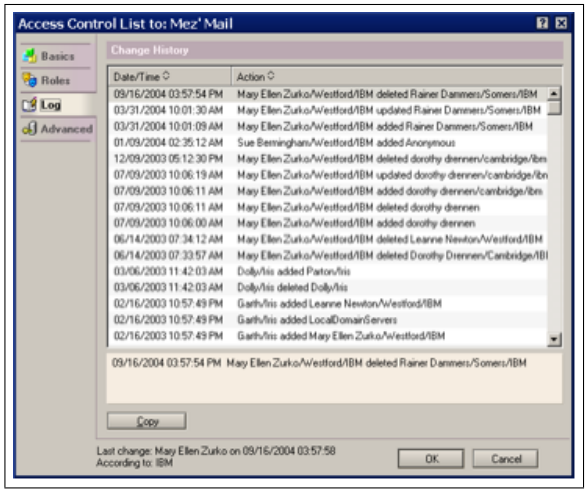

Figure 11-8: Lotus Notes logs each change to an object's Access Control List in a manner that is relatively easy to audit. This interface could be improved with a "search" facility and an "undo" button which allows individual changes to be undone.

of a user on a particular object, as shown in Figure 11-7; it is likely that this is a general pattern that could be expanded.

11.3.3 Distinguish "tainted" content

The Perl programming language[WCO00] has a concept of "tainted" data. The purpose of tainting is to wall off any data that is provided from an untrusted source.[1] When invoked with the -T option or when the `use taint` command is executed, Perl won't allow a tainted string to be used for a filename or in a string that's passed to a shell. Tainting protects against a variety of data-driven attacks — for example, when an attacker enters a string inside backquotes on a CGI-based web form, potentially causing a subshell to execute the contents of the string before passing it on the command line to another program.

This concept of tainting can and should be extended to user interfaces, so that potentially tainted strings are distinguished from pure strings that are generated from within an operating system or application program. Such an approach would limit a large number of spoofing attacks that currently plague computer users.

Today's computers frequently display data provided by outside data sources using the same typography and ornamentation as information that is generated by the computer's own operating system and applications. Neumann has noted that puns, while they may sound pleasant, can carry significant risks to both safety and security.[Neu90]

[1] Perl's tainting rules are relatively straightforward: any string that comes from an untrusted source (e.g. data that is provided by the user, an environment variable, read from a file, read from a network connection, or so on) is tainted. If any input to a string operator or function is tainted, then the result is tainted. The only way to untaint a string is through regular expression matching.

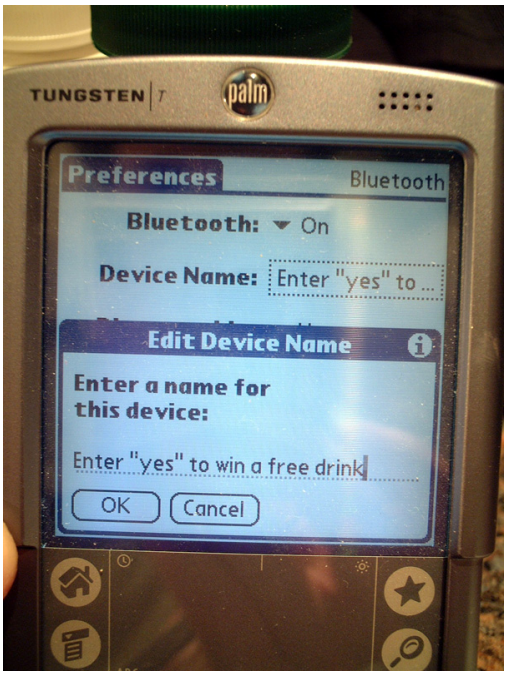

Figure 11-9: To configure a computer for bluejacking, simply give the device a name that might be mislead as an instruction. In this case, the attacker is using a Palm Tungsten T

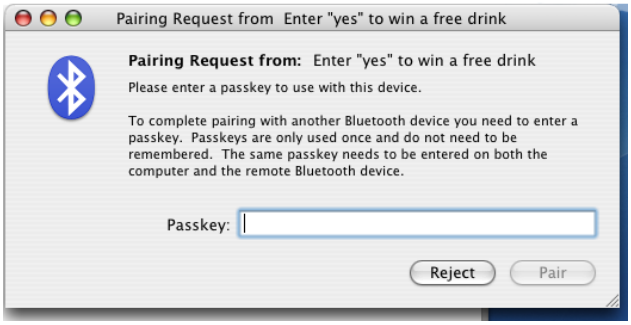

Figure 11-10: The pairing request from the attacker's Bluetooth device is confusing to the victim because the name of the attacker's device appears to be an instruction from a trustable party.

There are many opportunities for this sort of pattern on today's computers. For example, hostnames, email addresses and other names that are created by third parties could all be displayed with specific typography that sets them off from other operating system controls. Context-sensitive help could give a full explanation of the risk to users.

Example: Internet Advertising
One of the reasons that tainting information from third parties is so hard is that some mechanism must be developed for tainting images. Web sites have the ability to display arbitrary images on the user's computer. Attackers have used this ability to display simulacra web browsers for spoofing attacks. Figure 11-11 demonstrates such an attack in an Internet advertisement. Ye and Smith discuss this vulnerability in detail and propose a solution in which a web browser's outline constantly change in a manner that an attacker cannot predict.[YS02] Unfortunately, this solution is distracting.

