Evaluating Web Browser Security Interfaces

for a More Meaningful Design

A Thesis
in TCC 402

Presented to

The Faculty of the
School of Engineering and Applied Science
University of Virginia

In Partial Fulfillment

of the Requirements for the Degree

Bachelor of Science in Computer Science

by

Jennifer Kahng

March 26, 2001

On my honor as a University student, on this assignment I have neither given nor received unauthorized aid as defined by the Honor Guidelines for Papers in TCC Courses.

_____
Jennifer Kahng

Approved _____ (Technical Advisor)
David Evans

Approved _____ (TCC Advisor)
Rosanne Welker

Table of Contents

Table of Figures

Abstract

As more and more services become available on the Internet, the issue of online security gains importance. The World Wide Web, a common method for interacting with the Internet, provides mechanisms for people to take advantage of many services: accessing bank accounts, purchasing materials, conducting research, etc. Web browsers, software used to access the Web, also provide protection against security vulnerabilities online. However, current web browser security messages lack meaningful content and often display in inappropriate situations, interrupting the user unnecessarily. Thus, users learn to ignore or remove the messages, even though they may be helpful in certain situations. Web browsers utilize security policies to determine when to display security warnings but currently they are too generic. Before developing stronger policies, some mechanism to regain user attention should be in place or the policies may be ineffective. This thesis project evaluated alternate designs for security warnings. The results illustrate that attracting a user's attention in appropriate situations is difficult. Simply modifying the format or layout of a security message is not sufficient to capture the user's attention and sustain it. Combining new warning designs with stricter policies and promotion of user education in security should help users become aware and alert of their computing environment.

Chapter 1 - Introduction

The World Wide Web is one of the most accessible parts of the Internet. Its ease of use allows people to perform tasks quickly and easily. But the creation of online shopping, banking and other personal services leads many users to pass information that should be kept private in an environment to which potentially everyone could have access. Web browsers attempt to circumvent unauthorized interception of personal information by providing warnings when users try to send information unencrypted somewhere. Similarly, browsers also attempt to notify the user when applications download and try to execute on the user's machine. However, warnings often go unread or are configured never to show again because the messages are deficient in content and understandability. This thesis project evaluated the effectiveness of alternatively designed web browser security messages by tracking user reactions to new designs. The results illustrated that attracting user attention is very difficult, even if security warnings are drastically different from standard warnings. The more unusual, though, the higher the rate of always evoking the appropriate response the first time the user saw the message. This behavior introduces the possibility of gaining user attention before high-risk actions can occur, thereby increasing user awareness of risks.

1.1. Problem Context

Security and privacy messages appear due to a policy, or set of rules, created by web browser developers. But the policies encompass a wide range of general situations. Any time a user enters text into a form field (e.g. Web searching, online store product search, etc.) and submits the form, a policy is flagged and a warning appears notifying the user that he or she may be sending personal or private data unencrypted over the Internet. If a user is actually sending

personal data (e.g. credit card number, phone number, address, Social Security number, etc.) and the method is actually secure, a message similar to a security warning appears alerting the user that they are sending data over a secure channel and that the information cannot be intercepted. The messages look the same yet display very different messages. Also, when a user accesses a secure web site, a security message appears to declare the site secure. Then, when the user leaves a secure web site, another security message appears to state the user is leaving a secure page and any information transmitted is unencrypted. Without carefully reading the messages, users cannot differentiate between messages. As a result of exposing users to many similar looking messages, users become annoyed and frustrated at being interrupted during their web activities.

Almost all pop-up messages have a similar appearance: a small box of some default color with buttons and text. The text in the pop-up box describes a general scenario for the policy that generated the warning. Because the messages have a common format, users tend to mistake one for another. In time, users learn to ignore all pop-up boxes. Most people click on the "Continue" or "Cancel" buttons every time any message appears without bothering to read the associated message. Users also uncheck the "show next time" type of button, thereby disabling the pop-up warning. The actual function of the checkbox is ambiguous as well. If the user unchecks the box, he or she has no idea what security policy is disabled. In the worst case, all security pop-up messages are disabled and the user will never be warned when a potentially non-secure transaction occurs. An experiment, described in Chapter 3, revealed over 70% of the test group always clicked "Continue" to a message that should have evoked a "Cancel" response. That type of automatic behavior is undesirable when valid security or privacy situations arise. Figure 1 shows a standard security message as seen in Netscape.

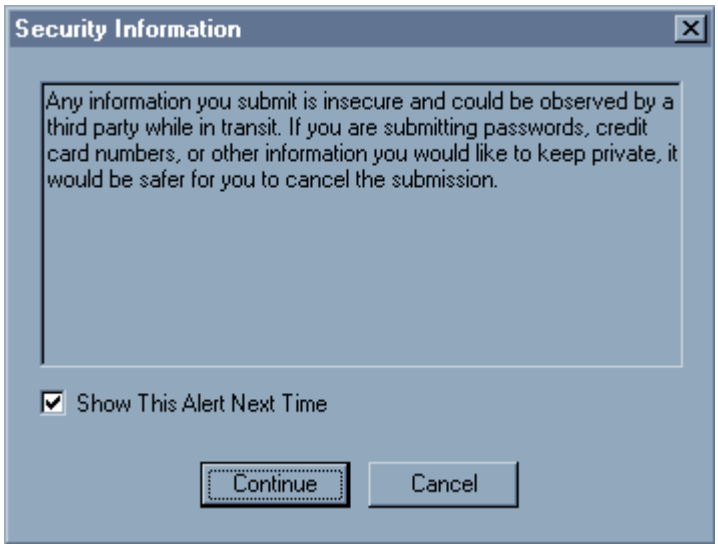

Figure 1 - A standard security message from Netscape Navigator.

The functionality of security within the web browser is yet another problem. Security settings assume the user understands what security features his or her web browser has. Also, the settings come preset to some default that is often inadequate for full protection while browsing the Web. Users are not encouraged to sift through the security settings and other preferences to optimally protect themselves. In Netscape's Navigator, users not only need to alter settings in "Preferences," but they must also go through the "Security Info" dialog to understand what Netscape can do. Similarly, Microsoft's Internet Explorer has a very detailed security interface that requires the user to understand "signed" controls, enabling cookies, and scripting, as well as choosing from "High" and "Low" pre-configured security settings. People who browse the Web often do not understand the underlying concepts of web security and do not care to learn. User ignorance and the dismissal of potentially helpful messages make for a dangerous combination when performing transactions online. Users could potentially be tricked into giving out their private information to unauthorized parties, unknowingly perform some action in a non-secure fashion, or even harm themselves by being attacked by some hidden program like a virus.

1.2. Project Description and Impact

As web browsers evolve, it may be possible to develop security and privacy policies that are more specific and accurate. But if users continue to ignore the messages, generating new and improved warnings will not be effective. I developed my project to address the problem of current user behavior. Through experimentation, I determined that new designs were effective when combined with meaningful descriptions of the security problem and an extremely unusual appearance. All of the unusual security messages I created had the same warning message and only differed in physical appearance. In looking at the pop-up, the user immediately understands what the web page is attempting to do and should also understand that clicking "Continue" is not appropriate. Ideally, having warnings that users pay attention to will make them aware of their environment online and compel them to act safely and responsibly. Figure 2 is an example of a security message I designed with a meaningful warning and altered appearance.

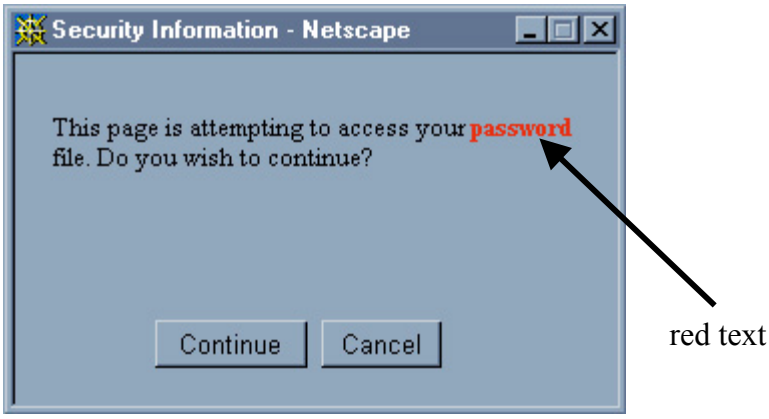

Figure 2 - An example of an unusual security warning.

