Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science

Proposal for Thesis Research in Partial Fulfillment Of the Requirements for the Degree of Doctor of Philosophy

Title: Usable Security: Design Principles for Creating Systems that are Simultaneously Usable and Secure

Submitted by:	Simson L. Garfinkel 634 Pleasant Street Belmont, MA 02478	_____ (Signature of Author)
Date of Submission		October 15, 2003
Expected Date of Completion:		September 1, 2004
Laboratory where thesis will be done		CSAIL

Brief Statement of the Problem: Usability and security are widely seen as two antagonistic design goals for complex computer systems. This thesis argues that conventional wisdom is wrong: for the majority of users and applications, increased security cannot be achieved with technology that decreases usability. This thesis aims to develop a set of design principles for creating and evaluating security systems — principles that, when followed, simultaneously provide for increased security and increased usability. These principles will be used to analyize a significant number of security-related applications that have been deployed in recent years, and will be applied to several areas of active concern in the computer security community, including the leakage of confidential information on repurposed digital storage media, secure messaging, and firewall configuration.

Massachusetts Institute of Technology

Department of Electrical Engineering and Computer Science

Cambridge, Massachusetts 02139

Doctoral Thesis Supervision Agreement

To: Department Graduate Committee
From: David D. Clark, Thesis Supervisor

The program outlined in the proposal:

Title:	Usable Security: Design Principles for Creating Systems that are Simultaneously Usable and Secure
Author:	Simson L. Garfinkel
Date:	October 15, 2003

is adequate for a Doctoral thesis. I believe that appropriate readers for this thesis would be:

Reader 1:	Ronald Rivest
Reader 2:	Daniel J. Weitzner

Facilities and Support for the research outlined in the proposal are available. I am willing to supervise the research and evaluate the thesis report.

Signed: _____

David D. Clark
Title: Senior Research Scientist, MIT CSAIL

Date: _____

Comments: _____

Massachusetts Institute of Technology

Department of Electrical Engineering and Computer Science

Cambridge, Massachusetts 02139

Doctoral Thesis Supervision Agreement

To: Department Graduate Committee
From: Robert C. Miller, Thesis Supervisor

The program outlined in the proposal:

Title:	Usable Security: Design Principles for Creating Systems that are Simultaneously Usable and Secure
Author:	Simson L. Garfinkel
Date:	October 15, 2003

is adequate for a Doctoral thesis. I believe that appropriate readers for this thesis would be:

Reader 1:	Ronald Rivest
Reader 2:	Daniel J. Weitzner

Facilities and Support for the research outlined in the proposal are available. I am willing to supervise the research and evaluate the thesis report.

Signed: _____

Robert C. Miller

Title: Assistant Professor, MIT Department of Electrical Engineering and Computer Science

Date: _____

Comments: _____

Massachusetts Institute of Technology

Department of Electrical Engineering and Computer Science

Cambridge, Massachusetts 02139

Doctoral Thesis Reader Agreement

To: Department Graduate Committee
From: Ronald Rivest, Thesis Reader

The program outlined in the proposal:

Title:	Usable Security: Design Principles for Creating Systems that are Simultaneously Usable and Secure
Author:	Simson L. Garfinkel
Date:	October 15, 2003
Supervisors:	David D. Clarkand Robert C. Miller
Other Reader:	Daniel J. Weitzner

is adequate for a Doctoral thesis. I am willing to aid in guiding the research and in evaluating the thesis report as a reader.

Signed: _____

Ronald Rivest

Title: Andrew & Erna Viterbi Professor Of Computer Science and Engineering

Date: _____

Comments: _____

Massachusetts Institute of Technology

Department of Electrical Engineering and Computer Science

Cambridge, Massachusetts 02139

Doctoral Thesis Reader Agreement

To: Department Graduate Committee
From: Daniel J. Weitzner, Thesis Reader

The program outlined in the proposal:

Title:	Usable Security: Design Principles for Creating Systems that are Simultaneously Usable and Secure
Author:	Simson L. Garfinkel
Date:	October 15, 2003
Supervisors:	David D. Clarkand Robert C. Miller
Other Reader:	Ronald Rivest

is adequate for a Doctoral thesis. I am willing to aid in guiding the research and in evaluating the thesis report as a reader.

Signed: _____

Daniel J. Weitzner
Title: Principal Research Scientist
Date: _____

Comments: _____

Contents

1 Introduction

It is widely believed that security and usability are two antagonistic goals in system design. A classic example of this argument is passwords: systems without passwords are thought to be usable, but not very secure, while systems with long passwords that must be frequently changed are thought to be secure, but not very usable.

I believe that this reasoning is flawed. Systems that sacrifice security for usability may work fine in the laboratory, but they fail when exposed to the hostile environment of the outwide world. Alternatively, systems that sacrifice usability in favor of security fail because the security features are disabled by users, or because the systems are used far less than they would be otherwise.

This is not a novel argument. In 1975, Saltzer and Schroeder[SS75] identified eight design principles for building secure computer systems. These eight principles have become standards of the computer security lexicon: economy of mechanism; fail-safe defaults; complete mediation; open design; separation of privilege; least privilege; least common mechanism; and psychological acceptability. On the subject of psychological acceptability, Saltzer and Schroeder wrote:

> "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 other words, it must to be easier to use security mechanisms than to bypass them.

For the two decades following the publication of Saltzer and Schroeder's article, the need for deploying security systems that provide for "psychological acceptability" was frequently repeated but rarely expounded upon.

Indeed, it is only recently that computer security researchers have turned their attention to the issue of Human Computer Interface (HCI) design — or that the HCI community has turned its attention to the topic of computer security. For example, in 1999 Adams and Saase noted "it seems that currently, hackers pay more attention to the human link in the security chain than security designers do."[AS99]. Similar sentiments were expressed by Whitten and Tygar, who wrote: "We have found very little published research to date on the problem of usability for security."[WT99]

At the same time, there is growing recognition that many security breaches are not the result of technical failures, but instead are the result of the failure of humans to use computer systems in a secure manner. [Sch02, MS02]

Some researchers go further. Carl Ellison and others have argued that the problem facing security software is not poor interfaces, but the fact that they have user interfaces at all. At the February 2000 meeting of the American Association for the Advancement of Science, Ellison went further and proposed[1] this somewhat charming rule-of-thumb:

> "The user base for strong cryptography declines by half with every additional key press or mouse click required to make it work."

Even if not strictly true, the sentiment expressed by this observation seems to carry much weight: users generally do not comply with security measures that require additional work.

1.1 Work in Security Requires Realistic Threat Models

Phrases like *security breaches* and *computer security* mean different things to different people. In this day of computer worms and viruses, one might be tempted to define a secure computer as a computer that is not susceptible to attack over the network. By this definition, a laptop that is not plugged in to a network could be thought to be "secure." The owner who leaves such a laptop unattended at a hotel bar may be disappointed to find that the "secure" computer has been stolen upon his return. Clearly, being able to withstand an attack over a network is not the only measure of security.

In 1991, Gene Spafford and I developed a simple operational definition for computer security:

> **Computer Security: "A computer is secure if you can depend on it and its software to behave as you expect it to."[GS91]**

This definition has been widely cited since publication. But does it make sense? It may be overly broad, as it appears to encompass many apparently unrelated activities. Certainly security professionals are charged with protecting computers against many "security threats" that are foremost in the public's mind, such as hostile insiders and computer viruses. But also included in this definition is the need to protect against hardware failure by providing

[1] Because he made this statement at a public forum where journalists were present, this statement has since come to be known as "Ellison's Law." In fact, Ellison attributes the statement to Steve Lipner, with whom Ellison shared an office when they both worked Trusted Information Systems.

redundant or super-hardened systems; assuring for the continuity of operations in the event of power failures or natural disasters by having multiple data centers; and protecting against software failures through the use of heterogeneous systems or verified software development procedures.

I believe that the definition, while broad, remains useful in analyzing risk and protection against risk. Others concur. For example, the CISSP[2] professional certification administered by the International Information Systems Security Certifications Consortium, Inc., requires mastery of ten different areas that the organization refers to as the Common Body of Knowledge. The ten areas are:

- Access Control Systems & Methodology
- Applications & Systems Development
- Business Continuity Planning
- Cryptography
- Law, Investigation & Ethics
- Operations Security
- Physical Security
- Security Architecture & Models
- Security Management Practices
- Telecommunications, Network & Internet Security[Con03]

This list goes significantly beyond purely technical systems for securing computer systems — for example, those systems articulated by the DOD's "Orange Book." [ASODC85] But it is a list that is consistent with the goal of assuring expected operations.

The 2003 CSI/FBI Computer Crime and Security Survey[CSI03] defines 12 types of attacks or computer misuse, 10 of which result in direct financial loss, according to the survey's 398 respondents. Those attacks are:

[2]Certified Information Systems Security Professional

Attack	Percentage of Organizations Reporting	Reported Dollar Loss
Theft of Proprietary Information	21%	$70,195,900
Denial of Service	42%	$65,643,300
Computer Virus	82%	$27,382,340
Insider Abuse of Net Access	80%	$11,767,200
Financial Fraud	15%	$10,186,400
Laptop Theft	59%	$6,830,500
Sabotage	21%	$5,148,500
System penetration	36%	$2,754,400
Telecom Fraud	10%	$701,500
Unauthorized Access by Insiders	45%	$406,300
Active Wiretap	1%	n/a
Passive eavesdropping	6%	n/a

Arguably, the threats identified by the CSI/FBI survey involve a failure of security according to the (perhaps overly broad) 1991 definition. A computer that is "secure" is not a computer allows the theft of proprietary information, it is not a computer that is susceptible to denial of service attacks or computer viruses, it is not a computer that can be abused by insiders, and so on.

But with more than three decades of serious attention to computer security issues, how is it that the respondents to the CSI/FBI survey could document more $200 million worth of loss? Aren't they using secure systems?

The answer, of course, is that while many organizations are using operating systems that implement various kinds of security technology, vulnerabilities remain. Here are some possible explanations as to why:

1. Many users do not understand the security implications of their actions. Thus, in normal operations of their computer system, they may make configuration changes that leaves their system open to attack.

2. Today's systems are extremely complex, with many configuration parameters and background processes. As a result, it is difficult for a user to assess if their system is secure or not.

3. Because of #1 and #2, it is relatively easy to trick users into compromising the security of their own systems. One example of this is social engineering, in which an attacker might telephone a an employee of a large organization, pose as a member of the organization's

system administration team, and ask the user to download a "software update" that is in fact a Trojan horse.[MS02] A second example might be a computer virus that propagates by sending copies of itself through email, each copy promising the recipient free access to pornography if only they will open the attached Microsoft Word document, as was the case with the Melissa Virus (W97M_Melissa).[Sha99]

4. Security systems are complicated and frequently require extended user interaction in order to function properly. As a result, users may disable these systems so that they can get their jobs done faster.

5. As a result of operating system design, a single bug or programming error can frequently be exploited to give an attacker or attacking program complete control of a computer system. This vulnerability is magnified by large numbers of computers on the Internet running similar or identical software configurations, allowing for the rapid propagation of network worms.[SPW02]

Most of these explanations have a common thread: they blame humans for the admittedly human failure of not mastering complex technical systems. (Even the last explanation, which is the commonly-held explanation for the emergence of flash worms[SPW02], is another version of "blame the human" — in this case, blaming IT departments for their decision to reduce operational complexity by standardizing on a single operating system.) But Norman [Nor83] and others argue that what we frequently view as human error is actually the result of design flaws that can be overcome.

If humans are responsible for security problems, one logical approach is to eliminate the need for so many humans to be making security-critical decisions. A second approach is to educate them so that they will make more responsible decisions. To date, neither of these approaches has been tremendously successful.

1.2 A Workable Threat Model for Today's Desktop Computing Environment

After beginning the previous section arguing that phrases like *security breaches* and *computer security* mean different things to different people, I wish to argue the reverse. That is, there appears to be an emerging common threat model that is shared by the majority of computer users. In part, this is because the majority of desktop users are using their computers for more-or-less the same purpose: email, web browsing, word processing, and simple finance operations. They have a (possibly intermittent) Internet connection, one or more computers for which they are responsible, and little or no outside assistance. This brief sketch describes home users but it increasingly describes users in large organizations as well.

For such users, I conjecture that security priorities are remarkably similar:

- Users engaged in Internet banking or other online financial activities do not wish to have their money stolen.
- Users do not want to lose their data. For most users, data integrity is more important than privacy or even availability. (Justification: many users that I have met have had their laptop break while traveling. Almost without exception, they report that it was upsetting to be without their computer, but their main concern was for their data — at least it was safe.)
- Once the safety of their data is assured, users would like to safeguard the physical security of their computer.
- After the integrity of data is assured, users would like for their computer to operate in a reliable fashion.
- After integrity and availability are assured, users would like to have some protection for the privacy of their data.
- Users do not want their computers to crash. They do not want the access to their data or to web sites slowed by denial-of-service attacks or computer viruses.[3]
- Finally, users would like to be free from intrusion. Specifically, they would like to be free of spam, pop-up advertisements, and "spyware."

Despite these security priorities, most users are operating with threat model that is not entirely accurate. Specifically, I believe that most users are concerned with the following threats, in order of perceived decreasing likelihood:

1. Attack over the Internet by hackers, computer viruses, and spammers.
2. Theft of a computer system.
3. Loss of data due to operator error, software error, or computer theft.

Clearly, this list is based on anecdotal evidence: it might be worthwhile to conduct a survey of users to find out their true security priorities and threat models. But even this list is useful for guiding research on the topic of synergy between security and usability: success in addressing the issues above should have immediate payoff for many users.

[3]Interestingly, one of the primary complaints against so-called spyware seen in the press is not that it violates the user's privacy, but that it slows down the user's computer.

1.3 Previous Attempts at Assuring Desktop Security: Mandatory Access Controls and User Education

One way to assure operational security is for computer systems to simply disallow actions that are defined by policy to be not secure. In a military classification environment, for example, information from a document classified at the Top Secret level should not be incorporated into a Confidential document without an explicit review process. The Bell-LaPadula security model[BL73] is an attempt to formalize these requirements so that a single computer system could be used to store and process information at multiple security levels. This model relies on mandatory access controls, data labeling, information tainting, and a reference monitor that enforces the system policy.

Clark and Wilson [CW87] argue that commercial data processing systems have a different set of requirements that are implemented with a complementary set of mandatory access controls. Specifically, Clark and Wilson argue that while the primary security requirement of military and intelligence customers is confidentiality and data secrecy, commercial customers are far more interested data integrity.

An alternative to mandatory access controls is aggressive user education. User education is a particularly seductive idea because it offers the promise that computer security could be improved without the need to deploy new software, and that the burden of securing computers could be shifted from programmers and administrators to the users themselves.

In August 2003 the Microsoft Corporation ran a series of full-page advertisements in major metropolitan newspapers throughout the United States asking the American Public to “Protect your PC” (see 1). The text of the advertisement was quite straightforward:

> Microsoft wants to help ensure your PC is protected from the latest *Blaster* virus, as well as from future threats. Please go to www.microsoft.com/protect and follow these steps today:
>
> 1. Use a firewall, like the Internet Connection Firewall already in Windows® XP.
> 2. Use Microsoft® Windows® Update to get and keep your PC up-to-date.
> 3. Install antivirus software and ensure it’s up-to-date

Although these instructions seem simple enough, the devil lies in the details. Those details appear on the website mentioned in the advertisement (see Figure 2). The first recommendation, “Use a firewall, like Internet Connection Firewall already in Windows XP,” requires users to follow an 8-step sequence on their computers. Completing this sequence requires the user to

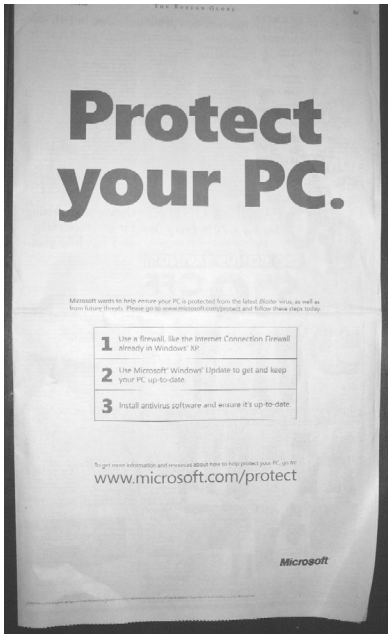

Figure 1: A Full-page Microsoft advertisement telling people to secure their computers

Figure 2: The Microsoft website www.microsoft.com/protect

identify their type of Internet connection, click on an "Advanced" tab, and select a check-box "Protect my computer and network by limiting or preventing access to this computer from the Internet." Unfortunately, the check-box was unchecked by default. Perhaps the purpose of this very public newspaper campaign is to overcome configuration errors with Microsoft's shipping product.

Microsoft's second recommendation, "Use Windows Update," requires that the user choose an option from the start menu, download and run an ActiveX applet, and then decide which "Critical Updates" to install. The third step, "Install antivirus software," apparently cannot be accomplished without purchasing software from a third-party vendor. (In fact, there are free anti-virus solutions available for Windows home users, [Sof03] but Microsoft neglects to mention the free products on its website).

The success of the Blaster virus in spreading and compromising computers indicates that even these relatively simple instructions are still too complicated for many of Microsoft's users — presumably the same users that Microsoft needs to reach by advertising in the newspaper.

When I first entered the field of computer security 15 years ago, I was convinced that many security problems could be addressed by properly educating users. Acting upon this belief, the majority of my efforts in the field revolved around writing about security issues for the computer-using public, as exemplified by [Gar87] and [GS91].

