7-2012

Browser Web Storage Vulnerability Investigation: HTML5 localStorage Object

Dan Bogaard
Rochester Institute of Technology

Daryl Johnson
Rochester Institute of Technology

Robert Parody
Rochester Institute of Technology

Browser Web Storage Vulnerability Investigation

HTML5 localStorage Object

Daniel Bogaard[1], **Daryl Johnson**[2], **and Robert Parody**[3]
[1]Information Technology, Rochester Institute of Technology, Rochester, NY, USA
[2]NSSA, Rochester Institute of Technology, Rochester, NY, USA
[3]CQAS, Rochester Institute of Technology, Rochester, NY, USA

Abstract - *Along with the introduction of HTML5 a new data storage technique, Web Storage, has been added to browsers. This technique stores larger amounts of data for an extended period of time on a client system. This technology does not (as of this writing) have a fully implemented interface to support end user control.*

The authors interest is modeling the use of Web Storage to store illicit data. The authors built a web application that would take a file, encrypt it, split it into multiple parts and distribute it to as many clients as possible. At a later time, the system could then watch for return visits and retrieve data parts as clients interact with a host website. The recidivism rate of clients returning to the host website and the number of copies of each distributed part needed to achieve a reliable recovery rate of the entire file are under study.

Keywords: browser security, Web Storage, evasion, forensics, obfuscation

1 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 – 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. A web application was built 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?). Phase 2 has been initiated and 67 days of data have been collected as of this writing and preliminary findings will be presented herein. Potentially, the effects of the choice of the number of segments to divide the original file could be studied, but for now they are held constant. 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.

2 Problem Examined

The illicit users have the same needs for information management and security that the rest of the world has, if not greater. The needs can be broken into two classes. The first class would be one shared by all digital users, Confidentiality, Integrity and Availability or CIA[1], and the second would be one that is not so common, evasion. Each of these classes 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, were all of the pieces retrieved and reassembled 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 deniability 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. Since the clients are not owned by the illicit user being investigated, possible jurisdiction problems arise investigating any potential involvement of the clients.

3 HTML5 and Web Storage

With the advent of HTML5 and its subsequent adoption in all modern web browsers (to varying degrees[2]), 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[3].

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 the scope of this paper. The area of the HTML5 improvements that the authors are planning on exploiting is the advanced data storage, or Web Storage[4]. While Web Storage is not directly part of HTML5, it has been repeatedly attributed to being part of HTML5 enough that most sources currently attribute it to HTML5. 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, both needed to be considered. Upon a quick examination it was found that sessionStorage matched its name – it is storage that exists solely for a browser session (sessions expire when the browser is closed and the data automatically 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[5]. 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). 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. Server frameworks (.Net, PHP, JSP) often implement these identifiers but occasionally they are 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[4]. Today, localStorage, as a storage mechanism, is limited to 5Mb per origin (domain)[6], or 655,360 bytes of storage (8 times larger). If the browser manufacturers maintain the size of the specification (currently IE9 allows more - 10Mb per origin), the possibility of using various client's hard drives for illicit storage becomes tempting.

