**Securing the HTML5 LocalStorage**

Karan Desai

Department of Computer Science

Rutgers University

**ABSTRACT**

Since HTML5 was introduced, its client side storage techniques have been talked about in the security community. In this project we will specifically deal with LocalStorage. With the boom in the mobile industry and ever increasing HTML5 demand, this is an issue that needs to be addressed but hasn’t been yet. Most modern browsers, desktop and mobile support HTML5 LocalStorage. We will first look at how LocalStorage is different than other client side storage techniques, then give some extended applications of LocalStorage and try to minimize the security risks associated with it. As it currently stands, LocalStorage is not made to store sensitive information and it is plain text and its uses are limited because of this. It has not substituted Cookies. HTML5 LocalStorage is quite vulnerable in situations where XSS attacks are possible. We will provide a JavaScript based solution that can be implemented by a host website and try to secure it. We will then challenge the use of JavaScript for cryptography and explain why we need something better. Finally we will develop a chrome browser extension for securing LocalStorage and look into possible future implementations that browsers can have built in. The goal of this project is to completely study LocalStorage and its usefulness if we can successfully tackle the security issues and provide practical solutions.

**1. INTRODUCTION**

This section briefly describes HTML5 and its client side storage technologies. It gives benefits of using local storage. Later it is compared to other types of client side storage technologies. We also look at websites implementing local storage and the usage of local storage by numbers.

**1.1 HTML5**

Html5 is the newest version of HTML. HTML5 will be the new standard for HTML. The previous version of HTML, HTML 4.01, came in 1999. HTML5 is designed to deliver almost everything someone wants to do online without requiring additional plugins and is cross-platform. Amazing web application can be very easily made and run on various kinds of devices. Every modern browser supports HTML5 and it is becoming very popular among developers. It enhances the standard of local storage, which has existed in some form or the other but never has it become so readily supported.

IE FIREFOX SAFARI CHROME OPERA IPHONE ANDROID

8.0+ 3.5+ 4.0+ 4.0+ 10.5+ 2.0+ 2.0+

*Table 1. Html5 local Storage browser support*

**1.2 HTML5 Client Side Storage ( Web Storage )**

HTML5 Web Storage defines two types of key-value storage types: sessionStorage and localStorage. The primary behavioral difference is how long the values persist and how they are shared. The differences are highlighted in the Table2 [1]. The differences highlight the various technical specifications of local storage. We are really interested in localStorage, wherein a developer can create a key value store for the application. This storage can be accessed from anywhere in the application. It can store 5MB per app per browser. It is also persistent and so unlike session storage exiting browsers or restarting operating systems does not affect it. Session storage is not really helpful for the kinds of applications that we trying to make a case for and so it is irrelevant.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Storage Type** | **Max Size** | **Persistence** | **Availability to other**  **Windows/tabs** | **Data Type Supported** |
| LocalStorage | 5MB per app per browser ( browsers decide the standard ) | On disk until deleted by user (delete cache) or by the app | Shared across every window and tab of one browser running same web app | String only, as key-value pairs |
| SessionStorage | Limited only by system memory | Survives only as long as its originating window or tab | Accessible only within the window or tab that created it | String only, as key-value pairs |

*Table 2. Comparison of localStorage and sessionStorage*

Table 3 highlights the main benefits of using HTML5 Local storage in web applications. The main points from it are the ease of use, persistency, offline capabilities and performance enhancement. We will later read in section 4 about application of these benefits and how they can prove really helpful in specific scenarios. Table 4 highlights the main function in the localStorage API. It also gives sample usage scripts.

