

---

## CHAPTER 2

---

# Prior Work

Many people believe that software developers have a hard time creating systems that are both usable and that provide strong security. Many researchers in the newly emergent field of usability and security (HCI-SEC) claim that security issues have been largely ignored by usability researchers, and that usability issues have been largely ignored by security professionals.

Section 2.1 shows that HCI-SEC issues, while underplayed, have not been *ignored* entirely for the past thirty years. Section 2.2 examines specific rules, techniques and principles that have emerged in the field of HCI-SEC. Section 2.3 discusses properties, models, and principles that have been developed for software that aligns security and usability. Section 2.4 discusses some of the specific techniques that have been identified for creating systems that are both secure and usable. Section 2.5 discusses prior work on the problem of media sanitization. Section 5.2.1 discusses prior surveys on security attitudes and email usage. Section 2.6 discusses prior work on regulatory and other non-technical approaches for aligning security and usability. Throughout this entire discussion, the reader will note that what has been lacking has been a systematic approach for taking research on security and usability and moving it into the marketplace. We argue that such movement has been hampered by the lack of patterns that are specifically designed to aid this transition.

### 2.1 Early Work in HCI-SEC

Whitten and Tygar observe in their 1999 paper “we have found very little published research to date on the problem of usability for security.”[WT99, p.183] In her PhD thesis, Whitten observes that the computer industry has had substantial success designing web browsers, email clients, and instant messaging systems that are used daily by “people who have very little technical background. But when similar user interfaces have been created for security, they have had little or no success.” [Whi04a, p.1]

This analysis is largely correct. A review of the history of both the security and usability literature reveals that while many security researchers have long considered usability issues, and usability

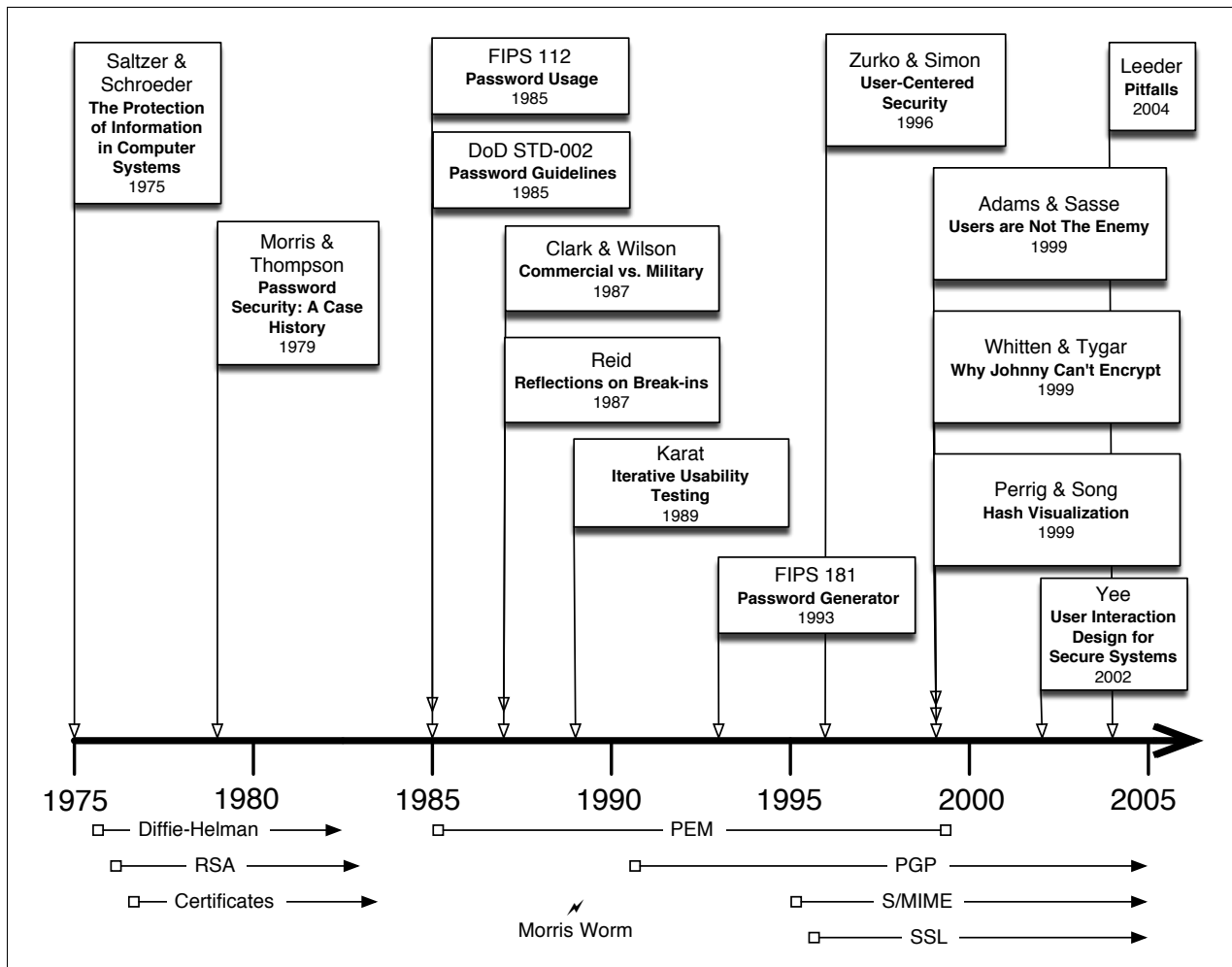


Figure 2-1: A timeline showing some of the significant HCI-SEC literature from the security field.

researchers have long considered security issues, the topic has only rarely received significant attention as a subject of primary study. Further hampering such research has been the fact that HCI-SEC was not recognized as an independent field. Formal usability tools have rarely been used to analyze security software in the open literature.

For example, Adams and Sasse correctly note that “[T]o date, research on password security has focused on designing technical mechanisms to protect access to systems; the usability of those mechanisms has rarely been investigated.”[AS99] But even though it is true that there were few published user studies that had *investigated* the usability of password systems, the usability of such systems had long been *considered*.

For example, the 1985 US Department of Defense *Password Management Guideline* noted that security was improved by adopting “user-friendly passwords that were easier to remember.”[DoD85, p.15]

#### A.4 “User-Friendly” Passwords

To assist users in remembering their passwords, the password generation algorithm should generate passwords or passphrases that are “easy” to remember. Passwords formed by randomly choosing characters are generally difficult to remember. Passwords that are pronounceable are often easy to remember, as are passphrases that are formed by concatenating real words into a phrase or sentence.[DoD85, p.15]

Certainly, work on HCI-SEC has been dwarfed by work in practically every other field of computer science, computer security, or usability, but to say that there was no work between the time that Saltzer and Schroeder identified the principle of “psychological acceptability” in 1975 and the resurgence of interest in security and usability at the end of the 1990s is a vast exaggeration.

##### 2.1.1 Early recognition of the HCI-SEC problem

As noted above, Saltzer and Schroeder identified the need to consider usability as a primary factor in developing secure systems in their landmark 1975 paper.[SS75] That paper identified eight design principles for building systems that can protect information: Economy of mechanism; fail-safe defaults; complete mediation; open design; separation of privilege; least privilege; least common mechanism; and psychological acceptability. For the last principle, the authors wrote:

**h) Psychological acceptability:** It is essential that the human interface be designed for ease of use, so that users routinely and automatically apply the protection mechanisms correctly. Also, to the extent that the user’s mental image of his protection goals matches the mechanisms he must use, mistakes will be minimized. If he must translate his image of his protection needs into a radically different specification language, he will make errors.[SS75]

In their seminal article on password security, Morris and Thompson wrote that the underlying goal of passwords is to provide “security at minimal inconvenience to the users of the system.” The authors conducted a study and found that 2,831 out of 3,289 examined passwords were easy to find through a dictionary attack. The authors notes that “users could be urged (or forced) to use either longer passwords or passwords chosen from a larger character set, or the system could itself choose passwords for the users.”[MT79]

Perhaps the most important lesson of the Morris and Thompson article is contained in the last paragraph of the paper’s introduction:

“Although the security of a password encryption algorithm is an interesting intellectual and mathematical problem, it is only one tiny facet of a very large problem. In practice, physical security of the computer, communications security of the communications link, and physical control of the computer itself loom as far more important issues. Perhaps most important of all is control over the action of ex-employees, since they are not under any direct control and they may have intimate knowledge about the system, its resources, and its methods of access. Good system security involves realistic evaluation of the risks not only of deliberate attacks but also of casual authorized access and accidental disclosure.”[MT79, p.594]

That is, Morris and Thompson acknowledge that the most intellectually interesting problems to solve in the area of computer security are not necessarily the questions that are the most relevant to overall system security! (Of course, the authors then proceed to attack the tiny facet that they find intellectually interesting.)

Reid considered the issue of security and usability in 1987 and concluded that “programmer convenience is the antithesis of security, because it is going to become intruder convenience if the programmer’s account is ever compromised.”[Rei87] Implicitly assuming that usability and security are antagonistic, Reid argued that Unix should be made less usable and more secure:

“UNIX was created as a laboratory research vehicle, not as a commercial operating system. As it has become more widely used commercially, many of the properties that made it attractive in the laboratory have created problems. For example, the permission file mechanism described above lets me easily give my colleagues full access to the files on my computer. When UNIX systems are installed in non-laboratory applications by people who are not trained to think about operating system security, however, the same mechanism that is convenient in the laboratory becomes dangerous in the field. There is no way to assign fault or blame for these security problems, because if the UNIX system is used as its designers intended, security is not a problem.”[Rei87, p.104]

Wood *et al.* relate a story in which a file containing two years worth of research data was inadvertently deleted because of a usability problem resulting from wildcard expansion.<sup>1</sup> Apparently the research organization had neglected to back up this critical file onto another media. Attempts to recover the file with an “unerase” command failed because the deleted file was so big that it had been fragmented into many different locations on the disk. Fortunately, two local data recovery experts working for 70 hours were able to recover the file’s fragments and restore most of the information. The authors conclude that systems must include provisions for making backups of critical files and that potentially dangerous commands should have protections.[WBG<sup>+</sup>87, pp.123–124]

Gong, Lomas, Needham and Saltzer discuss how poorly chosen passwords can be made resistant to “guessing attacks” through the use of advanced cryptographic protocols. Although this perhaps seems obvious now, it was not at the time. For example, as the authors note, Project Athena’s Kerberos system does not have such protections: any client on the network can request a Kerberos ticket and then mount a guessing attack against this ticket until the correct password is found. After discussing this flaw, the authors go on to present a series of protocols that do not have this flaw and are thus resistant against such attacks.[GLNS93] This paper is important because it shows how an apparently secure protocol—in this case, Kerberos—might need to be hardened to protect against the way that real systems are used by real people.

This sampling of security literature is not meant to imply that usability has been an ever-present theme in the history of computer security research: clearly it has not. But it is also incorrect to argue that the issues of usability have been generally ignored by security researchers.

---

<sup>1</sup>In the case described, the computer operator had told the computer to erase the file specification “FAULTS\*.DBS” with the intent of erasing the files “FAULTS1.DBS” and “FAULTS2.DBS”, without realizing that the file “FAULTS.DBS” would also be erased by the wildcard. On this system there was no confirmation for deletions and nor provisions for easily undoing file system modifications.

### 2.1.2 Work on HCI-SEC from usability researchers

Just as security researchers have long been aware of usability issues, usability researchers have long drawn on examples from the world of computer security in their writings and research.

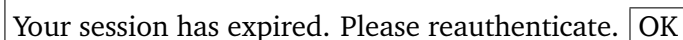
For example, Norman's 1983 article discussing design rules to accommodate user error (see section 2.2 below) specifically addresses the question of file deletion and the tension between the desire to actually erase information to prevent malicious recovery and the techniques that allow a user to recover a file in the event of accidentally deletion.[Nor83, p.258] (This topic will be discussed at length in Section 3.6.1.)

In *Usability Engineering*, Nielsen notes in a somewhat resigned tone that security realities frequently require that systems be made less helpful than they might otherwise be. His example is password authentication: it is widely accepted that systems requesting a username and a password should give users the same feedback whether the username is valid or not. Otherwise, an attacker could probe the system to determine a list of valid usernames, and then target those usernames for a password-guessing attack.[Nie93b, p.42]

Nielsen also addresses the issue of file deletion in his book, arguing that operating systems should not use icons such as a paper shredder to represent file deletion because these icons imply that the file contents are actually destroyed:

“Users with sensitive data on their disks can therefore not rely on file deletion to safeguard their data in cases where others have access to the disk—for example, because it is sold or sent in for repair. The paper-shredder icon may give users a false sense of security due to the connotations of physical paper shredders with respect to the destruction of confidential paper documents. In contrast, the trash-can icon at least implicitly suggests that others might look through the discarded documents.”[Nie93b, p.128]

Johnson holds up security-related user interfaces as objects of scorn and ridicule in his collection of the 82 common usability failings that are common in programs with graphical user interfaces.[Joh00] For example, Johnson describes an application that he oversaw in which the user was presented with a dialogue that appeared if the session was idle for too long:



Your session has expired. Please reauthenticate.

Johnson commented:

“When users acknowledge the message by clicking OK, they would be returned to the ‘User Authentication’ page. I advised the developers to get rid of the word ‘user,’ change all uses of the term ‘authenticate’ to ‘login,’ and increase the automatic timeout of the servers (since they didn’t want to eliminate it altogether).” [Joh00, p.206]

But despite this obvious attention to security-related issues both in his book and in his consulting practice (from which many examples in the book were drawn), the word “security” does not even appear in the book’s index.

Johnson and other usability specialists have long delighted in making fun of the poor interfaces surrounding the security-relevant parts of systems. The fact that such interfaces are jeered at, rather than simply ignored, shows that the specialists were frequently thinking about HCI-SEC.

### 2.1.3 HCI-SEC emerges as a distinct field

While HCI-SEC is not a *new* field, there is certainly truth to the notion that HCI-SEC has only recently emerged as an independent discipline. One reason is likely the late emergence of usability as an academic discipline itself.

Although humans have interacted with computers since the first machines were created, it was only in the 1980s that the field of Computer Human Interaction emerged as one that was distinct from other fields of computer science research. The Association for Computing Machinery's Special Interest Group on Computer Human Interaction (SIGCHI) traces its history to the ACM's Special Interest Group on Social and Behavioral Computing (SIGSOC).

SIGSOC started in 1969, when ACM members who were using computers to further professional interests in the social and behavioral sciences decided to start their own special interest group. The first panel presentation on the computer-human interface was probably at the December 1978 ACM Conference in Washington DC entitled "People-oriented Systems: When and How?" The following year *Communications of the ACM* appointed an Editor for Human Aspects of Computing. In February 1982 Allen Newell was an invited speaker at the Computer Science Conference with the topic: "Human Interaction with Computers: The Requirements for Progress." [Bor96, p.4]

In 1982 "a conference on human factors in computer systems was planned and conducted by volunteers" in Gaithersburg, MD, without support of the parent organization. [Bor96] That year SIGSOC changed its name to SIGCHI. The first SIGCHI Conference on Human Factors in Computing Systems, CHI'83, took place in December 1983, with SIG GRAPHICS providing assistance.

Nevertheless, researcher interest in formally studying the interaction of security and usability was slow to mature. It wasn't until 1989 that Karat presented her paper "Iterative Usability Testing of a Security Application" at the 33rd Annual Meeting of the Human Factors Society. [Kar89] Ten years passed before Whitten conducted her *Johnny* experiment in 1998. [WT98] The following year Adams and Sasse published their study of password behavior, *Users Are Not The Enemy*. [AS99]

Whitten created the "hcisec" mailing list of Yahoo! Groups in May 2000. [Whi00] Andrew Patrick, A. Chris Long, and Scott Flinn organized a "Workshop on Human-Computer Interaction and Security Systems" at CHI2003. [PLF03]

In 2004 *IEEE Security and Privacy Magazine* published a special issue on the topic of Security and Usability, with contributions by 13 researchers in the field. [BDSG04, YBAG04, Jus04, PKW04, Yee04]. Later this year, O'Reilly will publish a volume of edited papers entitled *Usability and Security*, with contributions from more than 50 researchers in the field. [CG05] So while it is true that usability issues have long been important to security researchers, and *vice versa*, it is also true that the field of HCI-SEC is now well on its way to being a recognized specialty all its own.

## 2.2 Rules and Principles for Designing Usable Systems

There are of course no set of rules, principles or formalisms that, when followed, are guaranteed to produce usable computer systems. If such rules existed, we would almost certainly all be using them, and the usability problem would be solved.

The most common methodology for building usable systems appears to be a combination of task analysis followed by an iterative process involving interface redesign and user testing. It is commonly accepted that paper prototypes should be employed early in the process because they are easier to change than code; as a result, test subjects confronted with these prototypes are more likely to suggest big changes that could represent usability breakthroughs. This is the methodology described at length by Karat, Brodie and Karat, who argue that the iterative approach dramatically cut the post-release technical support costs for the application, resulting in a return-on-investment of at least 10:1.[KBK05]

But as Cooper points out, simply “iterating until something works” can be wasteful without understanding the flaws in the current system and having some idea of where you want to be going.[Coo99, p.50] Ideally this kind of navigation is informed through a *user-centered design process*, which evaluates software from the user’s point of view. Zurko and Simon introduced the phrase “user-centered security” to describe this process applied to security problems.[ZS96]

### 2.2.1 Norman’s design rules and error analysis

Norman observed in 1983 that many users new to a computer system will make the same common errors. “Experienced users ... often smiled tolerantly as new users repeated well-known errors.” [Nor83]

Arguing that errors were probably the result of design flaws, rather than poor training or user ineptitude, Norman classified errors as being either *mistakes* or *slips*. A mistake occurred when a user’s intended action (the *intention*) was itself in error. A slip, on the other hand, occurred when the user’s intention was correct but an error was made in the intention’s execution.

Because mistakes are frequently the result of poor training, Norman’s analysis concentrated on slips. He classified slips into three categories, each of which he divided into further subcategories. He argued that many slips with computers arise from either the existence of modes or the inability of people to correct their errors—that is, actions that cannot be undone.

“People will make errors, so make the system insensitive to them,” wrote Norman. What’s needed, he argued, is software “safeties” that make irreversible actions difficult, and improved undo systems so that fewer actions are in fact irreversible.

In applying Norman’s work to the subject of this thesis, one of Norman’s most important observations is that so-called *activation errors* can be overcome through the use of *memory aids*. As Norman defined the term, an activation error is an inappropriate action being performed or an inappropriate action being activated. Memory aids in the form of on-screen notices, status indicators, or pop-up warnings can overcome activation errors by activating the correct response. “In many ways the old saying, out of sight, out of mind, is apt,” writes Norman, who argues that “a good system design” will give the user visual reminders of actions that need the user’s attention (e.g., partially

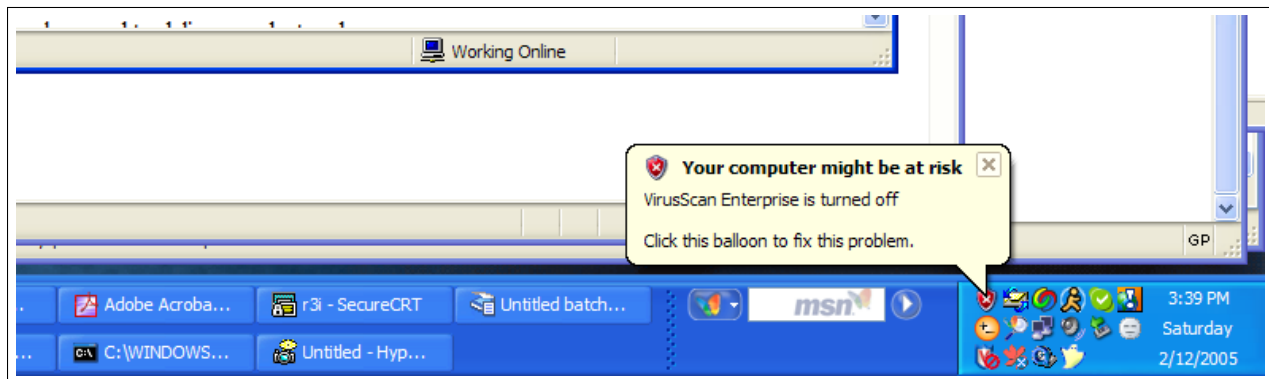


Figure 2-2: Microsoft Windows XP SP2 warns the user if their antivirus system has been disabled, an example of a memory aid.

completed tasks that need to be finished.)

