HTML5 IndexedDB Encryption: Prevention against Potential Attacks

Stefan Kimak, Jeremy Ellman
Faculty of Engineering and Environment
Northumbria University
Newcastle Upon Tyne, UK

Abstract

Over the past 20 years web browsers have changed considerably from being a simple text display to now supporting complex multimedia applications. The client can now enjoy chatting, playing games and Internet banking. All these applications have something in common, they can be run on multiple platforms and in some cases they will run offline. With the introduction of HTML5 this evolution will continue, with browsers offering greater levels of functionality. This paper outlines the background study and the importance of new technologies, such as HTML5's new browser based storage called IndexedDB. We will show how the technology of storing data on the client side has changed over the time and how the technologies for storing data on the client will be used in future when considering known security issues. Further, we propose a solution to IndexedDB's known security issues in form of a security model, which will extend the current model.

1. Introduction

This paper attempts to answer several questions. Firstly, what is HTML5 IndexedDb; where did the technology come from, and what motivated its development; what is its current status and what is inhibiting its take up, and finally, what is the future for HTML5 IndexedDb.

HTML5 is the latest W3C standard for the language in which web pages are written. It also defines Application Programming Interfaces (APIs) that are expected to be provided by a web browser that supports HTML5. The motivation for the changes and enhancements coming with HTML5 is that the web browser should be capable of running browser based applications in the same way that it supports desktop applications. That is, client side processes will be able to avoid the ineffectiveness and network connectivity issues found in server side applications and the inherent visual instability caused by their required web page refreshes. Consequently, major browsers now support the majority of the new HTML5 components and APIs. Therefore, HTML5 browser based storage may well contain stored data from online services that makes use of the new HTML5 functionality [1]. The process of accessing this data might in some cases be slow, due to network latency or database query process [2]. It is suggested that this new level of browser based storage will ensure that such HTML5 enabled browsers are going to be a significant target for cyber-attacks [3].

Web browsers store history, and other data using cookies on client computers, which is attractive for those marketing products or services. Being browser based is critical for developers, because modern browser based web applications are able to store large amount of data and access that data much faster than any server side database. Consequently, HTML5 opens up entirely new security challenges and issues [4]. As is well known, user information is tracked on every move on the Internet. eCommerce sites store customer details, orders and saved products, sites store cookies on user computers to track returning customers. The data can be later used for marketing purposes and to target new customers. Sometimes consumers and the general public do not realize the quantity of personal data that is shared over the Internet and how that data can be used or misused. Data privacy and information leakage is then a serious concern.

2. Importance of IndexedDB

In this section we consider the drivers behind HTML5 IndexedDB. That is, why the technology was considered at all, and what motivates current standard. We proceed as follows, firstly we overview the status of eCommerce, with particular attention to its mobile variant mCommerce. Then we consider

browser based cookies that are IndexedDB's intellectual antecedents.

2.1. eCommerce

The term eCommerce began to be widely used in early 2000, and it is defined as commercial transactions conducted electronically on the Internet, such as purchasing goods and services online. eCommerce has a significant and positive impact on businesses everywhere [5,6]. The eCommerce market grew slowly until 2007 when its proportion of GDP was about 3%, but the biggest expansion happened in the last decade when the retails sales increased to 40 %. In 2012, eCommerce accounted for 18% (£492 billion) of UK business turnover. 21% of UK businesses in 2012 made eCommerce sales to their own country; 9% of UK businesses made eCommerce sales to EU countries and 7% to non-EU countries based on the Office of National statistics (ONS) [7]. Of the United Kingdom's total £1.45 trillion GDP, the Internet value chain represented 2.6% of GDP, and the eCommerce conducted over it a further 3.1%. The UK boasts 54.6 million Internet users and has a penetration rate of 86%, with a typical user spending 42 minutes per week browsing virtual stores and buying on the web. The use of eCommerce, by both organizations and individual consumers continues to grow, as more people are connected to the Internet and with the increased availability of fibre broadband.

eCommerce has several advantages over offline stores and mail catalogues. Online stores eliminate the 3rd party middleman costs required by wholesalers and distributors. It also removes the overheads of physical shops, both of which lower operational costs. eCommerce stores also provide search functionality so that a customer is looking exactly the item for which they are looking. Additionally, customers can easily browse through large amounts of products and services eCommerce has been expanded to the business to business (B2B) and business to consumer (B2C) [8] markets. Many retailers have moved to invest in online sales that target more online customers in categories such as electronics, books, and transport. eCommerce gives retailers an opportunity to expand outside their domestic markets with minimum upfront investment [9]. Consumers can also see prices, allowing simple price comparison and can then place orders quickly. eCommerce stores allow the customer to add products to a wish list, which can then be sent to friends or family to be paid for.

