

Northumbria Research Link

Citation: Kimak, Stefan (2016) An Investigation into Possible Attacks on HTML5 IndexedDB and their Prevention. Doctoral thesis, Northumbria University.

This version was downloaded from Northumbria Research Link:
<http://nrl.northumbria.ac.uk/30260/>

Northumbria University has developed Northumbria Research Link (NRL) to enable users to access the University's research output. Copyright © and moral rights for items on NRL are retained by the individual author(s) and/or other copyright owners. Single copies of full items can be reproduced, displayed or performed, and given to third parties in any format or medium for personal research or study, educational, or not-for-profit purposes without prior permission or charge, provided the authors, title and full bibliographic details are given, as well as a hyperlink and/or URL to the original metadata page. The content must not be changed in any way. Full items must not be sold commercially in any format or medium without formal permission of the copyright holder. The full policy is available online: <http://nrl.northumbria.ac.uk/policies.html>

www.northumbria.ac.uk/nrl



An Investigation into Possible Attacks on HTML5 IndexedDB and their Prevention

S Kimak

PhD

2016

An Investigation into Possible Attacks on HTML5 IndexedDB and their Prevention

Stefan Kimak

A thesis submitted in partial fulfilment of the
requirements of the University of
Northumbria at Newcastle for the degree of
Doctor of Philosophy

Research undertaken in the Faculty of
Engineering and Environment

February 2016

Abstract

This thesis presents an analysis of, and enhanced security model for IndexedDB, the persistent HTML5 browser-based data store. In versions of HTML prior to HTML5, web sites used cookies to track user preferences locally. Cookies are however limited both in file size and number, and must also be added to every HTTP request, which increases web traffic unnecessarily. Web functionality has however increased significantly since cookies were introduced by Netscape in 1994. Consequently, web developers require additional capabilities to keep up with the evolution of the World Wide Web and growth in eCommerce. The response to this requirement was the IndexedDB API, which became an official W3C recommendation in January 2015. The IndexedDB API includes an Object Store, indices, and cursors and so gives HTML5 - compliant browsers a transactional database capability. Furthermore, once downloaded, IndexedDB data stores do not require network connectivity. This permits mobile web- based applications to work without a data connection. Such IndexedDB data stores will be used to store customer data, they will inevitably become targets for attackers.

This thesis firstly argues that the design of IndexedDB makes it unavoidably insecure. That is, every implementation is vulnerable to attacks such as Cross Site Scripting, and even data that has been deleted from databases may be stolen using appropriate software tools. This is demonstrated experimentally on both mobile and desktop browsers. IndexedDB is however capable of high performance even when compared to servers running optimized local databases. This is demonstrated through the development of a formal performance model. The performance predictions for IndexedDB were tested experimentally, and the results showed high conformance over a range of usage scenarios. This implies that IndexedDB is potentially a useful HTML5 API if the security issues can be addressed.

In the final component of this thesis, we propose and implement enhancements that correct the security weaknesses identified in IndexedDB. The enhancements use multi-factor authentication, and so are resistant to Cross Site Scripting attacks. This enhancement is then demonstrated experimentally, showing that HTML5 IndexedDB may be used securely both online and offline. This implies that secure, standards compliant browser-based applications with persistent local data stores may both be feasible and efficient.

Table of Contents

ABSTRACT.....	I
TABLE OF CONTENTS	II
LIST OF FIGURES.....	V
LIST OF TABLES.....	VI
LIST OF EQUATIONS.....	VII
GLOSSARY OF TERMS	VIII
DECLARATION	IX
CHAPTER 1. INTRODUCTION.....	1
1.1. BACKGROUND OF THE RESEARCH.....	1
1.2. MOTIVATION	10
1.3. OBJECTIVES.....	11
1.4. SCOPE OF RESEARCH AND TIMELINESS OF RESEARCH TOPIC	12
1.5. CONTRIBUTIONS AND IMPACT.....	12
1.6. ORGANISATION OF THE THESIS	12
1.7. SUMMARY	13
CHAPTER 2. LITERATURE REVIEW	15
2.1. INTRODUCTION.....	15
2.2. RELATED WORK.....	15
2.3. HTML5 FUNCTIONALITY.....	17
2.4. SERVER-SIDE DATABASES.....	20
2.5. CLIENT-SIDE WEB DATABASES	21
2.6. DIFFERENCE BETWEEN CLIENT AND SERVER-SIDE DATABASES	26
2.7. DIFFERENCES BETWEEN SQL AND NoSQL DATABASES.....	26
2.8. INDEXEDDB	27
2.9. SECURITY ISSUES	31
2.10. ONLINE AND OFFLINE ATTACKS	34
2.11. CRYPTOGRAPHY	42
2.12. BIOMETRICS AND MULTIFACTOR AUTHENTICATION (MFA).....	46
2.13. WEB SOCKETS	47
2.14. CONCLUSION	48

<u>CHAPTER 3. A PERFORMANCE MODEL FOR CLIENT-SIDE DATABASES.....</u>	<u>50</u>
3.1. MOTIVATION	51
3.2. QUEUING MODEL.....	52
3.3. DATABASE REPLICATION	53
3.4. DATABASE FRAGMENTATION	54
3.5. PERFORMANCE FACTORS.....	54
3.6. STRUCTURE OF TESTED DATABASES	56
3.7. DEFINE THE PERFORMANCE MODEL BASED ON THE QUEUING MODEL	57
3.8. EXPERIMENT VALIDATION	61
3.9. EXPERIMENTAL EVALUATION OF MODEL.....	62
3.10. CONCLUSION	67
<u>CHAPTER 4. A SECURITY INVESTIGATION OF INDEXEDDB.....</u>	<u>68</u>
4.1. BROWSER SECURITY EXPERIMENTS	68
4.2. MOBILE DEVICES SECURITY EXPERIMENTS.....	79
<u>CHAPTER 5. BROWSER-BASED LOCAL STORAGE SECURITY MODEL (BLS).....</u>	<u>89</u>
5.1. INTRODUCTION.....	89
5.2. BACKGROUND	90
5.3. ALGORITHM	91
5.4. IMPLEMENTATION	94
5.5. EVALUATION.....	99
5.6. CONCLUSION	101
<u>CHAPTER 6. CONCLUSIONS AND FUTURE WORK.....</u>	<u>102</u>
6.1. CONCLUSIONS	102
6.2. FINDINGS	104
6.3. FUTURE WORK	105
6.4. EXTENSIONS TO THE PERFORMANCE MODEL	107
6.5. INDEXEDDB USAGE STATISTICS	108
6.6. SUMMARY	108
<u>REFERENCES</u>	<u>110</u>
<u>APPENDIX A: LIST OF PUBLICATIONS</u>	<u>132</u>
<u>APPENDIX B: FIREFOX EXTENSION OF ENCRYPTION LIBRARY</u>	<u>133</u>

**APPENDING C: AN EXPERIMENTAL ANALYSIS AND POSSIBLE SOLUTION FOR THE
CROSS SITE REQUEST FORGERY ATTACK..... 139**

**APPENDIG D: AN INVESTIGATION INTO POSSIBLE ATTACKS ON HTML5 INDEXEDDB
AND THEIR PREVENTION 147**

**APPENDIG D: AN INVESTIGATION INTO POSSIBLE ATTACKS ON HTML5 INDEXEDDB
AND THEIR PREVENTION 148**

**APPENDIX E: PERFORMANCE TESTING AND COMPARISON OF CLIENT SIDE
DATABASES VERSUS SERVER SIDE 153**

**APPENDIX F: SOME POTENTIAL ISSUES WITH THE SECURITY OF HTML5 INDEXEDDB
..... 160**

**APPENDIX G: THE ROLE OF HTML5 INDEXEDDB, THE PAST, PRESENT AND FUTURE
..... 167**

**APPENDIX H: HTML5 INDEXEDDB ENCRYPTION: PREVENTION AGAINST POTENTIAL
ATTACKS 174**

List of Figures

Figure 2-1 HTML5's IndexedDB functionality	18
Figure 2-2 Cookie setup.....	25
Figure 2-3 IndexedDB structure	30
Figure 2-4 Same Origin Policy.....	33
Figure 2-5 Cross origin resource sharing.....	34
Figure 2-6 XSS attack explained	35
Figure 2-7 Man in the middle attack	38
Figure 2-8 RSA Encryption and decryption process	43
Figure 2-9 AES Encryption and decryption process.....	43
Figure 2-10 Principle for saving a fingerprint scan.....	46
Figure 3-1 Queuing model Diagram – Define the structure	55
Figure 3-2 Performance testing: Insertion of records into database (IndexedDB in Firefox and Chrome).....	64
Figure 3-3 Insertion of records into database (IndexedDB, WebSQL, Local Storage, Mysql) with predictions	66
Figure 4-1 Exported deleted database file.....	73
Figure 4-2 The physical address, and data in database file.....	74
Figure 4-3 Proposed Encryption Library	77
Figure 4-4 Proposed Encryption Library for mobile device	83
Figure 4-5 Logical extraction, view in XRY.....	84
Figure 4-6 Logical extraction, view in XACT	85
Figure 4-7 Web Application on mobile	86
Figure 4-8 Restored file from mobile running on desktop computer.....	87
Figure 5-1 Overall structure of the proposed security model.....	96
Figure 5-2 Insertion of data into IndexedDB with and without encryption in Firefox browser	100

List of Tables

Table 3-1 Waiting time variables.....	58
Table 3-2 Average waiting time in the queue.....	58
Table 3-3 Data transfer variables	59
Table 3-4 The table is showing insertion times results of tested databases	65
Table 5-1 List of possible consideration of JavaScript encryption libraries	90

List of Equations

Equation 3-1 Calculating queuing system waiting time	58
Equation 3-2 Calculating average waiting time in the queue	58
Equation 3-3 Calculating the minimum access time to total data	59
Equation 3-4 Calculating the data transfer time	59
Equation 3-5 Disk speed calculation	60
Equation 3-6 Disk latency calculation	61
Equation 3-7 Physical disk performance.....	61
Equation 5-1 Big O Notation	100
Equation 5-2 Big O Notation calculating time	100

Glossary of Terms

ACID	Atomicity, Consistency, Isolation, Durability
API	Application Programming Interfaces
BLOB	Binary Large Object
CPU	Central Processing Unit
HTML	Hyper Text Markup Language which consists of markup tags (Willard, 2013)
HTTP	Hyper Text Transfer Protocol - let know the web browser that a requested URL address is a Internet address
InnoDB	Storage engine for MySQL database
Local Storage	Local storage is browser-based database which has advantages over the use of cookies (stored amount of data).
MyISAM	Storage engine for MySQL database
NoSQL	Not only SQL
OWASP	<i>“Open Web Application Security Project is a worldwide free and open community focused on improving the security of application software.”</i> (OWASP, 2010)
RDBMS	Traditional relational database management systems
SATA	Serial ATA, is a computer bus interface
SOP	Same origins policy
SQL	Structured Query Language
URL	Uniform Resource Locator – it is an address where the page resides on the Internet.
W3C	The World Wide Web Consortium
WebSQL	Structured database, which uses standard SQL syntax for storage.

Declaration

I declare that the work contained in this thesis has not been submitted for any other award and that it is all my own work. I also confirm that this work fully acknowledges opinions, ideas and contributions from the work of others.

Any ethical clearance for the research presented in this thesis has been approved. Approval has been sought and granted by the Faculty Ethics Committee / Department of Engineering and Environment on 05/07/2013.

I declare that the Word Count of this Thesis is ...36579..... words

Name:

Signature:

Date:

Chapter 1. Introduction

Introduction

This chapter outlines the background to the study and the importance of new technologies; this is presented to serve as a basic foundation for the study. Additionally, we present the arrangement of the overall thesis, along with the objectives to be completed.

1.1. Background of the research

This thesis attempts to answer several questions. Firstly, what is HTML5 IndexedDB; where did the technology come from and what has motivated its development; what is its current status and what is inhibiting its take-up; and finally, what is the future of HTML5 IndexedDB?

HTML5 is the latest W3C standard for the language in which web pages are written and for Application Programming Interfaces (API) that are expected to be provided by a supporting web browser. The motivation behind 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 process 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 page refreshes. Consequently, major browsers now support the majority of the new HTML5 components and API. Therefore, HTML5 browser-based storage may contain stored data from online services that utilise the new functionality of HTML5 (Naseem and Majeed, 2013). The process of accessing this data might be slow in some cases, due to network latency or database query process (Zhanikeev, 2013). 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 (De Ryck et al. 2011).

Web browsers store history and other data using cookies on the client computer, which is considered attractive for marketing purposes. The importance of being browser-based is critical for developers, as modern browser-based web applications are able to store large amount of data and access this faster than any server-side database. Consequently, HTML5 opens up entirely new security challenges and issues (Anttonen et al. 2011). User information is tracked on every move on the Internet (Castelluccia, 2012; Atterer et al. 2006): eCommerce sites store customer details, orders and saved products, and web

sites store cookies on user computers to track returning customers. The data can be used later for marketing purposes and for targeting new customers. Sometimes consumers and the general public do not realise the quantity of personal data that is shared over the World Wide Web (WWW) (Rosenfeld and Morville, 2006) and how the data can be used or misused. Data privacy and information leakage is, therefore, a serious concern (Ruiz et al. 2015).

HTML5 is not supposed to replace Flash videos, but extends the functionality, which works on any device without the need of installing additional software. This means that the customer will be able to stream videos from native Web browser. Online videos are suggested to increase on online stores popularity and sales, where video has evolved into a powerful marketing tool that online retailers can prospect from (Xu, 2015). Videos are gaining on popularity, where they influence online reviews on customer perceptions and decisions to purchase products.

In 2009 and 2010 large corporations as Apple stopped implementing Adobe Flash to IOS devices, saying that will be instead using only HTML5 standard for mobile devices (Prince, 2013). This has led to a dramatic expansion of HTML5 usage over Adobe Flash by developers.

1.1.1 IndexedDB – the Past

In this section, we consider the drivers behind HTML5 IndexedDB, that is, why the technology was considered at all and what motivates current standards? We proceed as follows. Firstly, we provide an overview of the status of eCommerce, with particular attention paid to its mobile variant, mCommerce. Then we consider browser-based cookies that are IndexedDB's intellectual antecedents.

The term 'eCommerce' began to be used widely in early 2000 and is defined as commercial transactions conducted electronically on the WWW, such as purchasing goods and services online. Business worldwide prosper from eCommerce, because is fast-growing method for trading (Chuang et al. 2007; Pool et al. 2006). 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 retails sales increased to 40% (eMarketer, 2014). In 2012, eCommerce accounted for 18% (£492 billion) of UK business turnover. In 2012, 21% of UK businesses made eCommerce sales to their own country, 9% to EU countries and 7% to non-EU countries – based on Office of National statistics (ONS) data (Jones, 2014). Of the UK's total GDP of £1.45 trillion, the Internet value chain represented 2.6% and the eCommerce generated a further 3.1%. The UK

claims 57 million Internet users and has a penetration rate of 89%. Users usually spend nearly an hour per week browsing eCommerce stores and buying goods online. The use of eCommerce, by both organisations 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. The existence of online stores eliminates the third-party costs required by wholesalers and distributors and also removes the overheads of physical shops. Both of these aspects lower operational costs. eCommerce stores also provide search functionality to find the exact product a customer is looking for. Customers can easily browse through large amounts of products and services (Reynolds, 2000). eCommerce has been expanded to the business to business (B2B) (Vargo and Lusch, 2011) and business to consumer (B2C) (Ta et al. 2015) markets. Many retailers took the step of investing in online sales, which targets more customers in categories such as electronics, books and transport. eCommerce also provides retailers a global market opportunity with minimal initial investment (Xiaojing et al. 2012). Consumers can also see prices, allowing simple price comparison, and can then place orders quickly. eCommerce stores allow the customer to add products to their wish list, which can be sent to friends or family to be paid for.

Consumers can check existing online product reviews and also compare prices for best offer before buying any goods or services. Some eCommerce stores provide a video review of products, where the customer can observe the product without the need to leave the house (Sunil, 2015). One of the most important advantages of eCommerce stores is global market and creating new business opportunities (Wang et al. 2015). Online stores are available to everyone, everywhere. For most businesses, eCommerce is an excellent alternative supply channel that is not only cost-effective but continuous and extensive in reaching consumers directly.

The Internet helps companies to engage in eCommerce by collecting, storing and exchanging personal information obtained from visitors to their websites (Boritz et al. 2008). eCommerce stores can target customers in many ways; the most widely used are email and online ads which have a cost advantage over offline stores where printed flyers must be produced. eCommerce can target a larger number of customers in a shorter period of time, as everything is done over the Internet. With the growth in popularity of social media, eCommerce retailers took advantage of this to attract new customers. Additionally, online retailers use online customer buying habits to target new and existing customers with social media advertisements and special offers (Huang and Benyoucef, 2013). Since the late 1990s, it was foreseen that every step on the Internet

could be traced and that this information might be stored (Gehling and Stankard, 2005). Information from web browsers is stored in computer history, and eCommerce sites store user preferences and shop orders to understand their customers better and target them with advertising related products. eCommerce stores use cookies (Stalker et al. 2004) to store customers' preferences, which makes the online buying experience more convenient. Web technology also allows retailers to track customer preferences and to deliver individually tailored marketing (Zhao and Lin, 2014). Security experts predicted that user information on the Internet would be valuable for marketing and targeting customers with special offers (Wang and Zhang, 2010; Gómez and Lichtenberg, 2007).

HTML was first focused on the desktop computer, for in 1993 when the web started, 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, mobile phones and tablets combined now account for 38% of all web pages served around the world (StatCounter, 2016). Smart phone user penetration was 9.6% in 2011 and by 2015, it was 28% (Statista, 2016), while in UK the mobile penetration rate is 72%. The UK's Internet ecosystem is worth £82 billion a year, with mobile devices connections accounting for 16% of this. mCommerce sales will continue to grow where by 2019 it is predicted to reach £37 billion and drive over 60% of online retail sales. Since 2007, mCommerce sales have grown rapidly from less than 5% to 21% of all sales in 2015 (eMarketer, 2014), which has opened a new market for online stores. The latter needed to change their strategy and target the mobile market.

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

Cookies are small quantities of data stored by websites in the client browser and sent to that web site with each Hypertext Transfer Protocol (HTTP) or Hypertext Transfer Protocol Secure (HTTPS) request. Cookies were introduced by Netscape Communications the in 1994. Cookies allow user to store their sessions and state with a website. Later they provided functionality like customer shopping carts and the recognition of returning customers (Uehara et al. 2001), whereby online stores can display recommended products and offers.

The problem with cookies was with implications for privacy, in that third-party applications could potentially steal information from cookies (Ayenson, 2011). There are also several security issues with cookies, such as cookie poisoning (Buja et al. 2014; Saha and Das, 2012) and cookie injection (Choi and Gouda, 2011). Cookie poisoning attacks involve the modification of the contents (user IDs, passwords, account numbers,

time stamps) in order to bypass security mechanisms (Saha and Das, 2012). The cookie poisoning attacks allows attacker to impersonate as the victim with the purpose to access information about the victim. Cookie injection attacks operate by injecting a cookie string or code into the HTTP header to change its execution, which can lead to SQL injection (Appelt, 2014; Shar and Tan, 2013; Lee et al. 2012).

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 kind of information on a desktop computer or mobile device (Summers et al. 2013; Hayes, 2012). The Cookie Law was designed to protect online privacy of consumers. When a consumers entered a website or eCommerce site and cookies has been used, the website was required to alert them of how information about them is collected and used online, and to give them a choice to allow this or not. From 2015 the websites are not obligated to alert the visitor about storing cookies.

Cookies are limited in size to 4KB and therefore are rarely used to store site-specific information directly. Rather, a typical cookie will store a unique database key which 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/purchase history, and preferences.

1.1.2 IndexedDB at present

Browser storage has been proposed to extend cookie functionality by providing web developers and web applications with better alternatives to store data locally. With browser-based storage, eCommerce 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.

With browser-based storage, eCommerce sites can be stored locally and used offline. The user will have the ability to add products to the shopping cart, even if the network connection is down. The advantage of using local storage is with mobile devices, where network connection and data quotas are concerns. Local storage could be used when a service is limited or allows only a certain number of calls per hour, but the data does not change that often. A web application could store the information in local storage and prevent users from using the limit (Karthik et al. 2014). Online stores could save new images every couple of hours, rather than every minute, which would improve the bandwidth. Local storage keeps users from being banned from services and also means that when a call to the application program interface (API) fails, the user will still

have information to display. For example, the cart can be stored locally and synchronised with the eCommerce site when network connectivity is restored.

The main problem with HTTP protocol as the main transport layer of the web is that it is stateless (Fielding and Reschke, 2014). This means that when an application is closed, its state will be reset the next time it is opened. If a desktop application is closed and then re-opened, its most recent state is restored. Local storage has advantages over cookies which includes better performance, storing larger amount of data, storage across multiple windows in web browser and data persist even after the web browser is closed (Ayenson, 2011). Therefore, local storage provides functionality similar to desktop applications, where the state is persistently stored.

In 1995, Netscape's vision of the future was to run multimedia applications, spreadsheets and word processing programs from the web browser. Netscape's main product was a browser, which was written to run across multiple operating systems (Windows, Linux and Macintosh). The vision was that applications would run on top of any Operating System with their sets of APIs, so that third-party application developers would not need to worry about the underlying Operating System and computer hardware. Today, this vision is a reality, where applications like YouTube and Facebook run from browsers. Also, Chrome introduced its new Operating System, which is web browser-based, partially based on Netscape's vision. The application can be run as a web application but developers have introduced applications that can be run offline as well (Gihan et al. 2011) .

As the HTML5 standard evolved, new browser-based storage was introduced and was able to store larger amounts of data. Additionally, it provided the non-functional requirement such as speed, because the stored data was not transmitted with every HTTP request. Non-functional requirements are not used to perform a specific function, sometimes also referred as quality factors or attributes (Chung et al. 2000).

HTML5 provides two new featured to store data locally. First, browser-based storage is called local storage. It allows the storing of information locally within web browsers in object stores (SQL databases have tables), that persists on disk. The storage is limited to 5MB and the stored data is in name/value pair.

HTML5 browser-based storage technology is called IndexedDB, known previously as WebSimpleDB. It is a Not only SQL (NoSQL) (Kuznetsov and Poskonin, 2014; Atzeni et al. 2014; Zachary et al. 2013; Strozzi, 1998) key-value asynchronous browser-based storage.

IndexedDB API offers fast access to unlimited amounts of structured data. The current state of IndexedDB is regarded as insecure, because security was not considered

in its specification. As documented in Chapter 4, we have used forensic tools and identified security issues with IndexedDB.

NoSQL is the database solution that is not relational or object orientated. NoSQL does store data in key/value format. The database can handle a large amount of data, where the relational model is not needed.

IndexedDB came from the W3C specification of implementing web storage into web browsers in 2009. IndexedDB is a persistent client-side database implemented into browsers and is an alternative to already deprecated WebSQL. Mozilla and Microsoft supported the change, while Oracle's Berkley DB mostly influenced this. The application uses local data stored on a client system (Casario et al. 2011). It caches large amounts of data from server to client-side using JavaScript Object Stores, equivalent to tables in relational databases.

Files and data stored by the browser are retained on the user's hard drive storage system. The client-side database, IndexedDB, stores the data, even when the browser terminates. IndexedDB is a persistent client-side database, which means that the data can be retrieved even 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 Binary Large Object (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; the performance of storing a file in IndexedDB is just as good as storing it in a file system. 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 Operating System file; therefore, IndexedDB is just as fast as a file system. The Firefox IndexedDB implementation is even smart enough that if storing the same BLOB multiple files, it creates only one copy. Writing further references to the same BLOB simply adds to an internal reference counter. This is completely transparent to the web page; it writes data faster whilst using fewer resources. Browser-based storage as IndexedDB (W3C, 2015) 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 using APIs.

IndexedDB caches large amounts of data from server to client-side using JavaScript Object Stores, equivalent to tables in relational databases. Files and data

stored by the browser are retained on the user's hard drive. The client -side database, IndexedDB, stores the data, even when the browser terminates.

Web applications can use browser-based storage without the need for network connection (Gihan et al. 2011). The HTML5 standard provides all of the functionality where data can be stored on client machines and accessed at any time. An important aspect of HTML5 is that 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. This reduces the barrier to entry for new customers, since clients can begin taking advantage of the web applications just by visiting the relevant web site.

IndexedDB extends local storage by providing web application offline storage. With this functionality, web applications such as eCommerce stores can take advantage of storing customer preferences, which are not sent with every HTTP request. Consequently, the request and response traffic will decrease and the preference or information will be accessed only when requested. An important aspect of HTML5 is that web applications can run offline using local storage. This means that client data will be stored on the client-side and accessed any time the application requires it. Offline storage and cached pages provide a better experience for users where the network latency is minimal.

With the new HTML5 IndexedDB functionalities, new security issues arise because it increases access to the computer's resources. One of the biggest disadvantages is that the new standard does not provide additional security. HTML5 video and audio support replaces third-party applications as Adobe, which closes a common attack vector with FLASH application or plug-ins. Additionally, HTML5 provides much more access to computer resources, which includes local storage, and therefore provides new opportunities for potential attacks.

The problem with the current browser-based storage such as IndexedDB is that there is concern over reading offline data that another application has stored on the client-side.

The security mechanism in web browsers that prevents web applications from reading data from other sources is known as Same Origin Policy (SOP) (Gollman, 2011, Ss. 2.3.2). The data can be accessed only if the hostname of web application (`www.example.com`), port number (web browsers run on port 80) and protocol (HTTP or HTTPS) match against the origin record. The same principle applies when web application wants to access data stored on users device (browser-based storage, history).

When the origin of the data and requesting origin match, the access is allowed.

From the experiments conducted, we have found that the storage of data in an unencrypted state is not the only problem. Browser-based storage have another issue, where the data is not fully deleted from the hard drive. With the help of forensic tools, it is possible to restore deleted data from desktop and mobile devices.

The issue of restoring deleted data merely extends the security concern of storing data in an unencrypted state, where the attacker could get multiple versions of browser-based local storage. The deleted data persists on hard drive and when a request to delete the data is executed, the data is just marked as deleted but occupies the unallocated space on the device. New requests to store data to browser-based storage will assign new partition space and the old data will persist on the hard drive – it will be not overwritten.

1.1.3 IndexedDB – future

The future of IndexedDB is to support secure browser-based offline usage. Existing browser-based storage is not popular with web developers, as they face several problems. The first is the complexity of code, where the developers need extra time to understand the program structure. Yet, there are many examples to help developers 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 cannot be recommended for the storage of personal information. This makes it limited in functionality. As with data stored on a desktop, or mobile device in an unencrypted state, an attacker can access data without bypassing any protection. With a Cross Site Scripting attack (XSS) (Elhakeem and Barry, 2013; Wassermann and Su, 2008; Vogt et al. 2007; Di Lucca et al. 2004), such as hidden in an email link, an attacker could find the stored data. Hence, IndexedDB is inherently vulnerable to such attacks.

Security flaws are inevitable when considering web applications and storage of information, because no technology is 100% resistant to vulnerability.

By design, browser-based storage security is a concern, but it can be corrected. The method is to use client-side encryption, which will mean that browser-based storage is at least as secure as the sever-side.

We have proposed a security model that 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 stored on a client's machine. 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. The data stored locally will be safe even should an attacker gain access to the database, because the stored data will be encrypted.

The security model is built around an encryption framework that will help to secure the data. The encryption framework consists of an encryption library that is implemented into a browser but does not provide a full security protection. This thesis argues that such protection is not achievable in a single machine, since any single browser could be the target of an XSS attack. Therefore, external browser functionality needs to be implemented. In addition the encryption library, a multifactor authentication (MFA) or two-factor authentication (2FA) (Ss. 2.12) has been implemented.

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 added to the browser in a form of an extension. The proposed security model extension covers the security issue IndexedDB has by design. Also, the implemented security model fulfils the security requirements.

1.2. Motivation

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 desktop applications can be used. That is, client-side process will be able to avoid the ineffectiveness and network connectivity issues found in server-side applications. Consequently, major browsers now support the majority of new HTML5 components and APIs. Therefore, HTML5 browser-based storage may well contain stored data from online services that makes use of the new functionality of HTML5 (Naseem and Majeed, 2013). The process of accessing this data might, in some cases, be slow because of network redundancy or database query process (Zhanikeev, 2013). 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 (De Ryck, 2011). Consequently, HTML5 opens up entirely new security challenges and issues (Anttonen, 2011).

User information is tracked with every move on the Internet, and eCommerce sites store customer details, orders and saved products (Castelluccia, 2012). Sites store cookies on user computers to track returning customers. The data can be used later for marketing purposes and to target new customers. Sometimes the end customer does not even realise how much personal data is shared over the Internet and how the data can be used. Data privacy and information leakage is a pressing concern that needs to be addressed. Web

browsers store history, cookies and data in local storage on the user's computer, which is considered attractive for marketing purposes.

Using browser-based storage is important for developers, because modern web applications are able to store large amounts of data and access it much faster than any server-side database. The importance of HTML5 (Anthes, 2012) browser-based storage components would come in their benefit to eCommerce, as the customer could use online store without an Internet connection to browse products and add them to their shopping cart.

The following research questions are addressed in this thesis:

Considering the speed performance of databases, does offline storage performs faster than online services?

How secure is running standalone applications in a web browser with browser-based storage in its current state and development?

Considering security by design, can browser-based storage ever be secured against online and offline attacks?

1.3. Objectives

The objectives of this project are:

1. Develop a performance model for browser-based storage which estimates the time to perform read and write of data.
2. Perform experiments with developed performance benchmark to compare the results. Provide a conclusion based on the results and comparisons, which shows if there is a case for browser-based storage.
3. Critical investigation on security issues of IndexedDB.
4. Critically evaluate security on desktop and mobile devices based on previous investigation.
5. Develop a browser-based storage security model, this will resolve the identified security issues.

1.4. Scope of research and timeliness of research topic

IndexedDB is still in the process of development; therefore, the problem is relevant. As the IndexedDB database is not yet widely adopted and used, many issues and problems are still not identified. As of early 2015, IndexedDB has gained a W3C Recommendation, which means that is officially a web standard. The scope of the research is to develop HTML5 applications to run IndexedDB as their backend database, undertake comparison of performance and find possible security vulnerabilities.

1.5. Contributions and impact

This thesis presents a novel synthesis and enhanced security model for browser-based storage. We believe that existing web security models do not protecting the end user data, and our investigation intends to confirm this belief. The contribution consider whether browser-based storage is still resistant to attack when a client-side security model is applied. The data stored locally will be safe and even if an attacker accesses the stored data, it will be encrypted. Furthermore, we implemented a web browser extension with a client-side encryption library that will help to secure the stored data on client machines. The security of the end user is becoming an everyday concern, as many vulnerabilities exists with web applications, such as XSS, SQL injection, Cross - Site Request Forgery (CSRF) (Willis, 2009; Stuttard and Pinto, 2011; Burns, 2005) and JavaScript attacks. The main concern is how the data is secured on server or client-side. In the case of client-side, there are many protections against attacks; but with new technologies, these protections are not enough. We demonstrate the effectiveness of the current client-side protections. The second contribution is ineffectiveness of IndexedDB browser-based storage, whereby experiments show security design flaws. The third contribution is a performance model to show that browser-based storage performs faster than server-side databases. This model was also confirmed by our benchmarking experiments.

1.6. Organisation of the thesis

This thesis argues that current HTML5 browser-based storage is not secure by design. We have used an experimental research methodology to prove this and are currently developing a supporting theory.

Chapter 2 presents an overview of existing solutions for browser-based local storage and security issues associated with it. Additionally, we have gathered background information of IndexedDB and performed initial investigations on possible

security issues within using and storing client – side data. The initial investigation was published in 2012 and that paper is in Appendix D.

Chapter 3 designs a speed model to demonstrate the effectiveness of using browser-based storage (IndexedDB) in comparison to other browser-based storage or server-side databases. We have also compared the results of browser-based storage to the MySQL server-side database. The benchmarking experiments and the results were published in 2013 and that paper is in Appendix E.

Chapter 4 have investigated browser-based storage and browser security. The investigation was focused on data storage for the browser-based databases in the Operating System file. To help analyse the results, a forensic tool called EnCase (Encase, 2004) was used. The experiment concentrated on investigating how IndexedDB is saving the data and how it can be retrieved after deletion. The desktop forensic investigation experiment and results were reported in 2014 and the paper is in Appendix F. Additionally, investigations on mobile devices based on steps included in the desktop forensic investigation were performed. This was conducted on an Android mobile phone, with the help of the mobile forensic tool, XRY (XRY, 2015) and EnCase 7 (Bunting and Wei, 2006; EnCase, 2004).

Chapter 5 presents a novel synthesis and enhanced security model for browser-based storage. This model extends the current web browser security model and an overview of the architecture is shown in Figure 5.1. Furthermore, we have implemented a web browser extension with a client-side encryption library, which will help to secure the stored data on client machines; the steps are described further in Chapter 5. We are also proposing a new client-side security model that will extend the current web browser model and we will evaluate its effectiveness.

Chapter 6 concludes the thesis by looking at each chapter. Also future work will be discussed in areas not covered in thesis, which are out of the scope. The last part of the chapter will make main conclusion to summarise the whole thesis and covered contributions.

1.7. Summary

Based on these findings, it may be concluded that there is a case for browser-based databases. The experiments in this thesis have demonstrated that all data stored on the user's machine is in an unencrypted state. When a web application is vulnerable to online attacks stored data is could be compromised, retrieved or deleted by an attacker. The same applies to offline attacks, where the attacker could get a physical access to device

where the data is stored, such example implies a scenario when a user lost a mobile device.

As a part of thesis a JavaScript encryption framework has been implemented, which is part of the security model implemented into the browser in a form of an extension. This extension covers the security issue that IndexedDB suffers from by design. Also, the security model implemented fulfils the security requirements. The model extends the current security model which is insufficient for complex HTML5 browser-based applications. Based on the evaluation, we can suggest that the model will correct the defence ineffectiveness by protecting client-side data. In order to decrypt it and obtain private data, an authentication and private key is required.

Browser-based databases face problems that prevent them from being widely used. The first is the complexity of code, where the developer needs extra time to understand the structure. However, there are many examples that can help developers to start implementing browser-based storage in their web applications.

The second issue with browser-based storage is security. Browser-based storage is not secure by design, which means that the storage cannot, or is not recommended to, be used to store personal information. This limits potential functionality. Despite the issues and concerns about storing data locally, browser-based storage has the potential to be used widely. The main advantages are performance speed, being cross platform (desktop, mobile, tablet), browser availability and offline usage.

Chapter 2. Literature Review

2.1. Introduction

This chapter reviews the research related to the new HTML standard functionalities and possible security issues which might arise. Section 2 identifies related work and work which is the thesis build on; section 3 introduces the new functionalities; section 4 identifies and discusses the research closely connected with the subject of this thesis - while provides the main insight into security within the research.

There is a trend and a demand for moving traditional desktop applications, such as word processors and spreadsheets, to the Internet: and replace them with web applications. This implies a trend to where the only client software that most computers and mobile devices will need in the future is a Web browser (Stuttard and Pinto, 2011). However, there will most likely always be places without Internet access, and it should be possible to use these web applications regardless of this, so offline forms (many of which currently exist) of these applications are an important part of this development. To enable secure use of offline web applications – even when online - security testing of these applications is important: they should be made as secure as possible.

When looking at this from a security point of view, there are certain disadvantages with Web applications. Making web applications secure is even more important than with other software, as they are exposed to millions of users on the Internet (Meucci et al. 2008).

2.2. Related work

The web browser is arguably the most security-critical component in our information infrastructure. It has become the channel through which most of our information passes. Integrated database in browser does not have any kind of protection, similar to server side databases, where an authentication is required to access the stored data. At the moment browser based databases have only Same Origin Policy (Ss. 2.3.2), which can be bypassed with online attacks (Ss. 2.3.2) and the attacker could get all data from database without any kind of authentication or client authorisation. Most of the studies into online web applications security and online attacks consider the fact, that user data is stored on web server rather than on client.

Several recent studies propose a new security model for browsers, but this do not apply to browser based databases.

The current studies on browser based databases security suggest, that securing web application against online attacks can also protect the end user data, as suggested by Weissbacher et al. (2015) and Gupta et al. (2016). This can be true in most of the cases, but data stored locally can be accessed physically, or the mobile device can be lost, which leaves all the data unprotected and vulnerable.

As discussed by De Ryck et.al (2011) improvements to browser security are required to protect the end user and stored data locally. The browser model is required to change, to progress with the new technologies. The authors also suggesting, that there should be a user authorisation for accessing local storage, and the SOP needs to be improved to prevent against attacks which can bypass the restriction.

It has been argued that HTML standard and Web has evolved and therefore current security measure are not sufficient to protect data stored on client (HTML5 storage). West et. al (2012) done some work on identifying security concern and suggest that it is necessary to constantly address modern security and privacy concerns through consistent updates of HTML specifications.

HTML5 local storage data residues can be accessed within the memory images, and the forensic investigation results by Matsumoto and Sakurai (2014) showed that the acquisition of Web Storage content on the browsers were possible and revealed its formats. Values of Web Storage was checked in the residuals that left by all of three web browsers (Chrome, Firefox, IE). They arguing that user with the knowledge of values will be able to find the location of the evidence to hint values.

Research has proposed a secure space for storage, with the attempt to secure user data. The approach of Jemel and Serhrouchni (2014) proposes that the browser devote to each user a secure space for data storage. Thus, all data will be stored safely in the client side before their synchronization over the different machines of the same user using the Cloud. The data protection can be applied either on PC or on smart-phone for mobile Internet application.

The security work of Kun and Yizheng (2014) investigates the security of HTML5 Client side storage, where they argue that local storage cannot be fully controlled by the server-side, which brings data security risk. The work presents test analysis of different approaches of storage and potential safety hazards.

The security model for browser based database is not sufficient to protect the end user data as suggested by Bugliesi et al. (2014) work. Therefore, this thesis proposes an additional security model, which will help to protect the data with both encryption and additionally with end user authentication.

2.3. HTML5 Functionality

The HTML5 standard provides new functionality that helps to develop better web applications. These functionalities are described in more detail with both their advantages and disadvantages. The security of HTML5 is an important aspect, and we will look at what types of attacks could be performed to bypass the existing security safeguards of the new functionalities.

2.3.1 Introduction

The motivation for the changes and enhancements embedded in HTML5 is that the web browser should be capable of running HTML5 client-side web applications (similar to current desktop applications). Traditional desktop applications, like word processors and spreadsheets, might be used as HTML5 client-side web applications. This means that the client will no longer need to install any software on the computer: only a Web browser will be needed (Stuttard and Pinto, 2011).

For example, many applications can run on multiple platforms e.g. tablets, mobiles and others. The problem for developers is that on mobile devices there can be multiple OSs: Android, IOS, Windows Phone. Each of the operating systems provides different programming languages/environments for development. Therefore developing an application to run on multiple platforms can be expensive and time consuming. HTML5 provides a functionality whereby an online application can be used on any platform and on various browsers. That way the developers need to develop the application only in one language and it can be used on multiple platforms, such as desktop, mobile and tablet. HTML5's browser-based database, IndexedDB provides a functionality whereby the web application can run offline so the end user does not need an Internet connection. The offline usage provides full functionality, but obviously is limited in some ways – e.g. where some data is required from 3rd party applications, such as payment gateways for eCommerce sites.

2.3.2 Client-side databases

Client-side data is passed to the browser's storage API which stores data on the local device (Xu et al. 2013). The importance of client-side application usage provides user experience and functionality of desktop applications (Maras et al. 2011), which can also be used in the same way on mobile devices (Asif and Krogstie, 2013). This application therefore can be customised to fit the needs and improve Web accessibility in web browsers (Garrido et al. 2013). Therefore an HTML5 browser client-side database may

well contain stored data from online services that make use of the new functionality of HTML5 (Naseem and Majeed, 2013). The process of accessing this data from online sources might in certain cases be slow because of network redundancy or the database query process (Zhanikeev, 2013). It is suggested that this new level of client-side data storage will mean that HTML5 enabled browsers will attract security violations (De Ryck et al. 2011). Consequently HTML5 opens up entirely new security challenges and security issues (Anttonen et al. 2011).

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 the client-side and accessed any time that the application requires it.