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 (see TABLE 1). Combining the amount of storage space with a lack of user control makes this an invisible attach vector for illicit users to exploit. At the time of this paper, there is no unified 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);                                                    (1)
```

Adding to the problem of knowing if your localStorage is being used, there is no clear way for a common or average user to turn it off. Additionally, once it is written it doesn't have an easy affordance to remove or review the data. As an example of how this can be confusing, for Firefox's DOM Storage (Firefox's moniker for Web Storage) can be cleared via the menus "Tools -> Clear Recent History -> Cookies" ONLY when the range is "Everything"[7].

TABLE I. BROWSER COMPARISONS

Current Browsers	**Access to Web Storage**		
	Disable Storage	***Clear Storage***	***Examine Storage***

Current Browsers	Access to Web Storage		
	Disable Storage	*Clear Storage*	*Examine Storage*
Firefox 10.0.2	Yes, by turning cookies off in preferences(but does not clear old values) or in about:config	Select "Tools" » "Clear Recent History", open "Details", check "Cookies" and select "Everything" as time range.	Not without an external extension
Safari 5.1.2	No, with cookies turned off, localStorage is still set	Select "Safari" » "Reset Safari…" » Remove all website data	Have to go into preferences and turn on developer menu, then navigate to domain where it was set
Chrome 17	Yes, turn off cookies in preferences (but does not clear old values)	Select "Tools" » "Clear browsing data…", check "Delete cookies and other site data", select "Everything" from "Clear data from this period" and click on "Clear browsing data".	Developer Tools » Tools » Developer Tools - can see localStorage, but only for domain I'm currently visiting
Opera 11.61	Yes, opera:config Persistent»Storage turn off global quota (then have to turn it on on a per/domain basis)	Preferences » advanced » Storage (can delete one at a time)	Preferences » advanced » Storage See domain and size, not content
IE 9	Yes, internet options»Advanced»unclick Enable DOM Storage	Select "Tools"» "Internet Options" »"General" » check "Delete browsing history on exit", click on "Delete", check "Cookies" and click on "Delete" once more.	can see being set by running profiler in Developer Tools » Profiler

A more universal 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.

4 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. It was found that from this formula it was possible to hide the parts on different clients and on subsequent visits those parts could be retrieved and reconstituted into 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.

4.1 Web Environment

For the implementation of the web application, the authors chose the open source LAMP architecture for it's ubiquitous nature. LAMP is an acronym for Linux, Apache HTTP Server, MySQL database, and PHP server-side scripting environment.

4.2 Web Software

From a top-level view, the implementation of the application via web browsers consists of an interface to take a textual file and use the processed described above to split the file and insert the parts to a database. When the authors were ready to populate the parts to the visitors that come to our site, 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 stored in the client's localStorage. Once the illicit user decides there are enough copies distributed for his purposes, he can wipe out his file, the database AND all traces of the information.

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

For a deeper explanation, there are two sets of scripts that execute this process. One set is used to distribute the parts out to various clients and the other set is used to retrieve the data back. Each set has both a client and a server script used to access the database for storage or retrieval as is applicable.

The first small client-side script (2) can be injected into any html page. It tests if localStorage is implemented on the particular browser. Next, if the browser doesn't already have a piece of the text file from our domain, a jQuery AJAX call is triggered to the server for the part of the file that has been distributed to the fewest clients. The 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);
    });
  }
}                                                    (2)
```

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

Once the authors are confident that a sufficiently large enough number of targets have been populated, the original nefarious file and the database table holding the parts were destroyed. For the truly paranoid a forensic wipe of the drive and the user would be worry free of being searched.

The second small client script (Algorithm 3) can be employed at a later date, when the data is to be reconstituted. For this, a jQuery AJAX call is employed to send the contents of the specific localStorage data back to the server.

```
$.post(‘localStorageBack.php’,{
  d:localStorage.getItem(‘uid’)
});                                                    (3)
```

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 for future reference. Once all of the parts are recovered, the entire file is reconstituted, decrypted to the original state and the checksum verified.

5 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.

5.1 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 simple 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 was being attempted – so therefore two 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 (Algorithm 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’) (Algorithm 3). The server-side code this executed, localStorageBack.php, decrypts the data and checks the checksum. If the checksum was accurate the data was stored.

In both cases, localStorageSet.php and localStorageBack.php all calls were logged and recorded for future study.

5.2 The client setup

To emulate 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.

First, the scripts make sure that the browser is not still running by executing a taskkill. This was necessary to ensure that localStorage was not preserved for only a single browser session. By terminating Firefox the session was stopped.

5.3 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.

