Crying Wolf: An Empirical Study of SSL Warning Effectiveness

Joshua Sunshine, Serge Egelman, Hazim Almuhimedi, Neha Atri, and Lorrie Faith Cranor
Carnegie Mellon University
{sunshine, egelman, hazim}@cs.cmu.edu, natri@andrew.cmu.edu, lorrie@cs.cmu.edu

Abstract

Web users are shown an invalid certificate warning when their browser cannot validate the identity of the websites they are visiting. While these warnings often appear in benign situations, they can also signal a man-in-the-middle attack. We conducted a survey of over 400 Internet users to examine their reactions to and understanding of current SSL warnings. We then designed two new warnings using warnings science principles and lessons learned from the survey. We evaluated warnings used in three popular web browsers and our two warnings in a 100-participant, between-subjects laboratory study. Our warnings performed significantly better than existing warnings, but far too many participants exhibited dangerous behavior in all warning conditions. Our results suggest that, while warnings can be improved, a better approach may be to minimize the use of SSL warnings altogether by blocking users from making unsafe connections and eliminating warnings in benign situations.

1 Introduction

Browsers display Secure Socket Layer (SSL)[1] warnings to warn users about a variety of certificate problems, for example when the server's certificate has expired, mismatches the address of the server, or is signed by an unrecognized authority. These warning messages sometimes indicate a man-in-the-middle or DNS spoofing attack. However, much more frequently users are actually connecting to a legitimate website with an erroneous or self-signed certificate.

The warnings science literature suggests that warnings should be used only as a last resort when it is not possible to eliminate or guard against a hazard. When warnings are used, it is important that they communicate clearly about the risk and provide straightforward instructions for avoiding the hazard [19, 22]. In this paper we examine user reactions to five different SSL warnings embodying three strategies: make it difficult for users to override the warning, clearly explain the potential danger facing users, and ask a question users can answer. By making it difficult for users to override the warning and proceed to a potentially dangerous website, the warning may effectively act as a guard against the hazard, similarly to the way a fence protects people from falling into a hole. While some people may still climb the fence, this requires extra effort. By clearly explaining the potential danger, warnings communicate about risk. Finally, by asking users a question they can answer, the system can tailor a warning to the user's situation and instruct users in the appropriate steps necessary to avoid any hazard.

We conducted a survey of 409 Internet users' reactions to current web browser SSL warnings and found that risk perceptions were the leading factor in respondents' decisions of whether or not to visit a website with an SSL error. However, those who understood the risks also perceived some common SSL warnings as not very risky, and were more likely to override those warnings.

[1] The Secure Socket Layer (SSL) and Transport Layer Security (TLS) protocols secure web communication by encrypting data sent between browser and server and by validating the identity of the server. For the remainder of the paper we will use the common convention of using the term "SSL" to refer to both protocols.

We followed up this survey with a between-subjects laboratory experiment involving 100 participants who encountered SSL warnings on an online banking website that requested their credentials and a library website that did not request any credentials. We tested the Firefox 2 (FF2), Firefox 3 (FF3), and Microsoft Internet Explorer 7 (IE7) SSL warnings. We also tested two new warnings designed to take advantage of the lessons we learned in the survey. The first warning was designed with risk in mind: it succinctly explained the risks and consequences of proceeding to the website. The second warning was context sensitive: it appeared to be more severe when the participants visited websites that required them to enter personal data. We found that most participants ignored the FF2 and IE7 warnings on both websites. Many participants who used FF3 were unable to override that warning and were thus prevented from visiting both websites. Finally, we found that participants who viewed our redesigned warnings better understood the risks and made their decisions based on the type of website they were visiting. However, despite the fact that the warnings we examined embodied the best techniques available, none of the warnings provided adequate protection against man-in-the-middle attacks. Our results suggest that, while warnings can be improved, a better approach may be to minimize the use of SSL warnings altogether by blocking users from making unsafe connections and eliminating warnings in benign situations.

In the next section we provide an overview of other studies that have been conducted on web browser security indicators. In Section 3 we present our online SSL warning survey methodology and results. In Section 4 we present our laboratory experiment methodology and results. Finally, we discuss our findings and conclusions.

2 Background and Related Work

Much previous research has indicated that users do not understand SSL. A study in 2002 found that half of the participants could not identify a secure browser connection [8]. A 2005 study tracked eye movements and found that participants paid no attention to web browser security cues such as SSL icons. Only after priming participants to be on the lookout for security information, 69% of participants noticed the lock icon [21]. Schechter et al. tested the usability of security indicators by removing SSL indicators from a banking website and observed that all 63 participants still provided their passwords [17].

The major web browsers now include support for extended validation (EV) certificates. A regular certificate only tells a user that the certificate was granted by a particular issuing authority, whereas an EV certificate also says that it belongs to a legally recognized corporate entity [2]. FF3 and IE7 indicate a website has an EV certificate by coloring the address bar green and displaying the name of the website owner. However, a study by Jackson et al. found that EV certificates did not make users less likely to fall for phishing attacks. Many users were confused when the chrome of the web browser was spoofed within the content window to depict a green address bar. Additionally, after reading a help file, users were less suspicious of fraudulent websites that did not yield warning indicators [11]. Sobey et al. performed an eye tracking study in 2008 to examine whether participants would notice simulated versions of the EV certificate indicators that are used by FF3. They found that none of their 28 participants examined the address bar when making online shopping decisions, and therefore none of them encountered the secondary SSL dialogs containing information about the website owners [18].

Usability problems with security indicators in web browsers go beyond SSL. Wu et al. conducted a study of security toolbars used to help users identify phishing websites. The researchers examined three different styles of passive indicators—indicators that do not force user interactions—that appeared in the browser chrome. They discovered that 25% of the participants failed to notice the security indicators because they were focused on the primary task. In fact, many of those who did notice the indicators did not trust them because they believed the tool was in error since the website looked trustworthy [23]. The factors that go into website trust have been extensively studied by Fogg et al., who found that the "look and feel" of a website is often most important for gaining user trust [7]. Thus users might trust a professional looking website despite the presence of a passive security indicator. Dhamija et al. corroborated these findings by performing a study on why people fall for phishing websites. In their study, users examined a set of websites and were asked to identify which ones were phishing websites. They found that 23% of their study participants did not look at any of the web browser security indicators when making their decisions, even though the participants were primed for security. The researchers concluded that passive security indicators are ineffective because they often go unnoticed [4].

Because of the problems with passive security indicators, many web browsers now display "active" warnings that require the user to take an action—usually deciding whether or not to visit the destination website—in order to dismiss the warning. While these warnings force the user to acknowledge them, they still allow the user to ignore their advice and proceed to the website despite the security error. In 2008, Egelman et al. performed a study on active web browser warnings used to warn users about potential phishing websites. They discovered that users who claimed to have seen the warnings before were significantly more likely to ignore them in the laboratory. They concluded that many of the participants had become habituated to seeing similar-looking warnings when browsing legitimate websites, and are now likely to ignore all future similarly-designed warnings, regardless of the danger they represent [6].

Jackson and Barth address the problem of users ignoring SSL warnings with the ForceHTTPS system [10]. Websites with CA signed certificates deploy a special ForceHTTPs cookie to a user's browser, which from then on only accepts valid SSL connections to the website. This strategy is elegantly simple, but it does not protect users when they encounter a website for the first time.

Wendlandt et al. created the Perspectives system to prevent habituation by only displaying warnings when an attack is probable. Perspectives transforms the CA model into a "trust-on-first-use" model, similar to how SSH works. "Notaries" keep track of all previously viewed SSL certificates and only warn users when they encounter a certificate that has changed over time. This eliminates many common SSL errors, thereby only displaying warnings when an attack is probable [20]. However, when users do encounter certificates that have been altered, it is unclear how the warnings should be designed so as to maximize their effectiveness.

Xia and Brustoloni implement a system to help users better react to unverified certificates [24]. The system requires websites interested in using private CA signed certificates to distribute tokens to their users by physical media. In 2007, Brustoloni and Villamarín-Salomón explored the idea of creating polymorphic dialogs to combat habituation. While their preliminary results were promising for warning users about malicious email attachments, it is unclear what the long-term efficacy would be if such a system were created for SSL warnings [1].

The pervasive nature of SSL errors raises questions about the efficacy of SSL warnings. A survey of 297,574 SSL-enabled websites queried in January 2007 found 62% of the websites had certificates that would trigger browser warnings [5]. A January 2009 study performed using a list of the top one million websites found that at least 44% of the 382,860 SSL-enabled websites had certificates that would trigger warnings [13].[2] Given this large sample, many of the errors may appear on websites that are not frequently visited. Our own analysis of the top 1,000 SSL-enabled websites yielded 194 SSL errors, which is still an alarming number. Unfortunately, we do not have data on the proportion of certificate errors that appear on legitimate websites versus malicious websites, making it unclear whether these particular errors are indicative of an ongoing attack. However, we believe it is likely that most certificate errors occur on non-malicious websites, and therefore many users view the associated warnings as false positives. This means that if a web browser displays a particular warning each time it encounters any type of certificate error, users will quickly become habituated to this warning regardless of the underlying error.

[2] This estimate is likely low as the 2009 study did not catalog domain name mismatch errors.

3 SSL Survey

In the summer of 2008 we conducted an online survey of Internet users from around the world to determine how they perceived the current web browser SSL warnings.

3.1 Methodology

We presented survey respondents with screenshots of three different SSL warnings from the browser that they were using at the time they took the survey[3] and asked them several questions about each warning. These questions were followed by a series of questions to determine demographic information.

We showed participants warnings for expired certificates, certificates with an unknown issuer, and certificates with mismatched domain names.[4] Each warning was shown on a separate page along with its associated questions, and the order of the three pages was randomized. We included a between-group condition to see if context played a role in users' responses: half the participants were shown a location bar for *craigslist.org*—an anonymous forum unlikely to collect personal information—and the other half were shown a location bar for *amazon.com*—a large online retailer likely to collect personal and financial information. We hypothesized that respondents might be more apprehensive about ignoring the warning on a website that was likely to collect personal information. Below each warning screenshot, participants were asked a series of questions to determine whether they understood what the warnings mean, what they would do when confronted with each warning, and their beliefs about the consequences of ignoring these warnings.

We were also interested in determining how computer security experts would respond to our survey, and if the experts' answers would differ from everyone else's answers. In order to qualify respondents as experts, we asked them a series of five questions to determine whether they had a degree in an IT-related field, computer security job experience or course work, knowledge of a programming language, and whether they had attended a computer security conference in the past two years.

We recruited participants from Craigslist and several contest-related bulletin boards, offering a gift certificate drawing as an incentive to complete the survey. We received 615 responses; however we used data from only the 409 respondents who were using one of the three web browsers under study.

[3]We used screenshots of the warnings from FF2, FF3, and IE7. Users of web browsers other than FF2, FF3, or IE7 were only asked the demographic questions.

[4]We examined these three warnings in particular because we believed them to be the most common.

3.2 Analysis

Our 409 survey respondents used the following browsers: 96 (23%) used FF2, 117 (29%) used FF3, and 196 (48%) used IE7. While age and gender were not significant predictors of responses,[5] it should be noted that 66% of our respondents were female, significantly more males used FF3 ($\chi^2_2 = 34.01$, $p < 0.0005$), and that IE7 users were significantly older ($F_{2,405} = 19.694$, $p < 0.0005$). For these reasons and because respondents self-selected their web browsers, we analyzed the responses for each of the web browsers separately.

We found no significant differences in responses based on the type of website being visited. We found that respondents' abilities to correctly explain each warning was a predictor of behavior, though not in the way we expected: respondents who understood the domain mismatch warnings were less likely to proceed whereas we observed the opposite effect for the expired certificate warnings. This suggests that participants who understood the warnings viewed the expired certificate warnings as low risk. Finally, we found that risk perceptions were a leading factor in respondents' decisions and that many respondents—regardless of expertise—did not understand the current warnings. In this section we provide a detailed analysis of our results in terms of warning comprehension and risk perceptions, the role of context, and the role of expertise.

[5]All statistics were evaluated with α=0.05. We used a Fisher's exact test for all statistics where we report a p-value only.

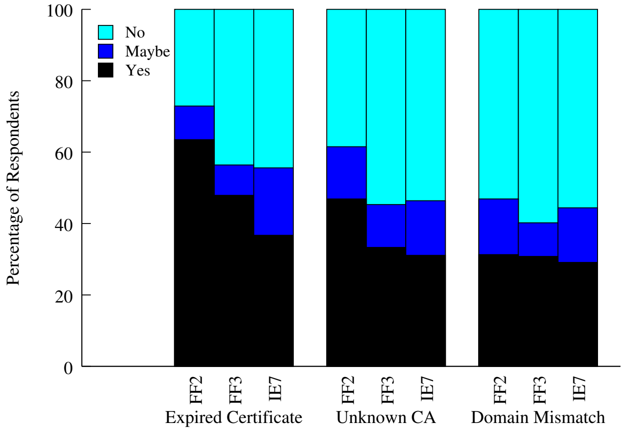


Figure 1: Participant responses to the question: *If you saw this message, would you attempt to continue to the website?*

3.2.1 Comprehension and Risk Perceptions

We were primarily interested in whether respondents would continue to the destination website if they saw a given warning. As shown in Figure 1, less than half the participants claimed they would continue.

We expected to see differences in behavior for each of the three types of warnings. In order for this to be the case, participants needed to be able to distinguish each of the three warnings. We asked them to explain what they thought each warning meant and coded the answers in terms of whether or not they were correct. As shown in Table 1, we discovered that FF2 users were significantly more likely to understand the domain mismatch warnings, while FF3 users were significantly more likely to understand the expired certificate warnings.

We explored warning comprehension further by examining whether those who understood the meaning of the warnings were more likely to heed or ignore them. In general, we found that users who understood the warnings tended to behave differently than those who did not. Across all three browsers, users who understood the domain mismatch warning were more likely to say they would heed that warning than users who did not understand it. In addition, FF3 and IE7 users who understood the expired certificate warnings were more likely to indicate that they would ignore these warnings and proceed to the destination website. These results are detailed in Table 1 and indicate that users likely perceive less risk when encountering an expired certificate, and therefore are likely to proceed. However, when encountering a domain mismatch warning, knowledgeable users perceive greater risk and are likely to discontinue.

The three warnings that we examined are displayed when the authenticity of the destination website's SSL certificate cannot be guaranteed. While each of these warnings represents a different underlying error, they represent the same threat: the user may not be communicating with the intended website or a third party may be able to eavesdrop on her traffic. In both cases, sensitive information may be at risk (e.g. billing information when performing an online purchase). In order to determine whether or not respondents understood the threat model, we asked them to list the possible consequences of ignoring each of the warnings. Responses that specifically mentioned fraud, identity theft, stolen credentials (or other personal information), phishing, or eavesdropping were coded as being correct. We coded as correct 39% of responses for FF2 warnings, 44% of responses for FF3 warnings, and 37% of responses for IE7 warnings.