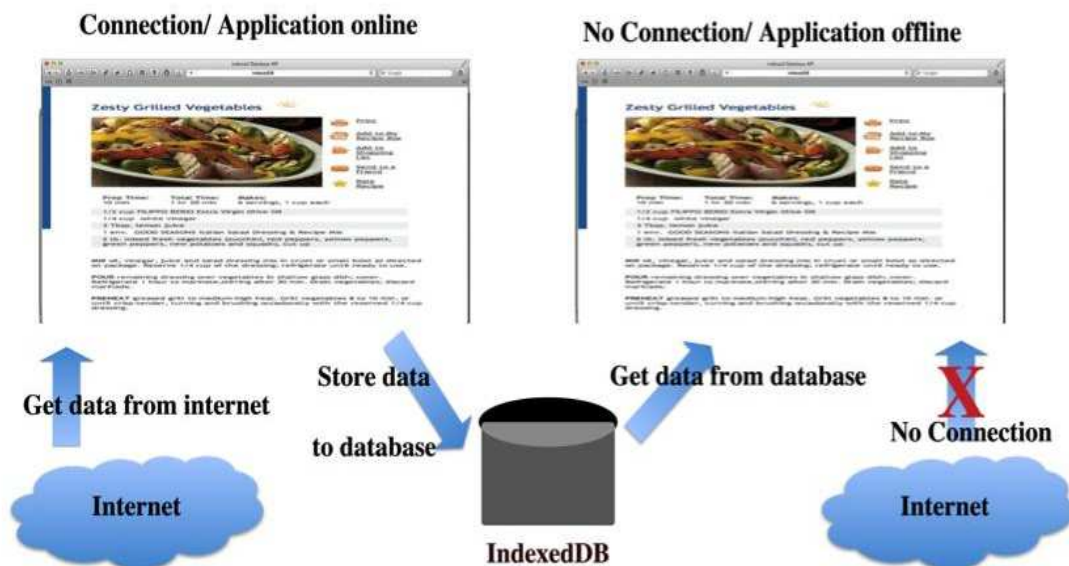


Figure 2-1 HTML5's IndexedDB functionality

When a client connects to an HTML5 web application for the first time, an API transaction will be created. The application will ask the client to store data locally. This data will be stored in a client-side database - IndexedDB. If a network failure occurs, the data from the database will be read and so the client will still be able to use the application. This means that an application can be run offline. Pictures and text from pages can be stored in IndexedDB.

The advantage of HTML5 applications as compared to desktop programs is that web applications do not require any installation or start-up configuration and will also run on any device which supports HTML5 - laptops, phones or tablets. This reduces the barrier of entry for new customers since clients can begin taking advantage of the web application just by visiting the relevant web site (Harjono et al. 2011). The benefits of client-side

storage include the ability to deal with connectivity failure: an application can be used even when a connection is not available. Offline content also allows access to, and creation or modification of, data stored locally that the application can then use entirely offline. Using this technology, websites behave like desktop applications; the application reloads the content instantly, and without needing to reload the page. The performance improvements include less bandwidth usage as data is stored on the client-side and the data is transferred only when the web application requires it (Hilerio, 2011).

Web-based software is increasingly popular as applications are constantly becoming available on the Web as services. This means that client software will increasingly be developed using web technologies (Taivalseeri and Mikkonen, 2011). Such applications and services consist of data and code which can be located anywhere in the world. This allows a wide range of applications to support multiple clients and share data worldwide. With the help of client-side storage, data can be periodically saved to the browser while the client completes it. After the data has been processed the information is then transmitted to the server. This will speed up application load time (Wisniewski, 2011). Further details and examples of existing client-side databases are described in section 2.7.

2.3.3 Advantages of new HTML5 functionalities

HTML5 provides build in video playback; this radically impacts on the use of third party programs such as Adobe Flash, Quicktime and Silverlight. HTML5 also offers offline storage, meaning that the user can load certain elements or the whole Web page without an active Internet connection. HTML5 will enable web designers to use cleaner code with semantic HTML5 elements.

One of the new functionalities of HTML5 is the new local storage feature. Local storage is an improvement over the storage facilities of cookies because it has better performance, meaning that the data is not transmitted in every HTTP request. Local storage can save larger amount of data, up to 5MB, where the stored data can be used across multiple windows and persist stored once the web browser is closed (Ayenson, 2011).

The new features are available cross browser and cross platform. One of the biggest advantages is the fact that it is mobile ready. Browsers on mobile devices fully support HTML5 therefore creating mobile ready projects easier, which works in the same way as on desktop computer.

2.3.4 Disadvantages of new HTML5 functionalities

One of the biggest disadvantages or disappointments is that the new standard does not provide any additional security (West and Pulimood, 2012). HTML5 video and audio support replaces third party applications such as Adobe; this closes a common attack vector via FLASH application and plug-ins (Eilers, 2012). On the other hand, though, HTML5 provides much more access to local resources, including local storage, and therefore opens new opportunities for attacks (Taivan et al. 2014).

2.3.5 Difference between HTML5 and HTML4

The differences include new functionalities, such as streaming video and audio without need for third party plug-ins such as flash, and local Web storage - which is a replacement for cookies – also there are new structural elements to replace div tags for creating page templates (Sharma, 2012). In addition, HTML5 allows the storage of data locally, on client-side. The stored data can be accessed to support web applications when requested (e.g., loading images stored on client-side). Stored data can even be accessed when the user does not have a network connection. HTML5 introduces new semantic (<header>, <nav>, <section>, <footer> etc.) and non-semantic elements <speed> and features that allow developers to improve interoperability, whereby the new functionalities can be used on multiple platforms.

2.4. Server-side databases

Relational databases (Martinez-Cruz et al. 2012; Ramakrishnan and Gehrke, 2002) are the most used type of database these days according to DB engines ranking. The most popular and widely used databases include Oracle, MySQL, MS SQL Server, PostgreSQL and MS Access. Relational databases are computerized programs used to store information in tables. These tables contain rows and columns used to sort and retrieve information. The rows and columns contain related information about the subject of the table. The database administrator can define the relationships among the various types of data. Relational databases require data to be entered as integers, strings or real numbers. This data must then be accessed through SQL queries (Conolly and Begg, 2004). The Entity Relationship (ER) model has been known for decades now, and is still working for most of the current scenarios.

A relational database-management system (RDBMS) includes a collection of data items organized as a table, with the columns representing data categories and the rows representing the data itself (Sumathi, 2007). Relational databases are good for managing

large amounts of structured, alphanumeric data. For example, companies use them to maintain records of transactions or personnel files. However, relational databases are inflexible because their only data structure is tables. And they work only with limited, simple data types, such as integers, and thus have had trouble handling complex and user-defined data types, including multimedia (Chengjiong, 2012).

2.4.1 MySQL

MySQL (Ullman, 2012; Gehani, 2011) is one of the most widely used Open Source Relational SQL database management system (RDBMS). MySQL is an RDBMS and is used for developing web-based software applications. The MySQL Database system can be used on client or server side. The database provides a wide range of APIs, provides different back ends (MyISAM and InnoDB), administrator management tools and can be set up on Windows, Linux or Unix environment (Kofler, 2001).

2.4.2 SQLite

SQLite is popular transactional SQL database engine. It is popular because it can be embedded into end-user programs such as browsers or mobile phone GUIs (Patil et al. 2012).

The main advantage of SQLite is its availability (it is used on Android for mobile applications and browsers and on iOS for mobile browsers). The main disadvantage of SQLite is that the W3C no longer support it, and browsers such as Firefox, have removed the SQLite support for their latest versions.

Embedding SQLite in web browsers has resulted in the addition of SQLite to the HTML5 Web Storage standard and, after some discussion, inside the W3C Web Applications Working Group (W3C, 2015).

Next we are going to look at client-side databases and the main reasons for using these rather than server-side databases (in situations where this is, indeed, appropriate).

2.5. Client-side Web Databases

In this section we compare the existing databases used for web development with the newly proposed databases. Also we look at how these new databases will impact the end user experience and a depth comparison of client-side as opposed to server-side databases.

2.5.1 WebSQL

WebSQL is a structured database, which uses standard SQL syntax for storage (West and Pulimood, 2012). W3C (2010) wrote that the WebSQL database API is off active maintenance. They cited the lack of independent implementations as being the reason - due to the fact, at least at the time, that most browsers relied on SQLite as the underlying database. The WebSQL database system brought a real relational database implementation to browsers. Data could be stored in a very structured way.

2.5.2 NoSQL

NoSQL (Not only SQL, see Kuznetsov and Poskonin, 2014, Atzeni et al. 2014, Zachary et al. 2013, Strozzi, 1998) is a database solution which is neither relational nor object oriented. NoSQL does store data in a key/value format though. Key/value means that the key is the identifier for the data and value can be data or pointer to file location. NoSQL store data in unstructured records, which means that the data can be (Atzeni et al. 2014). A NoSQL database can handle a large amount of data for which the relational model is not needed. They came to the use when the designers of web services with large numbers of users discovered that the traditional relational database management systems (RDBMS) are fit either for small databases with frequent read/write transactions or for large batch transactions with rare write accesses, but not for heavy read/write workloads (which is often the case for these large scale web services as Google, Amazon, Facebook, Yahoo and such)(Tudorica and Bucur, 2011).

NoSQL databases use various models - the key-value model being the simplest. Other models include ordered key-value, document full text search, graph and big table.

The main feature of NoSQL databases is the abandonment of the relational data model and SQL. NoSQL databases offer pieces of Atomicity, Consistency, Isolation, Durability (ACID) (Connolly and Begg, 2014) transactions and use distributed architecture (Kuznetsov and Poskonin, 2014).

Atomicity is related to transactions involving multiple separate pieces of data where either all of the pieces of data are committed or none are.

Consistency of a transaction in database is the requirement that on failure a new state is created and the data is returned to state before the failure.

The lack of support for ACID transactions leads to compromised consistency. Banking sites use consistency in their applications; therefore usage of NoSQL databases in that arena could be problematic (Shashank, 2011). On the other hand, NoSQL provides better performance and scalability.

Isolation is the requirement that a transaction must remain inaccessible from other transaction.

Durability is the condition that where any committed data remains in a valid and consistent state, even if unexpected failure or interruption occurs.

NoSQL databases performance of data processing is faster than relational databases (Leavitt, 2010) and this can be demonstrated, for instance by the comparison undertaken by Zachary et al. (2013) and Van der Veen et al. (2012). Also NoSQL databases are often faster just because their data models are simpler (Banker, 2010).

Data that is too large and complex to store, capture, analyse, process and understand using recent methods and available tools is referred as Big Data (Barbierato et al. 2014; Gudivada et al. 2014). NoSQL databases are used for Big Data, because of their indexing performance advantage, consistency, and single-digit millisecond latency at any scale (Barbierato et al. 2014; Gudivada et al. 2014). Additionally, another reason for NoSQL database usage for Big Data is the fact that their key/value data storage regime is often relevant to this arena (username/password, etc).

There are advantages to using NoSQL databases, but there are disadvantages and downsides to NoSQL as well. NoSQL databases face several challenges: for instance, overhead and complexity. Also they do not work with SQL queries, which means that they need to be manually programmed. In cases of simple tasks they perform fast, but it is time consuming to program for complex queries such as joins (Zachary et al. 2013, Leavitt, 2010).

ACID is supported by relational databases, while NoSQL databases have only partial or no support. Therefore, NoSQL databases do not offer the level of reliability that ACID provides. With additional programming NoSQL databases can apply ACID to data.

Most organizations are unfamiliar with NoSQL databases and thus may not feel knowledgeable enough to choose one or even to determine that the approach might be better for their purposes (Stonebraker, 2010). Unlike commercial relational databases, many open source NoSQL applications do not yet come with customer support or management tools. Each NoSQL database has its own set of Application Programming Interfaces (API), libraries and preferred languages for interacting with the data they contain.

Examples of document-oriented NoSQL database systems are as follows: MongoDB, Level DB, and BerkeleyDB (IndexedDB is based on Oracle BerkeleyDB (Brooks, 2011)). The BerkeleyDB database system provide persistence, replication, high

availability and transaction processing (Yubin et al. 2013). MongoDB is a cross-platform document-oriented database. These databases use an ordered key/value store - LevelDB also uses Bigtable. Bigtable is high performance distributed storage system designed by Google for managing structured data. The design allows scaling large size of data (petabytes- 2^{50} bytes) across a large number of servers (Chang et al. 2008). These additionally provide higher level of availability, scalability and reliability (Xu et al. 2014).

2.5.3 Web Storage (Local Storage)

Local storage (Myeong et al. 2012) is a mechanism for storing structured data on the client-side. Web browser Local storage stores data in key/value pairs which are always stored as strings. Local storage defines two objects, the first is the localStorage object, and the second is sessionStorage object. localStorage has the same SOP restriction as other client-side storages. Data stored in localStorage persists even after the web browser is closed, while data stored in sessionStorage does not persist. sessionStorage can run in multiple browser windows for the same web application. Local storage provides the same functionality as cookies, but like sessionStorage, carries additional advantages over the use of cookies (West and Pulimood, 2012). The advantages include storing up to 5MB of key-value data per domain.

2.5.4 LevelDB

LevelDB (LevelDB, 2011, Pillai et al. 2013) is a persistent key-value store which was formerly developed at Google. LevelDB provides sorting by keys and ordered mapping from string keys to string values. Google Chrome uses LevelDB as an embedded database. The DataBase Manager (Dbm) library stores arbitrary data by the use of a single key. Like other NoSQL and Dbm stores, LevelDB does not use any relational data model, does not support SQL queries, and has no support for indexes.

2.5.5 Cookies

Cookies (Stalker et al. 2004) are small text files (4KB) stored on the client machine. They can have several functions in web applications as session, authentication or personalization storage. Due to their storage limit (4KB) other methods have had to be developed. HTML5 local storage is the standard's replacement for cookies.

Cookies used for session allow to identify the device associated with a particular user for the web application. The user is recognised and the web application is not treating the user as new visitor.

Cookies used for authentication are used to identify unique visitors to the web application or website. After successful authentication the web application remember who the user is so that web application can provide access to pages personal to that user.

Personalization cookies enable the web application to remember the user preferences for the web application. For example, if the user has set the web application to display specific background, layout or format these cookies will remember those settings.

HTTP works as request-response protocol between client and server as shown in Figure 2.7 (Stockhammer, 2011). Cookies are sent to the server with every HTTP request - which slows down the connection.



Figure 2-2 Cookie setup

HTML5 introduces several alternatives to cookies for storing data on the client-side. HTML5's client-side browser-based database is part of the state of the art for web applications but this can lead to the risk of client data being disrupted.

2.5.6 HTML5 File API

HTML5 standard provides a way where web applications can interact with files stored on client-side, via the File API specification (Crowther et al. 2014). File API allows a web application to save files to a temporary file location and reference the file while the user is offline (W3C, 2013). File API is another of the new functionalities embedded in HTML5.

A directory traversal (Han, 2015) (or path traversal) is a exploiting where security validation or sanitization of user inputted file names is insufficient. The characters, which represent a traverse to a parent directory, are passed through to the File API. The purpose of this attack is to allow a web application to access local files which is not intended to be accessible. This attack exploits a lack of security, where the File API is functioning as it is supposed to.

2.6. Difference between client and server-side databases

One of the advantages of using client-side storage over server-side storage for holding large amounts of data is the, non-functional, requirement of speed. Speed is important in cases where the Internet or the data connection is slow.

2.7. Differences between SQL and NoSQL databases

The difference between relational and NoSQL databases lie primarily in the storing of data. Relational databases (Connolly and Begg, 2014) have tables with rows and columns containing typed data. The IndexedDB NoSQL requires the creation of an object store for a type of data and the saving of JavaScript Objects to that store. Each object (type) can have a collection of indexes that make it faster to query and search across. NoSQL does not support joins directly whereas relational databases do. The different types of database system are different in scalability and performance. Comparison of query results using joins shows that NoSQL performs this kind of query, and renders the data faster. On the other hand, the code for NoSQL is much more complicated, as all the code needs to be manually written in JavaScript – this logic is provided natively by SQL. NoSQL databases have advantages over SQL databases because they allows scaling of an application to new levels. The new data services require scalable structures which can work in the cloud.

As has been said, NoSQL does not support SQL joins, and relations between tables need to be manually programmed (Pokorny, 2013). SQL joins mean that data are stored in multiple tables and are referenced by Ids. NoSQL developers turned their lack of joins into a feature, as complex join commands take a lot of resources and time to process. Therefore NoSQL stores everything in one place, which means that information can be obtained much faster. The difference between relational and IndexedDB lie in the storage of the data. Relational databases store tables with rows and columns of typed data. IndexedDB requires creating an object store for each type of data and saving JavaScript Objects to that store. Each object can have collection of indexes that make it faster to query and search across (W3C, 2015). IndexedDB does not support joins, where relational database does (W3C, 2015). The comparison of the query results using joins shows, that IndexedDB performs the query and renders the data faster. On the other hand the code in IndexedDB is much more complicated, as all the code needs to be manually done (W3C, 2015) in JavaScript that is otherwise provided natively by SQL. IndexedDB can split array in chunks of small pieces and using setTimeout, instead of loop inserted the data faster in database (W3C, 2015).

Next we are going to look at browser-based database IndexedDB in detail.

2.8. IndexedDB

IndexedDB, previously known as WebSimpleDB came from the W3C specification for implementing web storage into web browsers in 2009. IndexedDB is a persistent client-side database implemented into the browser and is an alternative to the already deprecated WebSQL. Mozilla and Microsoft supported the change: the use of Oracle's Berkley DB in IndexedDB mostly influenced this (Forfang, 2014). The application uses local data stored on a client system (Casario et al. 2011). It caches large amounts of data from server to client-side using JavaScript Object Stores - equivalent to tables in relational databases (Windows, 2011).

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 a persistent browser-based database, which means that the data can be retrieved even 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 (Flanagan, 2011). This is detailed in the IndexedDB specifications and implemented in both the Firefox and the Chrome applications of IndexedDB. Via this mechanism, all the relevant information can be stored in one place and a single query to IndexedDB can return it all.

In Firefox and Chrome's IndexedDB implementation, the files are stored transparently external to the actual database; in other words, the performance of storing a file in IndexedDB is just as good as that of storing the file in a filesystem. The storing of 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; therefore, IndexedDB is just as fast as a filesystem. The Firefox IndexedDB implementation is even smart enough that if it is storing the same BLOB multiple times, 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: data is written faster while using fewer resources.

2.8.1 Structuring the database

Unlike other web-based databases such as SQL databases that use tables for storing data, IndexedDB uses object stores. Multiple object stores use a single database. Keys are assigned to every value in an object store within a database; keys are assigned by key path or by a key generator. IndexedDB was created to allow local storage of data. However, it

has a somewhat limited feature set; for instance, it does not include the following features:

1) Internationalised sorting - Internationalised sorting cannot be supported with IndexedDB due to the wide variety of scripting languages in use in modern day web applications. While the database can't store data in a specific internationalised order, the client software can sort the data that is read out of the database itself.

2) Synchronising – Server-side databases currently cannot be synchronised with an IndexedDB database because of the time-consuming development which would be required for the implementation of this feature. Developers have to write code that synchronises a specific browser-based IndexedDB database with a server-side database, which is time consuming.

3) Full text searching - The API does not have an equivalent of the LIKE operator in SQL. The LIKE operator is used to search for a specified pattern in a column.

Clearly it is believed that these limitations are tolerable and do not counteract the lightweight system's real advantages. For instance in security terms, IndexedDB is a NoSQL database, which means that is not possible to perform an SQL injection. IndexedDB is built on a transactional database model. Everything the client does in IndexedDB always happens in the context of a transaction. A transaction is an atomic and durable set of data-access and data-modification operations on a particular database. It is how the browser interacts with the data in a database. Any reading or changing of the data in the database must happen within a transaction (MSDN, 2012). The IndexedDB API provides lots of JavaScript Objects that represent indexes, tables, cursors, etc., but each of these is tied to a particular transaction: applications cannot execute commands or open cursors outside of a transaction. Transactions have a defined lifetime, so if someone attempts to use a transaction after it has completed the process the API will throw out an exception error. One of the transaction advantages is to prevent user to run multiple instances of a web application at the same time. The purpose is prevention for database issues and affecting functionality.

2.8.2 Value in IndexedDB

Each record has a value which can include anything that can be expressed in JavaScript: including Boolean values, numbers, strings, dates, general objects, arrays, regexp, undefined, and null. IndexedDB enables the storage of structured data. Unlike cookies and DOM Storage, IndexedDB provides features that enable the grouping, iteration, search, and filtering of JavaScript Objects (MSDN, 2012). Each record consists of a key path and a matching value. These can be of a simple type, such as string or date; or more advanced, such as JavaScript Objects and arrays. Records can include indexes for faster retrieval of

(other) records and can store large amounts of objects. IndexedDB is a key-value store in the same way that Local storage is. Local storage retains just a string only key; therefore, the usual approach with local storage is to `JSON.stringify` all data to be stored. `JSON.stringify` (Ihrig, 2013) is a method that converts a JavaScript value to a JavaScript Object Notation (JSON) string. The `JSON.parse` method parses a string as JSON; this is suitable for finding an object with a unique key. However, the only way to retrieve the properties of `myObject` from local storage is to `JSON.parse` the object and then examine it. This is appropriate if the database only has one, or a few, objects. For example, if the database contains a thousand objects, all of which have a property `b`, and the user wants only to search values where `b==2`, it is necessary, using local Storage, to loop the entire store and check `b` on each item.

IndexedDB can store data other than strings in the value; this includes simple types such as `DOMString` and `Date` as well as `Object` and `Array` instances (Mehta, 2012). Furthermore, it can create indexes on object property values. Thus, IndexedDB can hold the same one thousand objects but then create an index on the `b` property and use that to retrieve only the objects where `b==2` without having to scan every object in the store.

IndexedDB is aware of ranges; therefore, it can search and retrieve all records beginning with `'ab'` and ending with `'abd'` in order to find `'abc'` etc. IndexedDB is implemented differently across browsers. Firefox uses `SQLite` and Chrome, `LevelDB`. The need for IndexedDB comes from the need to store more complex data on the client-side. One of the main reasons for using client-side databases is to reduce an application's dependence on a good Internet connection. Running a web application that requires a lot of reading and writing data from/to a server-side database depends on continuous Internet connectivity: if the connection is lost, parts of the data may get corrupted. The browser-based database standard was proposed in order to fulfil these needs and solve the issues mentioned. W3C, Mozilla and Chrome state that their new browser-based database systems do not have storage limits (MDN, 2012). The implementations merely ask for user permission to store larger amounts of data after a certain threshold has been reached, and what this threshold is, depends on the browser implementation. In this connection, it should be noted that where IndexedDB is used in Firefox, it is implemented via `SQL-backed` technology (MDN, 2014).

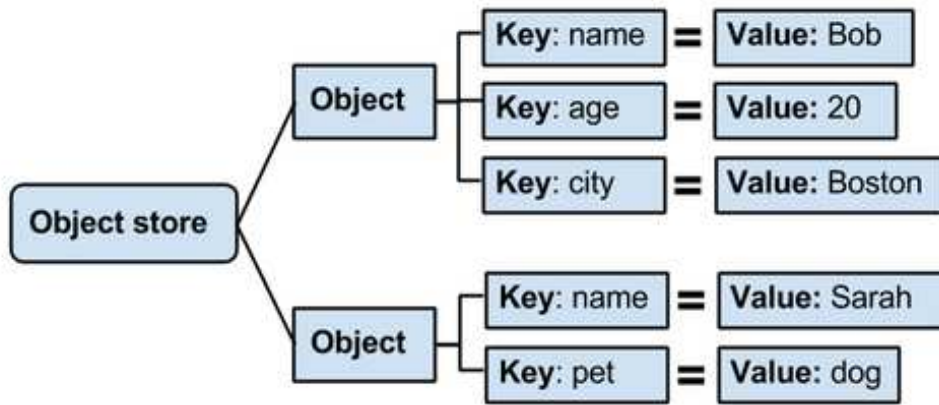


Figure 2-3 IndexedDB structure

Google Chrome, can have up to 5 MB of storage for IndexedDB, by default. Installed apps can make use of unlimited IndexedDB storage if the manifest file is set to have unlimited storage and the user grants that permission to the application. When web application request to store larger amount of data then the default, web browser alert the user. By permitting, unlimited quote of data such as text, videos, images can be stored on client-side.

JSON-formatted manifest file provides important information such as the name of the application, its description and the URLs that the application uses (Google developers, N/A). An application with a manifest.json file can be hosted on any website. The manifest file can specify the storage size of the application - the data storage can be unlimited. JSON is a lightweight data-interchange format (Crockford, 2008).

The values (in IndexedDB) can be complex structured objects and keys can be properties of those objects. Indexes use properties of the objects for quick searching and sorted enumeration. A key is a data value by which stored values are organized and retrieved in the object store.

IndexedDB does not use SQL; it uses queries on an index that produces a “cursor”, which is used to iterate across the result set. An index is a data structure (a way of storing and organizing data) that improves the retrieval of data from the database. The structure of an IndexedDB database can only be modified during a version change transaction. This means that the only time ObjectStores or indexes can be created or removed is during a version change transaction. Basically, the IndexedDB API automatically creates a versionchange transaction whenever a database is opened through the open method and one of the following two conditions occur:

- The requested database does not exist.

- The requested database version number is greater than the version number of the database on the client machine.

2.8.3 Disadvantages of IndexedDB

IndexedDB has significant disadvantages which have been identified and discussed in the literature. These include the lack of browser support (Opera) or partial support (IE, Edge, Safari) as well as difficulties with complex queries, also the fact that the data is stored in a non-encrypted form, and that older, deleted versions of a database can exist on a disk and can be accessed with forensic tools. The code is too complex to implement (currently) a fully working web version and it has no BLOB support in Chrome browsers older than v.37. Browsers such as Chrome, Firefox, IE and Safari 8 support IndexedDB (2015). As the database is NoSQL, there is a problem when constructing complex queries (Zachary, 2013) such as joint tables.

2.9. Security Issues

This section describes the security issues which have been identified in relation to the new HTML5 functionalities. Additionally it describes how the current security attacks can impact the end user and their data via the new HTML5 functionalities.

2.9.1 Origin of the problem

With the new HTML5 functionalities, new security issues arise because it provides more access to the computer's resources (Taivan et al. 2014). In addition, with local storage and offline caching available in HTML5, the web browser might store sensitive data, such as that originating from the client's email account (De Ryck et al. 2014).

The client-side security model is basically the Web browser security model. On the release of the new standard, it was found that the security had not been updated to reflect the new functionalities (De Ryck et al. 2012). Therefore, browser vendors will need to develop a richer security model, much like those that exist for operating systems (Livshits et al. 2013). The majority of attacks in relation to HTML5 have their effect on the browser and the end user, and thus do not have a direct impact on the server (Tian et al. 2014).

Web browser security operates by using the same-origin policy (SOP) (Gollman, 2011, Ss. 2.3.2), which involves linking stored data to a particular domain or sub-domain, and ensuring that the data cannot be accessed from any other source.

SOP applies to any storage in the browser. When SOP is implemented, the web browser checks the hostname (`www.someurl.com`), port number (web browsers run on port 80)

and protocol (HTTP or HTTPS) against the origin record of stored data. Only when they match, the web application will be allowed to access the data. The SOP is the only form of browser-based protection against potential security threats, available in the standard. SOP even disallows 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). Thus, the prevention of data or attacks coming from a different domain is possible. Web browsers use this preventative technique against untrusted site attacks. Attackers use multiple techniques which make use of the ease with which browser history can be inspected (Weinberg et al. 2011). These techniques include "sniffing", which is where web browsers render web pages with malicious JavaScript code (Barua et al. 2011). The code needs to be placed on the attacker's web page; this will trigger the code to "sniff" the history. Based on this fact, we can assume that the attacker might get access to other stored information via the browser. SOP weaknesses have led to attacks such as Cross-Site Request Forgery (CSRF), Cross-Site Scripting (XSS), and Web cache poisoning. Using HTML5 localStorage to replace session data stored in cookies improves the application's scalability and prevents simple CSRF attacks because, unlike a cookie, data in localStorage is not automatically sent.

2.9.2 Same origin policy (SOP)

The only existing security mechanism for browsers is this Same Origin Policy (SOP). SOP restricts loading content or script from one origin which is not the same as the origin requesting the content (Huang et al. 2010). When the data is stored on the client-side, the only point of access for that data is through the user's local machine. (Saiedian and Broyle, 2011) does not consider SOP to be the appropriate security mechanism, because a cross site scripting attack might bypass the mechanism, executing a malicious script, similar to CORS (Ss. 2.9.3). Stored data on the client is more likely to be compromised than data stored on server (Hanna et al. 2010).

HTML5's new functionality allows attackers to access untrusted sites, even if they are on a different domain, meaning that the SOP will not apply to this situation. Security vulnerability and potential attacks might be possible here since the attacker will be able using hacking techniques to reach and access the database from a different domain (Stuttard and Pinto, 2011). If the website or application is vulnerable to XSS attacks, then the attacker could steal the user's data from the client-side database. When the SOP is not correctly configured, then content from different web sites will allow attackers to

manipulate the data through their code's access.

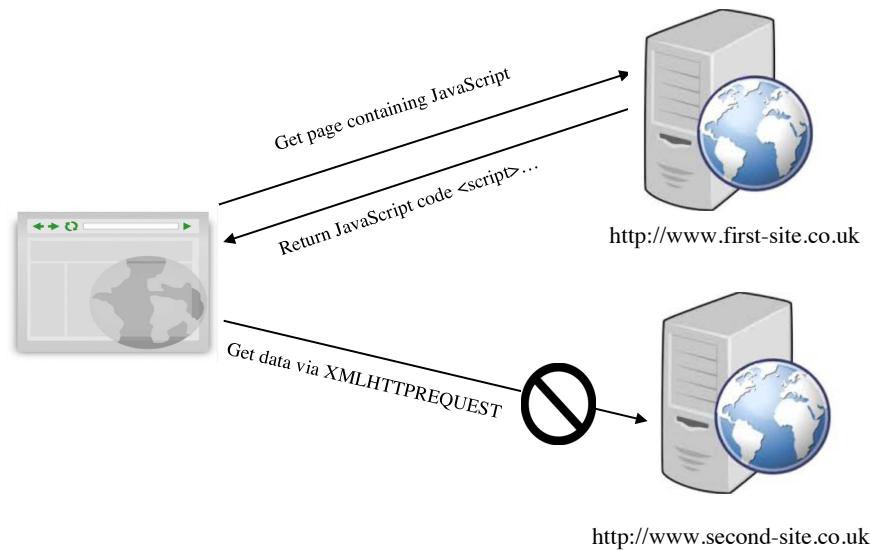


Figure 2-4 Same Origin Policy

The SOP security measure is not enough to prevent an attacker from getting data from a different domain (Stamm et al. 2010). When data is stored on the local machine in the database, the applications are limited to access only data created by particular applications on a domain. This is a security vulnerability of web browsers, where the client database is situated: an attacker might compromise the client data (Stuttard and Pinto, 2011).

Since the current Same Origin Policy is not secure, other solutions might include using an entirely different policy. Potential policies might include Content Security Policy (CSP). The CSP restricts common attack vectors in the client browser. The CSP employs a set of directives that define the security policy for all types of Web page content (Saiedian and Broyle, 2011).

2.9.3 Cross origin resource sharing (CORS)

Cross origin resource sharing (CORS) is a specification that gives JavaScript on a web page the ability to make XMLHttpRequests (XHR) (Rauti & Leppänen, 2012, Fang et al. 2011) to another domain, not the originated from. XHR is defined as API that can be used by JavaScript to make transfers between the client and server. Normally, Web browsers would 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 (Zakas, 2010). Letting third party applications access the data created by another domain's application can lead to security issues such as information leakage. Therefore user agents must implement CORS with IndexedDB in mind. 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 if an application creates an IndexedDB database, which is saved with the domain name, another application cannot access the database files, as the access is restricted for the particular domain. One possible 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 (Stuttard and Pinto, 2011). This is just to mention one possible CORS based attack which can be used to bypass the restrictions and so read data from other domains.

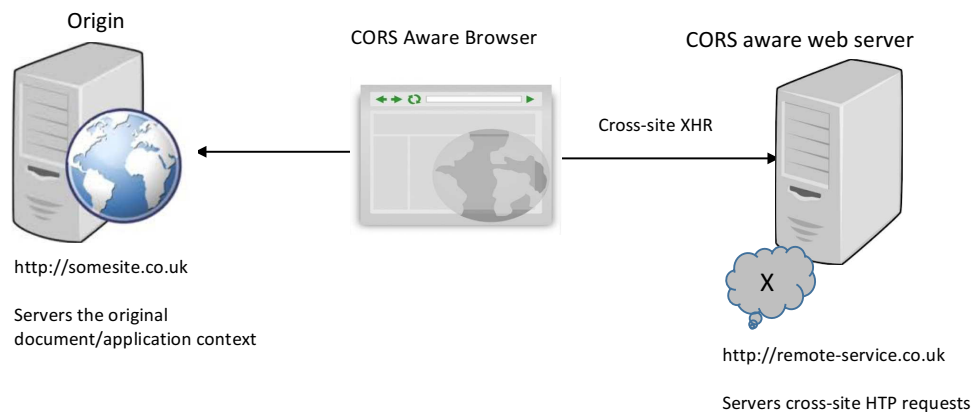


Figure 2-5 Cross origin resource sharing

2.10. Online and offline attacks

2.10.1 Cross-site scripting (XSS)

XSS is one of the most popular web application attacks - third on the OWASP list (OWASP, 2013). WhiteHat (WhiteHat, 2014) security has provided statistics whereby XSS regains the position of the number one web application vulnerability. XSS is a popular attack because even where the web application is secure, the attack can rely on the end users who can be tricked to click a link and therefore authorize the attack.

2.10.1.1 Definition

XSS (Elhakeem et al. 2013; Wassermann and Su, 2008; Vogtet al. 2007; Di Lucca et al. 2004) is a security vulnerability where the attacker injects malicious JavaScript code into web application. Victim’s web browser executes the code with victim’s privileges and the code can modify or transmit any data through the victim’s browser to the attacker.

In order to bypass access controls, such as the SOP, attackers may use a cross-site scripting vulnerability.

Di Lucca et al. (2004) identified a number of cross-site scripting vulnerabilities in web applications, in which the attacker takes the dynamic analysis approach. XSS is the most best known advanced threat to offline web applications and web databases according to the Open Software Security community (OWASP).

2.10.1.2 Types of XSS attack

Stored XSS (Persistent) attacks (Wang et al. 2011) generally occur when user input is stored on the target server database. The attack consists of storing potentially dangerous scripts to this database which get executed every time a user makes a request to access data on it.

A reflected XSS (Non-Persistent) attack (Pelizzi and Sekar, 2012) occurs when user input is instantly returned by the web application in a form such as search result or error message. Reflected attack is engineered to trick a user to click a link which will trigger a script. Figure 2.4 shows the XSS attack principle.

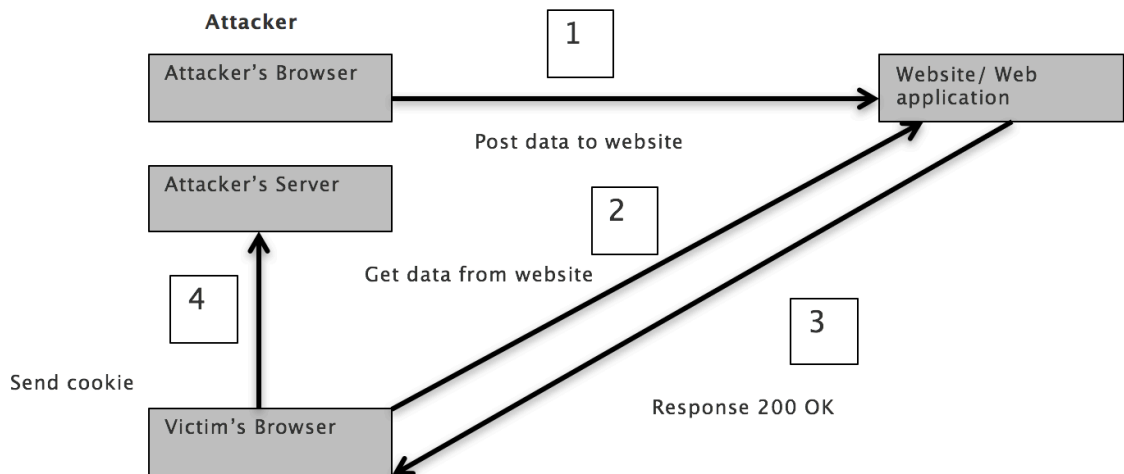


Figure 2-6 XSS attack explained

1. The attacker can use one of the web application forms (login, search) to insert a malicious code formed as a string into the web application back database.
2. The victim requests a web page from the web application.
3. The web application includes the malicious code from the database in the response and sends it to the victim browser.
4. The victim's web browser executes the malicious code inside the response, which will send the victim's cookies to the attacker's server.

Document Object Model (DOM) Based XSS (Lekies et al. 2013) is a form of XSS where the attack where the malicious code is being executed in the DOM rather than HTML. If a page contains a script which writes the data requested from a URL, the attacker can simply add dangerous script to the URL, which gets executed. The script gets written to DOM, and thus the attacker can get access to cookies or change the page behaviour.

Server XSS occurs when the manipulated data comes from the server HTTP response.

Client XSS occurs when malicious data includes JavaScript code to update the DOM.

2.10.1.3 Cause of the attack

XSS takes advantage of web applications where the user input is not filtered properly. XSS filtering is a process of filtering out parameter values that look suspicious (Pelizzi and Sekar, 2012) - this includes special characters. XSS attack can be used to manipulate not only direct form (search or login form) but also session cookies or data stored in database. Bates et al. (2010) argues that many XSS exploits are possible not because implementation but design errors. Most XSS attacks can be prevented by sanitizing or validating user input (Shar and Tan, 2012).

2.10.1.4 Structure of the attack

XSS makes use of the fact that attackers can input html code or other client-side scripts like JavaScript. Using XSS an attacker can create cookies which are used to bypass access control. In most cases malicious code is added to a hyper-link which is added to a website to which the user has legitimate access. Via this mechanism, the hyper-link is executed, so parsing also the malicious code, and thus the attacker might obtain sensitive data from the victim. XSS can enable the attacker to, for instance, steal authentication information and hijack accounts, and of course this will allow the attacker to change user settings and/or information (Malviya et al. 2013). Other attacks might be to damage the site or to steal cookies. This might lead to the accessing of the admin account.

HTML Inline Frame (IFrame) allows the embedding of content from sources other than the main page (Mansfield-Devine, 2010). An IFrame is an HTML embedded inside another HTML on a website. Main purpose of IFrame is to insert content from other source into web pages, usually used to embed advertisements.

Based on these XSS vulnerabilities, an attacker can inject a script, or any file, an html file, a css file, a script etc. via the IFrame element in HTML (Liu et al. 2013). Using

this, an attacker can change the appearance and/or behaviour of a web application, adding malicious content or otherwise controlling the web application and/or server database.

Mewara et al. (2014) identified remaining cross-site scripting vulnerabilities in web applications where a dynamic analysis approach is chosen as the protective mechanism. Protection via the dynamic analysis approach uses a mechanism on the server which removes any script in the input that is not intended by the web application.

2.10.1.5 Preventions against XSS attacks

There are preventions against XSS attack such as encoding output, filtering and validating user input. Encoding output based on input parameters (Lan et al. 2013) uses a web proxy to prevent execution of URL links which contain binary encoded characters.

Another prevention is to filter the input parameters (Yusof and Pathan, 2014), where any data from the input is filtered so that it is not executed in the browser. To avoid XSS attacks developers must sanitize the user input before the input data is stored in the database.

Validating user input is another important step to secure web applications. The web application should always check that the user input is in the correct format (string, integer) and the right length. Without validating user input the web application could be vulnerable and the attacker could change the site content or get data from database.

2.10.1.6 XSS impact on client-side databases (IndexedDB)

The new client-side database facility provides the functionality to store data on the user's 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 this client-side storage. The client-side storage data can be accessed through the browser, so the execution of an XSS attack might output the stored data.

When the attacker has gained access to information on the Web server via a XSS vulnerability, there are many things that they can do. Once they have access to the information, the attacker can copy it to a different location, change the data, delete the data or inject malware into it, etc.

It is important to secure the Web servers on which web applications exist against XSS vulnerabilities. A real example of this includes Google Docs where the user can choose to use the application offline. If there were even one XSS vulnerability anywhere on the Google Docs web-site pages, an attacker would be able to access and steal or modify the user data of anyone who logs in to the application (Liu, 2012).

2.10.2 Man in the middle attack

The man-in-the middle attack (MITM, MitM, MIM, MiM or MITMA) intercepts a communication between two systems. MITM attack alters and listens to the communication between two systems or people (routers, server-client, server-server, client-client) (Lyubashevsky and Masny, 2013).

When a user sends a public key to another user and the attacker is able to intercept it, a man in the middle attack is possible. A MITM attacks allows the attacker to intercept, send and receive the data from the intercepted communication. MITM is an eavesdropping attack, where the attacker is inserted into a communication session between systems or people. A MITM attack exploits the immediate processing of conversations, transactions or other transfers of data. MITM attacks allow attackers to intercept, send and receive data, without the end system knowing.