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

for /L %%i IN (101 1 126) do (
  %VMRUN% -T ws clone %SRCVM%
     %CLONE%%%i\WXP%%i.vmx linked
  %VMRUN% -T ws start %CLONE%%%i\WXP%%i.vmx gui
  timeout -T 60 /NOBREAK >NUL
  %VMRUN% -T ws suspend %CLONE%%%i\WXP%%i.vmx hard
)                                                              (4)
```

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.

5.4 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 the client’s 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.

```
set VMRUN="C:\Program Files (x86)\VMware\VMware
    VIX\vmrun.exe"
set CLONE=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 %CLONE%%%i\WXP%%i.vmx
  %VMRUN% -T ws -gu dgj -gp "ATest4LocalStorage!"
    runScriptInGuest %CLONE%%%i\WXP%%i.vmx -nowait ""
    "cmd.exe /k C:\setData.bat
  timeout -T 60 /NOBREAK >NUL
  %VMRUN% -T ws suspend %CLONE%%%i\WXP%%i.vmx hard
)                                                          (5)
```

The MakeGetVisits script differ from the MakeSetVisits script in (Algorithm 5) only in the target script that is run locally on the client system: getData.bat. 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.

6 PHASE 2 - DATA

6.1 Seeding

After proving that the authors could hide and retrieve information in a client's Web Storage in a controlled environment, our task was to discover what would happen in the wild. Interesting questions surfaced, such as how many copies of our user encrypted and obfuscated parts are needed to disseminate in order to ensure recovery and feel confident that the parts could be retrieved intact after 5 days, 30 days, or even 1 year.

In order to begin answering these questions, permission was obtained to use two of the author's departmental web presences (http://www.ist.rit.edu and http://www.nssa.rit.edu). To make the results of this testing more accurate, the decision was made to remove all visitors from the 129.21.0.0/16 domain (RIT's domain). This decision was made because most of the visitors to these sites from that domain are the universities' lab machines that are forced to visit those sites on browser launch and are re-imaged at startup. Since the set data on the lab machines would be removed at startup and the machines visit these sites multiple times a day, using results from these machines would skew the results in an unfavorable way.

6.2 Limits

The testing and data collection phase went live on December 17th, 2011. While the data setting and collecting is still ongoing, for this paper it was decided to cap the data analysis on February 22nd , 2012 – so the preliminary data in this paper is from 67 days. While this may be a small data set, interesting trends are already being seen.

6.3 Observations

An observational study was run with input variables number of sets available and the number of days until retrieval. The sample size for this study was n 3804 - full sets of data seeded. The response was measured as the number of days until a full set was received. Figure 1 contains the plot of the response and the number of sets available.

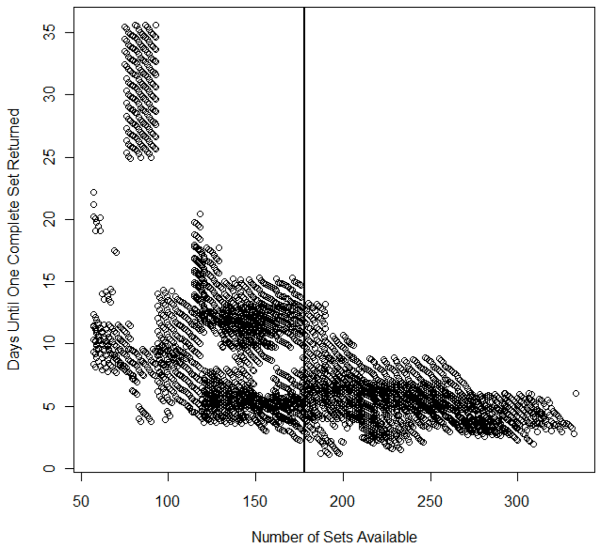

Figure 1. Days until one complete set returned against the number of sets available.

The vertical line on the plot occurs at 178 sets complete. This cut-off point was chosen since set 178 was the last set available for a total of 35 days. This is important because from the limited timeframe of our data collection, waiting 35 days for retrieval was determined to be our upper limit. As our data collection grows and ages, the authors look forward to seeing what the revisit rates will be for longer periods of time.