Experience has shown that many users are resistant to education attempts. One possible reason is that it takes time to properly secure a computer—time that detracts from a users' other goals and gives no obvious benefit. Another explanation is that computer systems have evolved over the past 15 years to the point where education, even if it were possible, can not be sufficient. Today's computer users are presented with a dizzying array of choices that they need to make on an almost continual basis: many of these choices have deep implications for system security. Users that do not concern themselves with security are therefore more likely to be more productive than they would be otherwise — provided that they do not suffer an incident, that is. Even then, the cost of the incident may be less than the deferred cost.

Such analysis need not be restricted to individual users. Organizations that invest heavily in security have a higher cost of operation than those organizations that do not; if the organization is not attacked, the money spent on security appears to have been wasted. Thus, over time, the market will favor companies that have poor security and are lucky enough to not be attacked, for these companies will have a lower cost of operation than companies that are properly secured.

As there is no way to assure luck, many users and organizations will try to secure their systems. (An alternative approach is to use insurance to minimize the risk; this approach is complicated because it is difficult to estimate the likelihood of a comptuer security incident and the potential damage that the incident might cause — two items necessary for the calculation of an insurance

policy.) Yet even educated users simply do not have enough information to make appropriate decisions.

Thirty years after the first publication of his paper, Professor Saltzer was fond of saying that his new Macintosh computer running the MacOS X operating system had more than 100,000 files.[4] How is the user supposed to audit his system and know if it is properly configured?

In fact, many security determinations can be made quite successfully using automated tools. Usable, deployable rule-based computer security checkers were created in the 1980s at MIT [Bal88] and Purdue [FS90]; this work was continued in the 1990s at Texas A&M [SSH93]. Although such checkers are less popular these days than they were in the 1990s, the approach has been applied to detecting network security vulnerabilities with significant technical [FV95] and commercial success [Int03]. Indeed, organizations that do not rely on some degree of automated network scanning are now deemed to be not following industry "best practices."

1.4 The "Usable Security" Thesis

I believe that many persistent security problems will only be addressed by designing systems that provide for *Usable Security*. By Usable Security, I mean that the software developer should provide for secure defaults; that the system should use automation to provide for secure operation; that the system should inform the user of the implications of decisions; and that user actions — especially those with security implications — should be reversible after the fact (if at all possible). As I shall show in my dissertation, some Usable Security elements have already emerged in the marketplace. I believe that we can systematize and formalize the design of such systems, rather than relying on improvements painstakingly discovered through the current process of trial-and-error.

My thesis is:

> Security and usability can be simultaneously improved by the adherence to a set of design principles. These principles can be inferred from a critical examination of existing systems and tested by relying upon them in the design of new systems. Key among these principles are minimizing user input; making decisions on behalf of the user; informing the user of actions taken upon his or her behalf; and providing the user the ability to undo those actions when possible, and otherwise to minimize their impact.

[4] In fact, on the version 10.2.5 MacOS X that was used to create parts of this document, the author counted a total of 147,197 files combined in the /Library, /System, /System Folder, /bin, /dev, /private, /sbin and /usr directories.

2 Related Work

3 Related Work on HCI and Computer Security

Interest in both the general usability of computer applications and the usability of security-critical aspects of desktop applications has increased steadily in recent years, much as computers have diffused throughout society.

3.1 Norman's Design Rules and Error Analysis

Norman observed in 1983 that many users new to a computer system make the same common errors. "Experienced users ... often smil[e] 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, wrote Norman, 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.

Perhaps because mistakes are frequently the result of poor training, Norman's analysis concentrates on slips. He classifies them into three categories, each of which he divides into further subcategories. He argues 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. When it comes to modes, Norman argues that it is hard to eliminate modes and, when this is accomplished, the resulting system is often awkward for novices and experienced users alike. Instead, he says, users need to be made aware of modes and commands from one mode should not be interpreted as other commands in other modes.

Even when users are made aware of modes, they will still occasionally make errors. "People will make errors, so make the system insensitive to them," Norman argues. Specifically, he argues for "safeties" to make irreversible actions difficult, and improved undo systems so that fewer actions are irreversible.

Finally, Norman's article proposes four principles for the field that we now call Computer Human Interaction. They are:

Feedback The state of the system should be clearly available to the user, ideally in a form that is unambiguous and that makes the set of options readily available so as to avoid mode errors.

Similarity of response sequences Different classes of actions should have quite dissimilar command sequences (or menu patterns) so as to avoid capture and description errors.

Actions should be reversible (as much as possible) and where both irreversible and of relatively high consequence, they should be difficult to do, thereby preventing unintentional performance.

Consistency of the system The system shoudl be consistent in its structure and design of command so as to minimize memory problems in retrieving the opeations.

3.2 Johnson's Usability Barriers in GUIs

Writing in 1983, Norman thought that many usability problems with the computers would be solved by the introduction of the graphical user interface (GUI). And indeed, many have been.

But the graphical user interface alone is not a panacea for usability. Johnson [Joh00] discuss 82 different barriers to usability that are commonly present in GUI-based programs. Johnson refers to these problems as "bloopers." Typical bloopers are *Duplicate menu items* (Blooper #2, a GUI component blooper), *Inconsistent terminology* (Blooper #33, a textual blooper), and *Misunderstanding what user interface professionals do* (Blooper #76, a management blooper).

From this list of bloopers, Johnson derives eight fundamental principles for the design of application software and complex websites:

Principle 1 Focus on the users and their tasks, not the technology

Principle 2 Consider function first, presentation later

Principle 3 Conform to the users' view of the task

Principle 4 Don't complicate the users' task

Principle 5 Promote learning

Principle 6 Deliver information, not just data

Principle 7 Design for responsiveness

Principle 8 Try it out on users, then fix it!

Johnson argues that usability can be improved by making programmers, designers and managers aware of these principles and bloopers. (This might be seen as another example of Bhagwati's "Dracula Effect:" evil exposed to light will wither and die.[Bha00])

3.3 HCI-SEC Bloopers, Definitions and Principles

Interfaces that manage security-relevant features of application programs and websites has always represented a specific challenge to designers. Although Johnson does not discuss security *per se* — in fact, the word "security" does not even appear in Johnson's index — several of Johnson's bloopers are commonly made in the design of security software. Some of these bloopers are:

- **Blooper #3: Hidden functions** Security interfaces typically hide important functionality under menus only accessible by right-clicks, under buttons that say "advanced," or under icons that are not obviously clickable.
- **Blooper #35: Speaking Geek** Confusing terminology is endemic among security programs. In part, this may be due to the specialized terminology of computer security — both its unique vocabulary and the fact that common words such as "privacy" and "key" frequently have different meanings in the security context.[5]
- **Blooper #43: Exposing the implementation to users** Underlying security implementations (e.g., references to keys, certificates, trust, etc.) are common in many security programs. Such references cannot help but decrease usability for untrained users.

There is a growing realization that poor security resulting from improper configuration or operating procedures is frequently the result of underlying usability problems. The term HCI-SEC (Human Computer Interaction — Security) is increasingly being used to label the field that merges the requirements of good user interface with good security.

Whitten and Tygar define security software as 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. [WT99]

[5]The example that Johnson uses for his *Speaking Geek* blooper example is not surprisingly drawn from the world of security: Johnson describes a message that a user is presented on returning to an inactive session: `Your session has expired. Please reauthenticate. [OK]` "When users acknowledge the message by clicking OK, they would be returned to the "User Authentication" page," Johnson writes. "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]

Whitten and Tygar pair further describe five properties that make security inherently difficult for user interface design. These properties are:

the unmotivated user property Security is (at best) a secondary goal of users. "If security is too difficult or annoying, users may give up on it altogether."

the abstraction property Security policies are usually phrased as abstract rules that are "alien and unintuitive to many members of the wider user population."

the lack of feedback property It is difficult to provide good feedback for security management and configuration.

the barn door property Once a secret gets out, its out. Information disclosure cannot be reversed.

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."

Taken together, these properties can argue for increased user training, or else they can argue for removing the user from security-critical decisions whenever possible. It has not been shown, however, that these principles can be used for the creation of usable security software.

Yee [Yee02, Yee03] has developed an *Actor-Ability Model* 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. The model is based on the capabilities available to the discrete actors resident on the user's computer. These actors might be the underlying operating system, programs like Microsoft Word, and web browsers. The first actor, A_0, is the user. The model proposes that users envision a finite set of actors $A = \{A_0, A_1, A_2, \ldots, A_n\}$, and for each actor, the user believes that the actor can perform an action P_i. Yee 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. Yee then argues that there is a *no-surprise condition* that is true when the user is more powerful than she realizes and the other actors on the system are less powerful. That is:

If:

$$\begin{aligned} \text{actors } A &= \{A_0, A_1, \ldots, A_n\} \\ \text{perceived abilities } P &= \{P_0, P_1, \ldots, P_n\} \\ \text{real abilities } P &= \{R_0, R_1, \ldots, R_n\} \end{aligned}$$

Then the *no-surprise condition* requires that:

$$P_0 \subseteq R_0 \qquad \text{and}$$
$$P_i \supseteq R_i \qquad \text{for } i > 0$$

Using this model as a basis, Yee has developed a list of "ten suggested principles for secure interaction design." He further divides these principles into *Fundamental Principles*, *Actor-Ability State*, and *Input and Output* principles.

Yee's Fundamental Principles include:

Path of Least Resistance. The most natural way to do any task should also be the most secure way.

Appropriate Boundaries. The interface should expose, and the system should enforce, distinctions between objects and between actions along boundaries that matter to the user.

The Actor-Ability State principles, based on his Actor-Ability Model, are:

Explicit Authorization. A user's authorities must only be provided to other actors as a result of an explicit user action that is understood to imply granting.

Visibility. The interface should allow the user to easily review any active actors and authority relationships that would affect security-relevant decisions.

Revocability. The interface should allow the user to easily revoke authorities that the user has granted, wherever revocation is possible.

Expected Ability. The interface must not give the user the impression that it is possible to do something that cannot actually be done.

Finally, Yee's Input and Output principles are:

Trusted Path. The interface must provide an unspoofable and faithful communication channel between the user and any entity trusted to manipulate authorities on the user's behalf.

Identifiability. The interface should enforce that distinct objects and distinct actions have unspoofably identifiable and distinguishable representations.

Expressiveness. The interface should provide enough expressive power (a) to describe a safe security policy without undue difficulty; and (b) to allow users to express security policies in terms that fit their goals.

Clarity. The effect of any security-relevant action must be clearly apparent to the user before the action is taken.

There are many interesting observations in Yee's paper. For example, Yee argues 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.

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."

Yee's paper considers five case studies of how his principles could be applied to existing HCI-SEC problems: Software installation and Maintenance; Website Password Prompts; Java Applet Privileges in Netscape; ActiveX and Code Signing; and E-mail and Macro Viruses. His most innovative ideas involve associating pop-up windows with their controlling windows through the use of lines drawn by the operating system (alternatively implemented by "sheets" in the Apple Aqua interface[Com02]), and his observation that the Java system could limit an applet's access to files on the user's computer to files that the user has chosen through the computer's "File Open" window. He notes that programs run from the e-mail client or Microsoft Word should have to be granted explicit permission before they could delete files or send e-mail — a feature that Microsoft subsequently implemented. Other observations in his case study are equally valid but less useful from a design prospective: he notes that ActiveX gives potentially untrusted pieces of downloaded code too many privileges, and says that downloaded code should not be allowed to modify the operating system. (Clearly, *some* downloaded code needs to be able to modify the operating system, or else users would not be able to download security patches.)

Yee's work represents the most sophisticated analysis of the HCI-SEC problem to this point in time. But the work can clearly be improved upon. There are a few obvious areas in need of clarification. For example, Yee's Trusted Path principle appears to be a subset of his Identifiably principle.

Yee has relayed in private correspondence to me that he believes his principles are general design principles. I disagree: the principles appear to be highly attuned to the problem of coordinating multiple processes operating on a single-user computer, especially when some of the processes might be hostile. Even though Yee's No Surprise Condition is based on the definition of computer security put forth in [GS91], it isn't immediately clear how traditional security requirements such as Audit or the need to make backups are handled by his principles.

Yee's principles and his underlying Actor-Ability Model are based on abilities that actors can perform or that the user think that the actors can perform. These *abilities* can be thought of as capabilities. The problem is that today's operating systems are not capability-based: they are based on restrictions imposed on users. Although it is possible to calculate a set of restrictions from a set of capabilities, doing so requires knowledge of all possible abilities; it is not clear that all users are capable of such explicit enumeration. Put another way, it may be that Yee's "Expected Ability" principle should be re-envisioned as a "Expected restriction" property — that is, the system must not give the impression that a thing is not possible, when in fact it is possible.

3.4 Other HCI-SEC Work

Adams and Sasse [AS99] investigated why users pick poor passwords and management techniques that can be employed to improve password selection. Key observations include that forcing users to change passwords on a regular basis can decrease overall security, since users are forced to write down passwords in order to remember them. This finding is in direct contradiction to the Federal Information Processing Standard #112, "Password Usage" [NIS85], which states that passwords "shall have the shortest practical lifetime," and that passwords "shall have a maximum lifetime of 1 year."

Adams and Sasse make some recommendations on user training, but do not make any substantive recommendations on technology or usability.

Smith [Smi03] argues that the security provided by PKI is illusory given the ease with which users can be spoofed using today's interfaces. As an example, he cites the case of `palmstore.com`, a "secure" site that is apparently branded by Palm Computing for the purpose of selling Palm products, but which has a certificate that belongs to "Modus Media International." The security of SSL depends not just on the fact that encryption is being used, but also on the fact that the encrypted connection terminates at an organization that is trusted by the user. In theory, the certificate (issued by a trusted authority) allows the user to verify the identity of the organization operating the SSL-enabled server. It is unlikely that most users doing business with Palm Computing's online store realize that they are actually dealing with a company named Modus Media — in fact, a semi-skilled computer user might suspect that this might be the precise

kind of spoofing attack that SSL and PKI were designed to prevent. (Since the publication of Smith's article, the name on the Palm Store's certificate has been removed and replaced with the empty string.)

Pereira [Per03] created PKIGate with provides for automatic signing and encryption of S/MIME mail for the Dartmouth College web mail system. The system is based on the DigitalNet S/MIME Freeware Library and uses the IBM 4758 secure coprocessor for key storage and public key functions. Sadly, Pereira did not complete the project prior to graduating and work on the project has been halted.

3.5 Other Factors That Can Affect Software: Economics and Regulation

Recent academic work suggests that economic factors have a significant impact on computer security. Recent regulatory experience suggests that changes in federal procurement regulations can have a significant impact on software and systems that are sold to non-federal customers. Using this work and experience as a guide, I believe that it should be possible to modify federal regulations to support the creation of systems that significantly more secure and usable, once it can be shown that techniques for doing so exist.

Schechter [SS03] suggests that economic factors are dominant when organizations decide which security technologies to deploy. Without perceived risk and demonstrated return-on-investment, organizations will simply not deploy strong security technology, but will instead tolerate the perceived (or not perceived) risk. Schechter suggests that actions on the part of decision makers may not always be made with a full understanding of risks and returns of security technology, and that by focusing on economic aspects it may possible to improve both the deployment of security technology and, at times, its very nature.

Oldyzko [Odl03] argues security practitioners sabotage their own efforts by failing to take into account economic factors. He argues that the cost of security, especially in terms of increase costs of operation, may not be justified when compared to the costs of actual security incidents. (Oldyzko also notes that economists have had difficulty in establishing the indirect costs of earthquakes and storms.) He also notes that what security practitioners believe to be necessary levels of security frequently do not match accepted levels in the business world. For example, Oldyzko writes, businesses routinely use faxed signatures to execute transactions, even though such signatures are easily forged. Secretaries routinely sign the names of their bosses and, in many offices, know their supervisor's password. Clearly, security software that works with real-world business requirements will be more likely to be adopted than software that works against such requirements.

Yet another approach that has been applied to changing software is regulation. The Workforce Investment Act of 1998 amended Section 508 of the federal code to requires that Federal

agencies purchase electronic and information technology that is accessible to people with disabilities. The section applies to both technology that is used by Federal employees and by members of the public.

The statute has had two primary effects. The direct and most obvious effect is that the federal government has helped to create standards defining usability and that procured technology now generally follows those guidelines. For example, federal websites can now generally be navigated by people using screen readers, something that was not true prior to the Act's passage. But the secondary impact of the Workforce Investment Act has been far more pervasive: because of the Federal government's buying power, vendors throughout the computer industry have modified their systems to be in general compliance with the law, even for systems are designed primarily for sale to the general public and not the Federal Government.

For example, the current versions of both Microsoft Windows XP and Apple Macintosh MacOS X have extensive support for people with vision or motor impairments. Prior to the passage of the Workforce Investment Act, built-in facilities were generally poor and many features were only available through the use of third-party additions.

Reviewing this literature, I conjecture that systems designed for Secure Usability will have a positive return on investment by lowing the cost of operation through improved usability. These systems should simultaneously lower risk through improved security. Given the Federal Government's new security awareness after the terrorist attacks of September 11th, 2001, it should be possible to turn the Usable Security into guidelines that could be used in procurement and regulation.

4 Designing for Usable Security

To prove my thesis, I intend to develop a set of design principles that can be followed by programmers, managers and product architects to simultaneously improve security and usability of desktop application programs, networked workstations, Internet servers, and handheld devices.

Some principles that I have developed over the past six months include:

Zero Impact If security features can be optionally enabled or disabled by the user, then the use of security features must not result in a decrease of other functions to the user. Otherwise, the user will invariably disable security features in order to achieve increased functionality.