Consumers can also check existing online product reviews and compare prices before buying any goods or services. Some of the eCommerce stores provide video product reviews where the customer can see the product in action without the need to leave the house. One of the most important advantages of eCommerce stores is access to the global market and the creation of new business opportunities Online stores can be available to everyone everywhere. For most businesses the eCommerce is an excellent alternative supply channel that is cost effective but continuously can reach consumers directly and extensively. The Internet helps companies to engage in eCommerce by collecting, storing, and exchanging personal information obtained from visitors to their websites [10]. eCommerce stores can target customers in many ways, most widely used is by email and online ads. This way has a cost advantage over offline stores, where print flyers must be produced. eCommerce can target a greater number of customer in a shorter space of time, as everything is done electronically. eCommerce retailers have also advantage of through popular social media to attract new customers. Additionally, online retailers use customer buying habits to target new and existing customers with social media advertisements and special offers. Since the late 90's it was predicted that every step on the Internet could be traced and that this information might be stored [11]. Information from web browsing is stored in the browser history, eCommerce sites store user preferences and shop orders to better understand the customer with the aim of targeting advertising at them for related products or services. [12].

2.2. The importance of mobile commerce (mCommerce)

HTML was first focused on the desktop computer, since when the web started in 1993 mobile (cell) telephones did not have Internet connectivity. Tablet computers (e.g. the Grid Compass) were a rarity and limited to specialised applications. In 2015, combined, mobile phones and tablets now account for 38% of all web pages served globally (StatCounter). Smartphone user penetration in 2011 was 9.6%, whilst by 2015 it is 28%. Here in UK the mobile penetration rate is 72%.

The U.K. Internet ecosystem is worth £82 billion a year, with mobile connections accounting for 16% of this. mCommerce sales will continue to grow at a double-digit rate until 2017 when it is expected to reach £17.2 billion and drive over 26% of online retail sales. Since 2007 mCommerce sales have rapidly grown from less than 5% to 21% of sales in 2015, which opened a new market for online stores. Online stores needed to change their strategy and embrace the mobile market.

Businesses operating over the Internet need to maintain relations with their customers to ensure continuity, recognize previous customers, and simplify the eCommerce process. This is done using cookies.

2.3. Cookies

Cookies are small quantities of data that are stored by websites in the client browser, and sent to that web site with each http request. Cookies were introduced by Netscape Communications in 1994 in their Netscape Navigator Browser. Cookies allow users to store their sessions and state with websites.

eCommerce applications use cookies to store customers' preferences. This both makes the online buying experience more convenient and focused as cookies allow the tracking of customer preferences. Cookies support functionality such as customer shopping carts and the recognition of returning customers who are then recommended appropriate products and made offers using individually tailored marketing.

The problem with cookies is privacy since third party applications could potentially steal information from cookies. There are also several security issues with cookies such as cookie poisoning [13], and cookie injection. Cookie poisoning attacks involves the modification of the cookie contents (user IDs, passwords, account numbers, time stamps) in order to bypass security mechanisms. Using cookie-poisoning attacks, attackers can gain unauthorized information about another user and steal their identity. Cookie injection attacks inject a cookie string or code into the HTTP header to modify server page execution, which may lead to SQL injection attacks [14].

The Cookie Law is the result of a EU directive in 2011 and enacted into law across the majority of the European Union. It requires websites to obtain visitors' agreement to store or retrieve any information on a computer, smartphone or tablet [15].

It was designed to protect online privacy, by making consumers aware of how information about them is collected and used online, and give them a choice to allow it or not.

Cookies are limited in size to 4KB, and therefore are rarely used to directly store site-specific information. Rather, a typical cookie will store a unique database key. That key will usually point into the web server's customer database that is not accessible publicly. That database may contain any amount of information about the customer including their personal details, transaction and purchase history, preferences and so on. Cookies, and their limitations in size and flexibility, have lead to the specification and development of HTML IndexdDB.

