Cleaning Up After Cookies
Version 1.0

Katherine McKinley — kate[at]isecpartners[dot]com

iSEC Partners, Inc
444 Spear Street, Suite 105
San Francisco, CA 94105
https://www.isecpartners.com

December 31, 2008

Abstract

Modern web browsers and plugins are rapidly expanding web developers' ability to store data on users' systems, while simultaneously adding features which allow users the perception of more control over that data. Users need to be confident that their perceptions match reality. Unfortunately, the privacy modes offered by browsers are still evolving (several are only available as betas), and none remove all the tracking data users might expect them to block. A tool was created to set and report on different data stores. This paper presents the findings from running this tool using several major browsers with two plug-ins across three common operating systems. We find current browsers are unable to extend tracking protection to third party plug-ins such as Google Gears and Adobe Flash. Some of these require no user prompting under common configurations and even expose tracking data saved with one browser sites visited by a different browser. We also recommend approaches for solving these problems.

1 Introduction

Modern web browsers and browser plug-ins provide a rich set of interfaces for web sites to store information on end-users' systems. This data is used for credentials (username/passwords and equivalents), tracking users, storing preferences (interface customizations, volume controls), site data (security questions, images, cached data), identifying tokens, or other data. User's desire to control tracking data (and other data third-parties store on their systems) has lead to a number of browser features, but the effectiveness of these tools is difficult for the average consumer to gauge.

1.1 History

Before 1994, the only way a web-site could receive data about the user on the other end was by placing a session identifier in the URL, or as a value in a form. Unfortunately, this makes exchanging links a security problem, since the session identifier of one user would be given to another. Developers from Netscape Communications created an extension to the HTTP specification to add a new type of data, which would be set by the server and sent by the browser in every request to that server. They called this piece of data a cookie, and support first

appeared in Mosaic Netscape 0.9beta. Its first use was to determine if the user had visited the Netscape website previously, and if they had, to show them a slightly different page.

Cookies provide four main advantages to a software developer: they can hold session data, as in a shopping cart, they can be used to store login credentials, they can provide customization or personalization features, or they can be used to track a user's activity. These uses are not exclusive, as a single site may use one or more of these techniques.

One of the main user tracking concerns arises when a third party, other than the web-site they are visiting, is allowed to store data on the user's system. This is possible when you visit a web page and content at from another site is referenced (e.g. an advertisement). The third party content is loaded by the browser, and their server may be able set their own cookies for their domain (some browser settings prevent this). When a user visits a web site such as http://www.happykittens.example, which specifies an image coming from a different domain, for example, http://ads.adsadsads.example, the server at ads.adsadsads.example sends the user's browser a cookie containing a small bit of data, a number indicating a unique user in their system, and stores which site you are coming from. When the user then visits http://important.news.example, who also uses ads from http://ads.adsadsads.example, the usre's cookie is sent to the server at ads.adsadsads.example, effectively letting the server at ads.adsadsads.example know that you are viewing http://important.news.example.

The concerns over privacy in particular led to an article in the Financial Times[3] in October of 1996, while in 1996 and 1997, the US Federal Trade Commission began holding privacy workshops to determine in part the risk of cookies to Internet users' privacy[4][5][1][2]. This led to regulation restricting the US government's use of cookies to identify visitors to its sites. At the same time in Europe, the EU developed a privacy directive applicable to all member countries[1] whereby users must be informed when a web site wishes to store data on their computer, what that data is used for, and how to prevent that data from being stored. This requires web sites operating in Europe to allow a version of their services which does not require cookies, allowing users to opt out of that mechanism.