1. It is very easy to implement as key value pairs
2. It uses strings for storage instead of complex databases (and one can store more complex data using JSON encoding)
3. It is well supported by browsers (desktop and mobile) and endorsed by companies
4. A lot of storage space (5 MB, much more than cookies)
5. On the client side, which means significantly speed up display times and less network traffic
6. Persists beyond a page refresh, browser restart and even system restart
7. No transition to the server helps security
8. Restore state upon app reentry and temporary state management for when the app looses network connectivity

*Table 3. Benefits of using LocalStorage*

|  |  |  |
| --- | --- | --- |
| **Method/Attribute** | **Returns** | **Usage** |
| setItem | -- | localStorage.setItem(“key”,”value”) |
| getItem | String value | localStorage.getItem(“key”) |
| removeItem | -- | localStorage.removeItem(“key”) |
| clear | -- | localStorage.clear() |
| key | String key | localStorage.key(i) |
| length | int length | localStorage.length() |

*Table 4. LocalStorage API Functions*

**1.3 Other Client Side Storage Technologies**

*WebSQL* Database [2] is a structured database with all the functionality and complexity of a typical SQL-powered relational database. It is more of a full fledged database. It has now been deprecated. Will not be supported on Internet explorer or Firefox, and will probably be phased out from the other browsers at some stage. *IndexedDB* [3] is another form of client side storage. It is quite robust and has various performance benefits. It is like a collection of object store. The major issue with it is that not all browsers support IndexedDB, especially mobile browsers do not support it. Chrome based *FileSystem API* is also an advanced standard that gives each domain a full hierarchical filesystem. It is a limited standard for now and it quite different from the other approaches that have been mentioned. It also does not have indexing or searching options as one can guess because of its file system type layout. Cookies can also be considered a type of client side storage but they are really limited by space. They are 4 KB’s each and 20 are allowed per domain.

**1.4 Websites Implementing Local Storage**

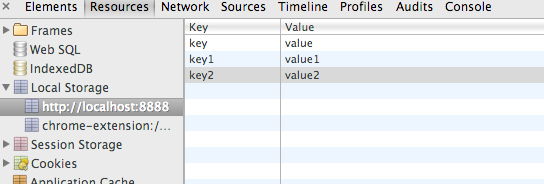
First consider this study “*Web Privacy Census by Berkeley Law - A study about privacy and tracking*” [4] . They do a study about privacy in turn analyzing websites for cookies, local storage, flash other types of tracking methods. They study top 100, 1000 and 25000 websites. The following is a statement from the privacy census: “*In 2011, when we first surveyed local storage, we found only 17 sites using HTML5. Our May 2012 crawl found 34, and now 38 sites are using HTML5 local storage*”(It is based on their survey of top 100 sites) One can imagine that it is a common technique now being used by a lot of websites and the number is continuously increasing. Their survey conclusion also mentioned the increasing use of local storage: “*In this first update to our original June 2012 Web Privacy Census, we observed statistically significant increases in the amount of tracking on all three of our samples--the top 100, 1,000, and top 25,000 websites. Flash cookies use is declining among the most popular websites, and HTML5 local storage is rising across all three groups*”. Now let’s look at two popular websites Facebook and YouTube and how they use local storage.

***Facebook*** mentions on it’s website why it uses local storage and why it is helpful to them. It clearly explains how effective local storage can be if used properly. The following is taken from Facebook directly [5] : We use local storage to understand and improve how our products and services perform and to enable certain features. For example, we may store certain parts of the Facebook website on your device so that those pages load faster the next time you visit them. Local storage also allows us to provide certain services to someone who doesn’t have access to the Internet. For example, you can read and compose messages in the Facebook Messenger app when you’re offline because we store those messages locally on your device. We may also use local storage to help us understand how you use websites, apps and computers or mobile devices. For example, we may use local storage on your device in connection with things like apps, where we cannot use cookies.

***YouTube*** uses local storage actively to store information like user clicks, user video history and preferences like user player volume. This gives a clear view on how such client dependent data that may not necessarily be required to go over the network and be stored on the server side can be easily stored using local storage.