Although the results from my experiments showed that users paid attention to security messages more when the messages deviated from standard format, altered messages are still not enough to capture their attention during appropriate situations. The message in Figure 2 is somewhat excessive. If the web browser could actually detect some web page component

attempting to steal passwords then that is obviously an action that should not be allowed and the web browser should take care of that situation automatically. However, I presented this same message to sets of people in different ways to test their reactions and there were still users who always clicked "Continue." It may be that those users are too accustomed to clicking "Continue" automatically, they realized the message was probably not real and selected that option for amusement, or any other number of reasons. More research and experimentation should be done to determine if it is feasible to create methods of notifying users appropriately or if a new approach should be considered.

1.3. Report Overview

Chapter 2 of this technical report describes the problems in current web browser security design as well as previous research in the field of user interfaces in security applications. Chapter 3 begins an explanation of the experimentation done during this project and presents preliminary results that guided the completion of the project. Chapter 4 provides details on an experiment done for this thesis project as well as a discussion of the results and their meaning. Chapter 5 presents the significance of the data collected during experimentation and recommends future work in the area of security interfaces.

Chapter 2 - Background Information

Web browsers are now an essential part of our daily lives. Many people use them to access e-mail, perform research, buy products and do other errands. Because web browsers are used for so many tasks, there are built in functions to perform those tasks as well as to protect users from malicious content on the World Wide Web. Often, the methods of protection are not completely adequate for user needs or are not fully utilized, putting users at even greater risk. This chapter explains the web browser security messages and their weaknesses, which my project addresses.

2.1. The Problems

The World Wide Web contains millions of web pages with a variety of different types of content. Consequently, there are potentially millions of different ways to compromise a user's personal information. Web browsers attempt to counteract malicious actions by providing methods to warn users when a security or privacy breach occurs via pop-up message boxes. However, these messages are not always helpful and are often ambiguous and confusing.

Additionally, the messages are based on a general set of security and privacy policies defined by web browser developers. For instance, since consumers can submit their credit card number via forms on a web page, a policy exists which flags any web page with content involving forms as potentially hazardous and displays a message. Not all web pages with forms involve credit card numbers, social security numbers or any other personal and private information, so this method of notification frustrates users over time. Users also quickly learn to ignore and even disable the warning messages that appear because of the general security

policies. While many years of user interface research exists difficulties arise in applying known interface design rules to security applications.

The basis for any user interface design allows the user to perform some task as quickly and efficiently as possible, but pop-up messages go against those design principles. Users do not like interruptions to their tasks since it slows them down or distracts them. Thus, pop-up messages and other disruptions should be kept at a minimum [4:178]. Obviously, an interruption should occur if some problem arises. Web browser security and privacy messages are all potentially important, so they must occur every time a security or privacy issue arises. Determining when such an issue develops also impacts when a message appears.

2.2. Web Component Difficulties

Web browser developers rely on security policies to determine if content on a web page is dangerous, such as attempting to access the user's information without permission. This detection is hindered by the numerous methods of creating web pages and the inclusion of web page components. For instance, methods such as cookies, Java applets, JavaScript, and ActiveX Controls were initially created to allow for versatility in web page functionality. However, these components can be used to threaten user's security and privacy.

Cookies are a method used to record a user's activity on a particular web site. They can also be used to automate user activities such as entering usernames and passwords on a site by storing the information on the user's machine. Cookies are often used by online advertising agencies to track user navigation. Because cookies are transmitted through web browsers by default, they violate user privacy by gathering personal and possibly private data without user consent.

A Java applet is a small, executable program written in the Java programming language that Sun Microsystems, Inc created. When accessing a web page containing a Java applet and the user has the Java Virtual Machine installed, the applet downloads to the user's machine and executes. A web author may program his or her own Java applet to do virtually anything but, Java's internal security model attempts to restrict applet access. If Java's security methods are circumvented the applet could potentially access all parts of the computer. Programmers have exploited this aspect of Java applets in the past. In 1996, researchers discovered that it was possible to program a Java applet to delete files on a user's hard drive without the user's knowledge [5:61]. Since then, Java's security has been modified to restrict applet operations such as reading and writing files on the user's machine and making network connections without the user's consent.

JavaScript is a language that can be interpreted by web browsers on the user's side, called the client-side, without downloading a full program to the user's computer. Also, JavaScript can be used to respond to user input such as mouse-clicks or input into a form field. Like Java applets, JavaScript has been exploited to try and access unsuspecting user's information. One known exploit, discovered accidentally, involved embedding JavaScript within a return e-mail address link through the web-based e-mail client Hotmail. The JavaScript could possibly redirect users to a web page that looks similar to the Hotmail login screen and request they login again due to some false network error. Thus, their passwords are stolen [1]. Victimized sites usually fix JavaScript exploits when they happen, rather than requesting a global JavaScript update.

ActiveX controls, created by the Microsoft Corporation, are Windows programs that can be embedded in web pages. Although similar to Java applets, ActiveX controls must be signed,

meaning they have to contain a certificate declaring that the control originated from a trusted source and agreed to abide by safe ActiveX rules set forth by Microsoft. However, malicious users have found a way around the certificate block. Some ActiveX controls have the ability to execute very low-level commands on any Windows computer with an Intel processor. One major example of misusing that feature, discovered intentionally, was an ActiveX control that randomly rebooted computers [5:64].

Attempting to guard against all of these types of vulnerabilities is virtually impossible. Existing policies search for web components within a page and alert the user to their existence as well as their potential hazards. However, this behavior is too generic. Not all forms require personal or private data. Not all Java applets or ActiveX components are malicious. In trying to protect users all the time, security policies instead annoy users by being overly intrusive. Stronger, more stringent security policies may help but are of no use if no one pays attention.

2.3. Making Users Pay Attention

There is virtually no direct research on web browser warning message design. However, groups are beginning to research security application design. The Association of Computer Machinery's Special Interest Group for Human-Computer Interactions (ACM/SIGCHI) is dedicated to researching interactions between people and computers. One major goal is creating user interfaces that maximize the ease of human-computer interactions. User interface design is not the same between security products and most of the rest of the market but more research is emerging in this area.

User interfaces for security applications are difficult to develop for many reasons. First and foremost, the developer must understand the underlying security principles behind the application. The developer also cannot afford to misunderstand how the application should be

used or the user cannot fully utilize the protection the software offers [6:3]. Alma Whitten, a Ph.D. student at Carnegie Mellon University, has created an online bibliography of researchers and some associated papers dealing with human factors in security. Similar to SIGCHI, her research focuses on security applications and making them more usable. Her evaluation of a well-known security product demonstrated that even if the user interface itself was well designed, users still did not understand how to use the product unless they spent a large amount of time learning about the software itself [7].

There are practical methods for determining what users will respond to as well as what they want out of their software. These methods are user observation and feedback analysis, but also require users to understand how the security software works and how it reacts in certain situations [2:4]. The general population, however, does not understand the details of the security features of their web browser, nor do they learn if it does not affect them directly. That is, users have little interest in security until they become victims of security vulnerabilities. Thus, there still remains the problem of alerting users to security problems in a meaningful manner.

The biggest obstacle to designing new warning messages is creating a design that will catch a user's eye and keep it. Ideally, modifying web browser security policies would solve this problem greatly, since messages would only appear when there was a high likelihood of a serious problem. Also, messages could be written more clearly to describe in greater detail what happened. However, since users are more accustomed to ignoring security and privacy warning messages all together, some method of attracting the user's attention at critical times must exist first in order to reverse the effects of initial web browser design flaws. Once a suitable display method is established, then serious work in newer, stronger security and privacy policies can move forward. Finding suitable display methods was the main purpose behind my thesis project.

Chapter 3 - Measuring User Behavior

Two experiments conducted throughout the academic school year helped determine if alternate web browser security message design could acquire a user's attention in important situations. This chapter explains the first experiment, the results of which provided a basis for the other experiment by analyzing current user behavior. The experiment noted a user's reaction to a standard web browser security message. The majority of the test group exhibited behavior indicating that the user did not read the message, did not care or did not feel the warning was hazardous enough to disallow.

3.1. User Responses to Standard Messages

In order to test current user behavior, I designed a suitable experiment that recorded the actions of users when confronted with a standard style security warning. With the help of my technical advisor, David Evans of the Computer Science Department, and Professor Jane Prey and Peter Miller, also from the Computer Science Department, I gathered a test group consisting of students from an introductory computer course (CS 110). Prey and Miller taught CS 110 the semester I ran the experiment. The experiment masqueraded as part of a homework assignment.