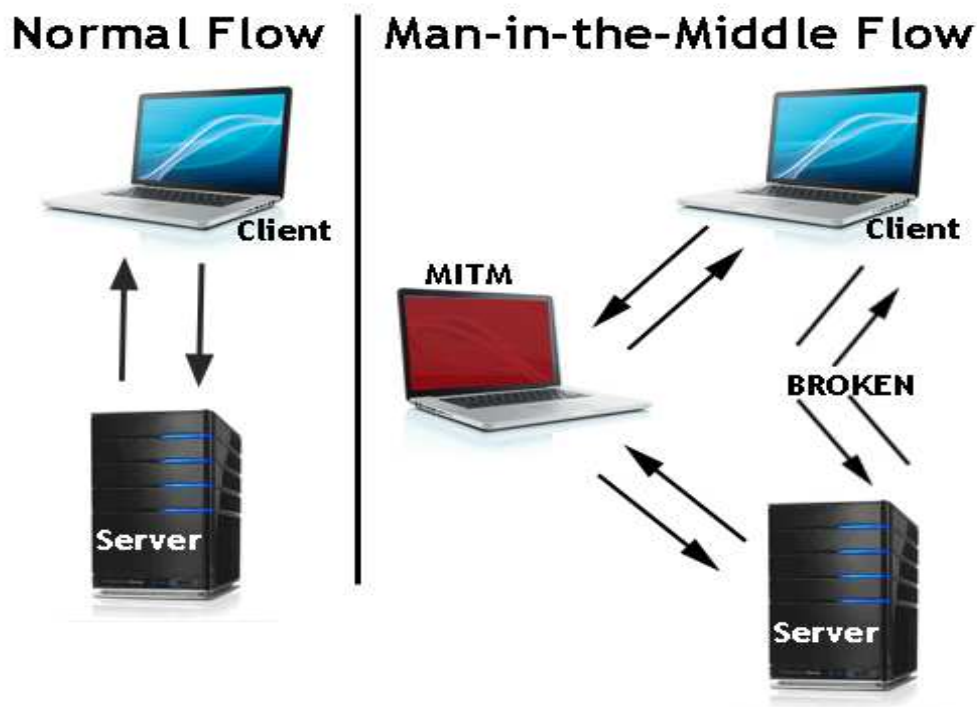


Figure 2-7 Man in the middle attack

MITM is a method of session hijacking. Other methods of session hijacking similar to MITM are sidejacking, sniffing and evil twin.

Sidejacking (Vishwakarma et al. 2015; Borders et al. 2012; Riley et al. 2011) is an attack that involves the attacker to sniff data packets with the purpose to steal session cookies. Session cookies can contain login or user information in unencrypted form, even if the site is secure and the attacker can hijack a user's session.

Evil Twin (Lanze et al. 2014; Nikbakhsh et al. 2012) is a rogue Wi-Fi network that looks and acts as legitimate. The attacker controls the evil twin network and therefore if a victim joins the network, the attacker launches a MITM attacks. All data on the network is intercepted and captured by the attacker.

Sniffing (Pandey et al. 2014; Barua et al. 2011) involves a attacker using available software to intercept data being sent from, or to, user device.

There are 2 types of attacks, intentional and unintentional. The intentional attacks tries to trick the user into clicking a link which will redirected the user to a malicious site. An unintentional attack happens when the site is vulnerable to malicious attacks, such as those associated with insufficient input validation (Alkhalaf et al. 2012).

There are two types of developer: friendly and hostile. Each type can be further categorized into competent and incompetent.

When using a site or application constructed by a friendly and competent developer we can assume that the web application will be secure and that the end user will be protected against online attacks, such as XSS. The main issue to be borne in mind when constructing a security regime is that XSS attacks can, and often do, succeed. The end user can be tricked into allowing XSS, for example, by clicking a link in an email. The email could have been sent from a known address, therefore the end user would not be wary of malicious code included in the link. When the end user clicks the link, an authorization to perform the attack will proceed.

2.10.3 Social engineering attacks

Social engineering, in this context, is a method of tricking and manipulating people so that they give up confidential or personal information. The attackers usually trick people into giving them passwords or bank information. Alternatively they might, in this way, gain access to the user's computer to secretly install malicious software with the purpose of accessing information or controlling the computer (Huber et al. 2011). Attackers use social engineering methods because it is usually easier to exploit human nature than it is to discover ways to hack web applications or software. For example, it is easier to trick a victim into providing the attacker with login details than it is to try to hack their password (Krombholz et al. 2011).

An example of social engineering might be an email from a known sender. If an attacker succeeds to socially engineer victims email password, then that attacker will have access to the victim's contact list. The attacker can then send emails or messages to members of the victim's contacts list with a link. If this link is then clicked on, then the result could be that the next victim's computer will be infected by malware or that they are

redirected to the attackers site (Kotenko et al. 2011). The link might also contain a download, such as a picture, movie, document or audio file which has malicious code embedded in it. When the victim downloads the file, the victim's computer will be infected and the attacker may then have access to the victim's computer, emails, accounts and contacts.

2.10.3.1 Types of social engineering attacks

Phishing is when an attacker or malicious group sends a fake email, looking like a legitimate email from a trusted source, with a message which is capable of installing malware on the victim's computer or sharing financial or personal information.

Baiting involves leaving a malware infected physical device unattended for someone to find and use it. If a victim loads the device (usb flash device, external drive or CD-ROM disk) they may then unintentionally install malware.

Pretexting is when one side lies to another to obtain access to personal confidential data. Example of pretexting could involve a scam where the attacker call a victim pretending to be a bank representative, but the victim needs to confirm the identity first. Attacker then gets financial and personal information about the victim.

A quid pro quo is when one side tries to obtain login details from other side in exchange for a desirable gift.

Spam is considered to be unwanted junk email.

Spear phishing is a kind of phishing which is personalized for a specific person or company.

Tailgating is when someone who is not authorized to access a secure location follows an authorised person when passing through a door. The purpose is to gain access to secure location where confidential or valuable items are stored.

2.10.4 SQL Injection

An injection attack is defined as any attempt to threaten the web application database by submitting unsupported or unexpected data as user input. SQL injection is a common type of injection attack (Clarke, 2012). SQL injection attack occurs when the SQL query is altered with illegal characters by not sanitising the user input. The query is passed on to the database server where the query is executed. Illegal characters might be semicolons, apostrophes or commenting characters, which can result in dangerous and unexpected queries (Shema, 2012). Any unchecked user input is considered unsanitised and can lead to dangerous and incorrectly formed queries. For example if the query expects a textual input and receives numeric input, the database can misinterpreted the

query as numeric command. Usanitised user input is considered to be a security vulnerability that must be corrected (Shulman, 2006).

Injection attacks differs from traditional attacks on the web server, where the attack is no attempting to gain access to server. Injection attack takes advantages of vulnerabilities in web application, such us usanitised or unchecked user input with the purpose to use or alter the data in database. Because JavaScript is an interpreted language, there is the potential for injection attacks related to this as well.

2.10.5 Physical access

Physical access (Szefer et al. 2012) is possible when the attacker can gain physical access to the user's machine. When a device, or stored data therein, is unencrypted, the attacker may be able to access all its data. The security solutions which prevent physical access attacks include physical access controls.

To gain access to restricted areas only personnel with authorisation should be allowed using access cards. In terms of physical access, the attacker or any person with access to the filesystem could potentially get the file and the data, which would mean that it could be transferred to an external drive and used via an the appropriate application.

A possible solution to this, to prevent an unauthorized person gaining access to a filesystem, is to lock the screen so that a password has to be entered before any of the files can be viewed. Mobile devices could be secured with two-factor authentication something the user knows with something the user have. For example mobile device could be locked with password or pin lock, but have a fingerprint scan as well. This way the authentication will be harder for an attacker to pass.

Security experts and security researchers do not recommend that users or web developers save any sensitive data on the client-side, and user machine. The recommendation is to store any sensitive data on the server.

2.10.6 Tracking privacy

With the new HTML5 functionalities new attacks are possible. For instance, HTML5 provides the functionality to track a user's physical/geographical location, which can cause a loss of their privacy (Mayer and Mitchell, 2012). This means that web applications can track user location and store it in their systems. Social networks provide a functionality whereby the user can check in to a particular location, and this location is stored on the system. Based on the check-in the user location can be obtained and can cause a loss of privacy (Wernke et al. 2014). Before the user location is stored the user is

notified by the browser that the application would like to access the user's physical location.

2.10.7 Cross-site request forgery

Cross Site Request Forgery (CSRF) (Barth et al. 2008, Käfer, 2008, Jovanovic et al. 2006) is web application vulnerability where a malicious web site can make legitimate requests to a vulnerable web site under the cover of a logged-in user without that user's knowledge. If a user can send email to his friend, then a malicious web site can do the same. This vulnerability has been rated as one of OWASP (Open Web Application Security Project) Top 10 vulnerabilities.

A CSRF hole is when a malicious site can cause a visitor's browser to make a request to server that causes a change on the server. The server thinks that because the request comes with the user's cookies, the user wanted to submit that form. CSRF flaws exist in web applications with a predictable action structure and which use cookies, browser authentication or client side certificates to authenticate users. CSRF are often GET requests collected and sent through the use of an automatic load (such as img or script tag).

The user typically thinks that are performing a different task but using HTTP requests that have side effects the attacks use the user's own browser to send HTTP attacks to the target site.

CSRF attacks can be prevented by not relying only on cookies. In secure applications, session's tokens are submitted via hidden fields in HTML forms. When the form is submitted the application verifies that the correct token is received. The attacker will be not able to perform attack without knowing the session token (Stuttard and Pinto, 2011) . The session token must be randomly generated unique number. So the application will send the cookie session with attached session token. After the form is submitted, the token will be checked and if is valid the action will be performed. Requiring a secret, user-specific token in all form submissions and side-effect URLs prevents CSRF so the attacker's site can't put the right token in its submissions.

In the next section we are going to identify client and server-side databases, and the main differences between them. Additionally, we are going to perform an investigation of which security issues, vulnerabilities and attacks can harm these kinds of databases.

2.11. Cryptography

Cryptography (Stallings, 2013; Katz, 2008; Konheim, 2007) is a method of using algorithms to transmit data in a non-readable and secure way. Cryptography has two parts, encryption and decryption.

Encryption (Stallings, 2013; Stinson, 2006) is an essential tool to protect sensitive information. The purpose of using encryption is to provide privacy in order to prevent disclosure and breaches of confidentiality during communications. Encryption is the process where original data (plaintext) will be transformed to encrypted data (ciphertext). The original data is the user input and ciphertext is the encrypted original data output, as shown in Figure 2.9 and Figure 2.10.

A cipher is a pair of algorithms that create the encryption and the reversing decryption. Many modern encryption systems use two keys, one is called the Private Key and the other one is called the Public Key (Stallings, 2013). The public key encrypts the message so that it can be sent to the recipient for decryption using the private key.

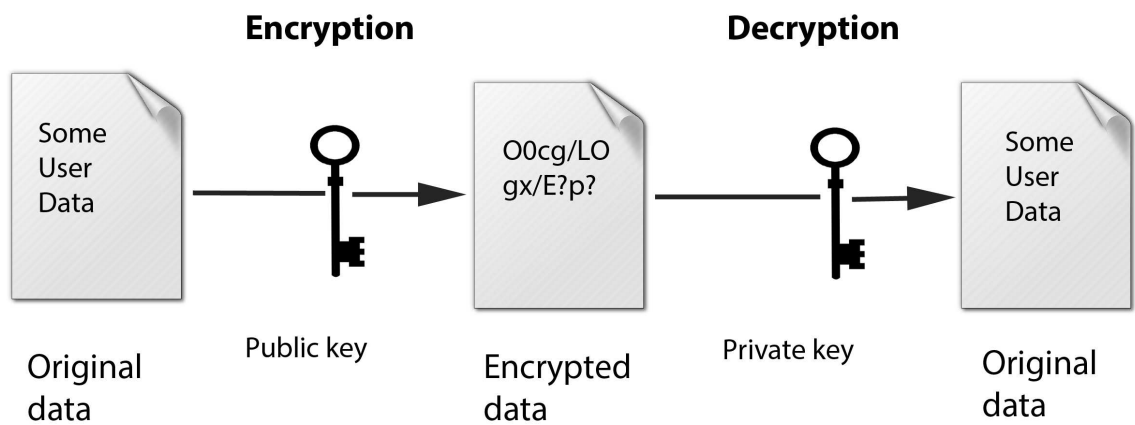


Figure 2-8 RSA Encryption and decryption process

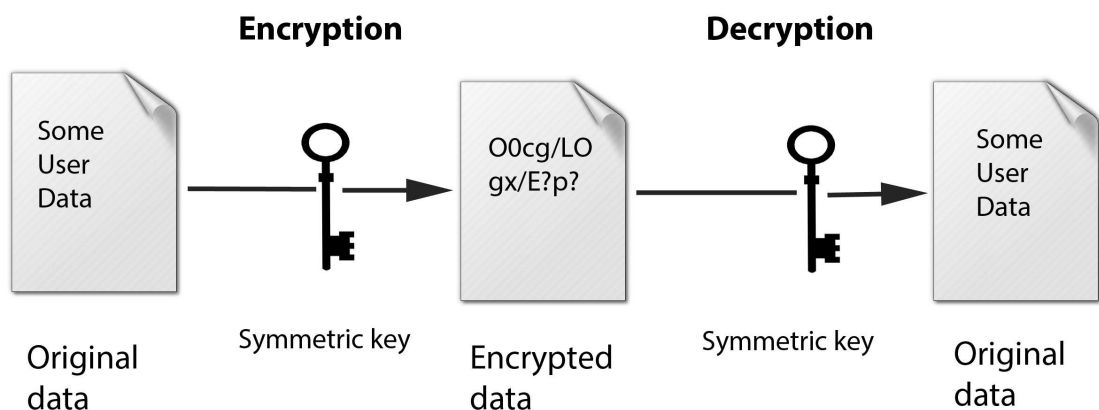


Figure 2-9 AES Encryption and decryption process

Plaintext is any information that a sender wishes to transfer to a receiver. It can be thought of as the input to any encryption algorithm or as information to be transmitted before an algorithm encrypts it. Examples of plaintext include email messages, word processor files, images, or ATM and credit card transaction information. This plaintext is converted to ciphertext, which is data that has been encrypted and is unreadable until it has been decrypted with a key (Thakur & Kumar, 2011).

Client-side encryption is still not mature, compared to server-side. Many of the client-side encryption libraries cannot yet deal with the complexity of all the case scenarios. It is relatively easy for an attacker to bypass client-side encryption in comparison to server-side. This also will depend on the depth of encryption and the location of public and private keys. Client-side encryption is essential when storing data on the client-side machine. It provides security for user data and prevents people from viewing the content of the user's files. Basharat (2012) discusses the importance of database encryption but suggests that strong encryption might reduce performance, but he also argues that encryption reduces the attack risk and protects sensitive data.

The salt is 64 bits (binary digits) of random data which is added to the key (before the use of the pass-phrase) in order to make it less predictable (Mechanic et al. 2007). A salt could be used on client-side for hashing (for server-side encryption), but it offers no additional security. Usual way for storing hash value is to combine salt and the password (OWASP, 2014). When using hashing on client-side, the salt would need to be sent from client to server. Then server sends back salt to user where the same would be generated. There is, therefore, the possibility of a MITM attack because the salt would be sent from server to client. However, hashing should be used for all client-side authentication and encryption. When the encrypted content is transmitted a MITM attack could get the encrypted content, but without the password the decryption would be not possible. The password would be stored on client-side and therefore not shared with the server.

The W3C working draft of its Web Cryptography API is intended to help web developers secure their web applications using encryption and hashing. This specification does not explicitly provide any new storage mechanisms for CryptoKey objects. Instead, by allowing CryptoKey objects to be used with the structured clone algorithm, any existing or future web storage mechanism that support storing structured clone-able objects can be used to store CryptoKey objects. This means, that the key can be stored in IndexedDB with additional meta data.

There are proposed solutions to the problem of protecting sensitive information on the client-side. One of these solutions includes the concept of CRYPTONS, which is a

software framework for remote storage whereby the remote server has no knowledge of what is being stored. It implements a database-abstraction layer that provides support for most major systems. Dong et al. (2013) and Xu et al. (2013) discuss the issue that storing sensitive data on client-side is not secure and the existing protection is not enough.

2.11.1 Cryptography Algorithm

An algorithm (Stallings, 2013) is a set of instructions intended to perform a specific operation or set of functions. There are many different ways to perform a specific operation or action. Therefore, algorithms serve to make the operation or action more efficient (Cormen et al. 2009). Algorithm functions accomplish a task as a small program that can be a part of a larger program. Based on the type of key being used, cryptography algorithms are classified into symmetric and asymmetric key algorithms.

Symmetric key algorithms (e.g., Advanced Encryption Standard or Rijndael Algorithm (AES), Data Encryption Standard (DES), Rivest Cipher (RC5), Blowfish, etc.) and Asymmetric key algorithms (e.g., Rivest-Shamir-Adleman (RSA), Message-digest algorithm (MD5), Secure Hash Algorithm (SHA), etc.).

Symmetric key algorithms use one, the same symmetric key for encryption and decryption as shown in Figure 2.10. Asymmetric key algorithms use different keys, public key for encryption and private key for decryption as shown in Figure 2.9.

Advanced Encryption Standard (AES) or Rijndael is an encryption algorithm where with encryption and decryption a single symmetric key is used. AES encryption replaces the DES and Institute of Standards and Technology (NIST) released the specification for AES. AES algorithm has block ciphers with 128, 192 or 256 bits. AES is very secure (Stallings and Brown, 2008) and provides flexible and fast functionality. For encryption the user needs to enter a password, where a random salt for new password is added. User can decide how secured password should be, by choosing the key size (128, 192, 256).

2.11.2 Difference between hashing and encryption

Hashing is used when checking the validity of input data. For example, when we have two input values and want to check to see if they are the same. This is mostly used for passwords and authentication (Park et al. 2010).

Encryption is used to transform data to make it not visible to others in plain text. Encryption is used when we want to store data and later retrieve the data (read). Encryption is mostly used when storing data, long term. When encrypting plaintext into ciphertext a key is generated. This key is required when decrypting ciphertext back into plaintext.

2.12. Biometrics and Multifactor authentication (MFA)

Nowadays, our lives are surrounded by electronic systems whose access is normally protected with simple authentication methods, such as passwords, pins or patterns. These traditional authentication methods are generally vulnerable to development mistakes or other issues: spy-ware, brute force and dictionary attacks.

Biometrics (Dozono et al. 2014, Yang et al. 2014, Yampolskiy et al. 2014) is a interesting future option, particularly for use with touchscreen devices. Biometrics is not considered to be a single, sufficient, authentication mechanism: it must be used alongside other factors in order to create a sufficiently secure system.

Existing mobile and tablet devices provide basic biometrics, and these can be used to extend the authentication process (Chuda et al. 2015). Instead of using something a person has (like a card or a physical key) or something a person knows (like a password), biometrics uses physical or behavioral characteristics to identify the person. Physical characteristics might include fingerprints, face identity, vein geometry, palm or eye iris scans. Behavioral characteristics can include voice, handwriting or type rhythm (Ramya et al. 2014).

Biometrics systems use three steps: enrollment, storage and comparison. Enrollment is the first step, at which the biometric characteristics are recorded as shown in Figure 2.11. The system will get an image of a fingerprint and then specific characteristics such as the pattern of ridges and valleys from the image are filtered and saved in binary form; these are used for verification. The algorithmic result cannot be reconverted to an image, therefore it should not be possible to duplicate fingerprints.

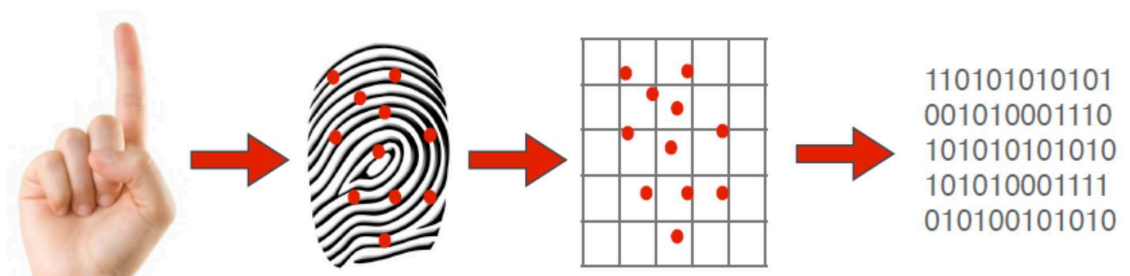


Figure 2-10 Principle for saving a fingerprint scan

Biometrics provide a convenient level of security, reducing fraud and attacks. They eliminate the problem caused by lost IDs or passwords by using psychological or physical

attributes. Prevention against unauthorized access is at a much higher level when using biometrics because their use make it possible to know who has accessed a restricted area.

The disadvantages of biometrics include the expense of the machinery which provides state of the art sensing – e.g. retina or iris.

Multifactor authentication (MFA) (Fleischhacker et al. 2014, Bruun et al. 2014, Bell et al. 2014) is a security system that requires more than one method of authentication from independent categories of credentials to verify the user's identity for a login or other transaction. This means that to be able to access a user account or certain data, the user is required to input additional security beyond basic authentication such as username and password. MFA generally uses an external device or software to generate a token - which is one time use only.

Methods for generating a token:

- SMS
- Mobile device application

MFA uses three basic elements, something the user knows (password, pattern or PIN), something the user owns (access card or mobile device) plus something the user is (voice, eye retina or fingerprint) (Singh and Chatterjee, 2015). To reduce the risk of exposure data a strong authentication process should be applied. Facebook and Google have users confirm their identity when accessing their account from an unrecognized device.

2.13. Web Sockets

The WebSocket protocol enables the server to communicate to the client-side code when the user requests to change the server-side data or other server-side action occurs. For existing Web Sockets the handshake is based on session cookies.

Webix is an existing Web Socket which supports IndexedDB saving and retrieving of data. The protocol doesn't handle authorization and/or authentication (Jemel and Serhrouchni, 2014).

The current problem with web sockets is cross-site WebSocket hijacking, as Web Sockets is not restrained by the SOP.

When a server does not check and validate the origin header from a Web socket request (i.e. handshake) then server might accept the connection. The WebSockets

connection, same as with CORS can originate from different origin. Because the connection is regular HTTP or HTTP request the cookies is being sent, even if cross site.

Attacker can begin a WebSocket request from a malicious site targeting a web application, where the victim is authenticated. The browser sends the victim cookie in WebSocket, where the attacks scenario is similar to CSRF attack. The difference is that attacker can perform reads and writes in the WebSocket connection.

2.14. Conclusion

HTML5 provides new functionalities for web developers, including the local storage standard (Jemel and Serhrouchni, 2014). With local storage all of the data from web applications can be stored locally instead of on the server (Jemel and Serhrouchni, 2014). Storing data locally allows web applications to respond faster since the data is retrieved locally as well.

In addition to these new functionalities it is suggested that new security issues will arise (De Ryck et al. 2011). Storing data locally has a performance advantage, but in terms of security there are possible problems XSS (Hydara et al. 2015; Liu et al. 2015), and saved data encryption (Jemel and Serhrouchni, 2015). This thesis then builds on work in number key areas, security of browser based databases and browser model De Ryck et.al (2011) and database performance work by Van der Veen (2012).

It can be concluded that attacks such as XSS, or physical access could impact the data stored in browser-based databases (You et al. 2015; West and Pulimood, 2012). Browser-based database technology is new and therefore not much is currently known about its associated security issues. Considering the fact that personal information could be stored in IndexedDB databases locally, security is a significant factor. As HTML5's new functionalities and API do not provide any additional security mechanisms but additionally does open the door to new, previously unknown attacks (such as File API attacks (You at al. 2015), Web Sockets attacks (Choo et al. 2015), bypassing SOP with CORS (Smith, 2015), the issues need to be addressed before IndexedDB, become widely adopted and used.

Existing solutions such as encryption are not sufficient on their own to protect against XSS attacks (Liu and Gong, 2013); methods such as Multifactor authentication (Tirumala, 2015) or biometrics (Gupta and Gupta, 2014) could make the situation more secure. Considering that the data can be stored on many different kinds of device such as tablets or phones, physical access attacks are likely from which sensitive information could be retrieved (Shin et al. 2015). Such attacks do not follow any particular steps, therefore it is nearly impossible to produce a methodology to describe these attacks, but

rather only some use case scenarios. The thesis builds on work of De Ryck et.al (2011) and Jemel and Serhrouchni (2014), where the thesis is proposing new browser based security model.

According to the W3C documentation (W3C, 2015), IndexedDB was not designed to be a secure browser-based database, and therefore could be considered vulnerable.

From the database area this work is build on Bagade and Dhende (2012) where the thesis is proposing and designing s performance model to theoretically measure the performance of client and server side databases.

Based on this research, we conclude that browser-based databases should perform, in theory, faster than server-side databases, but there is no existing model which can verify that. One of the principles in relation to identifying the performance of browser-based storages is to find out the effectiveness of using IndexedDB in contrast to server-side databases.

A model for measuring the performance of browser-based databases will be designed and tested in Chapter 3. The following chapter will consider the, non-functional, requirement, speed, and a performance model will be compared to the experimental results.

Chapter 3. A Performance Model for Client-side Databases

The aim of this chapter is to investigate the speed, effectiveness, and the possible advantages of using an IndexedDB client-side browser-based database as opposed to a server-side database. In this section a performance model will be proposed, for which the main objective is to identify the effectiveness of client-side databases. This chapter demonstrates the effectiveness of client-side databases by comparing theoretical results against experimental. The scope of the model is to show that client-side databases have significant performance advantage over server-side databases.

Simulations of the model have been completed with the purpose of comparing the results of simulation to a real life experiment. Based on the results and the comparison, the conclusion outlines the effectiveness of using client-side databases over server-side database.

The experiment will seek to compare the performance of traditional SQL databases (Van der Lans, 2006) on server to a NoSQL database (see section 2.6.2) (Kuznetsov et al. 2014, Atzeni et al. 2014, Zachary et al. 2013, Strozzi, 1998) on client-side.

In the literature review sections 2.8.1 and 2.8.2 we discussed the structure and the storage strategy for values in IndexedDB. The data which can be stored in IndexedDB can be a generic JavaScript Object such as a string, a number or a BLOB. The IndexedDB documentation defines the data storage limits for any IndexedDB database to be unlimited. This means that IndexedDB may use all the available space on the user's computer file system or mobile's internal memory.

One approach to testing the storing and reading limits of IndexedDB was via experiments. These experiments were important because they allowed us to see IndexedDB in action, and this helped us to understand the whole structure of how the data is saved and how the transaction functionality works.

We started the experiments with the purpose of overloading IndexedDB with data in order to find out its limits. The purpose was to find out how much data can be stored in IndexedDB in one query and to see if storing large amounts of data can break it. We performed the experiments using a tool developed to test reads and writes of data. This tool was automated: we could specify which database we wanted to test and how much data was to be inserted into this database. The results were shown in a table, with the time of insertion and any possible errors.

The experiment lead to the development of a performance model so that the experimental results could be compared to simulation experiments, in terms of storing and retrieving data from IndexedDB.

For the performance model we used a queuing model as a base. The model can be applied to any browser-based or server based database where variables such as disk speed, bandwidth and queuing time must be considered.

The conclusion of the experiment confirm that IndexedDB performs faster than any other existing database, when storing large numbers of records – e.g. more than 50,000 record in one insert query. Additionally, from the results (Ss. 3.9.2) it can be seen that IndexedDB performed faster in the Firefox browser, where the combination of back end databases is SQLite and IndexedDB (SQL and NoSQL).

3.1. Motivation

Client-side databases are assumed, by their very nature, to perform faster (Pokorny, 2013) than server-side databases, and the demonstration of the performance model, and the experimental results supplied this assumption. Additionally, client-side databases are considered to be a good solution for storage (Walker and Chapra, 2014), because of their speed, effectiveness and offline usage.

The key difference between a server and a client-side database is how the data is served to the user. On the client-side, this is from a local file - the data is not served over an external network - but on the server-side data is served from a server over an external network (McFarland and Nicholson, 2007).

Another notable difference between a server and a client-side database is the structure and the file access time.

There are advantages of using NoSQL (Ss. 2.6.2) over SQL, such as low latency, high performance and high scalability. NoSQL low latency means that the data is better cached which provides faster access (Konstantinou et al. 2011). NoSQL databases are highly scalable, which means that they can handle higher volumes of read and write operations (Tauro et al. 2012).

A database Index (Van der Lans, 2006) is a data structure which improves the efficiency and time of data retrieval operations (SQL statement or procedure) on a database table. The downside of database index is additional writes and increased storage space. The high performance depends on a number of aspects of the database structure, such as the indexing of data (Ayabakan and Kilimci, 2014).

Joins (Van der Lans, 2006) combine records from two or more tables. One of the disadvantages of using a NoSQL database is the lack of an embedded join operation:

complex queries must be handled by complex application code (Tendulkar and Phalak, 2011).

The security in the browser, i.e. SOP (SS. 2.3.2) make IndexedDB only work for web applications using the Hypertext Transfer Protocol (HTTP) or Secure Hypertext Transfer Protocol (HTTPS) (Clark, 2003).

3.1.1 Distributed Databases

A Distributed Database (Özsu and Valduriez, 2011) is a database where parts of the database are stored at multiple locations (multiple servers or computers). Analytical and simulation performance studies of distributed databases generally use queuing systems (Jain, 1991, Gross and Harris, 1985) as underlying models. The advantage of using distributed database is that they are secure by design, providing local storage with increased performance by running queries locally so there is no network bottleneck (Chen and Greenfield, 2013). A bottleneck can be defined as a stage in a process that causes the whole process to slow down or stop (Sevilla et al. 2014).

This section outlines the motivation for this work, as client-side databases are considered to perform faster (Pokorny, 2013) than server-side databases. The work described in this chapter was that of using a queuing model to design a theoretical performance model; the predicted results will then be compared to experimental results.

3.2. Queuing model

The use of a queuing model for both the server and the client-side was decided on in order to demonstrate the effectiveness, and the results were compared in terms of how latency and queuing time impact on performance.

(Medhi, 2003; Bunday 1996) defines Queuing theory as a mathematical study of queues or waiting lines. Therefore, a simple queuing model can be explained in terms of the user arriving at a busy server and joining the queue on that single server. (Whitt, 2000) classifies the queuing processes as either standard (light traffic) or growing (heavy traffic) and summarises limit theorems for each. In theory, as data from the server takes longer in processing because of the network latency and queuing time, as shown in the queuing model (Walker and Shan, 2015; Vilaplana et al. 2014), client-side data can be processed much faster as the latency and queuing time is minimal.

The aim of queuing theory is to get queuing or waiting time of a system, which can be applied to simulations to achieve performance results (Wang et al. 2012; Hohn, 2004; Kalashnikov, 1994).

It is important to realise that server-side databases always have network latency (Ghosh and Rau-Chaplin, 2006) and therefore the queuing time will apply. However, in client-side databases the network latency is minimal and the queuing time will be zero – especially since IndexedDB can handle only one transaction at a time (W3C, 2015). This transaction operation can be a read or a write.

3.3. Database Replication

Database replication (Yadav et al. 2013; Zhou and Wei, 2013) is the process of copying (duplicating) part of database from one database server to another server. Therefore duplicated copies of database are available to be used by users and share equal level of information.

The advantages of using data replication include increased performance and availability (Minhas et al. 2013; Zhou and Wei, 2013). Application reliability can be improved by spreading the data across multiple machines. Also, spreading the data reads across multiple machines can improve reading operation performance (Yadav et al. 2013).

3.3.1 Methods of performing Database Replication

Database replication can be performed via snapshot replication, merging replication or transactional replication. Snapshot replication (Elnikety et al. 2005) (isolation) is when the latest snapshot of one database server is duplicated to another database, which can be on the same or different server.

Merging replication (Mazilu, 2010) is when data from multiple databases are merged into a single database.

Transactional replication (Mazilu, 2010) is when users obtain complete initial copies of the database, with additional frequent updates as data changes, which means that changes are sent to users as they happen.

Database replication involves frequent updates of existing records with a large amount of data. This experiment will seek to compare the replication of data in server databases to that for client-side databases and look at how the frequent update operations will impact on performance.

This section describes an important method which is used to optimise the performance of databases. Another important method is database fragmentation, described in the next section.

3.4. Database fragmentation

Data can be stored on multiple computers by fragmenting (Khan and Hoque, 2012) where the whole database can be split up into several pieces called fragments. These fragments are logical data pieces stored in a distributed database system (Khan and Hoque, 2010; Gibbs et al. 2005; Tamhankar and Ram, 1998). The objective of planned fragmentation is to minimise the data transfer time acquired to execute multiple queries, by locating the required fragments (Abdalla and Amer, 2012). Fragmentation is used to improve database performance (Gorla et al. 2012) with one of three strategies, which includes Vertical Fragmentation, Horizontal Fragmentation and Mixed Fragmentation (Runceanu and Popescu, 2013; Gorla et al. 2012). Horizontal Fragmentation (Abdalla and Amer, 2012; Ezeife, and Zheng 1999) splits tables by row, whereas the tables remain the same as they were. Vertical Fragmentation (Goli e al. 2012; Lim and Ng, 1997) splits tables by column, i.e. one table splits into two or more tables.

3.5. Performance factors

There is a performance difference between server and client-side databases which depends on several factors, such as network latency, as described in 3.5.1 and 3.5.2. These factors played an important in proposing the performance model, which is specified in two parts.

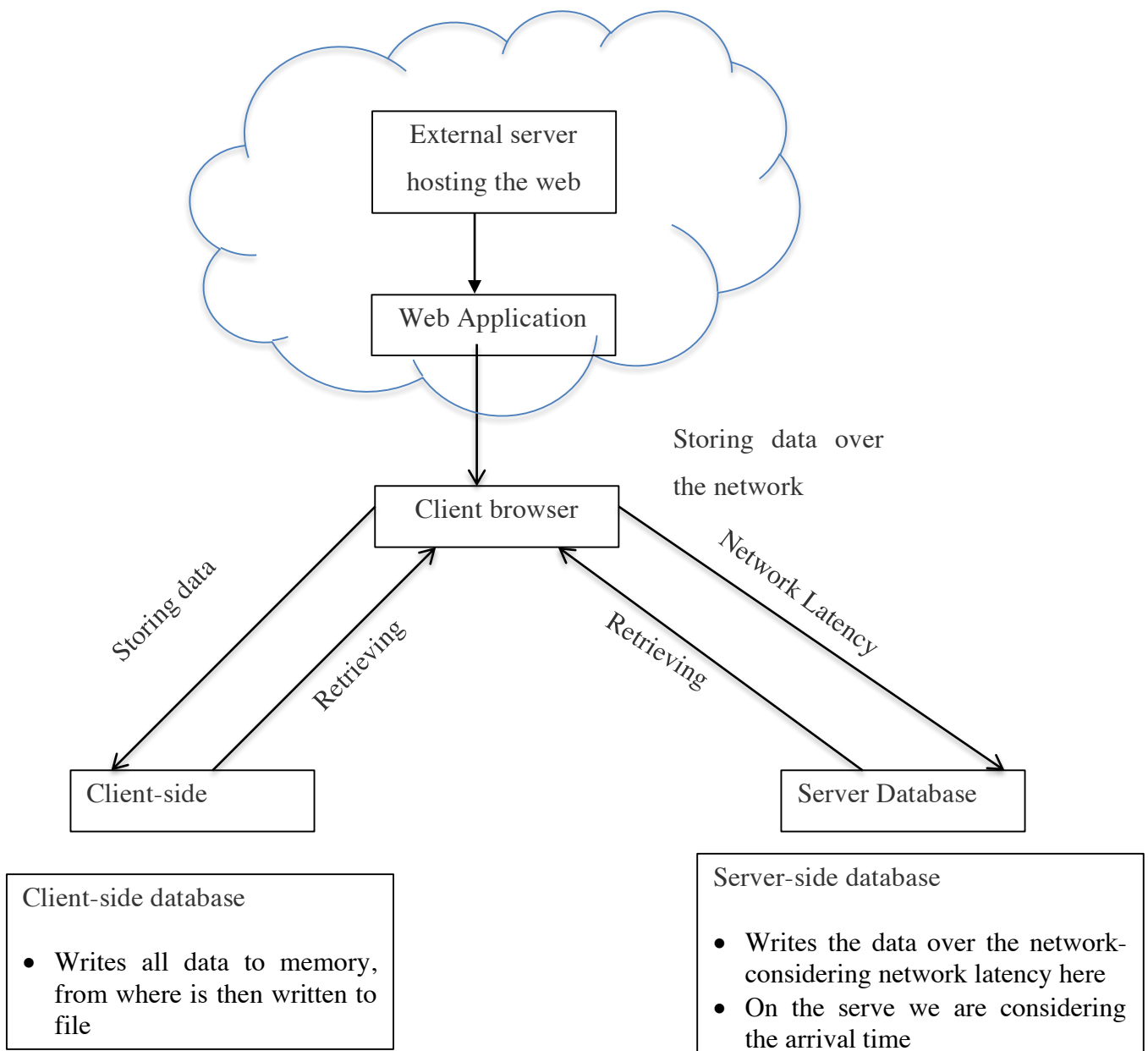


Figure 3-1 Queuing model Diagram – Define the structure

3.5.1 Factors for DB Performance on server

Factors for database performance on the server are as follows: the client network interface, the network bandwidth, the server network interface, server CPU loading, server memory usage, server disk bandwidth and configuration effects.

The client network interface factor can be described as the time taken when due to configuration errors such as network interface (bottlenecks) or hardware malfunction the user is not capable to send or receive packets (Biondi et al. 2014).

Network bandwidth of an overly overfilled network (a network which is too busy) with packets slows down both server replies and client transmissions (Totok and Karamcheti, 2011).

Server network interface delays occur when a server is overwhelmed with packets that cannot receive any more. Interrupt service routine influence the capability of the server to receive packets from the network (Bourke, 2001).

Server CPU loading refers to the server when the server has sufficient CPU cycles, and the summary of processes currently running in queue.

Server memory usage is the performance factor associated to the server memory size and availability (Van der Mei et al. 2001).

Server disk bandwidth delays or bottlenecks occur when the data is not available in memory and the server cannot read the data from the drives quickly (Biondi et al. 2014).

Configuration effects occur with misconfigured settings in which all server services run correctly but unproductively (Biondi et al. 2014).

3.5.2 Factors for DB Performance in local files

The factors for database performance (Bezemer et al.2014; Garrett, 2013) in local files include disk speed, and that can be broken down into average seek time, rotation speed, controller time, average latency, CPU load, memory usage and disk transfer rate.

3.6. Structure of tested databases

In this section, the varying characteristics of different types of databases will be explored. The section will focus on the tested browser-based database, IndexedDB.

As it can be see from Figure 2.8 the structure of IndexedDB (Ss. 2.8.2) consist of Object store, which can contain multiple objects. NoSQL database can contain any number of object stores. The value stored in object store is associated with a key. The object store can store objects as well. First the web application needs to open a connection to database and current version. Then an object store is created which will store the values. Lastly a new record is added in transaction.

For the purpose of testing we have used IndexedDB, LocalStorage (Ss. 2.7.3) (both key/value) and WebSQL (Ss. 2.7.1) (SQL database).

```
var request = indexedDB.open(DB_Name, 3);  
var objStore = db.createObjectStore("name of OS", { autoIncrement  
: true });
```

The person data will consist of these variables and each variable will be added as a separate value into database. `ssn:"111-11-1115",name:"Donna ",age:12,email:"donna123@gmail.org` (Ss.3.9.1.1).

```
var request = objectStore.add(personData[i]);
```

The structure of WebSQL consists of tables and rows where the data is saved.

Compared database on the server was MySQL (Ss.2.6.1) which is a SQL relational database. To insert data into MySQL database table a single query can be used.

```
INSERT INTO table_name (column1, column2, ...) VALUES (value1, value2, ...)
```

3.7. Define the performance model based on the queuing model

This section describes and analyses the proposed performance model. The main purpose of the model is to show the effectiveness of using client-side over server databases. Additionally, the performance model will apply to both client and server-side performance measurements. The proposed performance model is based on a queuing model which, in turn, is based on the Markov model (Shi, 2013).

3.7.1 Model description

The model takes into account the variables and constants which will be applied to it. The model will show the performance effectiveness, where the variable added to the speed model is an average figure for the amount of data. In addition, latency will also be considered for the situation where it is necessary to use over 10 queries for insert and read operations. For the number of queries lower than 10, the latency is considered to be minimal, as there is no queue. Web servers, such as Apache start by default with multiple threads. Where the server can keep multiple copies of itself to server multiple user at the same time.

Based on the theoretical research, it can be expected that it will be more efficient to store data locally after a certain number of queries.

The queuing model will provide the solution for the performance model. Based on the model variables, the experimental results can be further examined in order to evaluate the results.

The M/M/1 queue is the classic, single queue model and M/M/1/N is queuing model with a finite queue. The letter *M* refers to a memoryless (or Markovian) arrival process distribution, that is, to the exponential distribution or Poisson process. The number 1

represents single server. The letter N represents customers only. The $(N+1)$ th customer will not join the queue. $N-1$ represents the maximum number of customers in the queue.