Due to the limitations of cookies, specifically the number and size of cookies available to a given web-site, as well as the need to send cookies with every request, Internet Explorer 5 introduced a new mechanism for storing data on a user's system called the userData store. This was presented to web developers as a way to increase storage and site data management capabilities while maintaining the same-origin security policy. It has been referred to as a "Super Cookie"[2] because the default storage for IE userData limits for Internet web sites is up to 1024KB of storage in documents up to 128KB in size. In contrast, a web server can expect to be able to store only 20 cookies of up to 4KB each, although in practice most browsers allow a greater number of cookies. Since the introduction of this feature, the Firefox browser implemented a method of storing key-value pairs called DOM[3] storage, where a web site is allowed to store data locally on a user's system to be retrieved later. It shares the same-origin restrictions as userData, and comes in two flavors, one of which (sessionStorage) is cleared on exit, and the other (globalStorage) which is persisted permanently. Finally, the current, pre-release, version of the HTML 5 specification includes a storage proposal which has both a key-value storage mechanism similar to Firefox as well as a method which supports a relational database model. This database feature is currently only supported in Apple's Safari browser. Support for it has been added to WebKit, allowing subsequent versions of other WebKit-based browsers (Google Chrome, Konqueror, Android, etc.) to include support for this feature, and is planned for Mozilla Firefox.

In addition to methods supported directly by web browsers, a user can install third party plugins which may break out of the restriction on local system access which is normally imposed by the browser. Google's Gears, a plug-in for allowing more powerful web applications by caching data and web pages for offline use, is one such example, and Adobe's Flash plugin is another. Google's Gears is a relatively rare plugin, which is installed specifi-

[1] See http://europa.eu/eur-lex/pri/en/oj/dat/2002/l_201/l_20120020731en00370047.pdf for the current EU privacy directive.
[2] http://www.discovermountainbiking.com/userdata.asp
[3] Document Object Model, see http://en.wikipedia.org/wiki/Document_Object_Model

cally to provide this functionality, while the most visibile use of Adobe's Flash is presenting video and animations.

Gears gives users control over what data Gears will store on their computer. First, the user must explicitly install Gears[4]. Second, Gears also allows users to confirm whether or not they want Gears to be available for a particular web site. Finally, Gears makes its settings readily available via the browser's UI, where the user is able to explicitly allow or deny an individual site. If the site is denied, it appears to that site as if the user had not allowed the use of Gears at all.

Adobe's Flash is currently installed on more systems than any specific browser[5]. Additionally, it is used in embedded and small scale computing devices such as the Nintendo Wii and the Nokia N770/N800 Internet Tablets, the Sony Playstation Portable and Playstation 3 and Leapfrog Enterprises' Leapster Multimedia Learning System. By default, Flash movies are not allowed to access the client's microphone or camera until the user has given explicit permission. Adobe's published privacy article for Flash[6] does not state whether this permission applies only to movies loaded from the same origin as the user has given permission, or if it is granted more broadly, such as to any movie allowed by the crossdomain.xml file[7]. For a given site, users can right click on the flash movie to modify these settings. There is no permission required for a flash movie to store data locally. Although the Settings dialogue for a site is loaded locally, the global Settings Manager application is loaded over the internet. Although parts of it are loaded via HTTPS, it is hosted on an HTTP page, providing no visual indication to the user that the Adobe web site is genuine.

2 Analysis of Browser-based Storage

This paper presents a simple tool to test the efficacy of browser data clearing mechanisms. It was run on several different combinations of operating system and browser, including beta versions of upcoming browsers with new features such as HTML 5 storage and privacy modes. Tested data storage in the initial version include HTTP cookies, HTML 5 session storage, Mozilla Firefox perisistent storage, HTML 5 database storage, IE userData, Adobe Flash and Google Gears. Due to the time constraints, Microsoft Silverlight was not included in this initial version. A Silverlight test and Addendum to this paper is forthcoming.

2.1 Methodology

The test consists of a simple web page which loads a JavaScript test harness, several JavaScript files containing the individual tests, and a Flash movie. Each test consists of setting the data, and reading it back out. Each time a test is run, it first checks to see if the data exists and if so, the test displays the data and updates the modified time. If the data does not exist on the first run through, or if the that particular data store has been effectively cleared, the test will report that no data was found, and attempt to insert a small amount of data along with a creation and modification time. The user can then view that data by either reloading the page or clicking on the button to re-test a particular item. Additionally, the result column is updated to contain the result of the current run. This can be one of several states:

1. Data found

[4] Google's Chrome browser comes with Gears pre-installed

[5] For Adobe's analysis of Flash market penetration, see http://www.adobe.com/products/player_census/flashplayer/version_penetration.html. A discussion of browser market share can be found at http://en.wikipedia.org/wiki/Usage_share_of_web_browsers

[6] http://www.adobe.com/devnet/flashplayer/articles/privacy.html

[7] For a discussion of security issues with crossdomain.xml, see http://www.isecpartners.com/files/iSEC-Attacking_AJAX_Applications.BH2006.pdf

2. No data found, setting
3. Unsupported
4. Disallowed
5. Unable to determine status

See appendices for source code. Additionally, the most current version of the source code and the tool itself is available at https://labs.isecpartners.com/breadcrumbs/breadcrumbs.html. Your test data may be sent to iSEC Partners. The browsers tested include: IE 7 and IE 8 Beta 2 on Windows, Mozilla Firefox 3.0.2 and 3.1 Beta 2 on Linux, Windows, and Mac OS X, Safari 3.1.2 on Windows and OS X, Opera 9.62 on Windows, and Google Chrome 1.0.154.36 on Windows. Google Gears 0.5.4.2 was used on both Linux and Windows. The most current versions of Adobe Flash were used: 10.0.12.36 for Windows and OS X, and 10.0.15.3 for Linux. Test systems include Windows XP SP3, Windows Vista SP1, Mac OS X 10.5, and Gentoo GNU/Linux current as of December, 2008.

2.2 Browser storage

All of the browsers tested provide a clear and easy method for users to clear the data set on their system by web sites. In Firefox 3.0.2 and 3.1 Beta 2, the menu item Tools → Clear Private Data brings up a dialog box which allows the user to choose which data to clear. If a user checks the box next to a data type, such as cookies, then all cookies are cleared from the user's system. For Firefox, Opera, and Google Chrome, this functionality performs as advertised. Both Safari 3.1 and IE 8 Beta 2 did not perform entirely as expected, though. The HTML 5 Database store on Safari is not cleared when resetting the private data, the user must go to their preferences and select Security, then click the "Show Databases" button on that tab to review or delete databases. For IE 8 Beta 2, the browser must be closed to actually clear the data for the running instance. In each of these cases, it is necessary to perform additional actions to effectively clear this data.

A user wanting to view what data is stored can, in Firefox 3.0.2 and 3.1 Beta 2, choose Edit → Preferences in the menu, select "Privacy", and will be able to view all the cookies stored on their system as well as clear individual sites if they choose. It is clear, relatively well understood, and all tested browsers include this this functionality as part of the browser's user interface.

A major new feature of IE8, Firefox 3.1, Apple Safari and Google Chrome is private browsing. This feature exists to prevent cookies, local storage, history, and caches from being persisted after the completion of a session, even if the data normally would be. While they are very similar features, each one works slightly differently. See [8], [6], or [7] for discussion of how these modes are expected to work. In articles reviewed for this paper, the vendor or commenter makes claims that no data from a private session is recorded on the user's computer. In fact, all of the existing private browsing modes have some form of data which is not cleared when users enter or leave private browsing modes. Although Chrome cleared the only tested type of data it stored, it was surprising to find that Gears data was not cleared, since Gears is included in the browser. However, this behavior is consistent across all browsers tested, as we will see later. Firefox 3.1 Beta 2 clears cookies and session storage properly, but the persistent storage (window.globalStorage) is preserved between a normal and private browsing session. With IE 8 (Beta 2), both cookies and session storage were cleared properly, however the IE userData stores were not cleared between the normal and private browsing sessions.

Safari on Windows fared the worst of all in these tests with respect to private browsing, and did not clear any data at all, either before entering or after exiting the private mode. On OS X, Safari's behavior was quirky; in no case was the HTML 5 database storage cleared before or after private browsing. Previously set cookies seem

to continue to be available if the user entered a private browsing session, but if the user started the browser and went directly into private browsing, it seemed to behave as expected.