The students received a "Web Treasure Hunt" assignment that required using information retrieved via search engines like Google (http://www.google.com) or AltaVista (http://www.altavista.com). The hunt list consisted of about 30 items; each had the student use the Web to find answers fulfilling the requirements of the item. For instance, a sample item was "Find the homepage of your U.S. Congressman." Another of the included questions was "What city did 'danakate' stay in during the summer of 2000?" which pertained to my own personal website (http://www.danakate.com). Because the assignment was not directly related to security messages, the students had no particular reason to expect a security message to appear.

My main web page contained a link for "danakate's summer 2000." The page also had JavaScript for a false, standard-type security message. When a student navigated to the main page, the pop-up message appeared. If students clicked on "Continue," they were redirected to a page named "pictures.html." Otherwise, they were redirected to "piictures.html." Note that the names of the pages are slightly different in order to distinguish which button students clicked during results analysis. Figure 3 depicts the security message used on the web page. Appendix A contains a source code listing of the HTML and JavaScript used in this experiment.

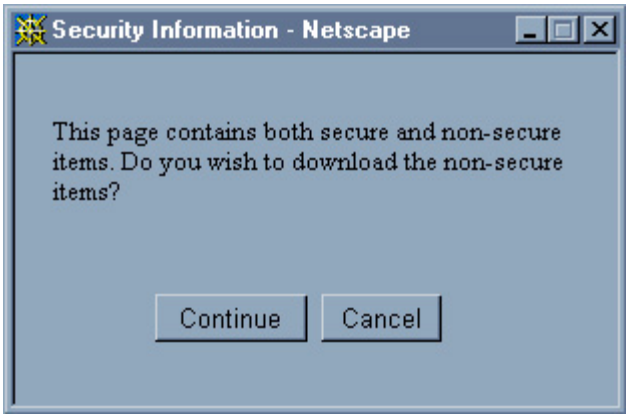

Figure 3 - Standard security warning message used in experiment one.

3.2. Results and Conclusions

The student's choice was recorded in the web server access logs. The web server, Apache version 1.3.9, runs on the Unix operating system FreeBSD 4.1-RELEASE. The system administrator, Yossarian Holmberg, had configured the logging system to write any web server activity for the site into my root directory as a text file so I could analyze the data at any time. By analyzing the log files, using a search and extraction script I created using the Perl programming language, I determined that users usually ignore security warnings.

Before explaining the results, I must make a note about the analysis process. First, students were able to complete the web hunt on their own time, so it was not possible to know

from where they were accessing the web page. In looking at the log files, I noticed web addresses from within the University network that accessed my web page at one time and again at a much later time. Investigation revealed that those machines were from a public lab and were probably different students. Additionally, some students do not live on University grounds and have Internet access not connected with the University. Determining which accesses were from students doing the hunt was difficult, but I decided to include addresses from known local service providers.

A total of 81 students encountered the false security dialog box. Of those students, 75 clicked on "Continue" and 6 clicked on "Cancel" the first time they saw the message. Some students accessed the page multiple times and did not exhibit the same behavior each time. The "Other Behavior" represents students who clicked "Continue" first and then "Continue" or "Cancel" in a seemingly random order. Table 1 shows the types of behavior and number of students that exhibited that behavior. Appendix A contains the script I used to analyze the entries.

Total Number	Always clicked "Continue"	Always clicked "Cancel"	Clicked "Continue" first, then "Cancel"	Clicked "Cancel" first, then "Continue"	Other Behavior
81	62 (76.5%)	5 (6.2%)	6 (7.4%)	1 (1.2%)	7 (8.6%)

Table 1 - Types of Behavior and Number of Students Showing each Behavior

The results suggest a number of possible characteristics about user behavior. From the high number of students who clicked on "Continue" every time they saw the security message, I concluded that they did not read the message and clicked on "Continue" to dismiss the message. The numbers for students who clicked on "Cancel" originally and then clicked on "Continue" or exhibited "Other Behavior" are more disturbing. I doubt people would willingly switch between not allowing and allowing a non-secure action to occur. Since there were students who were inconsistent in clicking, it further suggests that users do not read the pop-up messages and do not

pay attention to what action they take. Although only a small number of students clicked on "Cancel" after they originally clicked on "Continue," those results show people may notice the messages after repeated exposure and realize they should click "Cancel."

However, the goal is to attract users' attention the first time encounter a potential security hazard. This first experiment not only confirmed my original speculation that users do not read security messages, but also provided a method for further testing. The next step was to design security messages differently and determine if they had any effect on the user's web browsing behavior.

Chapter 4 - Evaluating New Designs

The first experiment showed that users do not read current security messages while performing online transactions. A second experiment attempted to determine how users reacted when given non-standard security warning messages. I tested a different audience because using another class would have required too much time to coordinate. I used a page from the Oracle of Bacon web page on the Computer Science Department server. The advantage of using the Oracle of Bacon page was that it exposed my alternate messages to a large number of people from all over the world. However, I was unable to test repeatability. This chapter explains the setup of the experiment and the results gathered. As the messages deviated from standard messages further and further, the number of people who clicked "Continue" decreased. However, some users selected "Continue" for all messages.

4.1. More Use of Unusual Messages

In order to conduct an experiment with a wide audience, I needed a web site that received traffic from around the globe. The Oracle of Bacon web pages are the most popular pages on the Computer Science Department server, receiving over 10,000 hits per day from all corners of the world. Evans, who is also the director of the CS web team, suggested I use a web page linked from the Oracle of Bacon, one that had only a couple hundred hits so as not to flood myself with thousands of log entries to analyze and to minimize the potential mail to the webmaster complaining about security vulnerabilities.

We decided to place one unusual security message on the "Star Links" page of the Oracle of Bacon for a 24-hour period. After that time, the message was changed to another. This cycle was repeated for a total of four days. There was a slight glitch, however, as the University's

network connection broke for about eight hours during the third day, so extra time was added to that day's security message evaluation period. At the end of the experiment, the access logs were analyzed using a script I created using Perl.

4.2. First Three Messages and Results

In developing the security messages, I decided to start with a design that was fairly similar to standard messages and then deviate from the standard as the experiment progressed. I decided not to change the text of the message so that I would not have to determine if the user reacted to the format or the text of the security warning. The text was the same as was used in previous experiments. The first three messages were similar in functionality and are described here. All of the JavaScript and HTML needed to create these messages as well as all files used in analyzing the results are available in Appendix C.

For the first day, I used a false security message with an animated component. The image of the lock, key, and exclamation point blinked every half-second in the hopes of attracting the user's attention. Figure 4 shows the format of the security message and indicates the animated component.

Figure 4 - Security message for day one of experiment three.

The second day of experimentation featured a message with all red text. Standard security messages use black text by default. I felt red text would be more noticeable and would force the user into reading the message completely. A screenshot of the message is shown in Figure 5.

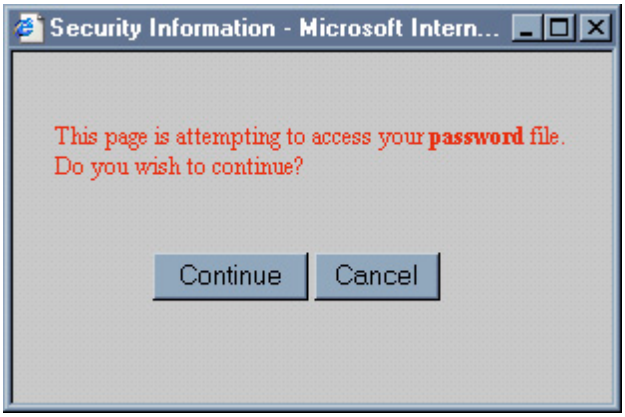

Figure 5 - Security message for day two of experiment three.

The third day utilized red in a new manner. Instead of changing the text, the entire background of the pop-up window was red. This particular message is especially noticeable because it definitely has a different look than any standard security message in either Netscape or Internet Explorer. The negative consequence of using an all red background is that the message may surprise the user or annoy the user to such an extent that he or she will dismiss the message quickly and not read it. Figure 6 portrays the third pop-up message used in the experiment.

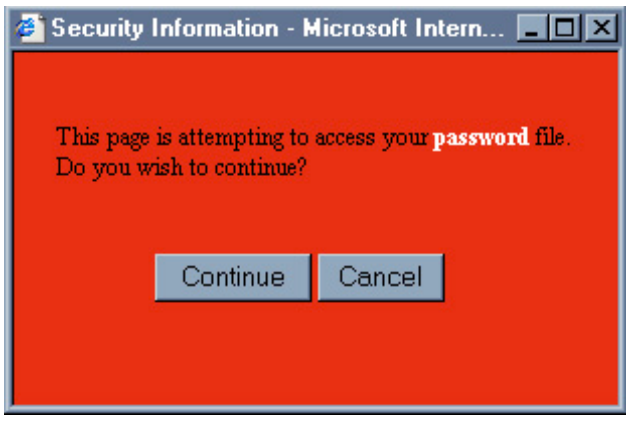

Figure 6 - Security message for day three of experiment three.