3. Current state of IndexedDB

Browser storage has been proposed in HTML5 to extend the cookies functionality to provide web developers and web applications with better alternatives to store data locally. With browser based storage eCommerce companies can store user preferences, shopping cart and product images locally. This can help eCommerce applications to speed up the process of loading products and displaying them to end-users.

By using browser-based storage, eCommerce sites can also be used offline. The user will have the ability to add products to a shopping cart, even if the network connection is down. The greatest advantage of using offline storage is with mobile devices, where network connectivity and data caps are a concern.

If a Web service allows only a certain number of calls per hour but the data does not change that often, web applications could store the information in local storage and so help prevent mobile users breaching data limits [16]. Online stores could save new images every six hours, rather than every minute, which would improve the bandwidth utilization.

Local caching keeps users from being banned from services, and it also means that when a call to the Application program interface (API) fails, user will still have information to display. For example, shopping cart data could be stored locally and synchronized with the eCommerce site when network connectivity is restored.

The main problem with HTTP as the main transport layer of the Web is that it is stateless. This means that when an application is closed, its state will be reset the next time is opened. If an application on the desktop is closed and then re-opened, its most recent state is restored. Local storage is better than cookies since it allows for storage across multiple windows. It also has better security and performance and data will persist even after the browser is closed [17]. Therefore, local storage provides functionality similar to that of desktop applications, where application state is persistent.

In 1995 Netscape Corp's vision of the future was to run multimedia application, spreadsheets and word processing programs from the web browser. Netscape's main product was a browser (Naviagator), which was written to run across multiple operating systems (Windows, Unix, and Macintosh). The vision was that application would run on the top of any operating system with their sets of APIs, so that third party applications developers would not need to worry about the underlying operating system and hardware. These days, Netscape's vision is a reality, where applications like YouTube and Facebook can be run from any web browser.

As the HTML5 standard evolved, new browser based storage concepts were introduced. These are targeted at storing larger data volumes. Additionally they have satisfied the key non-functional requirement of speed, since stored data was not transmitted with every HTTP request, whilst cookies are. HTML5 provides two new features to store data locally. The first browser based storage feature is called 'local storage'. It allows the storage of information locally within a web browser in object stores, which are persistent and stored on disk. The storage is limited to 5MB and the stored data is in name/value pairs.

IndexedDB is another HTML5 browser based storage technology. It is a NoSQL (Not only SQL) [18], asynchronous, key-value browser-based data store, where NoSQL is an approach to databases that is not relational or object oriented. Rather, NoSQL stores data in key/value format. The database can handle a large amount of data. IndexedDB supports an API that offers fast access to unlimited amount of structured data. IndexedDB may be considered to be insecure, since security was not considered in its specification. In a previous paper [19] we have described how standard forensic tools may be used to identify data stored, and then deleted from IndexedDB data stores.

IndexedDB, which was previously known as WebSimpleDB came from the W3C specification of implementing web storage into web browser in 2009. IndexedDB is a persistent client-side database implemented into browser and is an alternative to WebSQL, which has been deprecated. Mozilla and Microsoft supported the introduction of IndexdDB, which was most influenced by Oracle's Berkley DB. The application uses local data stored on a client system [20]. It caches large data from server to the web browser client using JavaScript Object Stores, which may be considered to be equivalent to tables in relational databases.

Files and data stored by the browser are retained on the user file storage system, on the user's computer hard drive. The client-side database, IndexedDB, stores the data, even when the browser terminates. IndexedDB is then a persistent client-side database, which means that the data can be retrieved even the browser is offline. Therefore, the files reside on the user file system and can be recovered until other files overwrite them. IndexedDB treats file data just like any other type of data. An application can write a File or a Blob into IndexedDB, as well as storing strings, numbers and JavaScript objects. This is detailed in the IndexedDB specifications and, so far, implemented in both the Firefox and Chrome applications of IndexedDB. Using this, storing all information in one place and a single query to IndexedDB can return all the data.

