ANALYSIS OF PRIVACY AND SECURITY IN HTML5 WEB STORAGE*

William West and S. Monisha Pulimood
Department of Computer Science
The College of New Jersey
2000 Pennington Road, Ewing, NJ 08628
732-233-1624
west5@tcnj.edu

ABSTRACT

There is no doubt that the web has evolved from a simple media consumption device to an extremely complex programming platform over the past couple of decades. With the exponential growth of Internet use, web applications are becoming increasingly popular: they are easy to distribute, simple to update, and widely accessible. However, a uniform programming method for developing web applications does not currently exist. Developers must be experts in and juggle a combination of different languages in order to create fully functional web applications. W3C's introduction of HTML5 attempts to alleviate this problem [8]. Their Web Storage specification offers a method for storing client-side data as an alternative to the use of cookies in web applications. In this paper, the Web Storage specification is analyzed through an in-depth discussion of the privacy, security, and performance of current and future web technologies. The advantages and disadvantages of the localStorage and sessionStorage attributes are discussed, with special consideration given to their impact on privacy and security. Analysis is done in the context of a custom web application, offering a suggested framework for applications utilizing HTML5 Web Storage.

1. INTRODUCTION

HTML has been used as the standard method for developing web pages since it was first introduced in the early 1990s. Over the past twenty years, HTML, with the help of

---

JavaScript and CSS, has enabled the web to evolve from a simple document-sharing device into a complex and dynamic platform for application development. The introduction of JavaScript allowed web developers to construct intricate web applications with functionality rivaling that of modern desktop applications. However, most web applications require functionality beyond what JavaScript alone can provide; this forces developers to supplement these needs with server-side technologies such as PHP, Perl, or Ruby. This process is often tedious, as the programmer must develop different portions of the application in different languages. As hardware inevitably evolves to meet consumer requirements, web applications must grow more complex to compete with their desktop counterparts. To meet this growing need, HTML5 introduces methods for storing data that may allow future web applications to see vast improvements to performance, security, and privacy, without unnecessary supplemental languages.

1.1 HTML4

1.1.1 Cookies

Currently, the standard method of storing user data in web applications is through persistent and session cookies. Cookies are small text files stored on a user's computer and sent to a web server through HTTP messages to identify a user or save the application state. Cookies have been a primary source of controversy since they can be misused and lead to violations of Internet privacy. A recent study in The Wall Street Journal highlights several recent suits filed against websites misusing cookies for financial gain [6]. Although cookies may be set only by the visited domain, it is possible for a page to contain images or elements from other domains. Cookies set from external domains while retrieving such elements are called third party cookies. Usually found in advertisements, third party cookies can be used to track a user across multiple domains without explicit permission from the user. This data collected belongs exclusively to the server that holds it; thus the user has no control over the way their data is used. Domains may use data collected from one domain in conjunction with data collected on other domains to build extremely rich user profiles. Although these profiles are anonymous in terms of actually applying a unique name to a real-life person, such data is valuable and may be sold to data mining operations at the expense of innocent visitors. Additionally, since cookies are transmitted by HTTP requests, a packet-sniffing third party may intercept sensitive data. Without employing stricter security and privacy measures, cookies cannot be considered a completely secure or private method of data storage.

1.1.2 Browsers

Browser compatibility is a significant issue regarding HTML5. While the specification is still in the development stages, it becomes the decision of browser developers whether they should support new functionality or not. Some browsers may choose to not adopt a specific API due to an instability or ambiguity within the specification. Regardless, it is imperative that there is some real-world experimentation in the use of these new features to ensure that they are appropriate for use by developers upon standardization. The tension that ensues between browsers and developers is unfortunately inevitable when updating existing standards [4], and must be considered when deciding whether to implement these types of features in a real-world application.