**1.6 Viewing the Local Storage**

Viewing the local storage data is quite simple. One can simply go to the developer console in chrome and see the local storage data under resources. You can also fetch the local storage using your own commands in the browser console. In Mozilla you can use tools like Firebug and execute commands to fetch the local storage. Of course on the website you can simply use localStorage.getItem() to get any values from the local storage. Other tools like browser extensions can also be used to view the local storage. There are several of these available in the chrome store. These are quite helpful in not only viewing the data but also being able to delete and edit entries. Figure 1 shows the browser console being used to fetch the local storage for our custom application that has a few sample values.



*Figure 1 : viewing the local storage from the chrome developer console*

**2. SECURITY ISSUES WITH LOCAL STORAGE**

Local Storage by definition was not meant to be secure. LocalStorage is not a security risk given two conditions (1) Developers do not use it to store data that might be sensitive or in fact anything that might violate user privacy (2) The website is not vulnerable to XSS **.** Last year’s Blackhat presentation on HTML5 security [6] addressed this as an issue. It did not give any solutions. Research has been done highlighting the issues with the security of local storage [7]. None of them though have addressed the issue with a proper solution. The security of local storage is a chicken-egg problem. It will be later explained in section 3. This section describes the main reasons why local storage is insecure.

**2.1 Plain Text Data**

Local storage saves data unencrypted in string form in the regular browser cache. It is not secure storage. It should not be used for sensitive data, such as social security numbers, credit card numbers, logon credentials, and so forth. But it is not a doubt that people do actually use it for sessions and storing information like user addresses and phone numbers, which becomes a privacy issue in certain issues if information is leaked. So one issue is that all data is plain text. Note that this is not as big of an issue as it might look at first. The threat model for this is very limited. The browsers Same Origin Policy is rather strong and protects the local storage data between websites. Apart from the device owner and in fact any one using the browser no one can simply just view that data. The problem arises when you website is vulnerable to certain attacks. This is explained in the next section.

**2.2 Cross Site Scripting**

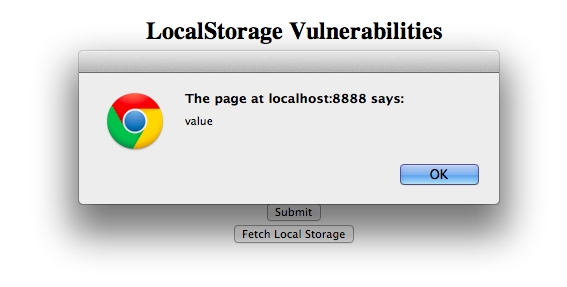
Local storage can be accessed through JavaScript and this allows an attacker to steal information, if the application is vulnerable to an XSS attack. XSS still remains a top of security vulnerabilities. WhiteHat’s report in May 2013 [8] states 43% vulnerabilities discovered are still XSS, which is a huge number (followed by information leak). In the OWASP Top 10 XSS made it to top 3 in most versions including the new 2013 version [9]. Facebook's XSS issues weren't resolved until 2008, and Twitter's weren't resolved until 2010. With new features and changes to such constantly evolving websites new types of attacks are often discovered. We will see how user XSS and alert the local storage information. Figure 2 shows simple code that can be used to extract the local storage data. The example shows simple alert box example but advanced scripts can be written to send the information somewhere or even do persistent XSS attacks which will keep sending the data over a log period of time once injected. This does not only apply to XSS that any attack that can inject a script. CSRF attacks can also make this possible.