Zero-Click Security features must not have mandatory user interface interactions that serve no purpose other than to control the security options. That is, defaults should allow for secure set-up and operation.

Visibility of Actions The system's security-relevant internal state and actions that are taken on behalf of the user should be visible.

Reversibility When possible, actions taken by the user or by the system on behalf of the user should be reversible. Actions taken by the system shoudl also be clearly documented to the user. The principle of reversibility can be extended beyond actions taken by the system. Reversibility as a user interaction principle has been explored by others — for example, "Time-machine computing" discussed in [Rek99].

Completion All tasks initiated by the user that the computer claims are completed should actually be completed. For example, if the user attempts to "cleared," "erased" or "deleted" through the user interface, the information should be truly removed from the computer and not left residing elsewhere in the system.

User Audit Provide the ability to correct or remove personal information where that information is displayed.

Override-ability There needs to be a way for the user to determine what action the system will take on the user's behalf. If the user doesn't want that action to be taken, there must be a way to override it.

No External Burden Security tools must not pose a burden on non-users who do not otherwise benefit from its use.

Clearly, the Reversibilty and Completion principles are somewhat at odds. A key aspect of this thesis will be developing a strategy for resolving this tense.

Note that these principles are somewhat similar to Yee's, although mine are generally more proscriptive than his.

Like Yee's principles, my principles seem like nothing more than common sense. For example, the User Audit principle can be thought of as an application of the Fair Information Practices[U.S73] to user interface design. Nevertheless, this principle has been violated by two of the most widely used pieces of consumer and business software on the planet: Netscape Navigator and Internet Explorer, as I demonstrate in section 4.2 of this document. Likewise, the No External Burden principle is violated by the PGP and S/MIME mail security standards, as I show in section 4.4.

As part of my thesis, I hope to extend this list to a rather detailed list of between 50 and 100 detailed design techniques ("patterns"), from which a more generalized list of principles will be formed. I then hope to show that systems that follow my principles generally provide their users with security and usability, while systems that violate the principles suffer from both poor security *and* poor usability. I will then create a number of desktop applications that modify existing systems to be in conformance with my principles. I will show that when the principles are followed, security and usability both improve.

4.1 Case Study #1: The Success of SSH and SSL

What are undeniably the most successful cryptographic systems in use today, SSH [Ylo96] and SSL/TLS [FK96], owe a large part of their success to their adherence to the Zero-Click principle. In fact, implementations of these protocols actually go further: they improve security and usability by removing user choice.

SSH is the Secure Shell, a replacement for the TELNET protocol that offers session encryption and optional certificate-based authentication. SSL/TLS is used to transparently encrypt web pages that are fetched using the "https:" protocol. SSL provides for both the privacy of Internet transmissions and protection against active "spoofing" attacks through the use of server certificates.

With respect to their privacy guarantees, both SSL and SSH appear to be a success. Although there is little hard data available on the success of either of these protocols in stopping attacks, anecdotal evidence suggests that the deployment of SSH over the past five years has largely eliminated the problem of "password sniffing" — a problem that was epidemic in the late 1990s. Meanwhile, there is not a single recorded case of an Internet attacker capturing a credit card number that was sent by SSL. These protocols are very effective in protecting against passive attacks.

```
[simsong@r2 ~] 637 % ssh localhost
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that the DSA host key has just been changed.
The fingerprint for the DSA key sent by the remote host is
e7:68:64:7d:d2:42:2e:51:1e:2f:c7:93:72:9f:ab:c5.
Please contact your system administrator.
Add correct host key in /home/simsong/.ssh/known_hosts to get rid of this message.
Offending key in /home/simsong/.ssh/known_hosts:17
DSA host key for localhost has changed and you have requested strict checking.
Host key verification failed.
[simsong@r2 ~] 638 %
```

Figure 3: SSH provides notification to the user when a server's public key changes.

The primary use of SSL and SSH is to provide cryptographic tunnels that are secure against eavesdropping. But tunnels to where? Without authentication, these protocols do not provide protection against active man-in-the-middle attacks. And when it comes to authentication guarantees, the story of SSL and SSH is more complicated.

- In the case of SSL, flaws in the way that certificates for Certificate Authorities have been distributed combined with user interface failings and lax business practices have effectively rendered the authentication guarantees of SSL all but meaningless. Specifically, today's web browsers are distributed with more than 100 trusted root certificates — all of them equally trusted. When contacting a website it is possible in principle to learn the CA that signed the website's certificate, but in practice the procedure is difficult and rarely performed. Some of these certificates belong to now-defunct dot-coms. Finally, many organizations provide Internet-based services using outsourcing arrangements; in these cases, the name that is on the certificate frequently has no obvious connection to the name on the certificate's website.[6]
- SSH provides no certificate-based authentication *per se*, but it does provide a warning to users when the public key associated with an SSH server changes (see Figure 3).

Clearly, neither SSL nor SSH as they are used today provide for strong server authentication. Perhaps the sufficiency of these tools is evidence that active attacks on the Internet are rare (a conclusion that is in line with the results of the CSI/FBI survey [CSI03]).

The apparent success of both SSH and SSL stands in marketed contrast to the failure of other widely-fielded cryptographic systems. The 1990s saw the deployment of *many* systems, such as the e-cash [CFN88] anonymous micropayment system, and the SET [VIS97] credit-card

[6]For example, the Internet banking service provided by the Massachusetts-based Cambridge Trust Company is authenticated with a certificate belonging to the Metavante Corporation in Milwaukee, Wisconsin.

processing system. Why did SSH and SSL succeed where other systems, such as fail? Some reasons may be:

- SSH and SSL added encryption to practices that users were already engaged in and required no additional effort on the part of the user, in accordance with the *Zero-Click Principle*. Both e-cash and SET imposed significant usability burdens upon the user and required that users significantly change their behavior.
- SSH and SSL addressed a protocol weakness in the TCP/IP protocol (its ability to be monitored) that was being actively exploited by attackers. SSL and especially SSH solved real problems. E-Cash[7] and SET[8] provided novel solutions to existing business practices that were not regarded as security problems by the industry.
- The decision to deploy SSH was made by system administrators who literally functioned as gatekeepers to specific computer systems. Once the decision was made to deploy the technologies, these individuals could tell their users that no access to the computer system would be granted by unsecure protocols. The decision to deploy SSL was made possible because web servers and clients were already equipped with the technology: webmasters who could, for the most part, deploy SSL by simply purchasing and installing a certificate onto an existing webserver, and then changing a few "http:" links to read "https:".

4.2 Case Study #2: Browser History in Microsoft Internet Explorer, Apple Safari, and Mozilla 1.4

The *User Audit Principle* above is well demonstrated by the contrasting approaches that the Microsoft Internet Explorer, Apple Safari, and Mozilla web browsers take to the removal of personally-identifiable information from the user's hard drive.

Web browsers retain a substantial amount of personal information during the course of normal operation. Among this personal information is:

[7] E-Cash was marketed to users as an anonymous payment system designed to protect privacy against invasive corporate and governmental eavesdroppers. Most consumers appear to have little concern about revealing their identity to a merchant. Those who have such concerns may simply avoid the Internet altogether and instead make purchases in person using cash.

[8] SET was designed to provide an additional layer of encryption so that merchants would never have access to raw credit-card numbers. One justification on the part of SET's inventors was that SET would enable small businesses without credit-worthiness to be able to accept credit cards. In practice, banks did not wish to deal with these small businesses. What's more, the merchants that actually did implement SET required that the banks provide them with decrypted copies of the customer's credit-card number so that the merchant's legacy systems could still function properly.

1. Browser history, used to implement the browser's "history" menu.
2. A cache of recently-visited web pages and recently-downloaded images, used to speed the browsing experience.
3. A database of "cookies" used for such functions as remembering use preferences and speeding log-in to websites that require authentication.
4. Personal information that is used to automatically fill in forms on a web page.

Thus, web browsers are effectively data custodians for a significant amount of personal information. Sometimes the user is made aware of this store of personal information, sometimes the user is not.

Because web browsers are frequently used on computers that are shared by more than one person, it is important for the programs to provide users with the ability to remove this personal information when the user wishes. Even in the case of computers that are not shared, users may still wish to "erase their tracks" under certain circumstances to guard against the possibility that their computer may be analyzed at a later time by another party.

Internet Explorer 6.0 (PC), Apple Safari 1.0 (Mac), and Mozilla 1.4 (Mac) all provide the user with the ability to remove personally-identifiable information. But the three web browsers take very different approaches. Explorer makes it difficult to remove this information, Safari makes it easy, and Mozilla is somewhere in the middle.

Clearing Explorer's browser history requires three steps:

1. The user must click on the "Tools" menu and select the "Internet Options" menu item.
2. The user must click on the "Clear History" button.
3. The user must confirm the question, "Are you sure you want Windows to delete your history of visited Web sites?"

This interface has some significant usability hurdles for an untrained user: the user must know in advance that the "Clear History" button is located on the "Internet Options" panel, the user must realize that having "Windows [to] delete your history of visited Web sites?" is the same as clearing Internet Explorer's history menu, etc. But probably the most significant usability problem is that there is no indication on Internet Explorer's history menu that there is any way to remove this personal information! An unsophisticated user might think that clicking the little "x" in the History menu clears the menu, as it makes the information disappear from the computer's screen. It doesn't: clicking the "History" button in the web browser's button bar makes the History menu reappear.

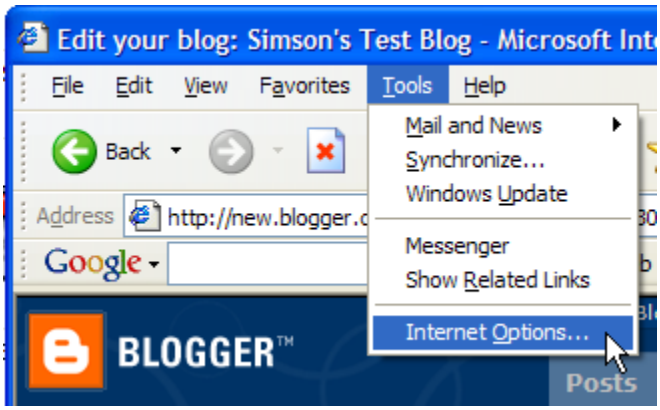

Figure 4: Internet Explorer's 'Clear History' button is confusingly accessed from the browser's "Internet Options" menu.

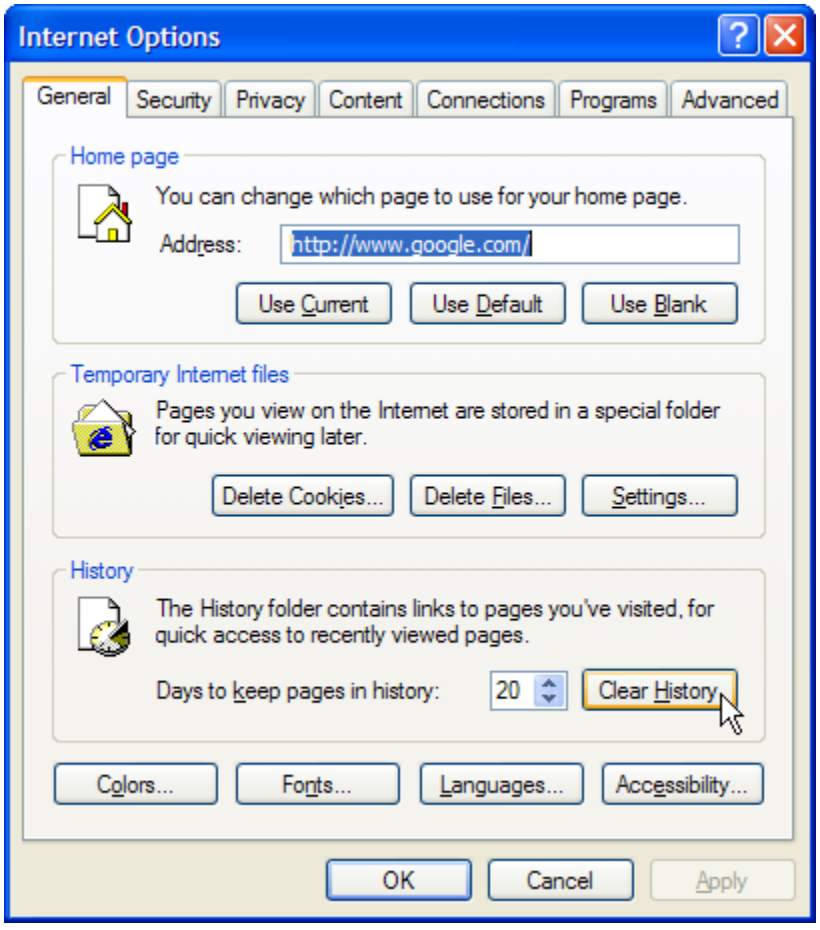

Figure 5: Internet Explorer's Clear History button.

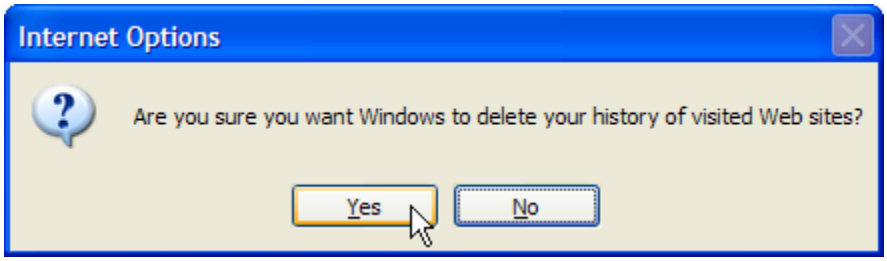

Figure 6: Asking Internet Explorer to Clear History causes this confirmation panel to be displayed.

Safari also has a History menu, but it takes a different approach to clearing the personal information that it contains: at the bottom of the Safari History menu is a menu item clearly labeled "Clear History" (Figure 8). Safari presents the ability to remove personal information where that personal information is displayed.

Explorer's interface offers considerably more power than Safari's. Explorer allows any history items older than a certain age to be removed (although this feature is probably provided out of a concern for tidiness, rather than out of concern for the user's security or privacy). Explorer also offers the ability to selectively remove individual history items by right-clicking on the history item and selecting "Delete."

Mozilla 1.4 on the Macintosh presents history information in two locations: underneath a menu labeled "Go" that appears in the command bar (Figure 9) and in a special History Window (Figure 10). There is no obvious way to erase information from the Go menu, but individual references can be removed from the History window using the standard Macintosh Edit menu. The History can also be from the Preference window (Figure 11).

Both Internet Explorer and Mozilla could be improved by the addition control displayed with the history list that would clear the list.

Another place that potentially confidential information is remembered by the web browser is the browser's search facility. Safari has a search field that allows searching of the Google search engine; a control on the field allows the user to select previous searches. This control also has the option "Clear Entries" at the bottom, another example of providing facilities for sanitizing information where that information is displayed. Internet Explorer does not have a direct analog to the Google search field, but Google the company makes available a free "Googlebar" that also provides for one-click searches. The Googlebar does not not have any obvious way to clear the history (Figure 15. There is, in fact, a "Clear Search History" option, but it is hidden under the "Google" menu (Figure 16).

In fact, both Explorer and Safari have a usability fault that can easily result in a security vio-

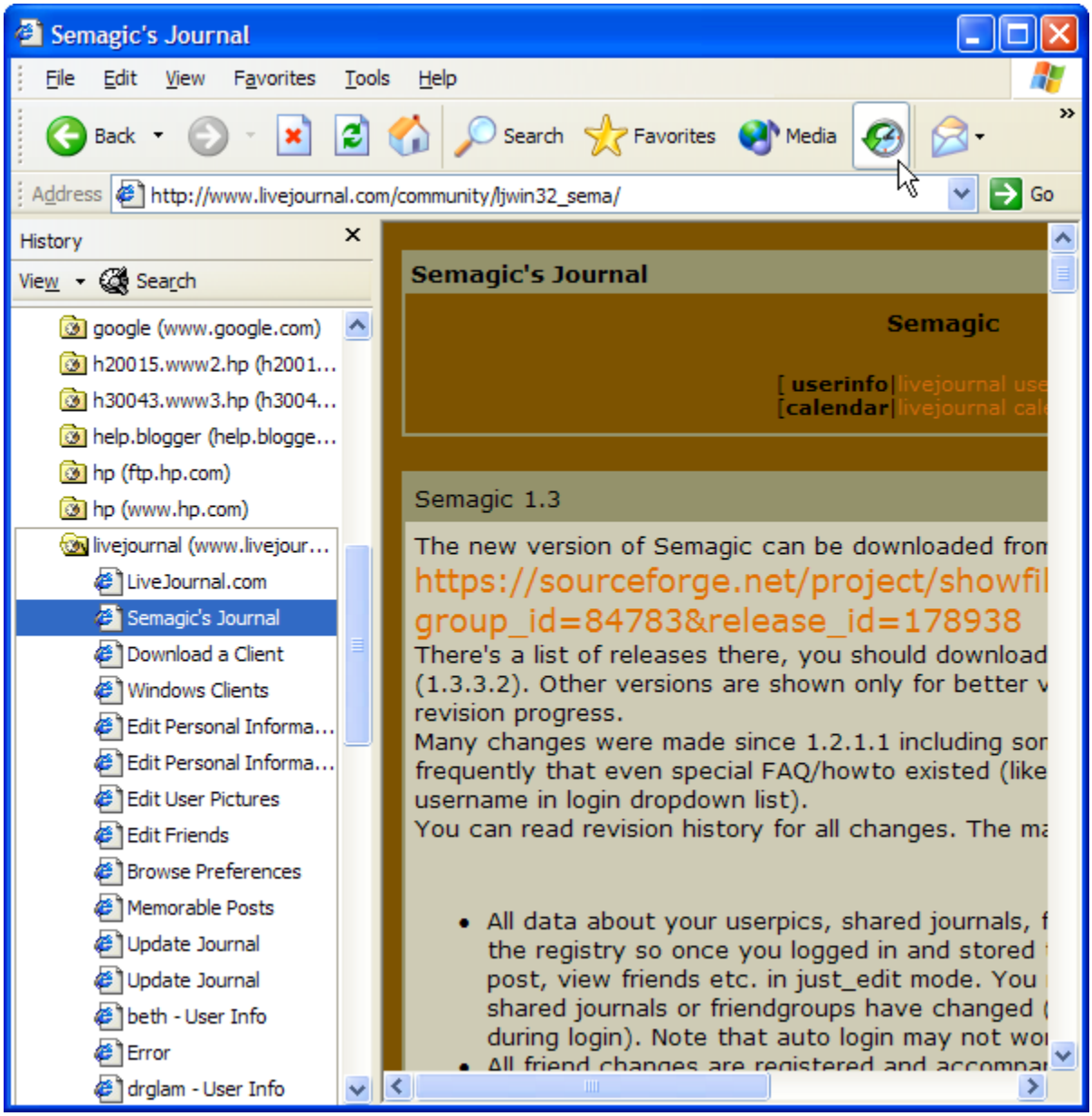

Figure 7: Internet Explorer's History panel appears when the unlabeled history icon in the Standard Buttons bar is clicked; this panel could be improved by placing a "Clear History" button next to the "Search" button.