Once again, access logs recorded the user's choice. After sorting out all extraneous log entries and creating a more elaborate Perl script to perform analysis, I separated users into categories and drew conclusions from their behavior. Table 2 presents user statistics for the first three days of experimentation.

	Message 1 (animated)	Message 2 (red text)	Message 3 (red box)
Number of Users	226	230	139
Always Clicked "Continue"	23.0%	27.8%	23%
Always Clicked "Cancel"	69.0%	63.0%	69.1%
Clicked "Continue" then "Cancel"	5.3%	6.1%	3.6%
Clicked "Cancel" then "Continue"	0.9%	2.2%	1.4%
Other Behavior	1.8%	0.9%	2.9%

Table 2 - Statistics for first three days of experiment three.

Although the number of people who visited the Star Links page the first three days varied, the percentages in each category were very similar. Because the majority of the visitors selected "Cancel" as their default choice the results suggest that users usually read security messages or habitually choose "Cancel" to any pop-up message. The visitors who chose "Continue" exhibited the opposite behavior. Those users may not read pop-up messages or consistently choose "Continue" to any message that appears. My hypothesis that the drastically different pop-up message, presented on day three, would attract more user attention was incorrect.

4.3. Final Message and Results

The final message was more complicated than the others. Although it featured the same security warning, I introduced a new component: a text box. There were four ways to dismiss the message, three of which I could track. The first was clicking the "Cancel" button. Standard security messages do not allow users to view the page they are attempting to browse if "Cancel" is selected. I did not implement that particular feature in any of my security messages. The

second and third methods are to type “yes” or “no” into the text box and click “Done.” If the user typed any other text into the box and clicked “Done,” the message would not close. The fourth method of dismissing a message is by clicking the “x” button in the top right corner of the pop-up window. That method is standard for closing almost any message and one I cannot track. Figure 7 presents the last security message used.

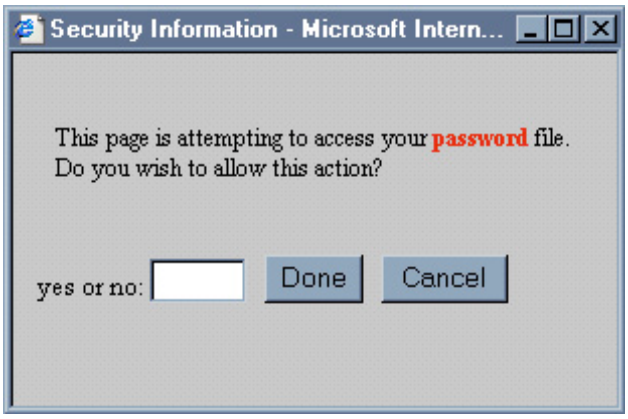

Figure 7 - Security message for day four of experiment three.

The final day’s statistics revealed more about user behavior. Table 3 shows the statistics of user behavior to the last message.

	Message 4
Number of Users	148
Always Typed “Yes”	2.0%
Always Typed “No”	25.7%
Always Chose “Cancel”	67.6%
Chose “Yes” then “No” or “Cancel”	0.7%
Chose “No” or “Cancel” then “Yes”	1.3%
Other Behavior	2.7%

Table 3 - Statistics for the last day of experiment three.

Note: "No" and "Cancel" were considered the same action.

A higher percentage of users selected “No” or “Cancel” but that behavior does not necessarily mean they read the security warning. The user attempting to dismiss the message as fast as possible is a more likely scenario but further analysis revealed 25.5% of the visitors actually typed the word “no.” Since the security warning was more than text and buttons, the

format probably attracted the user's attention and forced them to read the message. Sadly, there were visitors who selected "yes." However, because so few users exhibited that behavior, it is likely the choice was a fluke. I cannot substantiate that supposition, however, as I have no way of contacting the parties who chose "yes."

While my results indicate attracting user attention is possible, compelling users to take an active part in their online activities is not reflected. The security message used is fairly serious, albeit unrealistic. However, out of over 700 participants, only two visitors were concerned enough to send e-mail to the administrator of the supposed vulnerability. It may be that all other visitors deemed the messages too unrealistic to warrant any action on their part. Regardless, I am still surprised at the lack of response from the visitors. Thus, user awareness involves factors other than message content and format.

Chapter 5 - Conclusions

Attracting a user's attention is difficult. Presenting security information in different formats helps but that method is far from satisfactory. This chapter describes the findings gathered throughout this thesis project, the meaning behind the results, and recommendations for future studies. Further research in security application design is necessary to reduce vulnerabilities due to user error or application design and to increase user awareness during online transactions.

5.1. Summary

The results of one experiment illustrated typical user behavior. When confronted with a standard type of security message, and given the choice to allow or not allow non-secure content to display, most users selected "Continue," allowing the action to occur. Interestingly, even users who are security minded click "Continue" without reading security messages. Scott Ruffner, a system administrator for the Computer Science Department, helped me during the course of my project. At one point, my advisor and I had him navigate through some test pages to make sure they worked correctly. Although we told him a false message would appear and that he should select "Cancel," he chose "Continue" several times without reading the message.

Further experimentation revealed that modifying the warning format to reflect the severity of the security problem resulted in an increase in users not allowing non-secure events to occur. However, my original hypothesis that using security messages that deviated heavily from the standard would produce more noticeable results than from messages that were only slightly different was wrong. With the first three messages, the format of each deviated more from the standard than the previous message but roughly one third of the participants selected "Continue"

first to all of the warnings. The last message provided an improvement in the number of users that did not allow a non-secure action to occur but this was because I made it more difficult for the action to proceed.

5.2. Interpretation

The analysis of the experimental data revealed that changing the format of security messages is not enough to alert users to security problems. My results suggest the possibility of gaining user interest in important situations. However, in order to attract user attention, web browser security messages must be in a format that may approach a high level of obnoxiousness. In user interface design, interruptions to the user are kept at a minimum. Any interruptions at all must be brief and non-intrusive. Security warnings, on the other hand, must be intrusive due to the seriousness of the matter.

The second experiment's results suggest users pay more attention to unusual looking messages but that behavior may not be due to actually reading the messages. More plausible explanations include habitual behavior and clicking any button to try and dismiss the message quickly. Another explanation refers to users who chose "Continue" or typed "yes." Since security messages follow a similar format, users who saw my false messages may have realized they were false and chose to allow the action just to see what would happen, if anything. This type of behavior could be extremely dangerous if methods like mine are actually implemented in current web browsers. Initially, there could be a large number of users who willingly submit themselves to high-risk actions because they believe the security messages are false. Although my experiments were useful in capturing user behavior, the results demonstrated nothing about the reasons behind that behavior.

5.3. Recommendations

This evaluation project indicates that user behavior can be changed for the better but not without much research and effort. Other components, such as stronger web browser security policies and more education about security matters are necessary and should be integrated with new message designs to generate a more effective response from the user. Interviewing users after exposing them to unusual security messages may also give more insight into user behavior and, as a consequence, give developers more ideas for a better design. The difficulty arises in constructing a suitable experiment that does not require that users know their behavior to security matters is being observed.

The World Wide Web is one of the most accessible components of the Internet. Research and testing in the field of online security must continue to ensure the protection of user information during online transactions. Increasing knowledge of security matters is also important to not only save users from victimization but also compel them to be more active in the online environment. Educated users aid researchers and developers in making the World Wide Web and the Internet safer by being aware, understanding the importance of security issues, and notifying the proper authorities when a security issues arises.

References

1. Festa, Paul. *Hotmail Flaw Exposes Passwords.* 19 Mar. 2001. <http://news.cnet.com/news/0-1004-200-332525.html>

2. Holmström, Ursula. *User-centered Design of Security Software*. Human Factors in Telecommunications. Copenhage, Denmark. May 1999. 18 Oct. 2000. <http://www.tcm.hut.fi/Research/TeSSA/Papers/Holmstrom/hft99.ps>

3. Oppliger, Rolf. *Security Technologies for the World Wide Web*. Boston: Artech House. 2000.

4. Raskin, Jef. *The Humane Interface: New Directions for Designing Interactive Systems*. Reading: Addison Wesley Longman, Inc. 2000.