Incorrect responses fell into two categories: respondents who had no idea (or said there were no consequences) and respondents who mentioned other security threats. Many of those in the latter category mentioned viruses and worms. While it is possible that a malicious website may exploit web browser vulnerabilities or trick visitors into downloading malware, we considered these outside the scope of our survey because they either impact only users of a specific browser version—in the case of a vulnerability—or they rely on the user taking additional actions—such as downloading and executing a file. Several responses mentioned malware but additionally claimed that those using up-to-date security software are not at risk. Others claimed they were not at risk due to their operating systems:

> "I use a Mac so nothing bad would happen."
> "Since I use FreeBSD, rather than Windows, not much [risk]."

Browser	Understood	Expired Certificate				Unknown CA			Domain Mismatch			
				Ignored				Ignored			Ignored	
FF2	Y	48	50%	71%		37	39%	43%	57	59%	19%	$\chi^2_2 = 9.40$
	N	48	50%	56%		59	61%	49%	39	41%	49%	$p < 0.009$
FF3	Y	55	47%	64%	$\chi^2_2 = 21.05$	35	30%	31%	46	39%	15%	$\chi^2_2 = 8.65$
	N	62	53%	34%	$p < 0.0005$	82	70%	34%	71	61%	41%	$p < 0.013$
IE7	Y	45	23%	53%	$\chi^2_2 = 11.81$	44	22%	27%	62	32%	16%	$\chi^2_2 = 7.50$
	N	151	77%	32%	$p < 0.003$	152	78%	32%	134	68%	35%	$p < 0.024$

Table 1: Participants from each condition who could correctly identify each warning, and of those, how many said they would continue to the website. Differences in comprehension within each browser condition were statistically significant (FF2: $Q_2 = 10.945$, $p < 0.004$; FF3: $Q_2 = 11.358$, $p < 0.003$; IE7: $Q_2 = 9.903$, $p < 0.007$). For each browser condition, the first line depicts the respondents who could correctly define the warnings, while the second depicts those who could not. There were no statistically significant differences between correctly understanding the unknown CA warning and whether they chose to ignore it.

> "On my Linux box, nothing significantly bad would happen."

Of course, operating systems or the use of security software do not prevent a user from submitting form data to a fraudulent website, nor do they prevent eavesdropping. We further examined risk perceptions by asking participants to specify the likelihood of "something bad happening" when ignoring each of the three warnings, using a 5-point Likert scale ranging from "0% chance" to "100% chance." We found significant differences in responses to each warning for all three web browsers: respondents consistently ranked the expired certificate warning as being less risky than both of the other warnings. Table 2 depicts the perceived likelihood of risk for each of the web browsers and each of the three SSL warnings.

To examine whether there were differences in risk perception based on the underlying SSL error, we asked respondents to quantify the severity of the consequences of ignoring each of the SSL warnings using a 5-point Likert scale that ranged from "none" to "moderate" to "severe." As shown in Table 3, we found that respondents in every web browser condition were likely to assign significantly lesser consequences to ignoring the expired certificate warning than when ignoring either of the other two warnings.

3.2.2 The Role of Expertise

Finally, we wanted to examine whether respondents' level of technical expertise influenced their decisions to heed or ignore the warnings. As described in Section 3.1, we asked respondents a series of five questions to gauge their technical qualifications. We assigned each respondent a "tech score" corresponding to the number of questions they answered affirmatively. The first column of Table 4 lists the average scores for each of the web browser conditions. We classified those with tech scores greater than or equal to two as "experts." The expert group represented the top 16.7% of FF2 users, the top 26.5% of FF3 users, and the top 12.2% of IE7 users. We compared our "experts" to the rest of our sample (i.e. respondents with scores of zero or one) and found that responses did not significantly differ in most cases. We found significant differences only among FF3 users when viewing the unknown CA and domain mismatch warnings: experts were significantly less likely to proceed to the websites (Table 4).

Finally, we examined whether the experts were better able to identify the individual warnings than the rest of the sample. We found that while the experts were more likely to identify the warnings than non-

	Expired Certificate	Unknown CA	Domain Mismatch	
FF2	37%	45%	54%	$\chi^2_2 = 25.19$ $p < 0.0005$
FF3	42%	52%	50%	$\chi^2_2 = 13.47$ $p < 0.001$
IE7	47%	52%	53%	$\chi^2_2 = 12.79$ $p < 0.002$

Table 2: Mean perceptions of the likelihood of "something bad happening" when ignoring each warning, using a 5-point Likert scale ranging from 0 to 100% chance. A Friedman test yielded significant differences for each browser.

	Expired Certificate	Unknown CA	Domain Mismatch	
FF2	1.70	2.10	2.29	$\chi^2_2 = 20.49$ $p < 0.0005$
FF3	1.96	2.36	2.32	$\chi^2_2 = 9.00$ $p < 0.011$
IE7	2.14	2.36	2.34	$\chi^2_2 = 16.90$ $p < 0.0005$

Table 3: Mean perceptions of the consequences of ignoring each of the three warnings, using a 5-point Likert scale ranging from 0 to 4. A Friedman test shows that respondents in every web browser condition were likely to assign significantly lesser consequences to ignoring the expired certificate warning than when ignoring either of the other two warnings.

experts, even in the best case, the experts were only able to correctly define the expired certificate warnings an average of 52% of the time, the unknown CA warnings 55% of the time, and the domain mismatch warnings 56% of the time. This indicates that either our metric for expertise needs to be improved, or that regardless of technical skills, many people are unable to distinguish between the various SSL warnings.

3.2.3 Conclusion

Our survey showed how risk perceptions are correlated with decisions to obey or ignore security warnings and demonstrated that those who understand security warnings perceive different levels of risk associated with each warning. However, a limitation of surveys is they collect participants' self-reported data about what they think they would do in a hypothetical situation. Thus, it is useful to validate survey findings with experimental data.

4 Laboratory Experiment

We conducted a laboratory study to determine the effect of SSL warnings on user behavior during real tasks.

4.1 Methodology

We designed our laboratory study as a between-subjects experiment with five conditions: FF2 (Figure 2(a)), FF3 (Figure 3), IE7 (Figure 2(b)), a single-page redesigned warning (Figure 4(b)), and a multi-page redesigned warning (Figure 4). We asked participants to find information using four different types of information sources. Each task included a primary information source—a website—and an alternate source that was either an alternative website or a phone number. The primary information source for two of the tasks, the Carnegie Mellon University (CMU) online library catalog and an online banking application, were secured by SSL. We removed the certificate authorities verifying these websites from the trusted authorities list in each browser used in the study.[6] Therefore, participants were shown an invalid certificate warning when they navigated to the library and bank websites. We noted how users reacted to these warnings and whether they completed the task by continuing to use the website or by switching to

[6] Ideally we would have performed a man-in-the-middle attack, for example by using a web proxy to remove the websites' legitimate certificates before they reached the browser. However, due to legal concerns, we instead simulated a man-in-the-middle attack by removing the root certificates from the web browser.

	Tech score		Expired	Unknown CA		Domain Mismatch	
FF2	$\mu = 0.61$	*Experts*	69%	44%		31%	
	$\sigma = 1.14$	*Non-Experts*	63%	48%		31%	
FF3	$\mu = 0.99$	*Experts*	52%	13%	$\chi^2_2 = 12.37$	10%	$\chi^2_2 = 11.42$
	$\sigma = 1.42$	*Non-Experts*	47%	41%	$p < 0.002$	31%	$p < 0.003$
IE7	$\mu = 0.47$	*Experts*	42%	33%		29%	
	$\sigma = 1.02$	*Non-Experts*	36%	31%		29%	

Table 4: Percentage of experts and non-experts who said they would continue past the warnings. The first column shows respondents' average tech scores.

the alternative information source. Finally, we gave users an exit survey to gauge their understanding of and reaction to the warnings.

4.1.1 Recruitment

We recruited participants by posting our study on the experiment list of the Center for Behavioral Research at CMU. We also hung posters around the CMU campus. Participants were paid $10–20 for their participation.[7] All recruits were given an online screening survey, and only online banking customers of our chosen bank were allowed to participate. The survey included a range of demographic questions and questions about general Internet use.

In total, 261 users completed our screening survey and 100 users qualified and showed up to participate in our study. We randomly assigned 20 users to each condition. Half the users in each condition were given the bank task first and half were given the library task first. Participants took 15–35 minutes to complete the study including the exit survey.

We tried to ensure that participants were not primed to think about security. The study was presented not as a security study, but as a "usability of information sources study." Our recruitment postings solicited people who were "CMU faculty staff or students" and had "used online banking in the last year." However, we also required that participants have "purchased an item online in the last year" and "used a search engine" to avoid focusing potential participants on the banking tasks. Finally, our screening survey asked a series of questions whose responses were not used to screen participants (e.g. "How often do you use Amazon.com?"), to further obfuscate the study purpose.

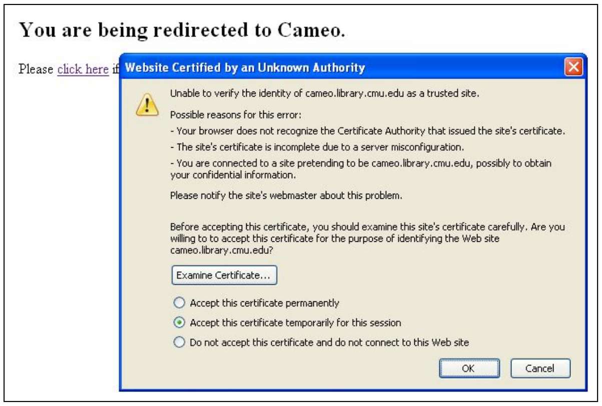
[7]Initially participants were paid $10, but we raised the payment to $20 to reach our recruiting goals.

4.1.2 Conditions

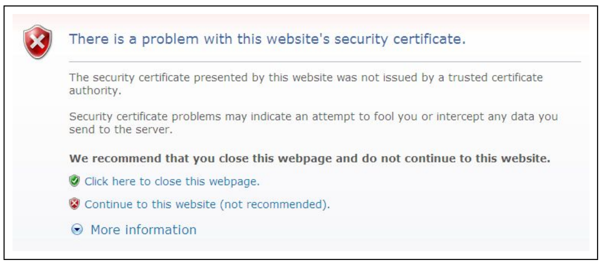
The FF2 warning, displayed in Figure 2(a), is typical of invalid certificate warnings prior to 2006. This warning has a number of design flaws. The text contains jargon such as, "the website's certificate is incomplete due to a server misconfiguration." The look and feel of the warning, a grey dialog box with a set of radio buttons, is similar to a lot of other trivial dialogs that users typically ignore, such as "you are sending information unencrypted over the internet." The default selection is to accept the certificate temporarily. This is an unsafe default for many websites, including the online banking application in our study.

A more subtle problem with the FF2 warning, and those like it, is that it asks users a question that they cannot answer. The warning asks the user to determine if the certificate problem is the result of a server/browser configuration problem or a legitimate security concern. Since users are not capable of making this determination, the dialog is, in the words of Firefox project co-founder Blake Ross, "a dilemma to users." Ross calls on browser designers to do everything possible to make decisions for their users. When designers have to ask questions of their users, they should ask questions that users can answer [16].

The FF3 warning should be more noticeable to users than its predecessor because it takes over the entire page and forces users to make a decision. Additionally, it takes four steps to navigate past the warning to the page with the invalid certificate. First



(a) Firefox 2



(b) Internet Explorer 7

Figure 2: Screenshots of the FF2 and IE7 warnings.

the user has to click a link, mysteriously labeled "or you can add an exception..." (Figure 3), then click a button that opens a dialog requiring two more button clicks. The first version of the FF3 warning required 11 steps.[8] This clearly represented a decision by Firefox developers that all invalid certificates are unsafe. They made the original version of the warning so difficult for users to override, that only an expert would be likely to figure out how to do it. While FF3 was in alpha and beta testing, many users erroneously believed the browser was in error when they could not visit websites that they believed to be legitimate.[9]

The IE7 warning, shown in Figure 2(b), occupies the middle ground between the FF2 and FF3 warnings. It takes over the entire page and has no default option, but differs from the FF3 warning because it

[8] https://bugzilla.mozilla.org/show_bug.cgi?id=399275
[9] https://bugzilla.mozilla.org/show_bug.cgi?id=398915

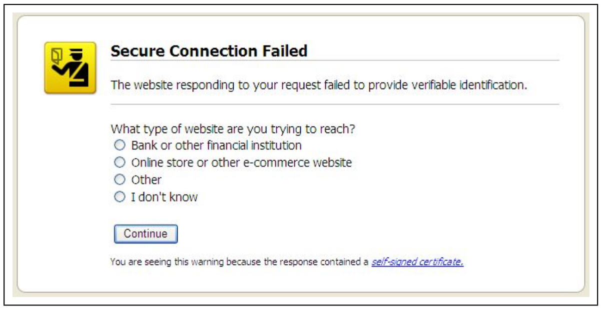
Figure 3: Screenshot of the initial FF3 warning.

can be overridden with a single click on a link labeled "Continue to this website." It has a slightly scarier look and feel than the FF2 warning: the background color has a red tint and a large X in a red shield dominates the page. The warning also explicitly recommends against continuing. Finally, when viewing this warning the background of the address bar is red and continues to be red after one overrides the warning.

We designed two warnings using techniques from the warning literature and guided by results from our survey. Our multi-page warning first asks the user a question, displayed in Figure 4(a), and then, depending on the response, delivers the user either to the severe warning page shown in Figure 4(b) or to the requested website. The second version of the warning shows only the severe warning (Figure 4(b)). Both versions were implemented in IE7. We used the resourcemodify tool[10] to replace the HTML file of the native warning in an IE DLL with our HTML files.

The second version of our warning serves two purposes. First, it attempts to see how users react to a simple, clear, but scary warning. The warning borrows its look and feel from the FF3 phishing warning. It is red and contains the most severe version of Larry the Firefox "passport officer."[11] The title of the page is clear and harsh: "High Risk of Security Compromise." The other context is similarly blunt (e.g. "an attacker is attempting to steal information that you are sending to *domain name.*"). Even the

[10] http://deletethis.net/dave/xml-source-view/httperror.html
[11] http://news.cnet.com/8301-10789_3-9970606-57.html



(a) Page 1

(b) Page 2

Figure 4: Screenshot of redesigned warning.

default button, labeled "Get me out of here!" signifies danger. The only way for a user to continue is to click the tiny link labeled "Ignore this warning" in the bottom right corner. The second purpose of the single page warning is to help us interpret the results from our multi-page warning. We compare the multi-page results to the single-page results to see how the question affects user actions independent of the the scary second page.

The original FF3 warning aimed to avoid asking users questions, and instead decided on users' behalf that invalid certificates are unsafe. However, even the Firefox designers eventually realized this could not work in the real world because too many legitimate websites use invalid certificates. Instead, our warning aims to ask the users a question that they can answer and will allow us to assess the risk level. Our question is, "What type of website are you trying to reach?" Users were required to select from one of four responses: "bank or other financial institution," "online store or other e-commerce website," "other," and "I don't know." If users selected the first two options, they saw the severe warning that discouraged them from continuing. We tested this question as a prototype for leveraging user-provided information to improve security warnings. It is not a complete solution as our question neglects many other types of websites that may collect sensitive information. We decided to show the secondary warning on bank websites and online stores because these are the most frequently attacked websites [15].

