2011

Browser web storage vulnerability investigation: HTML5 localStorage object

Daryl Johnson

Browser Web Storage Vulnerability Investigation

HTML5 localStorage Object

Authors Name/s per 1st Affiliation *(Author)*
line 1 (of *Affiliation*): dept. name of organization
line 2: name of organization, acronyms acceptable
line 3: City, Country
line 4: e-mail: name@xyz.com

Authors Name/s per 2nd Affiliation *(Author)*
line 1 (of *Affiliation*): dept. name of organization
line 2: name of organization, acronyms acceptable
line 3: City, Country
line 4: e-mail: name@xyz.com

***Abstract*— With the introduction of HTML5, the latest browser language, a new data storage technique, called localStorage, has been added to allow websites to store larger amounts of data for a long period of time on the user's local system. This new technology does not (as of this writing) have a fully implemented independent interface to support end user control. Unlike cookies, there is not yet an interface for the user to block, alter or delete localStorage in web browsers.**

Nefarious users have files of data they utilize in their illegal activities that they need to preserve (stolen user information, credit card numbers, etc.). These users do not want to have a copy of this data on their personal machines in case of an investigation. Therefore, nefarious users are constantly looking for a new method to preserve and store this data, concealing it in such a way that it won't be associated with them but available when needed.

Our project is to model this process by building a web application that would take a file, encrypt it, slice it up into 26 parts and distribute it to as many client systems as possible. At a later time, a second web application would watch for return visits by the holders of the parts of the original file and retrieve the parts as clients interact with the website. We would be studying the recidivism rate of clients returning to the website and the number of copies of each part distributed necessary to achieve a reliable recovery rate of the whole file.

We will first test this prototype in a controlled laboratory setting to ensure that it works as intended. Next we have chosen two websites, the XXXX(http://XXX.XXX.edu/) and XXX(http://XXX.XXX.edu/) departmental websites, as a test bed. We have secured permission from the chairs of these departments to utilize these resources. These sites were chosen primarily because their viewers are adult learners and because of their high traffic patterns.

Keywords-component; localStorage, HTML5, evasion, forensics, obfuscation)

I. INTRODUCTION

Suppose a nefarious user has a file of incriminating material (credit card number, account number, username/password or Personally Identifiable Information, drug client list…) that the user does not want to be apprehended with but needs access to from time to time. The users goal would be to store the file somewhere that can be reliably retrieved but does not reside locally (for very long) and is not usable or discernable for what it is if found where stored.

The authors propose a solution– HTML5's Web Storage or localStorage. If the nefarious user has access to a domain (simple Internet Service Provider will suffice) they could hide parts of any incriminating file on various client systems without keeping a local copy that he/she might be caught with. At a later time, when the information is needed, the user could get the parts back from the clients and reconstitute the original data.

To explore this scenario, the authors have split the experiment into 3 parts. The first part (testing phase) of this study has been completed. We have built a web application that proves the hypothesis that localStorage can be used for such a purpose. The second part of this study is to install the application on a working production site and statistically determine how many copies of the parts need to be disseminated in order to ensure retrieval – both over the short term and long term (would there be a difference between trying to get the data back in 10 days versus 90 days?). Potentially the effects of the choice of the number of segments to divide the original file could be studied. The third part of the study will look at possible detection characteristics for this sort of behavior and the development of tools and techniques for defense.

II. PROBLEM EXAMINED

The illicit users have the same needs for information management and security that the rest of the world has if not greater. Often their stakes are even bigger. We can break the needs into two classes. The first class would be one shared by all digital users, CIA or Confidentiality, Integrity and Availability[1], and the second would be one that is not so common, evasion. Each of these issues is addressed in the proposed solution.

Confidentiality is the limiting of access to data to authorized or intended users. The data in this case is encrypted and then segmented into many sections. The sections are then separated, encrypted and dispersed to disassociated unaware clients. If any piece or subset of the collection is discovered and reassembled it is unusable.