Example: Bluejacking
One newly emerging attack that tainting in the interface would limit is *bluejacking*. There are two versions of this attack that have been documented. In the first (mostly harmless) version, an attacker creates an anonymous VCARD in which all of the fields are empty except for the first name, which might contain a little message like "You are Being Watched!" Because the transmission of a VCARD is considered a reasonably benign operation, the Bluetooth protocol and security model

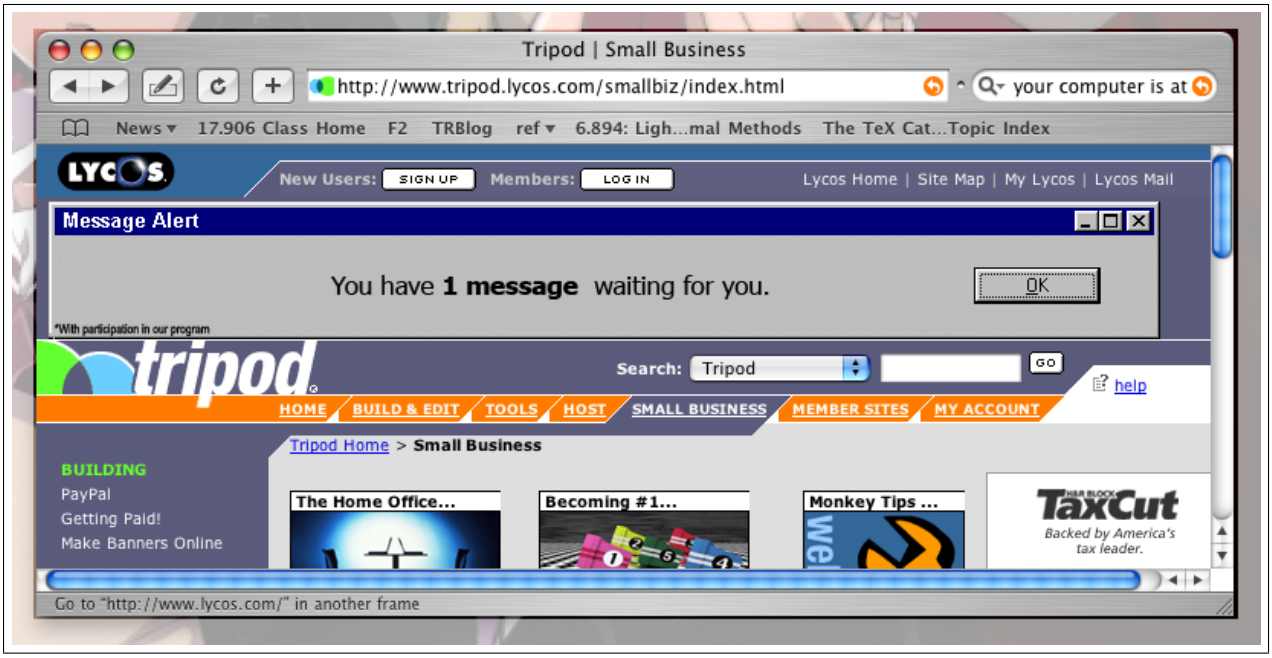

Figure 11-11: Advertisements on web sites have the ability to spoof operating system controls. For example, this advertisement on Tripod appears to be a Windows pop-up window with an important message. The fact that this pop-up is being displayed on a Macintosh computer gives away the spoof, but on Windows machines these spoofs can be quite realistic.

allows VCARDS to be sent anonymously and without authentication. The recipient of such a card sees the message on their screen, but doesn't know where it comes from. This attack was first seen in December 2002.[McF03]

A more aggressive form of the attack occurs when the user of the Bluetooth devices is tricked into "pairing" their device with an attacker's Bluetooth device. Bluetooth pairing gives paired devices unrestricted access to each other's trusted databases.[Geh02] Because Bluetooth devices may be assigned long human-readable names, and because these names are displayed during a pairing request without tainting, attackers can unwittingly persuade users to pair their devices with the attacking device by using long device names that can be misleadingly interpreted.

To understand how such a pairing attack could work, imagine that a computer is given the Bluetooth name "Enter 'Yes' To Win A Free Drink," as shown in Figure 11-9. When an attacker then attempts to use this device to pair with a victim's machine, the victim sees the message that "Enter 'Yes' to Win A Free Drink" wishes to pair: do you wish to pair? (Figure 11-10) The attacker enters "Yes" as the Bluetooth passkey. (The bluetooth Passkey is a one-time authenticator that is entered on each device being paired to set up the initial Bluetooth pairing. Many devices have pre-assigned passkeys.) It is believed that many untrained individuals in a bar or coffee shop would answer this question "yes" and inadvertently pair with the attacker.

Although these kinds of semantic attacks are exceedingly difficult to eliminate with clever programming, a good starting approach would be to distinguish text that is generated by applications and

the operation system with text that is directly provided by untrusted entities.

Proto-pattern: Distinguish Tainted Content
This pattern is designed to help users resist a wide range of spoofing attacks which are based on data supplied by an attacker being mistaken for data supplied by the system.

11.4 In Conclusion

The security and usability *professions* have long been at odds. Nevertheless, today's computer users must be able to achieve both security and usability *in practice*, otherwise they will increasingly have neither.

This thesis has shown more than 20 specific techniques that can be used to simultaneously increase usability and security. Many more techniques undoubtedly exist. What is needed now is the will of industry to move these techniques from the research laboratory to the products that are used by the world's computer users, so that everyone may experience a computing future that is simultaneously more secure and yet more usable.