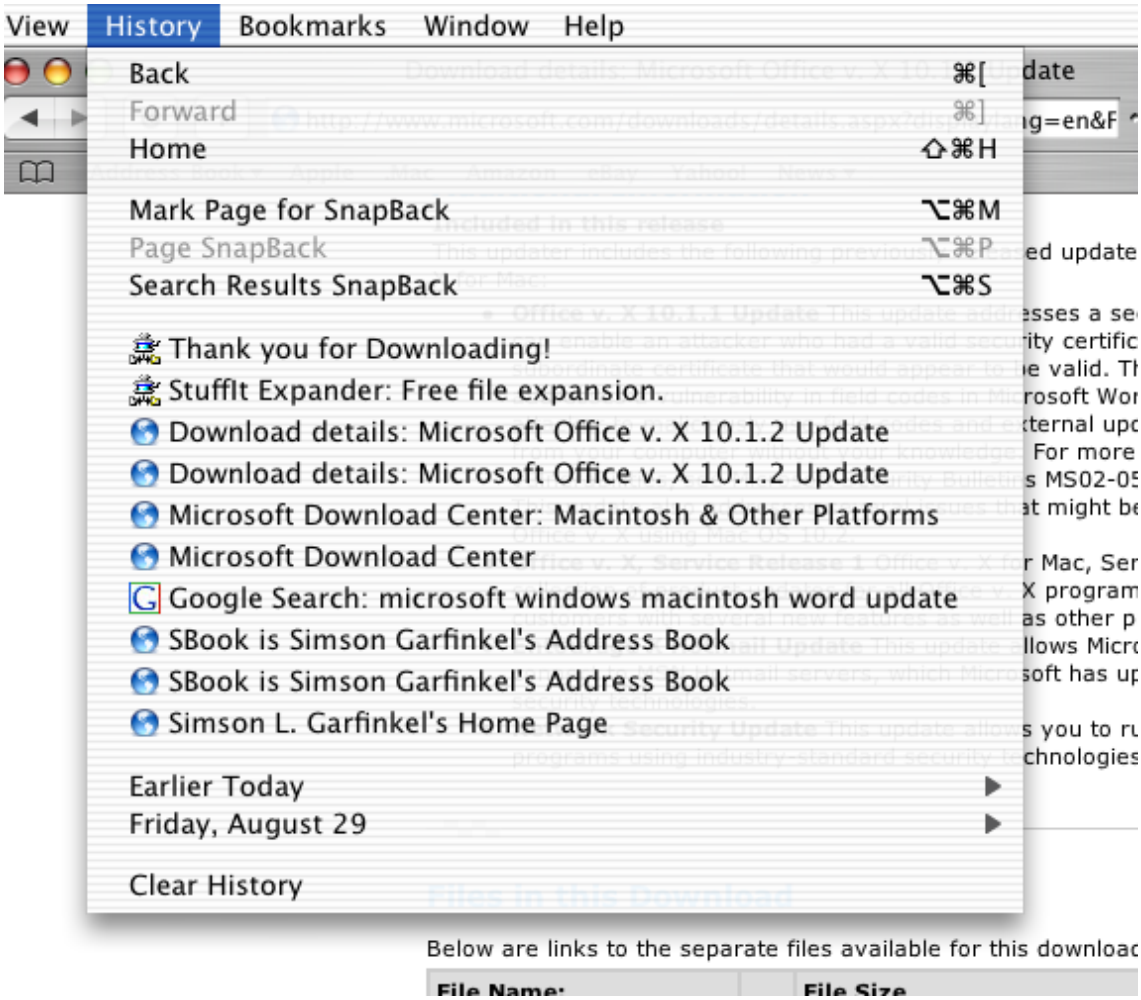

Figure 8: Safari displays a simple “Clear History” menu item at the bottom of the history list.

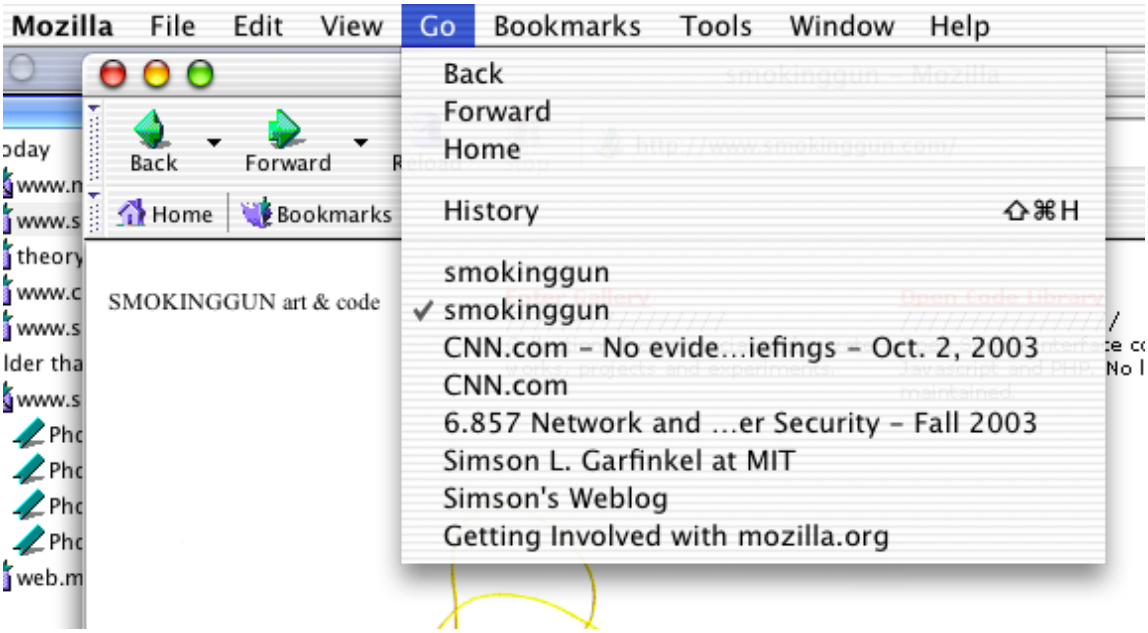

Figure 9: Mozilla 1.4's "Go" menu shows web sites that have been recently visited.

lation when the browsers are used for viewing confidential information: the browsers' "clear history" function does not actually remove the contents of the history from the computer's hard disk — that is, the commands do not erase the browser's web cache. This fault is a violation of my second principle.

Both Explorer and Safari provide the user with a means to clear the browser cache. Explorer has a section called "Temporary Internet files" on the "General" tab of the "Internet Options" panel; in this section there are buttons with labels "Delete Cookies...", "Delete Files...", and "Settings..." These buttons are on the same panel as the "Clear History" button, so perhaps a person who seeks to remove all traces of websites that they have visited will think to delete their "Temporary Internet files" as well.

Safari, by contrast, allows the user to clear the cache with a "Empty Cache..." menu item on the "Safari" application menu. Choose this menu and Safari provides a one-sentence explanation about what the cache does ("Safari saves the contents of web pages you open in a cache so that it's faster to visit them again.") and asks for the user's confirmation.

Finally, Safari has a menu option called "Reset Safari..." which clears the cache, erases the history, and erases all cookies. Within the context of what is possible using the Macintosh 10.2 operating system, this feature appears to work as advertised. In fact, it's quite effective. But because MacOS 10.2 does not implement a secure file deletion function, it is still possible to recover information from the computer's hard drive using forensic utilities.

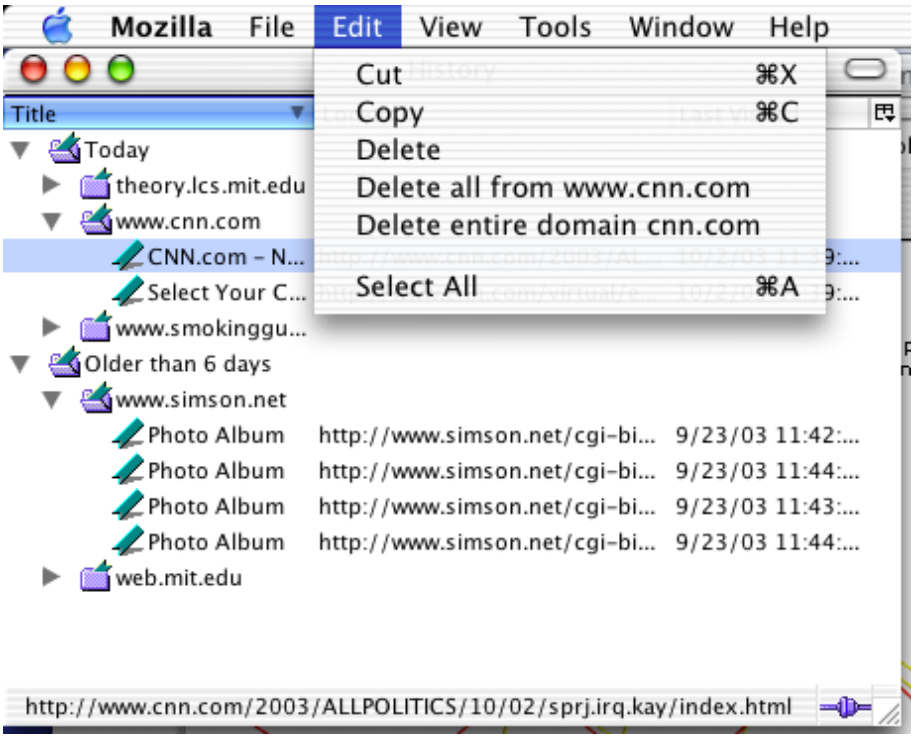

Figure 10: Mozilla 1.4's "History" window shows a more detailed view of the sites that have been visited by the browser. Individual entries in this list can be deleted by selecting the entry with the mouse and then choosing "Cut" or "Delete" from the "Edit" menu. Other options on the edit menu give additional options of deleting all of the web pages from a particular host or domain.

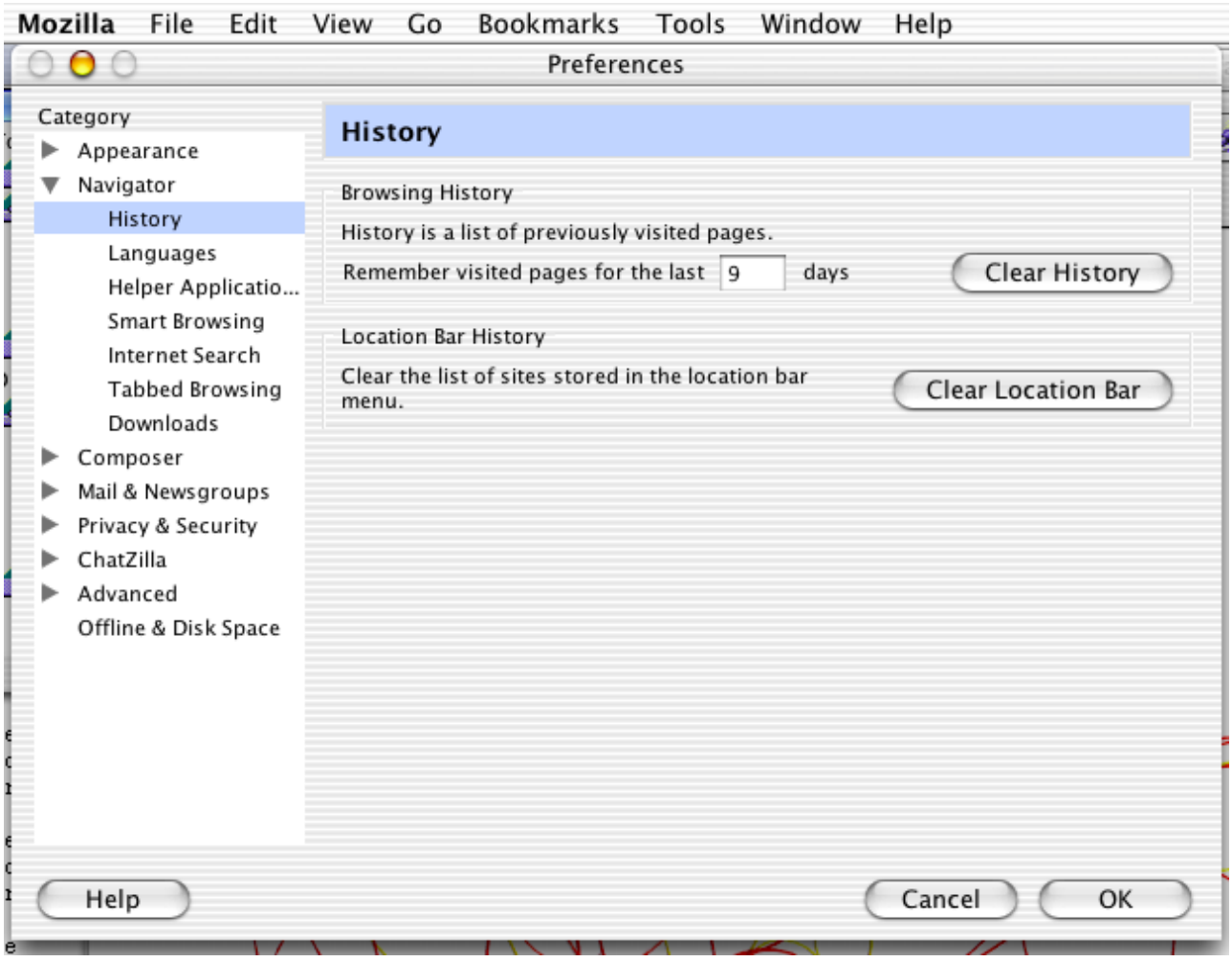

Figure 11: Many Mozilla 1.4 users have been trained to remove history entries through the History pane of the Preferences window.

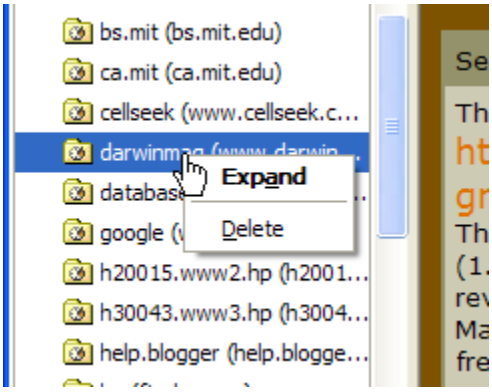

Figure 12: Internet Explorer's History Panel allows individual history links to be deleted with a right-click of the mouse.

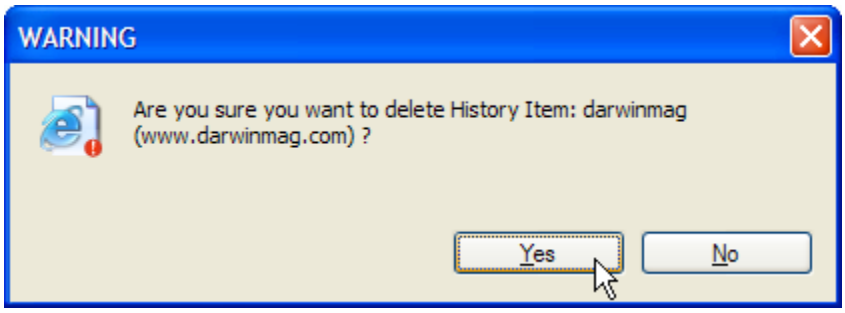

Figure 13: Internet Explorer's history link deletion confirmation panel confusingly shows the internal filename (darwinmag) that is used by the Internet Explorer history mechanism.

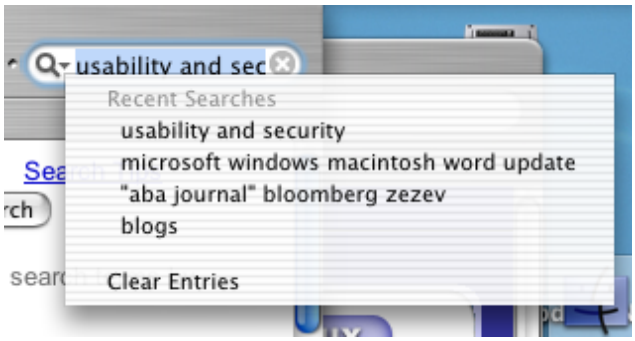

Figure 14: Safari's Google search facility includes a "Clear Entries" menu item along the bottom.