5. Tiwana, Amrit. *Web Security*. Boston: Digital Press. 1999.

6. Whitten, Alma, J.D. Tygar. *Usability of Security: A Case Study*. Technical Report CMU-CS-98-115. Carnegie Mellon University, School of Computer Science. December 1998. 18 Oct. 2000. <http://reports-archive.adm.cs.cmu.edu/anon/1998/CMU-CS-98-155.pdf>

7. Whitten, Alma, and J. D. Tygar. "Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0." Proceedings of the 8th USENIX Security Symposium, August 1999. 12 Sept. 2000. < http://www.cs.cmu.edu/~alma/johnny.pdf>

Bibliography

Association of Computer Machinery. *ACM/SIGCHI*. 22 Mar. 2001. <http://www.acm.org/sigchi/>

Electronic Privacy Information Center. *The EPIC Cookies Page*. 19 Mar. 2001. <http://www.epic.org/privacy/internet/cookies/>

Festa, Paul. *Hotmail Flaw Exposes Passwords*. 19 Mar. 2001. <http://news.cnet.com/news/0-1004-200-332525.html>

Holmström, Ursula. User-centered Design of Security Software. Human Factors in Telecommunications. Copenhage, Denmark. May 1999. 18 Oct. 2000. <http://www.tcm.hut.fi/Research/TeSSA/Papers/Holmstrom/hft99.ps>

Netscape Communications Corporation. *Introduction to JavaScript.* 19 Mar. 2001. <http://home.netscape.com/eng/mozilla/3.0/handbook/javascript/getstart.htm>

Oppliger, Rolf. Security Technologies for the World Wide Web. Boston: Artech House. 2000.

Raskin, Jef. The Humane Interface: New Directions for Designing Interactive Systems. Reading: Addison Wesley Longman, Inc. 2000.

SecurityPortal. 19 Mar. 2001. <http://www.securityportal.com>

Sun Microsystems, Inc. *Applets*. 29 Mar. 2001. <http://www.javasoft.com/applets/index.html?frontpage-spotlight>

Tiwana, Amrit. Web Security. Boston: Digital Press. 1999.

Web Design Group. *CGI Programming FAQ: Basic Questions*. 19 Mar. 2001. <http://www.htmlhelp.com/faq/cgifaq.1.html>

Whitten, Alma, J.D. Tygar. Usability of Security: A Case Study. Technical Report CMU-CS-98-115. Carnegie Mellon University, School of Computer Science. December 1998. 18 Oct. 2000. <http://reports-archive.adm.cs.cmu.edu/anon/1998/CMU-CS-98-155.pdf>

Whitten, Alma, and J. D. Tygar. "Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0." Proceedings of the 8th USENIX Security Symposium, August 1999. 12 Sept. 2000. < http://www.cs.cmu.edu/~alma/johnny.pdf>

Appendix A

The material in this appendix refers to the experiment described in Chapter 3. The JavaScript used to generate the pop-up message and the HTML of the false message is listed here. Additionally, the Perl scripts used during log file analysis are given. All of these files and the log files themselves may be accessed online at the following address: http://www.danakate.com/thesis.

A.1. JavaScript for Pop-up Message

The JavaScript used in the first experiment was designed to display a false security warning positioned in the center of the viewer's monitor screen. This is a listing of the JavaScript code used to generate the message.

```
<script language="javaScript">
<!--

// Store the height and width of the user's monitor.
var maxwidth = screen.width;
var maxheight = screen.height;

// Define the height and width of the pop-up message.
var p0pupwidth = 300.0;
var p0pupheight = 200.0;

// Calculate the center of the monitor for the pop-up message position.
var newleft = (maxwidth / 2.0) - (p0pupwidth / 2.0);
var newtop = (maxheight / 2.0) - (p0pupheight / 2.0);

window.screenX = (screen.width / 2.0) - (window.outerWidth / 2.0);
window.screenY = (screen.height / 2.0) - (window.outerHeight / 2.0);

// Display the pop-up with "alert.html" as the source.
remote =
window.open('alert.html','alert','toolbar=no,location=no,directories=no,statu
s=no,menubar=no,scrollbars=no,resizable=no,width='+p0pupwidth+',height='+p0pu
pheight+',top='+newtop+',left='+newleft+'')

//-->
</script>
```

A.2. HTML for Standard Pop-up Message

The HTML for the false pop-up message had a similar look to a standard security message. The only difference between my message and a standard message was that mine redirected users to a different web page based upon their choice of "Continue" or "Cancel." I also did not give the user the option of disabling the security warning. The HTML for the message is given below.

```
<html>
<head>
<title>Security Information</title>
</head>
<body bgcolor="#8098b0">
<center>
<br>
<table width="95%" cellpadding="2" cellspacing="2" border="0">
   <tr>
      <td>
      <font size="-1">This page contains both secure and non-secure items.  Do
           you wish to download the non-secure items?</font>
      <br>
      <br>
      <br></td>
   </tr>
</table>
</center>
<br>
<br>
<center>
<table width="100%" cellpadding="1" cellspacing="1" border="0">
   <tr>
      <td width="50%" align="right">
      <form>
      <input type="button" value="  Continue  "
      onClick="opener.document.location='pictures.html';self.close()">
      </form></td>
      <td width="50%">
      <form>
      <input type="button" value="   Cancel   "
      onClick="opener.document.location='piictures.html';self.close()">
      </form></td>
   </tr>
</table>
</center>
</body>
</html>
```

A.3. Perl Scripts for Analysis

After the first experiment finished, I had to collect the log files and separate the extraneous information from the relevant information. I created two Perl scripts to facilitate my information gathering.

A.3.1. Separating out All Users

The first script traversed all log files generated during the experiment and created a master file with a list of all users who accessed the web page with the false message. The Perl for the script is listed below.

```
#!/usr/bin/perl
$filename = "full-output";

# Open the output file, exit with an error if a problem exists.
open(INDEX, ">$filename") || die("$filename: $!");

# Select the output file for writing.
select(INDEX);

# Create an array of all text files in the directory.
@filenames = glob("*.txt");

# Go through each file in the array.
foreach $file (@filenames) {
      if (!open(READFILE, "<$file")) {
            die("$file: $!");
      }

      # Only print lines that are relevant to the experiment.
      while ($line = <READFILE>) {
            ($datetime, $gmt, $somenum, $url, $host, $requesttype, $request, ,
                  $protocol, $ip) = split(" ", $line);
            if ($url =~ /pictures\.html/) {
                  print $line;
            }
            elsif ($url =~ /piictures\.html/) {
                  print $line;
            }
      }
}

# Reselect the standard output and close files.
select(STDOUT);
close(INDEX);
close(READFILE);
```

A.3.2. Displaying Relevant Information

The second script displayed the sorted log file in a meaningful format. That is, only the address of the user, the page the user accessed and the time of the access were really needed for analysis. Below is a source listing of the Perl script.

```
#!/usr/bin/perl
$filename = "relevant-output.txt";

# Open the output file, exit with an error if a problem exists.
open(INDEX, ">$filename") || die("$filename: $!");

# Select the output file for writing.
select(INDEX);

# Define an input file and open it for reading.
$infile = "full-output.txt";
open(READFILE, "<$infile") || die("$infile: $!");

# Print the relevant information.
while ($line = <READFILE>) {
      ($datetime, $gmt, $somenum, $url, $host, $requesttype, $request, ,
            $protocol, $ip) = split(" ", $line);
      print "$ip\t\t$url\t\t$datetime]\n";
}

# Reselect the standard output and close files.
select(STDOUT);
close(INDEX);
close(READFILE);
```

Appendix B

The second experiment, described in Chapter 4, was more complicated than the first and thus required more JavaScript and HTML files. Additionally, I used a CGI script to simulate communication with the web server. This appendix lists the HTML, JavaScript and CGI script used during the experiment. The questionnaire utilized is also included. No scripts were needed for analysis, however, due to the lack of substantial results. A collection of all the files and the results that were gathered are available online at the following address: http://www.danakate.com/thesis.

B.1. Questionnaire for Students

Students attempted to find the answers to a questionnaire during the second experiment. Ideally, the students would navigate to only one web page that had the answers. Some of the pages contained false security messages. Although the web server used with this survey may not be available any more, all of the answers should still be found on the main Computer Science Department web pages located at: http://www.cs.virginia.edu. This section contains the list of questions used in the experiment.

Web Usability Experiment

Please type the following URL into Netscape and bookmark it:
http://fowler.cs.virginia.edu:8080/
Then answer the following questions in order. After each question, return to this URL. If you find yourself spending more than 5 minutes on a question, feel free to skip it.
Please do not consult with anybody on these questions. Just try to answer them to the best of your ability.