In Firefox and Chrome's IndexedDB implementation, the files are stored transparently external to the actual database; in other words, the performance of storing some data in IndexedDB is just as efficient as storing it directly in the OS filesystem. Storing files does not extend the database size and slow down other operations. Moreover, reading from the file means that the implementation reads from an OS file. The Firefox IndexedDB implementation is even smart enough to recognize if is storing the same Blob in multiple files. If this happens it creates only one copy. Writing further references to the same Blob just adds to an internal reference counter. This is completely transparent to the web page, so it writes data faster while using fewer resources.

Browser based storage such as IndexedDB can be used on multiple browsers and is cross platform compatible. Web applications can take advantage of using IndexedDB on desktop, mobile and tablet, without additional programming. Web applications can use browser-based storage without the need for network connections. The HTML5 standard provides the functionality where data can be stored on client machine, and can be accessed anytime without the need of network connection.

An important aspect of HTML5 is that the web applications can run offline using local storage. The advantage of HTML5 compared to desktop programs is that web applications do not require any installation or start-up configuration and will also run on any device that supports HTML5, such as laptops, phones or tablets. In an eCommerce scenario, this reduces the entry barrier to new customers since customers can begin taking advantage of web applications just by visiting the relevant web site.

IndexedDB extends local storage by providing web applications with offline storage. This may be used by eCommerce stores to store customer preferences without sending these with every HTTP request. Consequently, HTTP request and response traffic will decrease and customer preferences or other information will be accessed only when requested. An important aspect of HTML5 is that the web applications can run offline using local storage. This means that client data will be stored on user's browser and accessed anytime that the application requires. Offline storage and cached pages provide a better user experience, since network latency is minimal.

The new HTML5 IndexedDB functionalities bring new security issues, since there is increased access to the client computer's resources. One of the biggest disadvantages or disappointments is that the new standard does not provide any additional security. HTML5 video and audio are replacing third party application as Adobe and closing a common attack vector with FLASH applications or plug-ins. Additionally HTML5 provides greater access to computer resources, which includes local storage, and therefore opens new opportunities for attacks.

The problem with the current browser based storage, such as IndexedDB is that there is concern that another application on the client computer may also access that offline data. To prevent web applications from reading each other's data, a mechanism known as the same origin policy (SOP) applies to all of the web storage technologies. By implementing the same origin policy, browsers check and record the origin of all the data they store based on the hostname of the web application (www.example.com), the port number on which the web application runs (80) and the protocol through which the data was delivered (typically http or https). When a web application wants to access data stored locally, the browser will check the current origin and

the origin of the data and only allow access if these match. Data is protected through the use of the same origin policy.

The SOP is the only form of browser protection against potential security threats. SOP works by not allowing access to client data from sources that could be deemed to be the original source, perhaps by the use of cross-site scripting (XSS) for example. That is, if applications in multiple windows or frames are downloaded from different servers, they should not be able to access each other's data and scripts (Takesue, 2008). The prevention of data or attacks coming from a different domain is possible. Web browsers are using this prevention technique against untrusted site attacks. Attackers use multiple techniques that can easily inspect the browser history or get data of another domain.

From the experiments performed, we have confirmed IndexedDB stores data as received, so that it is not encrypted. However, this is not the only problem. Browser based storage faces another issue, where the deleted data is not fully deleted from the hard drive. With the help of standard forensic tools we were able to restore current and deleted IndexedDB data from both desktop and mobile drives.

The issue of restoring deleted data just extends the security concern of storing data in unencrypted state, where the attacker could get multiple versions of browser based local storage. The deleted data persists on the hard drive and when delete data request is executed, the data is just marked as deleted but still occupies the associated space. A further data storage request just assigns additional disk space but the old data will persist on the hard drive and it will be not overwritten.

We have than a complex scenario with IndexedDB. It has the advantages of persistence, storage size, and better network utilization, but the disadvantages of security weakness.

4. Potential attacks

4.1. CORS

Cross origin resource sharing (CORS) is a mechanism that allows JavaScript on a web page to make XMLHttpRequests (XHR) to another domain, not the domain the JavaScript originated from. XHR is an API available to web browser scripting languages such as JavaScript. It is used to send HTTP or HTTPS requests to a web server and load the server response data back into the script.