Different compatibility issues exist with different browsers, and the lack of support for a storage feature among multiple browsers can pose serious security problems for developers wishing to use that feature. In addition, each browser may implement its own methods of maintaining a user's data, causing confusion among users who may use more than one different type of browser. Thus, it is useful to keep in mind a browser's level of compatibility when implementing the Web Storage API in an application.

1.1.3 Data Management

One of the primary motivations of using HTML5 Web Storage is to provide the user with more control over their data. In order to provide their users with the appropriate level of control, browsers should explicitly state how the user may view, modify, and delete their data through the browser. If the user is unsure how to fully erase their personal data, there is a strong possibility for security and privacy vulnerabilities from within the browser itself. In Mozilla Firefox, the browser groups web storage and cookies into the same category when the user chooses to delete their data [3]. Thus, when the user deletes their cookies, their storage data is also deleted. Google Chrome implements a similar function, but labels it "Delete cookies and other site data". Safari uses a much different implementation; instead of grouping the two types of storage together, they separate them completely. To delete their web storage, users must manually remove each domain's storage one by one through the main Security menu [5].

2. HTML5 STORAGE

Web Storage in HTML5 uses a Storage object that represents a list of key-value pairs, much like those implemented in cookies. The object allows for simple functionality, with methods including set, get, remove, and clear methods. The list of key-value pairs is typically implemented to be transparent to the user through the browser's data management system, much like cookies. However, it is imperative to note that this data is not transferred over HTTP messages; all processing and storing of data occurs through the use of client-side script. Data is suggested to be limited to an arbitrary 5MB per Storage object, but browsers may implement this however they like. By default, the user agent must acquire the storage mutex before accessing or modifying any data within the associated Storage object; however, "the use of the storage mutex to avoid race conditions is currently considered by certain implementers to be too high a performance burden, to the point where allowing data corruption is considered preferable" [8]. This portion of the specification will need to be fixed before standardization, as data corruption on a client-side application is not easily detected; cookies, on the other hand, have the advantage of constantly being checked by the server with each HTTP request.

2.1 Session Storage

Session storage can be used in a similar way to a session cookie, but with several performance improvements. Since the data in a user's session storage is stored by window rather than by browser, data 'leaking' is prevented. For example, if a user has two windows open for the same shopping cart transaction, it is possible that a purchase on one window will process a purchase in both windows due to the sharing of the cookie across

those two windows. A session storage attribute is meant to hold data only for the lifetime of the top level browsing context; that is, if a window or tab is closed, that data should be deleted by the browser. Therefore, session storage should only be used for single-session visits, such as shopping cart transactions. [8]

2.2 Local Storage

Local storage can be used in the same way that persistent cookies are used, but like session storage, carry some significant advantages over the use of cookies. First of all, the amount of storage is significant. With a "mostly arbitrary limit" of 5MB per origin, domains may store large amounts of key-value data; implementation uses are abundant, and the data limit can be flexible depending on the user-agent. This large amount of data can also be used to store things beyond traditional user-id strings; good implementations should make use of the fact that data need not be sent to the server. Thus, one example of a possible application would be to store user-edited data such as documents or files. Moreover, in conjunction with offline application functionality, it would be possible for a user to edit stored files through the web application while not connected to a network. However, if storing sensitive data, it may be wise to encrypt that data since it is transparent to the user. It could pose a possible security risk since any user with access to that browser could see web storage data [8].

2.3 Goals of HTML5 Web Storage

2.3.1 Security

One of the primary goals of the Web Storage API is to increase client-side storage security. The specification outlines strong suggestions for the way in which browsers may handle data. One of the most important sections suggests that when a storage item is accessed by a domain that is not the original domain that stored the item, browsers should throw an exception [8]. This essentially prevents malicious domains from accessing data that they should not have access to. There are also inherent advantages to storing data with the client rather than with the server. One being that a security breach of a web application does not necessarily mean that user data is compromised [1]; since data is stored with the client, the only point of access for that data is through the user's local machine. In addition, while cookies need to be constantly sent back to the server to preserve the correct state, localStorage does not. Client-side JavaScript can be used to constantly alter a user's key/value pairs without communicating with the server. This reduces the possibility of a packet sniffing attack due to the reduced need for continuous HTTP requests.