3.7.2 Performance measures of MIM11N queuing system.

In order to highlight different aspects of database performance, a model has been identified. Initially, the queuing system waiting time is calculated. This is important, as there can be many concurrent users who will be performing insert, update or delete actions. Secondly, the average waiting time in the queue is calculated by using data from the previous result. Equation 3.1 is calculating queuing system waiting time.

$$W_s = \frac{L_q}{\lambda(1 - P_N)} + \frac{1}{\mu} \quad (3.1)$$

Table 3-1 Waiting time variables

W_s	Average queuing system waiting time
L_q	Average number of customers
P_N	Represents the probability for the user of not joining the system
μ	Service rate per server = $1/E[s]$
λ	The arrival rate = $1/E[\tau]$
E	Represents amount of times of arriving at state n , and L represent amount of times of leaving state n . $ E - L \in \{0,1\}$
τ	Inter-arrival time which represents time between two succeeding arrivals.
s	Represents service time per operation.

$$W_q = W_s - \frac{1}{\mu} \quad (3.2)$$

Table 3-2 Average waiting time in the queue

W_q	Average waiting time in the queue
W_s :	Average queuing system waiting time
μ	Service rate per server = $1/E[s]$

Equation 3.2 is calculating average waiting time in the queue. Traffic modelling is

concerned with the packet arrival process or Poisson process. The Poisson process is an important model used in queuing theory. Often a Poisson process can describe the arrival behaviour of customers (Kalashnikov, 1994).

3.7.3 Speed model of network transfer

The proposed performance/speed model will have set variables, and can be calculated as show in equation 3.3 (Calculating the minimum access time to total data) and equation 3.4 (Calculating the data transfer time).

$$T = \max\left(D * \frac{c}{N}, \frac{D}{N * I}, \frac{D}{M * I}, \frac{D}{P * R}\right) \quad (3.3)$$

$$S = \frac{D}{T} = \min\left(\frac{N}{c}, N * I, M * I, P * R\right) \quad (3.4)$$

Table 3-3 Data transfer variables

Value	Description	Inserted Value
c	The CPU time to compute each byte	10ms
D	The total of data	100000MB
I	Network speed	100 MB/s
M	The number of I/O nodes	1
N	The number of clients	10
P	The number of disks in parallel	1
R	Disk speed	7200rpm
T	The minimum access time to total data	1s
S	The maximum aggregate bandwidth (Limitation: P/M >=1)	1 machine * (1 gigabit * 2) = 40gbps

3.7.3.1 Other factors to consider

Causes of end-to-end delay described by (Gettys and Nichols, 2012) include processing delays, buffer delays, transmission delays and propagation delays.

The processing delay (Pinto et al. 2013) is the time it takes routers to analyse the packet on network system before it is send. The processing delay is an important factor in the total network delay.

The queuing (or buffer) delay (Xue et al. 2013) is the time a packet waits in a queue before requested packet could be transmitted. Packets which needs to be processed and transmitted arrive at a router and can only be processed one packet at a time.

Transmission delay (Hu et al. 2013) is the amount of time that all the bits of packet leave the sending point.

The propagation delay (Chitre et al. 2012) is the amount of time it takes for the first bit of packet to travel from the sending point to the receiving point.

The time required to insert a row is determined by the time taken to connect to the database, send the query to the server, parse the query, insert the row, update the relevant indexes and then close the connection (Connolly and Begg, 2014).

3.7.4 Hard drive speed

To calculate the disk speed, we can use the equation 3.5 as shown:

$$S = 8 * \frac{B}{\frac{D}{c}} + 8 * \frac{B}{C} \quad (3.5)$$

Where:

- c Speed of light = 299792458 m / s
- S Bits per second
- B Block Size
- D Distance in meters
- C Connect Speed

Also the disk latency can be calculated as shown in equation 3.6:

$$L = 1000 * \frac{D}{c} \quad (3.6)$$

Where:

- c Speed of light = 299792458 m / s
- L Latency in milliseconds
- D Distance in meters

The physical disk performance must also be considered, as disks can be of different types and models. IOPS are used to define the performance of a given disk or disk array, shown in equation 3.7.

$$IOPS = \frac{1}{A_L + A_S} \quad (3.7)$$

Where:

- A_L Average Latency
- A_S Average Seek

3.8. Experiment Validation

For the validation of the performance model, one user will be considered as the default server configuration. This section takes the performance model and applies the set variables to simulate the predicted results. These results will be later compared to experiments results, and then a conclusion of the effectiveness of the databases will be made.

3.8.1 Simulation of the performance model

For the simulation, the computational software program Wolfram Mathematica (Wellin, 2013), was used.

The first calculation to be performed was to find the value of T (the minimum access time to total data), with different sets of D (total amount of data). This variable is then applied to a second model, which calculates the speed to transfer and save the data.

The presumption of variable I (network speed) is 100 MB/s, c (CPU) is 10^{-9} seconds = 1 ns. The transferred and to be stored data is in range from 10 MB to 30k MB. Variables as N (number of I/O nodes), M (number of clients) and P (number on disks in parallels) are set to 1. The speed of disk is 200 MB/s. The results of the proposed model are shown in Figure 3.3.

3.9. Experimental Evaluation of model

This section describes the experimental evaluation which assessed the following:

- (1) Experimental Evaluation of performance comparison between client and server-side databases
- (2) Experimental Evaluation of different kinds of client-side databases
- (3) IndexedDB performance in different browsers.

Also, this section evaluates the theoretical performance model and compares the results to the experimental results.

3.9.1 Experimental Environment

The database was, on the client-side, IndexedDB. This browser-based database supports BLOB and JavaScript Objects. The application was tested in Firefox (v.15) and Chrome (v.22). Both of these browser fully support the IndexedDB API and its functionality. The application was set up on the server because IndexedDB does not support local servers. While Firefox uses the latest W3C specifications onupgradeneeded event to determine if a database should be created or upgraded, Chrome still uses the older, and now obsolete, setVersion method.

For the experiments, a Western Digital 7200rpm 40 GB hard drive was used. All the experiments and the conclusions are based on the use of the same SATA HDD.

3.9.1.1 Data model (Size of web record)

The data model consists of a query which saves and reads data from one simple table. The experiment consists of a functionality which adds records to a database. The records are randomly generated as per the following example: `ssn:"111-11-1115",name:"Donna",age:12,email:" donna123@gmail.org"`. The records are all objects of size 151 Bytes.

The ssn is the key of the object stored in the database, which is also randomly generated. The generator is a JavaScript random number generator. The purpose of the ssn key is as a keyPath (Atzeni et al. 2014) that is the property that makes an individual object in the store unique. The experiment measures the time needed to generate the values and the time needed for insertion. For the experiment evaluation and comparison the insertion is the only important part. The experiment firstly opens a database connection, and creates an object store for storing the generated records.

3.9.1.2 Experiments for the Server-side Database

For the server-side database a JavaScript function will generate a random name and surname and insert these data into the database. The JavaScript function consists of an array of names which will be randomly put together and inserted as one array into the database. The scenario is based on a real application where the information concerning a set of people is stored and retrieved to or from a database. The scenario is based on this specific experimental work; it does not consider the security of browser side databases. The code calls the onSuccess function for every record, as the cursor pointing to the records retrieves them one by one and then displays them. All the retrieved data is stored in memory and from there output to the browser; there is no other way to get all the records from the database - this way might be slow in some cases where the database contains lots of data. For the insert operation the experiment is, measure the time from request for insertion to time of response. For retrieval operation, the experiment is, measure time from request to actual data appearing on the screen. This code structure is not optimal, as the retrieving of records to actual output of the screen might take longer than just measuring the time of the response.

3.9.2 Results and analysis of performance

The performance testing of IndexedDB has been compared to other alternatives such as Local storage (Ss. 2.7.3) and WebSQL (currently deprecated) (Ss. 2.7.1). The tests performed were inserting data in a client-side database to show the time of actual insertion. IndexedDB has shown, that it can insert the data fast, in most cases faster than the other alternatives. This has shown that IndexedDB was chosen by W3C as a client-side database because of the potential of fast inserting and reading of the data. The tables below show insertion of records into MySQL with MyISAM and InnoDB types. Comparison between SQL and IndexedDB databases are visibly different and the results show the big potential of client-side storage.

Databases were tested on the web server and on the local machine. They show different times for the insertion of data. The results in the tables show the storage size of data in the databases, and it can be seen that IndexedDB uses more storage than a traditional relational SQL database.

In addition, the experiments examined the performance of IndexedDB client-side databases in multiple browsers. From the results it can be seen that IndexedDB performs faster in Firefox, but in Chrome the performance is lacking. Chrome still uses WebSQL, notwithstanding its deprecation since its performance is fast when compared to IndexedDB.

Additionally, the back-end technology in Firefox uses SQLite to implement IndexedDB, which supports SQL queries and indexes for search optimisation. In contrast, Chrome uses its own backend, LevelDB, which does not support SQL queries and indexes. It can be concluded that the Firefox implementation of IndexedDB is a better solution. The performance depends on the browser, as the Firefox implementation of the IndexedDB API is much more developed than that of Chrome or Internet Explorer (IE). Firefox uses SQLite as a back-end database, and IndexedDB is implemented on top of it. Researchers and developers note that IndexedDB performs faster with SQL as a back-end. In comparison, the Chrome implementation, where IndexedDB is implemented on top of LevelDB (which is NoSQL), is much slower than Firefox. On the other hand WebSQL (deprecated) (W3C, 2010) performs well in Chrome (v.22), whilst Firefox (v.15) support for WebSQL has ceased.

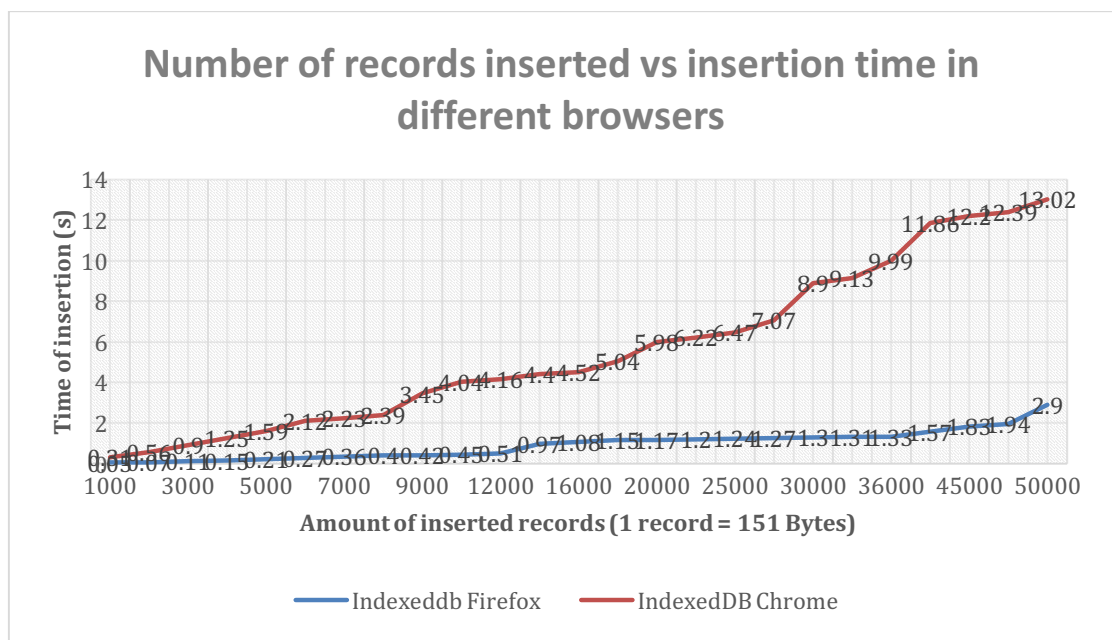


Figure 3-2 Performance testing: Insertion of records into database (IndexedDB in Firefox and Chrome)

From the results, it can be seen that IndexedDB performed, by the end, four times faster in Firefox. The reason is that Firefox implements and maintains the newest code architecture: Chrome is still behind. The results show a significant difference in time, and the possible reason is the use of a back-end database (WebSQL).

Table 3-4 The table is showing insertion times results of tested databases

Test number	Amount of data	WebSQL/Chrome (Insertion time)	LocalStorage/Chrome (Insertion time)	IndexedDB/FF (Insertion time)	MySQL MYISAM (Insertion time)
Test1	1,000	0.09s	0.05s	0.06s	1.09s
Test2	2,000	0.17s	0.2s	0.09s	2.05s
Test3	5,000	0.37s	0.75s	0.15s	5.58s
Test4	10,000	0.81s	1.38s	0.35s	12.178s
Test5	20,000	1.73s	2.93s	0.77s	23.47s
Test8	30,000	2.24s	5,1s	1.25s	31.84s
Test 9	50,000	3.45s	*	1.89s	*
Test 10	100,000	7.35s	*	2.21s	*
Test9	200,000	20.45s	*	4.57s	*
Test10	500,000	50s	*	12s	*
Test11	600,000	76s	*	21.2s	*
Test12	700,000	80s	*	34s	*
Test13	800,000	91s	*	45s	*
Test 14	1,000,000	Failed	*	113s	*

*Length limit of table exceeded.

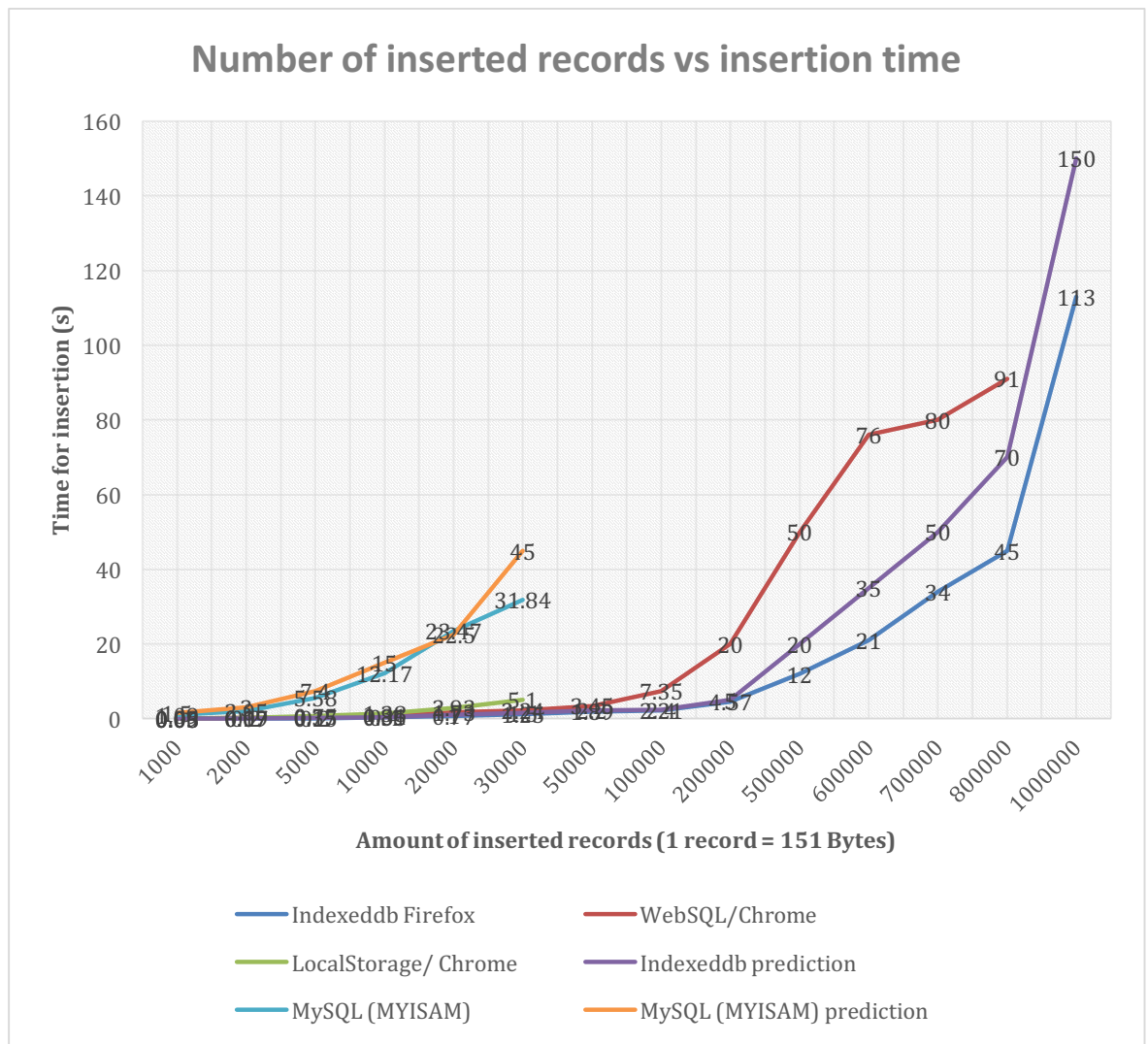


Figure 3-3 Insertion of records into database (IndexedDB, WebSQL, Local Storage, Mysql) with predictions

3.9.3 Discussion of Analysis

The comparison results of inserting records into the databases in Chrome and Firefox can be seen in Figure 3.2. This is that IndexedDB in Firefox handles the insertion of data faster than Chrome. In many forums and support discussion groups developers have noted that the performance is slower in Chrome because the support and code architecture is not updated to fully support IndexedDB. From the results it can be seen that the IndexedDB performs faster than the SQLite database. The insertion size of object data into the database started at 1K, goes up to 500K. The insertion of the objects comparison between the databases is shown in the table.

3.9.3.1 Network Latency

Client-side database process the data on the client-side where the network latency is minimal. All of the data is stored and retrieved from the client machine disk, based on

the web application coding. Comparing the results of experiments, the client-side database handles the data much faster.

3.9.3.2 Scalability

Data scalability is important for any web-based business. The web applications are becoming more scalable to fit the current market, so there is need for a database which will handle this requirement. Client-side databases may have an advantage as they avoid communications costs and other overheads that server-side databases may incur.

3.10. Conclusion

This chapter demonstrated the theoretical performance model and compared its results with experimental results and so analysed the performance behaviours. The purpose of the performance model is to prove that client-side databases have performance advantages. This can be stated as being true, based on the findings regarding network latency, propagation delay, queuing delay and transfer time.

Based on the theoretical performance model predictions and experimental results, it can be concluded that client-side databases perform faster than server-side databases. Also, by comparing the predictions against the experiment results, the final conclusion is that client-side databases perform faster, where IndexedDB was the fastest in read and write operations. Client-side performance efficiency was the main motivation for proposing the performance model. Based on the results there is a strong motivation for using client-side database to store large amount of data (as opposed to storing on server-side databases). The technological implementation differs in browsers (Firefox and Chrome) and the Figure 3.2 shows that browser-based IndexedDB performs four times faster in Firefox browser.

From the results of the performance model for client-side browser-based databases can be seen that IndexedDB browser-based databases perform faster than other comparable client-side databases. It can be also seen from Figure 3.3, that IndexedDB insertion of data was faster even with a great deal of data. Comparing the experimental results to the theoretical model, the results shown in Figure 3.2 indicates that the proposed model v.1 has the insertion time close to the experimental results. From the 3.3 graph can also be seen that the insertion time start to rapidly grow at 200k records. Based on these results, it may be concluded that there is a case for IndexedDB because of its superior performance. A practical usage of InedexedDB is to target mobile devices, where the network connection is not reliable (4G, WIFI).

Chapter 4. A Security Investigation of IndexedDB

In the chapter 2 we identified the security issues associated with the use of IndexedDB, and this chapter will describe experiments which were performed to look deeper at these issues. First we present the experiments which were performed on desktop computers with web browsers Firefox, and Chrome. Second we present the experiments which we performed on mobile devices - which were procedurally the same as the ones carried out on the desktop computer. For these experiments, we used the forensic tools Encase, and XRY (Ss 4.3.9).

The main objective of these investigations was to uncover and examine the security issues associated with the use of IndexedDB. The experiments were motivated by the work described in Chapter 3 which identified possible issues that might impact the security of browser-based databases and the data they hold. Previous chapter have outlined the necessary information on how data is structured and processed within IndexedDB.

The identified experimental steps were based on those used in forensic investigations: the way security professionals would perform the investigation.

The first of these steps was to forensically wipe the hard drive and then restore a previously created Windows 7 x86 SP1 image to ensure consisted results. The second step consisted of running a web application which used IndexedDB as a back-end database. The information stored in IndexedDB by the application is then deleted in a number of different ways: clear browser cache, send to recycle bin, and hard – delete (holding SHIFT +Delete keys). Each deletion method was then subject to a separate investigation. After each deletion process, a forensic acquisition was performed. 'Acquisition,' in this context, means that all relevant data, even that which has been marked as deleted, is made available to be viewed and potentially recoverable.

The result of these experiments was the conclusion that IndexedDB saves the data on the filesystem, and when this data is deleted it is marked as deleted but still physically persists on the media. When web applications run a query and saved new data the old - deleted data - is not overwritten.

4.1. Browser Security Experiments

4.1.1 Introduction

HTML5 reached the official Recommendation stage on 28 October 2014 (W3C, 2014). This means that the World Wide Web Consortium recommended HTML5 as an official

standard. HTML5 was in use by developers and in browsers even as the technology was at specification stage (as a standard). HTML5 adoption will greatly help developers resolve recognized problems such as media and online-data handling; thereby providing a more robust method for handling data (Sarris, 2014). Furthermore, the enhanced functionalities of HTML5, for instance a client-side database called IndexedDB (which is embedded within the web browser), will provide additional benefits such as reducing the web server load. However, while client-side databases have the advantage of reducing load on the web server, their performance will be dependent on the user's web browser - particularly how the browser implements the new client-side database API which is otherwise known as the IndexedDB API.

This chapter will focus on the security of this new browser-based storage capability, and a series of experiments will show how vulnerable the IndexedDB API is to attacks. These attacks will be described in more detail, after which we will propose methods of protection against such attacks. We also investigated how the web application will store the data in the client-side database, and a series of tests were conducted to retrieve deleted database files. A possible solution for storing and retrieving data in a secure manner is proposed and described in further detail.

The testing used the Firefox and Chrome web browsers, as they currently support the IndexedDB client-side database. The investigation will focus on the data storage mechanism of the client-side database. For analysis the results, a forensic tool called EnCase will be used; EnCase is an industry standard computer forensics tool, used in the majority of criminal cases involving the collection and presentation of digital evidence (Encase, 2004). EnCase is a software tool for accessing raw data and providing the functionality to create disk images, which is used to investigate acquired media.

4.1.2 Background

The development of new Web technologies involves compromise between stronger security (thereby protecting the user), and increased functionality (thereby helping the user). Unfortunately, consideration of this trade-off may have resulted in the development and implementation of an insecure API, the IndexedDB API. It should be noted that the current implementations of IndexedDB in existing Web browsers is mostly fully completed. However, it is to be hoped that existing security risks may not persist in future implementations of the IndexedDB API. The security issue resulting from the storage of unencrypted data by IndexedDB has a considerable structural flaw: The database is designed to store all of its data in an unencrypted state.

4.1.2.1 Problem identification

IndexedDB stores data in an unencrypted state. Not all the data stored is sensitive in its own right (as for instance, username and password information is) but it can often include items such as client name, address, place and/or date of birth. If these components are put together, then identity theft is possible.

To prevent the wholesale leakage of data via this security 'hole', we propose an algorithm to secure the information and thus protect the end user from identity theft.

IndexedDB also works on mobile devices, where the data is stored in internal phone storage. Therefore, the problem also exists on mobile devices, and this is an even more serious matter as compared to the situation with desktop PCs. The deleted data can be retrieved from any mobile device. As the data is unencrypted, and considering the situation where the mobile phone is lost or stolen, the security issue is, in fact, much more acute - it is possible to retrieve the deleted data, thus the risk of data exposure, due to the fact that the data stored via IndexedDB is not encrypted, is even higher. Also, compared to storing data on the server-side, where the data is not available to recover after deletion, client-side data storage is less secure.

4.1.2.2 IndexedDB Structure

Files and data stored by the browser are retained on the file storage system, on the device permanent storage. The client-side database, IndexedDB, is a persistent browser-based database, consequently the files reside on the user file system and can be recovered until they are overwritten by other files.

IndexedDB treats file data just like any other type of data. An application can write a file (or BLOB), into IndexedDB, or store strings, numbers or JavaScript Objects (Flanagan, 2011). This is as detailed in the IndexedDB specifications and included in both the Firefox and Chrome implementations of IndexedDB. In Firefox and Chrome's IndexedDB implementation, the files are stored transparently, external to the database; the performance of storing a file in IndexedDB is as good as storing it on filesystem. It does not bloat the database and slow down other operations. Moreover, reading from the file means that the implementation reads from an OS file; therefore, it is just as fast as a filesystem and results in faster retrieve.

The Firefox IndexedDB implementation will, if it is storing the same BLOB in multiple files, create 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; the data is written faster to the filesystem using fewer resources.

IndexedDB is implemented in the browser on top of another database (Ss. 3.9.2). IndexedDB stores the values/objects in the local filesystem which means that the limit of storage is that of the available space on the user's hard drive. When compared to other databases, IndexedDB updates the whole data object rather than just specific data values/fields.

4.1.3 Forensic Tools used

EnCase v.6.11.1 and v.7 (Bunting and Wei, 2006; EnCase, 2004) is a software application available for the Windows Operating System; It enables the forensic examination and extraction of data from a computer and wide range of mobile devices (only in v.7). It provides an efficient and secure method for analysing a wide range of mobile phones through a secure examination process -recovering data in a forensically secure manner. EnCase uses ADB (Android Debug Bridge) to communicate with the mobile device through an USB connection to access device file system. ADB is part of Android Software Developer Kit (SDK). EnCase provides physical and logical extraction of files. Physical extraction operates at a much lower level than Logical and gives access to protected and deleted data such as deleted SQL databases. The extraction dumps the content of the mobile devices' memory. Physical extraction or acquisition gives a bit-by-bit copy of the entire internal flash memory, which allows investigator to conduct analysis.

XRY v.6.7 (XRY, 2015) is a mobile devices forensic software application available for the Windows Operating System. XRY provides SIM card readings, mobile device logical examination and fast recover of live mobile data. XRY allows performing a secure forensic extraction of data from a wide range of mobile devices, e.g., smartphones, tablets, music players, modems and satellite navigation devices.

4.1.4 Potential attack vector

This section considers an unauthorised physical access attack on an IndexedDB file from outside the user's device.

4.1.4.1 Cross-origin resource sharing (CORS) attack.

CORS (Ss. 2.3.3) is a mechanism that can bypass SOP. CORS allows a JavaScript code on a web page from one domain to make `XMLHttpRequests` to another domain.

Scenario 1: Unauthorised physical access to the OS file system where the data from the browser database (IndexedDB) is stored, unencrypted.

Scenario 2 (Data Breach): Unauthorised access from an external machine, bypassing the SOP (SS. 2.3.2) to read the data and retrieve the information stored in the IndexedDB files.

Why is the ability to read and retrieve data stored in the IndexedDB files an issue? In order to highlight the problems which this causes, we shall first conduct an analysis of the IndexedDB database file.

4.1.4.2 Analysis of IndexedDB Database File

The first step in conducting this analysis was to build a ‘clean’ Hard Disk Drive (HDD) on a PC [HP Pavilion 8680 Desktop PC with a Dual Core processor and 4GB Ram, including an Operating System (Windows 7, 32-bit) and web browsers (Firefox v.20.0.1, Chrome v.29.0.1547.620)]. After this, initial Internet browsing was conducted. The HDD was then ‘acquired’ using EnCase v.6.11.1. This was the starting point for each investigation. After each experiment, the disk image needed to be restored to a ‘clean’ state, and so following each experiment the disk was forensically wiped and then the same components (operating system, web browsers) re-installed.

The aim of these experiments was to investigate and show how the data is deleted from an IndexedDB (local file) and also to discover whether the data is held in an unencrypted state. We also performed a re-use of a recovered file to see if that could be successfully achieved.

4.1.4.2.1 Experiment 1: Recovery of deleted IndexedDB SQLite database file

In this experiment, the SQLite database (Ss 2.6.2) file was deleted from a Hard Disk Drive (HDD), on a PC [HP Pavilion 8680 Desktop PC with a Dual Core processor and 4GB Ram running the Windows 7 32-bit Operating System]. Then, using EnCase v.6.11.1, the device was acquired to an image file for analysis of the content of the disk. A write-blocker was used at all times to ensure data writing did not occur during data recovery. The structure of the web browsers (Firefox v.20.0.1 and Chrome v.29.0.1547.62) was also examined to assess how the data is stored.

Experiment 1: Results

Firefox stores all data in a temporary table (SQLite database) from where the data is copied into an Object Store, complete with key/value link. After the data has been copied successfully, the temporary table is dropped. The browsers always store the SQL file in

the same location in the file system. For Firefox, this location is `C:\Users\[user-name]\ApplicationData\Mozilla\Firefox\Profiles\[profilename.default]\indexedDB\[domain-name]\[database-name]`, and for Chrome, `C:\Users[user-name]\AppData\Local\Google\Chrome\UserData\Default\IndexedDB`.

Consequently, any previously stored data is always overwritten, although when the data is deleted from the application (using the delete function), the location within the file system, for that deleted file, is still reserved. Operating system maintains the reserved location because the deleted file still persists on the HDD. So when running the application again, the browser always allocates a different location for the newly created Object Store.

Allocation of file storage in Chrome is slightly different; all of the databases are stored in the same file. Consequently, it was assumed that Chrome is using compression for storing browsing data.

In EnCase the option Copy/UnErase to recover the deleted file was executed and to export the file for further analysis. The option exported the deleted file with all the data. Although the deleted file data can be read from EnCase; instead, we chose to export the file and open it with SQLite Manager (Figure 4.1). The SQLite Manager tabulated the data in a readable way and the field values in the BLOB could be exported unencrypted.

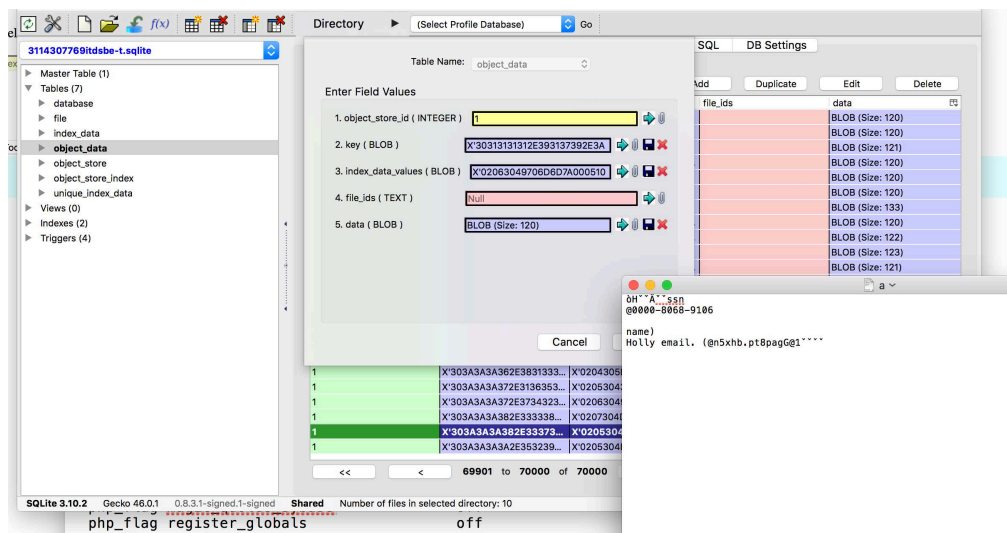


Figure 4-1 Exported deleted database file

4.1.4.2.2 Experiment 2: Clearing the browser cache

Experiments with Firefox included deleting the data by clearing the browser cache (deleting offline data option). Each experiment consisted of storing 300K records with a file size of 127MB. Experiments in Chrome also included deleting the data by clearing the browser cache (clearing browsing data/Hosted app data). Again, each experiment stored 300K records with a file size of 128MB.

4.1.4.2.3 Experiment 2: Results

Clearing the browser cache in Chrome clears the database and deletes the file where it is stored. In Firefox clearing the cache does not delete the database file from the local file system.

4.1.4.2.4 Experiment 3: Re-use of a recovered IndexedDB database

In this experiment the possibility of reusing a recovered IndexedDB database in a different web browser was investigated. This involved identifying the location (physical address on the HDD) of the file after it had been deleted. In addition, this experiment also considered whether the database name was changed after it has been deleted - to see if the web application can read a deleted file with a different filename. When deleting a database from the application, everything in the folders is deleted; including data components that can be stored locally (images, documents, videos, audio). SQLite is not a typed database, which means that any data type can be put into any cell, regardless of the data type declared for the column; the database will attempt to convert it. Similarly, if a different type, other than the column type is requested/retrieved, SQLite will also convert this value.

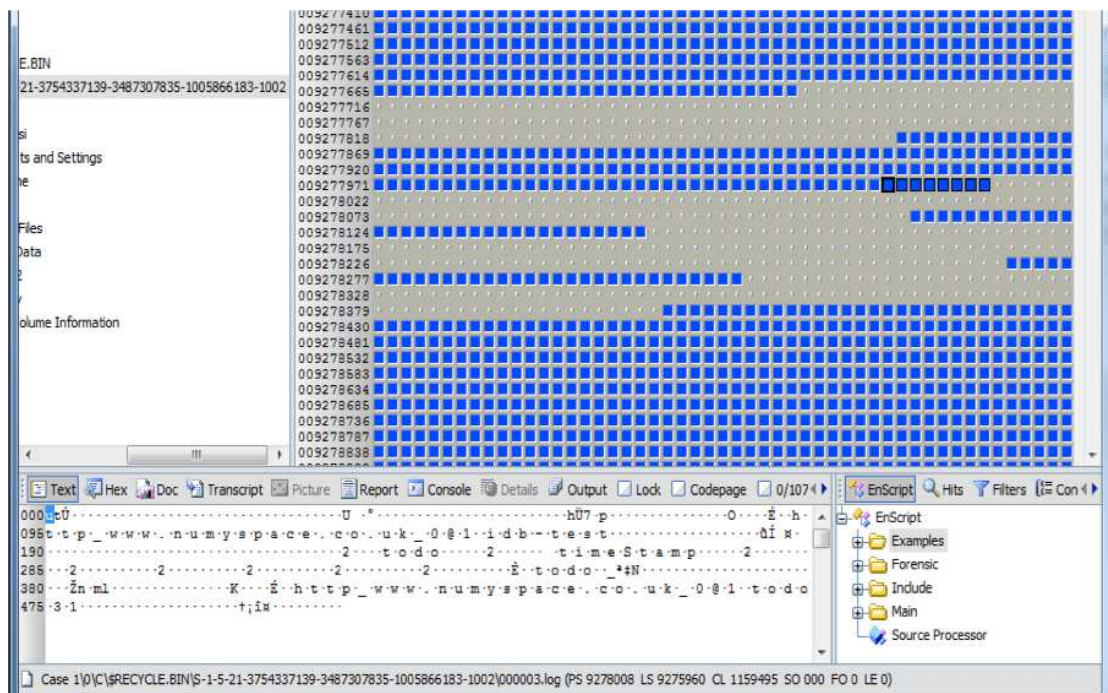


Figure 4-2 The physical address, and data in database file

4.1.4.2.5 Experiment 3: Results

Figure 4.2 displays the physical addresses of the file before and after deletion, which are the same. Deleted files are marked with a red cross. The file was recovered with EnCase

and exported to another hard drive; the file was then copied into a database folder, and the application was run to check if the data could be accessed. The result was that the application read the file and all of the data (in an unencrypted state) which was thus available publicly.

4.1.5 Analysis and Possible Solution

The results of the experiment were as expected; the deleted data has been marked as deleted, and so they can be exported and all the information inside the database, viewable for inspection. Moreover, exported data that has been imported to another PC which is running Windows 7 can be accessed and re-used. However a possible solution to this security issue is presented below.

4.1.5.1 A Proposed Solution to Security issue in IndexedDB

In this section we are going to propose a solution to the IndexedDB storage security issue. A preventative measure against the kind of scenarios which have been described might be the encryption of the files stored by the browser on the file system. All the data stored by the browser would be encrypted first. When retrieving the data, a secure key will be required to read the data from the file system. An encryption library will generate this key to permit access to read the data. Without this key, the data cannot be decrypted and so is impossible to read. The encryption key will be downloaded dynamically and the key (i.e. the password) will be stored in 'session key.' Once the key has been secured, it can be used to encrypt data. When a user closes the browser, the key is overwritten in RAM. This will help to prevent any attacker from getting access to the secure key when reading data from RAM.

The following are the steps required for writing or updating data to the database, also Ss 5.3.1 for algorithm steps.

1. Ensure a secure connection through OAuth2 (Hardt, 2012)- The first step is to provide a secure login functionality for the web application. The web application will use the login functionality to authenticate the user and securely log the user into the system.
2. Open a connection to the database - When an application requests a new transaction, requiring IndexedDB to open the database and save data, the encryption library based extension, which we have designed and implemented, will encrypt the data. This way the data will be stored in an encrypted state and thus is not readable to others.
3. The encryption library generates a secure (symmetric) key from the user entered password. Salt is added to the password. Before the data can be encrypted a key must be

generated and stored with the user information on the server. The client-side encrypts sensitive data using the the key, which will be generated and stored on the server-side. This key is used when encrypting information using the JavaScript library.

4. The key is created using Advanced Encryption Standard (AES) algorithm.
5. Encrypt data. When client-side encryption is enabled, an AES key is generated and the user will be given a session cookie with the key identifier. AES is the algorithm that is used to encrypt data with a key to produce a digital signature (Barth et al. 2011). The key, however, should not be revealed to anyone else. The key is used to decrypt data that has been previously encrypted (Bugiel, 2012). This process uses the AES (Ss 2.11.1) algorithm at 128, 192 or 256 bits with the keyed-hash message authentication code (HMAC) and the SHA256 hash function (Daemen and Rijmen, 2013).
6. Save the file and close the connection to the database.

When reading the data, the following steps need to be fulfilled:

1. Check user credentials. When the user asks to read the data from the database, the web application will first check user credentials (if the session is active) and get the key from the server to allow the decryption of data.
2. Get the key to decrypt the data. Upon successful authentication the user will be given a the key, which will then be used for the decryption of the data. The key will be stored on the server-side, along with all the other user information which is used for encrypting/decrypting the data. OAuth2 was used, which is an open standard for identity authorisation. This standard was used to transfer the key to the server, securely.
3. Decrypt data. The encryption library will check for a matching key, and if this is found, perform the decryption of the data.
4. Display the decrypted data to the user.
5. Close the connection.

To ensure secure authentication (with the server), OAuth2 was used. This provides authentication between the application and the web server using a security token. We do not consider security issues with OAuth2, here, because this will be done in later chapters when the implementation is described.

The stored data in IndexedDB is stored, unencrypted, to the file system (which can be accessed by the web application). When the application sends a request to the web browser to store the data on the local file system, the cryptography library is used to encrypt the data so that it can be stored in a secure fashion. A secure key will be also be

generated and stored on the web server. Reading the data from the local file system will be possible only once a secure key has been provided and the authentication between web application and server is established. Assuming all of these conditions are met and the connection is securely established, the data is decrypted by the cryptography library and displayed through the web browser to the user. In Figure 4.3 we highlight the proposed solution, showing how the cryptography library will be used. The library will be implemented on top of the web browser API. The algorithm will consist of the following components, which are built into the browser (Figure 4.3).

- Mechanism for generating asymmetric key
- Mechanism to salt the password
- Encryption
- Decryption

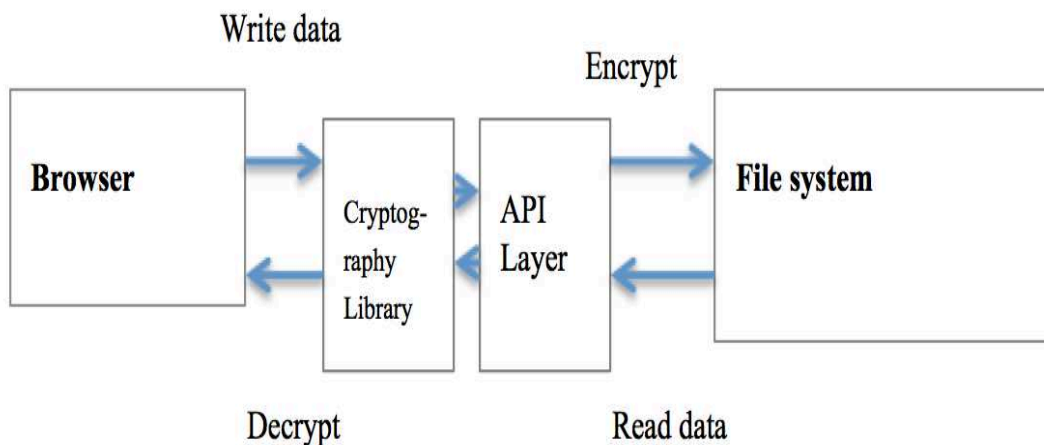


Figure 4-3 Proposed Encryption Library

The cryptography library encrypts readable text into unreadable data. This data can be accessed by using an encryption key. Examples of encryption libraries are listed in Appaendix B. All of these were considered for implementation into the browser. The library chosen provides the functionality to encrypt on the client-side, and also it is available, open source.