2.3 Google Gears

Gears is an open source project sponsored by Google to allow web sites to store data so that it can be used offline. This makes web-based alternatives to enterprise and communications applications feasible for use by laptop users who may not always have access to a network suitable for retrieving that information at the time they want to edit it. Users who install Gears must do so through the normal application installation process for their platform. Once it is installed, when the user visits a Gears-enabled web site and Gears has not previously been allowed for that site, the user is prompted to allow or deny the use of Gears. If they choose to allow Gears to store data, they are not prompted on subsequent visits. However, if they decide to block Gears at a later date, they can easily do so via a menu added to the browser. This does not delete the downloaded data, but does block access to it—the page is not aware that the data even exists. Under all tested browsers' privacy modes, however, Gears is still able to access all of its data, allowing a site with Gears access to continue to view previously stored data.

An installation of Gears creates a data store for each individual browser. This means that, for example, a user who downloads Firefox 3.0.2 and uses Gears there, and then visits the same site in IE 8 Beta 2, it will not have that data shared between browser instances. This sets up an equivalent expectation to privacy as user cookies—users do not expect that different browsers share access to this potentially sensitive information. Even though Google's Gears is a model plugin in many ways—requiring user consent to store data, browser isolation—it should warn users that their data is available even in privacy modes.

2.4 Adobe Flash

Adobe Flash offers developers the ability to create dynamic applications using a language similar to JavaScript called ActionScript[8]. These applications are tied to the network, and have broad capabilities to load code and data over the internet. Adobe includes methods for developers to bypass the web browser's same-origin security policy, allowing an application hosted on one domain to read data or code hosted on another. In the current version, Flash supports very limited access to local resources. If a user has a camera and / or microphone, a web site may request to use them. Sites may not turn them on without the user's consent. Flash does not get user consent when handling locally stored data via the *SharedObject* store. Data is silently persisted on the user's system and no indication in the browser is available to indicate that Flash has stored data.

Flash is also browser-independent in its storage location. When a user loads a web site with one browser which sets a Flash object, they may wish to view that page anonymously in a different browser. By using a universal data store, a Flash provides developers with a method for persisting data across all the browsers a user might use. This is an issue because the new private browsing modes becoming available also have access to the same Flash data as the user's regular browser instances. A survey of locally stored objects in Flash finds volume control preferences, potentially identifying information such as user aliases or identification numbers, and bank multi-factor authentication images or codes[9].

Flash does not use the browser interface to offer users the ability to modify their privacy or storage settings.

[8] Both languages are based on ECMAScript. See http://en.wikipedia.org/wiki/ECMAScript for more information on the ECMAScript standard

[9] Examples include the Bank of America http://www.bankofamerica.com/onlinebanking/index.cfm?template=site_key&statecheck=CA, Hampden Bank https://www.hampdenbank.com/news/whydoihavetoent_71/, and Security Bank Corporation http://www.reuters.com/article/pressRelease/idUS121553+10-Mar-2008+MW20080310, which uses Arcot Systems technology http://www.arcot.com/

Instead, Flash includes a Settings option in the menu when users right-click a flash movie. Adobe does not allow developers to hide the Settings menu, and only the settings for the current site can be viewed and manipulated via the right-click menu[10]. In order to view or modify the settings for all sites, or see which sites are storing data, the user must visit a special Flash movie hosted on www.macromedia.com via the insecure HTTP protocol. This loads a stub Flash application via HTTP, which then loads the remainder of the application via HTTPS. It does not indicate to the user that they are communicating with Adobe in a secure manner, although it appears to validate the SSL certificate correctly. A user who is unable to access the website hosting the control (e.g., restrictive firewall rules) is consequently unable to view or delete the data stored on their computer. Attempting to re-host the HTTP or HTTPS portions on a third party site was not attempted.

3 Conclusion

We have presented a tool for testing browsers functionality for clearing private data, and browsing in a private mode. While many browsers do a decent job of clearing data when requested, some have minor problems. Third party plug-ins like Adobe Flash, which is far more popular than any individual browser or platform, seem to undermine the data protection schemes offered by all common browsers, however. While browsers are introducing more features with privacy implications, such as persistent local storage, they have mostly integrated the management of this type of information into a single location. When users want to ensure their privacy with respect to information stored via the browser standard methods, they can go to a single location to clear the data, use a separate browser, or use a working private browsing mode, if available.