2.3.2 Privacy

Due to privacy concerns associated with the use of persistent and session cookies, the web storage specification offers client-side alternatives. Both the sessionStorage and localStorage attributes can be used as alternatives to cookies in order to provide the user with control over their data. One application that can make use of localStorage over persistent cookies would be the personalization of user-preferences. For example, if

www.example.com wanted to save a user's interface preferences for a future visit, the page may store a key/value pair containing: (theme, bw) to denote that the user prefers a black/white interface. Without using client-side storage, the server would need to create a cookie with a unique ID for the user and generate the user's interface based on data stored on the server database. This means that the server would constantly have access to that user's information. While our example may seem trivial, the same method could be used for any type of sensitive information as well.

2.3.3 Performance and Accessibility

The web storage specification also strives to improve the performance of web applications. Web applications that primarily run on the client rather than the server can greatly reduce server workload, reducing the need for expensive server equipment. Client-side applications may run without constantly contacting the server; thus, lack of bandwidth is not an issue. This may allow an application to be deployed in areas where Internet access is unreliable or nonexistent.

3. ANALYSIS

To enable a more detailed analysis of the Web Storage API, particularly testing the viability of the localStorage attribute, a budget management web application was designed and implemented using a combination of HTML, JavaScript, AJAX, and PHP. This simple application supports multiple users who may synchronously update a server-side XML database with their purchases. As purchases are reported, the server deducts the purchase amount from the current budget. The server always holds the current remaining budget for each section.

3.1 Local Storage

Our budget management system utilizes HTML5's localStorage attribute for storing all client-side data. At all times, a client using the application will have 6 key/value pairs in localStorage: id, rights, wages, props, costumes, and setmats. The application is run completely on the client until the user has finished making changes to the budget. When the data is ready to be sent to the server, the user commits the data and the page accesses a PHP script that copies all data in localStorage to the server XML file. This is the only point that data is processed over the server. All other operations take place through client-side JavaScript. Reading from the server is simple: the AJAX XMLHttpRequest() function is used to obtain a local copy of the XML file, read from it, update the localStorage data, and close the file. This framework for a client-server system can be applied to any application in which most interaction with the page may take place without contacting the server. With the help of client-side storage, data can be periodically saved to the browser while the user completes it. After data entry is complete, the information can be sent to the server in one step. Prior to HTML5, this functionality would have been provided through cookies, requiring continuous HTTP messages to be sent from the client to the server.

3.2 Synchronization

The methods of synchronization used in this application take into account the age of the data, so that at all times the XML file on the server has the most recent data. This is done through the use of a key/value pair of (id, [uniqueID]); in the case of this program, the unique ID is obtained through JavaScript's getTime() function, which returns the number of milliseconds since January 1st, 1970. This ensures that an ID acquired after a previous ID will always be greater in value. When a user visits the web page, the application will check if the user is a new user or not; this is done by ensuring that a key/value pair for 'id' exists in localStorage. If it does not, the application will pull the XML data from the server using AJAX, dynamically altering the page content without refreshing. If the user's browser does contain an 'id' key/value pair, the application will check if the server holds more recent data than the client; if it does, the application will fetch the data and dynamically alter the page with the new data.

3.3 Ease of Programming

The ability to use localStorage to set, retrieve, or delete data in one line makes programming a client-side application incredibly simple. Whereas programming a JavaScript application using cookies may take multiple lines of code with user-defined functions, all necessary functionality is contained within the Storage object. When used in combination with AJAX to dynamically fetch server-side content, it becomes possible to develop intricate web applications that run completely on the client. However, it should be noted that if at any time the web application needs to update any information on the server, the web application must use some type of server-side technology to do so. This can be seen in the budgeting application: the application required the client to update the server each time an update was submitted. Unfortunately, this functionality does not exist within AJAX, so there was a need to use PHP or some other server-side language to update the server data. Although the script file used was very small, it opens up the possibility for server-side security vulnerabilities; this could be avoided if there was some method of sending data to the server without using a server-side script.

