A Survey of Cookie Management Functionality and Usability in Web Browsers

Ka-Ping Yee

University of California, Berkeley
ping@zesty.ca
http://zesty.ca

Abstract. This paper presents an objective analysis of the usability of various kinds of software for handling cookies and managing stored cookies, including popular Web browsers and specialized cookie management tools. Cookie handling is probably one of the most common privacy-related issues affecting computer users, since browsing the Web is such a popular activity on personal computers, and many information sites, services, and advertising companies on the Web use cookies to track user behaviour.

1 Introduction

One definition of privacy is "the right to be let alone," suggested by Justice Louis D. Brandeis in Olmstead v. United States, 277 U.S. 438 (1928). It follows from this definition that tools for managing privacy should be especially concerned with the degree to which their use inconveniences a user. If the task of maintaining privacy becomes excessively distracting, the privacy management tools have defeated their own purpose.

One of the most common uses of a personal computer today is to browse the Web. Together with the rapid rise of the Internet, privacy concerns have come into the spotlight. More than ever before, people are using information and services in a medium where their activities can be recorded and traced. Privacy issues on the Internet can be broadly classified into two categories: those relating to information that users knowingly provide or publish, and those relating to information that is collected without their knowledge. This paper is concerned with the second category of information, which is primarily gathered from logs of user accesses to Web servers.

The communication protocol for transmitting information on the Web is the Hypertext Transfer Protocol [2]. HTTP is a stateless protocol: that is, although a particular user might visit many different pages on a Web site, each transaction is independent, carrying information about only a single request and response. Raw logs from HTTP servers contain only lists of these independent requests. In order to gather information about user behaviour, one would need to be able to determine which requests came from the same user.

Although the IP address of the requesting computer can sometimes be used to identify a user, there is no reliable correspondence: a single IP address might

be used by different users at different times, and many different IP addresses might be used by a single user. For instance, most users of dial-up and wireless Internet connections use DHCP to configure their IP address automatically, so they may be issued new IP addresses from session to session.

The cookie mechanism enlists the help of the Web browser to provide a way to relate requests to other requests, which helps servers to identify requests from the same users and to identify users returning to a site from session to session.

2 Cookies

"Magic cookies" (today called just "cookies") in the context of the World-Wide Web were first introduced in Netscape Navigator as an extension to the Hypertext Transfer Protocol [8]. Each HTTP request includes a list of headers giving details about the request, and the server's reply begins with a list of headers as well. To implement cookies, Netscape Navigator introduced two new HTTP headers, "Set-Cookie" and "Cookie". Although these headers did not become part of the HTTP specification when it was documented in RFC 2068 [1], they were standardized in a separate specification in February 1997, RFC 2109 [3]. In October 2000, RFC 2109 was superseded by RFC 2965 [4], which introduced the newer "Set-Cookie2" and "Cookie2" headers.

A Web server can include the "Set-Cookie" header in its reply to instruct the client to remember a small piece of state information, and to include a copy of that information in future requests. If the client supports cookies, it will include a "Cookie" header in future requests according to criteria specified in the "Set-Cookie" command. The state information in the cookie consists of a pair of strings, a name and a value. Setting a cookie has an effect much like setting the value of a string variable. For example, after Alice has logged into a Web site, she might be given a cookie with the name "user" and the value "alice" so that her browser labels all future requests to the same site as coming from her. Aside from the name and value, a "Set-Cookie" header may specify a comment, domain, maximum age, path, security flag, and version number (of the cookie protocol). Four of these parameters control the scope in which the cookie appears: the domain, maximum age, path, and security flag.

- The **domain** specifies the set of Internet hosts to which the browser is expected to send the cookie. It can either be a hostname, which specifies just a single host, or a domain prefixed with a dot, which specifies all hosts under that domain.
- The **maximum age** specifies the length of time, in seconds, after which the cookie should expire and be discarded.
- The **path** specifies the set of URLs for which the browser is expected to send the cookie. It gives a string which is to be prefix-matched against the requested URL.
- The **security flag** specifies whether the browser is to send the cookie only over an encrypted transport.

The "Set-Cookie2" header introduced in RFC 2965 introduces three additional parameters: comment URL, discard flag, and port list. Like the comment string, the comment URL is present merely to offer documentation on the alleged use of the cookie.