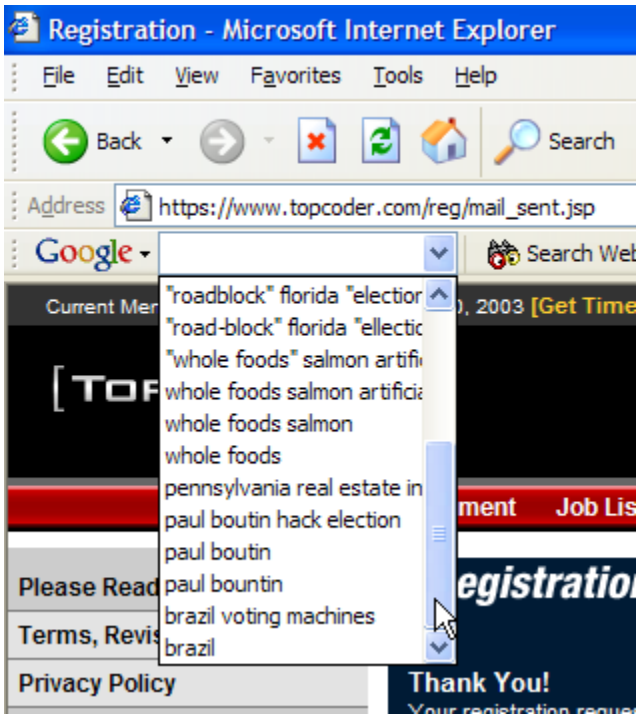

Figure 15: The Googlebar search history provides the user with no obvious way to clear the history—a history that potentially contains confidential information.

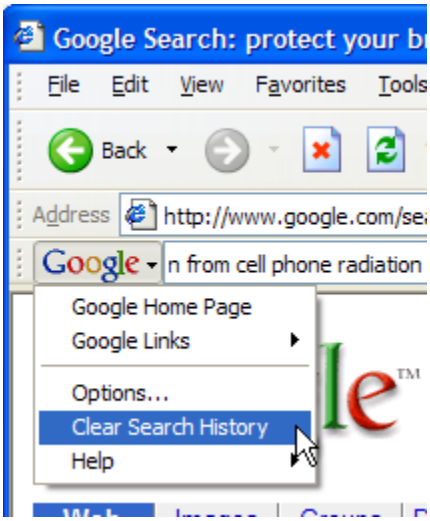

Figure 16: The menu item that clears the Googlebar search history is located underneath the Google menu.

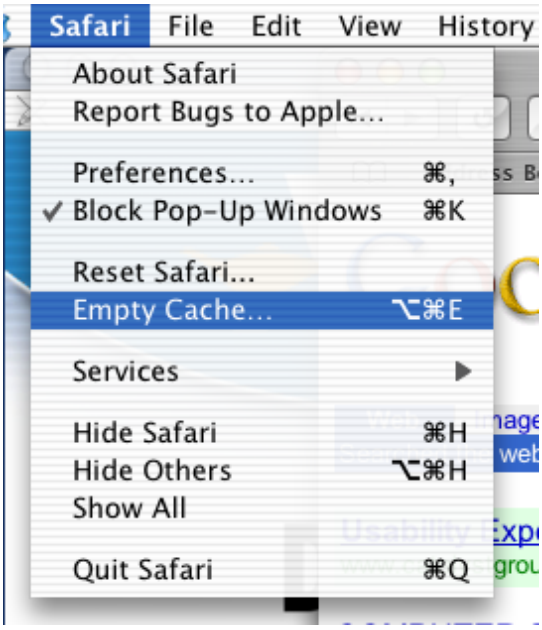

Figure 17: Safari's "Empty Cache" option is located under the "Safari" menu.

Figure 18: Safari's "Empty Cache" confirmation window explains what the cache is used for, but it doesn't explain if clearing the cache is a harmless activity.

```
C:\>help erase
Deletes one or more files.

DEL [/P] [/F] [/S] [/Q] [/A[[:]attributes]] names
ERASE [/P] [/F] [/S] [/Q] [/A[[:]attributes]] names

  names         Specifies a list of one or more files or directories.
                Wildcards may be used to delete multiple files. If a
                directory is specified, all files within the directory
                will be deleted.

  /P            Prompts for confirmation before deleting each file.
  /F            Force deleting of read-only files.
  /S            Delete specified files from all subdirectories.
  /Q            Quiet mode, do not ask if ok to delete on global wildcard
  /A            Selects files to delete based on attributes
  attributes    R  Read-only files            S  System files
                H  Hidden files               A  Files ready for archiving
                -  Prefix meaning not

If Command Extensions are enabled DEL and ERASE change as follows:

The display semantics of the /S switch are reversed in that it shows
you only the files that are deleted, not the ones it could not find.

C:\>
```

Figure 19: Documentation for the Windows XP DEL command

4.3 Case Study #3: Confidential Information on Discarded Media

Sometimes the security vulnerabilities caused by usability problems are not immediately obvious. Such is the case with the file deletion facility that is part of the Microsoft DOS and Windows operating systems.

Both DOS and Windows provide a command line utility called `DEL` to delete one or more files. The command is built-in to the `COMMAND.COM` and `CMD.EXE` shells. Documentation for the command is shown below in 19.

A second technique for deleting files in the Windows environment is to drag an icon representing the file to the Windows "Recycle Bin." Dragging a file to the recycler clearly does not delete the file: the user can double-click on the recycler and view all of the files that have been placed there. But if the user right-clicks on the Recycle Bin desktop icon and selects the menu option "Empty Recycle Bin," the Recycle Bin window empties and it is no longer possible to recover the files that had been in the Recycle Bin.

In fact, even after files are deleted through the Windows interface, it is still possible to recover the contents of these files using a variety of special-purpose tools. These tools range from simple disk recovery software (e.g. Norton Disk Doctor[Sym]), to sophisticated forensics tools costing hundreds or thousands of dollars (e.g. EnCase[Kei03]). No doubt, many users who have accidentally deleted files have rejoiced at the availability of such software to recover their data. But the fact that the tools provided by the operating system do not properly sanitize the computer's hard disk also creates a vulnerability that could be exploited by an attacker.

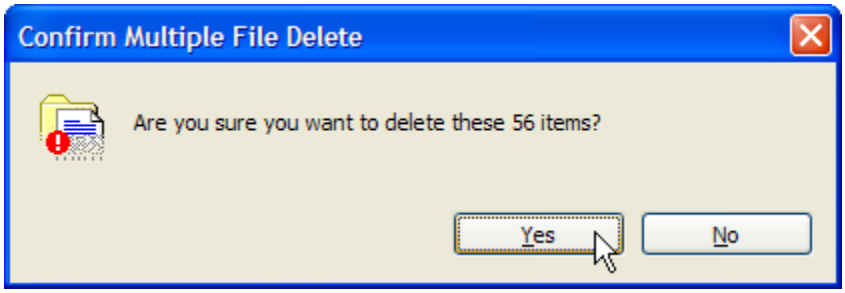

Figure 20: Microsoft Windows XP allows the user to confirm the deletion of files in the Recycle Bin; confirming the operation does not actual remove the contents of the files from the computer's hard disk.

A concise way to restate this vulnerability exists because the Microsoft operating systems violate the *Completion Principle*: the contents of files deleted through the Windows user interface is not properly removed from the computer's hard disk, but can be recovered using a variety of tools.

A simple way to conform to the Completion Principle is for the operating system to make every delete operation on the computer a secure sanitized delete — for example, by following the procedure outlined by the U.S. Department of Defense. [DOD95] This procedure involves overwriting every block of data with a character, that character's complement, and then a block of zeros.

Clearly, there appears to be a conflict between the principle that delete operations should properly sanitize and the desire that actions should be undoable — that is, a conflict between the Completion Principle and the Reversibility Principle. There are several possible approaches for resolving this apparent conflict:

- The operating system can do a poor job at sanitization, so that file recovery after accidental deletion can be accomplished with appropriate forensic tools. Users who require actual sanitization can then use additional tools. This is the approach taken by most of today's operating systems. Experience has shown that the problem with this approach is that many users do not realize that they need to use third-party sanitization tools to actually sanitize their media.

- The operating system can do a good job at sanitization, and force users to rely on their backups to recover files that are inadvertently deleted. One problem with this approach is that, while sanitization can be easily built into most operating systems, backup is typically performed by third-party systems. Thus, it is difficult to assure that the sanitization and backup systems will be properly harmonized.

- The operating system could implement some kind of automatic backup system whereby deleted files wouldn't actually be deleted, but would instead be put into some kind of holding place where they would be automatically deleted and the disk blocks sanitized after a specified amount of time had passed. This would give the user a kind of "event horizon" during which they could change their mind. A user interface could be set up so that the user could force an immediate sanitization.
- The operating system could impose a mandatory waiting period of perhaps one or five minutes on all deletes. This waiting period would give users a chance to reverse their decision. (Norman [Nor88] describes a similar "waiting period" for trash cans, in which each day's trash can is put in a holding room for seven days, before its contents are irretrievably disposed. The result of this system, Norman writes, is that workers are more likely to throw out excess junk in their offices.)

I believe that a combination of the third and fourth approaches is the correct one — that is, systems should provide a unified system for providing both multi-level on disk backups to protect against user error, and to provide for secure file erasure by appropriate operating system modifications. Of course, even with these systems, there is always a point after which some actions cannot be undone. A properly-designed system should make users aware of this point, rather than leaving them guessing.

4.3.1 The Remembrance of Data Passed Study

In January 2000 I started a research project aimed at determining the prevalence of confidential information contained on used computers and hard drives being sold on the secondary market. Between January 2000 and August 2003 I purchased 158 disk drives on the secondary market. Most of the drives were purchased from the eBay online auction service. Other drives were purchased at swap meets and from computer stores specializing in "used" or "surplus" equipment.

Once purchased, each drive was cataloged and then imaged using the Unix `dd` command to a disk file. At this point a significant portion of the drives were found to be non-functional, leading one to the conclusion that the secondary market is not a good place to buy reliable disk drives. Of the drives that were operational, an attempt was then made to mount each one using the FreeBSD `mount_msdos`, `mount_nfts`, and `mount_ufs` commands. Drives that could be mounted were then copied a second time using the Unix `tar` command.

The resulting information was analyzed using a variety of manual and automatic techniques. In particular, I wrote a program called a forensic file system that could be used to rapidly distinguish between files in the native file system, files in temporary directories, files that had

Level	Where Found	Description
Level 0	Regular Files	Information contained within the file system.
Level 1	Temporary Files	Temporary files, including print spooler files, browser cache files, files for "helper" applications, files in "recycle bins." Level 1 is a proper subset of level 0.
Level 2	Deleted Files	When a file is deleted from a file system, most operating systems do not overwrite the blocks on the hard disk on which the file is written. These files can be recovered using "undelete" tools.
Level 3	Retained data blocks	Data that can be recovered from a disk but which is not referenced as the contents of a deleted or undeleted file.
Level 4	Vendor-hidden data	Data on the drive that can only be accessed using vendor-specific commands. This level includes the drive's controlling program and blocks used for bad-block management.
Level 5	Overwritten data	Many individuals maintain that information can be recovered from a hard drive even after it is overwritten. Level 5 is reserved for such information.

Table 1: The Garfinkel-Shelat Sanitization Taxonomy

been deleted with the Windows "del" command but whose information was still available, and information on the disk that was not obviously a part of any file, deleted or undeleted. Abhi Shelat, a member of the LCS Theory Group, wrote an analysis program that rapidly scanned for email addresses and credit card numbers. The results of our analysis indicated that between one third and one half of the hard drives contained confidential information. In many cases, attempts had been made to remove this information, but it was still present. Our findings were published in [GS02].

As part of this project, Shelat and I created a taxonomy to describe the types of data that can be recovered using traditional and forensic techniques. That taxonomy appears in 1.

There are an increasing number of cases in which confidential information has been discovered on computer systems that were discarded or repurposed (e.g., [Mar97, Vil02, Has02, Lym01]). Interestingly, the vast majority of cases reported in the press involve Level 0 and Level 1 data, rather than Level 2 or Level 3. That is, in most cases the new owner of a computer or hard disk only becomes aware of the disk's contents because the files are in plain view. With the exception of the Garfinkel/Shelat study [GS02], there are no reported cases of confidential data being discovered through the use of forensic tools.

4.4 Case Study #4: How the PGP and S/MIME Encryption Standards have Produced Usability Problems that Blocked Their Own Adoption

Despite the widespread availability[9] of software that implements email security, most observers concur with the observation that there has been poor adoption of this technology by users.

In their paper "Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0," [WT99] Whitten and Tygar argue that despite having an attractive and professional user interface, fundamental problems in the design of PGP 5.0 have been responsible for the poor adoption of the cryptographic protocol. Specifically, Whitten and Tygar argue that the PGP user interface provides tools but not explanations. It gives the user capabilities for creating and distributing keys, but it does not explain to the user why it is necessary to do so. The pair's CHI2003 workshop presentation [WT03] and Whitten's unfinished doctorial thesis [Whi03b], introduce a technique called *safe staging* "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." [Whi03a]

Essentially, Whitten has developed an interface prototype that teaches people the ins and outs of public-key cryptography, including the tasks of key generation, key distribution, and key signing.

I believe that there are two unrelated reasons for the market failure of email encryption. The first is almost certainly software usability: as Whitten and Tygar point out, security interfaces must be judged by different standards than traditional user interfaces. I believe that this usability failure results, in part, from the adoption of the MIME standard to the purpose of email security. But I believe that a second reason for the failure of these systems to catch on is that they place high burdens on non-participants.

Both the PGP [Elk96] and S/MIME [DHR+98] standards for digitally-signed and encrypted email build upon the MIME standard [BF93] in a similar way:

- Messages that are encrypted are sent as a single MIME multipart. The MIME standard provides for unambiguous encoding of 7-bit text data (in the case of PGP) or 8-bit binary data (in the case of S/MIME).
- Messages that are digitally signed without encryption are sent as two MIME multiparts: the first part for the message, a second part for the signature.

[9] Support for the S/MIME encryption standard is available for free to every user of Windows 95 and above, as the technology is built into Microsoft's Outlook Express email client. Support for S/MIME is also built into Microsoft Outlook and Netscape Communicator. Support for PGP is available for Qualcomm's popular Eudora program. Users of other email systems can download free implementations of PGP from a variety of websites.

Note that because MIME only applies to message bodies, and not to RFC822 message headers, neither PGP, PGP/MIME nor S/MIME provide security for the Subject: line of email messages. This is unfortunate, as Subject: lines frequently contain information that one might wish to protect with a digital signature or obscure with an encryption algorithm. Nevertheless, there are no email interfaces that the author is aware of convey to the user that the Subject: line is not encrypted or signed, not even when encrypted messages are being sent. Users must instead be educated to this fact.

A second problem with both the PGP/MIME and S/MIME standards arise when the sender is sending out digitally-signed messages and the recipient does not have conformant software. Consider the case of a digital-signed PGP/MIME message, such as the message shown in 21. In some cases[10], this results in the recipient being presented with an empty message that has two attachments: one for the message, and one for the message signature. In other cases[11], the user is presented with a legible email message that is accompanied by an indecipherable attachment — attachments that build up in the user's "Attachments" directory. In yet other cases[12], the recipient of an S/MIME message is presented with a warning that the sender of the S/MIME message has used a client certificate that is untrusted. All of these cases, and many more, result in burdens for non-users, a violation of the *No External Burden* principle.

It is my experience that interaction burdens of the type mentioned in the previous paragraph invariably have the same result: the recipient of the email message asks the sender to stop using email security. And in many cases, the sender does stop.

I believe that the violation of the *No External Burden* Principle is one of the primary reasons for the failure of the PGP/MIME and S/MIME standards: non-users cause push-back on users, effectively blocking adoption. This is a kind of anti-network-effect. The mistake here is the use of the message attachment standards for the purpose of providing message security. Although such encapsulation seems to make sense from a programmer's point of view — encapsulation and code-reuse are both tried-and-true principle of computer science — in this case the result of such encapsulation is undesirable because of their impact on non-participants.