1) Who was the original desk czar for the Graduate Student Group?

2) From what university did Professor of Civil Engineering, Furman W. Barton, receive his Ph.D.?

3) Which three CS Faculty members have won ITR awards?

4) Whose office is adjacent to Professor Luebke's office?

5) If you are having a problem with an on-grounds computer outside of business hours, whom should you call first?

6) How many job openings are there for a research scientist in Wide-Area Metasystems?

7) According to the web site, what are the four main reasons why incoming students should choose Computer Science as their major?

8) Which faculty member founded the webteam, and what type of dancing does he consider a hobby?

9) What faculty member is Director of the CS web team and what course is (s)he currently teaching?

10) Who had the original idea for The Oracle of Bacon?

11) Which two professors joined the UVA CS department in 2001?

12) Which professor is in charge of the Graduate Student Orientation seminar?

13) Which computer science courses are undergraduates to take during their fifth semester?

14) In the CS lounge, if you wanted to play the game Vasdasz a 2x3x2 Colored Sliding Puzzle, would you be able to find it?

B.1. JavaScript for Pop-up Messages

Four of the questions in the survey were modified to contain false security messages. The code for each of the messages was identical except for the file used for the message content. Those files were named "alert1.html," "alert2.html," "alert3.html," and "alert4.html." The following is a listing of the JavaScript. The text in bold marks where the name of the message files should be written. This script is different from the one in the first experiment. Instead of displaying the pop-up message in the center of the user's monitor, it appears in the center of the web browser window. Unfortunately, this code does not work in Microsoft's Internet Explorer.

```
<script language="javaScript">
<!--

// Store the height and width of the user's monitor.
var winbrowx = screenX;
var winbrowy = screenY;

// Store the height and width of the browser window.
var maxbroww = outerWidth;
var maxbrowh = outerHeight;

// Define the height and width of the pop-up message.
var popupwidth = 300.0;
var popupheight = 175.0;

// Calculate the location of the center of the browser window.
var newleft = winbrowx + (maxbroww / 2.0) - (popupwidth / 2.0);
var newtop = winbrowy + (maxbrowh / 2.0) - (popupheight / 2.0);

// Display the pop-up with the appropriate source file.
remote =
window.open('../alertfile','alert','toolbar=no,location=no,directories=no,stat
us=no,menubar=no,scrollbars=no,resizable=no,width='+popupwidth+',height='+popu
pheight+',top='+newtop+',left='+newleft+'')

//-->
</script>
```

B.2. HTML for Standard Pop-up Messages

Two of the alert files used in the experiment looked similar to a standard security warning. The HTML for this message is similar to the message used in the first experiment. However, instead of redirecting the user to a web site, the buttons accessed a CGI script that triggered a response from the server and logged the user's choice. I also incorporated unique names, in the form of "alert<alert-number>" for the buttons. The text in bold marks where the button name is written. The HTML for the alert pages is given here.

```
<html>
<head>
<title>Security Information</title>
</head>
<body bgcolor="#d4d0c8" text="#000000">
<center>
<br>
<table width="95%" cellpadding="2" cellspacing="2" border="0">
   <tr>
      <td>
      <font size="3">This page contains both secure and non-secure items.  Do
      you wish to download the non-secure items?</font>
      <br></td>
</tr>
</table>
</center>
<br>
<br>
<center>
<table width="100%" cellpadding="1" cellspacing="1" border="0">
   <tr>
      <td width="50%" align="right">
      <form method="get" action="../cgi-bin/dummy.cgi"
      onSubmit="setTimeout('self.close();',100);">
      <input type="submit" name="alert#" value="Continue">
      </form>
      </td>
      <td width="50%">
      <form method="get" action="../cgi-bin/dummy.cgi"
      onSubmit="setTimeout('self.close();',100);">
      <input type="submit" name="alert#" value="Cancel">
      </form>
      </td>
   </tr>
</table>
</center>
</body>
</html>
```

B.3. HTML for Non-standard Pop-up Messages

The experiment also presented two non-standard security messages to the user. The message was designed to be informative and noticeable. Similar to the standard messages, these pop-ups incorporated a CGI script as well as unique button names. The text in bold marks where the button name is written. The HTML for the alert pages is given here.

```
<html>
<head>
<title>Security Information</title>
</head>
<body bgcolor="#d4d0c8" text="#000000">
<center>
<br>
<table width="95%" cellpadding="2" cellspacing="2" border="0">
   <tr>
      <td>
      <font size="3">This page is attempting to access your <font
      color="#e41b17"><b>password</b></font> file. Do you wish to continue this
action?</font>

      <br></td>
   </tr>
</table>
</center>
<br>
<br>
<center>
<table width="100%" cellpadding="1" cellspacing="1" border="0">
   <tr>
      <td width="50%" align="right">
      <form method="get" action="../cgi-bin/dummy.cgi"
      onSubmit="setTimeout('self.close();',100);">
      <input type="submit" name="alert#" value="Continue">
      </form></td>
      <td width="50%">
      <form method="get" action="../cgi-bin/dummy.cgi"
      onSubmit="setTimeout('self.close();',100);">
      <input type="submit" name=" alert#" value="Cancel">
      </form></td>
   </tr>
</table>
</center>
</body>
</html>
```

B.4. CGI Script

Because I was unable to redirect users to another web site based upon their choice of button, I had to find some other way to evoke a response from the server that would log which button they chose. When the user clicked on a button this CGI script executed informing the web server an event occurred. The script indicated that no return response was needed, so the web server only logged the event (i.e. the button the user pressed) and the JavaScript closed the message window. A source code listing of the CGI script is given below.

```
#!/usr/bin/perl5
print "Status: 204 No Content\n";
print "Content-type: text/html\n\n";

print "No content\n";
```

Appendix C

The lack of meaningful results in the second experiment prompted the development of a third. I used the Oracle of Bacon's Star Links page (http://www.cs.virginia.edu/oracle/star_links.html) to set up more false security messages. Again, JavaScript, HTML and a CGI script were necessary to create the pop-up messages and evoke a response from the server that logged the user's decision. This appendix describes the pop-up message files as well as the scripts used to analyze the results. Due to the size of the log files, I did not present them in this appendix. All files, including the access logs, are available online at: http://www.danakate.com/thesis.

C.1. JavaScript for Pop-up Messages

One problem observed in the second experiment, described in Chapter 4, was that centering the pop-up message in the user's web browser was not possible using Microsoft's Internet Explorer. A large portion of the public uses Internet Explorer. Restricting access to the Star Links page was unfeasible. Therefore, I had to come up with some way of fixing the problem. JavaScript has the ability to detect web browsers. Thus, I created a script that detected the user's web browser and displayed the pop-up message in the center of the browser window if the user used Netscape Navigator, in the center of the entire monitor if the user used Internet Explorer, and not at all if another browser was used. Attempting to account for all possible web browsers in use was not possible, so I had to limit myself to the two most popular. This section contains the JavaScript code I used on the Star Links page. The text in bold refers to text that was changed between the four days the experiment was run. The possible replacement texts were "alert1.html," "alert2.html," "alert3.html," and "alert4.html."

```
<script language="javaScript">
<!--
// Create a function to detect what browser the viewer uses.
function whatbrowser(browser) {
  browseris = false;
  browseris = (navigator.appName.indexOf(browser) != -1);
  return browseris;
}

// If the browser is Netscape, show the pop-up in the middle of the browser.
if (whatbrowser('Netscape') == true) {
  // Store the left edge and top edge of the browser window.
  var winbrowx = screenX;
  var winbrowy = screenY;

  // Store the outside height and outside width of the browser window.
  var maxbroww = outerWidth;
  var maxbrowh = outerHeight;

  // Define the height and width of the pop-up message.
  var popupwidth = 300.0;
  var popupheight = 175.0;

  // Calculate the location of the center of the browser window.
  var newleft = winbrowx + (maxbroww / 2.0) - (popupwidth / 2.0);
  var newtop = winbrowy + (maxbrowh / 2.0) - (popupheight / 2.0);

  // Display the pop-up with the appropriate source file.
  remote =
window.open('alert#.html','alert','toolbar=no,location=no,directories=no,statu
s=no,menubar=no,scrollbars=no,resizable=no,width='+popupwidth+',height='+popup
height+',top='+newtop+',left='+newleft+'')
}

// If the browser is Internet Explorer, show the pop-up in the middle of
// the monitor.
if (whatbrowser('Microsoft') == true) {
  // Store the height and width of the user's monitor.
  var maxwidth = screen.width;
  var maxheight = screen.height;

  // Define the height and width of the pop-up message.
  var popupwidth = 300.0;
  var popupheight = 175.0;

  // Calculate the location of the center of the monitor screen.
  var newleft = (maxwidth / 2.0) - (popupwidth / 2.0);
  var newtop = (maxheight / 2.0) - (popupheight / 2.0);

  // Display the pop-up with the appropriate source file.
  remote
=window.open('alert#.html','alert','toolbar=no,location=no,directories=no,stat
us=no,menubar=no,scrollbars=no,resizable=no,width='+popupwidth+',height='+popu
pheight+',top='+newtop+',left='+newleft+'')
}
//-->
</script>
```

C.2. HTML for Pop-up Messages

The experiment utilized four different pop-up messages. Each differed in look but had the same content. Graphical representations of each message are presented in Chapter 5. This section lists the HTML for the messages used.