- The **discard flag** indicates whether the browser is expected to discard the cookie when the user quits the browser.
- The **port list** specifies a list of TCP ports to which the cookie may be sent. By default, cookies are sent to any port.

3 Advantages of Cookies

Without cookies, Web applications must either be stateless or must implement other techniques for maintaining state.

Within a session, state can be embedded in the URLs of hyperlinks and the hidden fields of forms, if an application wants to avoid using cookies. In this way, further requests from the same user can be made to contain either the persistent information itself or a unique token that can be used to associate requests with a persistent session. However, this introduces an extra burden on the Web application and may add the need for a server-side database of user information. Furthermore, if the application wants to maintain session state while allowing the user to click on ordinary hyperlinks, then the presence of state tokens in such URLs can create problems for bookmarking.

Between successive sessions, the only way to maintain per-user state in Web applications without using cookies is to assign each user a unique URL for accessing the application. In a cookie-less application, as soon as the user leaves the site, all state information in the browser is lost. Consequently, if a cookie-less application is to be personalized to suit each user, then either every new session must begin with a login procedure, or each user must use a unique URL for accessing the application. Not many sites use the latter technique in current practice.

In summary, cookies can be used in order to make Web applications substantially easier to develop; they can enable a wider range of user interaction possibilities; and they provide a way to personalize Web applications that may be more convenient for most users.

4 Drawbacks of Cookies

Most implementations of cookies are based on the assumption that each installed copy of a Web browser has exactly one user, and also to some degree on the assumption that each user uses exactly on Web browser on one computer. These assumptions are not always true.

When a computer or browser is used by more than one person, as with an Internet kiosk or as when one lends the use of a computer to a friend, each user inherits the cookies left by the previous user. This may cause Web applications

to assume preferences or state information that does not apply. More seriously, since cookies are sometimes used for authentication and identification, they can cause users to gain unintended permission to access a previous user's private data or, for example, make purchases on a previous user's account.

When a person uses more than one browser on a computer, or uses multiple computers, cookie information remains with the original browser and computer (the browser that received the "Set-Cookie" command) and does not follow the user. This can cause confusion or inconvenience or, in extreme cases, can lead a user to expect that a trust relationship exists between the user and a Web site when in fact it does not. As a side note, some browsers store cookies on disk in a browser-specific location, and are vulnerable to corruption of cookie files when multiple instances of the browser are running at the same time.

5 Privacy Implications

Clearly, cookies can be used to track user behaviour, and to record information that users may not expect to be recorded. The use of cookies changes the "nymity" of Web transactions: without cookies each request is largely anonymous, but when cookies are used, requests become pseudonymous. They can even become fully identifiable if, at any point during a session, personal identifying information is associated with a request. Furthermore, some control over the boundaries of pseudonymity is given up to the Web sites issuing the cookies. Cookies prevent users from establishing a separate pseudonymous identity at each Web site or for each session. To achieve this, users must actively restrict the use of cookies. Because cookies are so often silently accepted by Web browsers, some users may expect their Web transactions to be anonymous already and not know that cookies are affecting their privacy. Though some users may expect or want each Web transaction to be anonymous

Finally, if cookies are activated without notifying the user, the user may be unaware of the security consequences of letting someone else use their browser.

5.1 Domain Crossing

As mentioned above, user privacy is adversely affected if cookies can be used to transmit information or associate user sessions across different domains. To try to prevent this, the cookie protocol has always required that servers can only set cookies with Domain values within their own domain. For example, a server with the Internet hostname `alice.example.com` can set a cookie that the browser sends to `bob.example.com` or `example.com` or any other host within `.example.com`, but not outside that domain. However, there remain two ways that cookie information can cross domains.

Embedded Links. HTML documents can contain links that are automatically traversed by the browser (such as embedded images, frames, or pop-up windows). If those links refer to a third party's site, this effectively transmits notification of

a user's browsing habits on a Web site to a third party. Furthermore, whenever a browser follows a link from one site to another, it includes the `Referer:` header[1] in its request, giving the URL of the referring site. Thus the third party can determine exactly which pages are being visited. If pages on different domains refer to embedded images on the same third-party site (for example, two sites display ads supplied by the same advertising agency), the third party can then correlate sessions across multiple domains.