One way to prevent push-back from non-participants is for a sender who wishes to send digitally-signed email to remember which of her recipients are able to receive and properly-process digitally signed email, and which cannot. Such information could be readily kept in the mail application's address book. This approach is not entirely satisfactory, unfortunately, as there is no obvious way for the sender's application to determine which recipients can properly process digitally-signed email other than by having a database of applications and trying

[10] Specifically, if digitally-signed PGP/MIME message is sent to a user who reads her mail with Microsoft Outlook Express

[11] Specifically, if the sender of the message uses S/MIME and the recipient uses Eudora

[12] Specifically, if the sender of an S/MIME message digitally signs their message with a certificate for which the recipient does not have the corresponding Certificate Authority key.

```
Return-Path: <jis@mit.edu>
Delivered-To: simsong@r2.nitroba.com
Received: from jis.mit.edu (JIS.MIT.EDU [18.7.21.84])
        by r2.nitroba.com (Postfix) with ESMTP id EA8BBE44342
        for <simsong@nitroba.com>; Mon, 11 Nov 2002 00:24:26 -0500 (EST)
Received: from jis.tzo.com (localhost [127.0.0.1])
        by jis.mit.edu (8.9.2/8.9.3) with ESMTP id AAA16802;
        Mon, 11 Nov 2002 00:24:24 -0500 (EST)
Received: from mit.edu (jis.tzo.com [127.0.0.1])
        by jis.tzo.com (8.12.3/8.12.3) with ESMTP id gAB5OJQG002477;
        Mon, 11 Nov 2002 00:24:20 -0500
Message-ID: <3DCF3EFC.7010007@mit.edu>
Date: Mon, 11 Nov 2002 00:24:12 -0500
From: "Jeffrey I. Schiller" <jis@mit.edu>
Organization: Massachusetts Institute of Technology
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.0.0) Gecko/20020607
X-Accept-Language: en-us, en
MIME-Version: 1.0
To: "Simson L. Garfinkel" <simsong@nitroba.com>
Cc: simsong@lcs.mit.edu
Subject: Re: forensics
References: <20021111041442.A9B5CE44325@r2.nitroba.com>
X-Enigmail-Version: 0.63.3.0
X-Enigmail-Supports: pgp-inline, pgp-mime
Content-Type: multipart/signed; micalg=pgp-sha1;
 protocol="application/pgp-signature";
 boundary="------------enig8C7ABE5DD4CCB084DAF76091"

This is an OpenPGP/MIME signed message (RFC 2440 and 3156)
--------------enig8C7ABE5DD4CCB084DAF76091
Content-Type: text/plain; charset=us-ascii; format=flowed
Content-Transfer-Encoding: 7bit

Let's shoot for 3pm on Tuesday the 3rd...

                        -Jeff

Simson L. Garfinkel wrote:
> Okay. How is your first week in December?
> I can do:
>
> Breakfast/morning/Lunch on Monday the 2nd or Wednesday the 4th.
> Breakfast/tea or lunch at 3 on Tuesday the 3rd or Thursday the 4th
> All day on Friday the 6th


--------------enig8C7ABE5DD4CCB084DAF76091
Content-Type: application/pgp-signature

-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1.0.7 (GNU/Linux)
Comment: Using GnuPG with Mozilla - http://enigmail.mozdev.org

iD8DBQE9zz8D8CBzV/QUlSsRAkskAJ9hyNsrChynbJMOhhbVTFMGGR7djwCfauN+
srszb5+CMJHH12pUWwMSLug=
=vFYk
-----END PGP SIGNATURE-----

--------------enig8C7ABE5DD4CCB084DAF76091--
```

Figure 21: A PGP-signed message created by Jeff Schiller with Mozilla. Note that the message is a MIME multipart/signed message with two parts: one that is type text/plain and one that is type application/pgp-signature. Some mailers will display this message as a single message and an attachment; other mailers will display this message as two attachments.

to infer each recipient's e-mail application from an examination of e-mail headers. And even *this* approach has problems: a recipient might read their email on *two* different computers — one that properly supports PGP/MIME, one that does not. (For example: the user might use a Macintosh laptop but have a desktop running Windows XP.)

I have come up with an alternative solution for digitally-signed email. The advantage of my solution is that it should not have a usability push-back problem because it does not impose a burden or cost on non-participants. The disadvantage of my solution is that it is not directly interoperable with either PGP/MIME or S/MIME implementations. However, messages in my format can be readily interconverted with the standards.

My solution is a new representation for PGP-signed email that is readily converted to and from the PGP/MIME format. This format is demonstrated in 22.

The key elements of this new format are:

- Instead of including the signature as an attachment, this format includes the signature as a header to the first MIME part.
- Instead of presenting the receiving mailer with the multipart/signed content type, which the mailer may be unable to properly process, this format preserves the original content type of the original message.
- This new format does not violate the MIME standard — the messages are valid MIME messages — but it does not follow the existing PGP/MIME standards. As a result, today's email systems cannot verify these PGP signatures. But at the same time, existing email systems are not troubled by these signatures.
- This new format can be readily converted back to the standard PGP/MIME format using either relatively simple programs or even a text editor. As a result, it is possible for a person to send out messages that are digitally signed using this format and to also distribute software for verifying those signatures, should the recipient ever do so.

What is the use of sending out messages that are digitally signed, but having them signed in a format that is not readily verifiable? I argue that the advantages are the same as sending out messages that are digitally signed with existing digital signature methods. In the event of a disputed authorship, the signature can be manually verified. In the normal case, the signature can be ignored.

Because of the burden that digitally-signed mail places on recepients, today little incentive for individuals or organizations to digitally sign all of their outgoing mail as a matter of course. However, if this usability problem could be overcome, digitally-signed mail could be a great help in solving the problem of distinguishing junk email (e.g. "spam") from non-junk email.

```
Return-Path: <jis@mit.edu>
Delivered-To: simsong@r2.nitroba.com
Received: from jis.mit.edu (JIS.MIT.EDU [18.7.21.84])
        by r2.nitroba.com (Postfix) with ESMTP id EA8BBE44342
        for <simsong@nitroba.com>; Mon, 11 Nov 2002 00:24:26 -0500 (EST)
Received: from jis.tzo.com (localhost [127.0.0.1])
        by jis.mit.edu (8.9.2/8.9.3) with ESMTP id AAA16802;
        Mon, 11 Nov 2002 00:24:24 -0500 (EST)
Received: from mit.edu (jis.tzo.com [127.0.0.1])
        by jis.tzo.com (8.12.3/8.12.3) with ESMTP id gAB5OJQG002477;
        Mon, 11 Nov 2002 00:24:20 -0500
Message-ID: <3DCF3EFC.7010007@mit.edu>
Date: Mon, 11 Nov 2002 00:24:12 -0500
From: "Jeffrey I. Schiller" <jis@mit.edu>
Organization: Massachusetts Institute of Technology
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.0.0) Gecko/20020607
X-Accept-Language: en-us, en
MIME-Version: 1.0
To: "Simson L. Garfinkel" <simsong@nitroba.com>
Cc: simsong@lcs.mit.edu
Subject: Re: forensics
References: <20021111041442.A9B5CE44325@r2.nitroba.com>
X-Enigmail-Version: 0.63.3.0
X-Enigmail-Supports: pgp-inline, pgp-mime
Content-Type: multipart/alternative; boundary="------------enig8C7ABE5DD4CCB084DAF76091"

This is a MIME message, created Stream MIME package version 0.21
--------------enig8C7ABE5DD4CCB084DAF76091
Content-Type: text/plain; charset=us-ascii; format=flowed
Content-Transfer-Encoding: 7bit
X-Content-Type: multipart/signed; micalg=pgp-sha1;
 protocol="application/pgp-signature";
 boundary="------------enig8C7ABE5DD4CCB084DAF76091"
X-PGP-Sig-1: -----BEGIN PGP SIGNATURE-----
X-PGP-Sig-2: Version: GnuPG v1.0.7 (GNU/Linux)
X-PGP-Sig-3: Comment: Using GnuPG with Mozilla - http://enigmail.mozdev.org
X-PGP-Sig-4:
X-PGP-Sig-5: iD8DBQE9zz8D8CBzV/QUlSsRAkskAJ9hyNsrChynbJM0hhbVTFMGGR7djwCfauN+
X-PGP-Sig-6: srszb5+CMJHH12pUWwMSLug=
X-PGP-Sig-7: =vFYk
X-PGP-Sig-8: -----END PGP SIGNATURE-----

Let's shoot for 3pm on Tuesday the 3rd...

                        -Jeff

Simson L. Garfinkel wrote:
> Okay. How is your first week in December?
> I can do:
>
> Breakfast/morning/Lunch on Monday the 2nd or Wednesday the 4th.
> Breakfast/tea or lunch at 3 on Tuesday the 3rd or Thursday the 4th
> All day on Friday the 6th


--------------enig8C7ABE5DD4CCB084DAF76091--
```

Figure 22: The same PGP-signed message as appeared in 21, but modified by Stream to have a low-burden signature.

Today's PGP/MIME and S/MIME standards impose a significant burden. My new email format does not. By not violating the *No External Burden* Principle, I believe that this format promotes email that is both secure against spoofed senders (a symptom of spam) without imposing a usability cost.

4.5 Case Study #5: Why "Decrypt for Display" Has Caused Barriers To Adoption

Both PGP-based and S/MIME-based email clients share a common implementation strategy with regards to the treatment of received encrypted mail: these programs decrypt the mail for viewing, but they leave the email messages encrypted in the mail store. The reason for this decision appears to be largely historical: the original developer of PGP, Phil Zimmermann, read his email on a shared multi-user computer. The only way that he could provide for email security was to leave the mail encrypted on the mail store; when Zimmermann wished to read the contents of an encrypted mail message, he could download the message to his computer, decrypt it with PGP, and view it.

This design decision has been copied over time. Practically all systems deployed today leave messages in the mail store encrypted, and simply decrypt the messages when necessary for view.

This policy is not appropriate for today's computing environments:

1. Presumably, the email messages are left encrypted in the user's mail store to provide them with a higher level of security than they would have if they were not encrypted. The sender of the email message might want to send an encrypted message to guard against negligence on the part of the recepient, but why should such protections be limited to email that the sender deemed sensitive? Why shouldn't other email messages in the user's mail store have similar levels of security? And if a mechanism is devised to assure the security of those other email messages, why couldn't that same mechanism be used for mail that was sent with encryption?

2. Other information on the user's hard drive arguably needs to have a similarly high-level of security to assured the continued security of the user's email store. Otherwise, the added security is illusionary. For example, the user's address book must be properly secured, or else when the user thinks that he is sending email to one address, the email might actually go to another. Likewise, the user's applications and operating system require a high level of security, otherwise they might be modified and the user's passphrase and cryptographic keys stolen.

3. Leaving encrypted mail encrypted to the recipient's public key means that the recipient of the encrypted email messages needs to retain her private key as long as she wishes to retain access to the stored electronic mail. In practice, this creates another security vulnerability.

4. Many organizations demand the ability to review the mail that their employees have received. This creates the need for key escrow of the user's private key — a key that is used for many other purposes.

5. Encrypted email cannot be searched with a full-text search system unless the entire document is previously indexed. If the document is indexed, much of the security provided by the encryption is lost unless the index is, in turn, encrypted.

I believe that users and organizations are aware of these disadvantages and that they are also part of the reason that encrypted email has failed in the marketplace. Placed within the theoretical framework presented at the beginning of this section, leaving the email store encrypted violates the *Zero Impact* Principle: email encryption is optional and, when it is used, the resulting email messages are less useful than the corresponding mail messages sent without encryption.

A far better alternative to leaving the email in the email store encrypted with the original encryption key is to decrypt the email when it is received and then apply disk-based encryption to the entire email store, the user's address book, and other aspects of the operating system.

4.6 Case Study #6: How The Distribution of Public Keys Causes Usability Problems, and Approaches For Resolving The Problem

Although digitally-signed email is useful for addressing the problem of spam, it has long been believed by network researchers that there is a demand on the part of users for mail that is sealed with an encryption function to prevent accidental reading by an unauthorized recipient. [13]. Nevertheless, despite the fact that both the PGP and S/MIME mail standards provided for sealed email, and despite the fact that software that implements these standards is widely distributed, sealed email remains a tiny minority of Internet message traffic.

I believe that the root cause for the failure of digitally-sealed mail is the same as the failure of digitally-signed mail: the PGP and S/MIME standards, as a side-effect of intermixing email security with the MIME standard, invariably place an unacceptable burden on non-users. This

[13] RFC822 specifies an optional "ENCRYPTED" header, noting "Sometimes, data encryption is used to increase the privacy of message contents." [Cro82] Prior to RFC822, encrypted Internet messages were envisioned by RFC 753 [bP79] in 1979.

burden, in turn, causes push-back against the would-be users of the email security systems. However, I believe that the mechanism for this burden is significantly different.

There exist several email systems that provide for mostly-transparent encryption of mail that is being sent and the mostly-transparent decryption of encrypted email that is received. Among these are Microsoft's Outlook Express, which can be configured to automatically decrypt encrypted mail that is received, and to automatically encrypt mail that is sent in response to a received encrypted message. GPGMail is a plug-in for the Macintosh MacOS X Mail.app mail client that provides for transparent decryption of encrypted mail and the automatic encryption of outgoing mail.

The problem faced by users of Outlook Express, GPGMail, and other transparent email encryption systems is that of key distribution: there is no obvious way for a user to programmatically determine if the recipient of a message has an encryption key.

Consider these cases:

- Alice and Bob are both PGP users, but since Alice has never told Bob that she is a user, he doesn't know.
- Bob may know that Alice has created a key, but he doesn't know whether it is easy or difficult for Alice to receive PGP-encrypted messages. As a result, Bob doesn't encrypt the email messages that he sends Alice.
- Bob's PGP software may automatically check the PGP key servers to see if there is a key matching Alice's email address. But even if Bob's software finds a key, he doesn't know if he can use it. Alice may have lost her private key after publishing the public key onto the web server. Alternatively, some other user may have published a key with Alice's public key onto the PGP key server.

Bob could simply ask Alice if she uses PGP and, if so, would she like him to encrypt the email that he sends her. But since only a minority of Internet users have PGP today, it is likely, for any given Alice that Bob asks, that the answer to his question will usually be "no." Following the *No Burden* Principle, Bob will eventually stop asking his correspondents if they use PGP.

Early in the deployment of PGP, one way that users of the program attempted to avoid this problem was by printing their PGP public key fingerprints upon their business cards. [Gar94] Such notations served two purposes:

- The business card with the fingerprint told all potential electronic correspondents that the business card's author knew what PGP was about and had the ability to receive PGP-encrypted email that was encrypted with a particular key.

- The business card provided a kind of validation of the person's key. The fingerprint on the business card didn't prove that the name on the key (and presumably on the business card as well) was factually accurate, but the fingerprint on the business card gave the recipient of the card a strong indication that the key whose fingerprint was on the card actually belonged to the person who was handing out the card. After all, why would a person hand out business cards that with someone else's key fingerprint?

Of course, handing out business cards is vulnerable to a man-in-the-middle attack. If instead of giving his business card to Alice, Bob gives his business card to Eve and asks that Eve hand the card to Alice, then Eve will have an opportunity to substitute Bob's business card with a card that looks the same but has a different PGP fingerprint — one corresponding to a key that she controls — and a different email address. Eve can then intercept all messages that Alice sends to this second email address, decrypt them with her key, and re-encrypt them with Bob's public key.

Two solutions to this sort of active attack are listed below. The first relies on cryptography, the second relies on biometrics:

1. Bob can put his key on the PGP key server and make sure that his key is digitally signed by someone whom Alice trusts. If Alice then goes to the PGP key server to download Bob's key, she will discover two keys: Bob's true key, and the one planted there by Eve. Discovering two keys, Alice will be able to use the digital signature to distinguish between the one that is truly Bob's and the one that is not.

2. Alice can telephone Bob and ask him for his key's fingerprint. She can then compare the fingerprint that Bob gives her with the one on the card — and with the one on the key that she has downloaded. This approach relies on Alice being able to recognize Bob's voice.

Notice that Bob *cannot* simply attach a copy of his photograph to his PGP key and digitally sign it himself, since Eve would be able to repeat this process with a copy of Bob's photograph and her own key.

At the Eleventh Conference on Computers, Freedom and Privacy in Cambridge, MA, I suggested that a simple way to solve the PGP key distribution problem would be for people to put their PGP public key in the email header of their outgoing mail messages. Although this approach would be susceptible to the same sort of active attack mentioned above, such attacks could be defeated through either key signing or through biometric verification. During the spring of 2003 I started working on this proposal, and have developed two places in the email header where a public key can be placed. Figure 23 shows a simple text message generated

```
Date: Mon, 20 Jan 2003 07:39:34 -0500
Subject: This is a test message
Content-Type: text/plain;
        delsp=yes;
        charset=US-ASCII;
        format=flowed
Mime-Version: 1.0 (Apple Message framework v551)
From: Simson L. Garfinkel <simsong@lcs.mit.edu>
To: Simson L. Garfinkel <simsong@ex.com>, simsong@mit.edu, simsong@lcs.mit.edu
cc: simsong@lcs.mit.edu
Content-Transfer-Encoding: 7bit
In-Reply-To: <3.0.5.32.20030120072125.00dfde30@pop3.ex.com>
Message-Id: <3ACA2842-2C74-11D7-B00F-00039303C716@lcs.mit.edu>
X-Mailer: Apple Mail (2.551)

This is a test message.
Just a test.
Okay, a test.
```