Integrity is knowing if the data is trustworthy or in this case did we get all of the pieces and reassemble it correctly. In this proposed solution, the individual pieces have a checksum or digest calculated and appended to the end before delivery to the client systems. Upon retrieval the checksum is recalculated and verified to ensure that the chunk of data has returned intact. Once the pieces have been reassembled, the original message is decrypted. A final checksum for the entire original message is verified assuring that the message has been retrieved intact.

Availability is being able to access the data when and where needed. In this situation the concept of availability relates to the reliability of future access to the data. This is currently being studied as phase two of this project. The trade off is speed of access for deny-ability or "it's not on my drive!" The file is available to the owner with an access time of hours, days or months depending on many factors. The benefit is that the file is unavailable to anyone else.

The last issue is evasion. Evasion is an act of subterfuge, avoiding or eluding detection. The idea here is to hide the data from an examination of the local system. Once the pieces are distributed, the local system and web database can be forensically cleaned and all evidence of the data eradicated. Even if it were suspected that the web clients might be involved, a moderately trafficked web site could have hundreds, thousands or even millions of individual clients to investigate. Add that the clients are not owned by the nefarious user being investigated and you have a huge jurisdiction problem investigating any potential involvement of the clients.

III. HTML5 AND WEB STORAGE

With the advent of HTML5 and its subsequent adoption in all modern web browsers (to varying degrees - http://html5test.com/), programming for a browser based internet experience recently turned to the better. HTML as a standard has been around since 1990 and was standardized as HTML 4 in 1997. HTML5 is still under development (as of November 2011) and is meant to subsume not only HTML4, but XHTML1 and DOM2 HTML (JavaScript) as well[2].

Some of the advantages of HTML5 (ubiquitous coding APIs, numerous new media types, embedded semantic meanings) while a boon to both developers and users alike, are outside of the scope of this paper. The area of the HTML5 improvements that we are planning on exploiting is the advanced data storage, or Web Storage[3]. Many developers may think that web storage includes cookies, various browser dependent client side databases, as well as storage objects. However, by the specification, the term Web Storage is limited to the storage objects – specifically localStorage and sessionStorage.

Since Web Storage includes both localStorage and sessionStorage, we needed to consider both. Upon a quick examination we found that sessionStorage matched its name – it is storage that exists solely for a browser session (sessions expires when the browser shuts down and the data is cleared). Because sessionStorage is implemented effectively, it is of little use to the user for our purpose. localStorage, on the other hand, works perfectly for what is needed. From a developer's point of view, localStorage is an associative array or hash – a name=value pair that can hold any textual content.

To understand the need for a localStorage object, a little history is needed. Since the inception of the HTTP protocol, it has been stateless and anonymous, so a mechanism had to be created to make the tracking of state possible. The 'HTTP State Management Mechanism' proposal was created to fill this void[4]. The outcome of which is commonly known as cookies. The cookie mechanism is a name value pair that is served up from the client to the server inside of the HTTP Request phase (based upon various criteria: path on the server, domain to be served to, protocol to be served up to – http or https). While cookies have been used in various ways through the history of the web, more often than not they are used to hold a session identifier or token – one created by the framework the server is implementing (.Net, PHP, JSP) or one created by hand by the developer.

Historically, cookies were the sole means web browsers had for long-term storage capabilities. They had limited length (4096 bytes) and a limited number could be written per domain (20) for a total of 81,920 bytes of storage space. Today, localStorage, as a storage mechanism, is limited to 5Mb per origin (domain)[5], or 655,360 bytes of storage (8 times larger). If the browser manufacturers maintain this suggestion of the specification (currently IE9 allows more - 10Mb per origin), the possibility of using the various client's hard drives for other kinds of storage is worth looking at.

As often happens with newer technologies, they are implemented before they are fully tested. localStorage works flawlessly in the modern browsers, but the tools that the end user has to allow, view, update or delete them is very limited. Combining the amount of storage space with a lack of user control, a nefarious user would only be encouraged to use it for ill. At the time of this paper, there is no specific user interface for localStorage. If a user wants to find out what is stored on their various browsers there is no easy way. An advanced user would have to visit the domain they are interested in and then run a bit of code to see if they had any localStorage recorded.

```
for (i=0; i<localStorage.length; i++) {
  key = localStorage.key(i);
  pairs += "key:"+key+" value:"+localStorage.getItem(key);
}
console.log(pairs);
```

Figure 1. Script for testing what is stored in localStorage.