It is worth noting that both RFC 2109 and RFC 2965 specifically discuss privacy concerns and make recommendations for browser behaviour. They draw a conceptual distinction between *verifiable* and *unverifiable* transactions, where a verifiable transaction is defined to be one where the user is able to know the URL being requested before the request takes place. Both specifications declare that browsers should not send cookies to third parties in unverifiable transactions, in order to address the concern just described. However, in practice most browsers do send such cookies.

Three-Level Domain Names. The cookie specification allows servers to set cookies within domains that contain at least one embedded dot. Hence, it is legal to set a cookie for all domains under `.example.com`, but illegal to set a cookie for all domains under `.com`. This is reasonable if we assume that organizational boundaries appear at the second level of the domain name hierarchy. However, this is not an adequate representation of trust boundaries in all cases. Many other countries have adopted a three-level domain naming convention. For instance, non-profit organizations in Australia have domain names of the form `.example.org.au`, and companies in the United Kingdom have domain names of the form `.example.co.uk`. Because the string "`co.uk`" contains an embedded dot, any server under `.co.uk` can legally set cookies for all domains under `co.uk`, and so sites in multiple domains can exchange user-specific information with each other in this way.

The original Netscape cookie proposal [8] explicitly attempted to rule out this problem, but for some reason there is no such restriction in the current cookie standard [4]. Moreover, even though the potential for this problem has been recognized, many browsers still exhibit information-leaking behaviour [5].

6 Motivation for Cookie Tools

We have observed that cookies have legitimate uses (indeed, may sometimes serve an essential function) but can also do serious harm to privacy. Moreover, they are in such widespread use on the Web that it is virtually impossible to avoid them. We conclude that it is infeasible to simply reject all cookies or accept all cookies unconditionally. Therefore, finer-grained control of cookies is necessary, and the provision of such control is a significant responsibility of Web browsers.

[1] Unfortunately, the word "referrer" was misspelled by the implementors of this header at Netscape. The misspelling became too widely used to retract, and was eventually standardized in this form in the HTTP specification.

7 Browser Comparison

This survey of browser functionality examined browsers on Linux, MacOS, and Windows platforms. The browsers tested were as follows:

- Galeon 1.2.5 (Linux)
- Konqueror 2.2.2 (Linux)
- Mozilla 1.0.0 (Linux)
- Opera 6.10 (Linux)
- Internet Explorer 5.1 (MacOS)
- OmniWeb 4.1 (MacOS)
- Opera 6.0 (MacOS)
- Internet Explorer 5.0 (Windows)
- Internet Explorer 6.0 (Windows)
- Mozilla 1.2.1 (Windows)
- Opera 6.05 (Windows)
- Opera 7 beta (Windows)

Each browser was examined to find the cookie-handling options it offered and the default settings of those options, as well as all of the functionality it provided for handling cookies upon their arrival, setting up rules for automatically handling cookies, and managing stored cookies. For each option or functional capability, the number of keyboard or mouse operations required to change the option or execute the function was measured.

When measuring keyboard operations, each keystroke is counted as one operation. For a key combination, each key is counted as one operation; for example, pressing Alt+Q is considered two operations.

When measuring mouse operations, each targeting movement is counted as one operation. For example, clicking on a button is one operation. Selecting a command from a pull-down menu is two operations (point-and-click on the menu title, then point-and-click on the command). Selecting an item in a drop-down list is two operations (point-and-click on the field to pop down the list, then point-and-click on the desired option in the list).

In all cases, the count of operations starts from a view of a Web page in a single window with the focus, and includes all operations necessary to return to that starting point (for example, operations to dismiss any dialogs that were opened in the course of performing the task).

Please refer to the included tables for the results of the analysis. Each cell contains the number of operations needed to perform a particular task using only the mouse or only the keyboard. In some cases the number of operations may vary depending on the current state of the user interface, and so a range of values is given. Where a feature is unavailable or inaccessible in the user interface, a cell is left blank. In certain cases, there exist some (but not all) user interface states in which a task is impossible to perform using only the keyboard, so the maximum required number of operations is given as infinity.