Figure 23: A simple text message

```
Date: Mon, 20 Jan 2003 07:39:34 -0500
Subject: This is a test message
Content-Type: text/plain;
        delsp=yes;
        charset=US-ASCII;
        format=flowed
Mime-Version: 1.0 (Apple Message framework v551)
From: Simson L. Garfinkel <simsong@lcs.mit.edu>
To: Simson L. Garfinkel <simsong@ex.com>, simsong@mit.edu, simsong@lcs.mit.edu
cc: simsong@lcs.mit.edu
Content-Transfer-Encoding: 7bit
In-Reply-To: <3.0.5.32.20030120072125.00dfde30@pop3.ex.com>
Message-Id: <3ACA2842-2C74-11D7-B00F-00039303C716@lcs.mit.edu>
X-Mailer: Apple Mail (2.551)
x-mailer-key-01: -----BEGIN PGP PUBLIC KEY BLOCK-----
x-mailer-key-02: mQGiBD7FLSoRBADaaxWjYtqr/2YMU9SMEf71onn+ygV06T4Tuw9t
x-mailer-key-03: o6M8nXRkl+EHWUsZBPtkrVbw76RIV1f1GSSuC/DQ5dVQhi2FpwDM
x-mailer-key-04: EM//LaoD6VSvfPuxvWsoxAxYC1rVyqHcyTc3GUfGHQ1ZVlYMxyNW
x-mailer-key-05: rUgSHq0obnUFPEyvKrAgoWr7BuNVvwCgk7HoXy3xr0vlItgZaAmL
x-mailer-key-06: Q1Wuo58D/1LQzZ9R6qej7lXOnsaf1uxfLbEmuDiyXgTHF2WXASKh
x-mailer-key-07: MfClsHNMgOoJcqc9K9uOB21KBpluZeo/682b7283zEXaT28kqfE2
x-mailer-key-08: KxoCSvGOROuoSLLx5AiiS1uG0Yt4kccUADhGCOqB2D4XBl+laYMh
x-mailer-key-09: mdYYGb42AaO0fWTLL93xvwOmBACJ/frOaxAP976KQB6h3ssSOyk4
x-mailer-key-10: 6XmeH2kJCt/xaqoIEWFTQFMV6MwDo7bZXiWPCu9OMYtUNTS+z6Vk
x-mailer-key-11: +uMbhfNfFcOi1BRj5+ozot+wpOewiM3Ixeh7f4Y+/p7KA7mnMLFt
x-mailer-key-12: r427MqU118Kk/9HfBurhAHg67HHrjsEkHytXE/fT77QpU2ltc29u
x-mailer-key-13: IEwuIEdhcmZpbmtlbCA8c2ltc29uZ0BsY3MubWl0LmVkdT6IWQQT
x-mailer-key-14: EQIAGQUCPsUtKgQLBwMCAxUCAwMWAgECHgECF4AACgkQRjQQsgVD
x-mailer-key-15: Bg/nOACfajuZ8TLlLXVO/0o2lIjdpRAMtcwAmwYE8vNR+euaaNGL
x-mailer-key-16: it/lC5/EPluLuQENBD7FLTEQBACeDZR92l+hTBwrH8OzHed+eGRz
x-mailer-key-17: 9cP4n8W1BWf9J+EsixLsSbMMhz6Zuamyq5UMhNi21zIpE1uICYzU
x-mailer-key-18: pAz6shFjpQg6PMCMHFoX8ygZmv9eZ5fYXmvgAckXMasO6LMb/1l4
x-mailer-key-19: Bqy3ju6fg+3AzWhhdSPBnud6V/oarCVmBzOSp9AoWwADBgP/cwtL
x-mailer-key-20: VJm3w6sa9T8IoXuRPYexqlHG4cBE3ayseJsBu+OlEzt7xR5i4K/s
x-mailer-key-21: aTVjWJyquW8J1z260aL8yV+foGmZh2yz2YYnwPHBTCG/C27sp+jk
x-mailer-key-22: Ftt7jsyNrA978svuUThlyAaAYEndV8XufxIwIFqeI872tZIzqlv2
x-mailer-key-23: l/aNgrOJsd+IRgQYEQIABgUCPsUtMQAKCRBGNBCyBUMGD/8TAJ9E
x-mailer-key-24: YvcmiGtR98sQ7lF4aSZQ6h38sQCePg3S/AC9OfAQFudw28hHGbmn
x-mailer-key-25: FP8==rk7U
x-mailer-key-26: -----END PGP PUBLIC KEY BLOCK-----

This is a test message.
Just a test.
Okay, a test.
```

Figure 24: A simple test message that has a key added

with the Apple Mail client; Figure 24 shows how this message is modified to have the sender's public key added. A paper describing this technique was published in [Gar03].

There are several advantages to this approach:

- Keys distributed in this manner pose no burden on people who receive the message but lack the software to process it.[14]
- The key is easily incorporated into the database of an email encryption system using the appropriate software.
- The presence of the key indicates that the sender is willing and able to receive an encrypted message — using this very key.
- If a user changes their key, the change is evident to their correspondents's mail software when the next message is received.

As previously noted, the problem with this approach is it is susceptible to an active attack. Fortunately, the key-signing and biometric verification approaches discussed earlier in this section provide a technique for defending against these attacks.

One problem with the approach outlined above is that the popular Sendmail [CA02] e-mail system limits the size of RFC822 mail headers to 32,767 bytes. This limit can be overcome with MIME-formatted by hiding the key in the MIME header, as shown in 25. Clearly, since any plain text message can be turned into a MIME message, the 32K limit is not significant.

Notice that this approach is different than using the MIME format to add the key as an attachment; a mandatory attachment to every outgoing message would create a burden on the recipient and push-back on the user.

4.7 Case Study #7: PC Firewall Configuration

Firewalls have been controversial since their inception. A properly run and secured computer system doesn't require a firewall: it's host-based defenses should be more than adequate to handle attacks. And while a firewall might properly be seen as a part of "defense in depth,"

[14] All mail clients tested, including Outlook Express, Outlook, Eudora on Windows, Eudora on the Macintosh, and Apple Mail will not show a header that begins with the string `x-mailer-key` unless the user explicitly chooses to view the message headers. Most of the mailers, in fact, will not display any message header other than the standard `To:`, `From:`, `Subject:`, and `Date:` headers. The exception is the Macintosh version of Eudora, which instead appears to have a list of headers that should *not* be displayed. Due to an implementation bug in the Macintosh version of Eudora, the program suppresses any header that is prefixed by one of the headers on the do-not-display list. This is why the string `x-mailer-key` was chosen in Figure 24.

```
Received: from e3 (KITCHEN.NITROBA.COM [192.168.1.9])
        by nitroba.com (8.12.9/8.12.9) with SMTP id h86GxepW045403
        for <simsong@ex.com>; Sat, 6 Sep 2003 12:59:40 -0400 (EDT)
        (envelope-from slg@ex.com)
Message-ID: <019301c37498$21c95700$0901a8c0@e3>
From: "Simson Garfinkel" <simsong@ex.com>
To: <simsong@ex.com>
Subject: Second Test
Date: Sat, 6 Sep 2003 12:58:41 -0400
MIME-Version: 1.0
Content-Type: multipart/alternative;
        boundary="----=_NextPart_000_0190_01C37476.98C3FD50"
X-Priority: 3
X-MSMail-Priority: Normal
X-Mailer: Microsoft Outlook Express 6.00.2800.1158
X-MimeOLE: Produced By Microsoft MimeOLE V6.00.2800.1165

This is a multi-part message in MIME format.

------=_NextPart_000_0190_01C37476.98C3FD50
Content-Type: text/plain;
        charset="iso-8859-1"
Content-Transfer-Encoding: quoted-printable
x-mailer-key-01: -----BEGIN PGP PUBLIC KEY BLOCK-----
x-mailer-key-02: mQGiBD51D8URBACJ1Z/v870R1YaBhxzHodxOJHfOEsq2FPFSHuXi
x-mailer-key-03: WRf0sSQqxUJQPLvhilNX5mRcH+HwuZtIsrBE/72NFg+6DvyxXik0
x-mailer-key-04: mMfbOr+4RcsDte0g7/q7Z32RxTfycd+Dn7jdxDFCwZ9wRD5U7AHk
x-mailer-key-05: MIopWsqSE8fL5oBSOF/Vhu0EUHG//wCg1Ph15rwq9GIDy7BXIsBR
x-mailer-key-06: xlNefYMD/iQ56Lytoj/nSFZqhoGZMqNvXlNlN59Ec71IPh+m1lvR
x-mailer-key-07: 5e4qrlrNcn00RACnJJOrXNGwugObhk5y7vpK7kucjjvAUUC0c0O5
x-mailer-key-08: V2HXf1nZm/5hvBkfhkeuFoDCaZEhZs/FsLmZtUD+RN1JOKOwgv5A
x-mailer-key-09: mgtIwc4x3e5ry+Bug7z6QXmjA/44p+cou11SQ+S/uRBRvesw+WG2
x-mailer-key-10: jKoISggGJhOFMRDvfKY969PeOiOU1gZ2g+UQGMxU/QVSyREY2ydl
x-mailer-key-11: uApw7LWcuxMaAQRtoCKF9pHs89n9DR1WCxaCny2jU5tMGpyNx5d5
x-mailer-key-12: Gu9Fs/XQgQgQSQ02fZiGsf+9LkBHh+2wPe578kwyMbQkU2ltc29u
x-mailer-key-13: IEwuIEdhcmZpbmtlbCA8c2ltc29uZOBleC5jb20+iFgEExECABkF
x-mailer-key-14: Aj51D8UECwcDAgMVAgMDFgIBAh4BAheAAAoJEAy2yZ/8Ozz2KHEA
x-mailer-key-15: oM7SY8ClUTJBnPKF/aRlk63DeqDNAJdArryOXMVAeGq47UMKPivJ
x-mailer-key-16: QJDCuQENBD51D8oQBACfcZHR/xA6yZGEOjqnmBIV47AWfBhdQDjE
x-mailer-key-17: FrjwjdOPbKSzasVMEw47mM4jLPoKMw5N25KOmZ2nApnf+g5kkvAG
x-mailer-key-18: ZOiclagBmmXNhDdkhv52hP4618jbwi91IDx8QV1Zv8Qave25kLpm
x-mailer-key-19: fHE2Lmbcc4PVCO7ph/MlZzO9VYGZmQT4XwADBQP+JL6ecjXshJMI
x-mailer-key-20: +w6LYk1/QUjjkqRW63/1OXFtyKe3z0umjMXeceQXMqvMoxWT+2CU
x-mailer-key-21: y8qOYUkmkWaLnqjkwg5GSxd8c9pYJ9exKTSbytZhNX1TQEuqlfx8
x-mailer-key-22: Y2dvl/iTV1MAQdCFvUFiKDQhF19/fI6BouDof/+ry+w89QD7fQW4
x-mailer-key-23: utqIRgQYEQIABgUCPmUPygAKCRAMtsmf/Ds891dwAKCIbSbsOuFh
x-mailer-key-24: jidKo/e9LT4gaADOhwCdHZKmkKVb61NtqULGlDiersiJCyo==xfJ
x-mailer-key-25: z
x-mailer-key-26: -----END PGP PUBLIC KEY BLOCK-----

This is a second test message. It is a MIME message
------=_NextPart_000_0190_01C37476.98C3FD50--
```

Figure 25: A simple test message that has a key added to the first MIME header.

the experience with firewalls in industry is that they tend to encourage lax practices behind the firewall, while simultaneously becoming more porous over time. (This is *Farmer's Law*: "The Security of a Computer System Degrades in Direct Proportion to the Amount of Use the System Receives," as relayed in [**?**])

Unfortunately, many computers are equipped with operating systems that are no longer adequate to handle many attacks to which they are exposed. In part, this is because attacks are both *external* and *internal*. In part, this is because the software vendors have delivered software with substantial vulnerabilities.

Today there are fundamentally two kinds of firewalls available for personal compter users. Both MacOS and Windows XP are delivered with host-based firewalls that can automatically drop incoming IP packets that are not destined for a predetermined TCP or UDP port. These firewalls are primarily useful against attacks that exploit bugs in network servers — sometimes in servers that the user doesn't even know are running. (A good example of this is CERT Vulnerability VU#951555 [Ins01], a remote buffer overflow in the Microsoft Windows Universal Plug and Play server.)

The second kind of firewalls, exemplified by Zone Labs Zone Alarm, will also block individual programs on a personal computer from initiating outbound connections. These kinds of firewalls are primarily useful against Trojan horse programs and so-called *spyware* that may be inadvertently running on a person's computer.

People who use host-based firewalls, especially those that block outbound connections, report considerable frustration in dealing with their constant alerts and warnings. The problem is that the programs do little to help the user decide which connections represent a potential security or privacy violation, and which do not.

5 Research Basis for this Thesis

This section of my thesis proposal discusses the specific research projects that I plan to undertake for the purpose of finding and support the Usable Security design principles.

5.1 The Remembrance of Data Passed Traceback Study

In [GS02] I hypothesized with Shelat that there were several possible reasons for the existence of confidential information on discarded media:

Lack of knowledge The individual (or organization) disposing of the device simply fails to consider the problem (they might, for example, lack training or time).

Lack of concern for the problem The individual considers the problem, but does not think the device actually contains confidential information.

Lack of concern for the data The individual is aware of the problem—that the drive might contain confidential information—but doesn't care if the data is revealed.

Failure to properly estimate the risk The individual is aware of the problem, but doesn't believe that the device's future owner will reveal the information (that is, the individual assumes that the device's new owner will use the drive to store information, and won't rummage around looking for what the previous owner left behind).

Despair The individual is aware of the problem, but doesn't think it can be solved.

Lack of tools The individual is aware of the problem, but doesn't have the tools to properly sanitize the device.

Lack of training or incompetence The individual attempts to sanitize the device, but the attempts are ineffectual.

Tool error The individual uses a tool, but it doesn't behave as advertised. (Early versions of the Linux wipe command, for example, have had numerous bugs which resulted in data not being actually overwritten. Version 0.13, for instance, did not erase half the data in the file due to a bug.)

Hardware failure The computer housing the hard drive might be broken, making it impossible to sanitize the hard drive without removing it and installing it in another computer—a time-consuming process. Alternatively, a computer failure might make it seem that the hard drive has also failed, when in fact it has not.

Among non-expert users, we suggested that lack of training might be the primary factor in poor sanitization practices.

Among expert users we suggested a different explanation: Many sophisticated users are aware that the Windows `format` command does not actually overwrite a disk's contents. Paradoxically, the media's fascination with exotic methods for data recovery might have had the result of decreasing sanitization among these users by making it seem too onerous. In several informal interviews, users told us things like: *The FBI or the NSA can always get the data back if they want, so why bother cleaning the disk in the first place?* We hypothesized that some individuals fail to employ even rudimentary sanitization practices because of these unsubstantiated fears. This reasoning is flawed, of course, because most users should be concerned with protecting their data from more typical attackers, rather than from US law enforcement and intelligence agencies. Even if these organizations do represent a threat to some users, we noted that today's readily available sanitization tools can nevertheless protect their data from other credible threats.

In February 2003 I filed an application with COUHES for permission to contact the individuals who had left personally identifiable information on the hard drives that had previously been purchased. This research was entitled "Remembrance of Data Passed' trace-back study," as the goal of the study is to trace the data on the hard drives back to the data subjects.

This study hopes to discover:

- What measure, if any, were taken to properly sanitize the media before the media was recycled.
- Whether the identifiable individuals had the responsibility (legal, moral, or otherwise) to sanitize the media, or whether that responsibility rested with another individual or organization.
- What the individual would like done with the hard drive now in my possession.

A survey designed to collect information that could be used to answer these questions has been constructed. Permission by COUHES was granted on March 20th.

Subjects for this study will be obtained by performing forensic analysis of the data that is currently in our possession. Primary analysis of this data will reveal names, phone numbers, e-mail addresses and other personal information of the original owners. I have started contacting the data subjects, but to date the results have been inconclusive.

5.1.1 Research Justification for the Traceback Study

The Traceback Study has two purposes: to verify our conjectures as to why the disks were not properly sanitized, and to determine what measures, if any, could have prevented the disclosure of the confidential information that the drives contain.

The majority of disk drives obtained for the "Remembrance of Data Passed" study were formatted in the FAT16 or FAT32 file systems used by the Microsoft DOS, Windows, and IBM OS/2 operating systems. Even those disks that were not mountable still had many file names and directory structures in FAT16/FAT32 scattered throughout the disk surface. Shelat and I hypothesized that the primary reason we were able to obtain significant amounts of personal information due to the fact that the Windows `DEL` command does not actually overwrite files being deleted and the `format` command does not actually overwrite the entire disk. Indeed, even on the disks where some personal information was found in visible files (Level 0 and Level 1 files using the taxonomy presented in Table 1), invariably there would be significantly more information found in the deleted files and unallocated disk blocks (Level 2 and Level 3 files using the taxonomy). Thus, the first goal of the trace back study is to validate the hypothesis that it is a usability failure that was responsible for the majority of the information leakage.

The Remembrance of Data Passed article proposed several free or low-cost modifications to existing operating systems and hardware that could be used to mitigate the disk sanitization problem. For example, it would be possible for each disk drive to be equipped with an encryption device and to automatically encrypt data that is written to the disk and decrypt data that is read back; the data on the disk could be sanitized simply by destroying the encryption key.

By determining the circumstances under which a hard disk was disposed of, the trace-back study will allow me to determine if hardware modifications could have prevented the unintentional release of confidential information.

Finally, the trace-back study hopes to collect anecdotal stories from the data subjects that help argue the mandatory inclusion of sanitization technology into systems sold on the commercial market.

5.1.2 Related Work on Disk Sanitization

Although there have been no other studies examining data found on recovered hard drives, there has been substantial work on techniques for properly sanitizing hard drives.

There are many programs on the market that can be used to sanitize computer disk drives. Some of these programs are designed to remove traces of temporary and deleted files (level 1,

2 and 3 files to use the Garfinkel/Shelat taxonomy), wihle other programs are designed to be used on disks that are being discarded. Some of these programs appears in [GS02], more are listed at [shr].

Gutmann [Gut96] describes the problem of properly sanitizing disk drives and explores the possibility of recovering data from blocks that have been overwritten. Following the publication of his article, many of the commercial tools were modified to use the sanitization patterns that Gutmann's article recommends.

Despite the availability of these tools, before the publication of our 2003 paper, there was only anecdotal evidence that the used computer market was awash in confidential information. I are not aware of any similar "traceback" study being conducted at this time.

5.2 Stream: A System for Providing Usable Encrypted Electronic Mail

The second technical project for my thesis will consist of the continued development of an email encryption system called Stream and the testing of this system on a small population of users. Parts of Stream have already been written and were used to produce the examples shown in 22, 24 and 25.

Stream is an application consisting of a program called **sproxy** that is a TCP/IP proxy for the SMTP and POP protocols; a program called **sfilter** that allows for the decryption of email received by an IMAP server. Additional items that need to be developed for deployment of stream include an installer; documentation; and promotional materials (the stream website). The system does not contain an encryption engine, but instead uses Gnu Privacy Guard (GPG) for cryptographic functions.