Adding to the problem of knowing if your localStorage is being used (it is effectively an invisible attack vector), there is no clear way to turn it off. Additionally, once it is written it doesn't have an easy affordance to remove the data. For Firefox, DOM Storage (Firefox's moniker for Web Storage) can be cleared via "Tools -> Clear Recent History -> Cookies" ONLY when the range is "Everything"[6]. There are multiple problems with this interface, but the top are:

- User has to know that all DOM Storage is under the header 'Cookies'

- If a user solves the labeling problem they might only want to clear recent localStorage and not select "Everything"

For Internet Explorer, the story is similar:

"…users can clear storage areas at any time by selecting **Delete Browsing History** from the **Tools** menu in Internet Explorer, selecting the **Cookies** check box, and clicking **OK**. This clears session and local storage areas for all domains that are not in the Favorites folder and resets the storage quotas in the registry. Clear the **Preserve Favorite Site Data** check box to delete all storage areas, regardless of source"[7].

Clearly, a more specific interface is needed. While it might not be necessary to split localStorage out from other data storage capabilities, listing it under Cookies may not be intuitive for average users. Also, the ability to clear stored data in a more chronologically granular way would be useful.

IV. Problem Exploited

To exploit this possible weakness, the authors devised a web application that would take any textual file, calculate and attach a checksum, encrypt it, split it into a some number of parts (26 in our testing), give each part an identifier (both for the part of the whole and an identifier for which file it came from), calculate a checksum for the part and append it to the string then re-encrypt it. We found that from this formula we could hide the parts on different clients and on subsequent visits we could get the parts back and reconstitute our original data. Should a non-textual file be the target, a simple binary to text translation tools such as base64 or uuencode would suffice.

A. Web Environment

For the implementation of our web application, we chose the open source LAMP architecture. LAMP is an acronym for Linux, Apache HTTP Server, MySQL database, and PHP server-side scripting environment.

B. Web Software

From a top-level view, the implementation of our application via web browsers consists of an interface to take a textual file and use the system described above to split it and populate a database. When we are ready to populate the visitors to our site with the parts of our file, we introduce a small client-side script that communicates covertly (via AJAX - Asynchronous JavaScript and XML) with a server-side script. The result of the server-side script is recorded in the client's localStorage. Once the nefarious user decides there are enough copies distributed for his purposes, he can wipe out his file AND the database.

Some time later, when we wanted to reconstitute our data we inserted a different client-side script that checks return visit clients for our data. If any data was found, be it a piece we hadn't gotten back yet or one we already had, we decrypted it, checked the checksum and stored it. After a period of time, we will have the entire file back.

For a deeper explanation, there are 2 sets of scripts to get this to work. One set is used to distribute the parts out to various clients and the other set is to retrieve it back. Each set has both a client and a server script used to accesses the database for storage or retrieval as needed.

The first small client-side script (7 lines of well formatted code used to proliferate our data out into the net) can be included on any html page (in our case we employed PHP with a MySQL database or a LAMP). In it we test if localStorage is implemented on the particular browser and if the browser doesn't already have a copy of a part from our domain a jQuery AJAX call is triggered to the server for a part that has been distributed the least amount. That part is then written to the browsers localStorage under a commonly used token name (we used 'uid') to help hide our data and intentions.

```
if(window.localStorage) {
  if(localStorage.getItem('uid')==null){
    $.getJSON('localStorageSet.php',function(data){
      localStorage.setItem("uid",data.uid);
    });
  }
}
```

Figure 2. JavaScript to retrieve a piece of the file from server and assign to localStorage.

The localStorageSet.php file that the AJAX call is hitting goes into the database of encrypted parts, finds the part that has been least copied to browsers and sends it back to the client to be injected into the localStorage of 'uid'. While it is doing this it also updates the total disseminated count on the part that it just served up and logs the visit.