			Global Cookie Policy								Cookie Arrival Prompt											
	Browser	Default Policy	accept all	accept first-party	accept all for session	prompt per cookie	prompt per site	reject all	view policy	Default Option	accept cookie	accept cookie for session	reject cookie	accept all from site	allow from site for session	reject all from site	accept all from domain	reject all from domain	accept all	accept all for session	reject all
	MOUSE OPERATIONS																				
Linux	Galeon 1.2.5	accept all	4–6	4–6		4–6		4–6	3–5	accept	1		1	2		2					
Linux	Konqueror 2.2.2	prompt all	6			6		6	4	reject	1–2		1–2	1–2		1–2			1–2		1–2
Linux	Mozilla 1.0.0	accept if P3P okay	6	6		6		6	5	accept	1		1	2		2					
Linux	Opera 6.11	accept all	4–5	4–6	4–5	4–5		4–5	3–4	accept	1	3	1	3		3	3	3	2		2
MacOS	Internet Explorer 5.1	accept all	5–7			5–7	5–7	5–7	3–5	accept	1		1								
MacOS	OmniWeb 4.1	accept for session	5–6		5–6	5–6		5–6	3–4	accept	1–2	1–2	1–2	1–2	1–2	1–2			1–2		1–2
MacOS	Opera 6.0	accept all	5–6	5–8	4–5	5–6		3	3–4	accept	1	3	1	3		3	3	3	2		2
Windows	Internet Explorer 5.0	accept all	9–10		9–10	10		9–10	7	accept	1		1						2		
Windows	Internet Explorer 6.0	complicated*	6	7–9	8–10	7–9		6	4–6	accept	1		1	2		2					
Windows	Mozilla 1.2.1	accept first-party	6	6	6–7	6		6	5	accept	1		1	2		2					
Windows	Opera 6.05	accept all	5–6	6	5	5–6		5–6	3–4	accept	1	3	1	3		3	3	3	2		2
Windows	Opera 7 beta	accept all	5–6	6	5	5–6		5–6	3–4	accept	1	3	1	3		3	3	3	2		2
	KEYBOARD OPERATIONS																				
Linux	Galeon 1.2.5	accept all	6–20	5–21		5–23		5–22	4–17	accept	1		1	2		2					
Linux	Konqueror 2.2.2	prompt all	19			19–20		19–20	11	reject	2–4		1–3	2–4		1–3			2–4		1–3
Linux	Mozilla 1.0.0	accept if P3P okay	12–14	12–14		15		12–14	9	accept	1		1	2		2					
Linux	Opera 6.11	accept all	6–8	7–12	5–7	6–8		5–6	3–4	accept	1	5	2	7		7	5	5	5		4
MacOS	Internet Explorer 5.1	accept all							20	accept	1		1								
MacOS	OmniWeb 4.1	accept for session	15–˅		15–˅			15–˅	10–˅	accept	1–5	5–7	6–8	1–4	5–6	6–7			1–5	5–7	6–8
MacOS	Opera 6.0	accept all								accept	1		1								
Windows	Internet Explorer 5.0	accept all	20		20	21		19	13	accept	1		1						3		
Windows	Internet Explorer 6.0	complicated*	10–14	16–21	18–22	16–19		10–14	8–11	accept	1		1	2		2					
Windows	Mozilla 1.2.1	accept first-party	11–12	11	14–16	13		11–12	9	accept	1		1	2		2					
Windows	Opera 6.05	accept all	5–13	10–14	5–7	9–12		5–13	3–5	accept	1	4	1	5		6	7	8	2		2
Windows	Opera 7 beta	accept all	5–˅	10–˅	5–˅	9–˅		5–˅	3–˅	accept	1	4	1	5		6	7	8	2		2

* For a description of the default settings in Internet Explorer 6, see section 7.1.