An example of such a memory aid is the task bar status indicator in Windows XP SP2, indicating that the computer's antivirus system has been disabled and needs to be re-enabled (Figure 2-2). In this case the computer's antivirus system was disabled in order to work with a disk image that contained several viruses. A few hours later, with the task completed and the antivirus still disabled, Windows XP displayed a pop-up message to warn of the potential risk.

### 2.2.2 Nielsen's heuristics for usability engineering

Nielsen has developed a technique he calls "Discount Usability Engineering" that he argues can dramatically improve the return on investment when applied to product development.[Nie89, Nie90, Nie94] The approach includes a set of "Usability Heuristics" that he employs for evaluating the usability of interfaces:

- **Simple and natural dialogue:** Dialogues should not contain information that is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility. All information should appear in a natural and logical order.
- **Speak the users' language:** The dialogue should be expressed clearly in words, phrases, and concepts familiar to the user, rather than in system-oriented terms.
- **Minimize the user's memory load:** The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.
- **Consistency:** Users should not have to wonder whether different words, situations, or actions mean the same thing.
- **Feedback:** The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.
- **Clearly marked exits:** Users often choose system functions by mistake and will need a clearly



marked “emergency exit” to leave the unwanted state without having to go through an extended dialogue.

- **Shortcuts:** Accelerators—unseen by the novice user—may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users.
- **Good error messages:** They should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.
- **Prevent errors:** Even better than good error messages is a careful design that prevents a problem from occurring in the first place.
- **Help and documentation:** Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, be focused on the user’s task, list concrete steps to be carried out, and not be too large.

Nielsen’s recommendations on “language” and “consistency” are especially critical in HCI-SEC, where it is common for developers to use terminology that is complex and inconsistent. This is discussed in Section 8.2. The security-relevance of his “feedback” recommendation was made clear during the *Johnny 2* study (see Chapter 7), in which many users wanted some kind of feedback from Outlook Express that email sent when the “encrypt” button was pressed would actually be encrypted.

### 2.2.3 The Apple Human Interface Guidelines

The *Apple Human Interface Guidelines* [App04a] and *Apple Software Design Guidelines* [App04d] stand alone in the computer industry as a single comprehensive document describing a wide variety of aspects of computer-human interaction in a desktop environment. In addition to detailed chapters on the working of user interface elements such as buttons, scrollers and windows, the *Guidelines* has chapters on human interface design principles and philosophy; the development process; and the importance of guiding the user’s attention through a complex system.

Apple’s guidelines are important because of their breadth of coverage, their quality, and because of Apple’s longstanding commitment to both usability and “friendliness” towards novice users.

Interviews conducted at Apple Computer on January 12, 2004 with Apple’s security group, the developers of its Mail application (which includes support for S/MIME), and its Vice President of Software Technology revealed that Apple places a priority on security that is either invisible or, at very least, exceedingly easy-to-use. The developers, for example, were particularly pleased with their work on Apple’s keychain and its authentication panels—two subsystems that are designed to provide protection against an array of automated attacks while simultaneously conveying benefits to the user. The developers explained how they had struggled hard with HCI-SEC issues, looking for ways to align the two apparently disparate fields.

Given such statements on the part of Apple employees, it is surprising that so little of the company’s user interface guidelines specifically address security issues. For example, the word “security” does not appear in the 255-page *Human Interface Guidelines* at all! The 81-page *Software Design Guidelines* has one page discussing security issues—mostly advice to factor out code that

requires privileges, to use the Apple Keychain Services to store passwords, and to use the Apple-provided authentication interfaces. In contrast, searching for the word “usability” in the *Human Interface Guidelines* brings the reader to three different sections on the importance of keyboard shortcuts, icon design, and menu design for improving usability. Searching the *Software Design Guidelines* brings the reader to a page on the importance of conducting user testing (with step-by-step instructions), a section explaining that “aesthetic integrity” can either enhance or detract from usability, and a plea to use the standard interface elements and only create new ones when necessary. “Usability testing is essential for determining whether a new element works.”[App04d, p.29]

Apple does spend some time discussing its security Keychain, a single encrypted storage area for passwords and other secrets:

“The keychain mechanism in MacOS X adds value because:

- It provides a secure, predictable, consistent experience for users to deal with passwords.
- Users can modify settings for all of the passwords as a group (the default behavior) or set up different keychains for different activities with unique activation settings.
- The Keychain Access application provides a simple user interface for users to manage their keychains and their settings, relieving you of this task.”[App04b, p.67]

A more detailed discussion of the Apple keychain can be found in *Enabling Secure Storage With Keychain Services*. [App04e]

Despite this lack of emphasis on HCI-SEC, the topic appears several times in Apple’s Human Interface Guidelines—even when the examples in the guidelines inadvertently contradict the philosophy of usability that Apple is trying to convey:

- The triangular disclosure button is used by Apple as an example of helping the user to manage complexity by hiding information. Interestingly, the example used in the discussion of the disclosure button is Apple’s **Authenticate** panel—a system that allows the user to perform tasks that require privilege without having the user “su” to root or having *setuid* programs.

The information hidden inside the disclosure triangle apparently has a lower standard of usability than information that is prominently displayed. In both [App04b, p.37] and [App04c, p.192], the information revealed by clicking on the disclosure panel, shown in Figure 2-3 is quite cryptic.

- In its description of Alert panels, Apple notes that “In dangerous situations, the default button may be Cancel but, it should not be the action button and it should not be located in the action button position.”[App04c, p.212]. Interestingly, the illustration that Apple uses to illustrate this point (Apple’s Figure 11-10) is a screen shot from the Safari Web Browser’s dialogue, “Are you sure you want to empty the cache storing the contents of web pages.” As will be discussed in Chapter 4, there are many HCI-SEC problems with this panel, including both its language and its failure to actually sanitize the web pages that are “emptied” from the cache.
- Apple recommends that when passwords are entered into a text field, “each typed character s should appear as a bullet, matching the number of characters typed by the user.”[App04b,

```
Requested right: system.preferences
```

```
Application: /Applications/System Preferences.app
```

Figure 2-3: The information revealed by the Apple Authenticate panel when the disclosure triangle is clicked. This means that the program “System Preferences.app” has requested the right to modify the database of system-wide preferences. The value here to a Macintosh security expert is that a trusted channel is indicating that an application installed in the `/Applications` directory is asking for more privileges—rather than some other application, such as one in the browser’s cache directory. Although this highly granular information is presented to the user, its interpretation is not discussed in Apple’s documentation.

p.97][App04c, p.41] Pressing the Delete key should delete a single bullet. Finally, “when the user leaves the text field . . . , the number of bullets in the text field should be modified so that the field does not reflect the actual number of characters in the password.”

Overall, it seems, the usability community has generally done a better job is establishing guidelines, methodologies, and procedures for achieving their goals than the security community has. Security practitioners and researchers are well advised to consider the body of usability work—not just to explore ways that the usability guidelines can be reworked to integrate more secure operations, but also to look for ways that the specific developer education techniques can be adopted to security.

#### 2.2.4 Federal Information Processing Standards

“Federal Information Processing Standards Publications (FIPS PUBS) are issued by the National Institute of Standards and Technology after approval by the Secretary of Commerce pursuant to Section 111(d) of the Federal Property and Administrative Services Act of 1949, as amended by the Computer Security Act of 1987, Public Law 100-235.”[NIS85]

Two FIPS directly relate to the interaction of usability and security. FIPS PUB 112, Password Usage Standard,[NIS85] specifies standards for password composition, length and lifetime. The passwords specified by this standard are not strong: the standard specifies that passwords shall have a minimum range of 4 characters and a minimum of 10 possible symbols per character, for a minimum number of  $10^4$  (10,000) possible passwords. The standard specifies that passwords should be changed once a year, passwords for “medium protection requirements” should be changed every six months, and passwords for “high protection” should be changed every month.

FIPS PUB 181, Automated Password Generator (APG),[NIS93] specifies an algorithm in C based on the Data Encryption Standard (DES), for creating pronounceable and therefore more easily remembered passwords. FIPS PUB 181 actually specifies the algorithm in C, but because of export restrictions in place at the time of the publication and the algorithm’s use of DES, the algorithm was not distributed in electronic form.

Porter has reimplemented the FIPS PUB 181 algorithm in the Perl programming language.[Por00b] Instead of using DES, Porter uses Perl’s built-in random number generator (Figure 2-4).

In general, although the FIPS have been very successful in standardizing encryption algorithms, it appears that they have not been successful in helping to secure the adoption of usable security technology.

```
% perl RandPasswd.pm --count 10< /dev/null
ghepevef (ghep-ev-ef)
anckye (anck-ye)
inkilj (ink-ilj)
jeerea (jeer-ea)
fijisung (fij-is-ung)
ciebyegu (cieb-yeg-u)
ojexuno (oj-ex-un-o)
hiedilva (hied-ilv-a)
garnufa (garn-uf-a)
rokukiks (rok-uk-iks)
%
```

Figure 2-4: Ten randomly generated passwords using a modified version of FIPS 181[NIS93] that appears in [Por00a].

“Definition: Security software is usable if the people who are expected to use it:

- are reliably made aware of the security tasks they need to perform;
- are able to figure out how to successfully perform those tasks;
- don’t make dangerous errors; and
- are sufficiently comfortable with the interface to continue using it.”

Figure 2-5: Whitten and Tygar’s definition of usable security software.[p.170][WT99]

## 2.3 Properties, Models and Principles for Usable Security

Is there something about security software that makes it fundamentally more difficult to make usable? This section reviews recent work in the emerging HCI-SEC community that seeks to see if there are particular properties, models or principles that can be used to understand the interaction of usability and security.

### 2.3.1 Whitten and Tygar’s properties of security software

Whitten and Tygar have argued that software with security-related features is somehow different from other kinds of software. They call such software *security software*.

Although the researchers do not define what usable software is, they do create a definition for usable security software. That definition appears in Figure 2-5.

Using this definition, the pair argue that inherent properties in security software make such software inherently difficult for user interface design. The names and definitions of these properties, first presented in [WT99], are renamed and subtly modified in [Whi04a] and are summarized in Figure 2-6.

Taken together, these properties can argue for either removing the user from security-critical decisions whenever possible, software modifications to increase the usability of this security software, or increased user training to make errors and mishaps less likely. After raising and then discarding

Property	Explanation
The secondary goal property (previously “The unmotivated user property”[WT99])	Security is (at best) a secondary goal of users. “People do not generally sit down at their computers wanting to manage their security; rather, they want to send email, browse web pages, or download software.” [Whi04a, p.7] “If security is too difficult or annoying, users may give up on it altogether.”[WT99, p.3]
The hidden failure property (previously “The lack of feedback property”[WT99])	It is difficult to provide good feedback for security management and configuration because configurations are complex and not easy to summarize.
The abstraction property	Security policies are usually phrased as abstract rules that are easily understood by programmers but “alien and unintuitive to many members of the wider user population.” <sup>a</sup> [WT99]
The barn door property	Once a secret gets out, it’s out. Information disclosure cannot be reversed. Even worse, there is no way to know if an unprotected secret has been compromised is being privately circulated by others. “Because of this, user interface design for security needs to place a very high priority on making sure users understand their security well enough to keep from making potentially high-cost mistakes.”[Whi04a, p.8]
The weakest link property	The security of a system is like a chain: it is only as strong as the weakest link. “If a cracker can exploit a single error, the game is up.”[WT99, Whi04a]

<sup>a</sup>Although Whitten and Tygar do not provide an example of the abstraction property, Straub and Baier argue that “Especially in the field of public-key cryptography and PKI, it is a difficult task to find intelligible yet precise (real world) metaphors for abstract mathematical objects like key pairs or certificates.[SB04]

Figure 2-6: Whitten and Tygar’s properties that make usability for security software fundamentally different than usability for non-security software.

the possibility of “making security invisible,” Whitten introduces her two techniques for “making security usable:” *Safe Staging* and *Metaphor Tailoring*.

Several critiques with this part of Whitten’s otherwise excellent contribution will be described in the remainder of this section.

### Critique #1: the term “security software”

The first problem is the use of the phrase “security software.” What is it? Much of Whitten’s research focuses on email encryption software such as PGP. Clearly PGP is “security software.” The *Johnny* study used PGP to manage cryptographic keys and perform cryptographic functions. But the *Johnny* study also used the popular Eudora program to actually send and receive email. Users received a five-minute tutorial in the use of Eudora in an effort to minimize the chances that usability problems with Eudora would affect her results. So Eudora must *not* be security software, at least not for the purpose of *Johnny*.

But what about Microsoft’s Outlook Express (OE)—is Outlook Express security software? Like PGP, OE provides facilities for managing S/MIME certificates (called “Digital IDs”) and for sending email

that is digitally signed and sealed.

Are Microsoft Internet Explorer and Microsoft Word examples of “security software?” Given the orientation of Whitten’s research, it is hard to imagine that IE and Word could be considered security software. But given the number of security problems that have plagued both IE and Word—including remote attacks and the release of confidential information—it is hard to imagine that they are not. What’s more, Word has provisions for both encrypting and digitally signing documents. What about the Palm operating system? In 2001, researchers at the security firm @Stake found a way to bypass the Palm’s password lockout feature and allow attackers to expose information that the Palm’s owner had declared “private.”[Kin01] A fundamental flaw such as this in the Palm system is certainly an example of the “hidden failure property” and “weakest link property.” The fact that the flaw is the result of a fundamental design error on the part of the Palm’s designers might be an example of the “secondary goal property” on the part of the designers—when these designers set out to create a revolutionary handheld computer, security was not their primary goal.

One way to save this term “security software” would be to reinterpret it to apply to the components of any system that provides security services. Alas, much of the last 15 years of research in the field of computer security has been aimed at showing that *a flaw in practically any part of any program can have a devastating impact on overall system security*. Consider a hypothetical bug in the Microsoft Windows operating which would cause the string “-R” on a button to be displayed as “-I”, but only on the first Tuesday in November of each year. Such a bug could have a significant impact on voting software built on top of Windows and would almost certainly be considered a security bug. On the other hand, if the term “security software” needs to be expanded to include the entire graphical user interface library, then it is a hard to imagine what would not be part of the security system.

It appears that the Whitten/Tygar properties are general properties should not be restricted to “security software,” but instead should be applied to software in general. As developers have repeatedly learned in recent years, security suffers when security is not a primary concern—of developers or users.

### **Critique #2: the emphasis on disclosure control**

Textbooks on computer security typically use terms such as *Confidentiality*, *Integrity*, *Availability* and *Audit* to describe the discipline’s goals.[GS91] The second major problem with the Whitten/Tygar contribution is that the “properties of security software” emphasize disclosure control—the goal of “confidentiality”—above all others.

Disclosure control is a goal that is fundamentally different from the other goals of computer security and, in many ways, a goal that is significantly harder to achieve. Disclosure control is indeed hard for all of the reasons that Whitten and Tygar identify: it is hard to know if a system is properly configured to prevent private information from being disclosed, it is impossible to know when information is stolen, and once information is stolen, closing the barn door is no longer relevant—the data is out.

But is disclosure control the most important goal of computer security? Clark and Wilson argue that the emphasis on disclosure control comes from the role that military users have played in the development of computer security requirements. The researchers argue that commercial users have

different priorities and requirements, giving goals such as integrity management a higher priority than disclosure control.[CW87]

It's easy to understand why the military has focused so heavily on disclosure control. Although there are surprisingly few areas in which the disclosure of information is an unrecoverable event, military intelligence is one such an area. If a Russian operative learns the name of a person in the Kremlin who is on the payroll of the US Central Intelligence Agency, that person will probably be killed. It is better for a computer system that contains such powerful information to destroy itself rather than to release this information to an adversary.

On the other hand, few personal or commercial operations require such drastic means to prevent the disclosure of information. Leder *et al.* describe Faces, a cellphone-based system that provides location information to friends and family. The Faces system keeps a log of what kinds of disclosures are made—for example, telling Tina's mother that Tina and her friends went to the mall after school—and then gives the user a control for blocking such disclosures from happening again in the future:

“Some might object to the Faces disclosure log by claiming that informing the user about a disagreeable disclosure *after the fact* is too late to be useful. While this may apply to highly sensitive disclosures, a significant component of privacy maintenance is the regulation of mundane disclosures *over time* to influence observers' historical, evolving impressions of one's self. People are remarkably capable of finessing the consequences of the occasional—and inevitable—disagreeable disclosure, and they learn to minimize repeat occurrences. The Faces disclosure log was intended to help users transfer such iterative behavior refinement to the domain of the sensed environment.”[LHDL05] (emphasis in original)

Although it is unfortunate and costly if a database containing thousands or millions of credit card numbers is stolen, many opportunities exist to recover from such an event. For example, the stolen credit card numbers can be monitored with increased vigilance for evidence of fraud. Alternatively, the credit-card numbers can be canceled, for instance, and new credit cards can be issued to the affected consumers. The disclosure of RSA Security's RC2 and RC4 algorithms in the 1990s did not stop the company from successfully asserting trade secret status of the algorithms and stopping their incorporation into the products of companies that had not obtained licenses for them—at least for a period of time until there was no longer an incentive for RSA to be licensing proprietary encryption algorithms.

Whitten argues: “As with safeware, computer security users must avoid making a variety of dangerous errors, because once those errors are made, it is difficult or impossible to reverse their effects.”[Whi04a, p.6] This statement is hard to reconcile with Matt Bishop's statistic that “configuration errors are the probable cause of more than 90% of all computer security failures”[Bis96]—a statistic that Whitten cites. Configuration errors, when they are made, can persist for many weeks or months without being exploited. Yet once discovered, they can be readily corrected.

Configuration errors can result in a number of different kinds of security problems. Some, like disclosure, can be exploited without detection. But others, such as assaults on integrity, can be

readily detected through integrity management tools such as Tripwire.[KS94] Sometimes reversing a configuration error is as simple as re-enabling one's own antivirus protection.

Although the discussion of disclosure control is interesting and important, by ignoring other important goals of computer security, Whitten and Tygar miss the opportunity to identify other properties of software that make it difficult to align security and usability.

### **Critique #3: the case against making security invisible**

Before she can discuss techniques for "Making Security Usable" (the title of her dissertation) Whitten briefly discusses and then discards another approach: "Making Security Invisible."

To the uninitiated, making security invisible certainly sounds like an attractive approach. Programmers should work hard to make the system always do the right thing, and eliminate the possibility of any security problem. Unfortunately, she writes, making security invisible is a tempting but unworkable proposal. "In practice, making a particular component of security invisible will often lead to a different set of security risks, equally as serious as any that were prevented." [Whi04a, p.8]

Instead of making security invisible, Whitten argues that it is better to teach users to manage their own security. Her thesis contains four such arguments supporting this cause:

1. The argument for making security invisible is "self-perpetuating: if security is hidden from users, then users will remain ignorant about security technology, and their continuing ignorance will be used to justify continuing to hide security from them."
2. The argument for making security invisible contains a conflict-of-interest on the part of manufacturers: "to argue that users cannot manage their own security is to argue that software manufacturers must manage users' security for them. Those same software manufacturers often have a strong financial interest in collecting data on users' habits, actions and preferences, and in privileging their own software over that of competitors in matters of access control. To put them in control of the very security policies that are intended to guard user privacy and resources is thus to put the fox in charge of the henhouse."
3. Telling users that security should be invisible because visible security will annoy users is both a self-perpetuating and "risky" argument "in which software manufacturers make security invisible, despite the risks that creates, market their products as protecting user security, and thus generate and support a widespread user expectation that security can be provided invisibly."