Once we are confident that we have populated a sufficiently large enough number of targets, we felt free to shred our original nefarious file and the database table holding the parts. For the truly paranoid a forensic wipe of the drive and the user would be worry free on being searched.

The second small client script (3 lines of well formatted code) can be employed at a later date, when we want to reconstitute our data. For this we also used a jQuery AJAX call to send the contents of the specific localStorage data we want back to the server.

```
$.post('localStorageBack.php',{
  d:localStorage.getItem('uid')
});
```

Figure 3. JavaScript to send data back to server for retreval.

The data this sends back to the server is decrypted, checksum is checked and split into our original encryption, part and file identification. The data is then populated in a database table by its part identifier and filename (all data was logged for future references). Once all of the parts are recovered, the entire file is reconstituted, decrypted to our original state and it's checksum verified.

V. PROOF OF CONCEPT TESTING ENVIRONMENT

The laboratory proof of concept testing environment is simple and easily duplicated. VMware Workstation 7.1.0 was the foundation for the test environment installed on a Lenovo T61p laptop with 6Gb of memory. The target web server was a stock BackTrack5 virtual machine image[8]. Apache 2.2.14, MySQL 14.14 and PHP 5.3.2 were used to support the testing environment on the server.

A. Configuring the Web Server

The server application used was the default install that came with BackTrack5. The only addition to this was an installation of phpMyAdmin, an open source tool for easier database access (http://www.phpmyadmin.net/). Starting Apache and MySQL was all that was necessary (no specialized settings like .httaccess was needed).

In the testing environment, there was no reason to hide what we were attempting – so we had 2 separate html files, one to set the localStorage, setData.php and one to get the localStorage back, getData.php. setData.php had the client-side code that executed the AJAX call (figure 2). The AJAX call triggered the server side localStorageSet.php to get the least distributed part of the file and send it back in JSON (JavaScript Object Notation) format.

getData.php had the client-side code that used AJAX to send the contents of the localStorage.getItem('uid') (figure 3). The server-side code this one executed, localStorageBack.php decrypts the data and checks the checksum. If the checksum was good the data is stored.

In both cases, localStorageSet.php and localStorageBack.php we logged all calls and recorded the pertinent information into our own log for future study.

B. The client setup

To do the work of the Internet client population at large, additional virtual machines were employed. For the initial test, a Windows XPpro base image was constructed with no service packs installed. This was not a necessary insecurity but established a baseline. A stock install of Firefox 4.0.1 was done with no add-ons. No special configuration of Firefox was performed. Two scripts were added to the C:\ directory of this initial configuration to aid in the automation of the test case: setData.bat and getData.bat.

```
REM Cause Firefox to place data in localStorage
taskkill /F /IM firefox.exe
ping –n 10 127.0.0.1
start /B "C:\Program Files\Mozilla Firefox\firefox.exe"
    http://192.168.77.134/evasion/setData.php
```

Figure 4. setData.bat script for Windows XPpro client.

```
REM Cause Firefox to retrieve data from localStorage
taskkill /F /IM firefox.exe
ping –n 10 127.0.0.1
start /B "C:\Program Files\Mozilla Firefox\firefox.exe"
    http://192.168.77.134/evasion/getData.php
```

Figure 5. getData.bat script for Windows XPpro client.

First, the scripts make sure that the browser is not still running by executing a taskkill. This was necessary to make sure that localStorage was not preserved only during a single browser session. By terminating Firefox the session is stopped. Originally the browser was started first followed by a delay for the localStorage access to take place and then terminating the browser. This was routinely unsuccessful. By reversing the order, the browser was offered more time to complete the exchange. The Windows command shell does not have a delay tool. The ping command is a mechanism for a controlled wait with the count parameter taking one second per count after the first.