4.1.3 Experimental Setup

All studies were conducted in our laboratory on the same model of laptop. Participants interacted with the laptop within a virtual machine (VM). We reset the VM to a snapshot after each participant finished the study to destroy any sensitive data entered by the participant (e.g. bank password). This process also ensured that all browser and operating system settings were exactly the same for every participant. Finally, experimenters read instructions to participants from a script and experimenters did not help particiants complete the tasks.

4.1.4 Tasks

After participants signed IRB consent forms, the experimenter handed them an instruction sheet and read this sheet aloud. Participants were reminded that they would be "visiting real websites and calling real organizations" and therefore should go about "each task in the way you would if you were completing it with the computer you usually use." Participants were also instructed to "think aloud and tell us what you are thinking and doing as you complete each task," in order to give us qualitative reactions to the warnings. The experimenter took notes throughout the study. The study was recorded (audio only), which allowed experimenters to retrieve details that were missed during note taking.

After the instructions were read and digested, the instruction sheets for each task were handed to the participant and read aloud by the experimenter one by one. The next task was not revealed until all previous tasks had been completed. The first task asked participants to find the total area of Italy in square kilometers using Google or Ask.com as an alternative. The second task was to look up the last two digits of the participant's bank account balance using the online banking application or using phone banking. The third task was to locate the price of the hardcover edition of the book *Freakonomics* using Amazon.com or the Barnes and Noble website. Finally, the fourth task was to use the CMU online library catalog or alternatively the library phone number to retrieve the call number of the book *Richistan* (i.e. no personal information was transmitted).

The first and third tasks were "dummy tasks," since the bookstore and search engine revealed no warnings. Instead, they reinforced to participants that the goal of the study was information sources, not security. Half the participants in each condition had the second and fourth tasks—the warning tasks—swapped so that we could control for the ordering of the warnings.

Researchers have found that study participants are highly motivated to complete assigned tasks. Participants want to please the experimenter and do not want to "fail" so they sometimes exert extreme effort to complete the task [12]. A closely related study [17] was criticized for not taking into account this "task focus" phenomenon [14]. Critics worried that participants were ignoring the warnings in the study because of task focus and not because this is what they would do in a more natural environment.

Our study design mitigates participants' task focus by presenting an alternate method for each task so that participants could "pass the test" without ignoring the warnings. We instructed participants to "try the suggested information source first," to ensure that participants would only call the library or bank as a reaction to the warning. As there were no obstacles to completing the dummy tasks using the suggested information source, none of the participants used the alternate method to perform the dummy tasks.

4.1.5 Exit Survey

After completing all four study tasks, participants were directed to an online exit survey hosted by SurveyMonkey. The exit survey asked 45 questions in six categories. The first set of questions asked about their understanding of and reaction to the bank warning in the study. The second question asked the same questions about the library warning. The third set asked questions to gauge their general understanding of certificates and invalid certificate warnings. The fourth set gauged participants' prior exposure to identity theft and other cyberthreats. The fifth set, which were also asked in the online SSL survey, asked them about their technical experience, including their experience with computer security. Finally, the sixth set asked general demographic questions like age, gender and education level.

4.2 Results and Analysis

The primary goal of any SSL warning should be to prevent users from transmitting sensitive information to suspicious websites. A secondary—but still important—goal is to allow users to continue in the event of a false positive (i.e. when a certificate error is unlikely to result in a security compromise). In our study we examined these goals by observing whether participants discontinued visiting the bank website while continuing to the library website. These results from our laboratory experiment are displayed in Table 5. Participants who saw our single-page or multi-page warnings were more likely to heed the warnings than participants who saw the FF2 or IE7 warnings, but not the FF3 warning. In contrast, participants who saw our multi-page warning were more likely to visit the library website than participants who saw the FF3 warning. In the rest of this section we discuss demographics, present more detailed comparisons of the conditions and tasks, and present interesting qualitative results from our exit survey.

4.2.1 Participant Characteristics

We did not find any statistically significant demographic imbalances between participants in our randomly assigned conditions. The factors we tested were gender, nationality, age, technical sophistication, and a metric we call "cyberthreat exposure" designed to measure participants' prior experiences

	FF2		FF3		IE7		Single-Page		Multi-Page	
Bank	18	(90%)	11	(55%)	18	(90%)	9	(45%)	12	(60%)
Library	19	(95%)	12	(60%)	20	(100%)	16	(80%)	19	(95%)

Table 5: Number (and percentage) of participants in each condition who ignored the warning and used the website to complete the library and bank tasks.

with information theft and fraud. Most demographic factors were determined by a single exit survey question (e.g. gender, nationality). Technical sophistication was measured by a composite score of five question, the same as in the online survey. Similarly, cyberthreat exposure was measured by asking participants if they have ever had any account information stolen, found fraudulent transactions on bank statements, had a social security number stolen, or if they had ever been notified that personal information had been stolen or compromised.

Our participants were technically sophisticated, mostly male, and mostly foreign students. We had 68 male and only 32 female participants. All of our participants were between the ages of 18–30, and all but two were students. Sixty-nine participants were born in India, 17 in the United States, and the remaining were from Asia (10) and Europe (4). The average tech score was 1.90, which is significantly larger than the 0.66 average among the survey respondents.

We do not have a large enough sample size to determine whether age, profession, or nationality influenced participant behavior. In addition, our participants had so little cyberthreat exposure—83 participants answered affirmatively to 0 out of 4 questions—that we could not determine if exposure correlated with our results. On the other hand, while our sample was large enough to observe behavioral differences based on gender and technical sophistication if large differences existed, we observed no statistical differences in participant behavior based on those factors. Finally, we found no statistical difference in behavior based on task order in any of the conditions.

4.2.2 Effect of Warning Design on Behavior

Our study focused on evaluating whether SSL warnings effectively prevent users from transmitting sensitive information to suspicious websites, while allowing them to continue in the event of a false positive.

We hypothesized that participants visiting the bank website who see our redesigned warnings would be significantly more likely to discontinue than participants who see the other warnings. We used a one-tailed Fisher's exact test to analyze our results. We found that significantly more participants obeyed our single page warning than obeyed the FF2 and IE7 warnings ($p < 0.0029$ for both comparisons). Similarly, our multi-page warning performed better than the FF2 and IE7 warnings ($p < 0.0324$). However, FF3 was equivalently preventative, and it was also significantly better than the FF2 and IE7 warnings ($p < 0.0155$).

We also hypothesized that participants visiting the library website who see our redesigned warning will be significantly more likely to continue than participants who see the other warnings. In this case our hypothesis turned out to be mostly false. Participants who viewed our multi-page warning were significantly more likely to use the library website than participants who saw the FF3 warning ($p < 0.0098$). However, users of our multi-page warning visited the library website at an equal rate to users of the FF2 and IE7 warnings. Our single page warning was not significantly different than any of the other warnings. The FF3 warning caused significantly more participants to call the library than the FF2 warning ($p < 0.0098$) or the IE7 warning ($p < 0.0016$).

Two participants in the FF3 condition and one in our multi-page warning condition thought the library and bank servers were down or that we had blocked their websites. One wrote in the exit survey "the graphics made me feel the server was down" and another wrote "I just saw the title and assumed that it is just not working on this computer." We suspect that users confuse the warnings with a 404 or server not found error, like the one shown in Figure 5. The

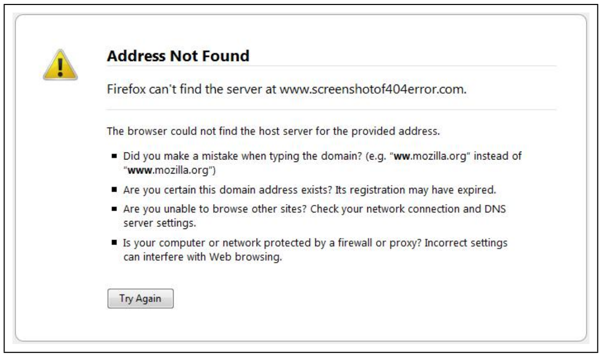


Figure 5: Screenshot of server not found error in FF3.

warnings have very similar layouts and coloring. The yellow Larry icon in the FF3 warning (Figure 3) and the first page of our multi-page (Figure 4(a)) warning is similar to the yellow triangle in Figure 5.

We took careful note of how participants in the multi-page warning condition answered the question "What type of website are you trying to visit?" presented to them on the first page of the warning. Fifteen participants answered exactly as expected – they selected "other" for the library and "bank or other financial institution" for the bank. The remaining five participants exhibited noteworthy behaviors: one participant did not answer the question for either task, while three participants performed the library task first and appropriately answered "other," but also inaccurately answered "other" when visiting the bank website. This is stark evidence of the ill-effects of warning habituation – these participants learned how to ignore the warning in the library task and immediately reapplied their knowledge to the bank task. Finally, one participant first performed the bank task and correctly answered "bank or other financial institution." However, when she saw the second page of the warning she clicked the back button and changed her answer to "other."

4.2.3 Risk Perception in Context

We hypothesized that participants who viewed our multi-page warning would be more likely to obey the warnings when they were visiting the bank website than when they were visiting the library website. Because this warning took context into account in determining severity, it appeared to be more severe on the bank website. All 14 participants in our study who heeded the library warning also heeded the warning at the bank. An additional 18 participants heeded the bank warning and proceeded past the library warning. Participants who viewed our multi-page warning ($p < 0.0098$) and our single-page warning ($p < 0.0242$) were significantly more likely to heed the warning at the bank than at the library.

We believe the behavior exhibited by users of our single page warning can be explained both by its success in raising awareness of risk and its clear communication of what users should do in response to the risk. When the 11 participants who heeded the single-page bank warning were asked in the exit survey "Why did you choose to heed or ignore the warning?" 9 out of 11 specifically mentioned the security of their information as the reason. In contrast only 2 participants in each of the FF2, FF3, and IE7 conditions mentioned risk in response to the same question. In addition, 10 of the 20 participants in our single-page warning condition when asked, "What action(s) did you think the warning at the bank wanted you to take?" responded that it wanted them *not* to proceed. Only 3 FF2, 2 FF3, and 4 IE7 participants answered the same way.

4.2.4 Impact of Reading and Understanding

In each of the first two sections of the exit survey we asked participants if they "read the text of the warning at the *bank/library* website." At the bank website, significantly more people read our multi-page warning than the FF2 ($p < 0.0128$), FF3 ($p < 0.0018$), or IE7 ($p < 0.0052$) warnings (Table 6). There were no other significant differences in reported readership across conditions or tasks. We used a chi-square test to see if there was a difference in how reading affected behavior. Among the participants who did not read the warnings, FF2 and IE7 users were significantly more likely to log in to the bank website ($\chi^2_4 = 13.56$, $p < 0.009$), whereas FF3 users were significantly less likely to log in to the library website ($\chi^2_4 = 18.38$, $p < 0.001$).

The exit survey asked participants "what did

Condition	Read		Didn't Read		Understood		Didn't Understand	
	Logged In	Called	Logged In	Called	Logged In	Called	Logged In	Called
FF2	4	2	14	0	7	2	11	0
FF3	2	2	9	7	4	2	7	7
IE7	4	1	14	1	8	2	10	0
Single-Page	4	6	5	5	4	7	5	4
Multi-Page	8	6	4	2	7	6	5	2

Table 6: Behavior in the bank task by reading, understanding, and condition.

you believe the warning at the *bank/library* website meant?" Answers were entered into a free response text box and we categorized the responses according to whether or not they demonstrated understanding of the warning, as we had done in the survey (Table 6). In particular, participants who wrote that their connection may be compromised or that the identity of the destination website could not be verified were deemed to understand the warning. All other responses were coded as not understanding the meaning. There were no significant differences in the number of participants who understood the warnings based on condition in either task. However, participants in the FF3 condition who did not understand the warning were significantly more likely to call than users in the FF2 ($p < 0.0078$) and IE7 ($p < 0.0188$) conditions. Seven of the 14 participants who did not understand the FF3 warning called the bank. This is evidence that the FF3 users may have been prevented from visiting the websites because they did not know how to override warnings, and not because they understood the risks of proceeding.

One expects that participants who claimed to have read the warnings would be more likely to understand their meaning. When we combined the data from just our two warnings, single-page and multi-page, we found a statistically significant correlation ($p < 0.020$). However, we do not have enough data to determine whether there is a correlation for the three native warnings (FF2, FF3, and IE7).

4.2.5 Other Observations

One worry for browser designers trying to design effective warnings is that they will cause users to switch browsers, in favor of a browser that shows a less severe warning. In fact, during our study a few participants who viewed our warnings or the FF3 warnings asked or attempted to perform one of the tasks in a different browser. We directed them to continue using the browser they had been using. No participants in the FF2 and IE7 conditions tried to switch browsers. This indicates that complex warning designs may cause a small number of users to switch browsers. Therefore, for the sake of these users' security, it may be best if all browsers converged on a single warning design.

Response	FF2	FF3	IE7	Single	Multi
Yes	8	7	10	4	1
No	8	11	5	16	16
Unknown	4	2	5	0	3

Table 7: Number of participants in each condition who claimed to have seen the warning before at the bank.

Among our strangest results were the answers to the questions: "Before this study, had you ever seen the warning you saw at the *bank/library* web site?" (Table 7). A total of 30 participants said they had seen the warning before at the bank website compared to only 16 at the library website. In addition, 5 participants in the bank task thought they had seen our warnings before. We do not think 30% of our participants have been scammed by man-in-the-middle attacks at their bank and we know for sure that the 5 participants had never seen our redesigned warnings before. This is dramatic evidence of memory problems, warning confusion, and general confusion with regard to certificate errors. At the same time, it is possible that the novelty of our new warnings contributed to more participants reading them (and consequently better understanding the risks of ignoring them). None of the participants who viewed our new warnings could have seen them before, while our randomized condition assignments resulted in the two Firefox conditions being assigned 27 participants who were pre-existing Firefox users (68% of 40) and the IE condition being assigned 6 participants who were existing IE users (30% of 20). Thus, it is likely that these 33 participants had already been exposed to the warnings prior to our study, but among our sample population we observed no significant differences in behavior among them and the participants in the IE and FF conditions who were accustomed to using different browsers.

In the exit survey we asked participants to use a 7-point Likert scale to report the influence of several factors on their decision to heed or ignore the warnings. The factors we included were: the text of the warning, the colors of the warning, the choices that the warning presented, the destination URL, and the look and feel of the destination website. We expected significantly more participants to grade the color and text of the website highly for our warnings. However, there was no statistically significant difference in participants' responses based on condition.

5 Discussion

Our warnings somewhat improved user behavior, but all warning strategies, including ours, leave too many users vulnerable to man-in-the-middle attacks. The five warnings we evaluated embodied three different strategies: explain the potential danger facing users, make it difficult for users to ignore, and ask a question users can answer. The strategies have differences that we will discuss later in this section. However, regardless of how compelling or difficult to ignore, users think SSL warnings are of little consequence because they see them at legitimate websites. Many users have a completely backward understanding of the risk of man-in-the-middle attacks and assume that they are *less* likely to occur at trusted websites like those belonging to banks. If they do become fraud victims, they are unlikely to pinpoint it to their decision to ignore a warning. Thus users' attitudes and beliefs about SSL warnings are likely to undermine their effectiveness [3]. Therefore, the best avenue we have for keeping users safe may be to avoid SSL warnings altogether and *really* make decisions for users—blocking them from unsafe situations and remaining silent in safe situations.