Another possible solution to the problem at hand might be to use an external device to store data from the browser. That is, a user could specify a location to which any IndexedDB files should be stored when browsing the web or running web applications. This would include an option whereby the data could be written to, and read from an

external source, such as an USB. The USB key would need to be secured with access encryption and restricted to only accessing data when the master password is entered.

4.1.6 Conclusion

In this chapter, security related flaws within IndexedDB have been demonstrated. While the browser can delete IndexedDB files stored on the local filesystem, these files can still be retrieved by EnCase. Unfortunately, the retrieved data is in an unencrypted format; thus, given the nature of the data held within the IndexedDB API, a potential security issue exists.

All the data stored by IndexedDB is exposed. A solution for this security issue, which includes a security library has been shown, located between the browser and the filesystem. All the data stored by the Indexed DB application will be encrypted and saved via the library. The application needs to read the data, an encryption key is required and without the key, data cannot be decrypted, and so the reading of these data will not be possible. This will help to secure data stored on the client-side and prevent any retrieval of it in an unencrypted state.

New section is going to investigate the security of mobile devices in forensic way, where forensic tools will be used to determine any possible vulnerabilities or security issues.

4.2. Mobile Devices Security Experiments

4.2.1 Introduction

Data storage on mobile devices is not new. People frequently use their smart phones, on the move, to help with everyday tasks - not just for making phones calls. Browsing the Web via mobiles devices is becoming an easier task – the availability of network connections is an important factor in this. Smart phones are powerful pieces of technology wrapped up into small packages; they are capable of tracking the communications, location, and contacts of their users. Everyone is looking to perform the same tasks on their mobile as they do on their computer. Storing data is a significant part of accomplishing these tasks. New web standards have started to define functionalities which extend the basic storage requirements of cookies. There is, therefore, a need for storing larger files on the user's device (file system). This need has resulted from a requirement for faster application response via decreased use of the network. Thus W3C introduced the HTML5 standard, which will help developers to resolve the storage and therefore network latency problem. With the new HTML5 standard come new functionalities, such as the client-side browser-based database called IndexedDB. This will be a standard for storing data on client computer or mobile device.

The requirement for a new client-side database came from developers' feedback highlighting the need for better storage. The web developers and users require more storage space which persists beyond page refresh and is not transmitted to the server. This applies also to mobile devices (smartphones and tablets) where the data can be stored on permanent storage.

Storing data on permanent storage can entail security risks. Based on previous tests and research we found that the stored data is in an unencrypted form. This is potentially dangerous when storing sensitive data: for instance, user personal information, bank or credit card details.

The work described in this chapter was focused on investigating the security of browser-based storage on mobile devices. The storage functionalities of new technologies such as HTML5 are vulnerable to attacks i.e. XSS (Ss. 2.4) and social engineering attacks (Ss. 2.5.2).

We investigated how web application will store the data in the client-side database and performed tests to retrieve ostensibly deleted database files. The possible solution to these security issues, involving storing and retrieving data in a secure manner, is described in further detail.

The testing encompassed both the Firefox and Chrome mobile browsers. These support IndexedDB client-side databases and storing offline data to phone memory. IndexedDB is implemented in mobile browsers in a similar way to that in which it is implemented for computer browsers. The main reason for these tests was to investigate client-side database and mobile browser security, and so the investigation was focused on data storage for the client-side database in the phone memory. To analyse the results and mobile device structure, a forensic tool called XRY was used. The XRY forensic tool is described in more details in section 4.3.9. The investigation's results indicate whether certain actions can be performed, and so show us the potential risks involved such as the retrieving of deleted data from a database. The tests which we decided to perform concentrated on investigating how IndexedDB saves the data and also how the data can be retrieved after deletion. The following section is going to describe the background information on cryptography (relating to storing the data in a secure way). The experiments showed how the data from the client-side could be retrieved with forensic tools. Results will be described in section 5 for Firefox and Chrome browsers.

Mobile applications are another possible means via which stored information on mobiles filesystems can be compromised. An attack can occur when the user downloads an application and installs it because, unknown to the user, the application might include built-in code which can access the user's filesystem. The application might send some of the data to an attacker; the code could be programmed (or could have a function) to search for some specific data such as stored usernames, passwords or personal information. Other possible attacks could be relevant when a web application is using IFrame.

4.2.1.1 Motivation for Work

The new features of web browsers, and new technologies such as HTML5 bring database technology to web browsers. We believe that these features are important for the future of upcoming technologies, especially where the performance for the end user is paramount. This is the motivation for this investigation into mobile security, and storage within the browser-based database system, IndexedDB.

4.2.2 Related Work

In this section we describe related work on mobile forensics and data storage. Also we discuss the limitations of new web technologies in terms of their security aspects.

The Koll (2012) study shows, that only a small amount of data stored on mobile devices is securely deleted. This means that the data can be retrieved from a device after it

has been deleted. The Koll (2012) study also showed that a higher security risk exists when the data is stored in an unencrypted state. This can lead to the theft of sensitive data.

4.2.3 Background

This section describes background information related to the storage of data in a client file system, in an encrypted fashion, and the possible encryption libraries which might be implemented at browser/file system layer. Possible attacks are described in Section 2.4. These attacks are considered to be possible and are known by the HTML5 security community and security researchers (Abgrall et al. 2014). As Abgrall et al. (2014) states, new technologies bring new security risks which need to be countered in order to protect the user.

Because of the increasing number of security vulnerabilities and attacks in the global communication environment, IT security engineers and researchers are constantly trying to develop new or improve existing secure algorithms. Algorithms should provide secure storage of data and made the data easily available for authenticated users (Chaitanya, 2012). One way to provide the secure storage is cryptography (Ss. 2.11). Cryptography is using a key to encrypt stored data on the storage device (Tang et al. 2012), and so the content will remain encrypted even after deletion.

4.2.3.1 Android Internal Memory and Removable Flash.

Android uses the Linux Memory Technology Device (MTD) subsystem to access flash memory storage.

NAND flash (Grupp et al. 2012) memory is a type of long-term persistent storage that retains data without requiring power. NAND flash memory is best suited for flash devices which require large capacity data storage.

The Samsung device uses Robust File System, Samsung (RFS), which supports larger files and journaling. With this, each time a new database is saved onto the file system, a journal file is created. It does this by keeping the file in the cache until the change is finalised. If the process gets interrupted while the file is being saved (for instance, if the battery is pulled or the phone is hard-rebooted), the file system doesn't get corrupted. In Android the SQLite databases are stored under `/data/data/appname/databases`.

4.2.4 Smartphone database file systems

The database files are stored in internal memory. By default the application data and

database files are stored in /data/data directory, and normal user can not access or view them. The files for any file system application are hidden. To access these directories and files, the mobile device needs to be rooted (Bunting and Wei, 2006). Rooting alters the device settings and acquire full administrator or super user privileges. Forensic tools, such as EnCase provide rooting option, which does not delete or alter any data when performed. Rooting a mobile device puts a 'su' binary file into the /system/bin or /system/sbin directory. The benefits of rooting include unlocking hidden features, removing preinstalled applications, boost the performance speed and battery life, installing custom versions of Operating System.

4.2.5 Attack Scenario- cracking the key storage

The first scenario involves unauthorised physical access to a smartphone file system, where the data from the browser database (IndexedDB) is stored unencrypted. The attacker can read the data and get all the information stored in the files, at will.

The second scenario involves unauthorized access from an external machine (running software which is able to bypass the SOP) which reads the data and so retrieves the information stored in the files.

4.2.6 Possible Prevention

The preventative measures against attacks might include the encryption of files stored by the browser on the file system. A browser extension can be built and located between the browser and the file system. This browser extension is built on top of the existing API enable faster implementation. All the data stored by the browser is encrypted before being stored in the file system. When retrieving the data, a secure key is needed in order to read the data from the file system. An encryption library will generate this key. Without this key, the data remains encrypted, making it impossible to read. For testing purposes, an encryption library was attached to a C++ read/write program. Data access was first tested without, and then with encryption in place.

First the data was stored unencrypted in the file system so that it can be accessed by all program.

Next, the encryption library was attached to the program which caused the data to be written in encrypted form - when reading the data, it is decrypted by the library.

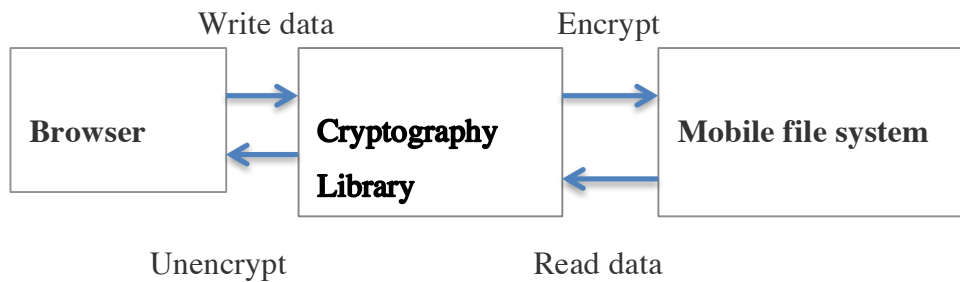


Figure 4-4 Proposed Encryption Library for mobile device

4.2.7 Testing

The testing was done in a number of stages. First we stored distinctive data and created a disk image. Then we deleted the data and obtained the disk image again. This last image was then used to recover the deleted data for comparison with the original image. The steps are described below in more detail.

4.2.7.1 Execute the Following in the Forensic Lab

The tests were done on an Android phone (v 4.3), which does not need to be rooted. This means that the tool can then have access to all system folders and files. The mobile device was a Samsung Galaxy Ace S5830. The EnCase software will root devices automatically (if the option is selected).

The data from browsers is stored in the `/data/data/` folder which by default is not possible to access or view. It is possible to do so only by using forensic tools described in section 4.2.4. Without the use of a forensic tool an export of data can be made but sensitive data might need root access privileges. In Firefox the IndexedDB database is stored in a file with a `.sqlite` extension. The database file pathname is: `data/data/org.mozilla.firefox/files/Mozilla/[randomnumber].default/indexedDB/[domain name]/idb`

4.2.7.2 Storing Data

The first step was to run the application and store the data. The data consisted of email addresses, integers, and text. This helped us cover each possible data format. Data was stored in a random order in mobile storage.

4.2.7.3 Acquisition

Acquisition is process of creation of a complete, physical bit-by-bit image of a mobile device (Ambhire and Meshram, 2012). The EnCase or XRY evidence file is an exact duplicate of the data, as on the mobile device and during the acquisition. Extracted data as shown in Figure 4.5 can be categorised in section, such as files (videos, images, databases), messages, contacts etc.

Before starting with the acquisition of the device, all network connections (WIFI and cellular) has been disabled and removed the SIM card. We performed the acquisition of the RAM disk, and we selected physical acquisition. This meant that more data was accessible– e.g. deleted SQL database files. The result of the acquisition was a full device image.

On mobile devices applications (i.e. browsers) save persistent application or temporary data into the directories /data/data. These directories are hidden - by default. Root permission allows access to this directory's files. Mobile phone memory will be “acquired” with EnCase and this becomes the starting point of the investigation.

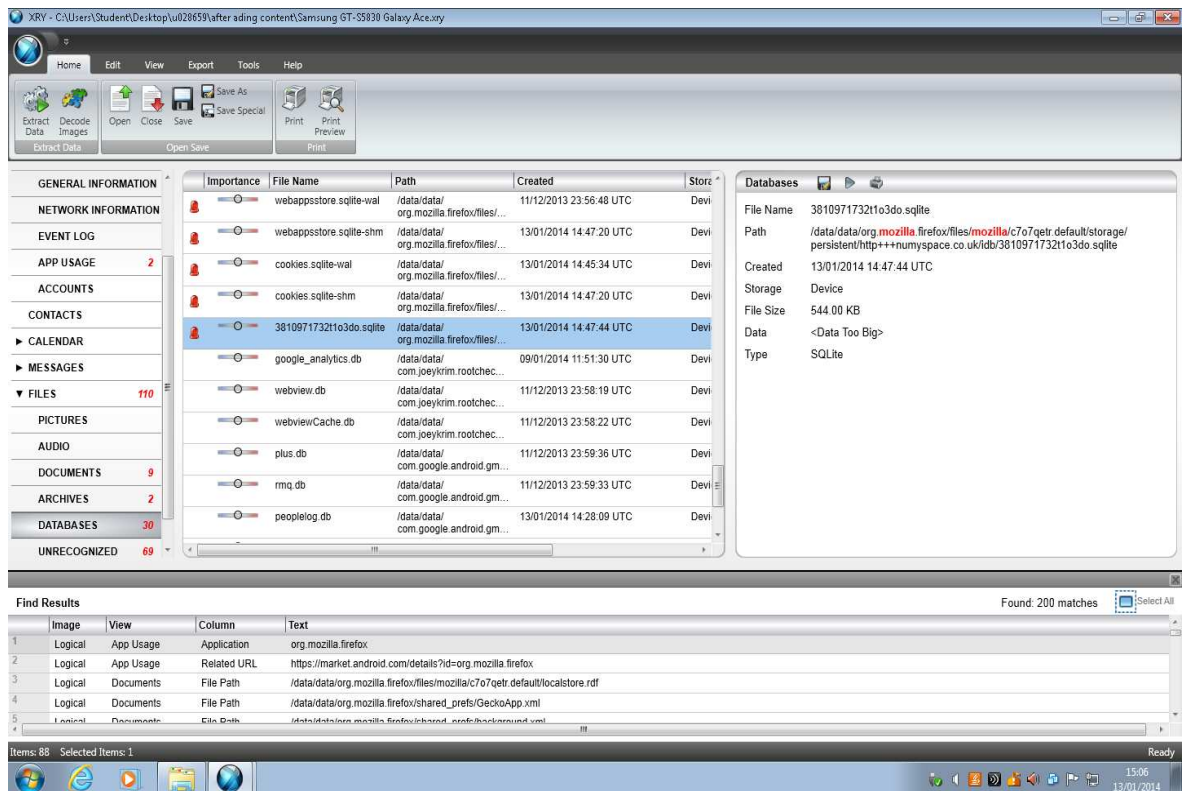


Figure 4-5 Logical extraction, view in XRY

4.2.7.4 Recovering Data

There are number of tools available for displaying deleted or destroyed data. XRY provides a tool called XACT, as show in Figure 4.6. XACT is a viewer which allows us to examine extracted raw hexadecimal data from the physical dump of a mobile device.

4.2.7.5 Analysing Results (Evidence)

With XACT we can examine the data and see where exactly the data is in the mobile device file structure as shown in Figure 4.6.

After each experiment, the disk image needed to be restored to a 'clean' state in order to ensure continuity between tests. We used Android recovery mode to backup and restore a clean install.

The experiments included deleting the data from the phone memory and then, EnCase, locate the deleted data and performing a data recovery. We looked at the structure of browsers (Firefox and Chrome) to see how the data is stored.

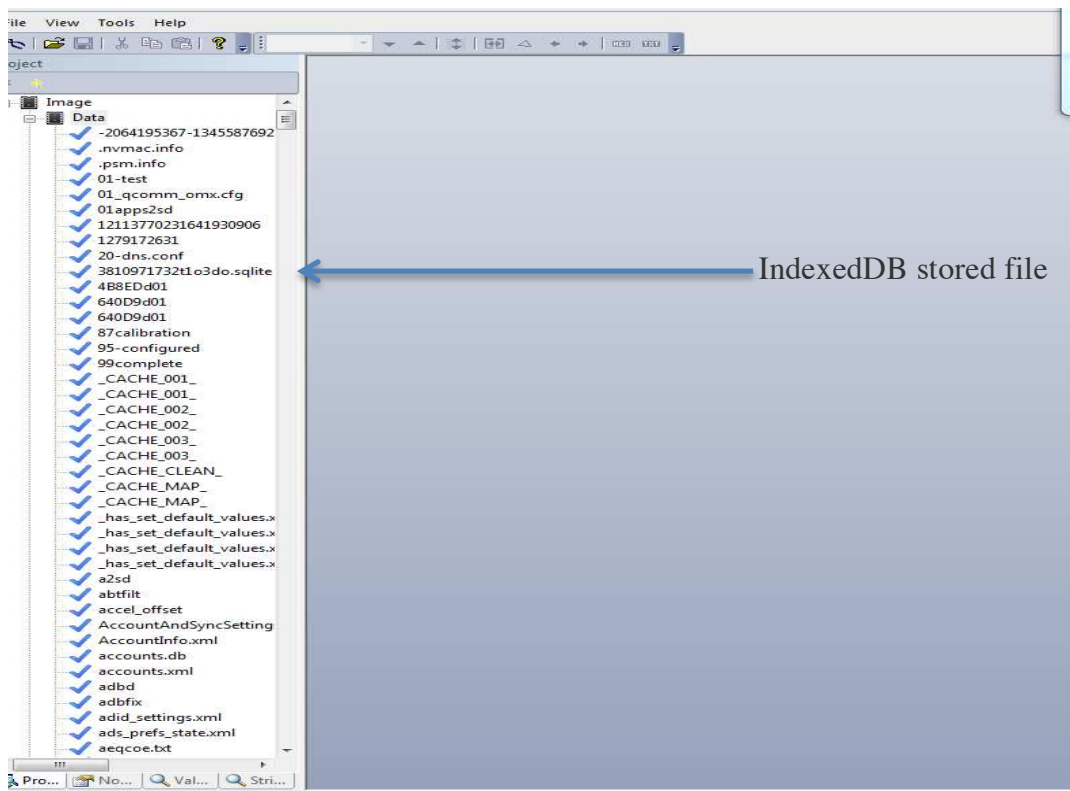


Figure 4-6 Logical extraction, view in XACT

4.2.8 Results

This section provides the test results relating to both, Firefox and Chrome browsers on the mobile devices.

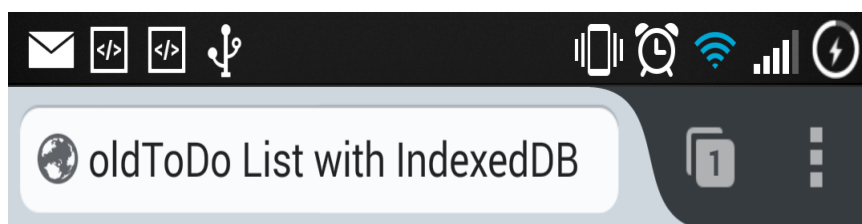
4.2.8.1 Firefox

The stored database file is located in `data/data/org.mozilla.firefox/files/Mozilla/[randomnumber].default/indexedDB/[domain name]/idb`.

From the tests conducted, it was discovered that clearing the cache and offline data, does not delete the database files on any mobile devices. The only possible option for deleting these files is to manually explore the structure, locate the files, and then explicitly delete them using an appropriate Operating System function. The investigation recovered the deleted files and restored the file to perform further testing.

4.2.8.2 Chrome

Deleting IndexedDB files in Chrome is possible by opening Content settings/Website settings and then checking the domain name. By checking the domain name an option to clear stored data will be displayed. After successful confirmation of clearing all data the domain name along with all the data will be deleted. The stored database file is located in `data/data/com.android.chrome/app_chrome/default /indexedDB/[domain name].indexeddb.level1db`. All databases are included in the same log file- `filename.log`; also there are `current`, `lock`, and `manifest` files – the latter providing a pointer to a specific database. Restoring the deleted data from the disk image shows us that all of the deleted database information can be retrieved and used. Deleted data does not differ from the original content in the `.sqlite` file. This confirms our theory that the deleted data can be retrieved, accessed and used with the help of forensic tools.



ToDo list - IndexedDB Example

- test data [Delete]
- 1234567890 [Delete]
- example@example.com [Delete]

What do you need to do?

Todos

Figure 4-7 Web Application on mobile

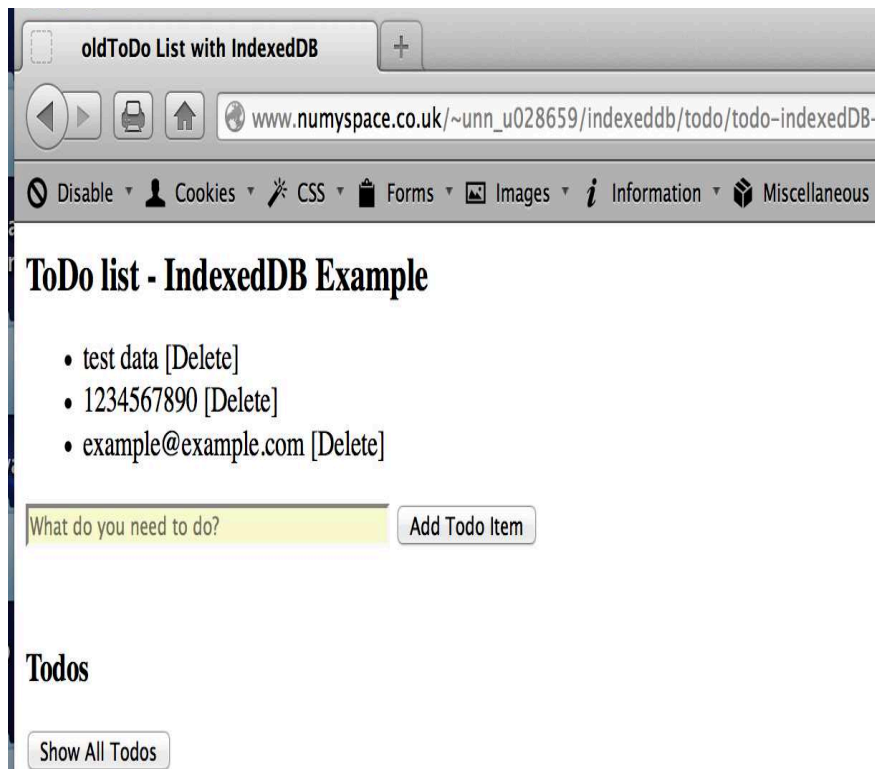


Figure 4-8 Restored file from mobile running on desktop computer

4.2.9 Results of extracted data

For the purpose of testing a web application was run to store data on mobile device as shown in Figure 4.7. The stored data has been deleted using the browser functionality to remove any stored offline data. The device has been connected to desktop computer and forensic tool XRY has been used to extract the data from the mobile device internal storage. The extraction of deleted data on mobile device is using the same structure as on desktop computer, therefore we could reuse the data running the same application on desktop computer as shown in Figure 4.8.

4.2.10 Conclusion

From the tests executed it can be concluded that database data on mobile devices is stored in a format which makes it initially meaningful. The file requires further processing and parsing in order to make the associated content accessible.

The test also showed that restoring the deleted data and accessing it with the application in order to read the data, works. The only significant difficulty discovered when performing these tests was that the file content could not be read just by opening the file.

To delete the data from the mobile device file system a manual file delete approach was needed, as just clearing the browser cache and offline data did not work.

Future work might include further experiments where, for instance, the data from the browser will be store to an external medium. For security reasons a user could specify a location (e.g. a memory card) at which IndexedDB files should be stored when browsing the web or running web applications.

Chapter 5. Browser-based local storage Security Model (BBLs)

The previous chapter identified security issues related to the use of IndexedDB and in this chapter we describe a solution to these problems - in terms of a proposal, and an implementation of this proposal.

In this chapter a security model is presented - which has been built into the Firefox browser as an extension. The chapter fulfils firstly client-side encryption and hashing; existing encryption libraries are considered as the base of the framework and use one for the implementation. Secondly, the framework is extended with Multifactor authentication (Ss 2.12). Lastly the effectiveness of the model is explored by performing attacks, such as XSS (Ss 2.4) from the browser-based local storage.

Based on the issues identified previously in Chapter 4 and in the literature review (Chapter 2), the security issues with IndexedDB can be corrected. This chapter proposes a security model, which will correct the build – in security issue that IndexedDB has by design and make it secure. The security model consists of an encryption library and Multifactor authentication (which is used to secure the database against XSS attacks).

5.1. Introduction

It appears that data stored by IndexedDB on the client file system is unencrypted (Chapter 4). Therefore, any stored data might be at risk of exposure. This means that IndexedDB is not secure by design. This is potentially dangerous when storing sensitive data, such as a user's personal information, bank or credit card details (Ma, 2008). IndexedDB treats file data just like any other type of data. An application can write a file (or BLOB), into IndexedDB, as well as store strings, numbers and JavaScript Objects (Flanagan, 2011).

This thesis intends to design and implement an algorithm that will contain the main components of the JavaScript Encryption Library, which will be a browser-based extension. With this extension, the data will be stored encrypted, and therefore there will be no security risk.

Several encryption libraries are available, which will be brought into consideration for the implementation as shown in Section 5.2.1. The steps of implementation will be described in more detail in section 5.3. An algorithm's steps will be described, which will help structure the functionality and show how the extension will work in several cases of use.

With the implementation of the encryption library in the browser (Firefox v. 29) the

thesis intends to address the ineffectiveness brought about by storing data in an insecure way. Proposing and developing an algorithm which will be implemented into the Mozilla Firefox browser in an extension format. The algorithm will also ensure that the database transaction for storing or retrieving data will only be granted when a secure and valid authentication process is completed. This also relies upon providing the key to encrypt/decrypt data.

5.2. Background

The current concern with browser-based local storage is that it stores data on the client-side unencrypted. In this section possible solutions are explored to minimise the risk of sensitive data being accessed inappropriately.

5.2.1 Client-side Encryption

As described on section 2.11, client-side encryption (encryption in the browser) is not developed as encryption on the server-side, therefore the number of encryption libraries that can be selected is limited. Below is a list of considered client-side encryption libraries.

Table 5-1 List of possible consideration of JavaScript encryption libraries

Library name	Available at	Description
WebCryptoAPI	http://www.w3.org/TR/WebCryptoAPI/	A JavaScript API for performing basic cryptographic operations in web applications, such as hashing, signature generation and verification, and encryption and decryption.
PolyCrypt	http://polycrypt.net/	Pure JavaScript implementation of the WebCrypto API.
crypto-js	https://github.com/glynrob/client-encryption	
gwt-crypto		Google Web Toolkit (GWT)

jscrypto	https://code.google.com/archive/p/crypto-js/	Growing collection of standard and secure cryptographic algorithms implemented in JavaScript.
Stanford JavaScript Crypto library (SJCL)	http://bitwiseshiftleft.github.io/sjcl/doc/	(Stark et al. 2009)

5.2.2 Consideration Results

The decision to choose an encryption library, which could be implemented into browser consist of those factors:

- Size of the library
- Multi browser and multi platform availability and implementation
- Different variable key length of 128, 192, or 256 bits
- Hash functions
- Keyed-hash message authentication codes (HMAC)
- Salt

SJCL offers SHA256 for digesting and AES for encryption with three bit lengths, at 128, 192 or 256 bits. Also provides the SHA256 hash function, the HMAC authentication code, the PBKDF2 password strengthener and the CCM and OCB authenticated-encryption modes. SJCL strengthens the passwords by a factor of 1000 and salts them to protect against rainbow tables, and it authenticates every message it sends to prevent it from being modified. SJCL provides the best security which is practically available in JavaScript. Unfortunately, this is not as great as in desktop applications because it is not feasible to completely protect against code injection, malicious servers and side-channel attacks.

5.3. Algorithm

The algorithm used to save data in this secure way was implemented using a JavaScript library, the proposed Stanford JavaScript Crypto library (SJCL) (Stark et al. 2009).

In the following, the required steps are described for read, write and update the data, using an algorithm in pseudocode.

5.3.1 Algorithm steps to secure data

We are going to propose an algorithm, where initial steps need to be completed to enable the encryption/decryption of data, this will provide functionality with the steps, as described below.

Algorithm 1 Encryption of data

```

procedure Authenticate
  if authenticate = TRUE then
    action login
  else
    action error
  end if

procedure write/read
  if success = TRUE then
    action open DB
  else
    action error
  end if

procedure connect to db
  if success = TRUE then
    action generate key
    action save key
  else if
    action regenerate key
  else
    action terminate
  end if

procedure encrypt
  if key != NULL then
    while Not end do
      action encrypt
    end while
    action success message
  else
    action key not found
  end if

```

Algorithm 2 Decryption of data

```

procedure Authenticate
  if authenticate = TRUE then
    action login
  else
    action error
  end if

procedure read
  if success = TRUE then
    action get key
  else if
    action retry
  else
    action terminate
  end if

procedure request key
  if success = TRUE then
    action keep key
  else
    action close
  end if

procedure decrypt
  if key1 = key2 then
    while Not end do
      action decrypt
    end while
    action success message
  else
    action key not found
  end if

```


5.3.1.1 Obtain 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 the system.

5.3.1.2 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 by others.

If connecting over HTTPS, then connection is more secure as the browser will detect a modified JavaScript file. The Secure Sockets Layer (SSL) of HTTPS protocol handles this.

5.3.1.3 Store the symmetric key

When the data is encrypted a key will be generated and stored with the user information on the server. The client-side encrypts sensitive data using the the key, which will be generated and stored on the server-side. This key is used when encrypting information using the JavaScript library. When client-side encryption is enabled, AES key is generated and the user will be given a session key, which can be used until the web browser is not closed. AES is the algorithm that is used to encrypt and decrypt data with the same key (Burnett, 2001). The key, however, is never revealed anyone else. The data is decrypted using the secure key. (Bernett, 2001)

The symmetric key is generated form the user password, which is also salted by AES algorithm.

5.3.1.4 Decryption of data

When the user makes a request to read the data from the database, the web application will check the user's credentials (if the session is active) and get the key from the server to allow decryption of data. The HTTPS protocol is always the preferred method of exchanging any confidential information.

5.3.1.5 User Authentication

Upon successful authentication, the user will be given a session key, which will be used for the encryption or decryption of the data until the browser is not closed. Then the session key will be destroyed. The key key will be stored on the server-side, with all the

user information which is used for decrypting the data. We are going to use OAuth2, which is an open standard for authorisation. This will be used to securely transfer the key from the server to the encryption library on the user device.

5.3.1.6 Deletion of data

Secure deletion of data will be required – i.e. the overwriting of the data with zeros. This means that the data cannot be read again, as all of the values are set to zero in place of the original data.

5.3.2 Proposal

Hashing (Ss.2.11.2) and encryption (Ss. 2.11) can be done within browsers through the JavaScript encryption library. Our algorithm will use a JavaScript encryption library (proposed SJCL), where the library will be implemented into the browser (Firefox) as an extension. This extension supplement the existing IndexedDB API and therefore every time during the reading or writing of data, the data will be encrypted. The library consists of encryption with symmetric keys. The key will be saved on the server. The session key identifier will be given to the user and stored on the user's machine in the same way as a cookie is. The proposed extension will provide encryption/ decryption of data on the user's machine; this will resolve the issue of storing data in an unencrypted state. It will also provide better security for potential attacks, in which the attacker uses user specific data.

- **Cryptography:** Encryption is the process of encoding original text (plaintext) into an unreadable ciphertext. The encryption key (secret key) specifies how a plaintext is to be encoded.
- **Authorised:** able to decode the ciphertext using the decryption algorithm that requires the secret key.
- **Unauthorised:** Must not have access to the key. By viewing the ciphertext, the unauthorized user should not be able to determine anything about the original plaintext, as it has not been decoded.

5.4. Implementation

The overall structure of the proposed model can be seen in Figure 5.1. The model will add an extra layer between the web browser and IndexedDB API. The security model consists of an algorithmic framework which adds extra protection against issues identified - reading other's data via XSS vulnerabilities.

The proposed encryption model consists of the JavaScript encryption library (proposed JSCL); this 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.

When decrypting the data, a secure key will be required to read the data from the file system. The encryption library will generate this key to permit access to read the data. Without this key, the data cannot be decrypted and so is impossible to read. The encryption key will be downloaded dynamically and the key (i.e. the password) will be stored in 'session key.' Once the key has been secured, it can be used to encrypt data. When a user closes the browser, the key is overwritten and destroyed in RAM. This will help to prevent any attacker from getting access to the secure key when reading data from RAM.

The library consists of encryption and decryption using the same key. As might be expected, the key will be saved on the server. The session key will be given to the user and stored on the user's device, in the same way as a cookie. The extension will provide encryption/decryption of data on the user's device, 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 uses SOP (Ss 2.3.2). This security mechanism, on its own, is not enough to preserve security confidence amongst end users.

The BBLs security model differs from WBSM in a number of ways; these include the security mechanism. The main difference is that the BBLs security model is trying to secure the data between browser and the end user file system, whereas the 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 the client-side database; the user should be able to visit other websites without their databases being compromised.

The current WBSM is not sufficient protection for complex web applications, and stored data on the client-side is becoming more important.

5.4.1 Structure of the library/ encryption

In order to encrypt and decrypt a predefined function `sjcl.decrypt("password", "encrypted-data")` is called. The library uses the AES (Ss 2.11.1) algorithm at 128, 192 or 256 bits, HMAC authentication code and the SHA256 hash function. Also the library can be used with numerous versions of Chrome, Firefox, Internet Explorer Safari and Opera

on Windows, Macintosh and Linux. (Stark et al. 2009).

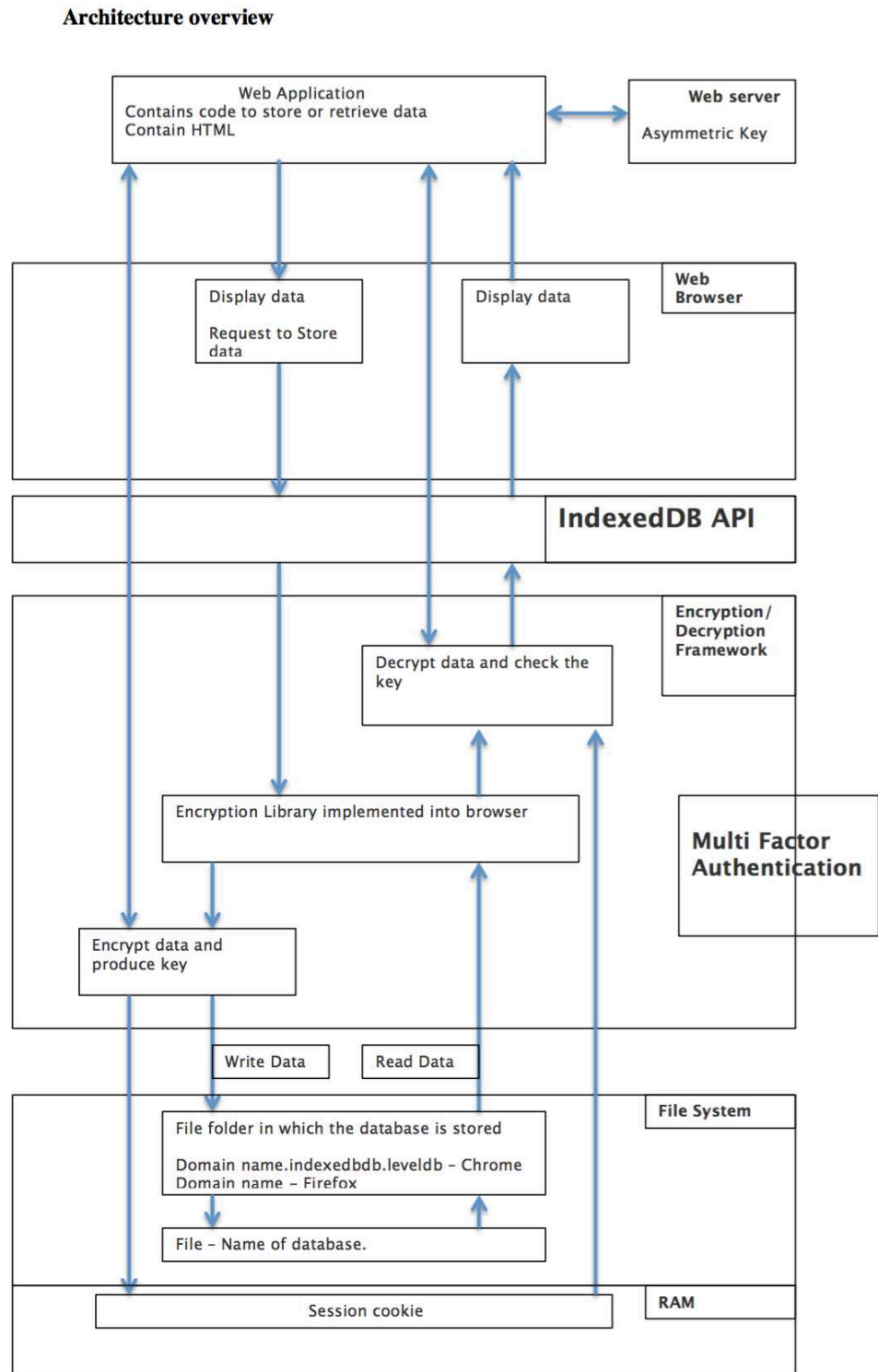


Figure 5-1 Overall structure of the proposed security model

5.4.2 Algorithm implemented

The AES used in the encryption library differs from typical AES implementations in that it uses a different approach, explained by Stark et al. (2009). The implementation speeds up encryption and decryption while keeps the code small.

The Advanced Encryption Standard (AES)

The source code is available for the AES algorithm, also called the Advanced Encryption Standard or the Rijndael algorithm (Daemen and Rijmen, 2013; Hoang, 2012).

The benchmarking tests performed by (Stark et al. 2009) have shown that the SJCL performs faster than other current client-side encryption libraries. The benchmark has been conducted in multiple browsers on Windows, Mac and Linux Operating Systems. One of the reasons we proposed to use and implement the library into our algorithm was the speed and its cross-platform usage.

The algorithm contains this JavaScript encryption library, implemented into the browser. The algorithm will consist of steps which, nevertheless, achieve much 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 asymmetric key will be used to encrypt/decrypt this data.

The Firefox extension file (XPI) file consists of Resource Description Framework (RDF) (it is like an Extensible Markup Language (XML) file with structure, Web Data) and the JavaScript library file. Also there is an installation file, which makes it possible for users to install the extension into browser automatically.

When an application sends a request to the web browser to store the data onto the local file system, the cryptography library will encrypt the data to be stored securely. A secure key will be also generated and stored on the web server.

Reading the data from the local file system will be possible only when a secure key is provided and the authentication between web applications and the server is established. Considering all of the points a connection is securely established, the data is decrypted by the cryptography library and displayed through the web browser to the user.

5.4.3 Implemented Multifactor Authentication

For implementation with the existing encryption library (Figure 5.1) we will use Multifactor authentication (MFA) (Ss 2.12). MFA is used to make the authentication process more secure by adding an extra layer of security. Mobile phones are used as a multi-factor authentication and replace token generated from software applications or physical secure key device for authentication. The extra layer will add something what the user have, for example a code sent to a mobile phone. Web applications such as Facebook or Twitter are using MFA when the user wants to sign in. This is used to verify the user when signing in with sending a verification code to phone via SMS message or

automated phone call. The user can sign in with password (something he know) and verification code from mobile device (something he have).

The extra authentication will need to be passed to make sure the encryption library decrypts the data.

5.4.4 Possible problems with IndexedDB

There are two major problems with the data storage mechanism within IndexedDB;

1. The data is stored in an unencrypted state on the disk. This allows for anyone with access to the device, potentially, to get access to that data.
2. The data remains on the disk until either the site removes it or until the user tells the browser to remove it (e.g. when the user clears the browser cache). That means the data may remain on the disk indefinitely.

Web applications can be used offline with the browser-based local storage, but without a secured connection to the server and an encryption key, a user will not be able to decrypt the database file. This might be an issue, because sometimes a user does not have an Internet connection, but has access to the database files (which, remain encrypted).

Several issues arise with any encryption undertaken by the browser. Firstly, if the encryption library stores the key on the client-side (which is necessary for offline use), can be read via XSS and by any malware on the client. Secondly, if the encryption library stores the key on the server-side, XSS attack code will still be able to read the decrypted data during its usage.

5.4.4.1 Same origin policy attack (spoofing)

A brief overview of the attack:

1. The user would land on an infected page.
2. The page would load a legitimate website by making a request from the attacker's server where Same Origin Policies are not applied.
3. The attacker would inject code in the response to monitor the victims activity.
4. After the victim's credentials were stolen attacker would stop the attack and redirect the user to the original requested page.

Below is an example of using JSON Padding (JSONP) with jQuery code.

```
$.ajax({
  url: "http://website.com/file.json",
  dataType: 'jsonp',
  success: function (data) {
    // Manipulate the response here
  }
});
```

JSONP allows setting url in the header, which is then sent to server and allowing to bypass the Same origin policy.

5.5. Evaluation

In order to evaluate the security model, it was necessary to conduct a number of tests on its implementation(s). The kind of tests which were considered included security attacks – e.g. XSS attacks which bypass the browser's SOP mechanisms.

First an attack on existing security was executed, without applying the new security model and then, with the enhanced and encrypted security model, executed the attack again. The thesis suggests that the model will prevent an attacker, from reading data, because of the authentication process in place. Also the data stored is now encrypted, which means that even if the authentication process is compromised, the data will not be readable in an unencrypted state.

5.5.1 Cost of adding encryption

To analyse the algorithm running, Big O notation (Rutanen, 2013; Danziger, 2010) will be used. The notation determines the amount of steps which needs to be taken to perform a certain function, the complexity of the particular function and the running time. The number of operations can be calculated as the number of basic steps. A basic step is one which is a major part of the algorithm. If $O(f(n))$ is a time bound for the number n of basic step, then $O(f(n))$ is also a bound for the total number of operations, and the running time of the algorithm is $O(f(n))$.