*/\* Inject the following script in a form field or url \*/*

<script> alert(localStorage.getItem(“key”); </script>

*/\* Instead of alert you can send it somewhere \*/*

*/\* An alert box should pop up with the value from the local storage \*/*



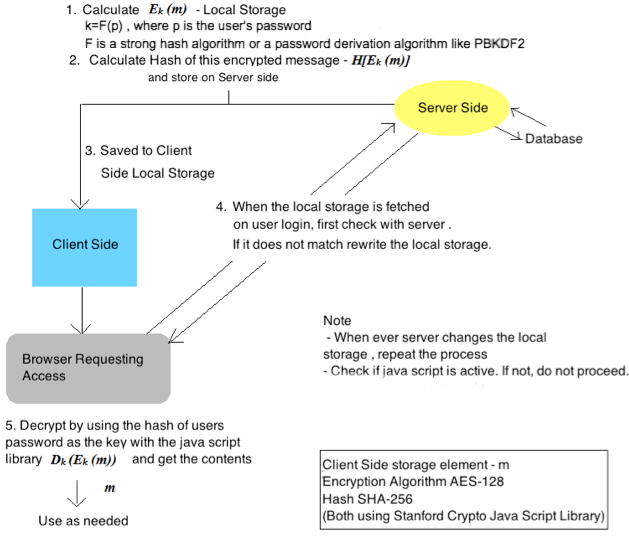
*Figure 2. Extract LocalStorage Data using XSS*

***3.* SECURING LOCALSTORAGE**

We are going to provide a couple of solutions on how to secure local storage. Securing the local storage has three major steps. (1) First encrypting all plain text (2) Second, proper management and generation of client side keys for decryption and (3) third being able to prevent the edit/delete operations or at least recovering from such attacks. The first solution is a server-based solution and second is a browser extension based solution. There could be a third solution which can be a hybrid but it has too many variables and too complex for the scope. It basically kills all the easiness of local storage. All currently proposed solutions use JavaScript based solutions and we will later argue the unreliability about such solutions.

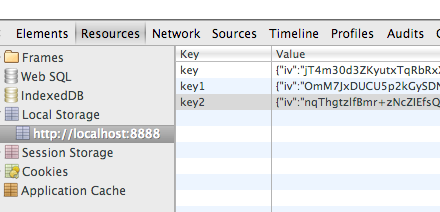
**3.1 Intelligent Server**

We are going to secure LocalStorage by strong encryption enforced by the server. The server over here controls the whole process. First, all data is encrypted from the server side before it reaches the client. The encryption key is of essence here and it is suggested that the key should be derived from the user’s password. One could use a simple hash of user’s password but stronger key derivation like PBKDF2[10] is strongly recommended (Password-Based Key Derivation Function 2). The server should maintain a copy of this entry and maintain a hash like SHA-256 of the complete local storage for authenticating the client storage later to make sure no changes were made to it. While using it locally the client should decrypt it by prompting the user for a password. This is not the ideal scenario. As one would notice the password management on the client side becomes an issue. Possible solutions are that one can store password somewhere other than the JavaScript code. A different technology like indexedDB or File System API can also be used. Another safer option is asking for the password over SSL. All these are good solutions but they really complicate the task. And if we had SSL in the first place then a lot of attacks could be mitigated anyways. Figure 3 gives a flow chart of the steps involved. Figure 4 shows how the local storage looks in the browser console after server side encryption.



*Figure 3. Intelligent Server - Process*

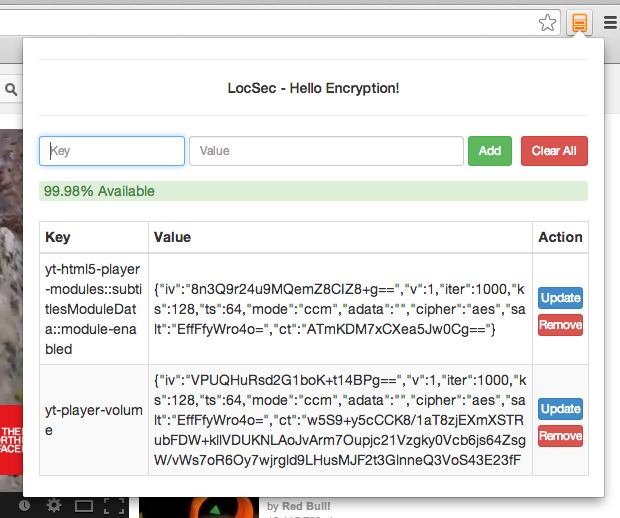
The benefits of this solution are (1) It is actually quite secure (2) Any non-server edits and deletes are detectable and reversible (3) Local Storage has now become a glorified cookie (4) Whatever happens on the client side the data is secured (5) No XSS can get the decrypted data without the user’s knowledge. The main drawbacks of this solution are (1) Too much server involvement (2) Sacrificing ease of use by making user enter his password (3) Requires a lot of changes to server code and cannot be generalized (4) It limits applications of local storage. The Stanford JavaScript Crypto Library (SJCL)[11] was used for JavaScript based encryption-decryption.