A telling footnote in her thesis elaborates on this point:

"This phenomenon was frequently observed during the software user testing that will be described later in this dissertation; when presented with a software program incorporating visible public key cryptography, users often complained during the first 10-15 minutes of the testing that they would expect "that kind of thing" to be handled invisibly. As their exposure to the software continued and their understanding of the security mechanisms grew, they generally ceased to make that complaint." [Whi04a, p.11, footnote]

4. Security should not be made invisible until there are better tools for assessing the success of such designs. "We need to be wary of assuming success based on a lack of negative feedback." [Whi04a, pp.10-11]



- 
1. If a user action in the application depends on a particular security function for protection, and there is any possibility that the security function may sometimes not be able to be executed, then, in the case that the security function cannot be executed, one of the following clauses MUST be met:
    - a. The user action MUST be completely disallowed, both inside and outside the application.
    - b. Or, the lack of protection for the user action MUST be made visible to the user, and tools for remedying the problem that prevents the execution of the security function SHOULD be made available to the user.
  2. If a security policy in the application determines who is granted access to resources that the user owns, then both of the following clauses apply:
    - a. That security policy MUST be made visible to the user.
    - b. Tools for modifying that security policy SHOULD be made visible to the user.
- 

Figure 2-7: Whitten's rules for making security invisible[Whi04a, Figure 2-2, p.9]

The argument for making security visible and managed by the user is not surprising, given that Whitten's dissertation presents a system designed to teach the fundamentals of public key cryptography. These arguments seem similar to those of a mathematics teacher arguing that students should learn how to perform long division rather than relying on handheld calculators. Yes, it is intellectually interesting and perhaps even important to learn long division, but most people rely on their calculators, even though most calculators present quotients as truncated decimal representations rather than as rational numbers or repeating decimal fractions.

The problem with these rules is that they assume that users will always make decisions correctly: security cannot be made invisible if there is a chance that the automatic system will make a mistake. But what if there is a class of attacks against which machines consistently make better judgments than humans? In these cases, it may make more sense to make the security policy and decisions *visible*, but not to allow the policy to be modified.

### 2.3.2 Yee's Actor-Ability model

Yee notes that secure usability is a system property, observing that "correct use of software is just as important as the correctness of the software itself:"

"[T]here is nothing inherently incorrect about a program that deletes files. But when such a program happens to delete files against our wishes, we perceive a security violation. In a different situation, the *inability* to command the program to delete files could also be a serious security problem.[Yee03]

What is at issue, Yee argues, is not the mere abilities of a program or a process, but how those abilities compare with the expectations of the user.

This insight is the basis of Yee's *Actor-Ability Model* [Yee02, Yee03], which he uses to describe the apparent conflict between the way that users expect their computers to operate and the ways that they can actually operate.

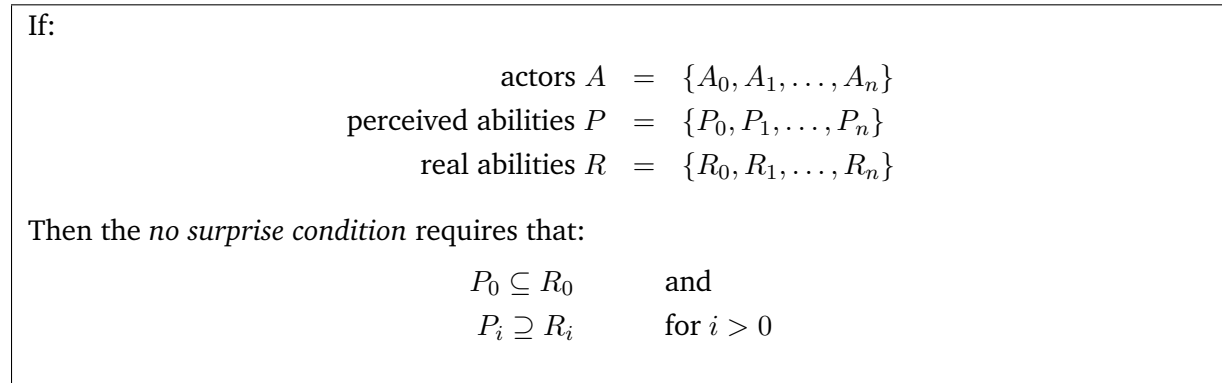


Figure 2-8: Yee's *No Surprise Condition* states that the computer user should be more powerful than she imagines, and that all of the software running on the computer should be less powerful than she imagines.

The Actor-Ability Model is based on the capabilities available to the discrete actors resident on the user's computer. The computer's primary actor,  $A_0$ , is the computer's user. But all computers have other actors—programs like Microsoft Word, and web browsers—which are capable of their own actions. Yee calls these actors  $A_1 \dots A_n$ .

Yee proposes that there are a set of *perceived abilities* that the user believes each actor  $i$  can perform, which he calls  $P_i$ . He notes that the range of actions available to the actor doesn't necessarily match the range of actions that the user believes the actor is capable of: he defines the range of actions that the actor can actually perform as  $R_i$ . He then argues that there is a *no-surprise condition* (Figure 2-8) that is true when the user is more powerful than she realizes and the other actors on the system are less powerful than she believes.

Using this model as a basis, Yee developed a list of ten “suggested principles” or “goals” for “secure interaction design.” He divides these principles into *Fundamental Principles*, *Actor-Ability State*, and *Input and Output* principles. In [Yee05a], he augments each goal with a *test* that can be used to determine if the goal is realized in a piece of secure, usable software. These goals are not the result of a systematic investigation, but are instead based on discussion with “security experts about their experiences designing software that had to be both usable and secure.” Yee's goals appear in Figure 2-9.

Many of Yee's goals cannot be accomplished on today's Windows or MacOS-based computers. For example, Ye and Smith [YS02] note that today's browsers make it possible for a hostile web site to use a combination of JavaScript and loadable images to simulate an entire web browser user interface, complete with address bar, pull-down menus, bookmarks, and even the SSL lock or key icon. As a result, they argue, creating a trusted path between the web browser and the user requires extraordinary measures. An approach that they suggest is a constantly changing visual context that is inaccessible to the spoofing web site. Another approach would be a shared-secret between the browser and the user—for example, a trusted icon or photograph created by the user and displayed by the browser.

Some of Yee's goals are in fact achieved in today's operating systems — although perhaps not in Windows. For example, the so-called Document-Modal Dialogs (Sheets) in MacOS X satisfy Yee's

Fundamental Principles	
Safe Path of Least Resistance	Is the most comfortable way to do any task also the safest way?
Relevant Boundaries	Does the interface draw distinctions along boundaries that matter for the user's task?
Actor-Ability State	
Active Authorization	Is the user's access given out without the user's active consent?
Visibility	Can the user view the authority relationships that affect security decisions?
Revocability	Can the user revoke access that he or she previously granted?
Accurate Expectation of Ability	Does the interface cause the user to overestimate his or her own abilities?
Input-Output Principles	
Trusted Path	Is the user's communication channel to an authority-manipulating agent vulnerable to interception or impersonation?
Identifiability	Can objects or actions have misleading or confusingly similar names or appearances?
Expressiveness	Are users given the means to express safe security policies in terms that fit their tasks?
Clear Consequences	Is the user aware of the consequences of authority-manipulating actions before they take effect?

Figure 2-9: Yee's Goals for Secure, Usable Software

*Identifiability* and *Clear Consequences* requirements, by clearly indicating which document window will respond to the buttons clicked on the modal dialog.[App04b]

Yee goes on to argue that one of the most common security violations today—the propagation of e-mail attachment viruses—do not obviously violate any security policy. “The e-mail client correctly displays the message and correctly decodes the attachment; the system correctly executes the virus program when the user opens the attachment. Rather, the problem exists because the functionally correct behavior is inconsistent with what the user would want.”

According to Yee, the fundamental problem with today's operating systems is that programs run with the user's full set of capabilities that may be applied to any object under control of the user. An alternative is to restrict application capabilities to a small set of objects that are designated by the user through some kind of trusted channel. This approach will be discussed in Section 2.4.4.

Yee's created his principles by considering the problem of computer viruses. Although they work well in this space, it is more difficult to apply them to other security requirements such as Audit or Availability.

### 2.3.3 Lederer *et al.*'s “Five Pitfalls in the Design for Privacy”

Lederer *et al.* have identified five “pitfalls” in the design of applications that are designed to protect privacy. These pitfalls were discovered while working on a program called “Faces” which controls the presentation of personal information in a ubiquitous computing environment. In [LHDL04] and [LHDL05] the authors generalize their collection of “pitfalls” and cite examples from other privacy-enhancing tools.

The pitfalls that Lederer *et al.* identify include:

- **Obscuring potential information flow.** Many systems that maintain and attempt to protect personal information do a poor job explaining when the potential for information flow exists. “Making the scope of a system’s privacy implications clear will help users understand its capabilities and limits. This in turn provides grounds for comprehending the *actual* flow of information through the system.”  
For example, Internet Explorer’s control panel allows the user to set a degree of privacy protection, but the meaning of Microsoft’s scale—from “Low” to “High”—is not clear to many users. Second, even though the control appears on a system control panel, it in fact only applies to Internet Explorer’s management of browser cookies—and not even to other privacy issues in the browser, such as the cache or the history of visited sites.
- **Obscuring actual information flow.** Many systems do not make clear *what* information is being conveyed to *whom*. For example, web browsers do not tell users about the existence of cookies and web bugs, let alone report when these devices are used to report personal information from the user back to the primary or a third-party web site.
- **Emphasizing configuration over action.** Systems exhibit this pitfall in two ways. First, many users are unable to clearly articulate their privacy needs in advance: few have ever been asked to do so. Second, even if users could predict their future privacy preferences, users are then forced to specify those preferences in detail using some kind of rule-based logic that is far removed from the day-to-day task of using a computer and then being frustrated by a privacy (or security) setting.
- **Lacking coarse-grained control.** Users frequently want a simple, obvious control that they can use to “make it safe” or “make it private”—even if pressing this button results in making their computer generally unusable. One obvious way to do this with today’s computers is by turning off the power or by pulling out the network cord. Zone Alarm provides a more elegant way of doing this: a button, accessible from the Windows toolbar, which logically disconnects the network.[Ber05b] A simple coarse-grained control for digital cameras is a mechanical shutter, as discussed in Section 9.2.1 on page 304.
- **Inhibiting established practice.** Society and individuals have developed techniques for providing privacy and security: systems should support these practices, but frequently inhibit them. Examples of such practices include *plausible deniability*, “whereby the potential observer cannot determine whether a lack of disclosure was intentional,” and the disclosure of *ambiguous information* such as pseudonyms and imprecise location. Lederer *et al.* note that “Technical systems are notoriously awkward at supporting social nuance.” Examples of systems that fall into this pitfall are location-based tracking devices which always disclose the user’s location. An example of a system that preserves such nuance are instant messaging systems that allow a user to avoid responding to an invitation to chat without having to explain why.

#### 2.3.4 The danger of hyperconfigurability

Sometimes approaches that are intended to promote both security and usability result in the reverse. Section 9.4 makes the argument that an over-indulgence in configuration options has actually made it harder to achieve either goal. Although this is a result that seems obvious to many

people, there seems to be surprisingly little academic work on the direct impact of hyperconfigurability.

Zurko [Zur05a] describes how the database ACLs used by the Lotus Notes/Domino server were made more complex over time in response to pressure from customers and standards committees. According to Zurko, the original Domino database ACLs were quite simple. “The general approach here and elsewhere was to provide something basic, secure and usable.” The system provided a single list of users and groups who had access to the database; each user could be given one of nine different access “levels” on a scale from “No Access” to “Manager.”

Over time, however, customers have requested the ability to create “custom access levels.” Although this is a useful idea in principle, Zurko writes, “each customer has its own notion of how fine grained permissions should be allocated across their organizational roles.”

To provide for greater flexibility, Lotus layered support for the LDAP ACL standards developed by the IETF[WKH97], resulting in “a substantial increase in complexity over what we had provided before.”

Realizing that fine grained control ‘might present usability problems, Lotus conducted a test of four experienced Notes administrators to see if they could complete thirteen tasks making use of the extended ACLs (xACLs). Although some straightforward user interface problems were fixed and addressed as a result of the user test, the experienced users nevertheless had difficulty auditing the fine-grained permissions provided by the xACLs. Writing about a test that involved auditing the xACLs, “only one of the four test subjects was able to complete that task successfully.”

Rather than remove support for xACLs from the product, Lotus added a button labeled “Effective Access” to the user interface “to help with that confusion.” The button determines what accesses a specific individual or group will have to the database.

Jendricke and Markotten discuss the issue of hyperconfigurability in [JtM00]. Arguing that Internet Explorer’s “low-medium-high” settings are too coarse to provide useful control, and that few users have the knowledge necessary to configure IE’s “custom security settings,” the authors instead argue for the creation of an “identity manager” that interposes itself between the user’s computer, the outside world, and all data stored on the system. This identity manager (Figure 2-10) has a task-oriented user interface (Figure 2-11) that keeps track of the role that the individual is playing and ensure that only the appropriate personal information will be shared with the appropriate web-based entities.

Cooper argues that the temperament of most programmers is to add controls. Without a strong manager, controls tend to proliferate. [Coo99, p.96] Indeed, experience has shown that even user interfaces that were deliberately made simple, such as those on the Macintosh and the Palm operating system, have become dramatically more complex with each successive product release.

### 2.3.5 Security engineering with patterns

There is a small but growing body of work that applies design patterns to security engineering.

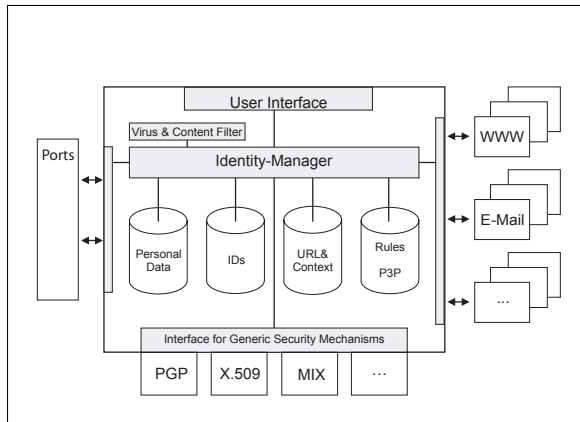


Figure 2-10: The structure of the Jendricke/Markotten iManager, from [JtM00, fig. 8]. Reprinted with permission.

Figure 2-11: The iManager prototype user interface, from [JtM00, fig. 9]. Reprinted with permission.

### The formal study of design patterns

It is generally recognized that the formal study of patterns themselves as a tool for design was initiated by architect Christopher Alexander in his books *The Timeless Way of Building* [Ale79] and *A Pattern Language: Towns, Buildings, Construction*. [AIS77] Alexander found that common architectural techniques—for example, an alcove with a window and a window seat—could be identified, evaluated, and even decomposed into smaller patterns. Alexander coined the phrase *pattern language* to mean a collection of interrelated design patterns that work together to accomplish a specific aim.

Schumacher traces the introduction of patterns into object-oriented design through the work of Ward Cunningham and Kent Beck, who experimented with design patterns in 1987 while working on a user interface consulting job. [Sch03b, p.12] This work was presented at the ACM OOPSLA Conference and published in a technical report. [BC87] Coad took up work and wrote an article popularizing the pattern-based approach. [Coa92]

Patterns took hold at the OOPSLA workshops organized by Bruce Anderson in 1991 and 1992. It was at these meetings that Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides—the so-called Gang of Four (GoF)—met and began working together. They began by collecting patterns and best practices in C++ and object-oriented programming, and ultimately published the textbook *Design Patterns: Elements of reusable Object-Oriented Software*. [GHJV95]

In August 1993 Kent Beck and Grady Boch organized a retreat in Colorado and formed the Hillside Group, which has become the leading force in the movement to integrate design patterns with object-oriented programming. The Group sponsored numerous gatherings that investigated the use of patterns, including the first Pattern Language of Programs (PloP) conference. [Hil05]

### Security patterns

There have been limited efforts in applying the patterns approach to computer security. The most significant contribution to date is Schumacher's treatise, *Security Engineering with Patterns: Origins, Theoretical Model and New Applications* [Sch03b]. Based on Schumacher's dissertation, [Sch03a] this book starts with a history of patterns and pattern ontologies. Schumacher then analyzes the security process, techniques used for improving security, and a security-specific ontology. Although Schumacher does have a chapter on usability, the chapter merely argues that usability is important for security. Finally he presents foundations of security patterns, a theoretical model for creating security patterns, and an example of ways that these patterns can be employed. Although comprehensive from a theoretical point of view, this work suffers from a lack of specific patterns that practitioners can use to actually improve the security of real-world systems.

While Schumacher devotes an entire chapter to the subject of usability and security—Chapter 4, “The Human Factor”—at no point does the book present a pattern for simultaneously improving security and usability. True to its title, *Security Engineering with Patterns* is a thorough treatment of how to do conventional security engineering with patterns—but it is security engineering from the point of view of security researchers who acknowledge the importance of the human factor, and then proceed to ignore it.

Others who have applied patterns to security engineering includes Mouratidis *et al.*, who have developed security patterns for agent systems [MGS03]; and Blakley *et al.*, who authored the book *Security Design Patterns* for The Open Group. [BHm04]

## 2.4 Specific Techniques for Aligning Security and Usability

This section reviews some of the techniques that have been identified for aligning security and usability. Many of the techniques in this section draw on the models and principles presented in the previous section.

### 2.4.1 “User-Centered Security” (Karat, Zurko, Simon)

Perhaps the most straightforward approach to aligning security and usability is to use a traditional user-centered design approach to develop both the non-security aspects *and* the security aspects of user-facing software. Karat pioneered writing about this approach, if not the approach itself, in her 1989 article “Iterative Usability Testing of a Security Application” [Kar89]—a paper that was significantly presented at the annual meeting of the Human Factors Society, rather than at a security conference. In this paper Karat describes how traditional human factors engineering of an IBM mainframe application, including user interviews, paper mock-ups, and the use of prototypes in user studies, resulted in substantial tech support savings and enhanced user acceptance.

Eight years later Zurko and Simon formally introduced the term “user-centered security,”... “to refer to security models, mechanisms, systems, and software that have usability as a primary motivation or goal.” [ZS96] Arguing that “users will not purchase or use security products they cannot understand,” they proposed a framework by which security researchers could return to the emphasis on usability that Saltzer and Schroeder had set forth in 1975. [SS75]

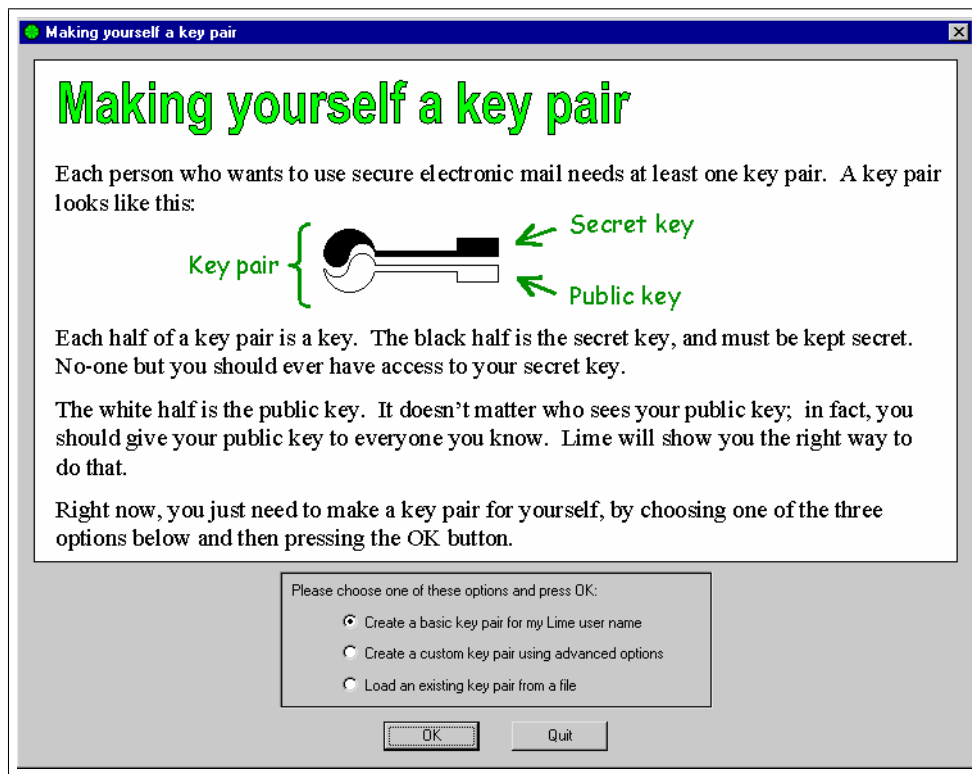


Figure 2-12: The “Making yourself a key pair” screen from Whitten’s “Lime” public key encryption simulation application demonstrates *Safe Staging* and *Metaphor Tailoring*, the two techniques that Whitten developed as part of her dissertation. [Whi04a] *Used with permission*.