Normally, web browsers would otherwise forbid such 'cross-domain' requests. CORS defines a way in which the browser and the server can interact to determine whether or not to allow the cross-origin request. By letting third party applications accessing the data created with other domains application can lead to security issues, such as information leakage. Therefore user agents must implement Cross-origin resource sharing with IndexedDB in greater security details. Also, CORS expands on the design of the Same Origin Policy. Each resource declares a set of origins, which are able to issue various kinds of requests (such as DELETE, INSERT, UPDATE) to, and read the contents of, the resource. CORS is a "blind response" technique controlled by an extra HTTP header (origin), which, when added, allows the request to reach the target. This means, that an application creates an IndexedDB database, which is saved with the domain name. Another application can not access the database files, as the access is restricted for the particular domain. This attack is based on bypassing the Same Origin Policy and establishing cross-domain connections to allow the deployment of a Cross-site Request Forgery attack vector. We mention a CORS attack, which can be used to bypass the restriction and read data from other domains.

4.2. XSS

Cross-site scripting is one of the most popular web application attack, third on the OWASP list. WhiteHat security has provided a statistic report where XSS regains the number one vulnerability in web applications. XSS is popular attack, because even the web application is secured the attack rely on the end user, which can be tricked to click a link and therefore authorize the attack.

XSS is taking advantage of web applications, where the user input is not filtered properly. Cross site scripting filtering is a process of filtering out parameter values that look suspicious, this includes special characters. Attackers may also manipulate indirect inputs such as session variables and database records. This can be prevented with sanitizing or validation of user input. XSS is an attack technique that forces a Web site to display malicious code, which then executes in a user's Web browser.

New client side database provide the functionality to store data on user machine. Stored data might contain information, which is considered sensitive, such as user personal information. If a web application is vulnerable to XSS attack, then an attacker could get access to client side storage. The client side storage data can be accessed through the browser, so the execution of XSS attack might output the stored data.

4.3. Social engineering attacks

Social engineering is the art of manipulating people so they give up confidential information. The attackers usually trick people into giving them passwords or bank information, or access to computer to secretly install malicious software with the purpose access information or control. Attackers use social engineering tactics because it is usually easier to

exploit human nature to trust than it is to discover ways to hack web applications or software. For example, it is much easier to fool someone into giving the attacker their password than it is trying to hack their password.

Example of social engineering can be a email from a friend. If attacker manages hack or socially engineer one person's email password, then the attacker would have access to the victim contact list. The attacker can send email or leave message to victim's contacts list with a link, which could be result that the victims computer will be infected by malware or the victim is redirected to attackers site. The link could also contain a download, such as picture, movie, document, or audio file that has malicious code embed in. When the victim downloads the file, the victim's computer will be infected and the attacker could have access to victims machine, emails, accounts and contacts.

4.4. Physical Access

Physical access is possible when the attacker has the physical contact to user machine. When the device or stored data is unencrypted, the attacker might get access to all data. Physical access controls to the location where the computers are kept. Employees who are authorized to work in that location can use either a RFID card or some magnetic stripe or barcode on their ID badge to gain access through a locked door. This allows the accesses to the location to be assessed on a per employee basis. When considering physical access, the attacker or any person with access to the filesystem could potentially get the file and the data, which will mean that it could be transferred to an external drive and used with the appropriate application.

Possible solution to prevent an unauthorized person to gain access to filesystem is to lock the screen, where a password would need to be entered before any of the files could be viewed.

5. Encryption for IndexedDB

The future of IndexedDB is to support secure of browser-based offline usage. Existing browser-based storage has not become popular with web developers, because they face several problems. The first problem is the complexity of code required, where the developers need extra time to understand the structure. Saying that, there are many online tutorial examples, which can help developers to start implementing browser, based storage into their web applications.

The second problem with IndexedDB is security. Currently IndexedDB stores data in an unencrypted state so that is neither protected, nor securely deleted. Therefore IndexedDB storage cannot be recommended for the storage of personal information. This makes it limited in functionality. As with data stored on desktop, mobile or tablet in an unencrypted state, an attacker can get the data without bypassing any protection. For example, with a Cross-site scripting attack (XSS), such as hidden in email link an attacker could find the stored data. IndexedDB is inherently vulnerable to such attacks.

Security flaws are inevitable when considering web applications and storage of information. This is due not only to the sophistication of the attacks, but also to the fact the many attacks, such as cross-site scripting, are based on social engineering and exploit human error, so are extremely difficult to protect against. Browser based storage security design is a concern, but that can be corrected. The correction is to use JavaScript client side encryption, which would mean that browser based storage is at least as secure as that on the server.