C.2.1. Pop-up Message for Day One

This is the HTML code for the pop-up message appearing on first day of the experiment.

```
<html>
<head>
<title>Security Information</title>
</head>
<body bgcolor="#bfbfbf" text="#000000">
<center>
<br>
<table width="95%" cellpadding="2" cellspacing="2" border="0">
   <tr>
      <td>
      <img src="anim-warn.gif">
      </td>
      <td>
      <font size="-1">This page is attempting to access your <font
      color="#e41b17"><b>password</b></font> file.  Do you wish to
      continue?</font>
      <br></td>
   </tr>
</table>
</center>
<br>
<br>
<center>
<table width="100%" cellpadding="1" cellspacing="1" border="0">
   <tr>
      <td width="50%" align="right">
      <FORM METHOD=get action="../../cgi-bin/dummy.cgi"
      onSubmit="setTimeout('self.close();',100);">
      <INPUT TYPE="SUBMIT" NAME="alert1" VALUE="Continue">
      </FORM></td>
      <td width="50%">
      <FORM METHOD=get action="../../cgi-bin/dummy.cgi"
      onSubmit="setTimeout('self.close();',100);">
      <input type="SUBMIT" name="alert1" value="Cancel">
      </FORM></td>
   </tr>
</table></center>
</body>
</html>
```

C.2.2. Pop-up Message for Day Two

This is the HTML code for the pop-up message appearing on second day of the experiment.

```
<html>
<head>
<title>Security Information</title>
</head>
<body bgcolor="#bfbfbf">
<center>
<br>
<table width="95%" cellpadding="2" cellspacing="2" border="0">
   <tr>
      <td>
      <font size="-1" color="#e41b17">This page is attempting to access your
      <b>password</b> file.  Do you wish to continue?</font>
      <br></td>
   </tr>
</table>
</center>
<br>
<br>
<center>
<table width="100%" cellpadding="1" cellspacing="1" border="0">
   <tr>
      <td width="50%" align="right">
      <FORM METHOD=get action="../../cgi-bin/dummy.cgi"
      onSubmit="setTimeout('self.close();',100);">
      <INPUT TYPE="SUBMIT" NAME="alert2" VALUE="Continue">
      </FORM></td>
      <td width="50%">
      <FORM METHOD=get action="../../cgi-bin/dummy.cgi"
      onSubmit="setTimeout('self.close();',100);">
      <input type="SUBMIT" name="alert2" value="Cancel">
      </FORM></td>
   </tr>
</table>
</center>
</body>
</html>
```

C.2.3. Pop-up Message for Day Three

This is the HTML code for the pop-up message appearing on third day of the experiment.

```
<html>
<head>
<title>Security Information</title>
</head>
<body bgcolor="#e41b17">
<center>
<br>
<table width="95%" cellpadding="2" cellspacing="2" border="0">
   <tr>
      <td>
      <font size="-1" color="#000000">This page is attempting to access your
      <font
      color="#ffffff"><b>password</b></font> file.  Do you wish to
      continue?</font>
      <br></td>
   </tr>
</table>
</center>
<br>
<br>
<center>
<table width="100%" cellpadding="1" cellspacing="1" border="0">
   <tr>
      <td width="50%" align="right">
      <FORM METHOD=get action="../../cgi-bin/dummy.cgi"
      onSubmit="setTimeout('self.close();',100);">
      <INPUT TYPE="SUBMIT" NAME="alert3" VALUE="Continue">
      </FORM></td>
      <td width="50%">
      <FORM METHOD=get action="../../cgi-bin/dummy.cgi"
      onSubmit="setTimeout('self.close();',100);">
      <input type="SUBMIT" name="alert3" value="Cancel">
      </FORM></td>
   </tr>
</table>
</center>
</body>
</html>
```

C.2.4. Pop-up Message for Day Four

The last day of experiment had a complicated pop-up message. The message incorporated HTML and JavaScript. The JavaScript verified that the user typed the word "yes" or "no" into the text box if they wanted to press the "Done" button. The code is listed below.

```
<html>
<head>
<title>Security Information</title>
<script language="javaScript">
<!—

// Check the text in the text box when the "Done" button is pressed.
function checktext() {
  if (document.secform.alert4.value == 'yes' ) {
    document.secform.submit();
    self.close();
  }

  if (document.secform.alert4.value == 'no') {
    document.secform.submit();
    self.close();
  }
}
//-->
</script>

</head>
<body bgcolor="#bfbfbf" text="#000000">
<center>
<br>
<table width="95%" cellpadding="2" cellspacing="2" border="0">
   <tr>
      <td>
      <font size="-1">This page is attempting to access your <font
      color="#e41b17"><b>password</b></font> file.  Do you wish to allow this
      action?</font>
      <br></td>
   </tr>
</table>
</center>
<br>
<br>
<center>
```

```
<table width="100%" cellpadding="1" cellspacing="1" border="0">
   <tr>
      <td width="50%">
      <form name="secform" method="get" action="../../cgi-bin/dummy.cgi"
      onSubmit="setTimeout('self.close();',100);">
      yes or no:
      <INPUT TYPE="text" NAME="alert4" size="5" maxlength="5"
      value="">  
      <input type="button" name="alert4sub" value="Done"
      onClick="checktext()">  
      <input type="submit" name="alert4can" value="Cancel"
      onSubmit="setTimeout('self.close();',100);">
      </FORM></td>
   </tr>
</table>
</center>
</body>
</html>
```

C.3. CGI Script

The CGI script used in this experiment is identical in function and code as in the third experiment. The code is located in Appendix B.

C.4. Analysis Scripts

In order to analyze the log files accurately, I created two Perl analysis scripts. The first script was used with the first three days of experimentation since they were all similar in format. The last day of the experiment had a message that was more complicated to analyze and required another script. This section provides a listing of both scripts and a description of their use. Samples from the log files are also presented for a clearer view of how the scripts work.

C.4.1. Analyzing the First Three Days

Analyzing the log files involved three major steps. First, I needed to sort the users by hostname. Then, I had to categorize each user, or unique hostname, by behavior. Finally, I had to tally the number and percentage of users in each category. The categories provided a basis for interpretation, which is presented in Chapter 5. To take care of the different categories in which each user could be a member, I created a state transition table. This table dictated what category, or state, the user would move to if the next action were "Continue" or "Cancel." The script is listed here.

```
#!/usr/bin/perl
# Create a state transition table using a hash table that determines
#   what category a user moves to if they click "Continue" or "Cancel."
# The key for the hash table is the user's current state.  The value, in
#   list format, is the state to which the user changes.  The first value
#   is the state change when the user clicks "Continue."  The second value
#   is the state change when the user clicks "Cancel."
%state_transitions = (
      initial                 => ["continue", "cancel"],
      cancel                  => ["cancel_then_continue", "always_cancel"],
      continue                => ["always_continue", "continue_then_cancel"],
      always_cancel           => ["cancel_then_continue", "always_cancel"],
      always_continue         => ["always_continue", "continue_then_cancel"],
      cancel_then_continue    => ["cancel_then_continue", "other"],
      continue_then_cancel    => ["other", "continue_then_cancel"],
      other                   => ["other", "other"],
);

# Define a sub-function called "get_line," which takes no parameters.
# This function reads one line of text from the specified file handler
#   and, if it exists, determines if the action was "Continue" or
#   "Cancel."  Then, it returns the hostname and the action.
sub get_line {
      $one_line = <READFILE>;
      if (!$one_line) {
            return 1;
      }

      ($hostname) = split(" ", $one_line);
      if ($one_line =~ m/Continue/) {
            $action_type = "continue";
      }
      elsif ($one_line =~ m/Cancel/) {
            $action_type = "cancel";
      }
```

```
      else {
            $action_type = "unknown";
      }

      return ($hostname, $action_type);
}

# Get the file to open from the command line and open it for reading.
$filename = $ARGV[0];
open(READFILE, "<$filename") || die("$filename: $!");

# Get the file to write to from the command line and open it for writing.
$outfile = $ARGV[1];
open(WRITEFILE, ">$outfile") || die("$outfile: $!");

# Select the output file as the default file handler.
select(WRITEFILE);

while ((($host, $action) = get_line()) != 1) {
      if (!exists($client_states{$host})) {
            $client_states{$host} = "initial";
      }
      $client_state = $client_states{$host};
      # Change the user's state accordingly.
      if ($action eq "continue") {
            $new_state = $state_transitions{$client_state}->[0];
      }
      else {
            $new_state = $state_transitions{$client_state}->[1];
      }
      # Store the new state.
      $client_states{$host} = $new_state;
}
close(READFILE);

# Print the hash table by hostname, then final state.
map { print "$_ => $client_states{$_}\n" } keys %client_states;

$client_count = scalar keys %client_states;

print "\n\ntotal number of clients: $client_count\n\n";

# Determine the number and percentage of users in each state.
foreach $state (values %client_states) {
      if (!exists($state_count{$state})) {
            $state_count{$state} = 0;
      }

      $state_count{$state}++;
}

foreach $state (keys %state_count) {
      print "$state_count{$state} clients in $state state - (",
            ($state_count{$state} / $client_count) * 100, " %)\n";
}
select(STDOUT);
close(WRITEFILE);
```