5.1 Limitations

We did not attempt to measure any long term affects of habituation to warnings. Many participants were likely to have previously seen the FF2 and IE7 warnings, while few users were likely to have seen FF3 warnings as that browser was released just before the study began. Our two warnings were new to all participants. We expect users were more likely to ignore the IE7 and FF2 warnings because of habituation, but this is not supported by our data.

Several artifacts of the study design may have caused participants to behave less securely than they normally would. Our study participants knew in advance that they would be using their bank credentials during the study and therefore the most security conscious potential participants may have decided not to perform the study. In addition, the study was performed at and sanctioned by Carnegie Mellon, and therefore participants may have trusted that the study would not put their credentials at risk.

In our study, users were much less likely to heed certificate warnings than in a previous study by Schechter et al. that also examined user responses to the IE7 certificate warning [17]. In our study 90% of participants ignored the IE7 warning while in the Schechter et al. study only 36% of participants who used their own accounts ignored the IE7 warning. We believe the differences may be due to the fact that in the previous study participants were told the study was about online banking, they performed four banking tasks prior to observing the warning, and they were given two other clues that the website might be insecure prior to the display of the warnings. The authors state, "responses to these clues may have been influenced by the presence of prior clues." Furthermore, the previous study was conducted while IE7 was still in beta and thus users were less likely to have seen the certificate warning before. In addition, our study participants were more technically sophisticated than the previous study's participants.

5.2 Explain the Danger

The FF2, IE7, and our single page warnings take the standard tactic of explaining the potential danger to users. The FF2 warning, which is an unalarming popup box with obscure language, prevented very few users from visiting the bank or library. The IE7 warning, which has clearer language and a more frightening overall look, does not perform any better. On the other hand, our single page warning, with its black and red colors, was the most effective of the five warnings at preventing users from visiting the bank website. In addition, only four users called the library, indicating that our single-page warning would be only a minor nuisance for legitimate websites. That said, we suspect our single page warning would become less effective as users are habituated to it when visiting legitimate websites.

5.3 Make it Difficult

The FF3 warning, as discussed at length in Section 4.2.2, prevents user from visiting websites with invalid certificates by confusing users and making it difficult for them to ignore the warning. This improves user behavior in risky situations like the bank task, but it presents a significant nuisance in safer situations like the library task. Many legitimate websites that use self-signed certificates have posted online tutorials teaching users how to override the FF3 warning.[12] We suspect that users who learn to use the warning from these tutorials, by simple trial and error, help from a friend, etc., will ignore subsequent warnings and will be left both annoyed and unprotected.

[12]See for example: 1) http://hasylab.desy.de/infrastructure/experiment_control/links_and_tutorials/ff3_and_ssl/index_eng.html, 2) http://www.engr.colostate.edu/webmail/, and 3) http://knowledgehub.zeus.com/faqs/2008/02/05/configuring_zxtm_with_firefox_3

5.4 Ask a Question

Our multi-page warning, introduced in Section 4.1.2, asks the user a question in order to collect contextual information to allow the browser to better assess the risk of letting the user proceed to the website. This warning suffers from two usability problems: users may answer incorrectly because they are confused and users may knowingly answer incorrectly to get around the warning. In addition, it leaves users susceptible to active attacks such as the finer-grained origins attacks [9]. These problems, plus the fact that the single-page warning was more successful in preventing users from visiting the bank website, lead us to recommend against our multi-page warning as it is currently implemented.

The multi-page warning depends on users correctly answering our question, but only fifteen of the 20 participants answered correctly at the bank website. As discussed in Section 4.2.2, we believe that five participants either knowingly gave the wrong answer in order to reach the destination website without interruption, or they confused the warning with a server unavailable error. However, many users still made mistakes even when answering our question correctly. They behaved no more securely than users of our single-page warning.

Users who answered our question correctly and followed its advice would still be susceptible to finer-grained origins attacks. As brought to our attention by an anonymous reviewer, an attacker with control over the network or DNS may circumvent the multi-page warning by forcing the browser to connect to a website other than the one the user intended. For example, let's say Alice goes to a webmail site (www.mail.com), but an attacker controls the network and wants to steal the password to her online bank (www.bank.com).

When Alice visits mail.com, the attacker sends a response to the Alice that forwards the browser to https://www.bank.com/action.js. Then, the attacker intercepts the connection to the bank with a self-signed certificate, which triggers the warning shown in Figure 4(a). The warning asks her what type of website she is trying to reach and she answers "other" because she believes she is visiting her webmail. Since

Alice answered “other” she is immediately forwarded to action.js. If Alice has an open session with the bank, the attacker steals her bank.com secure cookies with the script.

Even if Alice does not have an open session with the bank, the browser’s cache will store the attack script. Let’s say in its normal operation the bank site loads its version of action.js after a user logs-in. (If the site loads a different script, then the attacker simply poisons that script instead.) If Alice logs-into www.bank.com in the next year, then the attacker’s version of action.js will load instead of the bank’s version. As in the attack in the previous paragraph, the script steals her secure cookies. There are many other variations on this attack, but they all rely on Alice answering “what type of website are you trying to visit” based on the site she believes she is visiting instead of the site the attacker sends to her.

Designing an interface to collect contextual information from users without making them susceptible to active attacks such as those outlined above poses a challenge. While we can ask users simple questions about their intentions that they are capable of answering, we must be sure that attackers cannot intervene to mislead users. We may be able to improve the multi-page warning we proposed by asking users another question in certain circumstances. In particular, if the URL of the connecting website is substantially different than the URL the user typed (or clicked on, in the case of a link), then we would show the URL of the connecting website and ask the user if they intended to visit that URL. Unfortunately this is not a complete solution for websites with mixed content, like those using a third-party shopping cart provider. In addition, the usability of such a solution remains untested.

It remains an open research challenge to determine how to leverage contextual information—including user-provided information—in order to assess risks. In particular, an approach is needed that is not vulnerable to confused users, users trying to get around the system, or active attackers. It remains to be seen whether it is feasible to design a robust approach that uses user-provided information. Alternative approaches may leverage contextual information provided by sources other than the user.

5.5 Avoid Warnings

The ideal solution to SSL warning problems is to block access when users are in true danger and allow users to proceed when they are not. This ideal is probably unattainable, but two systems recently presented by the research community, ForceHTTPS [10] and Perspectives [20] (and discussed in Section 2), are steps in the right direction. Both systems identify websites likely to be unsafe and use warnings to stop users from proceeding. It would be better to block these unsafe websites entirely. We expect both systems to have extremely low false positive rates, but further evaluation is required to know for sure. Another possible way of identifying unsafe websites is to maintain a list of websites that are verified by a root certificate authority and block websites on the list when the browser receives a self-signed certificate instead.

6 Acknowledgements

Thanks to Dhruv Mohindra, Amit Bhan, and Stuart Schechter for their help in the early stages of this project. This work was supported in part by Microsoft Research and by the National Science Foundation under Grants No. 0524189 and 0831428. The first author is supported by a National Defense Science and Engineering Graduate Fellowship.

References

[1] J. C. Brustoloni and R. Villamarín-Salomón. Improving security decisions with polymorphic and audited dialogs. In *Proceedings of the 3rd symposium on Usable privacy and security*, pages 76–85, New York, NY, USA, 2007. ACM Press.

[2] Certification Authority/Browser Forum. Extended validation SSL certificates, Accessed: July 27, 2007. http://cabforum.org/.

[3] L. F. Cranor. A framework for reasoning about the human in the loop. In *Proceedings of the 1st Conference on Usability, Psychology, and Security*, pages 1–15, Berkeley, CA, USA, 2008. USENIX Association.

[4] R. Dhamija, J. D. Tygar, and M. Hearst. Why phishing works. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 581–590, New York, NY, USA, 2006. ACM.

[5] I. E-Soft. SSL server survey, February 1, 2007. http://www.securityspace.com/s_survey/sdata/200701/certca.html.

[6] S. Egelman, L. F. Cranor, and J. Hong. You've been warned: an empirical study of the effectiveness of web browser phishing warnings. In *Proceeding of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1065–1074, New York, NY, USA, 2008. ACM.

[7] B. Fogg, J. Marshall, O. Laraki, A. Osipovich, C. Varma, N. Fang, J. Paul, A. Rangekar, J. Shon, P. Swani, and M. Treinen. What makes web sites credible? a report on a large quantitative study. In *Proceedings of the SIGCHI Conference on in Computing Systems*, Seattle, WA, March 31 - April 4, 2001. ACM.

[8] B. Friedman, D. Hurley, D. C. Howe, E. Felten, and H. Nissenbaum. Users' conceptions of web security: a comparative study. In *Extended Abstracts on Human Factors in Computing Systems*, pages 746–747, New York, NY, USA, 2002. ACM.

[9] C. Jackson and A. Barth. Beware of finer-grained origins. In *Proceedings of the Web 2.0 Security and Privacy Workshop*, 2008.

[10] C. Jackson and A. Barth. ForceHTTPS: protecting high-security web sites from network attacks. In *Proceeding of the 17th International World Wide Web Conference*, pages 525–534, New York, NY, USA, 2008. ACM.

[11] C. Jackson, D. R. Simon, D. S. Tan, and A. Barth. An evaluation of extended validation and picture-in-picture phishing attacks. In *Proceeding of the 1st International Workshop on Usable Security*, pages 281–293, Berlin / Heidelberg, Germany, February 2007. Springer.

[12] S. Milgram. *Obedience to Authority: An Experimental View*. Harpercollins, 1974.

[13] J. Nightingale. SSL information wants to be free, January 2009. http://blog.johnath.com/2009/01/21/ssl-information-wants-to-be-free/.

[14] A. Patrick. Commentary on research on new security indicators. Self-published Online Essay, Accessed: January 15, 2009. http://www.andrewpatrick.ca/essays/commentary-on-research-on-new-security-indicators/.

[15] R. Rasmussen and G. Aaron. Global phishing survey: Domain name use and trends 1h2008. Anti-Phishing Working Group Advisory, November 2008. http://www.antiphishing.org/reports/APWG_GlobalPhishingSurvey1H2008.pdf.

[16] B. Ross. Firefox and the worry free web. In L. F. Cranor and S. Garfinkel, editors, *Security and Usability: Designing Secure Systems that People Can Use*, pages 577–588. O'Reilly Media, Inc., Sebastopol, CA, USA, August 2005.

[17] S. E. Schechter, R. Dhamija, A. Ozment, and I. Fischer. The emperor's new security indicators. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, pages 51–65, Washington, DC, USA, 2007. IEEE Computer Society.

[18] J. Sobey, R. Biddle, P. C. van Oorschot, and A. S. Patrick. Exploring user reactions to new browser cues for extended validation certificates. In *Proceedings of the 13th European Symposium on Research in Computer Security*, pages 411–427, 2008.

[19] D. W. Stewart and I. M. Martin. Intended and unintended consequences of warning messages: A review and synthesis of empirical research. *Journal of Public Policy & Marketing*, 13(1):1–1, 1994.

[20] D. Wendlandt, D. G. Andersen, and A. Perrig. Perspectives: Improving SSH-style host authentication with multi-path probing. In *Proceedings of the 2008 USENIX Annual Technical Conference*, Berkeley, CA, USA, June 2008. USENIX Association.

[21] T. Whalen and K. M. Inkpen. Gathering Evidence: Use of Visual Security Cues in Web Browsers. In *Proceedings of the 2005 Conference on Graphics Interface*, pages 137–144, Victoria, British Columbia, 2005.

[22] M. Wogalter. Purpose and scope of warnings. In M. Wogalter, editor, *Handbook of Warnings*, pages 3–9. Lawrence Erlbaum Associates, Mahway, NJ, USA, 2006.

[23] M. Wu, R. C. Miller, and S. L. Garfinkel. Do security toolbars actually prevent phishing attacks? In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 601–610, New York, NY, USA, 2006. ACM.

[24] H. Xia and J. C. Brustoloni. Hardening web browsers against man-in-the-middle and eavesdropping attacks. In *Proceedings of the 14th International World Wide Web Conference*, pages 489–498, New York, NY, USA, 2005. ACM.

The Multi-Principal OS Construction of the Gazelle Web Browser

Helen J. Wang*, Chris Grier†, Alexander Moshchuk‡, Samuel T. King†, Piali Choudhury*, Herman Venter*
*Microsoft Research †University of Illinois at Urbana-Champaign ‡University of Washington
{helenw,pialic,hermanv}@microsoft.com, {grier,kingst}@uiuc.edu, anm@cs.washington.edu

Abstract

Original web browsers were applications designed to view static web content. As web sites evolved into dynamic web applications that compose content from multiple web sites, browsers have become multi-principal operating environments with resources shared among mutually distrusting web site *principals*. Nevertheless, no existing browsers, including new architectures like IE 8, Google Chrome, and OP, have a multi-principal operating system construction that gives a browser-based OS the *exclusive* control to manage the protection of all system resources among web site principals.

In this paper, we introduce Gazelle, a secure web browser constructed as a multi-principal OS. Gazelle's browser kernel is an operating system that *exclusively* manages resource protection and sharing across web site principals. This construction exposes intricate design issues that no previous work has identified, such as cross-protection-domain display and events protection. We elaborate on these issues and provide comprehensive solutions.

Our prototype implementation and evaluation experience indicates that it is realistic to turn an existing browser into a multi-principal OS that yields significantly stronger security and robustness with acceptable performance.

1 Introduction

Web browsers have evolved into a multi-principal operating environment where a principal is a web site [43]. Similar to a multi-principal OS, recent proposals [12, 13, 23, 43, 46] and browsers like IE 8 [34] and Firefox 3 [16] advocate and support programmer abstractions for protection (e.g., `<sandbox>` in addition to `<iframe>` [43]) and cross-principal communication (e.g., `PostMessage` [24, 43]). Nevertheless, no existing browsers, including new architectures like IE 8 [25], Google Chrome [37], and OP [21], have a multi-principal OS construction that gives a browser-based OS, typically called the browser kernel, the *exclusive* control to manage the protection and fair sharing of all system resources among browser principals.

In this paper, we present a multi-principal OS construction of a secure web browser, called Gazelle. Gazelle's browser kernel *exclusively* provides cross-principal protection and fair sharing of *all* system resources. In this paper, we focus only on resource protection in Gazelle.

In Gazelle, the browser kernel runs in a separate protection domain (an OS process in our implementation), interacts with the underlying OS directly, and exposes a set of system calls for web site principals. We use the same web site principal as defined in the same-origin policy (SOP), which is labeled by a web site's origin, the triple of `<protocol, domain name, port>`. In this paper, we use "principal" and "origin" interchangeably. Unlike previous browsers, Gazelle puts web site principals into separate protection domains, completely segregating their access to all resources. Principals can communicate with one another only through the browser kernel using inter-process communication. Unlike all existing browsers except OP, our browser kernel offers the same protection to plugin content as to standard web content.

Such a multi-principal OS construction for a browser brings significant security and reliability benefits to the overall browser system: the compromise or failure of a principal affects that principal alone, leaving other principals and the browser kernel unaffected.