With the implementation of the encryption library in the browser (Firefox v. 29) we are hoping to address the ineffectiveness of the insecure storing of data. We are going to propose and develop an algorithm, which will be implemented into the Mozilla Firefox browser in an extension format. Also, our algorithm will ensure that the database transaction for storing or retrieving data will only be possible when a secure and valid authentication is completed. This also relies upon providing the private key to encrypt/decrypt data.

We have proposed a security model, which will be implemented as a browser extension. The proposed security model extends that of the current web browser. Furthermore, we have implemented a browser extension with a client side encryption library, which will help to secure the data on a client's machine. When an application requests a new transaction for IndexedDB to open the database and save data, the proposed library extension will encrypt the data. This data will then be as safe as the encryption scheme even should an attacker get physical access to the device, which would happen if a laptop were lost, or a mobile handset stolen.

Steps to encrypt the data are:

a)Get a secure Login

The first step is to provide a secure login functionality, which can be provided by the web application. The web application will use the login process to authenticate a user and securely log the user into system.

b)Encrypt data

When an application requests a new transaction for IndexedDB to open the database and save data, the designed encryption library extension will encrypt the data. This way the data will be stored in an encrypted state and will not be readable to others.

c)Store public and private key

When the data is encrypted a key will be generated and stored with the user information on the server.

Client-side encrypts sensitive data using the public key, which will be generated and stored on the server side. This public key is used when encrypting

information using the JavaScript library. When Client-side encryption is enabled, an RSA keypair is generated and the user will be given a specially formatted version of the public key. RSA is the algorithm that is used to encrypt data with a private key to produce a digital signature. The private key, however, is never revealed to the user or anyone else. The data is decrypted using the keypair's private key.
Public and private keys are created simultaneously using the same algorithm (RSA- Rivest-Shamir-Adleman). Private keys are used to decrypt text that has been encrypted with a public key.

d)Decryption of data

When the user requests to read the data from the database, the web application will check user's credentials (if the session is active) and get the key from the server to allow decryption of data.

e)User Authentication

Upon successful authentication, the user will be given a public key, which will be used for the encryption/decryption of the data. This private key will be stored on the server side, with all the user information which is used for decrypt the data. We are going to use OAuth 2, which is an open standard for authorization. This will be used to securely transfer the private key from the server to the encryption library.

f)Deletion of data

Secure deletion of data will be required to overwrite the space of data with zeros. This means that the data cannot be read again, as all of the values are set to zero.

If running over HTTPS, then things are more secure as the browser will detect a modified JavaScript file. The SSL layer of HTTPS protocol handles this.

5.1. Proposal

Hashing and encryption can be done within browsers through the JavaScript encryption library. Algorithm will use a JavaScript encryption library (proposed Stanford JS Encryption Library), where the library will be implemented into the browser (Firefox) as an extension. This extension will be based on the top of IndexedDB API and therefore every time during the reading or writing of data, the data will be encrypted. The library consists of encryption with private and public keys. The private key will be saved on the server. The public key will be given to the user and stored on the user's machine, the same way as a cookie. The extension will provide encryption/decryption of data on the user's machine, which will resolve the issue of storing data in an unencrypted state.

5.2. Algorithm Used

It differs from typical AES implementations (different approach that keeps the code small and speeds up encryption/decryption). The source code for the AES algorithm, also called Advanced Encryption Standard or the Rijndael algorithm. The benchmarking tests have shown that the Stanford JS Encryption library performs faster than other client side encryption libraries. The benchmark has been achieved in multiple browsers on Windows, Mac and Linux Operating Systems. One of the reasons we proposed to use and implement the library into algorithm was the speed and multiplatform usage.

The algorithm is going to contain the JavaScript encryption library, which will be implemented into the browser. The algorithm will consist of a few steps, with the higher security. This will allow the end user to save and retrieve data from IndexedDB. The data will be encrypted with the JavaScript library and a private and public key will be used to encrypt/decrypt this data.

5.3. Implementation

The model will add an extra layer between the web browser and IndexedDB API. The security model consists of an algorithm framework, which adds extra protection against issues identified, by reading each other's data through XSS vulnerabilities.