C.4.2. Analyzing the Final Day

The data collected for the last day of testing was somewhat different than the other days and required a different kind of analysis script. The most important change was that there were more states of which to keep track. There were also more options to test against. The script functions in the same manner as the script for the first three days. Below is a listing of the script used to analyze the final day of experimentation.

```
#!/usr/bin/perl
# Create a state transition table using a hash table that determines
#   what category a user moves to if they choose "yes," "no," or "Cancel."
# The key for the hash table is the user's current state.  The value, in
#   list format, is the state to which the user changes.  The first value
#   is the state change when the user chooses "yes."  The second value
#   is the state change when the user chooses "no."  The third value is
#   the state change when the user chooses "Cancel."
%state_transitions = (
   initial              => ["yes", "no", "cancel"],
   cancel               => ["cancel_then_yes", "cancel_then_no",
         "always_cancel"],
   always_cancel        => ["always_cancel_then_yes", "always_cancel_then_no",
         "always_cancel"],
   yes                  => ["always_yes", "yes_then_no", "yes_then_cancel"],
   no                   => ["no_then_yes", "always_no", "no_then_cancel"],
   always_yes                 => ["always_yes", "always_yes_then_no",
         "always_yes_then_cancel"],
   always_no                  => ["always_no_then_yes", "always_no",
         "always_no_then_cancel"],
   yes_then_no                => ["other", "yes_then_no", "other"],
   no_then_yes                => ["no_then_yes", "other", "other"],
   always_cancel_then_no      => ["other", "always_cancel_then_no", "other"],
   always_cancel_then_yes     => ["always_cancel_then_yes", "other", "other"],
   always_yes_then_no   => ["other", "always_yes_then_no", "other"],
   always_no_then_yes   => ["always_no_then_yes", "other", "other"],
   always_yes_then_cancel     => ["other", "other", "always_yes_then_cancel"],
   always_no_then_cancel      => ["other", "other", "always_no_then_cancel"],
   cancel_then_yes            => ["cancel_then_yes", "other", "other"],
   cancel_then_no       => ["other", "cancel_then_no", "other"],
   yes_then_cancel            => ["other", "other", "yes_then_cancel"],
   no_then_cancel       => ["other", "other", "no_then_cancel"],
   other                => ["other", "other", "other"],
);
```

```
# Define a sub-function called "get_line," which takes no parameters.
# This function reads one line of text from the specified file handler
#   and, if it exists, determines if the action was "yes," "no," or
#   "Cancel."  Then, it returns the hostname and the action.
# Note:  "Cancel" takes precedence over "yes" or "no" actions.
sub get_line {
      $one_line = <READFILE>;

      if (!$one_line) {
            return 1;
      }

      ($hostname) = split(" ", $one_line);
      if ($one_line =~ m/alert4\=no\&alert4can\=Cancel/i) {
            $action_type = "cancel";
      }
      elsif ($one_line =~ m/alert4\=yes\&alert4can\=Cancel/i) {
            $action_type = "cancel";
      }
      elsif ($one_line =~ m/\=Cancel/) {
            $action_type = "cancel";
      }
      elsif ($one_line =~ m/\=yes/i) {
            $action_type = "yes";
      }
      elsif ($one_line =~ m/\=no/i) {
            $action_type = "no";
      }
      else {
            $action_type = "unknown";
      }

      return ($hostname, $action_type);
}
# Get the file to open from the command line and open it for reading.
$filename = $ARGV[0];
open(READFILE, "<$filename") || die("$filename: $!");

# Get the file to write to from the command line and open it for writing.
$outfile = $ARGV[1];
open(WRITEFILE, ">$outfile") || die("$outfile: $!");

# Select the output file as the default file handler.
select(WRITEFILE);

while ((($host, $action) = get_line()) != 1) {

      if (!exists($client_states{$host})) {
            $client_states{$host} = "initial";
      }

      $client_state = $client_states{$host};
```

```
	# Change the user's state accordingly.
	if ($action eq "yes") {
		$new_state = $state_transitions{$client_state}->[0];
	}
	elsif ($action eq "no") {
		$new_state = $state_transitions{$client_state}->[1];
	}
	else {
		$new_state = $state_transitions{$client_state}->[2];
	}

	$client_states{$host} = $new_state;
}

close(READFILE);

# Print the hash table by hostname, then final state.
map { print "$_ => $client_states{$_}\n" } keys %client_states;

$client_count = scalar keys %client_states;

print "\n\ntotal number of clients: $client_count\n\n";

# Determine the number and percentage of users in each state.
foreach $state (values %client_states) {
	if (!exists($state_count{$state})) {
		$state_count{$state} = 0;
	}

	$state_count{$state}++;
}

foreach $state (keys %state_count) {
	print "$state_count{$state} clients in $state state - (",
		($state_count{$state} / $client_count) * 100, " %)\n";
}

select(STDOUT);
close(WRITEFILE);
```

C.4.3. Sample Results from Analysis Scripts

To understand how the scripts work, I have included partial results from the last day of testing, as it was the most complicated. The first value corresponds to the user. The last value is the user's final state. Because this listing is not complete, the final tally of responses will not correlate to what is presented here. Please refer to the online source (http://www.danakate.com/thesis) for a complete listing of results.

```
proxy03.dcsgroup.co.uk => always_cancel
nop-gw.oit.net => cancel
156.63.242.28 => cancel
d01a83cc.dip.cdsnet.net => no
209.43.103.181 => no
user-2ivekue.dialup.mindspring.com => cancel
smoore.clok.creaf.com => cancel
164.156.167.242 => cancel
cache-1.sbo.ma.webcache.rcn.net => no
senior06.tyler.temple.edu => cancel
64.241.230.20 => always_cancel
ckcfpxy1.att.com => cancel
hst-136.diablo.reshall.calpoly.edu => no
blv-proxy-06.boeing.com => always_cancel
van-58-3-209148247235.3web.net => cancel
38.204.42.243 => always_cancel
pat.ns.sigma.se => no
199.44.228.253 => other
hide195.nhs.uk => cancel
proxya.monroe.edu => always_cancel
cy397030-a.cospgs1.co.home.com => cancel
pm1-36.cak.dial.gwis.com => no
host62-7-34-184.btinternet.com => no_then_cancel
xcdfdcc3d.ip.ggn.net => no
atl200.turner.com => cancel
bp197-157.kellogg.nwu.edu => always_cancel

total number of clients: 148

4 clients in other state - (2.7027027027027 %)
33 clients in no state - (22.2972972972973 %)
65 clients in cancel state - (43.9189189189189 %)
3 clients in yes state - (2.02702702702703 %)
1 clients in always_yes_then_cancel state - (0.675675675675676 %)
2 clients in cancel_then_no state - (1.35135135135135 %)
6 clients in no_then_cancel state - (4.05405405405405 %)
2 clients in cancel_then_yes state - (1.35135135135135 %)
27 clients in always_cancel state - (18.2432432432432 %)
5 clients in always_no state - (3.37837837837838 %)
```