Although our architecture may seem to be a straightforward application of multi-principal OS construction to the browser setting, it exposes intricate problems that did not surface in previous work, including display protection and resource allocation in the face of cross-principal web service composition common on today's web. We will detail our solutions to the former and leave the latter as future work.

We have built an Internet-Explorer-based prototype that demonstrates Gazelle's multi-principal OS architecture and at the same time uses all the backward-compatible parsing, DOM management, and JavaScript interpretation that already exist in IE. Our prototype experience indicates that it is feasible to turn an existing browser into a multi-principal OS while leveraging its existing capabilities.

With our prototype, we successfully browsed 19 out of the top 20 Alexa-reported popular sites [5] that we tested. The performance of our prototype is acceptable, and a significant portion of the overhead comes from IE instrumentation, which can be eliminated in a production implementation.

We expect that the Gazelle architecture can be made fully backward compatible with today's web. Nevertheless, it is interesting to investigate the compatibility cost of eliminating the insecure policies in today's browsers. We give such a discussion based on a preliminary analysis in Section 9.

For the rest of the paper, we first give an in-depth comparison with related browser architectures in Section 2. We then describe Gazelle's security model 3. In Section 4, we present our architecture, its design rationale, and how we treat the subtle issue of legacy protection for cross-origin script source. In Section 5, we elaborate on the problem statement and design for cross-principal, cross-process display protection. We give a security analysis including a vulnerability study in Section 6. We describe our implementation in Section 7. We measure the performance of our prototype in Section 8. We discuss the tradeoffs of compatibility vs. security for a few browser policies in Section 9. Finally, we conclude and address future work in Section 10.

2 Related Work

In this section, we discuss related browser architectures and compare them with Gazelle.

2.1 Google Chrome and IE 8

In concurrent work, Reis *et al.* detailed the various process models supported by Google Chrome [37]: monolithic process, process-per-browsing-instance, process-per-site-instance, and process-per-site. A browsing instance contains all interconnected (or inter-referenced) windows including tabs, frames and subframes *regardless* of their origin. A site instance is a group of same-site pages within a browsing instance. A site is defined as a set of SOP origins that share a registry-controlled domain name: for example, *attackerAd.socialnet.com*, *alice.profiles.socialnet.com*, and *socialnet.com* share the same registry-controlled domain name *socialnet.com*, and are considered to be the same site or principal by Chrome. Chrome uses the process-per-site-instance model by default. Furthermore, Reis *et al.* [37] gave the caveats that Chrome's current implementation does *not* support strict site isolation in the process-per-site-instance and process-per-site models: embedded principals, such as a nested `iframe` sourced at a different origin from the parent page, are placed in the same process as the parent page.

The monolithic and process-per-browsing-instance models in Chrome do not provide memory or other resource protection across multiple principals in a monolithic process or browser instance. The process-per-site model does not provide failure containment across site instances [37]. Chrome's process-per-site-instance model is the closest to Gazelle's two processes-per-principal-instance model, but with several crucial differences: (1) Chrome's principal is site (see above) while Gazelle's principal is the same as the SOP principal. (2) A web site principal and its embedded principals co-exist in the same process in Chrome, whereas Gazelle places them into separate protection domains. Pursuing this design led us to new research challenges including cross-principal display protection (Section 5). (3) Plugin content from different principals or sites share a plugin process in Chrome, but are placed into separate protection domains in Gazelle. (4) Chrome relies on its rendering processes to enforce the same-origin policy among the principals that co-exist in the same process. These differences indicate that in Chrome, cross-principal (or -site) protection takes place in its rendering processes and its plugin process, in addition to its browser kernel. In contrast, Gazelle's browser kernel functions as an OS, managing cross-principal protection on all resources, including display.

IE 8 [25] uses OS processes to isolate tabs from one another. This granularity is insufficient since a user may browse multiple mutually distrusting sites in a single tab, and a web page may contain an iframe with content from an untrusted site (e.g., ads).

Fundamentally, Chrome and IE 8 have different goals from that of Gazelle. Their use of multiple processes is for failure containment across the user's browsing sessions rather than for security. Their security goal is to protect the host machine from the browser and the web; this is achieved by process sandboxing [9]. Chrome and IE 8 achieved a good milestone in the evolution of the browser architecture design. Looking forward, as the world creates and migrates more data and functionality into the web and establishes the browser as a dominant application platform, it is critical for browser designers to think of browsers as operating systems and protect web site principals from one another in addition to the host machine. This is Gazelle's goal.

2.2 Experimental browsers

The OP web browser [21] uses processes to isolate browser components (i.e., HTML engine, JavaScript interpreter, rendering engine) as well as pages of the same origin. In OP, intimate interactions between browser components, such as JavaScript interpreter and HTML engine, must use IPC and go through its browser kernel. The additional IPC cost does not add much benefits: isolating browser components within an instance of a web page provides no additional security protection. Furthermore, besides plugins, basic browser components are fate-shared in web page rendering: the failure of any one browser component results in most web pages not functioning properly. Therefore, process isolation across these components does not provide any failure containment benefits either. Lastly, OP's browser kernel does not provide all the cross-principal protection needed as an OS because it delegates display protection to its processes.

Tahoma [11] uses virtual machines to completely isolate (its own definition of) web applications, disallowing any communications between the VMs. A web application is specified in a manifest file provided to the virtual machine manager and typically contains a suite of web sites of possibly different domains. Consequently, Tahoma doesn't provide protection to existing browser principals. In contrast, Gazelle's browser kernel protects browser principals first hand.

The Building a Secure Web Browser project [27, 28] uses SubOS processes to isolate content downloading, display, and browser instances. SubOS processes are similar to Unix processes except that instead of a user ID, each process has a SubOS ID with OS support for isolation between objects with different SubOS IDs. SubOS instantiates a browser instance with a different SubOS process ID for each URL. This means that the principal in SubOS is labelled with the URL of a page (protocol, host name plus path) rather than the SOP origin as in Gazelle. Nevertheless, SubOS does not handle embedded principals, unlike Gazelle. Therefore, they also do not encounter the cross-principal display-sharing issue which we tackle in depth. SubOS's principal model would also require all cross-page interactions that are common within a SOP origin to go through IPC, incurring significant performance cost for many web sites.

3 Security model

3.1 Background: security model in existing browsers

Today's browsers have inconsistent access and protection model for various resources. These inconsistencies present significant hurdles for web programmers to build robust web services. In this section, we give a brief background on the relevant security policies in existing browsers. Michal Zalewski gives an excellent and perhaps the most complete description of existing browsers' security model to date [48].

Script. The same-origin policy (SOP) [39] is the central security policy on today's browsers. SOP governs how scripts access the HTML document tree and remote store. SOP defines the *origin* as the triple of `<protocol, domain-name, port>`. SOP mandates that two documents from different origins cannot access each other's HTML documents using the Document Object Model (DOM), which is the platform- and language-neutral interface that allows scripts to dynamically access and update the content, structure and style of a document [14]. A script can access its document origin's remote data store using the `XMLHttpRequest` object, which issues an asynchronous HTTP request to the remote server [45]. (XMLHttpRequest is the cornerstone of AJAX programming.) SOP allows a script to issue an XMLHttpRequest only to its enclosing page's origin. A script executes as the principal of its enclosing page though its source code is not readable in a cross-origin fashion.

For example, an `<iframe>` with source *http://a.com* cannot access any HTML DOM elements from another `<iframe>` with source *http://b.com* and vice versa. *http://a.com*'s scripts (regardless of where the scripts are hosted) can issue XMLHttpRequests to only *a.com*. Furthermore, *http://a.com* and *https://a.com* are different origins because of the protocol difference.

Cookies. For cookie access, by default, the principal is the host name and path, but without the protocol [19, 32]. For example, if the page *a.com/dir/1.html* creates a cookie, then that cookie is accessible to *a.com/dir/2.html* and other pages from that directory and its subdirectories, but is not accessible to *a.com/*. Furthermore, *https://a.com/* and *http://a.com/* share the cookie store unless a cookie is marked with a "secure" flag. Non-HTTPS sites may still set `secure` cookies in some implementations, just not read them back [48]. A web programmer can make cookie access less restrictive by setting a cookie's domain attribute to a postfix domain or the path name to be a prefix path. The browser ensures that a site can only set its own cookie and that a cookie is attached only to HTTP requests to that site.

The path-based security policy for cookies does not play well with SOP for scripts: scripts can gain access to all cookies belonging to a domain despite path restrictions.

Plugins. Current major browsers do not enforce any security on plugins and grant plugins access to the local operating system directly. The plugin content is subject to the security policies implemented in the plugin software rather than the browser.

3.2 Gazelle's security model

Gazelle's architecture is centered around protecting principals from one another by separating their respective resources into OS-enforced protection domains. Any sharing between two different principals must be explicit using cross-principal communication (or IPC) mediated by the browser kernel.

We use the same principal as the SOP, namely, the triple of `<protocol, domain-name, port>`. While it is tempting to have a more fine-grained principal, we need to be concerned with co-existing with current browsers [29, 43]: the protection boundary of a more fine-grained principal, such as a path-based principal, would break down in existing browsers. It is unlikely that web programmers would write very different versions of the same service to accommodate different browsers; instead, they would forego the more fine-grained principal and have a single code base.

The resources that need to be protected across principals [43] are memory such as the DOM objects and script objects, persistent state such as cookies, display, and network communications.

We extend the same principal model to all content types except scripts and style sheets (Section 4): the elements created by `<object>`, `<embed>`, `<img>`, and certain types of `<input>`[1] are treated the same as an `<iframe>`: the origin of the included content labels the principal of the content. This means that we *enforce* SOP on plugin content[2]. This is consistent with the existing movement in popular plugins like Adobe Flash Player [20]. Starting with Flash 7, Adobe Flash Player uses the exact domain match (as in SOP) rather than the earlier "superdomain" match (where *www.adobe.com* and *store.adobe.com* have the same origin) [2]; and starting with Flash 9, the default ActionScript behavior only allows access to same-origin HTML content unlike the earlier default that allows full cross-origin interactions [1].

Gazelle's architecture naturally yields a security policy that partitions all system resources across the SOP principal boundaries. Such a policy offers consistency across various resources. This is unlike current browsers where the security policies vary for different resources. For example, cookies use a different principal than that of scripts (see the above section); descendant navigation policy [7, 8] also implicitly crosses the SOP principal boundary (more in Section 5.1).

It is feasible for Gazelle to enable the same security policies as the existing browsers and achieve backward compatibility through cross-principal communications. Nevertheless, it is interesting to investigate the tradeoffs between supporting backward compatibility and eliminating insecure policies in today's browsers. We gave a preliminary discussion on this in Section 9.

4 Architecture

4.1 Basic Architecture

Figure 1 shows our basic architecture. A principal is the *unit of protection*. Principals need to be completely isolated in resource access and usage. Any sharing must be made explicit. Just as in desktop applications, where instances of an application are run in separate processes for failure containment and independent resource allocation, a principal instance is the *unit of failure containment* and the *unit of resource allocation*. For example, navigating to the same URL in different tabs corresponds to two instances of the same principal; when *a.com* embeds two *b.com* iframes, the *b.com* iframes correspond to two instances of *b.com*. However, the frames that share the same origin as the host page are in the same principal instance as the host page by default, though we allow the host page to designate an embedded same-origin frame or object as a separate principal instance for independent resource allocation and failure containment. Principal instances are isolated for all runtime resources, but principal instances of the same principal share persistent state such as cookies and other local storage. Protection unit, resource allocation unit, and failure containment unit can each use a different mechanism depending on the system implementation. Because the implementation of our principal instances contains native code, we use OS processes for all three purposes.

Our principal instance is similar to Google Chrome's site instance [37], but with two crucial differences: 1) Google Chrome considers the sites that share the same registrar-controlled domain name to be from the same site, so *ad.datacenter.com*, *user.datacenter.com*, and *datacenter.com* are considered to be the same site and belong to the same principal. In contrast, we consider them as separate principals. 2) When a site, say *a.com*, embeds another principal's content, say an `<iframe>` with source *b.com*, Google Chrome puts them into the same site instance. In contrast, we put them into separate principal instances.

The browser kernel runs in a separate protection domain and interposes between browser principals and the traditional OS. The browser kernel mediates the principals' access to system resources and enforces security policies of the browser. Essentially, the browser kernel functions as an operating system to browser principals and manages the protection and sharing of system resources for them. The browser kernel also manages the browser chrome, such as the address bar and menus. The browser kernel receives all events generated by the underlying operating system including user events like mouse clicks or keyboard entries; these events are then dispatched to the appropriate principal instance. When the user navigates a window by clicking on a hyperlink that points to an URL at a different origin, the browser kernel creates the protection domain for the URL's principal instance (if one doesn't exist already) to render the target page, destroys the protection domain of the hyperlink's host page, and re-allocates and re-initializes the window to the URL's principal instance. The browser

[1] `<input>` can be used to include an image using a "src" attribute.
[2] OP [21] calls this plugin policy the *provider domain policy*.

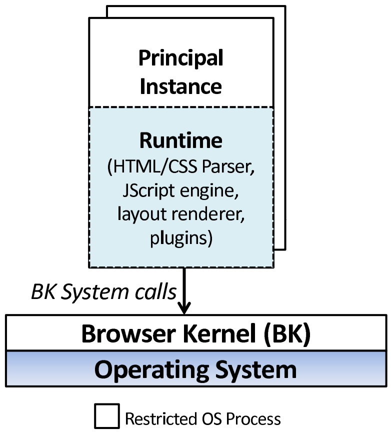


Figure 1: The Gazelle architecture

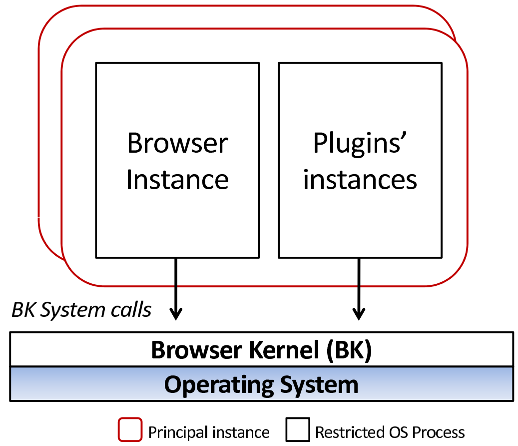


Figure 2: Supporting legacy protection

kernel is agnostic of DOM and content semantics and has a relatively simple logic.

The runtime of a principal instance performs content processing and is essentially an instance of today's browser components including HTML and style sheet parser, JavaScript engine, layout renderer, and browser plugins. The only way for a principal instance to interact with system resources, such as networking, persistent state, and display, is to use browser kernel's system calls. Principals can communicate with one another using message passing through the browser kernel, in the same fashion as inter-process communications (IPC).

It is necessary that the protection domain of a principal instance is a restricted or sandboxed OS process. The use of process guarantees the isolation of principals even in the face of attacks that exploit memory vulnerabilities. The process must be further restricted so that any interaction with system resources is limited to the browser kernel system calls. Native Client [47] and Xax [15] have established the feasibility of such process sandboxing.