#### 2.4.2 “Safe Staging” (Whitten & Tygar)

Whitten and Tygar’s CHI2003 workshop presentation [WT03] introduced a technique called *safe staging*, refined in Whitten’s dissertation, “that can be used to structure a user interface so that users may safely postpone learning how to use a particular security technology until they decide they are ready to do so.” [Whi03]

Safe Staging is a reaction to the user interface of PGP 5.0, which Whitten and Tygar criticize for providing tools without explanations. Safe Staging is implemented as a series of help screens that appear when the user attempts to create a new key; at this point the program provides documentation and gives the user the choice between creating a key pair, creating a “custom key pair using advanced options” (one with higher levels of security), and loading an existing key pair from a file. Figure 2-12 illustrates Whitten’s Safe Staging concept.

#### 2.4.3 “Metaphor Tailoring” (Whitten & Tygar)

Metaphor Tailoring is the second “specialized user interface design technique” that Whitten and Tygar developed. Presented in Whitten’s dissertation, “Metaphor tailoring uses conceptual model specifications that have been augmented with security risk information to create visual representations of security mechanisms and data that incorporate as many desirable visual cues as possible.” [Whi04a, p. iii] Metaphor tailoring is best illustrated by Whitten’s key pair diagram with



the black secret key and the white public key that fit together with Ying-Yang handles, as shown in Figure 2-12.

#### 2.4.4 “Security Through Designation” (Yee)

Yee argues convincingly that many of the issues regarding the access to resources by actors on a computer system can be overcome by giving most actors a minimal set of abilities and then expanding this set item-by-item in response to user actions.[Yee05b]

For example, Yee is critical of the way that programs such as Microsoft Word open files on today’s desktop operating systems. Briefly, programs run by the user have the ability to read or write any file belonging to the user. Thus, when the user wishes to open a file, the following sequence of actions takes place:

1. The user choose the “Open...” command from the “File” menu.
2. Microsoft Word instructs the operating system to display an “Open” dialogue.
3. The user selects the file that is to be opened.
4. The “Open” dialogue returns the name of the file to be opened to Microsoft Word as a string.
5. Word uses the `OpenFileEx()` Win32 API to open a file.
6. Word reads the file contents using `ReadFileEx()`.
7. Word closes the file with `CloseFile()` when it is finished.

The problem with this approach is that there is nothing to stop Word from opening a different file on the user’s hard disk; indeed, there is nothing to stop Word from opening *every file*, scanning for confidential information, and then posting this information to a web site in another country.

An alternative formulation that Yee proposes would remove from Word the ability to open a file by name. Instead, the application would only be able to access files that were opened by the operating system and were then presented to the Word application in the form of a file handle. Thus, the new sequence of actions would look something like this:

1. The user choose the “Open...” command from the “File” menu.
2. Microsoft Word instructs the operating system to display an “Open” dialogue.
3. The user selects the file that is to be opened.
4. The “Open” dialogue returns a file handle of the already-opened file to the Microsoft Word application.
5. Word reads the file contents using `ReadFileEx()`.
6. Word closes the file with `CloseFile()` when it is finished.

The advantage of this approach is that the word processor would be unable to open a file that was not specified by the user. (In such a system, Word would need to be given read-write access to a directory where preferences, templates, and other kinds of long-term persistent information is kept.)

Yee's chapter makes other observations on how security by designation could be easily incorporated into today's existing systems. For example, closing a window is a simple act of revocation that seems implicitly clear to most users: when a document window is closed, the program has lost its authority to access the contents of the document. If the program's main window is closed, the program has lost authority to continue executing and should terminate.

#### 2.4.5 “Rolling Blackouts” for password entry (Tognazzini)

Password entry has been a persistent usability problem for more than four decades. Passwords, by their nature, need to be entered perfectly in order to be used, yet need to remain secret.

Multi-user systems that ran on half-duplex printing terminals in the 1960s always printed what the user typed on the paper, leaving an indelible record. To prevent a user's printout from compromising his or her account, operating systems that used these terminals would prepare a password entry area by overprinting multiple symbols in a single spot. Typically, these systems would combine characters such as the \*, M, W, and O, resulting in 8 black boxes over which passwords could be typed.

Full-duplex printing terminals can disable character echo altogether when passwords are typed. This is the approach that many operating systems took in the 1970s when such terminals became available. Although it was not strictly necessary to disable password display on video terminals, this was commonly done as well because of the so-called “shoulder surfing” problem: users didn't want their passwords echoed on the screen where they might be seen by another person sitting at the terminal.

An alternative approach popularized by many web browsers in the 1990s is to echo passwords as a series of dots, one dot for each character typed. This allows the user to confirm each character typed has been received by the computer, but doesn't allow a nearby attack to observe which character is being typed from the screen. An attacker can, however, determine how many characters are in the password.

Apple's *Human Interface Guidelines* states that each character of the typed password should appear as a bullet, but that when the user leaves the password field the field should be filled with as many bullets as will fit, in order to obscure password's length.[App04c]. This behavior is confusing to some people, however.

Tognazzini presents yet another alternative for improving the typing of passwords: the *rolling blackout*. Tognazzini's password field shows the last three password characters typed using low-contrast light gray characters on a white background. When the fourth character of the password is typed, the first character is changed to bullet. Testing found that this compromise allowed users to visually verify and correct their passwords, but still prohibited shoulder-surfing.[Tog05]

#### 2.4.6 Hash visualization and graphical authentication (Lotus, Perrig and Song)

It is desirable to give users some sort of visual indication that the password they have typed is actually the password that they intended to type. This is especially important for systems that will lock out user accounts when an incorrect password is attempted on multiple occasions. Early versions of Lotus Notes included a system the designers called “password hieroglyphics.” As the

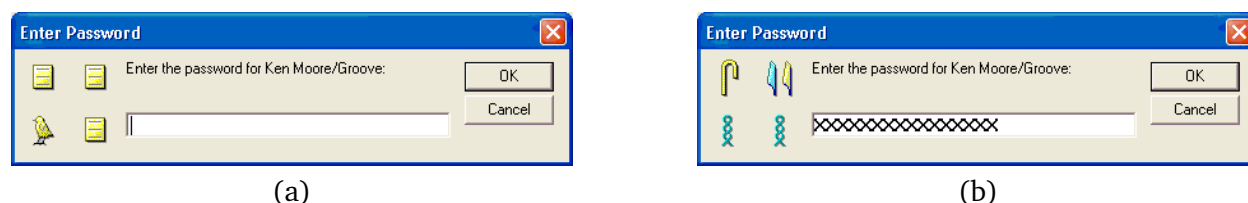


Figure 2-13: The so-called “password hieroglyphics” in the Lotus Notes client allows the user to see a visual hash of the password that has been entered. The hope is that users will learn their password icons and be able to tell in advance if the password that they have typed doesn’t match before they press the OK button.

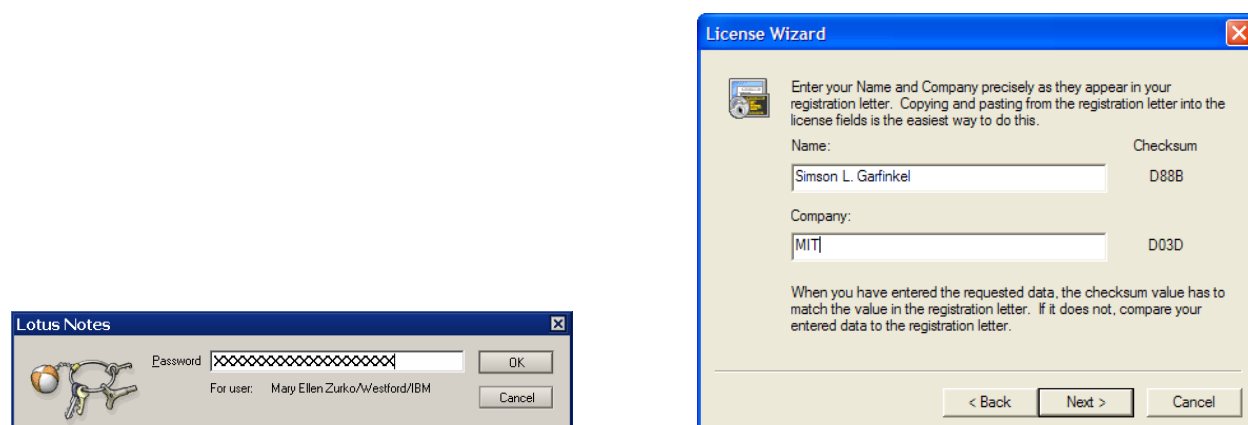


Figure 2-14: In Notes 6 the password hieroglyphics were replaced with a more culturally sensitive key-chain display.

Figure 2-15: The Secure CRT license wizard displays a checksum of the user’s typed Name and Company to facilitate correct entry of these items.

user typed each key of the password, a hash of the typed letters is computed and used to display a set of four glyphs, as shown in Figure 2-13.

In the latest release of Lotus Notes the hieroglyphics have been replaced with a more culturally sensitive display based on a set of cartoons, as shown in Figure 2-14. These can be combined to generate a large number of different password hashes, as shown in Figure 2-16.

Displaying a hash or checksum of what the user types is a useful technique whenever information must be typed precisely. The popular Secure CRT virtual terminal program uses a license management system where license strings are keyed to the exact spelling of a person’s Name and Company. To facilitate proper entry, the program displays a “Checksum” of the user’s typing to the right of the input field, as shown in Figure 2-15.

Perrig and Song propose using these kinds of visualization techniques to allow people to visually compare the hashes associated with cryptographic keys. The reason is that it is relatively easy for an attacker to create a key that has a chosen set of hexadecimal digits at the beginning and at the end as a target key but which differs in the middle. It is possible, the pair asserts, that many human beings will think that the fingerprints  $F_1 = 51:86:E8:39:87:87:F3:87:83:10:AA:87:35:98:E0:AA$  and  $F_2 = 51:86:E8:45:88:F0:F3:F9:F3:31:99:33:5F:98:E0:AA$  are the same, when in fact they are different.



Figure 2-16: A selection of 14 visualized password hashes from the Lotus Notes 6 Client. Different bits of the hash appear to select different keychain fobs (e.g., a ball, a tag, etc.), the coloring of the fob, the number of keys, and the placement of those keys. Although Lotus has apparently not documented the algorithm or the visual keyspace, it appears that at least  $2^{32}$  different keychains can be displayed from the variety present in these 14 images. *Courtesy Henry Holtzman, MIT Media Lab. Reprinted with permission.*

Instead of using computer-generated cartoons, Perrig and Song use random “art” that is generated using a set of mathematical functions that are controlled by a set of parameters (Figure 2-17). The idea is to use the hash to specify the parameters: in these sorts of chaotic systems, very small changes can have very large effects. The pair also suggest that palettes of computer generated art can be used as the basic building-block of a picture-based authentication system.[PS99]

Widespread use of random art for hashes has a number of potential problems, as Laurén noted in a recent posting to the HCI-SEC mailing list:

- If random art is used as the primary visualization, it is important that all possible values be distinguishable: no two different hashes can have representations that are visually indistinguishable.
- Reproducing visual hashes on business cards might be problematic.
- Individual and corporations might be concerned if the visualizations of their identifiers or keys are not aesthetically pleasing or that do not match the color schemes employed by the key holder’s web site.[Lau05]

There are also concerns that the random art might not look truly different for different hash values. If true, then it might be possible for an attacker to create two very different keys that nevertheless have very similar, and possibly indistinguishable, visual hashes.

Some of these concerns could be overcome if a strong visual hash algorithm were standardized. Such an algorithm would presumably not have the collision problem. Furthermore, an organization could keep creating new public keys until it found one with an attractive hash representation. On the other hand, an organization that changed the color scheme of its web site might wish to simultaneously change its public key to match the new scheme; such changes would defeat the purpose of human-verifiable hash visualizations in the first place!

#### 2.4.7 “Instant PKI” (Balfanz, Durfee and Smetters)

Balfanz, Durfee and Smetters at PARC have demonstrated that PKI systems can be made dramatically easier to configure and deploy by replacing certificates designed to convey identity with single-use certificates that are treated as capabilities. They call this approach “Instant PKI.”[BDSG04]

The PARC implementation is designed to provide a laptop with an X.509 certificate that can be

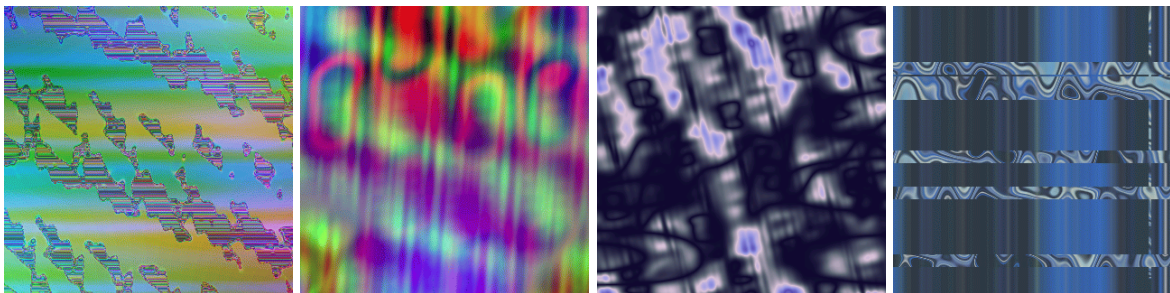


Figure 2-17: “Random art” images, courtesy of Adrian Perrig. Reprinted with permission.

used for authentication on a 802.11x EAP-TLS network. Although the technology to issue and install these certificates is widely available, there are many acknowledged usability problems with current implementations.

In their original experiment, eight subjects—most holding Ph.D.’s in Computer Science—were provided with a set of instructions that clearly described the 38 distinct steps required to configure Microsoft Windows XP to authenticate over a wireless 802.11x network. The average time for subjects to request and retrieve their certificates, then configure their laptops, was 140 minutes. Several of the subjects reported that the process was “the most difficult computer task that PARC had ever asked them to do.”[BDSG05]

The revised system, based on the Instant PKI concept, authenticates not individuals but the laptops themselves. The laptops are given a helper application that communicates with a local CA over an infrared link in a secure room. The laptop creates a public/private key pair, sends it over the wireless link to the CA, receives in response the signed certificate, and installs it. The total time from beginning to end is approximately 32 seconds.[BDSG05]

The PARC system is important because it shows that many concepts being discussed in the HCI-SEC community actually work in a commercial environment. These concepts include:

- The use of single-purpose, identity-free certificates.
- The leveraging of existing social methods of authentication. In this case, anyone who could convince the network administrator to open the locked room was assumed to be allowed access to the wireless network.
- The use of physical proximity as a surrogate for trust.

#### 2.4.8 E-mail Based Identification and Authentication (Garfinkel)

Gutmann observes that E-mail based identification and authorization (EBIA)—the ability to receive email at a previously registered address—has been widely adopted for automated password resets and mailing list subscriptions. Essentially, EBIA delegates web site identity management to the Internet Service Provider of the user’s choice. Such authentication techniques are probably “good enough,” Gutmann observes, “unless the opponent is the ISP.”[Gut04b] (Although there is one case in which the enemy was in fact the ISP [USA04], this does not seem to be the general case.)

A detailed analysis of EBIA options and present best practices is presented in [Gar03a]. E-mail Based Identification is one of the design patterns outlined in Chapter 10.

## 2.5 Prior and Related Work on Sanitization

There exists a significant body of work in the academic, commercial and hobbyist communities on the topic of disk sanitization. This body of work is surprising when one considers that sanitization tools are rare in commercial operating systems and automatic sanitization is all but nonexistent.

### 2.5.1 PGP's -w option

PGP version 1.0 supported a “-w” option to “wipe” information from the computer’s hard drive. When used in conjunction with an encryption function, the “-w” option would “destroy every trace of plaintext,” according to a comment in the program’s source code.[Zim91a] The “-w” option could also be used without the encryption function, in which case its performed a sanitized erasure of the file whose name was provided on the command line. PGP version 1.0 implemented file wiping in a function called `wipeout()` that moved the file pointer to the beginning of the file and then overwrote the contents with repeated calls to the `fwrite()` command using a zero-filled buffer. As discussed in Chapter 3, there are many cases in which this approach would have silently failed.

### 2.5.2 Secure file deletion under Linux

Remy Card introduced support for Linux file attributes with release 0.4 of ext2fs. [Car96] Card created three attributes: the “c” attribute, which marked a file for automatic compression, the “s” attribute, which marked the file for secure deletion, and the “u” attribute, which marked the file for undeletion.<sup>2</sup> Although all were documented, only the “s” secure deletion attribute was implemented in version 0.4.

The implementation of the “s” attribute required minor modifications to just four locations of the ext2fs code. But support for the “s” attribute was removed from the Linux kernel “as of Linux 2.2,” according to the Linux 2.4 `chattr` man page (Figure 2-18). An examination of the current Linux ext2fs source code shows no trace of Card’s original implementation. It is likely that support for secure deletion was removed when the file system’s block-handling routines were rewritten to achieve higher performance.

Bauer and Priyantha describe a modification to the Linux operating system to support secure deletion. Whereas Card’s implementation set a flag that required the blocks be zeroed before they were placed on the freelist—something that Card implemented in the file system itself—Bauer and Priyantha’s implementation overwrote deleted files asynchronously using a kernel thread. The authors correctly note that “an asynchronous overwriting process sacrifices immediate security but ultimately provides a far more usable and complete secure deletion facility.”[BP01]

Despite being distributed under the Gnu Public License,[Sta01] Bauer and Priyantha’s implementation was not incorporated into the Linux kernel. What’s more, the technology cannot be easily incorporated at this time, owing to the fact that the kernel has changed significantly in the years since Bauer and Priyantha did their work.

---

<sup>2</sup>“when the file is deleted, its contents are saved to allow a future undeletion.”[Car96]

CHATTR(1)	CHATTR(1)
NAME	
	<code>chattr - change file attributes on a Linux second extended file system</code>
SYNOPSIS	
	<code>chattr [ -RV ] [ -v version ] [ mode ] files...</code>
DESCRIPTION	
	<code>chattr changes the file attributes on a Linux second extended file system.</code>
	<code>The format of a symbolic mode is +=[ASacDdIijsTtu].</code>
	<code>The operator '+' causes the selected attributes to be added to the existing attributes of the files; '-' causes them to be removed; and '=' causes them to be the only attributes that the files have.</code>
...	
ATTRIBUTES	
...	<code>When a file with the 's' attribute set is deleted, its blocks are zeroed and written back to the disk.</code>
...	
BUGS AND LIMITATIONS	
	<code>As of Linux 2.2, the 'c', 's', and 'u' attribute are not honored by the kernel filesystem code. These attributes will be implemented in a future ext2 fs version.</code>

Figure 2-18: The documentation for the Linux “chattr” command promises an “s” attribute that, when set, causes files to be securely deleted. Only at the bottom of the page does the document make it clear that the feature has yet to be implemented. From [Car02]

Despite the fact that the Linux ext2fs and ext3fs file systems no longer provide secure deletion facilities, the “s” file attribute is still supported by the `chattr` command. It is only in the “BUGS AND LIMITATIONS” section of the command’s document does one learn that this attribute is ignored by the operating system (Figure 2-18). Given the way that security vulnerabilities seem to be understood in the Open Source community, the failure to implement a documented security feature is a missing feature, and not a security flaw, and is not likely to be fixed anytime soon.