AES is a symmetric block cipher with block length of 128 bits. It allows three different key lengths 128,192 and 256 bits. In encryption process processing of 128 bit keys required for 10 rounds, 192 bit keys required for 12 rounds and 256 bit keys required for 14 rounds. AES is a round based algorithm. In a brute force attack on a cipher with 128-bit keys, we have to check all 2^{128} key combinations by decrypting the ciphertext with each of these values. For encryption and decryption each round has four functions

excepting last round. Last round required three functions. The encryption algorithm has four round functions SubByte(), ShiftRows(), MixColumn() and AddRoundKey(). The decryption, also has the same number of rounds with reverse transformation, order of round function is different, InvShiftRow(), InvSubByte(), AddRoundKey() and InvMixColumn().

The algorithm will take different amounts of time with the same inputs depending on factors such as processor speed, instruction set, disk speed, brand of compiler. The way around is to estimate efficiency of each algorithm asymptotically. The measured time $T(n)$ is the number of elementary steps, considering that each step takes constant time.

Each pair of users will then have two symmetric keys, where only one key is necessary. The number of keys required will therefore be $0.5n(n-1)$ keys. This sum is a quadratic in n which is described in big O notation as show in equation 5.1.

$$O = (n^2) \quad (5.1)$$

The algorithm inner loop is iterating (rounds), therefore the time can be calculated as show in equation 5.2.

$$T(n) = O(n^2) \quad (5.2)$$

$O(n^2)$ represents an algorithm whose performance is directly proportional to the square of the size of the input data set.

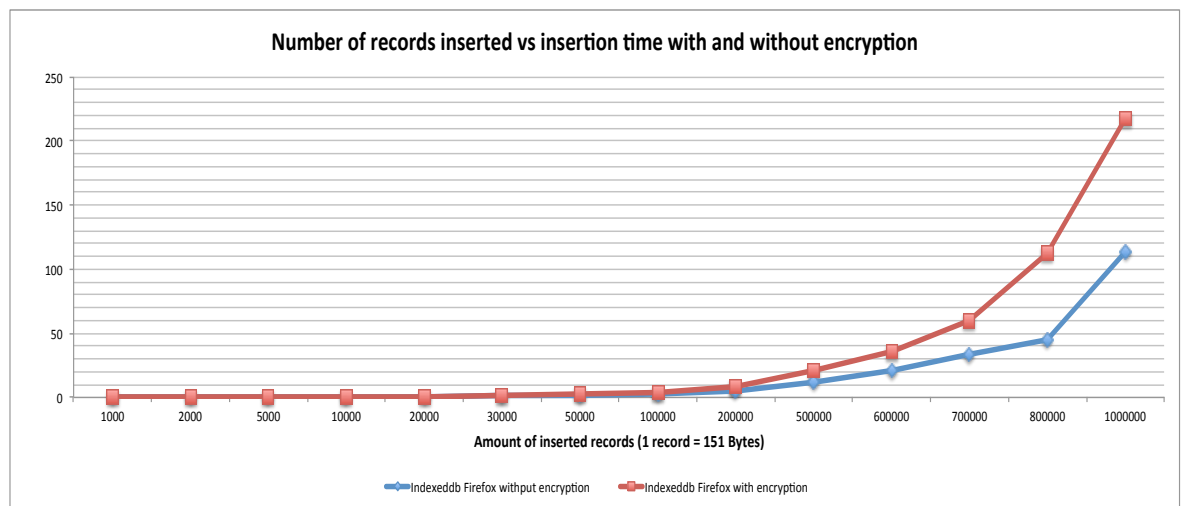


Figure 5-2 Insertion of data into IndexedDB with and without encryption in Firefox browser

The thesis examined the performance impact of adding an encryption/decryption step to the key-value stores of IndexedDB. The results have shown, that enabling encryption slows the process down to 10-40% of its original time as seen in experimental result Figure 5.1. The consequences for the systems read/write latency reach from almost minimal impact (IndexedDB write latency for single read/write operations) to a double increase of latency. Conclusively, it can be said that there is indeed a price to pay in terms of increased latency when applying encryption and decryption to data. However, when using a stronger AES encryption with longer key length (changed from 128 to 256bits) does not gain more overhead than using a weaker encryption with shorter key length. Additionally, the latency of the encryption/decryption process depends on the hardware configuration and optimization. Computers with improved multicore CPU and higher RAM can process the data faster. New CPU has already built in AES functionality, which help to decrease the time of encryption and decryption.

5.6. Conclusion

We have implemented a JavaScript encryption library within the browser in the form of an extension. This extension covers the security issue, that have been identified by the design of IndexedDB. Using the newly developed extension, all data stored on user's device is in an encrypted state, and in order to decrypt, an authentication key is required to obtain private data.

The thesis not considered non-functional requirements, such as speed, at this stage. The algorithm, together with the JavaScript library, will resolve the design issues that IndexedDB has – i.e. the storing of data in an unencrypted state. The steps undertaken in testing showed the functionality of the algorithm. This algorithm must be performed whenever data is to be saved or read in an encrypted form, locally. It will also resolve the issue whereby the data is deleted and then can be read in an unencrypted form afterwards, since the data will be saved in an encrypted state.

Chapter 6. Conclusions and Future Work

This chapter concludes the thesis by summarising the presented work and assessing the research aims and objectives set in Chapter 1. Firstly, a summary of the thesis is presented, then the significance of the contributions and the limitations issues are suggested for future work.

6.1. Conclusions

The Web, or World Wide Web (WWW) (Gauntlett, 2011), is a network of online content, formatted in Hypertext Markup Language (HTML) and accessed via the Hypertext Transfer Protocol (HTTP), where people can share information, photos and videos, buy and sell products or services and participate in online marketing. Therefore, in short, it provides a way of connecting people to information. However, aligned with sharing are concerns about privacy and how personal information is stored and secured.

Web technologies (Feizollahi et al. 2014; Stevens and Owen, 2014) and standards (Fink and Flatow, 2014; Jemel and Serhrouchni, 2014; Lim, 2014) have evolved, especially where different requirements such as local storage and cross-platform availability have been essential. Requirements for new standards arose from Web browser developers, when existing standards were unable to keep pace with rapidly evolving Web technologies (Baloian et al. 2013; Johansson and Andersson, 2013; Andersson and Johansson, 2012). One of these requirements involved finding a better way to store data locally, meaning that web applications (Karthik et al. 2014) could be used offline, without the need for a network connection. Another requirement was browser-based local storage, which would work on multiple browsers and have cross-platform compatibility (Mao and Xin, 2014; Heitkötter et al. 2013). Additionally, local storage or browser-based local storage provides a better alternative for storing data, which in turn reduces network latency (Rumble et al. 2011), reduces network traffic from the server, makes web applications available when a network or server is inaccessible and provides faster response times for web applications.

One HTML5 browser-based storage technology is called IndexedDB (W3C, 2015), which is an asynchronous client-side storage API. IndexedDB offers fast access to large amounts of structured data. The current state of IndexedDB can be considered insecure, as security was not considered in the original specification (W3C, 2015; Chapter 4.). The primary security issue associated with the use of browser-based local storage (IndexedDB) is that data stored locally is in an unencrypted state and readable immediately (Chapter 4). The existing security mechanisms do not provide sufficient

security protection for storing data locally especially since such data may include personal or of sensitive information. This investigation focused on browser-based data storage in operating system files. The analyses described was performed with forensics tools EnCase (Bunting and Wei, 2006; EnCase, 2004) and XRY (XRY, 2015). The experiments focused on an investigation of how IndexedDB saves the data, and also how the data can be retrieved after deletion. Additionally we performed an investigation on a mobile device based on the steps we took for the forensic investigation of the desktop computer. The investigation was conducted on an Android mobile phone and we found the same security issues as on the desktop personal computer. Despite these security issues, browser-based local storages has significant advantages. The primary advantage is the non-functional requirement as speed, which was the principal reason for the development of browser-based local storage (Chapter 3). We designed a model which demonstrates the effectiveness of using the IndexedDB browser-based storage as compared to other browser-based storage and server-side databases. The results of browser-based storage to MySQL server-side databases, restricted WebSQL, and LocalStorage were compared.

IndexedDB browser-based local storage still has issues: although its standardisation is completed (W3C, 2015). One of these issues is the complexity of code required to implement IndexedDB; another is the security implications of browser-based storage. The first concern is not as significant as the second: the code can be adopted from existing examples, and it is then cross browser and multi-platform compatible.

We believe that the existing Web security model does not protect end user data sufficiently and Chapter 4 in this thesis supports this belief. As discussed in Chapter 4, potential security vulnerabilities in web applications can affect the data which is stored via browser-based storage. The existing literature, and Chapter 4 experiments provide evidence for the existence and importance of this problem.

The current state of browser-based local storage design is currently insecure, but this thesis indicates that this can be remedied. An improved security model was proposed then implemented as a browser extension (Chapter 5). This proposed security model enhances the current Web browser security model; the Web browser extension has been implemented with a client-side encryption library (a JavaScript encryption framework). This helps to secure the data stored on the client's machine via the steps described in Section 5.3.1.

We demonstrated that the client-side database is resistant to attacks when our client-side security model is used. The data stored locally will be now remain safe since, even if an attacker gets to the data stored in the database because this data is now in an encrypted

form.

Based on the presented findings, there is a case for browser-based databases, provided that the data security issue is adequately addressed. Conducted experiments show that the enhanced security model implemented fulfilled this requirement. When using the presented security model, all the data stored in the user's machine is in an encrypted state, so that that in order to decrypt, an authentication key is required (to obtain private data).

To protect the data in the browser-based local storage against attacks such as XSS, the encryption library has been extended with Multifactor Authentication (MFA). Using MFA, data in the browser-based storage will be encrypted after successful authentication .The advantage of MFA is that it is resistant to XSS attacks since it is not in the browser.

The proposed model extends the current security model which is not sufficient for complex HTML5 browser-based applications. Based on the implemented evaluation, security model will correct the defence weaknesses in the current model, by protecting the client-side data. It will also resolve the issue of deleted data being available to be read in an unencrypted form, as the data will be saved in an encrypted state. This thesis argues that with the use of the proposed security mechanism, browser-based local storage will be at least as secure as server-side databases.

Existing browser-based local storage security uses the Same Origin Policy (SOP) (Gollman, 2011; Ss 2.3.2) as the main security mechanism. SOP prevents one web page with malicious code to read or obtain access to data on another web page if does not have the same origin. The method checks to see if a request has come from the requesting domain as it is in the list of domain created. If the request domain matches the stored domain value then the application allows the retrieval of the data. The SOP method can be spoofed (Cao et al. 2013; Ss 5.4.4); it is not bullet proof. An attacker can use a third party application which may be programmed to bypass the SOP. Using an XSS attack the attacker can obtain all the data stored in the local database (Ss 5.4.4). Additionally, as data is stored in an unencrypted state, the attacker can easily read the data without using an extra method to decrypt it. Thus SOP is not sufficient to protect the data stored locally.

6.2. Findings

From the experiments performed we have found that the storage of data in an unencrypted state is not the only problem. Browser-based storage faces another issue when deleted data is not fully removed from the hard drive. When utilising forensic and data recovery tools and techniques the recovery restoration of previously deleted data from a computers

and from mobile devices was achieved.

Consequently, these issues which are related to the possible restoration of deleted data by storing data in an unencrypted state - whereby the attacker can get hold of multiple versions of browser-based local storage. Deleted data persists on the hard drive; when a request to delete is executed, the data is merely marked as deleted but still occupies disk space until overwritten. New requests to store data to the browser-based storage will generally result in new partition space being assigned; the old data will persist on the hard drive: it will not be overwritten.

Browser-based local storage has a significant advantage over server-side database storage since the data can be read and written much faster. As part of this thesis, a performance model was developed which proved this speed advantage, and the results of the performance model were then compared to experimental results. There is a need for faster browser-based storage, and this study demonstrated that IndexedDB provides it.

The main contribution of this thesis is in terms of a novel synthesis: a new security model for client-side web databases. This security model includes an encryption framework which helps to secure the data. The encryption framework consists, mostly, of an encryption library which is implemented into the browser. This, however, does not provide full security protection. In addition, an external functionality was required. As well as the encryption library, a multifactor authentication (MFA), or two factors authentication (2FA) (Fleischhacker et al. 2014; Banyal et al. 2013) had to be implemented to prevent XSS (Ss 2.4) attacks; (Ss 5.4.3) for more details on this implementation.

Encrypting the browser-based storage data prevents information from compromise even if the hard disk drive is physically removed, or alternatively if the mobile phone or tablet is stolen.

The encryption used by the security model presented in Chapter 5 was tested from a speed perspective. The proposed performance model includes variables such as encryption and decryption speed, which have an impact on accessing browser-based storage while saving or retrieving data.

In practice the encryption libraries for client-side encryption are not as mature as server-side encryption.

6.3. Future Work

Although the proposed security framework was successfully applied, here, to encrypt browser-based local storage, further improvements could be made to extend it. For instance, biometrics such as fingerprints, retina scans could be used (Chuang and Chen, 2014; Haghghat et al. 2013; Rane et al. 2013; Tresadern et al. 2013), which would be

useful in relation to mobile devices (Meng et al. 2015), particularly since they provide existing hardware which can be used for biometrics. Additionally, present limitations in the performance model could be removed; it could be extended to cover the IE and Safari browsers. Future work might also include implementing a methodology for obtaining usage statistics.

6.3.1 Extensions to the Security Model's Encryption

For encryption, the Stanford JavaScript Crypto library (SJCL) was utilised which provided, an advanced and fully - functional encryption library. Additionally this library could be used across multiple browsers platforms: e.g. Chrome, Firefox, Internet Explorer, Safari and Opera on Windows, Macintosh and Linux. The SJCL library uses the industry standard AES (Ss 2.11.1) algorithm with the SHA256 hash function and the HMAC. SJCL library differs from typical AES implementations in that it uses an alternative approach which keeps the code small while speeds up encryption/decryption. The source code for the AES algorithm, also called Advanced Encryption Standard or the Rijndael algorithm, is available (Daemen and Rijmen, 2013; Hoang, 2012).

One limitation of using client encryption relates to offline storage. When an application is using browser-based local storage for offline usage without a secure connection to the server (and thus access to the encryption key) the user may not be decrypt the database file.

Therefore storing the key on the server-side may be an issue because sometimes a user does not have an Internet connection but has access to database files (in which the files are encrypted): the user will not be able to read or use the files (data).

The problems associated with implementing encryption in the browser include:

- If the encryption library stores the key on the client-side (which is necessary for offline use), then it can be read via XSS and by any malware present on the client.
- If the encryption library stores the key on the server, and the server is vulnerable to an XSS attack, then the attacker will be able to read and decrypt data whilst the key is in use. More details are available in Section 5.4.4.

6.3.2 Biometrics (Ss. 2.11)

The use of biometrics (Ss. 2.11) is an interesting option: particularly in the case of touch-screen devices as they have integrated sensors. Biometrics based verification should not be used independently but rather as an extension of existing password protection. Existing mobile and tablet devices often provide basic biometrics which can extend the

authentication process. Biometrics includes finger or face scanning - extending pattern or pin input. Dynamic biometrics provides better security than static biometrics; the latter can be easily circumvented. Static biometrics are difficult to use within a session-based security scheme where several levels of privacy and security policy are required – for more details Ss 2.11. Using dynamic biometrics, the security of stored data, especially on mobiles and tablets, could be significantly enhanced (Bo et al. 2013); this would be a valuable extension to the proposed security model. Such an extension might include biometrics as a way to authenticate web application requests to retrieve data from local storage - in the same way that biometrics are used for authenticating the action of unlocking the mobile or tablet device.

Mobile and tablet devices are gaining in popularity (Liu et al. 2012), and it is suggested that they will, in future, take precedence over computers in terms of usage.

Mobile and tablet devices have built in sensors so they are more suitable for biometrics.

6.3.2.1 How will biometrics work - future improvements

Existing research is focusing on improving authentication for systems. One of the solutions is biometrics. Static biometrics authentication is not secure and does not provide an adequate security mechanism for modern devices in relation to the level of authentication required. Therefore dynamic biometrics provides a better and much more reliable solution, which is an extension to authentication already in use, such as pin or pattern lock. Existing work on biometrics for mobile devices (Xu et al. 2014; Tresadernet al. 2013; Angulo and Wästlund, 2012; Choraś and Kozik, 2012) suggests that biometrics provide better security for personal data stored on mobile and tablet devices.

6.3.3 Automated attack (penetration) tests

The proposed security model has been tested for resistance to attacks - an automated attack test could be developed for XSS attacks. The security model was tested in relation to XSS attacks (Ss 5.5), but with the further development of current browser-based local storage other attacks could be possible, such as JavaScript injection. Automated attacks tests could be designed as a framework, which will run a series of tests to check the security of browser-based local storage systems. The attacks could include bypassing SOP with CORS (Ss. 2.3.3), XSS attacks, and CSRF (Ss. 2.5.6) attacks.

6.4. Extensions to the performance model

The limitations of this thesis include the lack of experiments on a wide range of web and

mobile browsers. The speed model was applied only to the Chrome and Firefox web browsers. To extend this work, Internet Explorer and Safari web browsers could be used. At the stage when the experiments were done, the functionality of Internet Explorer and Safari was limited: browser-based local storage was not developed to full functionality. Additionally, mobile browsers were not tested for performance. Future improvements of the performance model could include extensions of the model for SSD drives, both client-side and server-side.

6.5. IndexedDB usage statistics

Finally, in terms of future work, there is a lack of usage statistics for browser-based storage. The statistics could be constructed via a survey form, which might show the current usage of IndexedDB by web applications. Additionally, this information could be obtained from the number of API calls from each browser. Based on the statistics results, it could be deduced whether IndexedDB gained the expected popularity or not.

6.6. Summary

In this thesis the effectiveness of browser-based storage, and a solution to its associated security concerns was demonstrated and highlighted the speed advantage of the current browser-based local database IndexedDB. As a research contribution a proposed a model for evaluating this speed advantage in Chapter 3 was designed. The results predicted from this model were compared to experimental results, and the outcome showed that browser-based storage performs faster than server-side databases - even when other factors remain the same (CPU, disk speed, bandwidth, etc.). Additionally the experimental tests demonstrated that browser-based storage can generally store significantly more data for one query than can existing server-side databases without requiring additional resources.

The security issues which have been identified in Chapter 5 were addressed; this thesis primarily focused on correcting these security concerns. Browser-based local storage, such as IndexedDB is insecure by design, due to the stored data being unencrypted. In this thesis the first version of a prototype security model for browser-based storage was been proposed and implemented which helps to protect stored data. This security model corrects the aforementioned security issue, improving the security of the browser-based storage as a typical server-side database security.

The Web holds a great deal of information which needs to be secured in order to protect the privacy of users. Browser-based local databases such as IndexedDB provide a

better methodology for the implementation of web applications since their performance and speed is considerably better than that of server-side databases. Browser-based storage provides for the storage of data from web applications, and the data persists even when the browser session is terminated. Additionally, the browser provides storage which can be used offline, and enable program functionality to be run from the browser without additional installation. Therefore offline storage provides a better user experience, especially on mobile and tablet devices with limited network connectivity and data limitation.

References

- Abdalla, H.I. Amer, A.A. (2012) Dynamic horizontal fragmentation, replication and allocation model in DDBSs. *2012 International Conference on Information Technology and e-Services (ICITeS)*, pp.1,7.
- Abgrall, E. Le Traon, Y. Gombault, S. and Monperrus, M. (2014) Empirical investigation of the web browser attack surface under cross-site scripting: An urgent need for systematic security regression testing. *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 34-41.
- Alkhalaf, M. Bultan, T. Gallegos, J.L. (2012) Verifying client-side input validation functions using string analysis, *34th International Conference on Software Engineering (ICSE)*, pp.947, 957.
- Ambhire, V.R. Meshram, B.B. (2012) Digital Forensic Tools. *IOSR Journal of Engineering*, 2(3), pp.392-398.
- Andersson, K. Johansson, D. (2012) Mobile e-services using HTML5. *2012 IEEE 37th Conference on Local Computer Networks Workshops (LCN Workshops)*, pp. 814-819.
- Angulo, J. Wästlund, E. (2012) Exploring touch-screen biometrics for user identification on smart phones. *Privacy and Identity Management for Life*, pp. 130-143.
- Anthes, G. (2012) HTML5 leads a web revolution. *Magazine Communications of the ACM*, 55(7). pp. 16-17.
- 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*. pp. 800-807.
- Appelt, D. Nguyen, C. D. Briand, L. C. Alshahwan, N. (2014) Automated testing for SQL injection vulnerabilities: An input mutation approach. *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, pp. 259-269.
- Asif, M. Krogstie, J. (2013) Mobile client-side personalization. *2013 International Conference on Privacy and Security in Mobile Systems (PRISMS)* 1(4). pp. 24-27.
- Atterer, R. Wnuk, M. Schmidt, A. (2006) Knowing the user's every move: user activity tracking for website usability evaluation and implicit interaction. *Proceedings of the 15th international conference on World Wide Web (ACM)*. pp. 203-212
- Atzeni, P. Bugiotti, F. Rossi, L. (2014) Uniform access to NoSQL systems, *Information Systems*, 43, pp. 117-133.
- Ayabakan, I. Kilimci, P. (2014) Moving Object Databases-Indexing Algorithms.

References

- International Journal of Computer Theory and Engineering*, 6(6).
- 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].
- Bagade, P. Chandra, A. Dhende, A.B. (2012) Designing performance monitoring tool for NoSQL Cassandra distributed database. In *Education and e-Learning Innovations (ICEELI), 2012 International Conference on* (pp. 1-5).
- Baloian, N. Gutierrez, F. Zurita, G. (2013) An architecture for developing distributed collaborative applications using HTML5. *2013 IEEE 17th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pp.213-220.
- Banker, K. (2010) *MongoDB and e-commerce*. [online] Available at: <http://kylebanker.com/blog/2010/04/30/mongodb-and-ecommerce>. [Accessed 17 March 2012].
- Banyal, R. K. Jain, P. Jain, V. K. (2013) Multi-factor Authentication Framework for Cloud Computing. *2013 Fifth International Conference on Computational Intelligence, Modelling and Simulation (CIMSIm)*, pp.105-110.
- Barbierato, E. Gribaudo, M. Iacono, M. (2014) Performance evaluation of NoSQL big-data applications using multi-formalism models. *Future Generation Computer Systems*, 37, pp.345-353.
- Barth, A. Jackson, C. Hickson, I. (2010) *The web origin concept*. [online] Available at: <http://tools.ietf.org/html/draft-abarth-origin-09>. [Accessed 17 March 2012]
- Barua, A. Shahriar, H. Zulkernine, M. (2011) Server-side Detection of Content Sniffing Attacks. *22nd International Symposium on Software Reliability Engineering (ISSRE)*, pp. 20-29.
- Basharat, I. Azam, F. Muzaffar, A.W. (2012) Article: Database Security and Encryption: A Survey Study. *International Journal of Computer Applications*. 47(12). pp. 28-34.
- Bates, D. Barth, A. Jackson, C. (2010) Regular expressions considered harmful in client-side XSS filters. *19th international conference on World wide web (WWW '10)*, pp. 91-100.
- Bell, S. Vasserman, E. Andresen, D. (2014) Developing and Assessing a Multi-Factor Authentication Protocol for Revocable Distributed Storage in a Mobile Wireless Network. *Proceedings of the International Conference on Security and Management (SAM)*, pp. 1.
- Bernett, S. Paine, S. (2001) *RSA Security's Official Guide to Cryptography*. New York: Osborne/McGraw-Hill.

References

- Bezemer, C. Milon E. Zaidman A. Pouwelse J. (2014) Detecting and analyzing I/O performance regressions, *Journal of Software: Evolution and Process*, 26(12), pp.1193-1212.
- Biondi, A. Melani, A. Bertogna, M. (2014) Hard Constant Bandwidth Server: Comprehensive formulation and critical scenarios. *2014 9th IEEE International Symposium on Industrial Embedded Systems (SIES)*, pp.29-37.
- Bo, C. Zhang, L. Li, X. Y. Huang, Q. Wang, Y. (2013) Silentsense: silent user identification via touch and movement behavioral biometrics. *Proceedings of the 19th annual international conference on Mobile computing & networking*, pp. 187-190.
- Borders, K. Springer, J. Burnside, M. (2012) Chimera: A Declarative Language for Streaming Network Traffic Analysis. *In USENIX Security Symposium*, pp. 365-379.
- Boritz, E. Gyun, W. Sundarraj, P. (2008) Internet privacy in E-commerce: Framework, review and opportunities for future research. *Proceedings of the 41st Hawaii International Conference on System Sciences*, pp.204-256.
- Bourke, T. (2001) *Server load balancing*. Sebastopol: O'Reilly.
- Brooks, J. (2011) *Oracle Gives NoSQL Market a Boost*. Eweek, 28(17), pp.8.
- Bruun, A. Jensen, K. Kristensen, D. (2014) Usability of Single-and Multi-factor Authentication Methods on Tabletops: A Comparative Study. *Human-Centered Software Engineering*, pp. 299-306.
- Bugiel, S. Davi, L. Dmitrienko, A. Fischer, T. Sadeghi, A. R. Shastry, B. (2012) Towards taming privilege-escalation attacks on Android. *19th Annual Network & Distributed System Security Symposium (NDSS)*, 17, pp.18--25).
- Bugliesi, M. Calzavara, S. Focardi, R. Khan, W. (2014) Automatic and robust client-side protection for cookie-based sessions. *In Engineering Secure Software and Systems* (pp. 161-178). Springer International Publishing.
- 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. *2014 IEEE Symposium on Computer Applications and Industrial Electronics (ISCAIE)*, pp. 60-64.
- Bunday, B.D. (1996) *An introduction to queuing theory*. London : Arnold.
- Bunting, S. Wei, W. (2006) *Encase computer forensics: the official EnCE EnCase certified examiner study guide*. San Francisco: Wiley & Sons.
- Burns, J. (2005) *Cross Site Request Forgery: An Introduction to A Common Web Application Weakness*, [Online] Available at: <https://www.nccgroup.trust/us/our->

References

- research/cross-site-request-forgery-an-introduction-to-a-common-web-application-weakness/ [Accessed on 4 March 2012].
- Cao, Y. Rastogi, V. Li, Z. Chen, Y. Moshchuk, A. (2013) Redefining web browser principals with a configurable origin policy. *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp.1-12.
- Casario, M. Elst, P. Brown, Ch. Wormser, N. Hanguiez, C. (2011) *HTML5 Solutions: Essential Techniques for HTML5 Developers*. New York: Apress.
- Castelluccia, C. (2012) Behavioural tracking on the internet: a technical perspective. *European Data Protection: In Good Health?* pp. 21-33
- Chaitanya, K. S. K. Ashutosh, K. Ravi, K. (2012) SECURE STORAGE OF DATA USING CRYPTOGRAPHY FOR NETWORK. *International Journal of Engineering Science and Technology (IJEST)*.
- Chang, F. Dean, J. Ghemawat, S. Hsieh, W.C. Wallach, D.A. Burrows, M. Chandra, T. Fikes, A. and Gruber, R.E (2008) Bigtable: A Distributed Storage System for Structured Data. *CM Transactions on Computer Systems (TOCS)*, 26(2), p.4.
- Chen, S. NG, A. Greenfield, P. (2013) A performance evaluation of distributed database architectures. *Concurrency and Computation: Practice & Experience*, 25(11), pp. 1524-1546.
- Chengjiong, W. (2012) The design of the tree data structure based on relational database, *World Automation Congress (WAC)*, pp.1-3.
- Chitre, M. Motani, M. Shahabudeen, S. (2012) Throughput of Networks With Large Propagation Delays. *IEEE Journal of Oceanic Engineering*, 37(4), pp.645-658.
- Choi, T. Gouda, M. G. (2011) HTTPPI: An HTTP with integrity. 2011 Proceedings of 20th *International Conference Computer Communications and Networks (ICCCN)*, on pp. 1-6.
- Choo, H.L. Oh, S. Jung, J. Kim, H. (2015) The Behavior-Based Analysis Techniques for HTML5 Malicious features. In *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2015 9th International Conference on* (pp. 436-440).
- Choraś, M. Kozik, R. (2012) Contactless palmprint and knuckle biometrics for mobile devices. *Pattern Analysis and Applications*, 15(1), 73-85.
- Chuang, M. C. Chen, M. C. (2014) An anonymous multi-server authenticated key agreement scheme based on trust computing using smart cards and biometrics. *Expert Systems with Applications*, 41(4), 1411-1418.
- Chuang, T. T. Nakatani, K, Chen, J. C. H. Huang, I. L. (2007) Examining the Impact of Organisational and Owner's Characteristics on the Extent of E-commerce Adoption in SMEs. *International Journal of Business and Systems Research*, 1(1), pp. 1751-

References

2018

- Chung, L. Nixon, B.A. Yu, E. Mylopoulos, J. (2000) *Non-functional requirements in software engineering*. 5th ed. New York: Springer Science & Business Media.
- Chuda, D. Kratky, P. Tvarozek, J. (2015) Mouse Clicks Can Recognize Web Page Visitors!. In *Proceedings of the 24th International Conference on World Wide Web Companion*, pp. 21-22.
- Clark, M.P. (2003) *Data Networks, IP and the Internet: Protocols, Design and Operation*. Chichester: Wiley-Blackwell.
- Clarke, J. (2012) *SQL injection attacks and defense*. 2nd ed. Waltham: Elsevier.
- Connolly, T.M. Begg, C.E (2014) *Database Systems: A Practical Approach to Design, Implementation and Management*. 6th ed. Boston: Addison Wesley.
- Cormen, T. H. Leiserson, C. E. Rivest, R.L. Stein, C. (2009) *Introduction to Algorithms*. 3 ed. Cambridge: MIT Press.
- Crockford, D. (2008) *JSON in JavaScript*. [online] Available at: <http://www.json.org>. [Accessed on 17 March 2012]
- Crowther, R. Lennon, J. Blue, A. Wanish, G. (2014) *HTML5 in Action*. Shelter Island: Manning Publications.
- Daemen, J. Rijmen, V. (2013) *The design of Rijndael: AES-the advanced encryption standard*. New York: Springer Science & Business Media.
- Dai, W. (2004) Cryptoo++ Library. Available at: <http://www.cryptopp.com>. Last Accessed : 20 May 2014
- Danziger, P. (2010) Big O Notation. Available at: <http://www.scs.ryerson.ca/~mth110/Handouts/PD/bigO.pdf>. Last Accessed : 20 July 2016
- De Ryck, P. Desmet, L. Piessens, F. Johns, M. (2014) The Browser as a Platform. *Primer on Client-Side Web Security*. Springer International Publishing. pp. 25-32.
- De Ryck, P. Desmet, L. Piessens, F. Joosen, W. (2012) A security analysis of emerging web standards-extended version. *Informatics Section, CW Reports*. [online]. Available at: <https://lirias.kuleuven.be/bitstream/123456789/349398/1/CW622.pdf> [Accessed 10 October 2009].
- De Ryck, P. Desmet, L. Philippaerts, P. Piessens, F. (2011) *A Security Analysis of Next Generation Web Standards, (ENISA)*. European Network and Information Security Agency. [online] Available at: <https://distrinet.cs.kuleuven.be/projects/HTML5-security/> [Accessed 2 February 2016].
- Di Lucca, G. A. Fasolino, A. R. Mastroianni, M. Tramontana, P. (2004) Identifying cross site scripting vulnerabilities in web applications. *26th Annual International Energy Conference In Telecommunications INTELEC*. pp. 71-80.

References

- Dong, X. Chen, Z. Siadati, H. Tople, S. Saxena, P. Liang, Z. (2013) *Protecting sensitive web content from client-side vulnerabilities with CRYPTONS*. In Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security (CCS '13).
- Doxygen (2009) *BeeCrypt C++ API Documentation*. Available at: <http://beecrypt.sourceforge.net/doxygen/c++/index.html>. Accessed : 20 May 2014
- Dozono, H. Yamasaki, N. Nakakuni, M. (2014) A Method for Authentication using Behavior Biometrics on WEB. *Proceedings of the International Conference on Security and Management (SAM)*, pp.1.
- Eilers, C. (2012) *HTML5 Security*. [ebook], Developer Press.
- Elhakeem, Y. F. G. M. Barry, B. I. (2013) Developing a security model to protect websites from cross-site scripting attacks using ZEND framework application. *International Conference on Electrical and Electronics Engineering In Computing (ICCEEE)*. pp. 624-629.
- Elnikety, S. Pedone, F. Zwaenepoel, W. (2005) Database replication using generalized snapshot isolation. *24th IEEE Symposium on Reliable Distributed Systems (SRDS)*, pp.73-84.
- eMarketer (2014) *Worldwide Ecommerce Sales to Increase Nearly 20% in 2014*. [online] Available at: <http://www.emarketer.com/Article/Worldwide-Ecommerce-Sales-Increase-Nearly-20-2014/1011039>. [Accessed 20 September 2015].
- Encase (2004) *EnCase Forensic Edition User Manual*. v.4 [online] Available at: <http://www.guidancesoftware.com>. [Accessed 5 January 2014]
- Engelschall, R. S. (1999) *About the OpenSSL Project*. [online] Available at: <https://www.openssl.org/about>. [Accessed 17 March 2012].
- Ezeife, C.I. Zheng, J. (1999) Measuring the performance of database object horizontal fragmentation schemes. Database Engineering and Applications, 1999. *IDEAS '99. International Symposium Proceedings*, pp.408-414.
- Fang, W. Zhang, J. Hu, B., Zhang, Q. Ha, X. (2011) Graphics and data web publishing for local thermal power plant management information system. *2011 International Conference on Multimedia Technology (ICMT)*, pp. 337-340.
- Feizollahi, S. Shirmohammadi, A. Kahreh, Z. S. Kaherh, M. S. (2014) Investigation the effect of Internet Technology on Performance of services organizations with e-commerce orientations. *Procedia-Social and Behavioral Sciences*, 109, pp.605-609.
- Fielding, R. Reschke, J. (2014) *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*. [online] Available at: <http://tools.ietf.org/html/rfc7230> [Accessed 7 December 2014].

References

- Fink, G. Flatow, I. (2014) Introducing Single Page Applications. *Pro Single Page Application Development*, pp. 3-13.
- Flanagan, D. (2011) *JavaScript: The Definitive Guide Activate Your Web Pages*. 6th edition, Beijing: O'Reilly.
- Fleischhacker, N. Manulis, M. Azodi, A. (2014) A Modular Framework for Multi-Factor Authentication and Key Exchange. *Security Standardisation Research*, pp. 190-214.
- Forfang, C. (2014) *Evaluation of High Performance Key-Value Stores*. MsC, Norges teknisk-naturvitenskapelige universite
- Garrett, P. H. (2013) *Advanced Instrumentation and Computer I/O Design : Defined Accuracy Decision, Control, and Process Applications*. 2nd ed. [e-book], Wiley-Blackwell.
- Garrido, A. Firmenich, S. Rossi, G. Grigera, J. Medina-Medina, N. Harari, I. (2013) Personalized Web Accessibility using Client-Side Refactoring. *Internet Computing, IEEE*. 17(4). pp.58,66.
- Gauntlett, D. (2011) *Making is connecting*. Cambridge: Polity Press.
- Gehani, N. (2011) *The Database Book: Principles & Practice Using MySQL*. 2nd edn. Summit: Silicon Press.
- Gehling, B. Stankard, D. (2005) eCommerce security. *Proceedings of the 2nd annual conference on Information security curriculum development*, pp. 32-37.
- Gettys, J. Nichols, K. (2012) Bufferbloat: dark buffers in the internet. *Communications of the ACM*, 55(1), pp.57-65.
- Ghosh, P. Rau-Chaplin, A. (2006) Performance of Dynamic Web Page Generation for Database-driven Web Sites. 2006. *NWeSP 2006. International Conference on Next Generation Web Services Practices*, pp.56-63.
- Gibbs, M.R., Shanks, G. Lederman, R. (2005) Data quality, database fragmentation and information privacy. *Surveillance and society*, 3(1), pp. 45-58.
- Gihan, D. Karunarathna, D.G.M. Udantha, G.P.D.M. Gunathilake, J.A.I.M. Pathirathna, P.S.P. Rathnayake, R.A.T.L (2011) Database based and RESTful email system with offline web based email client. *Advances in ICT for Emerging Regions (ICTer), 2011 International Conference*.
- Google developers (N/A) *Hosted Apps*. [online] Available at: https://developers.google.com/chrome/apps/docs/developers_guide. [Accessed on 17 March 2012]
- Goli, M. Rankoohi, R. S. Taghi, M. (2012) A new vertical fragmentation algorithm based on ant collective behavior in distributed database systems. *Knowledge and Information Systems*, 30(2), pp. 435-455.

References

- Gollmann, D. (2011) Problems with Same Origin Policy. *Security Protocols XVI SE*, pp 86-92
- Gómez, J. M. Lichtenberg, J. (2007) Intrusion Detection Management System for Ecommerce Security. *Journal of Information Privacy and Security*, 3(4), 19-31.
- Gorla, N. NG, V. LAW, D.M. (2012) Improving database performance with a mixed fragmentation design. *Journal of Intelligent Information Systems*, 39(3), pp. 559-576.
- Gross, D. Harris, C.M. (1985) *Fundamentals of Queueing Theory*. New York: John Wiley & Sons.
- Grupp, L. M. Davis, J. D. Swanson, S. (2012) The bleak future of NAND flash memory. In *Proceedings of the 10th USENIX conference on File and Storage Technologies*, pp. 2-2.
- Gudivada, V.N. Rao, D. Raghavan, V.V. (2014) NoSQL Systems for Big Data. 2014 *IEEE World Congress on Management. Services (SERVICES)*, pp.190-197.
- Gupta, B.B. Tewari, A. Jain, A.K. Agrawal, D.P. (2016) Fighting against phishing attacks: state of the art and future challenges. *Neural Computing and Applications*, pp.1-26.
- Gupta, S. Gupta, B.B. (2014) BDS: browser dependent XSS sanitizer. In: *Book on Cloud-Based Databases with Biometric Applications*, IGI-Global's Advances in Information Security, Privacy, and Ethics (AISPE) series, pp. 174-191. IGI-Global, Hershey.
- Haghighat, M. Zonouz, S. Abdel-Mottaleb, M. (2013) Identification using encrypted biometrics. *Computer Analysis of Images and Patterns*, pp. 440-448).
- Han, E. E. (2015) Detection of Web Application Attacks with Request Length Module and Regex Pattern Analysis. *Genetic and Evolutionary Computing*, pp.157-165.
- Hanna, S. Shin, R. Akhawe, D. Saxena, P. Boehm, A. Song, D. (2010) The Emperor's New APIs: On the (In) Secure Usage of New Client-side Primitives. *31st IEEE Symposium on Security and Privacy*.
- Harjono, J. Ng, G. Kong, D. Lo, J. (2011) Building smarter web applications with HTML5. *Proceedings of the 2010 Conference of the Center for Advanced Studies on Collaborative Research (CASCON '10)*, pp. 402–403.
- Hayes, J. (2012) 'Cookie law': a hostage to fortune?. *Engineering & Technology*, 7(8), 66-69.
- Heitkötter, H. Hanschke, S. Majchrzak, T. A. (2013) Evaluating cross-platform development approaches for mobile applications. *Web information systems and technologies*, pp. 120-138.
- Hilerio, I. (2011) *Building offline access in Metro style apps and websites using HTML5*.

References

- [online] Available at: <http://channel9.msdn.com/Events/BUILD/BUILD2011/PLAT-376T>. [Accessed 20 February 2012].
- Hoang, T. (2012) An efficient FPGA implementation of the Advanced Encryption Standard algorithm. *2012 IEEE RIVF International Conference on Computing and Communication Technologies, Research, Innovation, and Vision for the Future (RIVF)*, pp.1-4.
- Hohn, N. Veitch, D. Papagiannaki, K. Diot, C. (2004) Bridging router performance and queuing theory. *Proceedings of the joint international conference on Measurement and modeling of computer systems (SIGMETRICS '04/Performance '04)*.
- Hoog, A. (2011) *Android Forensics - Investigation, Analysis and Mobile Security for Google Android*. Syngress Publishing.
- Hu, H. Zhang, H. Yu, H. Xu, Y. Li, N. (2013) Minimum transmission delay via spectrum sensing in cognitive radio networks. *Wireless Communications and Networking Conference (WCNC), 2013 IEEE* , pp.4101-4106.
- Huang, L-S. Weinberg, Z. Evans, C. Jackson, C. (2010) Protecting browsers from cross-origin CSS attacks. *Proceedings of the 17th ACM conference on Computer and communications security (CCS '10)*, pp. 619-629.
- Huang, Z. Benyoucef, M. (2013) From e-commerce to social commerce: A close look at design features. *Electronic Commerce Research and Applications*, 12(4), 246-259.
- Hydara, I. Sultan, A.B.M. Zulzalil, H. and Admodisastro, N. (2015) Current state of research on cross-site scripting (XSS)—A systematic literature review. *Information and Software Technology*, 58, pp.170-186.
- Ihrig, C. (2013) JavaScript Object Notation. *Pro Node.js for Developers, SE*, 17, pp. 263-270.
- Jain, R. (1991) *The Art of Computer Systems Performance Analysis of Techniques for Experiment Design, Measurement, Simulation and Modeling*. New York:John Wiley & Sons.
- Jemel, M. Serhrouchni, A. (2014) Content protection and secure synchronization of HTML5 local storage data. *11th Consumer Communications and Networking Conference (CCNC)*, pp. 539-540.
- Jemel, M. Serhrouchni, A. (2014) Security enhancement of HTML5 Local Data Storage. In *Network of the Future (NOF), 2014 International Conference and Workshop on the* (pp. 1-2).
- Jemel, M. Serhrouchni, M. (2014) Security assurance of local data stored by HTML5 web application, *10th International Conference on Information Assurance and Security (IAS)*, Okinawa, 2014, pp. 47-52.