C. Assembling the masses

Once the Windows XPpro client is prepared, it is shut down and only used as a master for cloning. The algorithm requires at least 26 clients to hold all of the pieces of the message. The following scripts automated the process of construction utilizing VMware's vmrun tool[9]. The tool can issue instructions to several of VMwares virtualization tools including Workstation. The following script creates 26 clones of the master Windows XPpro virtual machine.

```
REM Make Clones of WinXPpro client system

set VMRUN="C:\Program Files (x86)\VMware\VMware
    VIX\vmrun.exe"
set SRCVM="C:\LocalStorage\Masters\WinXPpro\win2000Pro.vmx"
set CLONEBASE=C:\LocalStorage\CLONES\WXP


for /L %%i IN (101 1 126) do (
  %VMRUN% -T ws clone %SRCVM%
    %CLONEBASE%%%i\WXP%%i.vmx linked

  %VMRUN% -T ws start %CLONEBASE%%%i\WXP%%i.vmx gui

REM needs 60+ seconds to get to login screen
  timeout -T 60 /NOBREAK >NUL

  %VMRUN% -T ws suspend %CLONEBASE%%%i\WXP%%i.vmx
    hard
)
```

Figure 6. MakeClonesWXP.bat script for Windows XPpro clients.

Vmrun is utilized to instruct VMware Workstation to clone the base Windows XPpro virtual machine 26 times. After starting the VM a delay of 60 seconds allows the client to fully boot before the client is suspended. Suspending allows for a faster cycle time for client visits to the web site.

D. Occupy localStorage

The next phase of the test is to have each of the 26 Windows XPpro clients start a browser, surf to the web server and run the code to cause data to be deposited in their localStorage area. It is important for the browser to be started and stopped to assure that localStorage has persistence beyond the current session. The following scripts are run on the host of the virtual machines to first set or download the data chunk to the client and second to get or retrieve the chunk from the client.

```
REM make WXP clients visit web server to download data
set VMRUN="C:\Program Files (x86)\VMware\VMware
    VIX\vmrun.exe"
```

```
set CLONEBASE=C:\LocalStorage\CLONES\WXP
set FIREFOX="C:\Program Files\Mozilla Firefox\firefox.exe"

for /L %%i IN (101 1 126) do (
  %VMRUN% -T ws start %CLONEBASE%%%i\WXP%%i.vmx

  %VMRUN% -T ws -gu dgj -gp "ATest4LocalStorage!"
     runScriptInGuest %CLONEBASE%%%i\WXP%%i.vmx -nowait
     "" "cmd.exe /k C:\setData.bat

  timeout -T 60 /NOBREAK >NUL

  %VMRUN% -T ws suspend %CLONEBASE%%%i\WXP%%i.vmx
     hard
)
```

Figure 7. MakeVisitsWXP-setData.bat script for Windows XPpro clients.

```
REM make WXP clients visit web server to download data
set VMRUN="C:\Program Files (x86)\VMware\VMware
     VIX\vmrun.exe"
set CLONEBASE=C:\LocalStorage\CLONES\WXP
set FIREFOX="C:\Program Files\Mozilla Firefox\firefox.exe"

for /L %%i IN (101 1 126) do (
  %VMRUN% -T ws start %CLONEBASE%%%i\WXP%%i.vmx

  %VMRUN% -T ws -gu dgj -gp "ATest4LocalStorage!"
     runScriptInGuest %CLONEBASE%%%i\WXP%%i.vmx -nowait
     "" "cmd.exe /k C:\getData.bat

  timeout -T 60 /NOBREAK >NUL

  %VMRUN% -T ws suspend %CLONEBASE%%%i\WXP%%i.vmx
     hard
)
```

Figure 8. MakeVisitsWXP-getData.bat script for Windows XPpro clients.

The two scripts differ only in the target script that is run locally on the client system: setData.bat (see Figure 4) and getData.bat (see Figure 5). This structure is only necessary in this test environment to ensure that the browser is successfully started and stopped and that sufficient time is given to the client and browser to complete the operations. Typically the setData.bat script is run first followed by the getData.bat script. The set/get operation takes about an hour to complete. The entire environment starting from making the clones to retrieving the data set takes about 2 hours. The use of linked clones keeps the storage requirements down to under 40GB for entire environment.

VI. Phase 2 – Large Scale

Now that we have proven that we can hide and retrieve information in a client's localStorage in a controlled environment, our task is to discover how many copies of our encrypted and obfuscated parts we need to disseminate in order to ensure recovery.