Plug-ins need to take extra steps to ensure the privacy of their users. The clear best practices in this area, as exemplified by Google's Gears, prompts users before allowing a site to store data on their system, holds a per-browser data store, and integrates their management UI into the browser UI. Adobe Flash does none of these things, instead silently allowing web sites to store data, uses one global data store for all browsers, and uses a settings UI accessible only when the user is connected to the Internet.

Browser vendors and plug-in vendors should cooperate to make their platforms more trustworthy. A set of standard APIs to communicate the need for plug-ins to clear data for a particular origin, all sites, or even a date range needs to be developed, and its use required of all plugins. In the absence of these APIs, plugins which require use of any local system resources should prompt before allowing web sites to store data locally, and integrate the management of interface into the standard browser API.

4 Acknowledgements

I would like to thank David Thiel for suggesting data storage mechanisms and review, as well as Jesse Burns, Chris Palmer, and April King for their review and encouragement.

[10] The author of this paper has observed the Settings menu greyed out, rendering it unusable. This behavior was not reproducable.

A HTML Source

The HTML Page source used for loading the tests:

```
<html>
<head>
  <title>Breadcrumbs Tracker</title>
  <script src='utils.js'></script>
  <script src='testharness.js'></script>
  <script src='cookie.js'></script>
  <script src="sessionStorage.js"></script>
  <script src="persistentStorage.js"></script>
  <script src="openDatabase.js"></script>
  <script src="userData.js"></script>
  <script src="flashTracker.js"></script>
  <script src="gears.js"></script>
  <link rel=stylesheet type="text/css" href="breadcrumbs.css"/>
</head>
<body onLoad='testHarness.init();testHarness.run();'>
<div align="left"><img src="iseclogo.png" alt="iSEC Parters, Inc"></div>
<h3>Breadcrumbs</h3>
<div>
  Click <a href="javascript:return false;" onclick="window.open('help.html', 'Breadcrumbs Help', 'height=256,wi
  The latest version of the paper can be found <a href="http://www.isecpartners.com/files/iSEC_Cleaning_Up_After
<div id='dynamicChecks'>
  <span>
    <div class="testname" style="text-align:left;font-weight:bold;"><span class="testname">Test name</span></div
    <div class="passfail" style="text-align:center;font-weight:bold;"><span class="passfail">Result</span></div>
    <div class="testvalue" style="text-align:center;font-weight:bold;"><span class="testvalue">Contents</span><
    <div class="button_container" style="text-align:center;font-weight:bold;"><span class="button_container">Run

    <div class="clear"> </div>
  </span>
</div>
</div>
<!--
<div>
<p>
<h4>Javascript Shell</h4>
<textarea id='inputField' cols=80 rows=5 name='input' value=''>
</textarea>
<br>
<input type=submit onClick='shellInput();return true;'>
</p>
</div>
-->
<div>
<h4>Log Output</h4>
<p id='logArea'>
</p>
</div>
</body>
</html>
```