*Figure 4.Local Storage after server side encryption.*

**3.2 Browser Extension (Chrome, LocSec)**

We develop a new solution for encrypting and decrypting the local storage that it completely server independent. We will develop a chrome extension for this task of encrypting, decrypting and hashing. It is limited to chrome as we are developing a chrome extension. The reason that we chose a browser extension is that it can be distributed openly and it works independently. We run the encryption and decryption in the backend so that the user runs the application seamlessly and is not affected by this change. What this extension does is it overrides the inbuilt localStorage.getItem() and localStorage.setItem() functions and it does the encryption before writing and decryption before reading. Note that this is not a permitted action for chrome extensions. We do some clever coding which is able to achieve this task. We inject a new JS file to the websites code. But this alone does not help. The main thing is when and for whom to decrypt. Which is why we use the websites cookie as a key to encrypt the data. The cookie must have the HttpOnly attribute. So now even if the website is vulnerable to XSS all basic kinds of attacks will fail because the XSS would fail to get full cookies which are protected by HttpOnly. The use of cookies as keys is the real idea here. If the attacker gets the cookie it can mostly login as the user itself and the whole threat scenario changes anyway. The browser prevents edit/delete of data by maintaining the hash as discussed earlier. It maintains a copy of local storage to restore any illegal changes to the local storage. The main achievements from this are (1) It is completely independent of implementation and will work for most websites (2) Prevents editing/deletion (3) No changes on the server side are needed. The main drawbacks of this solution are (1) Everything depends on the security of the cookie (2) This will not protect you from the very skilled attacker (3) though it is not a big issue, the user and other browser extensions can still see the plain text data. To make some changes to this implementation we learned something from the implementation *Privilege Separation in HTML5 Applications* [12]*.* The authors propose a very good idea about separating parts of the application into parent/children, where the children are unprivileged. Everything happens by messaging with the parent, which decides whether to allow action based on a security policy. But this does not exactly apply in our case. The parents and children need to share the local storage and so mainstream privilege separation in not a good idea for us. We learned from this implementation and came up with a *fake privilege separation*. In the extension implementation we confuse the JavaScript scope such that it only decrypts for certain parts of the implementation. The first main point is that it specifies when to override the localStorage functions. It also makes sure the user console does not show plain text data as such. Other extensions can also not see the decrypted data. This is of essence given the security issues with extensions and the rise in the number of extension. It also prevents simple attacks where the attacker attaches scripts to raw HTML data. The only requirement is that the developers keep script and HTML files separate which is anyways a suggested and secure method of development. Figure 5 below shows YouTube’s local storage would appear in the extension.



*Figure 5. YouTube’s localStorage after implementing the extension. Only the encrypted text is visible.*