This architecture can be efficient. By putting all browser components including plugins into one process, they can interact with one another through DOM intimately and efficiently as they do in existing browsers. This is unlike the OP browser's approach [21] in which all browser components are separated into processes; chatty DOM interactions must be layered over IPCs through the OP browser kernel, incurring unnecessary overhead without added security.

Unlike all existing browsers except OP, this architecture can enforce browser security policies on plugins, namely, plugin content from different origins are segregated into different processes. Any plugin installed is unable to interact with the operating system and is only provided access to system resources subject to the browser kernel allowing that access. In this architecture, the payload that exploits plugin vulnerabilities will only compromise the principal with the same origin as the malicious plugin content, but not any other principals nor browser kernel.

The browser kernel supports the following system calls related to content fetching in this architecture (a more complete system call table is shown in Table 3):

- *getSameOriginContent (URL)*: Fetch the content at URL that has the same origin as the issuing principal regardless of the content type.
- *getCrossOriginContent (URL)*: Fetch the script or style sheet content from URL; URL may be from different origin than the issuing principal. The content type is determined by the `content-type` header of the HTTP response.
- *delegate (URL, windowSpec)*: Delegate a display area to a different principal of URL and fetch the content for that principal.

The semantics of these system calls is that the browser kernel can return cross-origin script or style content to a principal based on the content-type header of the HTTP response, but returns other content if and only if the content has the same origin as the issuing principal, abiding the same-origin policy. All the security decisions are made and enforced by the browser kernel alone.

4.2 Supporting Legacy Protection

The system call semantics in the basic architecture has one subtle issue: cross-origin script or style sheet sources are readable by the issuing principal, which does not conform with the existing SOP. The SOP dictates that a script can be executed in a cross-origin fashion, but the access to its source code is restricted to same origin only.

A key question to answer is that whether a script should be processed in the protection domain of its provider (indicated in "src"), in the same way as frames, or in the protection domain of the host page that embeds the script. To answer this question, we must examine the primary intent of the script element abstraction. Script is primarily a *library* abstraction (which is a necessary and useful abstraction) for web programmers to include in their sites and runs with the privilege of the includer sites [43]. This is in contrast with the frame abstractions: Programmers put content into cross-origin frames so that the content runs as the principal of its own provider and be protected from other principals. Therefore, a script should be handled by the protection domain of its includer.

In fact, it is a flaw of the existing SOP to offer protection for cross-origin script source. Evidence has shown that it is extremely dangerous to hide sensitive data inside a script [22]. Numerous browser vulnerabilities exist for failing to provide the protection.

Unfortunately, web sites that rely on cross-origin script source protection, exist today. For example, GMail's contact list is stored in a script file, at the time of writing. Furthermore, it is increasingly common for web programmers to adopt JavaScript Object Notation (JSON) [31] as the preferred data-interchange format. Web sites often demand such data to be same-origin access only. To prevent such data from being accidentally accessed through `<script>` (by a different origin), web programmers sometimes put "while (1);" prior to the data definition or put comments around the data so that accidental script inclusion would result in infinite loop execution or a no-op.

In light of the existing use, new browser architecture design must also offer the cross-origin script source protection. One way to do this is to strip all authentication-containing information, such as cookies and HTTP authentication headers, from the HTTP requests that retrieve cross-origin scripts so that the web servers will not supply authenticated data. The key problem with this approach is that it is not always clear what in an HTTP request may contain authentication information. For example, some cookies are used for authentication purposes and some are not. Stripping all cookies may impair functionality when the purpose of some cookies are not for authentication purposes. In another example, a network may use IP addresses for authentication, which are impossible to strip out.

We address the cross-origin script source protection problem by modifying our architecture slightly, as shown in Figure 2. The modification is based on the following observation. Third-party plugin software vulnerabilities have surged recently [36]. Symantec reports that in 2007 alone there are 467 plugin vulnerabilities [42], which is about one magnitude higher than that of browser software. Clearly, plugin software should be trusted much less than browser software. Therefore, for protecting cross-origin script or style sheet source, we place more trust in the browser code and let the browser code retrieve and protect cross-origin script or style sheet sources: for each principal, we run browser code and plugin code in two separate processes. The plugin instance process *cannot* issue the `getCrossOriginContent()` and it can only interact with cross-origin scripts and style sheets through the browser instance process.

In this architecture, the quality of protecting cross-origin script and style-sheet source relies on the browser code quality. While this protection is not perfect with native browser code implementation, the architecture offers the same protection as OP, and stronger protection than the rest of existing browsers. The separation of browser code and plugin code into separate processes also improves reliability by containing plugin failures.

In recent work, Native Client [47] and Xax [15] have presented a plugin model that uses sandboxed processes to contain each browser principal's plugin content. Their plugin model works perfectly in our browser architecture. We do not provide further discussions on plugins in our paper.

5 Cross-Principal, Cross-Process Display and Events Protection

Cross-principal service composition is a salient nature of the web and is commonly used in web applications. When building a browser as a multi-principal OS, this composition raises new challenges in display sharing and event dispatching: when a web site embeds a cross-origin frame (or objects, images), the involved principal instances share the display at the same time. Therefore, it is important that the browser kernel 1) discerns display and events ownership, 2) enforces that a principal instance can only draw in its own display areas, 3) dispatches UI events to only the principal instance with which the user is interacting. An additional challenge is that the browser kernel must accomplish these without access to any DOM semantics.

From a high level, in Gazelle principal instances are responsible for rendering content into bitmap objects, and our browser kernel manages these bitmap objects and chooses when and where to display them. Our architecture provides a clean separation between the act of rendering web content and the policies of how to display this content. This is a stark contrast to today's browsers that intermingle these two functions, which has led to numerous security vulnerabilities [18, 44].

Our display management fundamentally differs from that of the traditional multi-user OSes, such as Unix and Windows. Traditional OSes offer no cross-principal display protection. In X, all the users who are authorized (through `.Xauthority`) to access the display can access one another's display and events. Experimental OSes like EROS [41] have dealt with cross-principal display protection. However, the browser context presents new challenges that are absent in EROS, such as dual ownership of display and cross-principal transparent overlays.

5.1 Display Ownership and Access Control

We define *window* to be a unit of display allocation and delegation. Each window is allocated by a *landlord* principal instance or the browser kernel; and each window is delegated to (or rented to) a *tenant* principal instance. For example, when the web site *a.com* embeds a frame sourced at *b.com*, *a.com* allocates a window from its own display area and delegates the window to *b.com*; *a.com* is the landlord of the newly-created window, while *b.com* is the tenant of that window. The same kind of delegation happens when cross-origin object and image elements are embedded. The browser kernel allocates top-level windows (or tabs). When the user launches a site through address-bar entry, the browser kernel delegates the top-level window to the site, making the site a tenant. We decided against using "parent" and "child" terminologies because they only convey the window hierarchy, but not the principal instances involved. In contrast, "landlord" and "tenant" convey both semantics.

Window creation and delegation result in a `delegate(URL, position, dimensions)` system call. For each window, the browser kernel maintains the following state: its landlord, tenant, position, dimensions, pixels in the window, and the URL location of the window content. The browser kernel manages a three-dimensional display space where the position of a window also contains a stacking order value (toward the browsing user). A landlord provides the stacking order of all its delegated windows to the browser kernel. The stacking order is calculated based on the DOM hierarchy and the CSS z-index values of the windows.

Because a window is created by a landlord and occupied by a tenant, the browser kernel must allow reasonable window interactions from both principal instances without losing protection. When a landlord and its tenant are from different principals, the browser kernel provides access control as follows:

- *Position and dimensions*: When a landlord embeds a tenant's content, the landlord should be able to retain control on what gets displayed on the landlord's display and a tenant should not be able to reposition or resize the window to interfere with the landlord's display. Therefore, the browser kernel enforces that only the landlord of a window can change the position and the dimensions of a window.

	Landlord	**Tenant**
position (x,y,z)	RW	
dimensions (height, width)	RW	R
pixels		RW
URL location	W	RW

Table 1: Access control policy for a window's landlord and tenant

- *Drawing isolation*: Pixels inside the window reflect the tenant's private content and should not be accessible to the landlord. Therefore, the browser kernel enforces that only the tenant can draw within the window. (Nevertheless, a landlord can create overlapping windows delegated to different principal instances.)
- *Navigation*: Setting the URL location of a window navigates the window to a new site. Navigation is a fundamental element of any web application. Therefore, both the landlord and the tenant are allowed to set the URL location of the window. However, the landlord should not obtain the tenant's navigation history that is private to the tenant. Therefore, the browser kernel prevents the landlord from reading the URL location. The tenant can read the URL location as long as it remains being the tenant. (When the window is navigated to a different principal, the old tenant will no longer be associated with the window and will not be able to access the window's state.)

Table 1 summarizes the access control policies in the browser kernel. In existing browsers, these manipulation policies also vaguely exist. However, their logic is intermingled with the DOM logic and is implemented at the object property and method level of a number of DOM objects which all reside in the same protection domain despite their origins. This had led to numerous vulnerabilities [18, 44]. In Gazelle, by separating these security policies from the DOM semantics and implementation, and concentrating them inside the browser kernel we achieve more clarity in our policies and much stronger robustness of our system construction.

The browser kernel ensures that principal instances other than the landlord and the tenant cannot manipulate any of the window states. This includes manipulating the URL location for navigation. Here, we depart from the existing descendant navigation policy in most of today's browsers [7, 8]. Descendant navigation policy allows a landlord to navigate a window created by its tenant even if the landlord and the tenant are different principals. This is flawed in that a tenant-created window is a resource that belongs to the tenant and should not be controllable by a different principal.

Existing literature [7, 8] supports the descendant navigation policy with the following argument: since existing browsers allow the landlord to draw over the tenant, a landlord can simulate the descendant navigation by overdrawing. Though overdrawing can *visually* simulate navigation, navigation is much more powerful than overdrawing because a landlord with such descendant navigation capability can interfere with the tenant's operations. For example, a tenant may have a script interacting with one of its windows and then effecting changes to the tenant's backend; navigating the tenant's window requires just one line of JavaScript and could effect undesirable changes in the tenant's backend. With overdrawing, a landlord can imitate a tenant's content, but the landlord cannot send messages to the tenant's backend in the name of the tenant.

5.2 Cross-Principal Events Protection

The browser kernel captures all events in the system and must accurately dispatch them to the right principal instance to achieve cross-principal event protection. Networking and persistent-state events are easy to dispatch. However, user interface events pose interesting challenges to the browser kernel in discerning event ownership, especially when dealing with overlapping, potentially transparent cross-origin windows: major browsers allow web pages to mix content from different origins along the z-axis where content can be occluded, either partially or completely, by cross-origin content. In addition, current standards allow web pages to make a frame or portions of their windows transparent, further blurring the lines between principals. Although these flexible mechanisms have a slew of legitimate uses, they can be used to fool users into thinking they are interacting with content from one origin, but are in fact interacting with content from a different origin. Zalewski [48] gave a taxonomy on "UI redressing" or clickjacking attacks which illustrated some of the difficulties with current standards and how attackers can abuse these mechanisms.

To achieve cross-principal events protection, the browser kernel needs to determine the *event owner*, the principal instance to which the event is dispatched. There are two types of events for the currently active tab: stateless and stateful. The owner of a stateless event like a mouse event is the tenant of the window (or display area) on which the event takes place. The owner of a stateful event such as a key-press event is the tenant of the current in-focus window. The browser kernel interprets mouse clicks as focus-setting events and keeps track of the current in-focus window and its principal instance.

The key problem to solve then is to determine the window on which a stateless or focus-setting event takes place. We consider a determination to have high *fidelity* if the determined event owner corresponds to the user intent. Different window layout policies directly affect the fidelity of this determination. We elaborate on our explorations of three layout policies and their implications on fidelity.

Existing browsers' policy. The layout policy in existing browsers is to draw windows according to the DOM hierarchy and the z-index values of the windows. Existing browsers then associate a stateless or focus-setting event to the window that has the highest stacking order. Today, most browsers permit page authors to set transparency on cross-origin windows [48]. This ability can result in poor fidelity in determining the event owner in the face of cross-principal transparent overlays. When there are transparent, cross-origin windows overlapping with one another, it is impossible for the browser kernel to interpret the user's intent: the user is guided by what she sees on the screen; when two windows present a mixed view, some user interfaces visible to the user belong to one window, and yet some belong to another. The ability to overlay transparent cross-origin content can be extremely dangerous: a malicious site can make an iframe sourced at a legitimate site transparent and overlaid on top of the malicious site [48], fooling the users to interact with the legitimate site unintentionally.

2-D display delegation policy. This is a new layout policy that we have explored. In this policy, the display is managed as two-dimensional space for the purpose of delegation. Once a landlord delegates a rectangular area to a tenant, the landlord cannot overdraw the area. Thus, no cross-principal content can be overlaid. Such a layout constraint will enable perfect fidelity in determining an event ownership that corresponds to the user intent. It also yields better security as it can prevent all UI redressing attacks except clickjacking [48]. Even clickjacking would be extremely difficult to launch with this policy on our system since our cross-principal memory protection makes reading and writing the scrolling state of a window an exclusive right of the tenant of the window.

However, this policy can have a significant impact on backward compatibility. For example, a menu from a host page cannot be drawn over a nested cross-origin frame or object; many sites would have significant constraints with their own DOM-based pop-up windows created with `divs` and such (rather than using `window.open` or `alert`), which could overlay on cross-origin frames or objects with existing browsers' policy; and a cross-origin image cannot be used as a site's background.

Opaque overlay policy. This policy retains existing browsers' display management and layout policies as much as possible for backward compatibility (and additionally provides cross-principal events protection), but lets the browser kernel enforce the following layout invariant or constraint: for any two dynamic content-containing windows (e.g., frames, objects) $win1$ and $win2$, $win1$ can overlay on $win2$ iff $(Tenant_{win1} == Tenant_{win2})$ || $(Tenant_{win1} \neq Tenant_{win2}$ && $win1$ $is\ opaque)$. This policy effectively constrains a pixel to be associated with just one principal, making event owner determination trivial. This is in contrast with the existing browsers' policy where a pixel may be associated with more than one principals when there are transparent cross-principal overlays. This policy allows same-origin windows to transparently overlay with one another. It also allows a page to use a cross-origin image (which is static content) as its background. Note that no principal instance other than the tenant of the window can set the background of a window due to our memory protection across principal instances. So, it is impossible for a principal to fool the user by setting another principal's background. The browser kernel associates a stateless event or a focus-setting event with the dynamic content-containing window that has the highest stacking order.

This policy eliminates the attack vector of overlaying a transparent victim page over an attacker page. However, by allowing overlapping opaque cross-principal frames or objects, it allows not only legitimate uses, such as those denied by the 2D display delegation policy, but it also allows an attacker page to cover up and expose selective areas of a nested cross-origin victim frame or object. The latter scenario can result in infidelity. We leave as future work the mitigation of such infidelity by determining how much of a principal's content is exposed in an undisturbed fashion to the user when the user clicks on the page.

We implemented the opaque overlay policy in our prototype.

6 Security Analysis

In Gazelle, the trusted computing base encompasses the browser kernel and the underlying OS. If the browser kernel is compromised, the entire browser is compromised. If the underlying OS is compromised, the entire host system is compromised. If the DNS is compromised, all the non-HTTPS principals can be compromised. When the browser kernel, DNS, and the OS are intact, our architecture guarantees that the compromise of a principal instance does not give it any capabilities in addition to those already granted to it through browser kernel system call interface (Section 4).