Stream is being developed for the purpose of demonstrating several of my Secure Usability principles, and in the hope that the program's development will bring additional principles to light.

5.2.1 The Need for Usable Email Encryption

Several factors have limited the general deployment of encrypted email:

- Existing email encryption schemes are difficult to use, even when built into mail clients.
- For people sending encrypted email, there is usually no guarantee that the recipient will be able to decrypt that mail.

- If a person wishes to send encrypted email and is unable to do so, most people will forgo encryption and accept the risk of sending unencrypted mail, rather than forgo sending email entirely.

Stream will overcome these usability problems through the application of the Usable Security principles outlined earlier in this proposal. Specifically, Stream provides a zero-click interface that simultaneously simplifies user interaction while eliminating the burden on non-users.

Stream automatically creates self-signed PGP keys for its users and distributes these keys in email headers. When a person replies to a mail header that contains a key, the user's mail agent notices the public key and automatically encrypts the outgoing email message. The mail client builds a database of public keys for email addresses; if there are multiple recipients to a message, that message will be encrypted to some recipients and not encrypted to others. Stream thus implements a policy that is similar to the policy that many computer users employ today: they use encryption if they can, otherwise they send messages without encryption.

Stream is based on the observation that many email messages are sent by using the "reply" feature of email browsers. That is, many email users send their messages to individuals with whom they have established correspondence. In a circumstance such as this, I believe that users care more about the continuity of identity rather than absolute identity. That is, users care that the person who is receiving their email today is the same as the person who sent the message to which they are replying; users rarely care if the legal name of the person that they are corresponding with is "John Joseph" or "Jonathan Joseph." (A determination of what users actually believe could be made with a user study, of course.)

I first suggested the underlying idea and security policy of Stream at the "PGP: The First 10 Years," luncheon at the Computers, Freedom and Privacy 2001 conference in Cambridge, Mass. Work commenced on Stream in December 2002.

5.2.2 Research Justification For Developing Stream

The purpose of Stream is to show that when users are given an email encryption system that obeys my principles for usable security, the barriers that have been traditionally associated with email encryption cease to exist. This will be shown by my completing stream, releasing it, gathering feedback from users, and finally conducting a small-scale usability test.

Stream is designed as a proxy to show that these usability principles can be retrofitted onto existing systems. This is an important point: if the only way that Stream could provide usable secure email was to replace the user's email client, it is unlikely that it would receive significant adoption. Indeed, the experience of HushMail, a secure web-based email system, shows that even systems that provide both security and usability may fail in the marketplace if they require other compromises on the part of the user.

5.2.3 Related Work on Opportunistic Encryption

There is a significant body of work on cryptographic proxies that are similar in design, philosophy and spirit to Stream. Sadly, none of the systems that I have found are suitable for using as a test bed for my Secure Usability design principles. If they were, I could use one of the existing systems and avoid the need to develop Stream.

There are literally dozens of programs that are designed to make the PGP and GPG encryption systems easier-to-use. Many of these programs incorporate some aspect of opportunistic encryption or decryption. These programs include the PGP plug-ins for Eudora, Microsoft Outlook, and Outlook Express distributed with PGP; the GPGMail add-on for Apple's OS X Mail application; and PGP Inc.'s recent announcement of "PGP Universal." As discussed in Section 2, Pereira [Per03] has done work applying the principle of opportunistic encryption and decryption to the S/MIME encryption standard.

Of the systems mentioned in the preceding paragraph, only PGP Universal includes automatic key generation similar to those in Stream. PGP Universal goes further, optionally making PGP keys for recipients as well — and approach that is similar in principle to Identity Based Encryption [ibe].

Nevertheless, I believe that even PGP Universal will have significant adoption problems because it violates my principle of No External Burden. As I have shown earlier in this document, this violation arises from the fact that both PGP/MIME and S/MIME use the MIME parts for both message security and for message content delivery, and this dual use of the MIME standard causes problems on mail clients that do not implement the corresponding MIME security standard.

5.3 Literature and Product Review

The final research project for my thesis will consist of an in-depth review of security and usability references, security products (both failed and successful), and operating systems. I will complement this review with interviews that I will conduct with developers, practitioners and analysts. The goal of this review is to attempt to summarize the past 20 years' experience in the design of security software. Some questions that I hope to answer with this review include:

- What are the design principles that have been generally followed in the design of security software? What principles have been tried and abandoned?
- What distinguishes successful smart-card and biometric systems from unsuccessful ones?
- Which password policies have been successful, and why?

- What compromises between security and usability have major software vendors been forced to make? Can they document the reasons for these compromises and/or changes to their systems? Have the compromises and changes been successful?
- Why are PC security products so difficult for their users to configure, and is there a way to make them more understandable?

Parallels for this section will be drawn from other industries as well. For example, the efforts and success of the Consumer Products Safety Commission (CPSC) to improve the safety of consumer products through the use of design specifications and safety labels. One possibility that will be explored is the use of standardized icons and symbols to tell computer about the implications of actions that they are given the choice to perform.

6 Work To Be Done

To finish my dissertation, I propose that I will complete the following work:

1. Development of a small number (between 10 and 20) of guiding design principles for the creation of systems that are both secure and usable.
2. Analyze a range of existing systems with respect to the principles.
3. Complete the "trace-back study" to learn the actual reasons that I was able to obtain the confidential information on the hard drives that were used in the *Remembrance* paper. [GS02].
4. Complete the implementation of the Stream encryption proxy on the Windows, Macintosh, FreeBSD, and Linux operating systems. Such implementation will include basic encryption functionality; a log system that will report in a user-friendly manner what the program has done on the user's behalf, and give the user the ability to reverse actions that are reversible.
5. Conduct a user study with Stream that is modeled on the Whitten/Tygar protocol discussed in "Why Johnny Can't Encrypt." [WT99]
6. Implement a prototype system that disentangles data sanitization, forensic, and backup issues. Deploy this system for several FreeBSD and MacOS-based computers.
7. Develop a workable model using my principles for assuring the privacy and security of consumers purchasing products containing RFID tags.

8. Explore how federal regulation could be used to simultaniously improve security and usability of desktop applications and web services, using the experience of Section 508 as a model.

9. Write a masterful analysis of items #2 through #8 that demonstrates the correctness of item #1.

6.1 Status of the Traceback Study

Although I have been slow to start up, I have obtained the names of several individuals and companies from the images of the hard drives. I have contacted several by email, but so far none of the individuals that I have exchanged email with were responsible for the disposal of the hard drive that is now in my possession.

6.2 Status of Stream

Stream is currently under development. The basic program needs to be finished

and tested on a wide variety of platforms and mail user agents. Platforms targeted for testing include Macintosh OS X, Microsoft Windows, Linux, and FreeBSD. MUAs include Apple Mail, Microsoft Outlook Express, Microsoft Outlook, Microsoft Entourage, Netscape Messenger, and Qualcomm Eudora. It is believed that many operational issues relating to usability and real-world practical issues will emerge from the testing and integration procedure.

Stream has a working interface to GPG and includes a complete implementation of the PGP/MIME message standard. I am debating whether or not to implement S/MIME as well. An S/MIME implementation is required for Stream to be truly useful to the community at large, since this would allow Stream users to freely communicate with either PGP/MIME or S/MIME users, but I do not think that it is necessary in order to complete my evaluation studies.

7 Committee and Timetable

It is proposed that my thesis committee will consist of:

- David D. Clark., Senior Research Scientist, MIT Laboratory for Computer Science.
- Robert C. Miller, Assistant Professor, MIT Department of Electrical Engineering & Computer Science.
- Ronald L. Rivest, Professor, MIT Department of Electrical Engineering & Computer Science.
- Daniel J. Weitzner, Principal Research Scientist, World Wide Web Consortium, MIT

Although some of the work discussed in this proposal has been completed, a significant amount of work remains to be done. A working prototype of the email encryption proxy has been created. It needs to be refined and tested by a broader user base.

October 2003	Completion and acceptance of this document.
December 2003	Apply to COUHES for permission to conduct user evaluation test of encryption proxy.
February 2003	Complete contacts of individuals in “trace-back” study.
February 2003	Proxy user trials begin
April 2004	Complete user trial of proxy.
Summer 2004	Final writing
August 2004	Completion and thesis defense.

Acknowledgments

Jean Camp (Harvard), David Clark, Rob Miller and Dagmar Ringe (Brandeis) provided valuable feedback on previous versions of this document. The case study of Internet Explorer versus Apple Safari was suggested by Marc Rotenberg of the Electronic Privacy Information Center.

References

[AS99] Anne Adams and Martina Angela Sasse. Users are not the enemy. *Communications of the ACM*, 42:41–46, 1999.

[ASODC85] Communications And Intelligence Assistant Security Of Defense Command, Control. Trusted computer system evaluation, 1985. URL `http://www.radium.ncsc.mil/tpep/library/rainbow/5200.28-STD.html`.

[Bal88] Robert W. Baldwin. *Rule Based Analysis of Computer Security*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA 02139, March 1988.

[BF93] N. Borenstein and N. Freed. MIME (multipurpose internet mail extensions) part one: Mechanisms for specifying and describing the format of internet message bodies, 1993.

[Bha00] Jagdish Bhagwati. *The Wind of the Hundred Days: How Washington Mismanaged Globalization*. MIT Press, 2000.

[BL73] D. Bell and L. LaPadula. Secure computer systems: Mathematical foundations and model. report MTR 2547 v2, November 1973.

[bP79] Jonathan b. Postel. Internet message protocol, 1979.

[CA02] Bryan Costales and Eric Allman. *sendmail, 3rd Edition*. O'Reilly & Associates, 2002.

[CFN88] David Chaum, Amos Fiat, and Moni Naor. Untraceable electronic cash. 1988. URL `http://www.wisdom.weizmann.ac.il/~naor/PAPERS/untrace_abs.html`.

[Com02] Apple Computer. Aqua human interface guidelines, 2002. URL `http://developer.apple.com/documentation/UserExperience/Conceptual/Aqu%aHIGuidelines/AquaHIGuidelines.pdf`.

[Con03] International Information Systems Security Certifications Consortium. About CISSP certification, 2003. URL `https://www.isc2.org/cgi/content.cgi?category=19`.

[Cro82] David H. Crocker. Standard for the format of ARPA internet text messages, 1982.

[CSI03] CSI. *2003 CSI/FBI Computer Crime and Security Survey*. Computer Security Institute, 2003.

[CW87] D. D. Clark and D. R. Wilson. A comparison of commercial and military computer security models. IEEE Computer Society Press, April 1987.

[DHR+98] S. Dusse, P. Hoffman, B. Ramsdell, L. Lundblade, and L. Repka. RFC 2311: S/MIME version 2 message specification, 1998. URL `http://www.ietf.org/rfc/rfc2311.txt`.

[DOD95] Cleaning and sanitization matrix, 1995.

[Elk96] M. Elkins. RFC 2015: MIME security with pretty good privacy (PGP), 1996. URL `http://www.ietf.org/rfc/rfc2015.txt`.

[FK96] A. O. Freier and P. Karltrons. The SSL protocol, 1996. URL `http://wp.netscape.com/eng/ssl3/ssl-toc.html`.

[FS90] Dan Farmer and Eugene H. Spafford. The COPS security checker system. In *the Summer Usenix Conference*. Anaheim, CA, USA, 1990.

[FV95] Dan Farmer and Weitse Venema. Improving the security of your site by breaking into it, 1995. URL `ftp://ftp.win.tue.nl/pub/security/admin-guide-to-cracking.101.Z`.

[Gar87] Simson Garfinkel. An introduction to computer security (part 1). *The Practical Lawyer*, 33:39–54, 1987.

[Gar94] Simson Garfinkel. *PGP: Pretty Good Privacy*. O'Reilly & Associates, 1994.

[Gar03] Simson L. Garfinkel. Enabling email confidentiality through the use of opportunistic encryption. 2003. URL `http://www.simson.net/clips/academic/2003.DGO.GarfinkelCrypto.pdf`.

[GS91] Simson Garfinkel and Gene Spafford. *Practical UNIX Security*. O'Reilly & Associates, 1991.

[GS02] Simson Garfinkel and Abhi Shelat. Remembrance of data passed. *IEEE Security and Privacy*, January/February 2002.

[Gut96] Peter Gutmann. Secure deletion of data from magnetic and solid-state memory. In *Sixth USENIX Security Symposium Proceedings*. USENIX, San Jose, California, July 22-25 1996. See http://www.cs.auckland.ac.nz/ pgut001/pubs/secure_del.html for the most recent version of this paper.

[Has02] Judi Hasson. VA toughens security after PC disposal blunders. *Federal Computer Week*, August 26 2002.

[ibe] Identity based encryption.

[Ins01] Carnegie Mellon Software Engineering Institute. Microsoft windows universal plug and play (upnp) vulnerable to buffer overflow via malformed advertisement packets, 2001. URL `http://www.kb.cert.org/vuls/id/951555`.

[Int03] Internet Security Systems. *2002 Annual Report To Shareholders*. Internet Security Systems, 2003. URL `http://media.corporate-ir.net/media_files/NSD/ISSX/reports/ar2002.pdf`.

[Joh00] Jeff Johnson. *GUI Bloopers: Dont's and Do's for Software Developers and Web Designers*. Morgan Kaufmann Publishers, 2000.

[Kei03] Richard Keightley. Encase version 3.0 manual revision 3.18, 2003.

[Lym01] Jay Lyman. Troubled dot-coms may expose confidential client data. *NewsFactor Network*, 8 August 2001. http://www.newsfactor.com/perl/story/12612.html.

[Mar97] John Markoff. Patient files turn up in used computer. *New York Times*, 4 April 1997.

[MS02] Kevin Mitnick and Simon. *The Art of Deception*. John Wiley & Sons, 2002.

[NIS85] Password usage, 1985. URL `http://www.itl.nist.gov/fipspubs/fip112.htm`.

[Nor83] Donald A. Norman. Design rules based on analyses of human error. *Communications of the ACM*, 26(4), April 1983.

[Nor88] Donald A. Norman. The design of everyday things. 1988.

[Odl03] Andrew Odlyzko. Economics, psychology, and sociology of security, 2003. URL `http://www.dtc.umn.edu/\~odlyzko`.

[Per03] Mindy Pereira. *Trusted S/MIME Gateways*. Dartmouth College, May 2003.

[Rek99] Jun Rekimoto. Time-machine computing: A time-centric approach for the information environment. In *ACM Symposium on User Interface Software and Technology*, pages 45–54, 1999. URL `citeseer.nj.nec.com/rekimoto99timemachine.html`.

[Sch02] Bruce Schneier. *Secrets and Lies: Digital Security in a Networked World*. John Wiley & Sons, 2002.

[Sha99] Stephen Shankland. Email virus spreading rapidly. *CNet*, 1999. URL `http://news.com.com/2100-1023-223602.html`.

[shr] URL `http://www.fortunecity.com/skyscraper/true/882/Comparison_Shredders.ht%m`.

[Smi03] Sean Smith. Position paper: Effective pki requires effective hci. 2003. URL `http://132.246.128.219/CHI2003/HCISEC/hcisec-workshop-smith.pdf`.

[Sof03] Grisoft Software. Avg anti-virus, 2003. URL `http://www.grisoft.com/`.

[SPW02] Stuart Staniford, Vern Paxson, and Nicholas Weaver. How to 0wn the internet in your spare time. In *Proceedings of the 11th USENIX Security Symposium (Security '02)*, 2002. URL `citeseer.nj.nec.com/staniford02how.html`.

[SS75] Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63:1278–1308, September 1975.

[SS03] Stuart E. Schechter and Michael D. Smith. How much security is enough to stop a thief? Gosier, Guadeloupe, January 2003.

[SSH93] David R. Safford, Douglas Lee Schales, and David K. Hess. The TAMU security package: An ongoing response to internet intruders in an academic environment. In *Proceedings of the Fourth USENIX Security Symposium*, 1993.

[Sym] Symantec. Norton systemworks 2003. URL `http://www.symantec.com/sabu/sysworks/basic/`.

[U.S73] Secretary's advisory committee on automated personal data systems, records, computers, and the rights of citizens, 1973.

[Vil02] Matt Villano. Hard-drive magic: Making data disappear forever. *New York Times*, 2 May 2002.

[VIS97] SET secure electornic transaction specification, version 1.0, 1997. URL `http://www.visa.com/set`.

[Whi03a] Alma Whitten. Alma whitten, 2003. URL `http://www.gaudior.net/alma/`.

[Whi03b] Alma Whitten. Making security usable, 2003.

[WT99] Alma Whitten and J. D. Tygar. Why Johnny can't encrypt: A usability evaluation of PGP 5.0. In *8th USENIX Security Symposium*, 1999. URL `citeseer.nj.nec.com/whitten99why.html`.

[WT03] Alma Whitten and J. D. Tygar. Safe staging for computer security. CHI, 2003. URL `http://132.246.128.219/CHI2003/HCISEC/hcisec-workshop-whitten.pdf`.

[Yee02] Ka-Ping Yee. User interaction design for secure systems. 2002.

[Yee03] Ka-Ping Yee. Secure interaction design and the principle of least authority, 2003. URL `http://www.sims.berkeley.edu/~ping/sid/yee-sid-chi2003-workshop.pdf`% fi.

[Ylo96] T. Ylonen. SSH - secure login connections over the internet. Proceedings of the 6th Security Symposium) (USENIX Association: Berkeley, CA):37, 1996. URL `http://citeseer.nj.nec.com/ylonen96ssh.html`.