In order to test this we have obtained permission to test our theories on the authors various departmental web presences (http://www.xx.xx.edu & http://www.xx.xx.edu). To make the results of this testing more accurate, we have decided to remove all visitors from the XXX.XX.x.x domain (XXX's domain) simply because many of our lab machines are forced to visit those sites on browser startup. Examining the logs for the past 2 months (September and October 2011) we have found that we can expect around 1 million original entries or 24,000 visitors who did multiple visits. From websites with this amount of traffic, one would assume that we could proliferate numerous copies in a matter of minutes. If we look at the numbers more closely we find that of clients who do multiple visits with 10 or greater days between the visits, the number shrinks to 8,000 – still statistically significant.

VII. Phase 3 - Detection

Once we have this working on a large scale, the authors are interested in studying the future application and usage of localStorage. The goal is looking for possible ways of monitoring and controlling localStorage activity, and identifying potential misuse. Intrusion Detection System tools such as Snort examine network traffic looking for digital signatures indicate that potential malicious activity is present. The development of signatures and other tools will be of primary interest during this phase.

VIII. Preventative Measures

The history of software interfaces is littered with examples of poorly designed and implemented user facing controls. The current state of the different browser interfaces to control Web Storage is lacking to say the least. The only preventative measure for not allowing something like this to happen on a client is to completely disable cookies. It should be noted that on all modern browsers there are different levels of cookie blocking (1st-party and 3rd party). However, since most trust the site they are visiting and 1st-party is what is being used, this number is relatively small. The number of visitors blocking 1st-party cookies varies greatly from one site to the next. Reports of 25% for sites about security and 1% for sites about general health are abundant. To know for sure one would need to test for their specific kind of site.. It should also be noted that once a localStorage value has been set, turning off cookies will not remove it, just make it inaccessible.

IX. Conclusions

The authors hope these findings motivate browser architects to realize what they are making possible with their implementations and web application developers to think about the attack vectors they are creating. The need for a new storage capability in web browsers is not in question. The need to have the storage be easy to use for both developers and users alike is not in question. Although it may be a good idea to often hide implementation details from users, not giving them simple and intuitive controls that provide the ability to at least see what is being stored on their machines is in question.

References

[1] M. Stamp, "Information Security: Principles and Practice", John Wiley & Sons, Inc., Hoboken, NJ, USA. doi: 10.1002/0471744190

[2] I. Hickson, World Wide Web Consortium, "HTML5, A vocabulary and associated APIs for HTML and XHTML, Editor's Draft." Last modified November 04, 2011. Accessed November 05, 2011. http://dev.w3.org/html5/spec/Overview.html.

[3] I. Hickson, World Wide Web Consortium, "Web Storage, Editor's Draft." Last modified October 04, 2011. Accessed November 05, 2011. http://dev.w3.org/html5/webstorage/.

[4] D. Kristol, and L. Montulli. Netscape Communications, "HTTP State Management Mechanism." Last modified February, 1997 . Accessed November 04, 2011. http://www.w3.org/Protocols/rfc2109/rfc2109.

[5] I. Hickson, World Wide Web Consortium, "Web Storage, Editor's Draft." Last modified October 04, 2011. Accessed November 05, 2011. http://dev.w3.org/html5/webstorage/#disk-space.

[6] Mozilla Developer Network, "DOM Storage." Last modified October 23, 2011. Accessed November 08, 2011. https://developer.mozilla.org/en/DOM/Storage.

[7] Microsoft Corporation, "Introduction to DOM Storage." Accessed November 08, 2011. http://msdn.microsoft.com/en-us/library/cc197062(v=VS.85).aspx.

[8] back|track-linux.org, "Downloads : BackTrack Linux – Penetration Testing Distribution." Accessed November 14, 2011. http://www.backtrack-linux.org/downloads/.

[9] VMware, "Using vmrun to Control Virtual Machines." Last modified 2009. Accessed November 14, 2011. www.vmware.com/pdf/vix162_vmrun_command.pdf.