Listing 1: breadcrumbs.html

B Test Harness Javascript

The code used to load and run the tests:

```
var TestHarness = function() {
  // each test consists of an entry of name value pairs, where value
  // is an object implementing the test() method
  this.tests = [];
  this.tests_idx = {};

  this.init = function() {
    var dchecks = getElementById('dynamicChecks');
    for(var k in this.tests) {
      this.init_tester(dchecks, this.tests[k]);
    }
  }

  this.results = {
    'data_found': 'Data Found',
    'nodata': 'No data found, setting ...',
    'undetermined': 'Unable to determine status',
    'unsupported': 'Method unsupported',
    'error': 'Error setting or retrieving data'
  };

  this.set_result = function(testname, result) {
    try {
      var elem = getElementById('passfail_' + testname);
      var oelem = getElementById('passfail_result_' + testname);
      var msg = this.results[result];
      if(!msg) {
        msg = this.results['undetermined'];
      }

      elem.removeChild(oelem);
      var nelem = mkElem('span', { 'id': 'passfail_result_' + testname, 'class': 'passfail'} );
      nelem.innerHTML = msg;
      nelem.className = "passfail";
      elem.appendChild(nelem);
      return msg;
    } catch (e) {
      printf("Exception in set_result: %s\n", e);
    }
    return 'error setting result';
  }

  this.run = function() {
    printf("running " + this.tests.length + " tests\n");

    for(var k in this.tests) {
      printf("Running %s\n", this.tests[k].name);

      this.run_test(this.tests[k].name);
    }
  }

  this.run_test = function(name) {
    var test = this.tests[this.tests_idx[name]];
    printf("run_test %s\n", test.name);
    if(test == null || !isDefined(test)) {
      return;
    }
    this.init_tester(getElementById('dynamicChecks'), test);
    var sp = mkElem('span', { 'id' : 'check_span_' + test.name });

    var hd = mkElem('div', { 'class': 'testname' });
    hd.innerHTML = '<span class="testname">' + test.print_name + '</span>';
    hd.className = "testname";
    sp.appendChild(hd);
```

```
66
67      var passfail = mkElem('div', { 'id': 'passfail_' + test.name, 'name': 'passfail_' + test.name, 'class': 'pa
68      passfail.innerHTML = '<span id=passfail_result_' + test.name + ' class="passfail"> </span>';
69      passfail.className = "passfail";
70      sp.appendChild(passfail);
71
72      var res = mkElem('div', { 'id': 'testvalue_' + test.name, 'name': 'testvalue_' + test.name, 'class': 'testva
73      var res_contents = mkElem('span', { 'id': 'testvalue_contents_' + test.name, 'name': 'testvalue_contents_' +
74      res_contents.innerHTML = test.test();
75      res.className = "testvalue";
76      res_contents.className = "testvalue";
77      res.appendChild(res_contents);
78      sp.appendChild(res);
79
80      var button_container = mkElem('div', { 'class': 'button_container' });
81      var bspan = mkElem('span', {'class': 'button_container'});
82      var rerun = mkElem('button', { 'onclick': 'testHarness.run_test("' + name + '");'});
83      rerun.onclick = function() { testHarness.run_test(name); };
84      rerun.innerHTML = "Run this test again";
85      rerun.className = "button_container";
86      bspan.appendChild(rerun);
87      button_container.appendChild(bspan);
88      sp.appendChild(button_container);
89      var cd = mkElem('div', { 'class': 'clear' });
90      cd.className = 'clear';
91      sp.appendChild(cd);
92      test.div.appendChild(sp);
93      test.finalize();
94      this.set_result(test.name, test.result);
95    }
96
97    this.register = function(obj) {
98      this.tests_idx[obj.name] = this.tests.length;
99      this.tests.push(obj);
100   }
101
102   this.init_tester = function(parentElem, obj) {
103     var ndiv = mkElem('div', { 'id': obj.name, 'name': obj.name, 'class': 'tester' });
104     if(obj.div) {
105       parentElem.replaceChild(ndiv, obj.div);
106     } else {
107       parentElem.appendChild(ndiv);
108     }
109     obj.div = ndiv;
110   }
111
112   this.getTests = function() { return this.tests; }
113
114   this.printStorage = function() {
115     var storSupp = "";
116     for(var i in window) {
117       if(i == "sessionStorage") {
118         storSupp += "Session Storage ";
119       }
120       else if(i == "globalStorage") {
121         storSupp += "Global Storage ";
122       }
123       else if(i == "localStorage") {
124         storSupp += "Local Storage ";
125       }
126       else if(i == "openDatabase") {
127         storSupp += "Database Storage ";
128       }
129     }
130     printf("%s\n", storSupp);
```

```
131     }
132
133     this.setPassFail = function (name, msg) {
134       var pf = getElementById("passfail_" + name);
135       if(pf) {
136         pf.innerHTML = "<span class=passfail>" + msg + "</span>";
137       }
138     }
139   }
140
141   var test = function(obj) {
142   }
143
144   var testHarness = new TestHarness();
145
146
147   var NullTest = function() {
148     this.name           = "NullTest";
149     this.print_name     = "Null Test";
150     this.div            = null;
151     this.result         = null;
152
153     this.finalize = function() { return; }
154
155     this.test = function() {
156       this.result = 'nodata';
157       return this.name + " Completed";
158     }
159   }
160
161   //testHarness.register(new NullTest());
```