From the plot, a relationship is apparent. The relationship seems to be a slow decay then level off as the number of sets increase. This relationship is anticipated since it is logical to expect to get a full set back faster with more sets available. There is also a set of points between 75 and 93 sets that seem to be an anomaly as compared to the rest of the data. These data points represent 5% of the overall data and seem to occur for responses larger than 25 days.

Figure 2 contains the plot of the response and the number of days before retrieval.

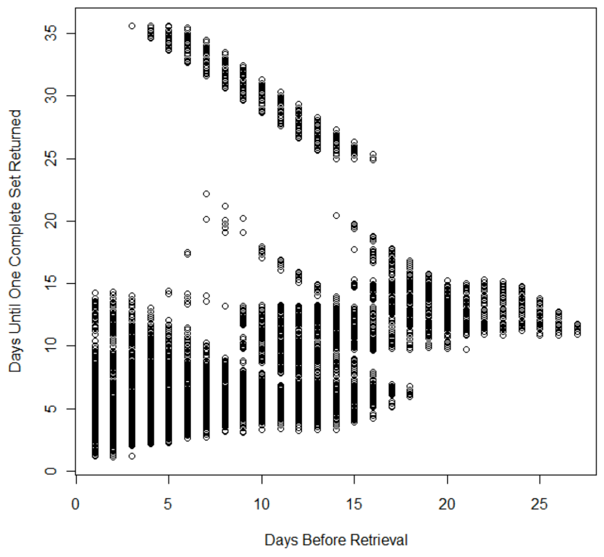

Figure 2. Days until on complete set returned over the number of days before retrieval

Based on Figure 2, overall there seems to be some sort of increasing relationship occurring. The same group of data points from Figure 1 does not seem to fit with the rest of the data on this plot as well.

Table II includes the information on the rank correlations between the response and each input variable.

TABLE II. CORRELATION TABLE

Input Variable	**Estimate**	**p-value**
Sets Available	-0.451	< 2.2E-16
Days Until Retrieval	0.361	< 2.2E-16

Based on the results in Table II, there is a significant correlation between the response and both inputs at the α 0.05 level. From the estimates, the relationship between the response and the sets available seems to be decreasing and the relationship between the response and the days until retrieval is increasing. This matches what is in the figures above.

6.4 Next Steps

From here the authors would like to increase the size and age of the study to assess whether or not the anomaly data that we observed is repeatable and allow for the all of the sets to be available for an equal length of time. We would like to create a model that can be used to predict the response based on the inputs and run this study in an experimental setting with other variables such as different websites, different configurations on the break-up of the set, etc.

6.5 Exclusions

From the capped data (as of 2/22/12), a total of 10263 initial visit parts have been set, with 24873 re-visits. Of the total 35136 visits to our sites, 1825 (5.19%) either had Web Storage turned off or their browser was unable to implement it. For these visits, the database tracked the User Agent to determine the reasons. Some of the browsers were simply too old (Internet Explorer 7) or aren't equipped to handle local storage (Opera Mini). Of the 35136 total visits, 328 had browser versions that have Web Storage implemented, but had it turned off (0.933 %), while the remaining 1497 visits were by browsers or devices that were incapable of Web Storage.

The authors postulate that the very low number of visitors who had disabled Web Storage (<1%) may be due to many factors, including the standard being new and unfamiliar or a less than standard confusing interface. While this is an area that could use future study, it only adds to the viability of Web Storage as being a useful tool for our purposes.

7 Phase 3 – Detection

Once a greater data set has been accumulated, 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.

8 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 (1^{st}-party and 3^{rd} party). However, since most trust the site they are visiting and 1^{st}-party is what is being used, this number is relatively small. The number of visitors blocking 1^{st}-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.

9 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.

10 References

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

[2] N. Leenheer, sights "The HTML5 Test." Last modified April 2012 – version 3.0, accessed April 22, 2012. http://html5test.com/

[3] 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.

[4] 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/.

[5] 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.

[6] 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.

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

[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.