Next, we analyze Gazelle's security over classes of browser vulnerabilities. We also make a comparison with popular browsers with a study on their past, known vulnerabilities.

- Cross-origin vulnerabilities:

  By separating principals into different protection domains and making any sharing explicit, we can much more easily eliminate cross-origin vulnerabilities. The only logic for which we need to ensure correctness is the origin determination in the browser kernel.

  This is unlike existing browsers, where origin validations and SOP enforcement are spread through the browser code base [10], and content from different principals coexists in shared memory. All of the cross-origin vulnerabilities illustrated in Chen et al. [10] simply do not exist in our system; *no* special logic is required to prevent them because all of those vulnerabilities exploit implicit sharing.

  Cross-origin script source can still be leaked in our architecture if a site can compromise its browser instance. Nevertheless, only that site's browser instance is compromised, while other principals are intact, unlike all existing browsers except OP.

- Display vulnerabilities:

  The display is also a resource that Gazelle's browser kernel protects across principals, unlike existing browsers (Section 5). Cross-principal display and events protection and access control are enforced in the browser kernel. This prevents a potentially compromised principal from hijacking the display and events that belong to another principal. Display hijacking vulnerabilities have manifested themselves in existing browsers [17, 26] that allow an attacker site to control another site's window content.

- Plugin vulnerabilities:

  Third-party plugins have emerged to be a significant source of vulnerabilities [36]. Unlike existing browsers, Gazelle's design requires plugins to interact with system resources only by means of browser kernel system calls so that they are subject to our browser's security policy. Plugins are contained inside sandboxed processes so that basic browser code doesn't share fate with plugin code (Section 4). A compromised plugin affects the principal instance's plugin process only, and not other principal instances nor the rest of the system. In contrast, in existing browsers except OP, a compromised plugin undermines the entire browser and often the host system as well.

  A DNS rebinding attack results in the browser labeling resources from different network hosts with a common origin. This allows an attacker to operate within SOP and access unauthorized resources [30]. Although Gazelle does not fundamentally address this vulnerability, the fact that plugins must interact with the network through browser kernel system

	IE 7	Firefox 2
Origin validation error	6	11
Memory error	38	25
GUI logic flaw	3	13
Others	-	28
Total	47	77

Table 2: Vulnerability Study for IE 7 and Firefox 2

calls defeats the multipin form of such attacks.

We analyzed the known vulnerabilities of two major browsers, Firefox 2 [3] and IE 7 [35], since their release to November 2008, as shown in Table 2. For both browsers, memory errors are a significant source of errors. Memory-related vulnerabilities are often exploited by maliciously crafted web pages to compromise the entire browser and often the host machines. In Gazelle, although the browser kernel is implemented with managed C# code, it uses native .NET libraries, such as network and display libraries; memory errors in those libraries could still cause memory-based attacks against the browser kernel. Memory attacks in principal instances are well-contained in their respective sandboxed processes.

Cross-origin vulnerabilities, or origin validation errors, constitute another significant share of vulnerabilities. They result from the implicit sharing across principals in existing browsers and can be much more easily eliminated in Gazelle because cross-principal protection is exclusively handled by the browser kernel and because of Gazelle's use of sandboxed processes.

In IE 7, there are 3 GUI logic flaws which can be exploited to spoof the contents of the address bar. For Gazelle, the address bar UI is owned and controlled by our browser kernel. We anticipate that it will be much easier to apply code contracts [6] in the browser kernel than in a monolithic browser to eliminate many of such vulnerabilities.

In addition, Firefox had other errors which didn't map into these three categories, such as JavaScript privilege escalation, URL handling errors, and parsing problems. Since Gazelle enforces security properties in the browser kernel, any errors that manifest as the result of JavaScript handling and parsing are limited in the scope of exploit to the principal instance owning the page. URL handling errors could occur in our browser kernel as well.

7 Implementation

We have built a Gazelle prototype mostly as described in Section 4. We have not yet ported an existing plugin onto our system. Our prototype runs on Windows Vista with .NET framework 3.5 [4]. We next discuss the implementation of two major components shown in Figure 2: the browser kernel and the browser instance.

Browser Kernel. The browser kernel consists of approximately 5k lines of C# code. It communicates with principal instances using system calls and upcalls, which are implemented as asynchronous XML-based messages sent over named pipes. An overview of browser kernel system calls and upcalls is presented in Table 3. System calls are performed by the browser instance or plugins and sometimes include replies. Upcalls are messages from the browser kernel to the browser instance.

Display management is implemented as described in Section 5 using .NET's Graphics and Bitmap libraries. Each browser instance provides the browser kernel with a bitmap for each window of its rendered content using a `display` system call; each change in rendered content results in a subsequent `display` call. For each top-level browsing window (or tab), browser kernel maintains a stacking order and uses it to compose various bitmaps belonging to a tab into a single master bitmap, which is then attached to the tab's PictureBox form. This straightforward display implementation has numerous optimization opportunities, many of which have been thoroughly studied [33, 38, 40], and which are not the focus of our work.

Browser instance. Instead of undertaking a significant effort of writing our own HTML parser, renderer, and JavaScript engine, we borrow these components from Internet Explorer 7 in a way that does not compromise security. Relying on IE's Trident renderer has a big benefit of inheriting IE's page rendering compatibility and performance. In addition, such an implementation shows that it is realistic to adapt an existing browser to use Gazelle's secure architecture.

In our implementation, each browser instance embeds a Trident `WebBrowser` control wrapped with an *interposition layer* which enforces Gazelle's security properties. The interposition layer uses Trident's COM interfaces, such as `IWebBrowser2` or `IWebBrowserEvents2`, to hook sensitive operations, such as navigation or frame creation, and convert them into system calls to the browser kernel. Likewise, the interposition layer receives browser kernel's upcalls, such as keyboard or mouse events, and synthesizes them in the Trident instance.

For example, suppose a user navigates to a web page `a.com`, which embeds a cross-principal frame `b.com`. First, the browser kernel will fetch `a.com`'s HTML content, create a new `a.com` process with a Trident component, and pass the HTML to Trident for rendering. During the rendering process, we intercept the frame navigation event for `b.com`, determine that it is cross-principal, and cancel it. The frame's DOM element in `a.com`'s DOM is left intact as a placeholder, making the interpo-

Type	Call Name	Description
syscall	getSameOriginContent(URL)	retrieves same origin content
syscall	getCrossOriginContent(URL)	retrieves script or css content
syscall	delegate(URL, delegatedWindowSpec)	delegates screen area to a different principal
syscall	postMessage(windowID, msg, targetOrigin)	cross-frame messaging
syscall	display(windowID, bitmap)	sets the display buffer for the window
syscall	back()	steps back in the window history
syscall	forward()	steps forward in the window history
syscall	navigate (windowID, URL)	navigates a window to URL
syscall	createTopLevelWindow (URL)	creates a new browser tab for the URL specified
syscall	changeWindow (windowID, position, size)	updates the location and size of a window
syscall	writePersistentState (type, state)	allows writing to origin-partitioned storage
syscall	readPersistentState (type)	allows reading of origin-partitioned storage
syscall	lockPersistentState (type)	locks one type of origin-partitioned storage
upcall	destroy(windowID)	closes a browser instance
upcall	resize(windowID, windowSpec)	changes the dimensions of the browser instance
upcall	createPlugin(windowID, URL, content)	creates a plugin instance
upcall	createDocument(windowID, URL, content)	creates a browser instance
upcall	sendEvent(windowID, eventInfo)	passes an event to the browser instance

Table 3: Some Gazelle System Calls

sition transparent to `a.com`. We extract the frame's position, dimensions, and CSS properties from this element through DOM-related COM interfaces, and send this information in a `delegate` system call to the browser kernel to allow the landlord `a.com` to "rent out" part of its display area to the tenant `b.com`. The browser kernel then creates a new `b.com` process (with a new instance of Trident), and asks it to render `b.com`'s frame. For any rendered display updates for either `a.com` or `b.com`, our interposition code obtains a bitmap of display content from Trident using the `IViewObject` interface and sends it to the browser kernel for rendering.

One intricacy we faced was in rerouting all network requests issued by Trident instances through the browser kernel. We found that interposing on all types of fetches, including frame, script, and image requests, to be very challenging with COM hooks currently exposed by Trident. Instead, our approach relies on a *local web proxy*, which runs alongside the browser kernel. We configure each Trident instance to use our proxy for all network requests, and the proxy converts each request into a corresponding system call to the browser kernel, which then enforces our security policy and completes the request.

One other implementation difficulty that we encountered was to properly manage the layout of cross-origin images. It is easy to render a cross-origin image in a separate process, but difficult to extract the image's correct layout information from the host page's Trident instance. We anticipate this to be an overcomable implementation issue. In our current prototype, we are keeping cross-origin images in the same process as their host page for proper rendering of the pages.

Our interposition layer ensures that our Trident components are never trusted with sensitive operations, such as network access or display rendering. However, if a Trident renderer is compromised, it could bypass our interposition hooks and compromise other principals using the underlying OS's APIs. To prevent this, we are in the process of implementing an OS-level sandboxing mechanism, which would prevent Trident from directly accessing sensitive OS APIs. The feasibility of such a browser sandbox has already been established in Xax [15] and Native Client [47].

To verify that such an implementation does not cause rendering problems with popular web content, we used our prototype to manually browse through the top 20 Alexa [5] web sites. We checked the correctness of Gazelle's visual output against unmodified Internet Explorer and briefly verified page interactivity, for example by clicking on links. We found that 19 of 20 web sites rendered correctly. The remaining web site exposed a (fixable) bug in our interposition code, which caused it to load with incorrect layout. Two sites experienced crashes (due to more bugs) when trying to render embedded cross-principal `<iframe>`'s hosting ads. However, the crashes only affected the `<iframe>` processes; the main pages rendered correctly with the exception of small blank spaces in place of the failed `<iframe>`'s. This illustrates a desirable security property of our architecture, which prevents malicious or misbehaving cross-origin tenants from affecting their landlords or other principals.

	Gazelle		Internet Explorer 7		Google Chrome	
	Time	Memory Used	Time	Memory Used	Time	Memory Used
1. Browser startup (no page)	668 ms	9 MB	635 ms	14 MB	500 ms	25 MB
2. New tab (blank page)	602 ms	14 MB	115 ms	0.7 MB	230 ms	1.8 MB
3. New tab (google.com)	939 ms	16 MB	499 ms	1.4 MB	480 ms	7.6 MB
4. Navigate from google.com to google.com/ads	955 ms	6 MB	1139 ms	3.1 MB	1020 ms	1.4 MB
5. Navigate to nytimes.com (with a cross-origin frame)	5773 ms	88 MB	3213 ms	53 MB	3520 ms	19.4 MB

Table 4: Loading times and memory overhead for a sequence of typical browser operations.

8 Evaluation

In this section, we measure the impact of our architecture on browser performance. All tests were performed on an Intel 3.00Ghz Core 2 Duo with 4GB of RAM, running 32-bit Windows Vista with a gigabit Ethernet connection. To evaluate Gazelle's performance, we measured page loading latencies, the memory footprint, and responsiveness of our prototype in comparison with IE7, a monolithic browser, and Google Chrome v1, a multi-process browser. We found that while Gazelle performs on-par with commercial browsers while browsing within an origin, it introduces some overhead for cross-origin navigation and rendering embedded cross-origin principals (e.g., frames). Nevertheless, our main sources of overhead stem from our interposition layer, various initialization costs for new browser instances, and the unoptimized nature of our prototype. We point out simple optimizations that would eliminate much of the overhead along the way.

Page load latency. Table 4 shows the loading times for a series of browser operations a typical user might perform using our prototype, IE7, and Google Chrome. The operations are repeated one after another within the same browser. A web page's loading time is defined as the time between pressing the "Go" button and seeing the fully-rendered web page. All operations include network latency.

Operation 1 measures the time to launch the browser and is similar for all three browsers. Although Gazelle's browser kernel is small and takes only 225 ms to start, Gazelle also initializes the local proxy subsystem (see Section 7), which takes an additional 443 ms. Operations 2 and 3 each carry an overhead of creating a new process in Gazelle and Chrome, but not IE7. Operation 4 reuses the same `google.com` process in Gazelle to render a same-origin page to which the user navigates via a link on `google.com`. Here, Gazelle is slightly faster than both IE7 and Chrome, possibly because Gazelle does not yet manage state such as browsing history between navigations. Finally, operation 5 causes Gazelle to create a new process for `nytimes.com` to render the popular news page [3]. In addition, NYTimes contains an embedded cross-principal `<iframe>`, which triggers window delegation and another process creation event in Gazelle. Gazelle's overall page load latency of 5773 ms includes the rendering times of both the main page and the embedded `<iframe>`, with the main page becoming visible and interactive to the user in 5085 ms.

Compared to both IE7 and Chrome, it is expected that Gazelle will have a performance overhead due to extra process creation costs, messaging overhead, and the overhead of our Trident interposition layer as well as Trident itself. Table 5 breaks down the major sources of overhead involved in rendering the three sites in Table 4.

Our Trident interposition layer is a big source of overhead, especially for larger sites like NYTimes.com, where it consumes 813 ms. Although we plan to optimize our use of Trident's COM interfaces, we are also limited by the Trident host's implementation of the hooks that we rely on, and by the COM layer which exposes these hooks. Nevertheless, we believe we could mitigage most of this latency if Trident were to provide us with a direct (non-COM) implementation for a small subset of its hooks that Gazelle requires.

Our local proxy implementation for network interposition constitutes another large source of overhead, for example 541 ms for NYTimes.com. Much of this overhead would disappear if Trident were to make direct network system calls to the browser kernel, rather than going through an extra proxy indirection. Another part of this overhead stems from the fact that the browser kernel currently releases web page data only when a whole network transfer finishes; instead, it could provide browser instances with chunks of data as soon as they arrive (e.g., by changing `getContent` system calls to the semantics of a UNIX `read()` system call), allowing them to better overlap network transfers with rendering.

Process creation is an expected source of overhead that

[3] In contrast, Chrome reuses the tab's old google.com process

increases whenever sites embed cross-principal content, such as NYTimes's cross-origin <iframe>. As well, each process must instantiate and initialize a new Trident object, which is expensive. As an optimization, we could use a worker pool of a few processes that have been pre-initialized with Trident. This would save us 275 ms on NYTimes's load time and 134 ms on `google.com`'s load time.

We encountered an unexpected performance hit when initializing named pipes that we use to transfer system calls: a new process's first write to a pipe stalls for a considerable time. This could be caused by initialization of an Interop layer between .NET and the native Win32 pipe interfaces, on which our implementation relies. We can avoid this overhead by either using an alternate implementation of a system call transfer mechanism, or pre-initializing named pipes in our worker pool. This would save us 439 ms in NYTimes's render time.