Listing 2: testharness.js

C Cookie test

The code for testing setting and retrieving cookies in Javascript:

```
1   /*
2   document.cookie="xyzzy=grue; expires=Jan 19 2038 03:14:08 UTC; path=/";
3   document.cookie="foo=bar; path=/";
4   (new Cookie()).parseCookie();
5    */
6
7   var Cookie = function() {
8     var that = new NullTest();
9     this.name = "Cookie";
10    this.print_name = "Cookie";
11    this.domain = document.domain;
12    this.div = null;
13    this.cookie = {};
14    this.result = null;
15
16    this.parseCookie = function() {
17      this.cookie = {};
18      if(null == document.cookie || '' == document.cookie) {
19        return;
20      }
21      var contents = document.cookie.split(';');
22      for (var crumb in contents) {
```

```
23          var tmp = contents[crumb].replace('s/\s+$//').split('=');
24          this.cookie[tmp[0]] = tmp[1];
25        }
26      }
27
28      this.finalize = function() { return; }
29
30      this.test = function() {
31        this.parseCookie();
32        document.cookie="xyzzy=grue " + (new Date()).getTime() + " ; expires=Jan 19 2038 03:14:08 UTC; path=/";
33        document.cookie="foo=bar " + (new Date()).getTime() + "; path=/";
34        if(null == this.cookie) {
35          this.result = 'nodata';
36          return "Cookies Disabled";
37        }
38        var ret = "<table>";
39        ret += "<tr><th>Name</th><th>Value</th></tr>";
40        try {
41          this.result = 'nodata';
42          for (var k in this.cookie) {
43            ret += "<tr><td>" + k + "</td><td>" + this.cookie[k] + "</td></tr>";
44            this.result = 'data_found';
45          }
46        } catch (e) {
47          printf("Caught exception: %s\n" + e);
48          this.result = 'error';
49        }
50        ret += "</table>";
51        if(this.result == 'nodata') {
52          ret = 'Attempted to set cookie to: ' + document.cookie;
53        }
54        return ret;
55      }
56  }
57
58  testHarness.register(new Cookie());
```

Listing 3: cookie.js

D Changelog

v1.0 30 December, 2008

* Initial revision

References

[1] J. Berman, J. Goldman, D. J. Weitzner, and D. K. Mulligan. Statement of the Center for Democracy and Technology before the Federal Trade Commission Workshop on Consumer Privacy on the Global Information Infrastructure. http://www.cdt.org/testimony/960605berman.shtml, June 1996. 2

[2] J. Berman and D. Mulligan. CDT Comments to the FTC Consumer and Children's Online Privacy. http://www.cdt.org/privacy/issues/pii/970415_cdt_ftc2.shtml, April 1997. 2

[3] L. Bransten. Cookies leave a bitter taste: Invasive data collection is widespread. Financial Times, London, October 1996. 2

[4] U. S. FTC. FTC Workshop On Consumer Privacy In Cyberspace To Be Held In June 1996. http://www.ftc.gov/opa/1996/05/privinit.shtm, May 1996. 2

[5] U. S. FTC. Consumers' and Children's Privacy Online, Computer Database, and Unsolicited E-Mail: To Be Explored at FTC Privacy Week — June 10-13. http://www.ftc.gov/opa/1997/06/privweek.shtm, June 1997. 2

[6] G. Keizer. Firefox finally gets privacy mode. http://www.computerworld.com/action/article.do?command=viewArticleBasic&a December 2008. 4

[7] R. Naraine. Google Chrome, the security tidbits. http://blogs.zdnet.com/security/?p=1837, September 2008. 4

[8] A. Zeigler. IE8 and privacy. http://blogs.msdn.com/ie/archive/2008/08/25/ie8-and-privacy.aspx, August 2008. 4