Algorithm is using JavaScript encryption library (proposed Stanford JS Encryption Library), where the library is implemented into the browser (Firefox) as an extension. This extension is placed on the top of IndexedDB API and therefore every time during the reading or writing of data, the data will be encrypted. The library consists of encryption with private and public keys. As expected, the private key will be saved on the server. The public key will be given to the user and stored on the user's machine, the same way as a cookie. The extension will provide encryption/decryption of data on the user's machine, which will resolve the issue of storing data in an unencrypted state. It will also provide better security for possible attacks, where the attacker can manipulate with user data.

The browser based local storage security model (BBLS) is relying on the web browser security model (WBSM), which is using Same origin policy. The security mechanism is not enough to preserve the security confidence among the end user.

The BBLS security model differs from WBSM in few ways, which includes the security mechanism. The main difference is that BBLS security model is trying to secure the data between browser and the end user file system, where comparing to WBSM, which is securing the data between web applications and user browser.

The goal of BBLS security model is to secure the data, which is stored in client side database. User should be able to visits other websites, without they databases to be compromised.

The current WBSM is not sufficient protection for complex web application and stored data on client

side is becoming more important.

The security model consists of an encryption framework, which will help to secure the data. The encryption framework cannot though provide full security protection. We argue that such protection is not achievable in a single machine, since any single browser could be the target of an XSS attack. Therefore, functionality external to be browser needs to be implemented. For implementation to existing encryption library we will use Multifactor authentication (MFA). MFA is used to make the authentication process more secure by adding an extra layer of security. The extra authentication will need to be passed to make sure the encryption library decrypts the data. Mobile two-factor authentication use phones to replace fobs or software-based tokens that were commonly used for remote authentication. When a person tries to log into an online service, a security pin is sent to his or her mobile phone via voice or SMS message, rather than to the token.

5.4. Evaluation

To evaluate the security model, we will run tests to conclude the effectiveness of the model. This will include attacks, which will be bypassing the SOP trough XSS attacks. First we will perform and attack with existing security, without applying the security model.

Then we will add the security model, and perform the attack again. We suggest that the model will prevent an attacker to read data from other source, by adding the authentication process to place. Also the data stored will be encrypted, which means that even the authentication process is compromised, the data will not be available to read in unencrypted state.

Based on our findings, we can state that there is a case for browser-based databases. We have implemented a JavaScript encryption framework, which is a part of the security model implemented into the browser in a form of an extension. The proposed security model extension addresses the security issue that IndexedDB has as a product of its design. Also, the implemented security model fulfils the security requirements.

6. Conclusion and Future Work

Based on these findings, we can state, that there is a case for browser-based databases. Browser based databases though face security problems over and above those on the server, and this has inhibited their uptake. Nevertheless, despite the existing issues faced by browser-based storage, there is a future for the technology due to its convenience, performance, reduced reliance on continuously available network connection.

Considering the issues and concerns of storing data locally, browser based storage has the potential to be widely used, where the main advantage is the performance speed, cross platform (desktop, mobile, tablet) and browser availability. The advantages of local storage outweighs the disadvantages, keeping in mind that the issues identified can be corrected and browser-based storage can be widely used by developers without any concerns of security issues introduced as by design limitations.

Although the proposed security framework has been successfully applied to browser based local storage, further improvements can be made in extending the security and performance model. These could be addressed by extending the current model to use further security factors such as biometrics.

7. References

[1] Naseem, S.Z. Majeed, F. (2013) Extending HTML5 local storage to save more data; efficiently and in more structured way. Eighth International Digital Information Management (ICDIM).

[2] Zhanikeev, M. (2013) A Practical Software Model for Content Aggregation in Browsers Using Recent Advances in HTML5. 37th Annual Computer Software and Applications Conference Workshops (COMPSACW). pp.151-156, Japan 22-26 July 2013

[3] Ryck, P. Desmet, L. Philippaerts, P. Piessens, F. (2011) A Security Analysis of Next Generation Web Standards, (European Union Agency for Network and Information Security - ENISA). Tech. Rep.

[4] Anttonen, M. Salminen, A. Mikkonen, T. Taivalsaari, A (2011) Transforming the web into a real application platform: new technologies, emerging trends and missing pieces. ACM Symposium on Applied Computing. New York, NY, USA. Pp. 800-807.