**3.3 JavaScript Cryptosystems**

The solutions that we gave have something in common: JavaScript cryptosystems. Relying on JavaScript based libraries for this task is completely debatable. We used one the most popular libraries SJCL – Stanford JavaScript Crypto Library. JavaScript isn't a serious crypto research environment, and suffers for it. The developers of this library have mentioned *“Unfortunately, this is not as great as in desktop applications because it is not feasible to completely protect against code injection, malicious servers and side-channel attacks”*. Currently, there is no way to generate keys or other entropy safely in a browser, as any element on the page (delivered from the server) can override built-in functions used to seed the PRNG. Our implementations in the two solutions suffer from these challenges too. You can you use native code to deliver proper random number but JavaScript will never know if these numbers are actually random or have been overridden. One can argue about using SSL getting random numbers from Random.org over a secure channel HTTPS, but then you have real cryptography and you do not need JavaScript cryptosystems. The problem is entropy. You can't safely generate or store keys. There are very simple attacks to exploit any kind of similar cryptosystems. The project was started for securing local storage but I ended up realizing the state of JavaScript based crypto. The main conclusion here is that we can try as hard as we want the browser has to take an initiative of securing the local storage or provide proper API’s for generating or storing keys. This would largely solve the problem of a lot of other related problems.

**4. APPLICATIONS OF LOCAL STORAGE**

The possible applications for LocalStorage are the reason that we are trying to secure it. It has a tremendous potential to change the way we look at client side storage in the industry that is now moving towards toward a very server and a client independent scenario. We have already seen examples of how Facebook and YouTube use it. Lets look at some the possible applications and some that are already in place.

1. *Permanently bound sessions* – Because of persistency, this can be of a huge advantage. And this allows sessions even without user registrations or user logins. Which means we application that don't force user login for interactivity and via local storage it is able to customize application by unique identification.
2. *Store sensitive information* – This has always remained a major concerns for all kinds of applications and we see privacy issues regarding the same every time. Even bigger companies are not good at protecting your data on the server side. With a good working solution to securing local storage storing phone numbers, contact data , addresses , form field values , etc on the client storage would be really useful. Note that these would bind to devices and you may need to have technologies where the user can choose to be copied to other devices by proper means.
3. *Draft Management* – Many times a user is writing forum comments, Facebook updates or tweets and he or she is interrupted and this leads to a loss of such temporal data. Local storage can be very handy for this task. Whenever a user visits the site again, his drafted text will be ready to post.
4. *Detailed user history and interests* – User browsing history for the applications, his interests and frequent clicks can all be maintained in localStorage. This saves the hustle of storing so much data on the server side. It also promotes user privacy.
5. *Temporary State* – The application can save using this technology. If a user was doing some work and looses network connectivity all changes can be saved to local Storage and committed once it is back online. The user does not realize the seamless transition.
6. *Maps and Location* – It can help store user geo-location. While using applications like Google maps it can help loading map tiles quicker.
7. *Cache data* – it can help in caching any kind of data. Specifically used in API or AJAX results.

**5. REFERENCES**

1. GWT documentation on HTML5 LocalStorage

http://www.gwtproject.org/doc

1. WebSql Database Documentation on W3Schools

http://www.w3.org/TR/webdatabase/

1. IndexedDB documentation on W3Schools

http://www.w3.org/TR/IndexedDB/

1. Chris Jay Hoofnagle & Nathan Good, The Web Privacy Census , October 2012, available at  http://law.berkeley.edu/privacycensus.htm
2. Facebook Local Storage

https://www.facebook.com/help/330525367015217

1. HTML5 Top 10 Threats Stealth Attacks and Silent Exploits, Shreeraj Shah, BlackHat 2012
2. Browser Web Storage Vulnerability Investigation:HTML5 localStorage Object, Daniel Bogaard, Daryl Johnson, and Robert Parody, RIT
3. Whitehat Vulnerability Survey

https://www.whitehatsec.com/resource/stats.html

1. OWASP Top 10 Security Vulnerabilities 2013

https://www.owasp.org/index.php/Top\_10\_2013RSA

1. RSA Specification on PBKDF2

http://www.ietf.org/rfc/rfc2898.txt

1. Standford crypto JavaScript library , Stanford University

http://crypto.stanford.edu/sjcl/

1. “Privilege Separation in HTML5 Applications“, Devdatta Akhawe, Prateek Saxena, Dawn Song University of California, Berkeley