### 2.5.3 Apple’s “Secure Empty Trash”

Following the initial publication of the sanitization work that will be presented in Chapter 3 of this thesis, Apple Computer added a “Secure Empty Trash” function to the Finder component of its MacOS operating system. According to interviews conducted with Apple’s security group on January 12, 2003, the group had long wanted to put a secure file delete function in the operating system’s interface: such efforts had been deemed low priority by Apple’s management until the

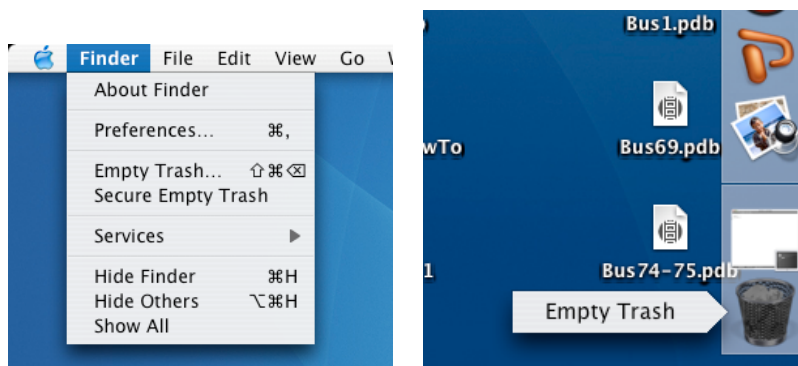


Figure 2-19: Apple added the “Secure Empty Trash” feature to the MacOS 10.3 operating system following the publication of the *Remembrance of Data Passed* paper. Apple’s addition of this feature was incomplete: although the feature was added to the Finder menu (left), it was not added to the control-click menu on the Apple trash can (right).

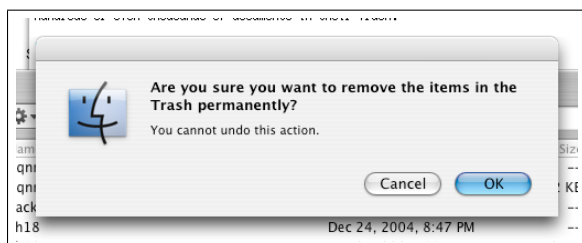


Figure 2-20: Choosing “Empty Trash...” from the File menu of MacOS 10.3’s Finder application causes this alert panel to be displayed.



Figure 2-21: Choosing “Secure Empty Trash” from the File menu of MacOS 10.3’s Finder application causes this alert panel to be displayed.

*Remembrance* paper was published.

Apple implemented “secure file delete” as a modification to its Finder program, rather than as a modification to the operating system’s kernel. After files are dragged to the Trash, the user may choose either the “Empty Trash...” or the “Secure Empty Trash” options from the Finder menu, as shown in Figure 2-19 (left). Choosing these options, respectively, causes the confirmatory alert panels shown in Figures 2-20 and 2-21 to be displayed.

Although it is personally rewarding that a paper published in January 2003 would result in a significant modification to an operation system used by tens of millions of people in less than a year’s time, there is much to critique in Apple’s initial implementation of secure file deletion.

Most obviously, the labeling in the Finder menu items and on the modal alert panel have the obvious marks of a rushed job—perhaps a direct result of the short time between the publication of the *Remembrance* paper and the release of MacOS 10.3:

- The menu title for the item “Empty Trash...” includes three trailing periods, indicating that running the command will not result in the command being run but will result instead in a dialog being displayed. On the other hand, the menu title for the item “Secure Empty Trash” does not include three trailing periods, leading the user to believe that the command will be instantly acted upon by the operating system. The lack of the ellipsis may make the user less



inclined or even fearful to choose the command.

- The subtle difference in text between the two modal panels does not convey the actual difference between the “Empty Trash...” and “Secure Empty Trash” commands. While both actions “remove the items in the Trash permanently,” the “Empty Trash” operation cannot be undone, whereas files deleted with the “Secure Empty Trash” command cannot be “recover[ed].”

Although subtle, the text is literally accurate: MacOS does not provide tools for undeleting files once the files have been removed from the Trash (that is, unlinked from the `~/ .Trash` directory), but third-party utilities do exist for recovering deleted files. These utilities will recover files that have been deleted with “Empty Trash...” but not recover files that have been deleted using “Secure Empty Trash,” as “Secure Empty Trash” overwrites the file blocks. This distinction is made clear by typing “Secure empty trash” into Apple’s Help Viewer and choosing the first answer that comes up. (See “Deleting files and folders” in Figure 2-22.)

- The “Secure Empty Trash” feature is not available from the Trash can’s context menu. A person who only empties the trash by control-clicking on the Trash can’s icon in the MacOS Dock will not discover the feature.
- “Secure Empty Trash” is a very slow procedure, during which time the Finder’s Trash may not be otherwise used: Attempting to drag a file to the Trash causes the Finder to display the message “You cannot move any items to the Trash because it is being emptied.” Double-clicking on the Trash icon causes the Finder to display the message “You cannot open the Trash because it is being emptied.” This is a disincentive to using the “Secure Empty Trash” sanitization facility.
- Because “Secure Empty Trash” is such a slow procedure, it seems that it would be advantageous to be able to specify files to be securely erased on a file-by-file basis. However, there is no way to make such distinctions.
- Likewise, there is no way to securely erase a specific file but leave the other files in the Trash untouched. This poses an inconvenience to users who habitually keep hundreds or even thousands of files in their Trash directories and who wish to securely delete a single file from time to time.<sup>3</sup>
- If the user inadvertently chooses “Empty Trash...” instead of “Secure Empty Trash,” there is no way to go back and securely overwrite the disk blocks once the files have been unlinked from the Trash directory.
- Implementing “Secure Empty Trash” in the Finder, rather than in the operating system’s kernel, means that there is no way to securely delete files that are deleted by programs other than the Finder (e.g., using `rm` or Emacs).
- There is no way to remove from a disk the information that was contained in files that have been “overwritten” using the “Save As...” feature.

Section 3.6.1 proposes another technique for implementing the functionality of file sanitization that should deliver a more usable solution in a way that builds upon both Bauer and Priyantha’s work with Linux [BP01] and Apple’s work with MacOS.

---

<sup>3</sup>There is a rather straightforward albeit annoying work-around for this problem: Simply move all of the files that are in the Trash into a second, temporary directory, leaving behind the files to be sanitized. Choose “Secure Empty Trash.” Finally, move the files from the temporary directory back into the Trash.

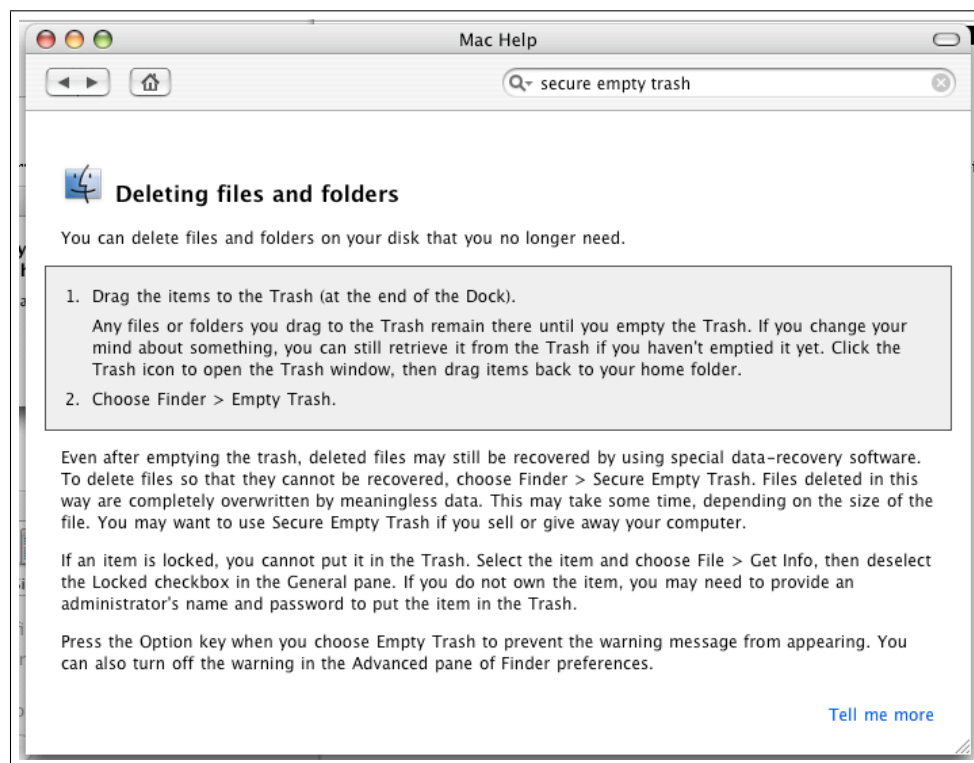


Figure 2-22: Typing “Secure empty trash” into the MacOS 10.3 Apple Help Viewer causes the application to display this help page describing the difference between Empty Trash and Secure Empty Trash.

#### 2.5.4 Cryptographic file systems

As discussed in Section 3.2, cryptographic file systems provide a partial solution to the data remanence problem: data is simply stored on a cryptographic file system. When it is time to throw away the drive, the user only needs to ensure that the key that was used to encrypt the data is properly destroyed. Once that key is gone, there should be no chance of recovering the data.

In practice the use of cryptographic file systems is slightly more complicated. Apple’s MacOS 10.3 operating system contains a cryptographic file system called File Vault. This system is implemented through Apple’s “Disk Utility” subsystem that allows a disk file to be mounted on the MacOS desktop as if the file were an external device. When used with File Vault, each block of the file is encrypted with the AES-128 cipher. The key, encrypted with the user’s login password, is stored at the beginning of the disk file.

File Vault is designed to be used with home directories on laptops and desktops. When the user logs in, MacOS automatically uses their login password to decrypt the key used for the particular File Vault file. The file is then mounted as the user’s home directory and login proceeds. By design MacOS applications keep all user-specific data inside the user’s home directory. For example, the popular Firefox web browser for MacOS keeps each user’s browser cache in the directory `~/Library/ApplicationSupport/Firefox/Profiles/`.

Although File Vault is very easy-to-use—once it is enabled, the user is more or less oblivious to its

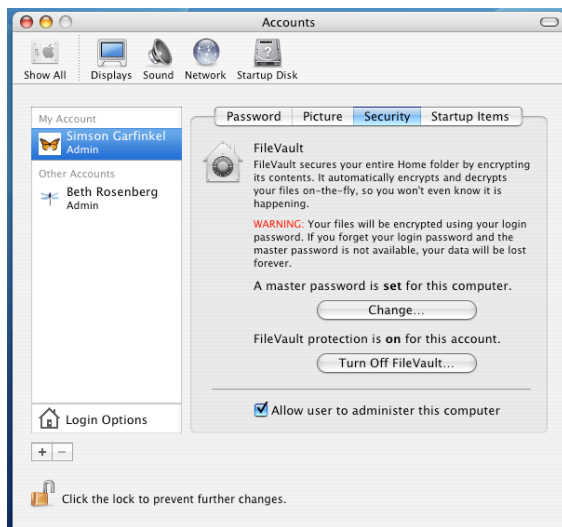


Figure 2-23: Apple's File Vault cryptographic file system is enabled and controlled through the System Preferences panel.



Figure 2-24: When File Vault is first enabled, it creates a cryptographically protected virtual disk, copies the user's files to that disk, and then deletes unencrypted files. Analysis of a disk revealed that File Vault does not use Complete Delete to delete the unencrypted files. As a result, they can be recovered using forensic tools.

presence—it does not implement the sanitization patterns described in Chapter 10. Using these patterns to guide an analysis of File Vault, the following problems were observed:

- When File Vault is first enabled for a user, a cryptographic volume is created, the user's files are copied to the volume, and finally the files are deleted but not overwritten. (File Vault will not engage unless the amount of free space on the hard drive is larger than the amount of space taken up by the user's directory.) As a result, after File Vault is engaged, unencrypted copies of *all of the user's files* can be recovered from the hard drive using standard “undelete” programs. (This claim was verified by recovering 100% of the blocks in a 512MB file that was in a user's home directory prior to File Vault being enabled.)
- After File Vault is enabled, there is no way for an unassisted end-user go back and use the MacOS “Secure Empty Trash” to actually overwrite the deleted files. This is because files, once deleted, cannot be unerased using the tools provided by the operating system. (The Disk Utility command of MacOS 10.4 includes the ability to erase the unallocated space on a volume. The function is implemented by a small program that creates a big file on the disk, then deletes the files after the disk is filled up. This is the approach used by \W switch of Microsoft's CIPHER.EXE, described in the next section, and is likely to have similar security problems.)
- There is no indication that the cleartext copies of the files deleted by the File Vault enabling process can still be recovered using forensic tools, a violation of the “User Audit” pattern.
- File Vault does not have clean integration to support media disposal. Ideally the operating system would have a simple provision for securely overwriting the File Vault encryption keys

when the computer is no longer needed. Because this functionality is missing, it is possible for an attacker who recovers a discarded computer to mount a guessing attack against the user's File Vault password. Such attacks are frequently successful.

But the most important reason that cryptographic file systems are not a solution to the file sanitization problem is that deleted files can be recovered from such a file system when the volume is mounted. A shared computer that is running in a library or in an office will still have files that are deleted but recoverable on its hard drive, even if password needs to be typed when the system is first turned on. An attacker who has access to such a system will be able to recover data that was intentionally deleted but not overwritten.

### 2.5.5 Microsoft's CIPHER.EXE

After a talk on the *Remembrance of Data Passed* project at Microsoft Research in July 2002, a member of the audience stated that Microsoft had already addressed the problem of deleted data with a command-line utility called CIPHER.EXE. This assertion was repeated during two days of interviews performed at Microsoft in January 2004.

CIPHER.EXE is a command-line tool for controlling aspects of the Microsoft Cryptographic File System (CFS) that was introduced with Windows 2000. A feature of CFS is that it can be enabled on a disk-by-disk or directory-by-directory basis. When CFS is enabled for a directory, the operating system encrypts each of the files in the directory and then unlinks the plaintext files when the encryption is done. This behavior generated consternation within the CFS group that the unencrypted files were still on the computer's disk at the end of this process—the files were deleted, but recoverable. (This is, in fact, the exact security problem that was discovered with Apple's File Vault, as discussed in the preceding section.)

Microsoft's solution to this problem was to add an option to the CIPHER.EXE that would sanitize deleted information on the hard drive, thus erasing the contents of the deleted files. Documentation for this option appears in Figure 2-25. The program accomplishes this goal by opening a single file for writing and then writing to that file until there is no more space on the device. This is the same procedure that several free and commercial tools use. Unfortunately, it does not fully sanitize sensitive information from the disk, as is discussed below.

Following the release of Windows XP with the improved CIPHER.EXE command, Guidance Software, makers of the EnCase forensics tool, published a provocative whitepaper entitled "Can Computer Investigations Survive Windows XP." In that whitepaper authors Stone and Keightley evaluated the sanitization capabilities of CIPHER.EXE and found them lacking:

**"Results:** All unallocated space was filled with random values (which greatly affected file compression in the evidence file); however, the cipher tool affected only the unallocated clusters and a very small portion of the MFT<sup>4</sup>; 10–15 records were overwritten in the MFT, and the majority of the records marked for deletion went untouched). The utility does not affect other items of evidentiary interest on the typical NTFS partition, such as: file slack, registry files, the pagefile and file shortcuts.

---

<sup>4</sup>Master File Table

“In terms of its anticipated end-user adoption, the cipher feature is a burdensome command-line utility that is difficult to find and operate. Notably, the cipher function is available on the Professional version, but not included in the Home version of XP and Windows 2000. Despite some speculation, the function is not set by default.”[SK03]

The white paper concludes:

“...The scrubbing feature is part of Windows XP, but it is not all that it was initially thought to be. It is a command line tool that is difficult to use, time consuming and nothing more than a good wiping utility. The average computer user will not know how to use it, and even if it is used, evidence artifacts still remain in certain system files.”[SK03]

As with Apple’s “Secure Empty Trash,” `CIPHER.EXE` is an example of a program that literally solves the problem that the operating system vendor set out to solve. However, the solution is limited in scope, burdensome to use, and ultimately doesn’t provide users with the protection that would be afforded by a more comprehensive solution.

### 2.5.6 Microsoft’s “Remove Hidden Data” tool

Starting in August 2004 Microsoft prepared a series of Knowledge Base articles that instruct users on fast saves and the proper procedure for removing metadata from their Word and PowerPoint documents. [Cor04a, Cor04b, Cor05b, Cor05c, Cor05a]

In August 2004 Microsoft also released its “Remove Hidden Data tool for Office 2003 and Office XP.”[Cor04c] This tool is designed to remove much of the metadata, revisions, and other potentially embarrassing information that had been the source of so many media reports. The Remove Hidden Data tool automatically removes more than a dozen kinds of hidden information and metadata, including:

- Comments
- Previous authors and editors
- The User name
- Personal summary information.
- Revision marks (if there are revisions pending in the document, the tool automatically accepts all revisions)
- Deleted text
- Previous versions and versioning information.
- Descriptions and comments are removed
- from Visual Basic Macros and modules
- The ID number used to identify the document (ID numbers are used by Word to allow changes to be automatically merged back into the original document using some document management systems)
- Routing slips
- E-mail headers
- Scenario comments
- Office 97 unique identifiers (these identifiers were removed from later versions of Word)[Cor04c]

Even though Microsoft sells a version of Word that runs on the Macintosh operating system, the Remove Hidden Data tool is only available for Microsoft Windows.

```
C:\Documents and Settings\simsong>cipher /?
Displays or alters the encryption of directories [files] on NTFS partitions.

CIPHER [/E | /D] [/S:directory] [/A] [/I] [/F] [/Q] [/H] [pathname [...]]

CIPHER /K

CIPHER /R:filename

CIPHER /U [/N]

CIPHER /W:directory

CIPHER /X[:efsfile] [filename]
...
/W      Removes data from available unused disk space on the entire
volume. If this option is chosen, all other options are ignored.
The directory specified can be anywhere in a local volume. If it
is a mount point or points to a directory in another volume, the
data on that volume will be removed.
...

directory A directory path.
filename  A filename without extensions.
pathname  Specifies a pattern, file or directory.
efsfile   An encrypted file path.

Used without parameters, CIPHER displays the encryption state of
the current directory and any files it contains. You may use multiple
directory names and wildcards. You must put spaces between multiple
parameters.
```

Figure 2-25: The /W option added to CIPHER.EXE in Windows XP Professional

Overall, the Remove Hidden Data tool has the look of a rush job (Figure 2-26) and does not have the level of professionalism evident in other parts of the Office application.

### Evaluation of the Tool

One problem with the “Remove Hidden Data” tool is that there is no obvious way to look at a Word file and determine if it has been processed with the tool or not. This is similar to the problem of being unable to determine if a hard drive has been properly sanitized after an overwriting program has allegedly been run.

It’s important to note that “Remove Hidden Data” doesn’t protect a publisher from accidentally publishing a document with confidential content that was constructed with the malicious intention of defeating Microsoft’s sanitization system. For example, a document could be created using a Word Macro that displays confidential material after a certain date. Because “Remove Hidden Data” does not remove macros from documents, the hidden content in such a document will not be removed because it is not technically “hidden:” it is simply active content that changes in form at a previously designated time.

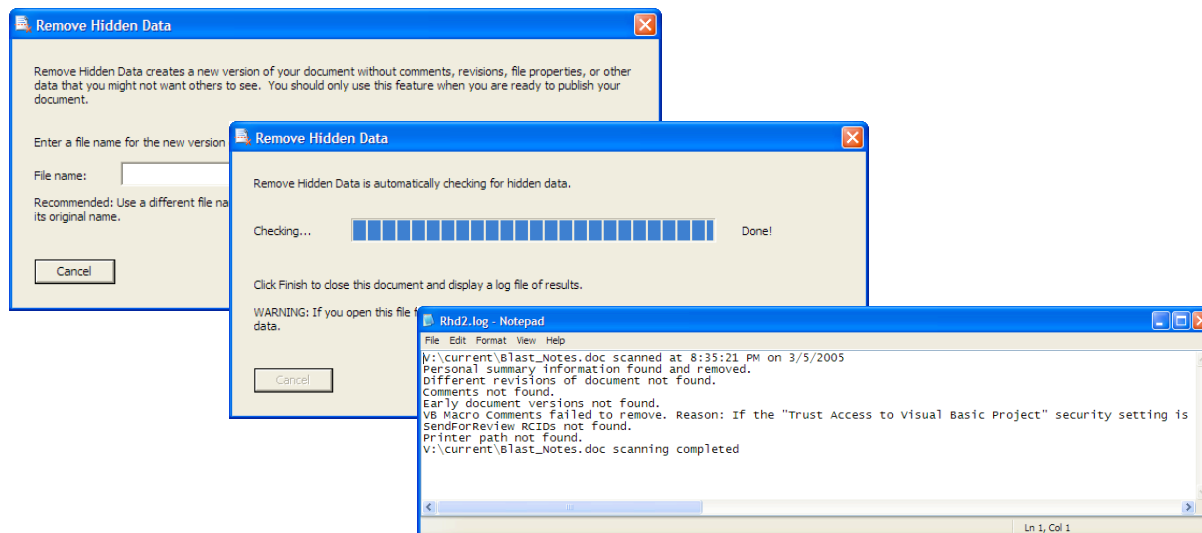


Figure 2-26: Microsoft's Remove Hidden Data before (left) and after (center) a file is sanitized. After the program runs, the program's log file is opened in the Windows Notepad application (right).