3.4 Security

3.4.1 DNS Spoofing

It is possible for localStorage data associated with a domain to become compromised if an attacker uses DNS spoofing to appear as that domain. In the context of our budget application, an attacker could simply use his/her own domain and hacked DNS server to reroute all traffic from the victim to another domain appearing to be the web application. From here, the spoofed domain may acquire any data in localStorage. Deciding which keys to access would be trivial; the attacker could simply open a copy of the web application on his/her browser and view the localStorage keys through the browser's database viewer.

3.4.2 Multi-User Environments

Like with cookies, several security vulnerabilities arise when running a client-side application in a multi-user environment. Since domains store data by browser rather than by user, any other user who accesses that browser may access the data stored in that browser. This can present a security risk for sensitive data. This vulnerability can be mitigated by encrypting all sensitive data prior to storing it, rendering raw data useless without a decoder. However, if the user may access this data, he/she may also access the web application and make use of the data; this situation can be avoided by authenticating users before decoding data stored in localStorage.

3.5 Privacy

A serious privacy issue with cookies is a domain's ability to set and access cookies as a third party on another domain's web page. Eventually this can lead to a third party creating rich user profiles that track activity over multiple domains. W3C strongly suggests that browsers "restrict access to the localStorage objects to scripts originating at the domain of the top-level document of the browsing context" [8]. However, since a similar suggestion is offered in the cookie specification [2] is not followed by most common browsers, it is unclear whether browsers follow the suggestion offered in the localStorage specification. Both Firefox and Safari developer specifications do not make any statements concerning third-party storage concerns.

4. CONCLUSION

With the rapid evolution of web applications, it is necessary to constantly address modern security and privacy concerns through consistent updates of HTML specifications. It is evident that client-side storage technology has several advantages over sever-side solutions, including the ability to offer users greater control over their own data. As user privacy becomes vulnerable to complex methods of activity tracking, such as third-party cookies, it is important to address such vulnerabilities with client-side solutions. Not only can these solutions offer increases in performance, but also greater security and privacy for the user. However, some issues must be addressed before making HTML5 Web Storage a standard for all developers, including the concurrency issues inherent in the specification and browser implementation problems.

5. BIBILIOGRAPHY

[1] Hsu, F., Chen, H., Secure file system services for web 2.0 applications, Proceedings of the 2009 ACM workshop on Cloud computing security (CCSW '09), 11-18.

[2] Kristol, D., Montulli, L., HTTP State Management Mechanism, 2000, tools.ietf.org/html/rfc2965 , retrieved March 23, 2011.

[3] Mozilla Developer Network, DOM Storage, 2011, developer.mozilla.org/en/DOM/Storage , retrieved March 20, 2011.

[4] Pilgrim, M., HTML5 Up and Running, Sebastopol, CA: O'Reilly Media, Inc, Aug. 24, 2010.

[5] Safari Developer Library, Key-Value Storage, 2010, developer.apple.com/library/safari/#documentation/iPhone/Conceptual/SafariJSDatabaseGuide/Name-ValueStorage/Name-ValueStorage.html , retrieved March 20, 2011.

[6] Valentino-DeVries, J., Steel, E. 'Cookies' Cause Bitter Backlash. The Wall Street Journal. Sept. 20, 2010.

[7] World Wide Web Consortium, HTML5 Test Suite Conformance Results, 2011, w3c-test.org/html/tests/reporting/report.htm , retrieved March 22, 2011.

[8] World Wide Web Consortium, Web Storage Editor's Draft 13 April 2011, dev.w3.org/html5/webstorage/, retrieved April 1, 2011.