		Automatic Filtering Rules							Cookie Storage Management					
	Browser	accept all from current site	reject all from current site	accept all from specified site*	reject all from specified site*	remove filtering rule**	remove all filtering rules	view rules	remove cookie**	remove and reject all from site**	remove all cookies in domain**	remove all cookies	view cookies	Special Abilities
	MOUSE OPERATIONS													
Linux	Galeon 1.2.5	3	3			6 + sel	5	4	6	6		5	4	remove cookie and block site
	Konqueror 2.2.2			8 + n	8 + n	6 + sel	5	4	7		7 + sel	6	5	cookie tree
	Mozilla 1.0.0		4			7 + sel	6	5	6			5	4	P3P policy criteria
	Opera 6.11			6–7 + n	5–6 + n	4–5 + sel	5–6	3–4				4		
MacOS	Internet Explorer 5.1								5–7				3–5	prompt per site
	OmniWeb 4.1					5–6 + sel	6–7	3–4	5		5 + sel	6	3–4	unified cookie and rule display
	Opera 6.0				7–8 + n	7–8 + sel	6–7	5–6						
Windows	Internet Explorer 5.0			8–9 + n					10				7	accept session cookies
	Internet Explorer 6.0			7 + n	12 + n	7 + sel	7	6	10			5	7	P3P policy criteria
	Mozilla 1.2.1		4			7 + sel	6	5	6			5	4	configurable XUL prefbar
	Opera 6.05			8 + n	10 + n	7 + sel		6				4		rules can specify first or third party
	Opera 7 beta			8 + n	10 + n	7 + sel		6				4		rules can specify first or third party
	KEYBOARD OPERATIONS													
Linux	Galeon 1.2.5	4	4			20 + sel	9	5	19 + sel	19 + sel		8	5	
	Konqueror 2.2.2			14 + n	15 + n	22 + sel	13	11	20 + sel		20 + sel	15	13	
	Mozilla 1.0.0		5			11 + sel	10	7	16 + sel			15	5	
	Opera 6.11			11–12 + n	10–11 + n	9–10 + sel	12–13	3–4				6		
MacOS	Internet Explorer 5.1												20	
	OmniWeb 4.1							10–ˇ					10–ˇ	
	Opera 6.0													
Windows	Internet Explorer 5.0								13 + sel				11	
	Internet Explorer 6.0			13 + n	12 + n	16 + sel	13	11	13 + sel			7	11	
	Mozilla 1.2.1		5			11 + sel	10	7	13 + sel			11	5	
	Opera 6.05			12 + n	12 + n	12 + sel		6				6		
	Opera 7 beta			12 + n	12 + n	12 + sel		6–ˇ				6		

* n is the number of keystrokes required to type in the specified domain name.

** sel is the number of operations required to select a rule or cookie from a scrolling list.

7.1 Global Cookie Policy

The default policy for most browsers is to silently accept all cookies, with only Konqueror, Mozilla, OmniWeb, and IE6 as the exceptions. Konqueror is the most conservative, prompting every time by default. OmniWeb accepts cookies only for the current session. Mozilla and IE6 break new ground by making cookie decisions based on the P3P policy of the originating site. However, the use of P3P significantly increases the complexity of the cookie policy.

In particular, the user interface for Internet Explorer 6 actually doesn't provide enough information for a user to determine what the default cookie policy does. Some more details about it can be obtained by reading the help documentation, but even that is underspecified. A precise definition of the default policy could only be found on Microsoft's website [6]:

> The Medium privacy setting blocks third-party cookies that do not have a compact policy (a condensed computer-readable privacy statement) or third-party cookies that have a compact policy that specifies that personally identifiable information is used without your implicit consent. First-party cookies that have a compact policy that specifies that personally identifiable information is used without implicit consent are "downgraded" (deleted when you close Internet Explorer). First-party cookies that do not have a compact policy are "leashed" (restricted so that they can only be read in the first-party context). Cookies that had been stored on your computer before you installed Internet Explorer 6 are also leashed.

The global policy setting for Internet Explorer 6 also has the surprising problem that, even after a user moves the setting to "Block All Cookies", the browser will continue to silently transmit the cookies it has stored, according to a FAQ from Microsoft [7]:

> Unless the privacy preference is set to "Block All Cookies", changing the privacy preference setting in Internet Explorer 6 does not affect the cookie acceptance policy for cookies that are already set.

Internet Explorer on MacOS uniquely offers the option to prompt the user once per site instead of once per cookie. The browsers offering easiest access to the global policy settings are Opera, OmniWeb, Galeon, and Konqueror.