### 2.5.7 Rivest's incremental key forgetting proposal

An excellent way to implement a sanitizing delete is to store files encrypted and then, when the files are no longer needed, both sanitize the key and unlink the files. This avoids what is sometimes known as the “Oliver North Problem”—deleting documents only to be done in by data stored on backups (Figure 2-27). Of course, it is important that the key actually be deleted and that it not be backed up in any location itself.

Instead of throwing away the key, Rivest proposed the idea of throwing away bits of the key on some sort of schedule.[Riv04] For example, if the document in question were encrypted with a 128-bit AES key, and if a modern computer can try 100,000 AES keys per second, then throwing away 24 bits of key would protect the data to be sanitized from most casual attackers (and certainly from a large-scale analysis of several hundred disk drives) yet require at most 168 seconds try all  $2^{24}$  possible keys. If the key recovery operation had not be initiated within an hour, the computer could proceed to throw away another 9 bits (for a total of 33 bits discarded), which would slow the data recovery time to roughly a day. If the data wasn't needed for another month, another 3 bits could be discarded, increasing the recovery time to a little less than a week, and so on.

The advantage of this approach is that the encrypted data is increasingly rendered difficult to retrieve as more and more bits of the key are thrown away, but the data is never placed beyond recovery if one is desperate enough. The theory is that the computer user will have the determination to recover the key (and the data) only if it is really needed—but a casual attacker (or an automated program) will not have the needed determination. Another advantage is that, although it is possible to recover an individual file or two, it becomes progressively harder over time for an attacker on a limited budget to recover *all* of the data that has been processed in this manner.



Figure 2-27: ‘We all sincerely believed that when we send a PROFS message to another party and pressed the button ‘delete’ that it was gone forever.’ — Lt. Col. Oliver North testifying before Congress, July 8, 1987 [Sip95]



Figure 2-28: The hypothetical Shredder Control Panel. Cartoon shredder by Clay Bennett, *The Christian Science Monitor*. Used with permission.

### 2.5.8 Ephemeral communications

Another approach to the data remanence problem is to use an *ephemerizing service*. Such a service creates secret encryption keys that have widely publicized expiration dates. To create a message  $M$  that will be unreadable after a particular date  $D$  the user simply encrypts a document with key  $K$  and then sends  $K$  to the ephemerizer with a request that  $K$  be encrypted with  $K_D$ , the key that expires

$$\langle \{M\}_K, \{K\}_{K_D}, D \rangle$$

A commercial system based on this approach was fielded in the late 1990s by a company called Disappearing Ink which changed its name to Omniva and was acquired by Liquid Machines in 2004. Omniva’s technology is described in [GS02b, pp.280-283] Perlman presents an improved solution to this problem in her technical paper *The Ephemerizer: Making Data Disappear*. [Per05a]

### 2.5.9 Understanding data lifetime via whole system simulation

Chow *et al.* have examined the lifetime of sensitive and confidential information in the Unix operating system using an approach they call “TaintBochs” which are tracked through a simulation of a running system.[CPG<sup>+</sup>04] Using their system, they have determined that conventional systems scatter sensitive data such as passwords, credit card numbers, and encryption keys throughout the system’s kernel memory, user memory, and swap space. Unless specific measures are taken to sanitize these memories, the researchers have found that such information may persist for the lifetime of the computer system. They have also found that relatively simple measures—such as explicitly overwriting memory that is freed—can drastically reduce the extent of the problem.



### 2.5.10 Other work on Disk Sanitization

There are a substantial number of programs available for sanitizing sensitive files and/or entire disk drives. Some of these programs are free, while others are commercial.

In general, programs that sanitize the entire hard drive are easier to write and easier to verify than programs that attempt to sanitize individual files. Horn has created an excellent program called DBAN that boots a customized version of Linux from a floppy disk or CDROM and proceeds to erase the computer's hard drive (after first asking for confirmation, of course!)[Hor05] The MacOS 10 Disk Utility program includes the ability to overwrite a disk with a pass of "zeros" (ASCII NULs), with eight passes of random data, or (in version 10.4) with 35 passes of data, when a disk is formatted; Figure 2-29 shows a screen shot of the version 10.3 Disk Utility at work. (The origin of the 35-pass process and the fact that it was *never* necessary to subject any given hard drive to 35 passes of overwriting is discussed at length in [Gut96].) The default is not to overwrite the media but to simply write a new root directory.

Programs that attempt to selectively sanitize some information have a much harder time—the problem is allowing the user to specify what information is to be sanitized in a manner that is usable. AccessData's Secure Clean and PGP Personal Privacy load an extension to the Windows File Manager that allows any file to be sanitized through right-clicking on the file and then choosing a menu option. More challenging is the job of programs that claim to generically remove private or compromising information. Geiger reviewed three such tools in December 2004 and found that all of these programs left significant amounts of information on the computer's hard drive that they claimed to sanitize. He concludes that reviews written in magazines and by users of these programs are based on product claims and feature comparisons, rather than on an actual evaluation of the program's abilities. He is conducting further research in this area.[Gei04]

Crescenzo *et al.* discuss the need to scrub a computer's storage after a cryptographic key is no longer needed and present a theoretical model for determining when media is sanitized.[CFIJ99]

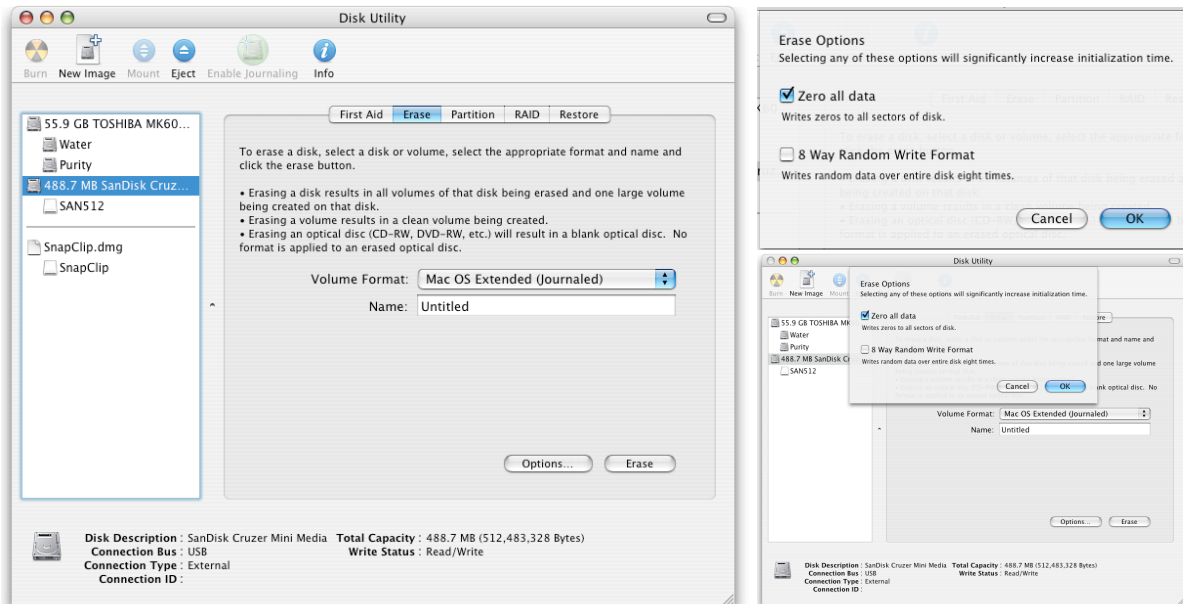


Figure 2-29: The Apple MacOS 10.3 Disk Utility (left) has an “Options” button which, when pressed, displays a subpanel (upper right) gives the user the option to overwrite the disk with zeros or eight passes of random data. The subpanel is displayed on the main window as a “Sheet” (lower right). Considering how much empty space is in the Disk Utility window, it probably would have made more sense—and been more usable—to display the options on the main panel and to have removed the “Options” button.

## 2.6 A Brief Survey of Regulatory and Other Non-Technical Approaches

There is a long history of both successful and unsuccessful attempts to control computer systems through the use of *defacto* or *dejure* regulations. Although some argue that it is inappropriate to have government regulation on technology because the market can solve technological problems faster and more effectively than the heavy hand of government, others argue that markets frequently fail to make optimum solutions when it comes to detailed technological issues.

Morgan and Newton argue for a middle ground, stating that “market-based standards are not relevant to most issues involving information technologies.”[MN04] In part, they write, this is because markets are simply not equipped to deal with the nuances and long-term implications of technological decisions. Instead, markets tend to focus on short-term results.

Of course, regulation can itself be ineffectual or produce unintended consequences. It is widely acknowledged that unsolicited email sent over the Internet did not stop after Congress passed the CAN-SPAM act.[CAN03] Likewise, an unfortunate byproduct of the regulations prohibiting the export of software containing strong cryptographic algorithms in the 1980s was that less software was made available inside the US that implemented those algorithms.[Koo99] Thus, any regulatory proposal should carefully consider the potential for both intended and unintended consequences. It is also useful to consider voluntary “regulations” that are backed by some kind of industry or even moral authority, rather than by the force of law.

To that end, Morgan and Newton describe a series of escalating approaches for the adoption of design principles to regulate technology so that it can achieve socially relevant ends:

1. **Best professional practice**, adopted by professional societies.
2. **Certification** of systems, attesting that the systems conform to the design principles.
3. **Acquisition specifications** used by purchasers to decide which products are considered for purchase and which are rejected.
4. **Legal frameworks** built upon proven best professional practice and certification standards.  
[MN04]

The patterns introduced in this thesis fit into the Morgan/Newton framework as well-developed best professional practices, and as templates for legal frameworks.

The remainder of this section explores a variety of past regulatory efforts, with specific attention to the regulation of drugs and warning labels.

### 2.6.1 Fair Information Practice

After nearly a decade’s worth of public disclosures and congressional hearings about the increased use of consumer databanks, the U.S. Department of Health, Education and Welfare issued a report in 1973 about the impact of databanks in American society.[UDoHoAPDS73] At the time, the consumer reporting industry was in the middle of a transition from manual records to computerized records. Some people believed that the federal government needed to adopt laws and regulations that would guarantee the right of individuals to access information about them stored in the databanks of American businesses. As a result of these concerns, the report’s authors recommended that a Code of Fair Information Practice be adopted. (See Figure 2-30.)

“The Code of Fair Information Practice is based on five principles:

1. There must be no personal-data record-keeping systems whose very existence is a secret.
2. There must be a way for a person to find out what information about the person is in a record and how it is used.
3. There must be a way for a person to prevent information about the person that was obtained for one purpose from being used or made available for other purposes without the person’s consent.
4. There must be a way for a person to correct or amend a record of identifiable information about the person.
5. Any organization creating, maintaining, using, or disseminating records of identifiable personal data must ensure the reliability of the data for their intended use and must take reasonable precautions to prevent misuse of the data.”

Figure 2-30: The Code of Fair Information Practice. [UDoHoAPDS73]

While the Code was not adopted in the United States, it became the basis for more than 30 years of privacy regulation in Canada, Europe and Asia.

The 1973 Code of Fair Information Practice can be applied to the many HCI-SEC issues involving the recording and display of sensitive information in computer systems. Much of the work in this thesis on the topic of sanitization (Chapters 3 and 4) and covert monitoring (Chapter 8) is based on a direct application of these principles to desktop computer systems.

### 2.6.2 Product labeling as a precedent for software labeling

Cranor suggests that it might be useful to take food nutrition labels as a starting point for the design of any privacy labeling system.[CAG02] This section is based on that suggestion.

There are in fact many similarities between the way that drugs work on the human body and the way that software works on computer systems. Both are made by individuals or organizations that are separated in space and time from the user. Both have stated actions and side-effects. And both have actions that are fundamentally unpredictable: while the user usually knows the trademark or brand name of *what* was consumed and what claims were made, the drug or software may or may not have these effects. There may be undisclosed parts or ingredients. There may also be unexperienced consequences as well.

#### The Pure Food and Drug Act of 1906

By the end of the 19th Century, consumers in the United States faced a serious problem: many foods, tonics and drugs contained significant quantities of addictive substances such as codeine or cocaine. Often these substances were placed in the foods specifically for the purpose of addicting consumers, so that consumers would have unexplained cravings for the products and continue to purchase them. Addictive drugs such as morphine, heroin, opium and laudanum were even put into “soothing syrups” designed to help babies cope with the pain of teething. (See Figure 2-31.)

After much public outcry, Congress passed The Pure Food and Drug Act of 1906 to deal with the



Figure 2-31: “Soothing syrups” containing morphine, heroin, opium, or laudanum (a mixture of alcohol and opium) were packaged for babies to stop their crying at the turn of the 20th Century. The Pure Food and Drug Act of 1906 required that the names of the narcotics and their dose be indicated on product labels. Once the information was made public, newspapers, magazines, and the American Medical Association could begin the fight against these so-called “soothers.” Image c.1910 from [oM98].

problem of food and drug adulteration. The 1906 Act did not outlaw the addition of addictive substances to foods or tonics: it simply mandated that any food or drug containing specific addictive drugs—such as alcohol, codeine, or cannabis—disclose the presence of those drugs on the package label. The words “may be habit forming” also had to be prominently displayed.

The Act also required that labels explicitly mention any artificial colors and flavors. After the law’s passage, drinks couldn’t be sold as “orange soda” unless that drink had flavoring that came from genuine oranges. Otherwise the drink had to be labeled with the words “imitation” or “artificial.”

The Act required that every bottle, box, and bag of food be clearly labeled to indicate the precise weight of the food that it contained.

In the case of drugs, the Act further specified that consumer packaging had to specify the strength, quality, and purity of the pharmaceutical the package contained if it differed from accepted standards. The dose of the drug had to be clearly printed on the outside of the package.

Although such labeling was designed to let consumers make informed decisions, in practice the disclosure caused many manufacturers to remove the addictive substances from their products. For example, following the passage of the 1906 law, cocaine was removed from the formula for

Coca-Cola, a popular beverage of the time.

Product labels made it possible for scientists and the nascent consumer groups to rapidly collect information over a broad segment of products—far more than could have been collection through laboratory investigation, spot inspections, or litigation. This information ultimately provided lawmakers with additional evidence that was used to justify future legislation that outlawed many of the more excessive practices.

### **Food labels today**

Today the product labels of the 1906 law have been expanded to include a more complete list of ingredients and nutritional information. The intent is for consumers themselves to read these labels and make their own decisions about diet and health.

But while food labels have proved to be a boon for government regulators, food scientists and academics, there is a growing body of research that finds these labels to be ineffective in reaching the very consumers who could benefit the most from the information that they contain.

A study of 631 shoppers in Sydney by Worsley found that there was no clear consensus as to what information should appear on food labels.[Wor96] Instead, the kind of information that people thought should be present tended to break down along “gender, educational background, and other demographic characteristics.” Worse, there were sharp disagreements between consumers and experts as to what information should appear on labels so as to be useful to consumers.

What’s more, consumers seem to be poor judges as to what information would best satisfy their needs. Given the labels that they prefer, consumers actually did worse on an “information-intensive task” than given labels that experts thought would be appropriate to the task, according to a study of shoppers over 18 years old conducted by Levy, Fein and Schucker.[LFS92]

A metanalysis study of 130 nutritional labeling studies (from a universe of 307 papers) published in June 2003 by the European Heart Network found the following summary results:

- “Most people claim to look at nutrition labels often or at least sometimes. Some claim that looking at labels influences their purchases, especially for unfamiliar foods.”[p.20]
- ... but while many people say that they look at labels in self-reported studies, analysis suggests that many people “look” at the nutrition information panel but do not actually process the information.
- The most common reason the people look at labels is to avoid certain nutrients or to assess nutrient content.
- ... but people do not look at labels when they are pressed for time or when they doubt the accuracy of the label information.
- Men are less likely than women to look at labels. Women who have higher income and people who have higher levels of education are more likely to read labels.
- People on special diets or who are interested in their health are more likely to read labels than the average consumer.[EHN03]

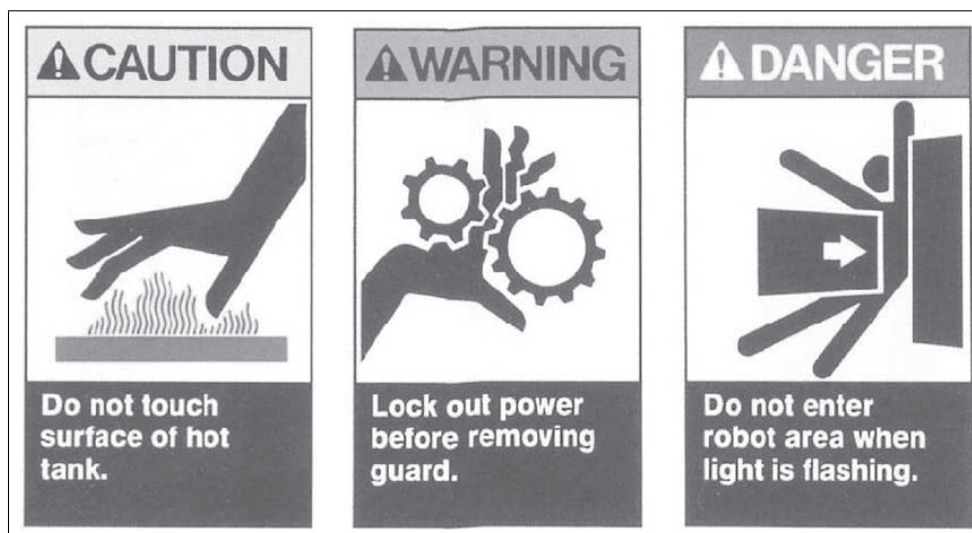


Figure 2-32: Representative warning designs used in the United States.[WCSJ02] *Used with permission*

These studies imply that labels conveying information about privacy or security aspects of software or services might be helpful for consumer activists or perhaps an elite subset of computer users, but alone they will not provide a general panacea to privacy or security problems.

On the other hand, these studies do not consider another widely recognized result of mandatory labeling: by forcing manufactures to label content that might be objectionable, such regulations frequently result in the removal of the objectionable material so that it will not have to appear on the label.

### 2.6.3 Safety and warning labels

Whitten and Tygar suggest that security warnings in consumer software applications could be informed by current research on standardized warning labels (e.g., Figure 2-32).[WT99, WT03] This section is based on that suggestion.

#### Visibility of warning labels

Wogalter and Young conducted a study of 44 college students to see if which of three presentations of warnings would achieve the highest degree of compliance. The warning was straightforward: “Glue can burn and kill skin on contact. Wear supplied gloves when using glue.”

This warning was presented to subjects in three ways: along the side of a small glue bottle (the control); along *wings* molded into the bottle’s body; and on a *tag* that was visible when looking down on the bottle. These bottle designs and the typography utilized in the study are shown in Figure 2-33.

The study found that *tag* presentation was dramatically more effective than the alternatives: 80% of those who were presented with the warning on the “tag” followed its instructions and put on supplied gloves when they were asked to use the provided glue to assemble a model airplane.