[5] Chuang, T. T., Nakatani, K, Chen, J. C. H. and Huang, I. L. (2007). Examining the Impact of Organisational and Owner's Characteristics on the Extent of E-commerce Adoption in SMEs,

[6] Pool, P. W., Parnell, J. A., Spillan, J. E., Carraher, S. and Lester, D. L. (2006). Are SMEs Meetings the Challenge of Integrating E-commerce into Their Businesses? A Review of the Development, Challenges and Opportunities, International Journal Information Technology and Management, 5(2/3), pp.97-113.

[7] Jones, J. (2014) E-commerce: measuring, monitoring and gross domestic product. ONS. Available at: http://www.ons.gov.uk/ons/rel/gva/national-accounts-articles/e-commerce--measuring--monitoring-and-gross-domestic-product/index.html (Accessed: 20 September 2015).

[8] Ta, H., Esper, T., & Hofer, A. R. (2015). Business-to-Consumer (B2C) Collaboration: Rethinking the Role of Consumers in Supply Chain Management. Journal of Business Logistics, 36(1), 133-134.

[9] Xiaojing, L., Liwei, Z., & Weiqing, W. (2012). The mechanism analysis of the impact of eCommerce to the changing of economic growth mode. In Robotics and Applications (ISRA), 2012 IEEE Symposium on (pp. 698-700). IEEE.

[10] Boritz, E., Gyun, W., and Sundarraj, P. 2008. Internet privacy in E-commerce: Framework, review and opportunities for future research. In: Proceedings of the 41st Hawaii International Conference on System Sciences. Hawaii, January 7-10 2008, pp.204-256.

[11] Gehling, B., & Stankard, D. (2005). eCommerce security. In Proceedings of the 2nd annual conference on Information security curriculum development (pp. 32-37). ACM.

[12] Gómez, J. M., & Lichtenberg, J. (2007). Intrusion Detection Management System for ECommerce Security. Journal of Information Privacy and Security, 3(4), 19-31.

[13] Buja, G., Jalil, K. B. A., Ali, F. B., Mohd, H., & Rahman, T. F. A. (2014). Detection model for SQL injection attack: An approach for preventing a web application from the SQL injection attack. In Computer Applications and Industrial Electronics (ISCAIE), 2014 IEEE Symposium on (pp. 60-64). IEEE.

[14] Appelt, D., Nguyen, C. D., Briand, L. C., & Alshahwan, N. (2014). Automated testing for SQL injection vulnerabilities: An input mutation approach. In Proceedings of the 2014 International Symposium on Software Testing and Analysis (pp. 259-269). ACM.

[15] Summers, S., Schwarzenegger, C., Ege, G., & Young, F. (2014). The emergence of EU criminal law: cyber crime and the regulation of the information society. Bloomsbury Publishing.

[16] Karthik, R., Patlolla, D. R., Sorokine, A., White, D. A., & Myers, A. T. (2014). Building a secure and feature-rich mobile mapping service app using HTML5: challenges and best practices. In Proceedings of the 12th ACM international symposium on Mobility management and wireless access (pp. 115-118). ACM.

[17] Ayenson, M. Wambach, D. J. Soltani, A. Good, N. Hoofnagle, C. J. (2011) Flash cookies and privacy II: Now with HTML5 and ETag respawning. Computer and Information Systems Abstracts. [Online]. Available at: http://dx.doi.org/10.2139/ssrn.1898390 (Accessed: 10 February 2015).

[18] Strozzi, C. (1998) NoSQL A Relational Database Management System. Available at: http://www. strozzi. it/cgi-bin/CSA/tw7/I/en_US/nosql/Home% 20Page (Accessed: 20 September 2015).

[19] Kimak, S. Ellman, J. Laing, C. (2014) Some Potential Issues with the Security of HTML5 IndexedDB. In: System Safety and Cyber Security 2014 (IET Conference), 14-16th October 2014, The Midland Hotel, Manchester, UK.

[20] Casario, M. Elst, P. Brown, Ch. Wormser, N. Hanguez,C. (2011) HTML5 Solutions: Essential Techniques for HTML5 Developers. Publisher: FRIENDS OF ED; 1 edition ISBN: 1430233869.