Retrieving bitmap display updates from Trident and sending them to the browser kernel is expensive for large, complex sites such as NYTimes.com, where this takes 422 ms. Numerous optimizations are possible, including image compression, VNC-like selective transfers, and a more efficient bitmap sharing channel between Trident and the browser kernel. Our mechanism for transferring bitmap updates currently performs an inefficient .NET-based serialization of the image's data (which takes 176 ms for NYTimes); passing this data directly would further improve performance.

Overall, we believe that with the above optimizations, Gazelle's performance would be on par with production browsers like Chrome or IE8; for example, we anticipate that NYTimes.com could be rendered in about 3.6 s.

Memory overhead. As a baseline measurement, the browser kernel occupies around 9MB of memory after a page load. This includes the user interface components of the browser to present the rendered page to the user and the buffers allocated for displaying the rendered page. Memory measurements do not include shared libraries used by multiple processes.

Table 4 shows the amount of memory for performing various browsing operations. For example, to open a new tab to a blank page, Gazelle consumes 14MB, and to open a new tab for `google.com`, Gazelle consumes an additional 16MB. Each empty browser instance uses 1.5MB of internal storage plus the memory required for rendered content. Given our implementation, the latter closely corresponds to Trident's memory footprint, which at the minimum consists of 14MB for a blank page. In the case of NYTimes, our memory footprint further increases because of structures allocated by the interposition layer, such as a local DOM cache.

Responsiveness. We evaluated the response time of a user-generated event, such as a mouse click. When the browser kernel detects a user event, it issues a *sendEvent* upcall to the destination principal's browser instance. Such calls take only 2 ms on average to transfer, plus 1 ms to synthesize in Trident. User actions might lead to display updates; for example, a display update for `google.com` would incur an additional 77 ms. Most users should not perceive this overhead and will experience good responsiveness.

Process creation. In addition to latency and memory measurements we also have tested our prototype on the top 100 popular sites reported by Alexa [5] to provide an estimate of the number of processes created for different sites. Here, we place a cross-origin image into a separate process to evaluate our design. The number of processes created is determined by the use of different-origin content on sites, which is most commonly image content. For the top 100 sites, the median number of processes required to view a single page is 4, the minimum is 1, and the maximum is 28 (caused by `skyrock.com`, which uses an image farm). Although creation of many processes introduces additional latency and memory footprint, we did not experience difficulties when Gazelle created many processes during normal browsing. Our test machine easily handles a hundred running processes, which are enough to keep 25 average web sites open simultaneously.

9 Discussions on compatibility vs. security

While Gazelle's architecture can be made fully backward compatible with today's web, it is interesting to investigate the compatibility cost of eliminating the insecure policies in today's browsers. We have considered several policies that differ from today's browsers but offer better security. We conducted a preliminary study on their compatibility cost. This is by no means a conclusive or complete study, but only a first look on the topic.

We mostly used the data set of the front pages of the top 100 most popular web sites ranked by Alexa [5]. We used a combination of browser instrumentation with automatic script execution and manual inspection in our study. We consider any visual differences in the rendering of a web page to be a violation of compatibility. We discuss our findings below.

Subdomain treatment Existing browsers and SOP make exceptions for subdomains (e.g., *news.google.com* is a subdomain of *google.com*) [39]: a page can set the `document.domain` property to suffixes of its domain and assume that identity. This feature was one of the few methods for cross-origin frames to communicate before the advent of `postMessage` [25]. Changing `document.domain` is a dangerous practice and violates the Principle of Least Privilege: Once a subdomain sets its domain to a suffix, it has no control over which other

		Latency		
Location	**Overhead**	blank site	google.com	nytimes.com
	Overhead before rendering			
Browser kernel	- process creation	44 ms	40 ms	78 ms
Browser instance	- creating interposed instances of Trident	94 ms	94 ms	197 ms
Browser instance	- named pipe initialization	137 ms	145 ms	439 ms
	Overhead during rendering			
Browser instance	- proxy-based network interposition	4 ms	134 ms	541 ms
Browser instance	- other Trident interposition	127 ms	122 ms	813 ms
	Overhead after rendering			
Browser instance	- bitmap capture	13 ms	35 ms	196 ms
Browser instance	- bitmap transfer	37 ms	67 ms	226 ms
Browser kernel	- display rendering	10 ms	11 ms	101 ms

Table 5: A breakdown of Gazelle's overheads involved in page rendering. Note that nytimes.com creates *two* processes for itself and an <iframe>; the other two sites create one process.

subdomains can access it. This is also observed by Zalewski [48]. Therefore, it would be more secure not to allow a subdomain to set `document.domain`.

Our experiments indicate that six of the top 100 Alexa sites set `document.domain` to a different origin, though restricting write access to `document.domain` might not actually break the operation of these web sites.

Mixed HTTPS and HTTP Content. When an HTTPS site embeds HTTP content, browsers typically warn users about the mixed content, since the HTTPS site's content can resist a network attacker, but the embedded HTTP content could be compromised by a network attacker.

When an HTTPS site embeds other HTTP principals (through <iframe>, <object>, etc.), HTTPS principals and HTTP principals will have different protection domains and will not interfere with each other.

However, when an HTTPS site embeds a script or style sheet delivered with HTTP, existing browsers would allow the script to run with the HTTPS site's privileges (after the user ignores the mixed content warning). This is dangerous because a network attacker can then compromise the HTTP-transmitted script and attack the HTTPS principal despite its intent of preventing network attackers. Therefore, a more secure policy is to deny rendering of HTTP-transmitted scripts or style sheets for an HTTPS principal. Instead of the Alexa top 100, we identified a few different sites that provide SSL sessions for parts of their web application: *amazon.com*, *mail.google.com*, *mail.microsoft.com*, *blogger.com*, and a few popular banking sites where we have existing accounts. This allows us to complete the login process during testing. These sites do not violate this policy. In addition, we have also gathered data from one of the author's browsing sessions over the course of a few months and found that out of 5,500 unique SSL URLs seen, less than two percent include HTTP scripts and CSS.

Layout policies. The opaque overlay policy allows only opaque (and not transparent) cross-origin frames or objects (Section 5.2). We test this policy with the top 100 Alexa sites by determining if any cross-origin frames or objects are overlapped with one another. We found that two out of 100 sites attempt to violate this policy. This policy does not generate rendering errors; instead, we convert transparent cross-origin elements to opaque elements when displaying content.

We also tested the 2D display delegation policy that we analyzed in Section 5.2. We found this policy to have higher compatibility cost than our opaque overlay policy: six of the top 100 sites attempt to violate this policy.

Sites that attempt to violate either policy have reduced functionality, and will render differently than what the web page author intends.

Plugins. Existing plugin software must be adapted (ported or binary-rewritten) to use browser kernel system calls to accomplish its tasks. Of top 100 Alexa sites, 34 sites use Flash, but no sites use any other kinds of plugins. This indicates that porting or adapting Flash alone can address a significant portion of the plugin compatibility issue.

10 Concluding Remarks

We have presented Gazelle, the first web browser that qualifies as a multi-principal OS for web site principals. This is because Gazelle's browser kernel *exclusively* manages resource protection, unlike all existing browsers which allow cross-principal protection logic to reside in the principal space. Gazelle enjoys the security and robustness benefit of a multi-principal OS: a compromise or failure of one principal leaves other principals and the browser kernel intact.

Our browser construction exposes challenging design issues that were not seen in previous work, such as providing legacy protection to cross-origin script source and cross-principal, cross-process display and event protection. We are the first to provide comprehensive solutions to them.

The implementation and evaluation of our IE-based prototype shows promise of a practical multi-principal OS-based browser in the real world.

In our future work, we are exploring the fair sharing of resources among web site principals in our browser kernel and a more in-depth study of the tradeoffs between compatibility and security in browser policy design.

11 Acknowlegements

We thank Spencer Low, David Ross, and Zhenbin Xu for giving us constant help and fruitful discussions. We thank Adam Barth and Charlie Reis for their detailed and insightful feedback on our paper. We also thank the following folks for their help: Barry Bond, Jeremy Condit, Rich Draves, David Driver, Jeremy Elson, Xiaofeng Fan, Manuel Fandrich, Cedric Fournet, Chris Hawblitzel, Jon Howell, Galen Hunt, Eric Lawrence, Jay Lorch, Rico Malvar, Wolfram Schulte, David Wagner, Chris Wilson, and Brian Zill. We also thank our paper shepherd Niels Provos for his feedback over our last revisions.

References

[1] Changes in allowScriptAccess default (Flash Player). `http://www.adobe.com/go/kb403183`.

[2] Developer center: Security changes in Flash Player 7. `http://www.adobe.com/devnet/flash/articles/fplayer_security.html`.

[3] Security advisories for Firefox 2.0. `http://www.mozilla.org/security/known-vulnerabilities/firefox20.html`.

[4] .NET Framework Developer Center, 2008. `http://msdn.microsoft.com/en-us/netframework/default.aspx`.

[5] Alexa, 2009. `http://www.alexa.com/`.

[6] M. Barnett, K. Rustan, M. Leino, and W. Schulte. The Spec# programming system: An overview. In LNCS, editor, *CASSIS*, volume 3362. Springer, 2004. `http://research.microsoft.com/en-us/projects/specsharp/`.

[7] A. Barth and C. Jackson. Protecting browsers from frame hijacking attacks, April 2008. `http://crypto.stanford.edu/websec/frames/navigation/`.

[8] A. Barth, C. Jackson, and J. C. Mitchell. Securing frame communication in browsers. In *In Proceedings of the 17th USENIX Security Symposium (USENIX Security)*, 2008.

[9] A. Barth, C. Jackson, C. Reis, and T. G. C. Team. The security architecture of the Chromium browser, 2008. `http://crypto.stanford.edu/websec/chromium/chromium-security-architecture.pdf`.

[10] S. Chen, D. Ross, and Y.-M. Wang. An Analysis of Browser Domain-Isolation Bugs and A Light-Weight Transparent Defense Mechanism. In *Proceedings of the ACM Conference on Computer and Communications Security*, 2007.

[11] R. S. Cox, J. G. Hansen, S. D. Gribble, and H. M. Levy. A Safety-Oriented Platform for Web Applications. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2006.

[12] D. Crockford. JSONRequest. `http://www.json.org/jsonrequest.html`.

[13] D. Crockford. The Module Tag: A Proposed Solution to the Mashup Security Problem. `http://www.json.org/module.html`.

[14] Document Object Model. `http://www.w3.org/DOM/`.

[15] J. R. Douceur, J. Elson, J. Howell, and J. R. Lorch. Leveraging legacy code to deploy desktop applications on the web. In *Proceedings of the Symposium on Operating Systems Design and Implementation*, 2008.

[16] Firefox 3 for developers, 2008. `https://developer.mozilla.org/en/Firefox_3_for_developers`.

[17] Mozilla Browser and Mozilla Firefox Remote Window Hijacking Vulnerability, 2004. `http://www.securityfocus.com/bid/11854/`.

[18] Security Advisories for Firefox 2.0. `http://www.mozilla.org/security/known-vulnerabilities/firefox20.html`.

[19] D. Flanagan. *JavaScript: The Definitive Guide*. O'Reilly Media Inc., August 2006.

[20] Adobe Flash Player 9 Security, July 2008. `http://www.adobe.com/devnet/flashplayer/articles/flash_player_9_security.pdf`.

[21] C. Grier, S. Tang, and S. T. King. Secure web browsing with the OP web browser. In *Proceedings of the 2008 IEEE Symposium on Securiy and Privacy*, 2008.

[22] J. Grossman. Advanced Web Attack Techniques using GMail. `http://jeremiahgrossman.blogspot.com/2006/01/advanced-web-attack-techniques-using.html`.

[23] W. H. A. T. W. Group. Web Applications 1.0, February 2007. `http://www.whatwg.org/specs/web-apps/current-work/`.

[24] HTML 5 Editor's Draft, October 2008. `http://www.w3.org/html/wg/html5/`.

[25] What's New in Internet Explorer 8, 2008. `http://msdn.microsoft.com/en-us/library/cc288472.aspx`.

[26] Microsoft Internet Explorer Remote Window Hijacking Vulnerability, 2004. `http://www.securityfocus.com/bid/11855`.

[27] S. Ioannidis and S. M. Bellovin. Building a secure web browser. In *Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference*, 2001.

[28] S. Ioannidis, S. M. Bellovin, and J. M. Smith. Sub-operating systems: a new approach to application security. In *Proceedings of the 10th workshop on ACM SIGOPS European workshop*, pages 108–115, New York, NY, USA, 2002. ACM.

[29] C. Jackson and A. Barth. Beware of Finer-Grained Origins. In *Web 2.0 Security and Privacy*, May 2008.

[30] C. Jackson, A. Barth, A. Bortz, W. Shao, and D. Boneh. Protecting Browsers from DNS Rebinding Attacks. In *Proceedings of ACM Conference on Computer and Communications Security*, 2007.

[31] JavaScript Object Notation (JSON). `http://www.json.org/`.

[32] D. Kristol and L. Montulli. HTTP State Management Mechanism. IETF RFC 2965, October 2000.

[33] T. W. Mathers and S. P. Genoway. *Windows NT Thin Client Solutions: Implementing Terminal Server and Citrix MetaFrame*. Macmillan Technical Publishing, Indianapolis, IN, November 1998.

[34] IEBlog: IE8 Security Part V: Comprehensive Protection, 2008. `http://blogs.msdn.com/ie/archive/2008/07/02/ie8-security-part-v-comprehensive-protection.aspx`.

[35] Microsoft security bulletin. `http://www.microsoft.com/technet/security/`.

[36] Microsfot Security Intelligence Report, Volume 5, 2008. `http://www.microsoft.com/security/portal/sir.aspx`.

[37] C. Reis and S. D. Gribble. Isolating web programs in modern browser architectures. In *Proceedings of Eurosys*, 2009.

[38] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper. Virtual network computing. *IEEE Internet Computing*, 2(1):33–38, 1998.

[39] J. Ruderman. The Same Origin Policy. `http://www.mozilla.org/projects/security/components/same-origin.html`.

[40] R. W. Scheifler and J. Gettys. The X window system. *ACM Transactions on Graphics (TOG)*, 5(2):79–109, April 1986.

[41] J. S. Shapiro, J. Vanderburgh, E. Northup, and D. Chizmadia. Design of the EROS TrustedWindow system. In *Usenix Security*, 2004.

[42] Symantec Global Internet Security Threat Report: Trends for July - December 07, April 2008.

[43] H. J. Wang, X. Fan, J. Howell, and C. Jackson. Protection and Communication Abstractions in MashupOS. In *ACM Symposium on Operating System Principles*, October 2007.

[44] Cross-Domain Vulnerability In Microsoft Internet Explorer 6. `http://cyberinsecure.com/cross-domain-vulnerability-in-microsoft-internet-explorer-6/`.

[45] The XMLHttpRequest Object. `http://www.w3.org/TR/XMLHttpRequest/`.

[46] W3C XMLHttpRequest Level 2. `http://dev.w3.org/2006/webapi/XMLHttpRequest-2/`.

[47] B. Yee, D. Sehr, G. Dardyk, B. Chen, R. Muth, T. Ormandy, S. Okasaka, N. Narula, and N. Fullagar. Native client: A sandbox for portable, untrusted x86 native code. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2009.

[48] M. Zalewski. Browser security handbook, 2008. `http://code.google.com/p/browsersec/wiki/Main`.