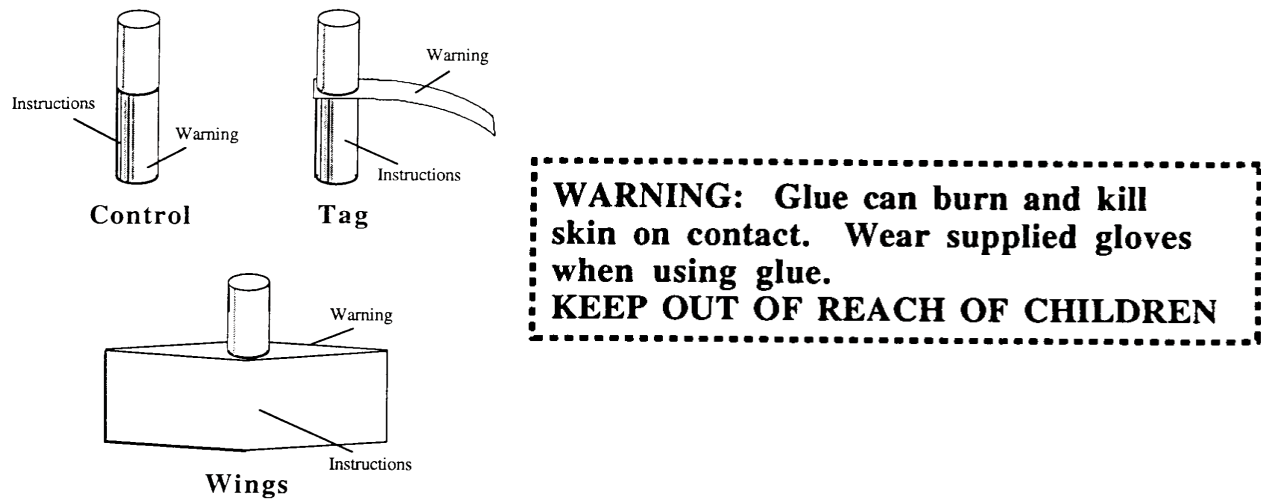


Figure 2-33: Wogalter and Young's three bottles experimented with different placement of warning labels (left). The researchers found that the "tag" style worked much better than the "Control" and "Wings" styles because subjects were forced to look at the warning as they opened the bottle. The warning is shown at the right. [WDL99] *Used with Permission.*

By contrast, only 35% of those presented with the label on wings followed the instruction, and only 13% presented with the traditional (control) label complied. Significance of these results was reported at  $\chi^2 = 14.05(2, N = 44), p < 0.001$ .

Wogalter and Young attributed the high rate of compliance to two facts. First, the tag warning was more readily seen: 100% of those in the tag group noticed the tag, compared with 50% of the wings group and 26% of the control group,  $\chi^2 = 17.39(2, N = 44), p < 0.001$ . Second, safety gloves were provided on the same table as the glue and the model airplane kit and required "little effort to don," [p.56], so the cost of compliance was relatively low. The authors indicate that this finding is in conformance with their other research on compliance, which finds that "social influence" [WFSB93], the cost of compliance, and whether or not the subject notices the warning all affect overall compliance. [WY94, p.56]

The results of this research imply that warning labels that are readily apparent to the user may have a positive impact on performance, but that compliance with warning labels will be more likely if compliance is easy. Pop-up warnings that tell the user "you are about to engage in a dangerous operation: continue?" probably won't be effective unless they give the user a low-cost alternative to the objectionable operation that will still accomplish the user's overall goal.

Wogalter, Konzola and Smith-Jackson have produced a set of guidelines for creating warnings and evaluating their use in research.[WCSJ02] Good warnings, they argue, are *salient* (they are immediately noticed and attended to); have effective *wording*; have clean *layout and placement*; and incorporate *pictures or symbols* (to increase the likelihood of being noticed, to improve memory, and so they can be used by those who are illiterate in the warning language). The wording itself consists of four message components: "(1) signal word to attract attention, (2) identification of the hazard, (3) explanation of consequences if exposed to hazard, (4) directives for avoiding the hazard." [WCSJ02, p.221]



**Product safety labels: ANSI Z535.4-2002**

Whitten and Tygar suggest that the American National Standards Institute standard ANSI Z535.4 for Product Safety Signs and Labels [Ins98] might have application to computer security, as the standard explains how to present warning information so that it is understandable by those who are relatively untrained.[WT99, WT03] Unfortunately, they did not follow this suggestion with an examination of the standard in question. Such an examination follows.

While the recommendations in Z535.4 have little to do with software, an analysis of Z535.4 for this dissertation found 15 specific recommendations in the standard that are directly applicable to the presentation of security warnings in desktop software. Those recommendations are presented in Figure 2-34.

In industry, the widespread adoption of Z535.4 by manufacturers dramatically increased the opportunities for *passive learning* because safety-critical information encountered by individuals in one context is relevant when the same symbols are used to present safety-critical information in other contexts.

It seems reasonable to suggest that software practitioners could similarly benefit from the standards recommendations that specific typography, graphic presentation, symbols, and “signal words” be used for the universal presentation of safety-critical messages.

**2.6.4 Existing information technology labels**

There is a small but growing collection of instances in which the labeling approach has been applied to information technology. A representative list appears below.

**Cranor *et al.* ’s Technology Inventory Icons**

In a report that cataloged tools available to parents for choosing or controlling online content for their children,[CRG97] Cranor *et al.* introduced six icons for describing the capabilities of the 41 tools that they evaluated:

**Suggest**

Used for web sites, printed publications, and filtering software that suggests sites for children to explore.

**Search**

Indicates a search service that can restrict its content to material that is appropriate for children.

**Inform**

Provides information about content. This includes PICS labels,[KMRT96] reviews, and other kinds of descriptive content.

**Monitor**

Records for later inspection information a list of the content accessed by the user. The record may consist of all content accessed or simply the content that is deemed inappropriate.

Section	Page	Recommendation
4.10	3	The Safety alert symbol (a equilateral triangle surrounding an exclamation mark) should only be used to alert individuals to a potential personal injury hazard; it should not be used to alert persons to property-damage-only accidents.
4.13	4	The signal words for product safety signs are DANGER, WARNING and CAUTION. “DANGER” is to be used for <i>imminent</i> hazard which, if not avoided, <i>will</i> result in <i>death or serious injury</i> . “WARNING” indicates a <i>potential</i> hazard which <i>could</i> result in <i>death or serious injury</i> . “CAUTION” indicates a <i>potential</i> hazard which <i>may</i> result in <i>minor or moderate injury</i> . “It may also be used to alert against unsafe practices.”
6.4	5	Signs should have contrasting borders to achieve distinctiveness from their background.
7.2.1	5	The word “DANGER” shall be in safety white letters on a safety red background.
7.2.2	6	The word “WARNING” shall be in safety black letters on a safety orange background.
7.2.3	6	The word “CAUTION” shall be in safety black letters on a safety yellow background.
8.1.1	6	Signal words shall be in sans serif letters in upper case only.
8.1.2	7	Message panels should be printed in a combination of upper and lower case letters.
B3.2	15	Hazard messages should come first, followed by action/avoidance messages—but only “if there is enough time to read the entire word message and still avoid the hazard.”
B3.3.1	16	Write short messages using “headline style.” For example, use “Moving parts can crush and cut” instead of “This machine has moving parts that can crush and cut.”
B3.3.2	17	Write in the active voice, rather than the passive voice.
B3.3.3	17	Avoid prepositional phrases.
B3.3.5	17	Write in outline format with bullets to enhance readability.
B3.3.6	18	Use left-aligned, ragged right text.
B3.3.10	19	Use black type on a white background or white type on a black background.

Figure 2-34: Specific recommendations found in ANSI Standard Z535.4 that are applicable to the presentation of security information in computer interfaces.

**Warn**

Provides information about content and warns the user against accessing content that is deemed inappropriate.

**Block**

Block the user from accessing information that is deemed inappropriate.

The primary use of the icons in the report are on the report's first page, as an attention-getting mechanism, and on pages 5 and 6, where the capabilities are introduced. The report does not use the icons on the pages describing the individual products that are reviewed, nor does the report recommend that standardized icons appear on products or on web sites describing the products. Nevertheless, this appears to be the first use of icons to describe generic functionality that might exist within a range of different software products.

**The Platform for Privacy Preferences Project (P3P)**

Developed under the auspices of the World Wide Web Consortium, P3P is a standard that allows web sites to publish privacy policies in machine readable form.[CDE<sup>+</sup>05] These policies can then be read either by P3P “user agents” built into web browsers such as Internet Explorer. The policies can also be used by search engines to automatically screen results—for example, so that a results page for a search of online merchandise will not display merchants who would share details of the sale with third parties, if the person making the purchase is opposed to this kind of secondary use.

**TRUSTe's “trust marks”**

TRUSTe is an independent organization dedicated to helping individuals and organizations “establish trusting relationships based on respect for personal identity and information in the evolving networked world.”[TRU04] TRUSTe is best known for its green and black seal which it licenses for use on the web sites of organizations that have a privacy policy, that agree to be audited by TRUSTe or by an outside third party, and that agree to participate in TRUSTe's dispute resolution processes.

Today TRUSTe offers five licensable seals:

- Web Privacy Seal
- Children's Privacy Seal (COPPA Safe Harbor)
- eHealth Seal
- EU Safe Harbor Seal
- Japan Privacy Seal

Each of the TRUSTe seals have specific minimum privacy standards such as allowing consumers to unsubscribe from email newsletters and to opt-out from the sharing of personally identifiable information. The seals also require that the organizations abide by specific minimum security measures and post a privacy statement which makes specific disclosures. A full list of the requirements appears at <http://www.truste.org/requirements.php>. The Web Privacy Seal trustmark is shown in Figure 2-35; other seals are similar, but with different lettering along the bottom.

TRUSTe also manages a “Bonded Sender” program. This program lists organizations that follow

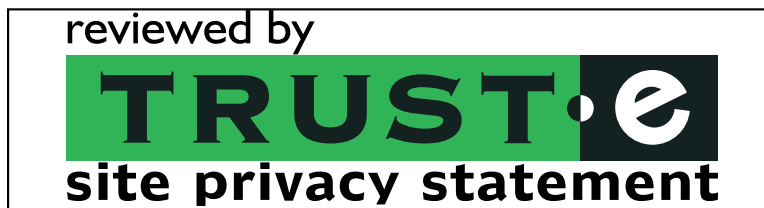


Figure 2-35: The TRUSTe “trustmark.” *Reprinted with permission.*

specific mailing guidelines and, as a result, are placed on a special whitelist so their mail is not blocked by mail filters.

When TRUSTe launched in June 1997 with the name eTRUST (the name was changed due to trademark restrictions), the organization’s original plan was to have different licensable seals called “trustmarks” that organizations could use to indicate the content of their privacy policies. Three trustmarks were proposed:

“No Exchange”	No personally identifiable information would be collected by the site.
“One-to-One”	The site would collect information, but not share it with others.
“3rd Party”	The site would both collect information and share it with others.

TRUSTe ultimately dropped its plans to use these informative icons. At the time, TRUSTe’s executive director said that the change was being made in the interest of simplicity.[Mac97] In fact, the real reason that the practice-specific icons were dropped is that there was no incentive for organizations to voluntarily license and display a mark indicating that they shared information with third parties—no matter how beneficial that sharing might actually be to the consumer.[Hod05]

The fact that TRUSTe was unable to find support for these highly informative icons in 1997 is an example of market failure—the very kind of market failure that typically justifies the need for regulation.

### EPCglobal guidelines

The Electronic Product Code (EPC) is a system that applies Radio Frequency Identification (RFID) technology to the task of supply chain tracking and supermarket check-out. Proponents of RFID describe a world where small EPC tags will be built into the packaging of consumer goods much in the way that barcodes are placed on packages today.

EPCglobal Inc. is an membership organization that oversees the development of EPC standards. The organization has adopted a set of “Guidelines on EPC for Consumer Products” which includes four key elements governing the use of radio frequency identification technology (RFID) in consumer products:

- **Consumer Notice.** Consumers must be given notice that a product contains an EPC tag that is embedded or in the product’s packaging. Notice is given through the use of the licensed EPC logo, the use of which is tightly controlled by EPCglobal.

- **Consumer Choice.** Consumers must be told that they are allowed to disable or discard the EPC tags that they receive.
- **Consumer Education.** Consumers must have the opportunity to obtain information about EPC tags.
- **Record Use, Retention and Security.** “Companies will publish, in compliance with all applicable laws, information on their policies regarding the retention, use and protection of any personally identifiable information associated with EPC use.”[EPC05]

These guidelines fall short of the RFID Bill of Rights discussed in Section 8.4. For example, the “Consumer Choice” principle says that consumers are allowed to disable or remove the EPC tag—but what if removing the tag voids the product’s warrantee? On that topic the guidelines are silent.

The second problem with the guidelines is that they lack any enforcement power. There is nothing to prevent a manufacturer from using EPC technology without abiding by the guidelines. Although such a manufacturer might be prohibited from using the EPC logo on their product, the manufacturer might not be concerned.

Nevertheless, the EPCglobal guidelines are a significant first step in an industry that has generally shied away from many other kinds of disclosure requirements. It will be interesting to see if this effort is successful.

### **Hosmer’s attack icons**

Hosmer proposed that icons could be used for visualizing risks and attack scenarios. Using icons, Hosmer argued, allows for “rapid comprehension and presentation of information security” in a variety of environments. “Visual attack scenarios help defenders see system ambiguities, imprecision, vulnerabilities and omissions, thus speeding up risk analysis, requirements gathering, safeguard selection, cryptographic protocol analysis, and INFOSEC training.”[Hos00]

In her paper, Hosmer presents more than 50 icons and rules for combining the icons to graphically depict scenarios. Although it is unlikely that the kinds of icons that Hosmer presents would be part of any mandatory labeling regime—it is clearly unreasonable to expect software pirates to label their warez sites with “piracy” icons—Hosmer’s work shows that reasonable icons can be used to rapidly convey a variety of security-critical events.

### **Williams’ software ingredients and software facts**

Jeff Williams, the CEO of Aspect Security, a Columbia Maryland consulting firm, has suggested that software vendors adopt literal software labels showing the ingredients and the results of automated threat analysis.[Wil05] An example of the figures from Williams’ presentation appear in Figures 2-37 and 2-38.

Drawing upon analogies from automobile safety and food labeling, Williams argues that today’s software is unsafe because of hidden internal failures. Arguing that manual auditing code is very difficult, Williams’ firm is currently developing a system that will issue these labels for any Java application that is uploaded. His labels are intentionally designed to resemble food nutrition labels, Williams says, because most people are familiar with food labels and are immediately curious as

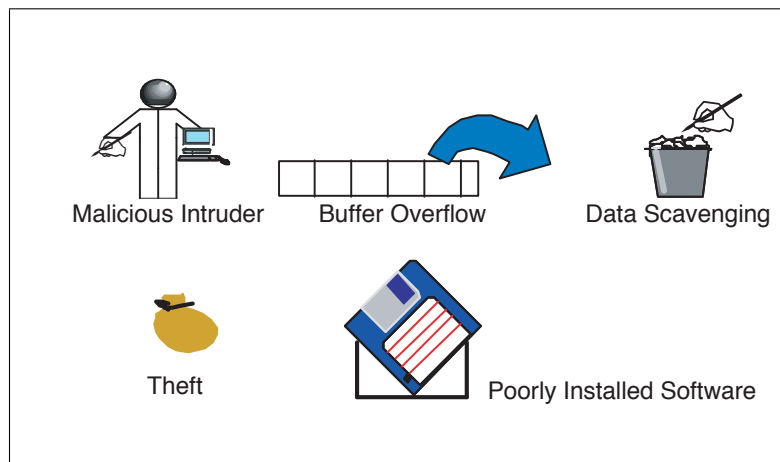


Figure 2-36: Hosmer's visual attack scenarios.[Hos00]

to how the concept could be applied to software. It is unclear whether or not this project will be successful.

### 2.6.5 Regulations addressing the data sanitization problem

Although no regulations have mandated that computer manufacturers provide systems that are easier to sanitize and easier to verify when sanitization has not occurred, a number of regulations have been passed that nevertheless mandate that sanitization must take place.

#### NSA IAM Section 15

Presidential Decision Directive 63 (PDD 63) signed by President Clinton on May 22, 1998, outlined civilian and governmental responsibility to protect the US Critical Infrastructure and established the framework of the National Infrastructure Assurance Plan. Under this plan, the National Security Agency (NSA) was mandated to perform information security assessments of US Government Systems. This assessment has since been standardized as the NSA Infosec Assessment Methodology (IAM). Section 15 of the NSA IAM discusses media sanitization and disposal.

As many organizations are now training individuals in the NSA IAM, it is likely that there are a growing number of computer security consultants and practitioners who are aware of the data sanitization issue.

#### BS 7799 and ISO 17799

On December 1, 2000, the International Standards Organization adopted ISO/IEC 17799:2000(E), "Information technology—Code of practice for information security management." (ISO 17799 is based on and supersedes BS 7799, which was passed in 1995.)

Section 7.2.6 of the standard addresses the issue of media sanitization prior to disposal:

#### ***"7.2.6 Secure disposal or re-use of equipment"***

Information can be compromised through careless disposal or re-use of equipment (see

<b>Ingredients:</b> Sun Java 1.5 runtime, Sun J2EE 1.2.2, Jakarta log4j 1.5, Jakarta Commons 2.1, Jakarta Struts 2.0, Harold XOM 1.1rc4, Hunter JDOMv1
--

Figure 2-37: A “software ingredients” label developed by Jeff Williams; *used with permission*.

Software Facts			
Typical Roles per Instance		4	
Expected Number of Users		15	
Amount Per Serving			
Modules		155	Modules from Libraries 120
% Vulnerability*			
Cross Site Scripting		22	65%
Reflected	12		15%
Stored	10		
SQL Injection		2	10%
Buffer Overflow		5	95%
Total Security Mechanisms		3	10%
Modularity		.035	0%
Cyclomatic Complexity		323	
Encryption		3	
Authentication		15	4%
Access Control		3	2%
Input Validation		233	20%
Logging		33	4%
* % Vulnerability values are based on typical use scenarios for this product. Your Vulnerability values may be higher or lower depending on your software security needs:			
	Usage	Intranet	Internet
Cross Site Scripting	Less Than	10	5
Reflected	Less Than	10	5
Stored	Less Than	10	5
SQL Injection	Less Than	20	2
Buffer Overflow	Less Than	20	2
Security Mechanisms		10	14
Encryption		3	15

Figure 2-38: A “software facts” label developed by Jeff Williams and used in his PowerPoint presentation to argue that programs should be given “software facts” labels in a manner similar to the way that foods are given nutritional labels today. *Used with permission*.

also 8.6.4)<sup>5</sup>. Storage devices containing sensitive information should be physically destroyed or securely overwritten rather than using the standard delete function.

“All items of equipment containing storage media, e.g. fixed hard disks, should be checked to ensure that any sensitive data and licensed software have been removed or overwritten prior to disposal. Damaged storage devices containing sensitive data may require a risk assessment to determine if the items should be destroyed, repaired or discarded.”[ISO00]

Unfortunately, the distribution of ISO standards are tightly controlled by the copyright holder and are extraordinarily expensive to purchase. For example, the web site [www.standardsdirect.org](http://www.standardsdirect.org) sells ISO 17799 as a downloadable PDF for £110 (approximately \$209). Nevertheless, the fact that organizations can be certified to be ISO 17799 compliant has created a growth industry in ISO 17799 training and certification courses. In March 2005 a Google search found 179,000

<sup>5</sup>Section 8.6.4, “Security of system documentation,” has nothing to do with sanitization. This section states that system documentation should be stored securely, protected from unauthorized access, and restricted to the minimum possible number of authorized individuals. The validity of such advice will not be considered in this thesis.

web pages that included the term “ISO 17799,” of which 35,200 specifically addressed the issue of ISO 17799 certification and compliance.

### **VISA's Cardholder Information Security Program (CISP)**

Since June 2001, merchants that accepted VISA cards and service providers that perform payment card processing have been required to follow VISA's 12-point Cardholder Information Security Program (CISP). [VIS05] In December 2004, the VISA CISP standard was folded into the Payment Card Industry Data Security Standard. [U.S04]

Although these standards apply to all merchants and processors, different levels of security are required for merchants of different sizes. Key elements that apply to all merchants are standards that protect the merchant's network, cardholder data, the institution of a vulnerability management program, access controls, requirements to monitor and test the network, and maintenance of an information security policy.