7.2 Cookie Arrival Prompts

When a cookie arrives and the browser has been set to prompt the user, the default option is to accept the cookie in all browsers but one. Only Konqueror sets rejection as the default button.

Opera offers the most options for dealing with a cookie, but OmniWeb and Konqueror tend to require fewer mouse clicks or keypresses while still offering many options. In particular, Konqueror's cookie arrival dialog is very efficient for keyboard use as compared with all the other browsers.

7.3 Automatic Filtering Rules

Almost all the browsers offer reasonably sophisticated filtering rules; only Internet Explorer 5 (on both Windows and MacOS) is weak in this regard. Konqueror and Opera provide slightly easier mouse operation, but the differences are slight. Galeon provides a shortcut to allowing or blocking sites with only 3 mouse clicks or 4 keypresses to allow or block cookies from the current site; it also offers the fastest access to view filtering rules.

Keyboard accessibility is conspicuously missing from the MacOS contenders; we discovered that some dialog elements are simply impossible to reach using only the keyboard.

7.4 Cookie Storage Management

Opera provides the fastest path to deleting all cookies with its unique "Delete Private Data" command, which offers to delete various kinds of cache, history, and cookies all at once. Galeon has the unique feature that the user can both remove a cookie from a particular site and block future cookies from that site with a single button.

Again, the MacOS browsers lack keyboard functionality. None of the MacOS browsers tested offered any ability to manage stored cookies from only the keyboard.

Konqueror and OmniWeb deserve special mention in that they display cookies in a tree-like hierarchy. This allows one to conveniently delete all the cookies in a particular domain just as easily as one can delete a single cookie. OmniWeb goes further and fully integrates the filtering rules and the stored cookies into a single unified view.

8 Conclusion

Konqueror, OmniWeb, and Opera probably offer the greatest power and flexibility for handling cookies. Of these, only Opera is available across all three of the popular platforms. OmniWeb's integrated cookie and filter display is much friendlier and easier to understand than many of the other browsers, which sometimes place the filter preferences and the stored-cookie management far apart in the interface. Konqueror, on the other hand, provides the most complete keyboard access to features.

Mozilla and IE6 offer P3P-based cookie control, but this introduces substantial additional complexity that may not be worth this feature.

It is important to mention that in some sense Mozilla is by far the most configurable of all the browsers, since its user interface is specified in XUL, an interpreted language that can be edited by users. Many people are already providing UI add-ons and customizations to suit their own needs. With the appropriate customizations Mozilla could probably beat all of the other browsers in terms of convenient cookie management – but then only a small fraction of users are likely to undertake such a customization effort.

References

1. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee. RFC 2068: Hypertext Transfer Protocol – HTTP/1.1. Proposed Standard, January 1997. Superseded by RFC 2616. Available at `ftp://ftp.rfc-editor.org/in-notes/rfc2068.txt`
2. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee. RFC 2616: Hypertext Transfer Protocol – HTTP/1.1. Draft Standard, June 1999. Available at `ftp://ftp.rfc-editor.org/in-notes/rfc2616.txt`
3. D. Kristol, L. Montulli. RFC 2109: HTTP State Management Mechanism. Proposed Standard, February 1997. Superseded by RFC 2965. Available at `ftp://ftp.rfc-editor.org/in-notes/rfc2109.txt`
4. D. Kristol, L. Montulli. RFC 2965: HTTP State Management Mechanism. Proposed Standard, October 2000. Available at `ftp://ftp.rfc-editor.org/in-notes/rfc2965.txt`
5. O. Lineham, A. Stephens. Cookie Monster: HTTP Cookie Bug Affecting Servers On Most Non-Generic Domains. Available at `http://homepages.paradise.net.nz/~glineham/cookiemonster.html`
6. Microsoft Corporation. Default Privacy Settings for Internet Explorer 6. Available at `http://support.microsoft.com/default.aspx?scid=KB;en-us;q293222`
7. Microsoft Corporation. Internet Explorer 6 Privacy Feature FAQ. Available at `http://msdn.microsoft.com/workshop/security/privacy/overview/privacyfaq.asp`
8. Netscape Communications Corporation. Persistent Client State: HTTP Cookies. Available at `http://wp.netscape.com/newsref/std/cookie_spec.html`