References

- Jemel, M. Serhrouchni, A. (2015) Toward user's devices collaboration to distribute securely the client side storage. In *2015 International Conference on Protocol Engineering (ICPE) and International Conference on New Technologies of Distributed Systems (NTDS)* (pp. 1-6).
- Johansson, D. Andersson, K. (2013) 4th generation e-services—requirements for the development of mobile e-services. *eChallenges e-2013 Conference Proceedings, IIMC International Information Management Corporation*.
- Jones, J. (2014) *E-commerce: measuring, monitoring and gross domestic product*. ONS. [online] Available at: <http://www.ons.gov.uk/ons/rel/gva/national-accounts-articles/e-commerce--measuring--monitoring-and-gross-domestic-product/index.html> [Accessed 10 January 2015]
- Jovanovic, N. Kirda, E. Kruegel, C. (2006) Preventing Cross Site Request Forgery Attacks, *IEEE International Conference on Security and Privacy in Communication Networks (SecureComm)*.
- Käfer, K. (2008) *Cross site request forgery*. [online] Available at: <http://dump.kkaefer.com/csrf-paper.pdf> [Accessed 15 March 2014]
- Kalashnikov, V. (1994) *Mathematical Methods in Queuing Theory*. Springer Science.
- 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. *Proceedings of the 12th ACM international symposium on Mobility management and wireless access*, pp.115-118.
- Katz, J. (2008) *Introduction to modern cryptography*. London: Chapman & Hall/CRC.
- Katz, R. (2000). Netscape and the Law in the Information Age+. *Yale Journal of Law & Technology Yale Journal of Law & Technology*, 2, pp.4-5.
- Khan, S.I. Hoque, A.S.M.L. (2010) A New Technique for Database Fragmentation in Distributed Systems. *International Journal of Computer Applications* (0975 – 8887), 5(9).
- Khan, S.I. Hoque, A.S.M.L. (2012) Scalability and performance analysis of CRUD matrix based fragmentation technique for distributed database. *2012 15th International Conference on Computer and Information Technology (ICCIT)*, pp.567-562.
- Koch, W. (1999) *GPGME – The GNU Privacy Guard*. [online] Available at: <https://www.gnupg.org/index.html>. [Accessed 20 May 2014]
- Koch, W. (2003) *Libgcrypt*. [online] Available at: <http://www.gnu.org/software/libgrypt>. [Accessed 20 May 2014]
- Kofler, M. (2001) *What Is MySQL?* Berkeley: Apress.

References

- Koll, U. T. (2012) *Increased Risk by Unsecured Data on Used Storage Devices*. [online] Available at: http://ww2.laplink.com/documentation/pdf/safeerase/Increased_Risk_WhitePaper.pdf. [Accessed on: 17 March 2012].
- Konheim, A. (2007) *Computer security and cryptography*. New York: Wiley-Interscience.
- Konstantinou, I. Angelou, E. Boumpouka, C. Tsoumakos, D. Koziris, N. (2011) On the elasticity of nosql databases over cloud management platforms. *Proceedings of the 20th ACM international conference on Information and knowledge management*, pp. 2385-2388.
- Kotenko, I. Stepashkin, M. Doynikova, E. (2011) Security analysis of information systems taking into account social engineering attacks. *2011 19th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pp. 611-618.
- Krombholz, K. Hobel, H. Huber, M. Weippl, E. (2015) Advanced social engineering attacks. *Journal of Information Security and applications*, 22, pp.113-122.
- Kuznetsov, S.D. Poskonin, A.V. (2014) NoSQL data management systems. *Programming and Computer Software*, 40(6), pp. 323-332.
- Lan, D. ShuTing, W. Xing, X. Wei, Z. (2013) Analysis and prevention for cross-site scripting attack based on encoding. *4th International Conference on Electronics Information and Emergency Communication (ICEIEC)*, pp.102,105.
- Lanze, F. Panchenko, A. Ponce-Alcaide, I. Engel, T. (2014) Undesired relatives: protection mechanisms against the evil twin attack in IEEE 802.11. *Proceedings of the 10th ACM symposium on QoS and security for wireless and mobile networks* pp. 87-94.
- Leavitt, N. (2010) Will NoSQL Databases Live Up to Their Promise? *IEEE Computer*. 43(2), pp.12 - 14
- Lee, I. Jeong, S., Yeo, S. Moon, J. (2012) A novel method for SQL injection attack detection based on removing SQL query attribute values. *Mathematical and Computer Modelling*, 55(1), pp.58-68.
- Lekies, S. Stock, B. Johns, M. (2013) 25 million flows later: large-scale detection of DOM-based XSS. *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security (CCS '13)*. pp.1193-1204.
- LevelDB (2011) *A fast and lightweight key/value database library*. [online] Available at: <https://github.com/google/leveldb> [Accessed 24 November 2014]
- Lim, S-J. Ng, Y-K. (1997) Vertical fragmentation and allocation in distributed deductive database systems. *Information Systems*, 22(1), pp.1-24.
- Lim, S. H. (2014) Design and Implementation of HTML5 based Hybrid Application for

References

- Mobile Social Networking Service. *In 8th International Conference on Future Generation Communication and Networking.*
- Liu, H. Gong Y. (2013) Analysis and Design on Security of SQLite. International Conference on Computer, Networks and Communication Engineering: Beijing.
- Liu, Q. Zhang, Y. Yang, H. (2013) POSTER: trend of online flash XSS vulnerabilities. *In Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security (CCS '13)*, pp.1421-1424.
- Liu, W. (2012) Research on cloud computing security problem and strategy. *2012 2nd International Conference on Consumer Electronics, Communications and Networks (CECNet)*, pp. 1216-1219.
- Liu, Y. Li, F. Guo, L. Shen, B. Chen, S. (2012) A server's perspective of internet streaming delivery to mobile devices. *2012 Proceedings INFOCOM, IEEE*, pp. 1332-1340.
- Liu, Y. Zhao, W. Wang, D. Fu, L. (2015) A XSS Vulnerability Detection Approach Based on Simulating Browser Behavior. *2nd International Conference on Information Science and Security (ICISS)*, (pp. 1-4).
- Livshits, B. Bace, R. Neville-Neil, G. (2013) Browser security: appearances can be deceiving. *Communications of the ACM*, 56(1).
- Lyubashevsky, V. Masny, D. (2013) Man-in-the-middle secure authentication schemes from LPN and weak PRFs. *In Advances in Cryptology–CRYPTO 2013*, pp. 308-325.
- Malviya, V.K. Saurav, S. Gupta, A. (2013) On Security Issues in Web Applications through Cross Site Scripting (XSS), *Software Engineering Conference (APSEC)*,1, pp.583,588.
- Mansfield-Devine, S. (2010) Divide and conquer: the threats posed by hybrid apps and HTML 5. *Network Security*, 2010(3), pp.4-6. [Online]. Available at: [http://dx.doi.org/10.1016/S1353-4858\(10\)70033-7](http://dx.doi.org/10.1016/S1353-4858(10)70033-7) (Accessed: 12 January 2015).
- Mao, X. Xin, J. (2014) Developing Cross-platform Mobile and Web Apps. *CIGR Proceedings*, 1(1).
- Maras, J. Carlson, J. Crnkovic, I. (2011) Client-side web application slicing. *26th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp.504-507.
- Martinez-Cruz, C. Blanco, I. J. Vila, M. A. (2012) Ontologies versus relational databases: are they so different? A comparison. *Artificial Intelligence Review*, 38(4), 271-290.
- Mayer, J. R. Mitchell, J. C. (2012) Third-party web tracking: Policy and technology. *2012 IEEE Symposium on Security and Privacy (SP)*, pp. 413-427.
- Mazilu, M.C. (2010) Database replication. *Database Systems Journal*, 1(2), pp.33-38.
- McFarland, D. Nicholson, D. B. (2007) Client/Server Computing Basics. *Handbook of*

References

- Computer Networks: Distributed Networks, Network Planning, Control, Management, and New Trends and Applications, 3, pp.1-15.
- Mechanic, A. Rubbelke, L. Kornellis, H. (2007) *Expert SQL Server 2005 Development*. Berkley: Apress.
- Medhi, J. (2003) *Stochastic Models in Queueing Theory*. 2nd ed. San Diego: Elsevier Science.
- Mehta, N. Sicking, J. Graff, E. Popescu, A. Orlow, J. (2012) *Indexed Database API*. [online] Available at: <http://www.w3.org/TR/IndexedDB>. Accessed on: 17 March 2012 [Accessed 10 January 2016]
- Meng, W. Wong, D.S. Furnell, S. Zhou, J. (2015) Surveying the Development of Biometric User Authentication on Mobile Phones. *Communications Surveys & Tutorials, IEEE* , 99, pp.1-1.
- Meucci, M. Keary, E. Cuthbert, D. (2008) *Open Web Application Security Project (OWASP) Testing Guide*. [online] Available at: https://www.owasp.org/images/5/56/OWASP_Testing_Guide_v3.pdf [Accessed 20 February 2014]
- Mewara, B. Bairwa, S. Gajrani, J. (2014) Browser's defenses against reflected cross-site scripting attacks. *International Conference on Signal Propagation and Computer Technology (ICSPCT)*, pp.662,667.
- Minhas, U. Rajagopalan, S. Cully, B. Abounaga, A. Salem, K. Warfield, A. (2013) RemusDB:transparent high availability for database systems. *The VLDB Journal*, 22(1), pp.29-45.
- MSDN (2012) IndexedDB. [online] Available at: [http://msdn.microsoft.com/en-us/library/ie/hh673548\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/ie/hh673548(v=vs.85).aspx). [Accessed 5 January 2014]
- Mozilla developer network (MDN) (2012) *IndexedDB*. [online] Available at: <https://developer.mozilla.org/en/IndexedDB> [Accessed 15 February 2013].
- Mozilla developer network (MDN) (2014) *Web Storage API*. [online] Available at: https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API [Accessed 15 February 2015].
- Myeong, H. W. Paik, J. H. Lee, D. H. (2012) Study on implementation of Secure HTML5 Local Storage. *Journal of Internet Computing and Services*, 13(4), 83-93.
- Naseem, S.Z. Majeed, F. (2013) Extending HTML5 local storage to save more data; efficiently and in more structured way. *Eighth International Conference on Digital Information Management (ICDIM)*, pp.337-340.
- Nikbakhsh, S. Manaf, A. B. A. Zamani, M. Janbeglou, M. (2012) A novel approach for rogue access point detection on the client-side. 2012 26th International Conference on Advanced Information Networking and Applications Workshops (WAINA), pp.

References

684-687.

- OWASP (2010) *OWASP*. [online] Available at: https://www.owasp.org/index.php/Main_Page [Accessed 10 October 2014].
- OWASP (2013) *Top 10 2013-Top 10*. [online] Available at: https://www.owasp.org/index.php/Top_10_2013-Top_10 [Accessed 8 April 2014].
- OWASP (2014) *Password Storage Cheat Sheet*. [online] Available at: https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet [Accessed 12 October 2014].
- Özsu, M.T. Valduriez, P. (2011) *Principles of Distributed Database Systems*. 3rd ed. New York: Springer Science & Business Media.
- Pandey, S. Chauhan, A. S. (2014) An Efficient RC4 Based Secure Content Sniffing for Web Browsers Supporting Text and Image Files. *Advanced Computing, Networking and Informatics*, 2, pp. 325-332.
- Park, J-Y. Yi, O. Choi, J-S. (2010) Methods for practical whitebox cryptography, *International Conference on Information and Communication Technology Convergence (ICTC)*, pp.474-479.
- Patil, P. Bhutkar, S. Ashtaputre, N. Kumar, A. (2012) Amanda Back-up of SQLite. *International Conference on Communication, Information & Computing Technology (ICCICT)*, pp.1-6.
- Pelizzi, R. Sekar, R. (2012) Protection, usability and improvements in reflected XSS filters. *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security (ASIACCS '12)*, pp. 5-5.
- Pigott, D.J. Hobbs, V.J. (2011) Complex knowledge modelling with functional entity relationship diagrams 2011) *VINE*, 41 (2), pp. 192-211.
- Pillai, T.S. Chidambaram, V. Hwang, J-Y. Arpaci-Dusseau, A.C. Arpaci-Dusseau, R.H. (2013) Towards efficient, portable application-level consistency. *9th Workshop on Hot Topics in Dependable Systems (HotDep '13)*, 8 , pp.6.
- Pinto, P. Pinto, A. Ricardo, M. (2013) End-to-end delay estimation using RPL metrics in WSN. *Wireless Days (WD)*, pp. 1-6.
- Pokorny, J. (2013) NoSQL databases: A step to database scalability in web environment (2013) *International Journal of Web Information Systems*, 9 (1), pp. 69-82.
- Pool, P. W. Parnell, J. A. Spillan, J. E. Carraher, S. Lester, D. L. (2006) Are SMEs Meeting 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.

References

- Prince, J. D. (2013) HTML5: Not Just a Substitute for Flash. *Journal of Electronic Resources in Medical Libraries*, 10(2), pp.108-112.
- Ramakrishnan, R. Gehrke, J. (2002) *Database management systems*. 3rd edn. New York: Osborne/McGraw-Hill.
- Ramya, T. Malathi, S. Pratheeksha, G. R. Kumar, V. D. (2014) Personalized authentication procedure for restricted web service access in mobile phones. 2014 Fifth *International Conference on the Applications of Digital Information and Web Technologies (ICADIWT)*, pp. 69-74.
- Rane, S. Wang, Y. Draper, S. C. Ishwar, P. (2013) Secure biometrics: Concepts, authentication architectures, and challenges. *Signal Processing Magazine, IEEE*, 30(5), 51-64.
- Rauti, S. Leppänen, V. (2012) Browser extension-based man-in-the-browser attacks against Ajax applications with countermeasures. *Proceedings of the 13th International Conference on Computer Systems and Technologies*, pp. 251-258.
- Reynolds, J. (2000) eCommerce: a critical review. *International Journal of Retail & Distribution Management*, 28(10), 417-444.
- Riley, R. D. Ali, N. M. Al-Senaidi, K. S. Al-Kuwari, A. L. (2011) Empowering users against sidejacking attacks. *ACM SIGCOMM Computer Communication Review*, 41(4), 435-436.
- Rosenfeld, L. Morville, P. (2006) *Information architecture for the world wide web*. 3rd ed. Sebastopol: O'Reilly Media.
- Ruiz, R.D.S. Amatte, F.P. Park, K.J.B. Winter, R. (2015) Overconfidence: Personal Behaviors Regarding Privacy that Allows the Leakage of Information in Private Browsing Mode. *International Journal of Cyber-Security and Digital Forensics*, 4(3), pp.404-417.
- Rumble, S. M. Ongaro, D. Stutsman, R. Rosenblum, M. Ousterhout, J. K. (2011) It's time for low latency. *Proceedings of the 13th USENIX conference on Hot topics in operating systems*, pp.11.
- Runceanu, A. Popescu, M. (2013) Innovations and Advances in Computer, Information, Systems Sciences, and Engineering. *Lecture Notes in Electrical Engineering*, 152, pp.839-847.
- Rutanen, K. Gómez-Herrero, G. Eriksson, S.L. Egiazarian, K. (2013) A general definition of the big-oh notation for algorithm analysis. *CoRR*. Available at: <http://kaba.hilvi.org/homepage/publications/big-oh/bigoh-slides.pdf>. Last Accessed: 20 July 2016

References

- Saha, A. Das, A. (2012) A detailed analysis of the issues and solutions for securing data in cloud. *Journal of Computer engineering (IOSRJCE)*, 4(5), pp.11-18.
- Saiedian, H. Broyle, D. (2011) Security Vulnerabilities in the Same-Origin Policy: Implications and Alternatives. *Computer Journal*, 44(9), pp.29 – 36.
- Sarris, S. (2013) *HTML5 Unleashed*. Indianapolis: Sams.
- Sevilla, M. Nassi, I. Ioannidou, K. Brandt, S. Maltzahn, C. (2014) SupMR: Circumventing Disk and Memory Bandwidth Bottlenecks for Scale-up MapReduce. *2014 IEEE International Parallel & Distributed Processing Symposium Workshops (IPDPSW)*, pp.1505-1514.
- Shar, L. K. Tan, H. B. K. (2013) *Defeating SQL injection*. *Computer*, (3), 69-77.
- Shar, L.K. Tan, H. B. K. (2012) Auditing the XSS defence features implemented in web application programs. *Software, IET*, 6(4), pp.377-390
- Sharma, T. N. Bhardwaj, P. Bhardwaj, M. (2012) Differences between HTML and HTML5. *International Journal Of Computational Engineering Research*, 2(5).
- Shashank, T. (2011) *Professional NoSQL*. Indianapolis: John Wiley & Sons.
- Shema, M. (2012). *Hacking web apps: detecting and preventing web application security problems*. 1st ed. Waltham: Syngress.
- Shi, C.G. (2013) Analysis of Markov Models. *Applied Mechanics and Materials*, 462, pp. 243-246.
- Shin, H. Kim, D. Hur, J. (2015) Secure pattern-based authentication against shoulder surfing attack in smart devices. *Seventh International Conference on Ubiquitous and Future Networks* (pp. 13-18).
- Shulman, A. (2006) *Top Ten Database Security Threats*. [online]. Available at: http://www.schell.com/Top_Ten_Database_Threats.pdf. [Accessed 16 February 2013]
- Singh, A. Chatterjee, K. (2015) A secure multi-tier authentication scheme in cloud computing environment. *2015 International Conference on Circuit, Power and Computing Technologies (ICCPCT)*, pp. 1-7.
- Smith, B. (2015) X-Origin Resources. In *Beginning JSON* (pp. 133-158). New York: Apress.
- Sunil, (2015) Trends and practices of consumers buying online and offline: An analysis of factors influencing consumer's buying. *International Journal of Commerce and Management*, 25(4), pp.442-455.
- Stalker, J. Gibbins, B. Meidl, P. Smith, J. Spooner, W. Hotz, H. R. Cox, A. V. (2004) The Ensembl Web site: mechanics of a genome browser. *Genome research*, 14(5), 951-955.

References

- Stallings, W. (2013) *Cryptography and Network Security: Principles and Practice*, 6 ed. New York: Prentice Hall.
- Stallings, W. Brown, L. (2008) *Computer Security: Principles and Practice*. New York: Prentice-Hall,
- Stamm, S. Sterne, B. Markham, G. (2010) Reining in the web with content security policy. *Proceedings of the 19th international conference on World wide web*, pp. 921-930).
- Stark, E. Hamburg, M. Boneh, B (2009) Symmetric cryptography in Javascript. , *ACSAC'09*, pp. 373-381.
- Statcounter.com, (2016) *StatCounter - Free Invisible Web Tracker, Hit Counter and Web Stats*. [online] Available at: <https://statcounter.com> [Accessed 24 February 2016].
- Statista (2016) *Statistics portal*. [online] Available at: <http://www.statista.com/statistics/203734/global-smartphone-penetration-per-capita-since-2005> [Accessed 10 January 2016].
- Stevens, L. Owen, R. J. (2014) The Truth About HTML5 Web Apps, Mobile, and What Comes Next. *In The Truth About HTML5*, pp. 153-164.
- Stinson, D. R. (2006) *Cryptography: theory and practice*. London: Chapman & Hall/CRC.
- Stockhammer, T. (2011) Dynamic adaptive streaming over HTTP--: standards and design principles. *Proceedings of the second annual ACM conference on Multimedia systems*, pp.133-144.
- Stonebraker, M. (2010) SQL Databases v. NoSQL Databases. *Communications of the ACM*, 53(4), pp.10-11.
- Strozzi, C. (1998) *NoSQL A Relational Database Management System*. [online] Available at: http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql/Home%20Page [Accessed 17 March 2012].
- Stuttard, D. Pinto, M. (2011) *Web Application Hacker's Handbook: Finding and Exploiting Security Flaws*. 2nd ed. Indianapolis: Wiley Publishing.
- Sumathi, S. Esakkirajan, S. (2007) *Fundamentals of Relational Database Management Systems*. Springer Berlin Heidelberg.
- Summers, S. Schwarzenegger, C. Ege, G. Young, F. (2014) *The emergence of EU criminal law: cyber crime and the regulation of the information society*. Oxford: Bloomsbury Publishing.
- Szefer, J. Jamkhedkar, P. Chen, Y. Y. Lee, R. B. (2012) Physical attack protection with human-secure virtualization in data centers. 2012 IEEE/IFIP 42nd International Conference on Dependable Systems and Networks Workshops (DSN-W), pp. 1-6.

References

- 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.
- Taivalseeri, A. Mikkonen, T. (2011) The Web as an Application Platform: The Saga Continues. *Software Engineering and Advanced Applications (SEAA)*, 37th EUROMICRO Conference, pp. 170-174.
- Taivan, C. José, R. Silva, B. (2014) Understanding the use of web technologies for applications in open display networks. *IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, pp. 500-505.
- Takesue, M. (2008) A Protection Scheme against the Attacks Deployed by Hiding the Violation of the Same Origin Policy. *Second International Conference on Emerging Security Information, Systems and Technologies, SECURWARE '08*, pp.133-138.
- Tamhankar, A.M. Ram, S. (1998) Database fragmentation and allocation: an integrated methodology and case study, *Systems, Man and Cybernetics, Part A: IEEE Transactions on Systems and Humans*, 28(3), pp.288-305.
- Tang, Y. Lee, P.P.C. Lui, J.C.S. Perlman, R. (2012) Secure Overlay Cloud Storage with Access Control and Assured Deletion. *IEEE Transactions on Dependable and Secure Computing*, 9(6), pp.903-916.
- Tauro, C. J. Aravindh, S. Shreeharsha, A. B. (2012) Comparative study of the new generation, agile, scalable, high performance NOSQL databases. *International Journal of Computer Applications*, 48(20), 1-4.
- Tendulkar, D. M. Phalak, C. (2011) Proactive performance testing using SQL performance assurance services (SQL-PASS). *Proceedings of the International Conference & Workshop on Emerging Trends in Technology (ICWET '11)*, pp.541-547.
- Thakur, J. Kumar, N. (2011) DES, AES and Blowfish: Symmetric key cryptography algorithms simulation based performance analysis. *International journal of emerging technology and advanced engineering*, 1(2), 6-12.
- Tian, Y. Liu, Y. C. Bhosale, A. Huang, L. S. Tague, P. Jackson, C. (2014) All your screens are belong to us: attacks exploiting the html5 screen sharing api. *IEEE Symposium on In Security and Privacy (SP)*, pp. 34-48.
- Tirumala, S.S. Sathu, H. Naidu, V. (2015) Analysis and prevention of account hijacking based incidents in cloud environment. In *2015 International Conference on Information Technology (ICIT)* (pp. 124-129).
- Totok, A. Karamcheti, V. (2011) Exploiting Service Usage Information for Optimizing Server Resource Management. *ACM Transactions on Internet Technology*, 11(1).

References

- Tresadern, P. Cootes, T. F. Poh, N. Matejka, P. Hadid, A. Levy, C. Marcel, S. (2013) Mobile biometrics: Combined face and voice verification for a mobile platform. *IEEE pervasive computing*, (1), 79-87.
- Tudorica, B.G. Bucur, C. (2011) A comparison between several NoSQL databases with comments and notes. *10th Roedunet International Conference (RoEduNet)*, pp.1-5.
- Uehara, S. Mizuno, O. Kikuno, T. (2001) An implementation of electronic shopping cart on the Web system using component-object technology. *Proceedings. Sixth International Workshop on Object-Oriented Real-Time Dependable Systems*, pp.77-84.
- Ullman, L. (2012) *PHP and MySQL for dynamic web sites*. 4th edn. Berkeley: Peachpit Press.
- Van der Lans, R. F. (2006) *Introduction to SQL: mastering the relational database language*. 4 ed. Harlow: Addison-Wesley.
- Van der Mei, R. D. Hariharan, R. Reeser, P. K (2001) Web Server Performance Modeling. *Telecommunication Systems*, 16(3-4), pp.361-378.
- Van der Veen, J.S. Van der Waaij, B. Meijer, R.J. (2012) Sensor Data Storage Performance: SQL or NoSQL. *2012 IEEE 5th International Conference on Physical or Virtual Cloud Computing (CLOUD)*, pp.431-438.
- Vargo, S. L. Lusch, R. F. (2011) It's all B2B... and beyond: Toward a systems perspective of the market. *Industrial Marketing Management*, 40(2), 181-187.
- Vilaplana, J. Solsona, F. Teixido, I. Mateo, J. Abella, F. Rius, J. (2014) A queuing theory model for cloud computing. *Journal of Supercomputing*, 69(1), pp. 492-507.
- Vishwakarma, S. Samant, P. K. Sharma, A. (2015) Attacks in a PKI-Based Architecture for M-commerce. *2015 IEEE International Conference on Computational Intelligence & Communication Technology (CICT)*, pp. 52-56.
- Vogt, P. Nentwich, F. Jovanovic, N. Kirda, E. Kruegel, C. Vigna, G. (2007) Cross Site Scripting Prevention with Dynamic Data Tainting and Static Analysis. In NDSS, pp.12.
- W3C (2010) *Web SQL Database*. [online] Available at: <https://www.w3.org/TR/webstorage> [Accessed 10 December 2015].
- W3C (2013) File API. [online] Available at: <http://www.w3.org/TR/file-upload/> [Accessed 10 April 2015].
- W3C (2014) *HTML5*. [online] Available at: <http://www.w3.org/TR/html5/> [Accessed 20 November 2015]
- W3C (2015) *Indexed Database API*. [online] Available at: <http://www.w3.org/TR/IndexedDB/> [Accessed 10 April 2015].

References

- W3C (2015) *Web SQL Database*. [online] Available at: <https://www.w3.org/TR/webstorage/> [Accessed 10 January 2016].
- Walden, D. (2014) The Arpanet IMP Program: Retrospective and Resurrection. *Annals of the History of Computing, IEEE*, 36(2), 28-39.
- Walker, J. D. Chapra, S. C. (2014) A client-side web application for interactive environmental simulation modeling. *Environmental Modelling & Software*, 55(0), pp.49-60.
- Walker, S. M. Shan, J. (2015) Using the DIMMACSS-PSG Intelligent Robotic Middleware to Control Real-World and Simulated Multi-Agent Systems. *AIAA Modeling and Simulation Technologies Conference*, pp.404
- Wang, D. Zhang, Z. H. (2010) Research and design of E-commerce component. *Computer Engineering and Design*, 31(2), 374-377.
- Wang, Y. Li, Z. Guo, T. (2011) Program Slicing Stored XSS Bugs in Web Application. *Fifth International Symposium on Theoretical Aspects*.
- Wang, J. P. Li, X. M. Jiao, C. L (2012) The Network Traffic Management Based on Queuing Theory. *Applied Mechanics and Materials*, 121-126, pp.191-194.
- Wang, N. Cheng, X. Gou, Q. (2015) Four Express Service Cooperation Modes for B2C E-Commerce: Models and Analysis. *International Journal of Knowledge-Based Organizations (IJKBO)*, 5(4), pp.1-18.
- Wassermann, G. Su, Z. (2008) Static detection of cross-site scripting vulnerabilities. *30th International Conference on Software Engineering, ICSE'08*. pp. 171-180.
- Weinberg, Z. Chen, E.Y. Jayaraman, P.R. Jackson, C. (2011) I Still Know What You Visited Last Summer: Leaking Browsing History via User Interaction and Side Channel Attacks. *IEEE Symposium on Security and Privacy (SP)*, pp.147-161.
- Weissbacher, M. Robertson, W. Kirda, E. Kruegel, C. and Vigna, G. (2015) ZigZag: automatically hardening web applications against client-side validation vulnerabilities. In *24th USENIX Security Symposium (USENIX Security 15)* (pp. 737-752).
- Wellin, P. (2013) *Programming with Mathematica*. Cambridge: Cambridge.
- Wernke, M. Skvortsov, P. Dürr, F. Rothermel, K. (2014) A classification of location privacy attacks and approaches. *Personal and Ubiquitous Computing*, 18(1), 163-175.
- West, W. Pulimood, S. M. (2012) Analysis of privacy and security in HTML5 web storage. *Journal of Computing Sciences in Colleges*, 27(3), 80-87.

References

- WhiteHat (2014) *2014 Website Security Statistics Report*. [online] Available at: <http://info.whitehatsec.com/rs/whitehatsecurity/images/statsreport2014-20140410.pdf> [Accessed 22 April 2015].
- Whitt, W. (2000) An overview of Brownian and non-Brownian FCLTs for the single-server queue. *Queueing Systems*, 36, pp.39-70.
- Willard, W. (2013) *HTML: a beginner's guide*. 5th ed. San Francisco : McGraw-Hill.
- Willis. Ch. (2009) *Preparing for the Cross site request forgery defense*. [online] Available at: <http://www.blackhat.com/presentations/bh-dc-08/Willis/Whitepaper/bh-dc-08-willis-WP.pdf> [Accessed 7 February 2013]
- Windows (2011) *IDBDatabase*. [online] Available at: <http://msdn.microsoft.com/en-us/library/windows/apps/hh441231.aspx>. [Accessed 21 September 2013]
- Wisniewski, J. (2011) *HTML5. Online journal*, 35(6), pp.53-56.
- Xiaojing, L. Liwei, Z. Weiqing, W. (2012) The mechanism analysis of the impact of ecommerce to the changing of economic growth mode. *2012 IEEE Symposium on Robotics and Applications (ISRA)*, pp. 698-700.
- XRY (2015) *XRY Mobile Forensic Tool*. [online] Available at: <https://www.msab.com/products/xry>. [Accessed 20 February 2016].
- Xu, J. Chang, E. C. Zhou, J. (2013) Weak leakage-resilient client-side deduplication of encrypted data in cloud storage. *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, pp. 195-206.
- Xu, C. Xia, F. Sharaf, M.A. Zhou, M. Zhou, A. (2014) AQUAS: A quality-aware scheduler for NoSQL data stores. *2014 IEEE 30th International Conference on Data Engineering (ICDE)*, pp.1210-1213
- Xu, H. Zhou, Y. Lyu, M. R. (2014) Towards continuous and passive authentication via touch biometrics: An experimental study on smartphones. *In Symposium On Usable Privacy and Security, SOUPS (Vol. 14, pp. 187-198)*.
- Xu, P. Chen, L. Santhanam, R. (2015) Will video be the next generation of e-commerce product reviews? Presentation format and the role of product type. *Decision Support Systems*, 73, 85-96.
- Xue, L. Kumar, S. Cui, C. Kondikoppa, P. Chiu, C-H. Park, S-J. (2013) AFCD: An Approximated-Fair and Controlled-Delay Queuing for High Speed Networks. *22nd International Conference on Computer Communications and Networks (ICCCN)*, pp.1-7.
- Yadav, S. K. Singh, G. Yadav, D. S. (2013) Analysis of Database Replication Algorithm in Local and Global Networks. *International Journal of Computer Applications (0975 – 8887)*, 84(6).

References

- Yampolskiy, R. V. Ali, N. D'Souza, D. Mohamed, A. A. (2014) Behavioral Biometrics: Categorization and Review. *International Journal of Natural Computing Research (IJNCR)*, 4(3), 85-118.
- Yang, B. Chu, H. Li, G. Petrovic, S. Busch, C. (2014) Cloud Password Manager Using Privacy-Preserved Biometrics. *2014 IEEE International Conference on Cloud Engineering (IC2E)*, pp. 505-509.
- You, W. Qian, K. Lo, D.C.T. Bhattacharya, P. Chen, W. Rogers, T. Chern, J.C. Yao, J. (2015) Promoting mobile computing and security learning using mobile devices. In *Integrated STEM Education Conference (ISEC)*, (pp. 205-209).
- Yubin, G. Liankuan, Z. Fengren, L. Ximing, L. (2013) A Solution for Privacy-Preserving Data Manipulation and Query on NoSQL Database. *Journal of Computers*, 8(6), pp. 1427.
- Yusof, I. Pathan, A. S. K. (2014) Preventing persistent Cross-Site Scripting (XSS) attack by applying pattern filtering approach. *The 5th International Conference on Information and Communication Technology for The Muslim World (ICT4M)*, pp.1-6.
- Zachary, P. Poe, S. Vrbsky, S.V. (2013) Comparing NoSQL MongoDB to an SQL DB. *Proceedings of the 51st ACM Southeast Conference (ACMSE '13)*. pp.5-6.
- Zakas, C. N. (2010) *Cross-domain Ajax with Cross-Origin Resource Sharing*. [online]. Available at: <http://www.nczonline.net/blog/2010/05/25/cross-domain-ajax-with-cross-origin-resource-sharing> [Accessed 10 April 2014].
- 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
- Zhao, Y. Lin, H. (2014) WEB data mining applications in e-commerce. *2014 9th International Conference on Computer Science & Education (ICCSE)*, pp. 557-559.
- Zhou, T. Wei, Y. (2013) Database replication technology having high consistency requirements. *2013 International Conference on Information Science and Technology (ICIST)*, pp.793-797.

Appendix A: List of Publications

Kimak, S. Ellman, J. Laing, C. (2012) An Investigation into Possible Attacks on HTML5 IndexedDB and their Prevention. In *The 13th Annual Post-Graduate Symposium on The Convergence of Telecommunications, Networking and Broadcasting (PGNet 2012)*, Liverpool, UK. (Appendix D)

Kimak, S. Ellman, J. (2012) CSRF attacks and their prevention. 6th International Conference on Software, Knowledge, Information Management and Applications (*Skima 2012*), Chengdu, China. (Appendix C)

Kimak, S. Ellman, J. (2013) Performance Testing and Comparison of Client-side Databases Versus Server-side. *Northumbria University*. In *The 14th Annual Post-Graduate Symposium on The Convergence of Telecommunications, Networking and Broadcasting (PGNet 2013)*, Liverpool, UK. (Appendix E)

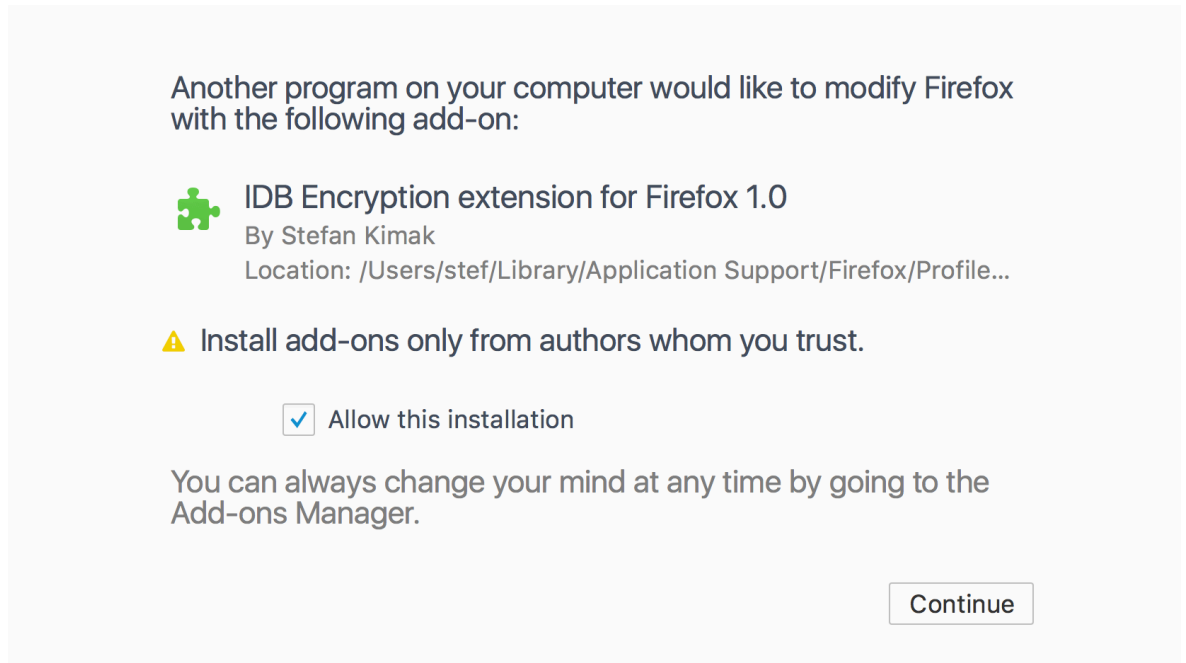
Kimak, S. Ellman, J. Laing, C. (2014) Some Potential Issues with the Security of HTML5 IndexedDB. 9th IET System Safety and Cyber Security, Manchester, UK (Appendix F)

Kimak, S. Ellman, J. (2015) The role of HTML5 IndexedDB, the past, present and future. The 10th International Conference for Internet Technology and Secured Transactions (ICITST-2015), London, UK (Appendix G)

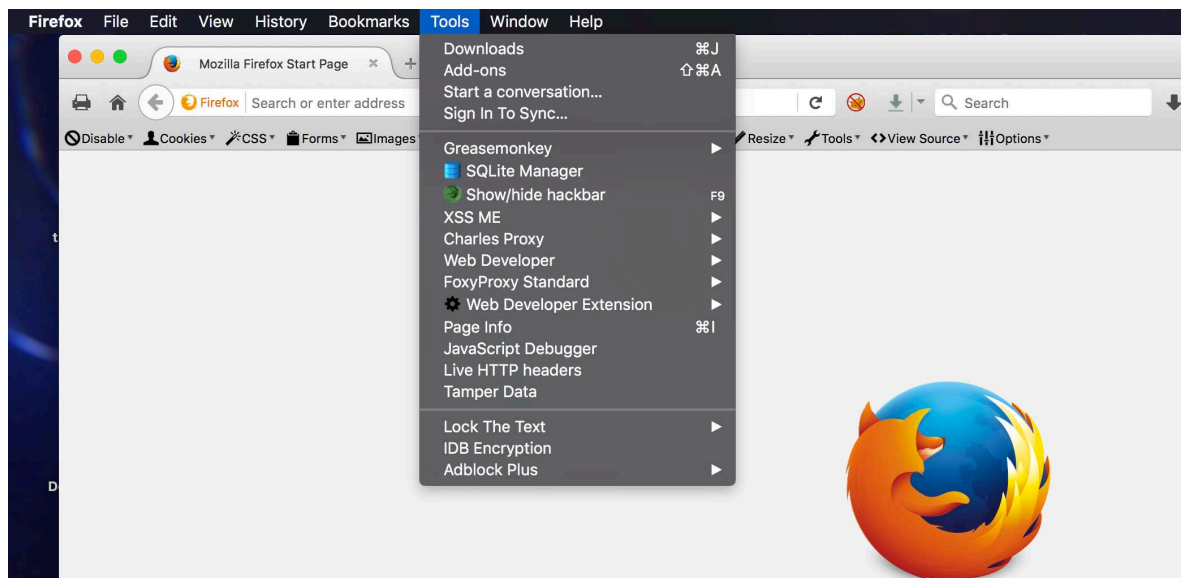
Kimak, S. Ellman, J. (2015) HTML5 IndexedDB encryption: Prevention against potential attacks. *International Journal of Intelligent Computing Research (IJICR)*, Volume 6, Issue 4, December 2015 (Appendix H)

Appendix B: Firefox extension of encryption library

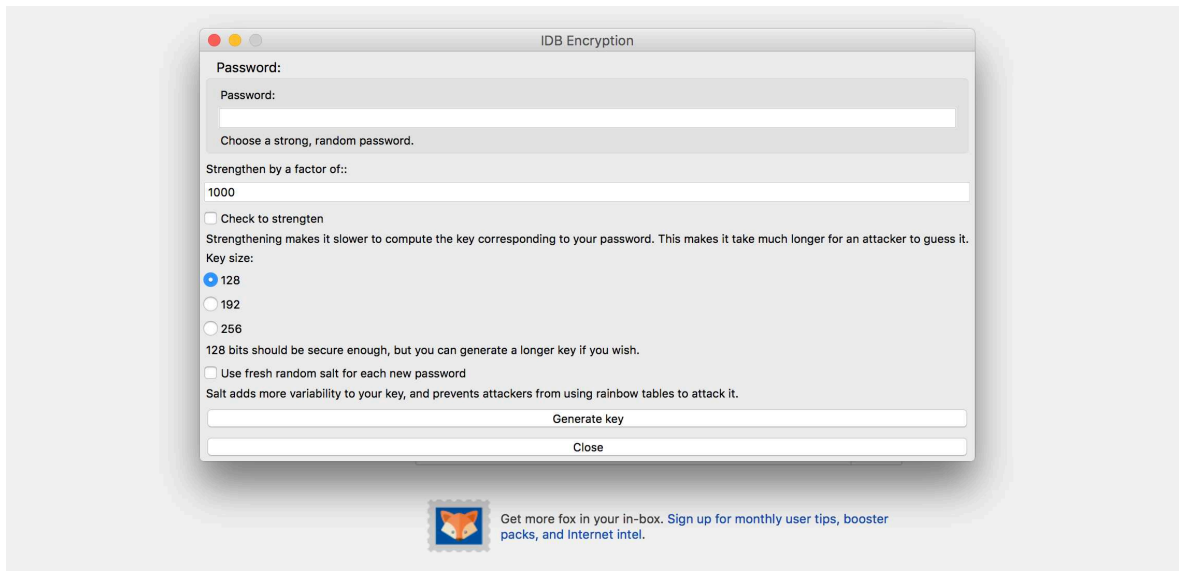
The Firefox extension needs to be first compressed to xpi format and then copied over to extension fielder. When the browser is restarted the user is prompted to install the extension, as shown in the image below. Firefox changed it regulations, so only validated extensions from the Firefox store can be installed. There is a workaround to disable the security measures for installing unknown extensions.