In conducting the Traceback study, disk #21 was determined to come from a major supermarket firm. The disk, which contained 3,722 credit card numbers, was removed from service in May 1999. The disk was acquired on November 11, 2000 and included the notation “Pulled from working system and tested good.”

In discussions with the senior manager at the company responsible for information security at the company, it was learned that the firm had adopted a data sanitization process as a result of the VISA CISP requirement. According to the manager, in 2000 the company had just started its data sanitization process and it is possible that some drives fell through the cracks. The company now wipes all of its hard drives with Norton Disk Wipe and it has a forensics department which, among other things, samples the wiped drives to make sure that they are actually wiped. “It really is a problem, asset disposal. The assets have little or no value by the time they depreciate. From an accounting perspective, no one cares. But the value of the data on these disks is really, really high. It just has to be managed.”[Gar04b]

### **Federal regulations on consumer information and records disposal**

The Fair and Accurate Credit Transactions Act of 2003 (FACTA) amended the Fair Credit Reporting Act to require that “any person that maintains or otherwise possesses consumer information, or any compilation of consumer information, derived from consumer reports for a business purpose” to “properly dispose of any such information or compilation.”[US03, §216, 15 U.S.C. 1681 w(a)(1)] On November 18, 2004, the Federal Trade Commission issued its Final Rule implementing the requirements of the FACTA.[Com04b] The Securities and Exchange Commission issued its own final rule on the “Disposal of Consumer Information” three weeks later on December 8.[SC04] Other Federal bodies charged with regulating portions of the nation's financial industry, including the Federal Reserve Board, the Office of the Comptroller of the Currency, the Federal Deposit Insurance Corporation, the Office of Thrift Supervision, and the National Credit Union Administration, have adopted consistent and comparable rules.

Designed to help combat the growing tide of identity theft, these rules cover a broad range of businesses and financial institutions in the United States that have sensitive consumer information on their computers. For example, the FTC Rule covers not only consumer reporting agencies, but



also “lenders, insurers, employers, landlords, mortgage brokers, car dealers, and other businesses that use consumer reports.”[Sot05]

Organizations collecting “consumer reports” are now required to properly dispose of that information “by taking reasonable measures to protect against unauthorized access to or use of the information in connection with its disposal.” The law specifically considers “abandonment ... as well as the sale, donation, or transfer” as forms of disposal. The term “consumer reports” is broadly defined in the law to include pretty much any personally identifiable information that could be used to make a decision in granting credit or insurance.

The rules apply both to paper and electronic records. While the rules do not specify what constitutes reasonable measures, they give examples. For paper records the FTC Rule notes that generally appropriate measures would include shredding or burning and parenthetically notes that a paper shredders are “available at office supply stores for as little as \$25.” For electronic records, the FTC notes that a “small entity” could comply with the disposal rule by a variety of means:

“If a small entity has stored consumer information on electronic media (for example, computer discs or hard drives), disposal of such media could be accomplished by a small entity at almost no cost by simply smashing the material with a hammer. In some cases, appropriate disposal of electronic media might also be accomplished by overwriting or ‘wiping’ the data prior to disposal. Utilities to accomplish such wiping are widely available for under \$25; indeed, some such tools are available for download on the Internet at no cost. Whether ‘wiping,’ as opposed to destruction, of electronic media is reasonable, as well as the adequacy of particular utilities to accomplish that ‘wiping,’ will depend upon the circumstances.”[Com04a, p.30]

According to the FTC, the Rule covers far more than just a person’s name and social security number, but also includes driver’s license numbers, phone numbers, physical addresses, and e-mail addresses. Significantly, the Rule also covers so-called “credit header” information—the portion of a credit report that does not actually have any credit information. It even covers information from public records, although the Commission noted that businesses may consider the sensitivity of consumer information when determining what sort of disposal methods should be used.

In its report, the FTC wrote that businesses would need to educate and train employees on how to properly dispose of paper and electronic records. But despite the requirements for new training and the purchase of paper shredders, the FTC noted that most of the organizations filing comments “stated that the proposed Rule would not create any undue burden for small businesses.”

All businesses that maintain consumer reports must comply with the FTC rule on June 1, 2005. Compliance for the SEC rule starts July 1, 2005.

### 2.6.6 Regulating accessibility with Section 508

The user interface of a surprising number of software, web sites, and telecommunications devices came under *de facto* Federal regulation in June 2001 when Section 508 of the Workforce Investment Act of 1998 (29 U. S. C. (SS) 794.d) came into effect. The law contained wide-ranging standards mandating that information technology be usable, where possible, by individuals with a variety

of disabilities. For example, Section 508 requires that the functionality of operating systems like Windows and MacOS be *accessible* without the use of a mouse or other pointing device because many people lack the manual dexterity or vision to use such devices properly. Likewise, Section 508 requires that web sites be accessible by someone who cannot read text that is embedded in downloaded images—as is the case for a blind person attempting to navigate a web site with a screen reader.

Making products accessible to those with disabilities is not merely a question of legality, compliance and markets. Many individuals view making systems accessible for those who are less fortunate as “morally the right thing to do.”[TR03]

Nevertheless, prior to the passage of Section 508, there was little or no support for screen readers in Microsoft Windows or for using Macintosh computers without a mouse. Thus, it seems that the moral argument needed the legal requirement to become a powerful driving force. There is also the issue of competitive pressure: Once support for accessibility moves forward on one platform, other vendors feel compelled to work harder.

Regulations such as Section 508 can have far-reaching impact because they affect not only end-user applications but also the tools that are used to create applications and the instruction of future application developers. Ludi reports that accessibility APIs were added to Java, Macromedia Shockwave, Flash, and Adobe Acrobat Reader only after the passage of Section 508. The popular Dreamweaver web site authoring tool was modified to do Section 508 compliance checking. Students in Ludi’s course “do seem to get the message, at least in the short term,” that it is important to design web sites for equal access by all. [Lud02]

Of course, it may be that Section 508 is not the cause of these changes, but instead reflects a growing awareness within our society of the need to design products so that they can be used by the disabled. It is impossible to say for sure whether the passage of Section 508 was the cause of these changes. But many people in the industry have written that they believe Section 508 is causative.

For example, the task of creating Section 508-compliant software was eased significantly over the past few years by the inclusion of new functionality within systems such as the Java Swing and TrollTech’s Qt[Tro05] application toolkits. Although Section 508 may not be the reason that the accessibility functionality was added to these systems, it is almost certainly the reason that the functionality has been widely used.

As a result of Section 508, much of the commercial software sold in the United States can now be readily used by people who have significant disabilities, whereas a decade ago this was not the case.

### **Procurement mandates**

Section 508 is a procurement law: its sole power comes from its prohibition on the Federal Government from purchasing technology for information services that do not meet its accessibility requirements.<sup>6</sup> Given that the Federal Government is the largest purchaser of information tech-

---

<sup>6</sup>Like many Federal regulations, the law includes a system for requesting waivers and obtaining practical exclusions.

nology in the United States—accounting for 10% of all information technology expenditures—few manufacturers are willing to give up this market.

Artman suggests that mandating specific usability requirements in law and regulation is more effective than delegating this function to contract officers. That's because contract officers frequently have little or no training in these issues. "If the contract does not contain explicit requirements for usability, it is generally one of the first considerations to be cut if time or finances are constrained." [Art02]

### What Section 508 covers

Previous attempts at using federal regulation to force the industry to comply with accessibility guidelines were less successful than Section 508. For example, Section 504 of the 1973 Rehabilitation Act had mandated that those with disability receive equal treatment—for example, that they have an equal opportunity for a full education—and Title II of the Americans with Disabilities Act (ADA) of 1990 required that people with disabilities have the same access to communications technologies as those without. But neither law provides clear and specific guidelines regarding how which barriers should be addressed and how. [OR04] Section 508 does, as evidenced by the standards shown in Figure 2-39

Corporations are free to create two different versions of their products: one for people who have disabilities and one for people who do not. But economics of software makes this approach less attractive. Once a product is adapted for use by those with disabilities with functionality that can be switched on or off, there is only a tiny incremental cost associated with putting that functionality into all versions of the company's product.<sup>7</sup> This tiny cost is invariably less than the cost of maintaining two separate product lines.

### Universal design

It is generally acknowledged that the beneficiaries of accessible design go far beyond the community originally targeted. For example, the keyboard controls built in to the Windows operating system are essential for those who cannot use a mouse, but they are also useful for "walk-about" mobile computing when no mouse is available, or for when the computer's mouse breaks. "Goods designed inclusively for all people inevitably lead to products and services that benefit not only the original target markets but other, mass markets as well." [Mar03]

To those who work in the field of accessibility, designing a product so that it can be used by either those with or without a disability is called *universal design*.

Coombs says that universal design can have immediate and far-reaching positive effects on a much broader population than was originally intended: "Curb cuts were made to assist people in wheelchairs, but they brought immediate benefits to people riding bicycles, pushing baby carriages, and so forth... Accessible Web design is the equivalent of electronic curb cuts. Everybody benefits." [OR04]

---

<sup>7</sup>The cost is not zero because the disability adaption must be tested and can result in technical support costs when users accidentally turn the feature on and do not know how to turn it off.

Section 508 requires that technology purchased by the federal government meet 16 standards of accessibility:

1. Usable by a person without vision
2. Usable by a person with low vision without relying on audio.
3. Usable by a person with little or no color perception.
4. Usable by a person without hearing
5. Usable by a person with limited hearing—for example, by providing audio amplification.
6. Usable by a person with limited manual dexterity, reach, and/or strength.
7. Usable without time-dependent controls or displays.
8. Usable without speech
9. Usable by a person with limited cognitive or memory ability.
10. Usable by a person with language or learning disabilities.
11. Availability of audio cutoff—Systems that deliver speech output must provide a mechanism for private listening or a mechanism for interrupting the speech.
12. Prevention of visually induced seizures—systems that flash must use rates of 3Hz or lower or 60Hz or higher to avoid inducing seizures in people with photosensitive epilepsy.
13. Bypass of biometric identification or activation systems—because biometrics invariably require existence or use of a piece of the body that not everybody has.
14. Usable with upper extremity prosthetics—systems that rely on capacitive sensing of the human body should be replaced by those which rely on pressure.
15. Compatibility with hearing aids, through magnetic wireless coupling, for example.
16. Usable from a wheelchair or similar mobility device.

Figure 2-39: Specific requirements for access in Section 508.

### Evaluating the impact of Section 508

Despite the incredibly wide-ranging impact that Section 508 has obviously had on the computer and telecommunications industries in the United States, there has been some disagreement on just how successful the measure has been.

For example, Podevin reported in December 2004 that 94% of the Fortune 100 web sites did not have “fully accessible home pages.” Specifically, 20% were found to have one Section 508 barrier, 17% were found to have 2 barriers, and a whopping 54% were found to have 3 or more barriers.[Loi04] The 17% success rate is actually lower than the 20% success rate found by Zaphiris and Zacharia in an analysis of 30,000 Cypriot Web sites—a set of web sites which would not be covered by Section 508.[ZZ01] In another report of failed Section 508 compliance, Zaphiris and Ellis report that only 30% of the top-50 US university web sites pass the usability requirements of the popular “Bobby” automated accessibility checker.[ZE01, Wat05]

```

```

Figure 2-40: An HTML tag that is not in compliance with the Bobby automated accessibility checker.[Wat05] A screen reader might read this HTML element as “STAR DOT GIF.”

```

```

Figure 2-41: An HTML tag that is in compliance with the Bobby automated accessibility checker.[Wat05] A screen reader might read this HTML element as “ORNAMENTAL STAR IMAGE NUMBER FIVE FOUR ONE TWO; PLEASE IGNORE!”

But there is a problem in basing an analysis of Section 508’s success solely on scores from an automated checker. While checkers like Bobby make it very easy for researchers to rapidly scan many web sites and get an accessibility score, changes in Bobby scores over time may not accurately capture the impact of Section 508 on its target population. This is because the Bobby score reports literal conformance with specific HTML coding standards—it does not actually measure the usability of web sites by users with disabilities.

For example, Bobby will declare a web site to be in violation of Section 508 if that web site has a single ornamental image that lacks a textual ALT tag, as shown in Figure 2-40. But Bobby will happily rate this the tag shown in Figure 2-41 as being in compliance with Section 508. For a blind person using a screen reader, the first HTML form is far more usable than the second. [TR03]

Yet another problem with using the web site Bobby ratings as the sole tool for judging the effectiveness of Section 508 is that the web is a moving target. Hackett *et al.* sampled 40 web sites from the years 1997 through 2002 using the Internet Archive’s Wayback Machine. They discovered that even though the absolute number of accessibility violations increased, the percentage of violations compared to the number of *potential violations* significant decreased—dropping from a 63% in 2000 to 41.7% in 2002. “This either suggests that some Web designers are becoming aware of accessibility guidelines or that general ‘good practice’ in Web design happens to include elements that also increase accessibility,” the authors conclude. [HPZ04] A more simplistic study that focused on Bobby ratings alone would have yielded the reverse conclusion.

Finally, focusing on web sites, while easy, ignores the significant investment made by US businesses in making desktop applications accessible.

Laura Ruby, program manager of Microsoft’s Accessible Technology Group, writes that Section 508 was criticized early on for having vague regulations that would lead to numerous lawsuits. “Today, two years after Section 508 was implemented, it looks as though the critics were wrong. By offering the technology industry a carrot instead of a stick, Section 508 set the stage for a proactive public/private partnership. Technology companies rushed to collaborate with government officials.” [Rub03]

### 2.6.7 Previous work on vocabulary as a barrier to usability and understanding

It is impossible for any regulatory effort to succeed without an agreement on underlying vocabulary. Section 8.2 of this thesis goes further, arguing that the confusion over vocabulary is an important factor in the current conflict between security and usability.

For an illuminating example of how confusion over basic vocabulary can contribute to failure, consider Artman's ethnographic study of a web-based application developed for a Swedish organization by the Swedish office of a US firm. Having taken a training course on usability issues, the procurement officer wrote language into the contract specifying that she should be able to review the "design" of the system's prototype. But the procurement officer and the contractor used the word "design" to mean different things:

- The **procurement officer** thought that the term "design" referred to the application's overall functional requirements, the flow of information inside the application, and interactive elements on the application's screens.
- The **contractor** thought that the term "design" referred solely to the application's "aesthetic values"—specifically the design of the application's screens. [Art02, p.68]

This fundamental confusion over a single word, *design*, disrupted the entire project's attempt at usability engineering. When the procurement officer requested paper prototypes, the contractor responded by having his art department create finished screen designs and then printed them out. The contractor thought that the request was unreasonable, given the current status of the project. The procurement officer never showed these designs to users—necessary for "user-centered design"—because they looked like finished pieces of work. And the procurement officer never went back and demanded documents about functional requirements and data flow, because the contractor had already fulfilled the requirement to present a "design."

Words are the primary tool that humans use to convey information and concepts. But words can be ambiguous or otherwise imprecise: In some cases a pair of words have the same meaning (e.g., *two* and *a couple*), while in other cases a pair of words can have meanings that are similar but subtly different (e.g., *heavy* and *weighty*). Since many words have multiple meanings (e.g. *white*), multiple readers of a document may walk away with meanings that are significantly different.

#### Technobabble

Barry explored the question of linguistic confusion in high tech in his 1991 book *Technobabble* [Bar91]. While humorous and somewhat dated, this volume nevertheless remains one of the best discussions of linguistic challenges in high tech. More than other areas of human endeavor, Barry asserts that the computing field lends itself to the rapid proliferation of new and inconsistent terminology as nouns are turned to verbs, verbs are turned to nouns, acronyms are turned to words, and so on. Perhaps this is just an excellent example of modularity and object re-use, but it is tremendously confusing to people who are not intimately familiar with the technology under discussion.

#### Academic IT babble

Confusion over vocabulary isn't just a problem for industry: it affects academia as well. Alter's 89-page article "Same Words, Different Meanings: Are Basic IS/IT Concepts Our Self-Imposed Tower of Babel" explores how different articles in academic research on Information Technology are

systematically using the same words to mean different things. The genesis of the article was a series of letters exchanged between Jim Sutter and Lorne Olfman in *Communications of the Association for Information Systems* arguing whether or not there was “too much user participation in IS projects.” Writes Altner:

“When I first glanced at Sutter’s letter my immediate response was disbelief since ‘anyone knows that user involvement is important and beneficial.’ Then I took another look and realized that Sutter’s users were functional area managers and their representatives, people with enough clout to become involved in discussions of technical IT strategy whether or not they had much knowledge to contribute. These are people CIOs and high level IT managers view as ‘their users’ but these aren’t the people I usually think of as users, namely, people who use information systems directly.”[Alt00, p.4]

Alter’s article goes on to review 10 articles published in *CAIS* between June and December 1999 and shows that the terms “System,” “User,” “Stakeholder,” “IS project,” “Implementation,” “Reengineering,” “Requirements,” “Solution,” and “Point of reference” have radically different meanings. He argues that by not standardizing terminology, it is hard to be rigorous as different concepts mean different things to different people. Imprecise terminology causes people to become confused, to misunderstand what others are arguing, and, ultimately, hampers the course of progress.

### **Standards babble**

Söderström comments on the same problem, arguing that the word “standard” has taken on so many meanings that it is no longer possible to understand what people are about when they use the word without qualification. A standard, Söderström notes, may be a specification, a recommendation, a framework, a pattern, or, in fact, a standard. Sometimes “standards bodies” create standards, but sometimes they create other things. And not all standards are created by standards bodys! [SÖ2]

Söderström gives and then mocks the European Software Institute’s definition of the word “standard.” A standard, she writes, is “a technical specification approved by a recognized standards body for repeated or continuous application, compliance with which is not compulsory.” That is, a standard is something that an organization *chooses to follow* because it has a choice not to follow. Thus, an organization can only standardize on Microsoft Windows if its employees might realistically have the option of running MacOS or Linux. If Windows is the only possible operating system to use, then there is no need to standardize on it!

The problem with this definition, notes Söderström, is that it makes the standard something that are in the eye of the beholder. One organization might standardize on Windows, but other organizations might simply use windows because it does not have a choice.

### **Why are programmers lax with vocabulary?**

Cooper hypothesizes that programmers are particularly bad at choosing appropriate vocabulary for user-facing applications because words are inherently less precise than source code:

“When the words are fuzzy, the programmers reflexively retreat to the most precise method of articulation available: source code. Although there is nothing more precise

than code, there is also nothing more permanent or resistant to change. So the situation frequently crops up where nomenclature confusion drives programmers to begin coding prematurely, and that code becomes the de facto design, regardless of its appropriateness or correctness.” [Coo99, p.186]

Another reason that programmers and mathematicians have difficulties in choosing a consistent vocabulary may be that their profession and training teaches them to work with interchangeable labels that stand for underlying values or concepts. Examples of such labels are variables used within a program or a mathematical expression. In this context, it is easy to think of words as just another set of interchangeable labels; as long as the underlying concept is the same, the actual word that is used may be considered to be immaterial.

### 2.6.8 Lessons from the prior work on regulation

This review seems to imply that regulation could be a tool that could be used to promote features that simultaneously increase security and usability in consumer software. Based on this analysis, the kinds of regulations that are likely to be the most successful are those that:

- Mandate specific principles and implementation goals, rather than the use of specific technologies and approaches;
- Emphasize the labeling and disclosure of objectionable functionality, rather than attempting to force its removal;
- Create typographical and linguistic standards for the presentation of security-critical information.

We shall return to the question of using regulatory practices to align usability and security in Chapter 8.

## 2.7 Conclusion

Like many other areas of computer science research, there is not a particularly good track record on the transitioning of HCI-SEC research from the laboratory to practice. Likewise, even when techniques for aligning usability and security have been developed and deployed in one application, these techniques have generally not migrated to other products or systems in the way that other good ideas have spread in the computer industry.

This thesis holds that one of the key factors limiting the diffusion of HCI-SEC practice is that good HCI-SEC techniques have not been systematically identified and discussed. A second gating factor has been the willingness to accept long-established models, mechanisms and designs for basic functionality provided by operating systems and application programs, rather than redesigning these systems so that they are more consistent with user expectations and can do a better job supporting actual user needs.