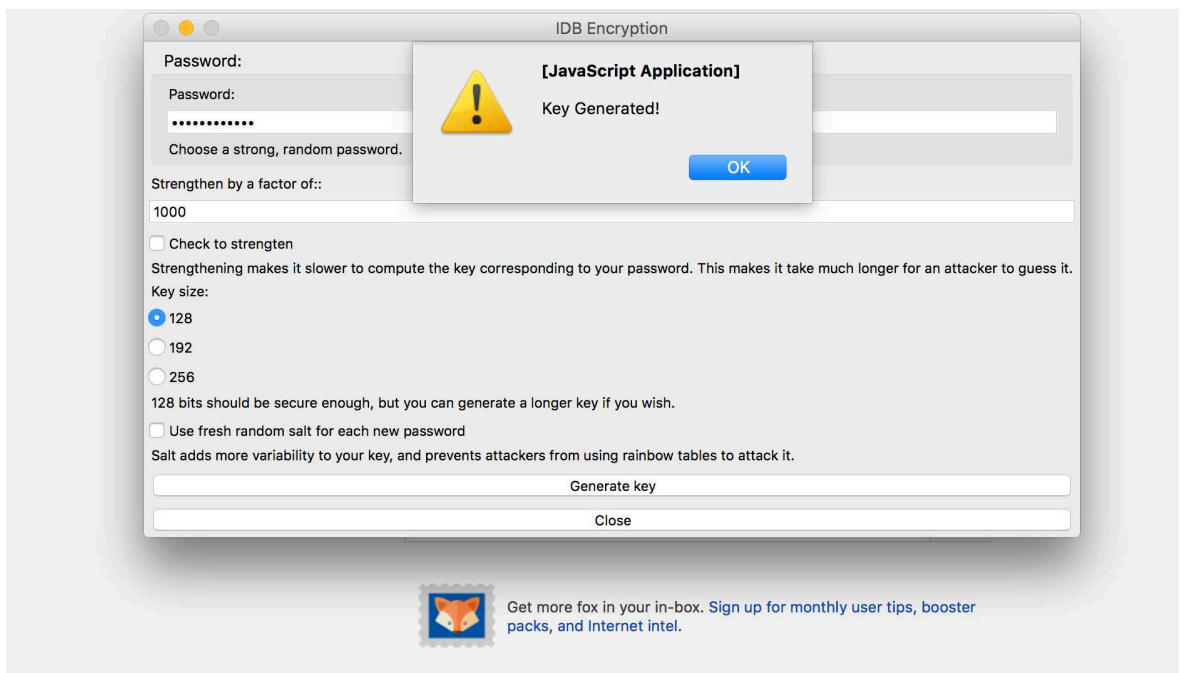
Above image is a security warning when installing, as it is not listed in Android store.



The IDB Encryption tab can be found in tools menu in Firefox browser after installing the package.



Above image is displaying different options, which can be chosen to generate new key when encrypting IndexedDB stored on local disk.



Above image displays successful generation of key, which means that the files on local disk are encrypted.

Below are helper functions for EAS encryptions, which are included in IDB Encryption Firefox extension.

```

ar t = void 0, u=!1;
var sjcl = {
  cipher: {}},

```

```

hash: {},
keyexchange: {},
mode: {},
misc: {},
codec: {},
exception: {
  corrupt: function(a) {
    this.toString = function() {
      return "CORRUPT: " + this.message
    };
    this.message = a
  },
  invalid: function(a) {
    this.toString = function() {
      return "INVALID: " + this.message
    };
    this.message = a
  },
  bug: function(a) {
    this.toString = function() {
      return "BUG: " + this.message
    };
    this.message = a
  },
  notReady: function(a) {
    this.toString = function() {
      return "NOT READY: " + this.message
    };
    this.message = a
  }
}
};

"undefined" != typeof module && module.exports && (module.exports = sjcl);
sjcl.cipher.aes = function(a) {
  this.j[0][0][0] || this.D();
  var b, c, d, e, f = this.j[0][4], g = this.j[1];
  b = a.length;
  var h = 1;
  4 !== b && (6 !== b && 8 !== b) && q(new sjcl.exception.invalid("invalid aes key size"));

```

```

this.a = [d = a.slice(0), e = []];
for (a = b; a < 4 * b + 28; a++) {
  c = d[a - 1];
  if (0 === a%b || 8 === b && 4 === a%b)
    c = f[c>>>24]<<24^f[c>>16 & 255]<<16^f[c>>8 & 255]<<8^f[c & 255], 0 === a%b
    && (c = c<<8^c>>>24^h<<24, h = h<<1^283 * (h>>7));
  d[a] = d[a - b]^c
}
for (b = 0; a; b++, a--)
  c = d[b & 3 ? a : a - 4], e[b] = 4 >= a || 4 > b ? c : g[0][f[c>>>24]]^g[1][f[c>>16 &
  255]]^g[2][f[c>>8 & 255]]^g[3][f[c &
  255]]
};
sjcl.cipher.aes.prototype = {
  encrypt: function(a) {
    return y(this, a, 0)
  },
  decrypt: function(a) {
    return y(this, a, 1)
  },
  j: [[[], [], [], [], []], [[], [], [], [], []]],
  D: function() {
    var a = this.j[0], b = this.j[1], c = a[4], d = b[4], e, f, g, h = [], l = [], k, n, m, p;
    for (e = 0; 0x100 > e; e++)
      l[(h[e] = e<<1^283 * (e>>7))^e] = e;
    for (f = g = 0; !c[f]; f^=k || 1, g = l[g || 1]) {
      m = g^g<<1^g<<2^g<<3^g<<4;
      m = m>>8^m & 255^99;
      c[f] = m;
      d[m] = f;
      n = h[e = h[k = h[f]]];
      p = 0x1010101 * n^0x10001 * e^0x101 * k^0x1010100 * f;
      n = 0x101 * h[m]^0x1010100 * m;
      for (e = 0; 4 > e; e++)
        a[e][f] = n = n<<24^n>>>8, b[e][m] = p = p<<24^p>>>8
    }
    for (e =
0; 5 > e; e++)
      a[e] = a[e].slice(0), b[e] = b[e].slice(0)

```

```

    }
};

```

Sent from the page to the add-on, when the user clicks an element in the page.

```

var pageModScript = "window.addEventListener('click',
function(event) {" +
    "    self.port.emit('click',
event.target.toString());" +
    "    event.stopPropagation();" +
    "    event.preventDefault();" +
    "}, false);" +
    "self.port.on('warning', function(message) {"
+
    "    window.alert(message);" +
    "});"

var pageMod = require('sdk/page-mod').PageMod({
  include: ['*'],
  contentScript: pageModScript,
  onAttach: function(worker) {
    worker.port.on('click', function(html) {
      worker.port.emit('warning', 'Do not click this again');
    });
  }
});

```

Helper functions to include an array of strings in the payload.

```

var pageModScript = "self.port.emit('loaded'," +
    " [" +
    "  document.location.toString()," +
    "  document.title" +
    " ]" +
    ");"

var pageMod = require('page-mod').PageMod({
  include: ['*'],
  contentScript: pageModScript,
  onAttach: function(worker) {
    worker.port.on('loaded', function(pageInfo) {
      console.log(pageInfo[0]);
      console.log(pageInfo[1]);
    });
  }
});

```

```

var element = document.createElement("MyExtensionDataElement");
element.setAttribute("application_state", "ready");
document.documentElement.appendChild(element);

```

```
//create a custom event and dispatch it
// using the custom element as its target

var ev = document.createEvent("Events");
ev.initEvent("MyExtensionEvent", true, false);
element.dispatchEvent(ev);
```

The Firefox extension of IDB Encryption can be downloaded from
<https://github.com/stefankim/IDB-Encryption>

Appending C: An Experimental Analysis and Possible Solution for the Cross Site Request Forgery Attack

6th International Conference on Software, Knowledge, Information Management and Applications (*Skima 2012*), Chengdu, China.

Appending C: An Experimental Analysis and Possible Solution for the Cross Site Request Forgery Attack

Stefan Kimak, Jeremy Ellman
CEIS, Northumbria University
Newcastle upon Tyne, UK
Stefan.kimak@Northumbria.ac.uk

Abstract— A Cross-Site Request Forgery (CSRF/XSRF) is web application vulnerability where malicious code placed on a web site can make requests to another web site using the victim’s credentials. This kind of attack appears to come from the victim who still logged in, but is done without his knowledge. The attack can be constructed in many ways, as send email to other user or change password without user acknowledgment, using the permission the user has given. This paper discusses two CSRF attacks and a defense against them. The most commonly known defenses are discussed and explained and implemented into experimental scenarios. These defenses are implemented as a tool in PHP and are tested. The results are evaluated and compared with reported work. From the tests performed a defense using randomly generated token is recommended.

Keywords-component web security; CSRF;XSRF; cross site request forgery

I. INTRODUCTION

The Open Web Application Security Project (OWASP) [1] has identified the Cross-Site Request Forgery (CSRF) as one of the top 10 web security vulnerabilities. The attacker creates a web site that appears innocent, but when a victim loads the page, his browser inadvertently sends a request to a vulnerable web application that performs an action useful to the attacker [2]. A link, image, iframes, JavaScript or other content can execute the malicious code.

A CSRF attack happens when a victim’s browser thinks that the request is coming from him, rather than from the attacker’s code on the malicious site. This is possible because the browser sends the victim’s cookie with the request, and the application assumes that request came from that particular user [3]. Cross-site request forgeries are often HTTP GET requests collected and sent through the use of some html feature that loads automatically (like an image, iframes or script tag). The user typically thinks that he is performing a different task but his web page requests have side effects. These exploit the users own browser to send the users security credentials to the attackers target site.

CSRF are possible on any site that allows images or links to be posted even if the actions use the HTTP POST method [4].

This paper proceeds as follows. Firstly we describe possible defenses against CSRF attacks, which cover existing work in this area. Then we describe the experimental scenario, and possible simulated attacks this supports. Next we discuss the design and implementation of a tool that can protect against CSRF attacks. We then conclude with discussions and future work.

II. DEFENCES

In this section we review several possible defenses against CSRF attacks. Such are attacks are difficult to defend against, as they exploit the automatic loading features of images, iframes, etc that execute automatically when a web page is loaded. The following defenses are all executed at the server. That is, they try to ensure that the credentials used in a secure transaction are legitimately those of the authorized user. We describe in turn the Secret Validation Token, Referrer Header, and Challenge-Response.

A. Secret Validation Token

The session token is a randomly generated unique number that ensures a unique relationship between a web application and the user’s browser. In a secure application session tokens are included as variables in the http header. When an http request (i.e. web page request) is received the application verifies that the correct token is included. The attacker will not be able to perform an attack without knowing the session token [5].

B. The Referrer Header

The referrer header defense consists of checking the HTTP header referrer URL to see if an action request comes from the same host that initiated the request [6]. This solution is not reliable since for privacy reasons the http referrer is not always sent. The user can also switch off sending the referrer in the browser. If a request ignores the referrer header then this request will be ignored. The referrer might be sent from another domain, but the server needs to have in place a cross-domain policy, which lists the authorized and secure domain [7].

This defense is also weak as is easy to spoof or trick, so that some web pages can have fake referrer header. Some

Appending C: An Experimental Analysis and Possible Solution for the Cross Site Request Forgery Attack

web pages do not use referrer checks, so this method is still not the most suitable.

C. Challenge-Response

CAPTCHA as the distorted graphic letters used to defend many web sites. The user must enter the correct letters to confirm the page on which they entered their password is the same page being sent to the server. This method prevents against robotic submissions of forms.

We now move on to describe experimental scenario.

III. EXPERIMENTAL SCENARIOS AND ATTACKS

In this section we describe the experimental scenario and possible variation on the CSRF attack. These exploit the login form, a change password request, and related attacks. These attacks follow the general scenario fig (1) as described in [6].

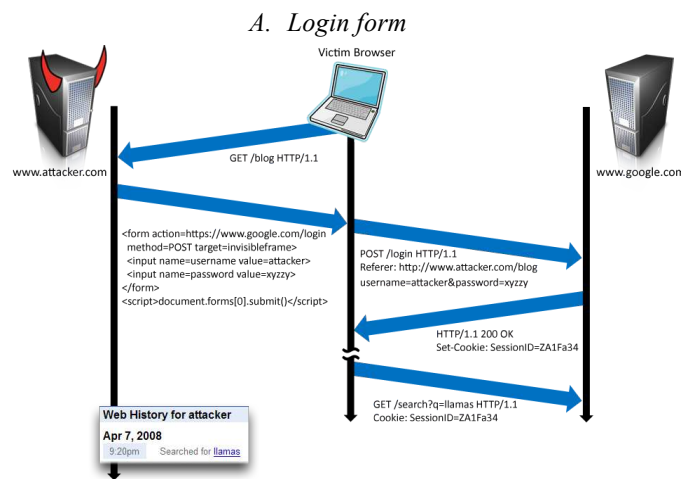


Fig. 12. Login CSRF, [6]

In the Login CSRF attack an attacker uses CSRF to log a user in to a site using the user credentials. The attacker then waits for them to submit sensitive information. The attacker then forges request to an honest site. Attacker logs in with his own credentials, establishing a user session of the attacker.

The following user requests to the honest site are done within the user session of the attacker.

Scenario:

- The victim visits a web page which requires user login
- The page will contain attackers attack code, which is a automated submission form for login
- The victim will be login to application, but with attackers credentials
- The victim may add a credit card to his account, because the application contains purchasing goods.
- The details will be added to attackers account, and

the victim might not even realize it.

Here, malicious code performs an automated submission of a web form. The form contains the attacker's username and password, which will cause that a victim will login to an application inadvertently using the attacker's credentials. Thus, all of the victim's actions will be performed using the attacker account.

An attack is possible even if the application is using POST method, as JavaScript code can be used here [9].

B. Change password

- Attacker finds a site1 from where he wants to steal victims or administrator account.
- The site1 has a function, which can change a password (most of web sites has this function).
- Now the attacker needs to trick the victim or administrator to change the password.
- The attacker can trick administrator as a fake message on bulletin forum, the image loads automatically and the code changes the administrator password with his credentials.
- The attacker sends an email to administrator with a fake link, which will point him to fake website and the action will be performed in the background without administrator knowledge.
- The attacker places an image with an attack code into site2.
- This site will execute the command, and unless the site had specific CSRF protections in place, the user's password would end up being changed.
- This attack succeed only when the victim or administrator are still logged on site1, where the session or cookies are still active.
- The attacker can now log into site1 with the victims or administrator's username, which is known and the changed password "1234".

C. Attacks

1) Attack number 1

This attack takes all the user's cookies, which enters a site, grab then and insert into txt file or database, it depend of the php file structure.

The user needs to click on the particular link to perform an attack.

2) Attack number 2

When a user load a site the attacking image loads a URL address, which will send an email on behalf and without knowledge of the user.

IV. DESIGN AND IMPLEMENTATION

PHPProxy is a web-based proxy written in PHP by the eProxies.info Team. It is used to hide ones identity on the Internet, protect privacy, and to speed up Internet connections.

Appending C: An Experimental Analysis and Possible Solution for the Cross Site Request Forgery Attack

PHPProxy is a web HTTP proxy designed to bypass restrictions such as global IP address permissions. The interface allows all the features to be switched on or off. All HTTP requests and responses are filtered by PHPProxy, which sits down between the user browser and server.

A. Implementation of the extension for the PHPMyProxy

The extension will include interception from the server, which will include the link (source) of the image or iframes, header information, referrer, cookies and active sessions. The implementation of the extension will be divided into few subsections, which will include functions to parse the html, interception of the header and form submitted, and checks for analysis of the code.

B. Parse the HTML

A function to parse the HTML page was implemented using the simple html DOM library from sourceforge [10]. The function parses the html source code into a string. The function creates a DOM object from the html page, which will be processed.

The code allows looping through the DOM objects, which will find all the images [11]. The images might be printed or passed to other function where their can be checked.

The code will output a source of the image, and will loop till all sources URL of images are shown. The images source will be displayed parsed or non-parsed, depended from the web page. The parsed source will include URL as `http://example.com/image.jpg`, and non-parsed source will list only path to image as `/example/images/bg/image.jpg`.

C. Interception

The code below gets HTTP request and print on screen all header information.

The code will output a response header, with hostname, referrer and cookies available for the current session.

The main information about the response header from the server will be shown, as it will be a necessary part when testing the extension of defenses. The response header cannot be modified, but the cookies for each session will be stored into text file for testing purposes.

D. Checks for analysis

The codes in table I are a part of the implemented library, which checks the URL for possible attacks and compares them with already known attacks. The URL is broken into parts and each part must be checked. The code checks for example for the extension of the URL, so if it is a jpg, png, or gif image. It also checks if the image is stored on the same domain, and informs the user if any possible attack is

found. The user can decide if wants to show the image or not, but by default the images will be discarded and not shown. Basically the code takes the URL strings, break them down into their domain and compare them. The steps involved are described in Table I.

CODES FOR PARSING HTML PAGE

Number	Parsing HTML code	
	Codes	Description
1.	<code>if(isset(\$_POST["url"])){ \$url = \$_POST["url"];</code>	Get the url address from the form entered in the main form
2.	<code>include('simple_html_dom. php');</code>	Create DOM from URL or file, which will include a Dom library
3.	<code>\$html=@file_get_html(\$url; \$html2 = "\$url";</code>	Read an html file, which will be the url entered.
4.	<code>"foreach(\$html->find('img') as \$element) {" [10]</code>	Find all the images tags and put then in element variable

The codes in table II checks if the image has the same host. If so then continue to display the image. If not, check the extension of the file and if it is not an image then do not display it.

CODES FOR THE ANALYSIS OF THE IMAGE SOURCE

Number	Analysis of the image source	
	Codes	Description

Appending C: An Experimental Analysis and Possible Solution for the Cross Site Request Forgery Attack

1.	<code>\$test_domains = \$element->src; \$test_domains = explode(",",\$test_domains);</code>	Get the source of each image stored in the variable element
2.	<code>\$looking_for=parse_url(\$html2, PHP_URL_HOST); foreach (\$test_domains as \$url) {</code>	Parse url and get the host of the particular url address
3.	<code>\$parsed = parse_url(\$url); if (array_key_exists('host', \$parsed)) { \$host = explode('.', \$parsed['host']); \$found = implode('.', array_slice(\$host, -2, 2));</code>	Parse the url and get the host
4.	<code>\$filename = basename(\$url); \$ext = substr(strrchr(\$filename, '.'), 1);</code>	Get the extension of the image file
5.	<code>if (preg_match("/\$looking_for/", "\$url", \$matches)) { echo "Match was found
"; echo \$matches[0]; }</code>	Check the whole host of image for matching word against the host from the url
6.	<code>if(\$ext == 'jpg' \$ext == 'png' \$ext == 'gif') {</code>	Checks if the extension of the image is an jpg, png or gif format
7.	<code>if(\$looking_for == \$found) { echo \$parsed['host'] . ' is a match with ' . \$looking_for . '
;</code>	<code>looking_for == \$found</code> where found is the entered url host and the looking for is the host of the image
8.	<code>else { echo "Not the same
";} else { echo "Not an image
";} else { echo "No parsed url
";</code>	Else statements, if the domains are not the same, images don't have an extension or the url can not be parsed

E. Checks for analysis Defences miniform

The definition miniform came from the existing PHPProxy. This means that it is a kind of form where the options can be checked, which means that will become active. The miniform is available on every page on default, for the purpose of user awareness.

Miniform with options to apply specific defense methods and functions will be included as an extensive div to existing miniform. The form will be hidden, but if the user clicks the show button the form will appear. From there the user can choose a specific method, which will be applied to proxy.

The defense extension form will be available on each page, and the user can click each defense, which will be straight applied to each shown page.

When the user request a page, page that shows some information, which might be sensitive, and after user opens new tab or windows, the defense can be applied to page, which the user wish to protect. So if the user opens new tab and the page requested include an attack file in form an image or iframes, the page will be analyzed for safe images or iframes and after display only safe elements. If the user wish to protect or prevent the page with sensitive data can apply defenses 1-3, which will be applied by refreshing the page. The best-known protection defense will be set by default, and will be evaluated.

F. DEFENCES MINIFORM

The tables III, IV and V show the defenses included in the miniform, with short description and a code example. The code examples are part of implementation of the extension for the existing PHP proxy.

SECRET VALIDATION TOKEN (TOKEN)

Defenses			
Defense name	Defense description	Code example	Generate random token
Secret Validation Token (Token)	Check against the session value (rather than cookie value) Rewrites the <form> and add CSRF token to them. This will be after submitting the form checked if the token is the same or not.	<code>\$input="<input type='hidden' name='\$name' value=\"\$tokens\">"; \$form=preg_replace('#<form[^\>]*method\s*=\s*["\']post["\'][^\>]*>#i', '\$1', \$input, \$form);</code>	<code>function generateFormToken(\$formName) { \$token = md5(uniqid(microtime(), true)); \$_SESSION[\$formName.' token'] = \$token; return \$token; }</code>

The table III shows a defense using tokens. The token will be applied to every form and the token value will be random md5 number. As this is a random number, it is hard to guess it, so the attack might be less efficient. When the user requests a page, the proxy will append a token to each form. This is possible, because the proxy is parsing the entire page and will look only for the forms tags. Every form tag will be secured with this random token and after the user submits the form the function of valid token will check if the token is valid or not. A token will be then assigned to every form, so that every form will have different token.


If the token is still valid, then the form will be submitted. Otherwise the page will send an error message. The whole operation will be done on the server side, so no user interaction is needed. In that case an attacker will need to compromise the server itself to configure the code. The token is submitted encrypted will the form value.

Appending C: An Experimental Analysis and Possible Solution for the Cross Site Request Forgery Attack

THE REFERRER HEADER

Defenses		
Defense name	Defense description	Code example
The Referrer Header	Authentication data in the same HTTP Request checks the referrer header from submitted page against all other pages opened in PHPProxy, compare them and submit request only if the header match.	<pre>"ereg(\$ _SERVER[" HTTP_HOST"], str_replace("www.", "", strtolower(\$ _SERVER["HTTP_REFERER"])))" [12]</pre>

CHALLENGE-RESPONSE

Defenses		
Defense name	Defense description	Code example
Challenge-Response	CAPTCHA, Re-Authentication (password), One-time Token	

This defense is based on the user authentication if a requested action will be performed. The main idea is to use a CAPTCHA (a challenge-response which is used to ensure that a request is made by human), which will generate a random word, and the user will need to enter it correctly to perform the requested action. This defense will be most used in forums or blogs, where the prevention of automated posting is necessary. Example is shown in table V.

Other defenses implemented will be checking the referrer header, as shown in table IV. In that case the code implemented checks if the submitted form is from the same page or different URL address. If the headers match then the form value is submitted to the requested page and the action would be done. PHPProxy compares these headers as shown in the code example, so if the header is coming from example.com and the host of the requested page matches with it, then the form value will be submitted. The main principle consists in cross-checking the referrer value.

V. TESTING

Table VI and VII contain test results for each scenario. In this scenario each attack and defense was applied. Whenever a different operating system or browser has been used, the attack implemented in particular scenario has been tested. The test was then repeated and the result recorded.

SCENARIO 1

Test number	Testing Results			
	Defense applied	Browser /OS	Attack	Result
Test number 1	Token	Firefox/Windows	Attack number 1	PASS
Test number 2	Token	Firefox/OSX	Attack number 1	PASS
Test number 3	Header	Safari/Windows	Attack number 1	PASS
Test number 4	Header	Safari/OSX	Attack number 1	PASS

From the test result can be seen that the applied defenses in the scenario 1 have been successful in all test numbers.

Scenario is described in Section III A, which is login scenario.

Attack number 1- Described in Section III C.

The defenses applied here are randomly generated tokens, which are applied to every submitted form and referrer header.

SCENARIO 2

Test number	Testing Results			
	Defense applied	Browser /OS	Attack	Result
Test number 1	Token	Firefox/Windows	Attack number 2	PASS
Test number 2	Token	Firefox/OSX	Attack number 2	PASS
Test number 3	Header	Safari/Windows	Attack number 2	PASS
Test number 4	Header	Safari/OSX	Attack number 2	PASS

From the test result can be seen that the applied defenses in the scenario 2 have been successful in all test numbers. The defenses applied here are randomly generated tokens, which are applied to every submitted form and referrer header.

Attack number 2- Described in Section III C.

Scenario is described in Section III A, which is message board (forum), change password.

A. Evaluation of the experiment results

The tests of the experimental tool have been performed and the results are shown in testing Section V.

Defense against tokens has been discussed in Section II.A as the most commonly used by today's web applications. The results indicated that generation of random tokens has stopped more than 90% of attacks. This defense is easy to implement and with a designed class, which can be included in every file, making it the best known defense for web applications.

Appending C: An Experimental Analysis and Possible Solution for the Cross Site Request Forgery Attack

The disadvantage to this defense is that the attacker can predict a random token. This can be done by observing the web application in detail, and find the generated algorithm.

The referrer header defense has stopped many attacks, but still has some major complications. The main advantage to this defense is when a server is using cross domain policy; the method works as expected and stopped most of the attacks. The disadvantage is that a user can switch off sending referrer header on their browser, which makes the defense useless.

Captcha is very simple and easy to spoof defense, and many attackers need just short time to pass it. The defense has the advantage to stop attacks from computer based attacks like robotic attacks, because a confirmation is required.

VI. FUTURE WORK

Securing web applications against flash attacks needs to be implemented. Increasing the extension of the PHPProxy to handle such kinds of attacks could be done in the future. The report has concentrated mostly on defense against images and iframes attacks, but it has been mentioned that flash files can affect the users and web pages in the same way.

The PHPProxy server is working on the server side, so it can handle only the responses, which are coming back from the server to user. The request is made by user side, which means that different programming language could be applied, as JavaScript. The future work may include implementation of the interception of the user side to process http requests. This could help the end user to see what requests browser makes to server.

VII. CONCLUSION

This paper has described several cross site request forgery (CSRF) attacks that are used against web applications. We have also given examples of the most widely used defenses. These defenses have been implemented in an existing tool, PHPProxy, which is a server side HTTP proxy. Attacks were modified and implemented into scenarios to test defenses. From the results it can be seen that some of the expected defenses, which are used in most web applications, have been completely successful.

The extension tool PHPProxy has been successful in demonstrating the defenses against possible attacks. CSRF tool has been designed to protect end user against several attacks.

Defenses using tokens are the most commonly used by today's web applications. The results indicated that generation of random tokens has stopped more than 90% of attacks. This defense is easy to implement and with a

designed class, which can be included in every file, making it the best-known defense for web applications.

The disadvantage to this defense is that the attacker can predict a random token. This can be done by observing the web application in detail, and find the generated algorithm.

Presented and demonstrated attacks show that CSRF attacks are dangerous and web applications need to be better secured. Web developers and users can use tools as the one designed to protect their web applications and themselves.

REFERENCES

- [1] Brodtkin, J. (2007) The top 10 reasons Web sites get hacked. *Network World*. Vol. 24 Issue 39, p1-20
- [2] Stuttard, D. Pinco, M. (2007) *Web Application Hacker's Handbook*. Published by: Wiley Publishing, Indianapolis, Indiana
- [3] Ahmad, D (2008) *The Confused Deputy and the Domain Hijacker*. [Online] IEEE Digital Library
- [4] Willis, Ch. (2009) *Preparing for the Cross site request forgery defense*. Available at: <http://www.blackhat.com/presentations/bh-dc-08/Willis/Whitepaper/bh-dc-08-willis-WP.pdf> Accessed on: 10 February 2011
- [5] Ramarao, R. Radhesh, M, Pais, A. (2009) *Tool for preventing image based CSRF attacks*. Available at: <http://isea.nitk.ac.in/rod/csrf/PreventImageCSRF/icscf09PreventImageCSRF.pdf> Accessed on: 22 February 2011
- [6] Barth, A. Jackson, C. and Mitchell, J.C. (2008) *Robust defenses for cross-site request forgery*. [Online] In Proc. ACM Conference on Computer and Communications Security (CCS)
- [7] Son, S. (2009) *Prevent Cross-site Request Forgery: PCRf*. Available at: http://www.cs.utexas.edu/~samuel/PCRf/Final_PCRf_paper.pdf Accessed on: 7 February 2011
- [8] Jovanovic, N. Kirda, E. Kruegel, CH. (2006) Preventing Cross Site Request Forgery Attacks. *Securecomm and Workshops*. p:1-10
- [9] Chen, S. (ND) *PHP Simple HTML DOM Parser*. Available at: <http://simplehtmldom.sourceforge.net/> Accessed on: 17 March 2011
- [10] Gabriel, C. (2008) *Scraping Data: PHP Simple HTML DOM Parser*. Available at: <http://www.bitrepository.com/php-simple-html-dom-parser.html> Accessed on: 21 March 2011
- [11] Anonymous (2005) *Digital point forum*. Available at: <http://forums.digitalpoint.com/showthread.php?t=101263> Accessed on: 15 March 2011

Appendig D: An Investigation into Possible Attacks on HTML5 IndexedDB and their Prevention

The 13th Annual Post-Graduate Symposium on The Convergence of Telecommunications, Networking and Broadcasting (PGNet 2012), Liverpool, UK.

Appendig D: An Investigation into Possible Attacks on HTML5 IndexedDB and their Prevention

Stefan Kimak , Dr. Jeremy Ellman, Dr. Christopher Laing
 Northumbria University, CEIS
 Newcastle upon Tyne, UK
 stefan.kimak@unn.ac.uk , jeremy.ellman@unn.ac.uk, christopher.laing@unn.ac.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 Hyper Text Markup Language v.5 (HTML5) this evolution will increase, with browsers offering greater levels of functionality. However, with the introduction of HTML5, new persistent database security vulnerabilities could impact on this functionality. IndexedDB functionality involves storing application data on the client computer. As client data including sensitive information is now stored locally, consequently vulnerabilities within HTML5's IndexedDB scheme could have devastating consequences. This paper will investigate potential vulnerabilities, and propose security framework for HTML5's IndexedDB files that could be included as part of an inherited web browser security.

Keywords- component web security; HTML5; IndexedDB

VIII. INTRODUCTION

HTML5 is still in the standardisation process. The motivation for the changes and enhancements coming with HTML5 is that the web browser should be capable of running client side applications. That is, client side process will be able to avoid the ineffectiveness and network connectivity issues found in server side applications. Consequently, major browsers now support the majority of the new HTML5 components and Application Programming Interfaces (API). Therefore an HTML5 browser client side database may well contain stored data from online services that makes use of the new functionality of HTML5. It is suggested that this new level of client side data storage will ensure that such HTML5 enabled browsers are going to be a "juicy target for cyber-attacks" [1]. Consequently HTML5 opens up entirely new security challenges and loopholes [2].

This paper is going to investigate possible vulnerabilities and attacks, which might be possible in HTML5's IndexedDB. These attacks are mostly known, as Web applications attacks, however, with HTML5 and greater level of data stored on the client side, then these attacks will have potentially greater consequences.

This paper presents a solution to possible attacks, which might be a framework to provide the client database with input validation. The following section will discuss the background to the new HTML5 standard, security issues and

vulnerabilities. In section IV a possible security framework designed to circumvent these issues will be presented.

IX. BACKGROUND

HTML is the main programming language for web pages. Since it arrived in 1990 [3] the versions have evolved to allow web applications to act as desktop applications [4]. The World Wide Web Consortium (W3C) and Web Hypertext Application Technology Working Group (WHATWG) are currently collaborating on the latest development of HTML and its features and capabilities. These are collectively known as HTML5 [5].

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 client side and accessed anytime that the application requires it [6].

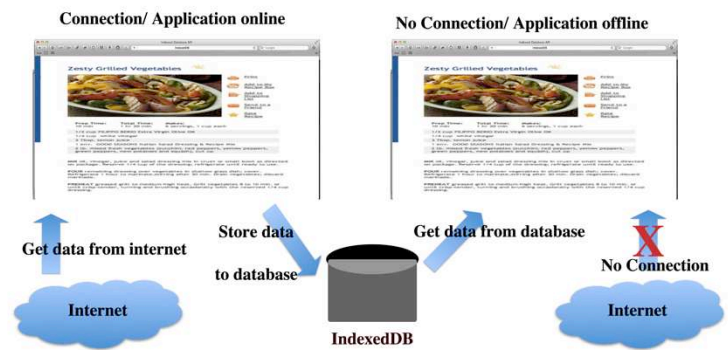


Fig 1. HTML5's IndexedDB functionality

When a client connects to a HTML5 web application for the first time, an API transaction will be created. The application will ask the client to store data locally. This data will be stored in a client side database, IndexedDB. If a network failure occurs, the data from the database will be read and the client can still use the application. This means that an application can be run offline as seen in Fig. 1. Pictures and text from pages could be stored in IndexedDB.

The advantage of HTML5 compared to desktop programs is that web applications do not require any installation or startup configuration and will also run on any device that supports HTML5, such as laptops, phones or tablets. This

reduces the barrier of entry for new customers since clients can begin taking advantage of the web applications just by visiting the relevant web site [7]. Benefits of client side storage include connectivity failure, where an application can be used even a connection is not available. Offline content also allows access to and creation or modification of data stored locally that the application may use offline. Currently, websites behave as desktop applications; the application reloads the content instantly, without needing to reload the page. The performance improvements include less bandwidth usage as data is stored on client side and the data is transferred only when the web application requires it [8].

Web-based software is increasingly popular data as applications are constantly available on the Web as services. This means that end client software will be developed using web technologies [9]. Such applications and services consist of data and code that can be located anywhere in the world. This allows a wide range of applications to support multiple clients and share data worldwide. With the help of client-side storage, data can be periodically saved to the browser while the client completes it. After the data has been processed the information is then transmitted to the server [10]. This will speed up application load time [11].

HTML5's IndexedDB is a client side database integrated into the client's web browser. The application uses local data stored on a client system. [12]. It caches large data from server to client side using JavaScript Object Stores, equivalent to tables in relational databases [13]. An application stores JavaScript objects into IndexedDB when the application is connected to the Internet. When the connection terminates for any reason the application can fetch data from the IndexedDB and the application may then be run offline [14]. The application runs as if it were a desktop application. This will be beneficial to mobile clients as they can use the application even if the connection is lost due to poor signal for example, however it can be run on many platforms such as tablets or smartphones [15].

A. Related Work

HTML5 is a new standard, so obviously not many security investigations and preventions against vulnerabilities exist. Anyways, client side vulnerabilities might be secured by existing preventions, such as input validation. Domain Name Systems (DNS) spoofing attacks can be prevented by using Transport Layer Security (TLS). Also encryption of client data is required for a better security. Some possible vulnerability in HTML5 and client side databases has been point out by West [5]. These vulnerabilities are going to be discussed more in detail, also possible attacks, which might be possible from these vulnerabilities.

B. Structuring the database

Unlike other web-based databases such as Structured Query Language (SQL) databases that use tables for storing data, IndexedDB uses object stores. Multiple object stores use a single database. Keys are assigned to every value in an

object store within a database, with keys being assigned by key path or by a key generator.

IndexedDB was created to allow local storage of data, however this does not include the following features:

- 1) *Internationalised sorting* – Internationalised sorting cannot be supported with IndexedDB due to the wide variety of scripting languages in use in modern day web applications. While the database can't store data in a specific internationalised order, the client can sort the data that is read out of the database manually.
- 2) *Synchronising* - Server-side databases currently cannot be synchronised due to the time-consuming implementation required for its development. Developers have to write code that synchronises a client-side indexedDB database with a server-side database, which is time consuming.
- 3) *Full text searching*- The API does not have an equivalent of the LIKE operator in SQL. W3Schools [16] describes it as, "The LIKE operator is used to search for a specified pattern in a column".

By assuming that these limitations do not have an important impact of security issue, the explanation is very crucial part of IndexedDB. IndexedDB does not use SQL; it uses queries on an index that produces a cursor, which is used to iterate across the result set. Index is a data structure (a way of storing and organizing data) that improves the retrieval of data from database. Anyways an IndexedDB is a No Sequel (NOSQL) database, which means that to perform an SQL injection is not possible. IndexedDB is built on a transactional database model. Everything the client does in IndexedDB always happens in the context of a transaction. The IndexedDB API provides lots of JavaScript objects that represent indexes, tables, cursors, but each of these is tied to a particular transaction. Although, applications cannot execute commands or open cursors outside of a transaction. Transactions have a defined lifetime, so if someone attempts to use a transaction after it has completed the process of passing the object, it will throw error message (exception). The transaction model carries many advantages, including the prevention of instances whereby a client may try to run more than one instance of a web application at the same time. Without transactional operations, the two instances could create database issues and affect functionality.

C. HTML5 vs. HTML4 storage

Web developers have used cookies for storing data on the client side since Netscape Corp introduced the idea in 1994 [17]. Cookies are limited as a website could only store a very small amount of data. Cookies are sent to server with every HTTP request, which is slowing down the connection. HTML5 introduces several alternatives to cookies and storing data on the client side, which is a Local storage [15]. Part of a local storage is indexedDB [18].

X. SECURITY VULNERABILITIES

As HTML5 can be run on multiple platforms, potential attackers may be more able to attack clients of a wider range

of browsers. Any security breaches that occur in a web application do not open the client's data to attack, as this information is stored only locally on the client machine, and therefore can only be accessed when this machine itself is compromised [19].

IndexedDB operates by using the same-origin policy (SOP), which involves linking stored data to a particular domain or subdomain, so that the data cannot be accessed from any other source [20].

The same-origin policy is the only form of browser protection against potential security threats. It 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 [21]. The prevention of data or attacks coming from a different domain is possible. Web browsers are using this prevention technique against untrusted site attacks.

HTML5's new functionality allows attackers to access untrusted sites, even if they are on a different domain, meaning that the SOP will not apply here. Security vulnerability and potential attacks might be possible here, and the attacker will be able using hacking techniques to reach and access the database from a different domain [22]. If the website or application is vulnerable to XSS attacks, then the attacker could steal the data from client database. When the SOP is not correctly configured, then content from different web sites will allow attackers to manipulate the data through their code access.

The SOP prevention is not enough to prevent an attacker to get the data from a different domain. As the data is stored on the local machine in database, the applications are limited to access only data created by particular application on a domain. This is a security vulnerability of web browsers, where the client database is situated and an attacker might compromise the client data [22].

[23] states, "The SOP is not the correct security mechanism and requires redesign to meet the access-control requirements of Web-based assets".

The data stored in IndexedDB is not using any kind of client input validation, which may be why possible flaws exist. The data is stored on a client's machine as unencrypted files. This means that any attacker with access to the file system can directly extract the information from the files. In order to mitigate this, the application must obviously encrypt any sensitive information before writing it to the database. In addition, operating system level mechanisms (file system permissions, file system level encryption, etc.) should be used to prevent access to the files by unauthorized users.

The validation hasn't been implemented by W3C, but needs to be implemented by the browser [24]. The database and API is still in draft, but the validation of data needs to be strictly applied. As there is not any input validation, any site can store potentially dangerous JavaScript code into client local machine. As the client is not aware what code is stored onto their local disk, security vulnerabilities apply here.

Coming back to same origin weaknesses, that can lead to attacks such as cross-site request forgery (CSRF), XSS, and Web cache poisoning [23].

Using HTML5 localStorage to replace session data stored in a cookie improves the application's scalability and prevents simple CSRF attacks because, unlike a cookie data in localStorage is not automatically sent [25].

An example of client side vulnerability might be XSS. XSS is an attack technique that forces a web site to execute malicious code in a client's Web browser [22]. XSS may be used to steal all the data stored in a client's browser or to change client settings [26].

Web application security is crucial in managing threats. If a security hole exists as XSS the whole client database might be compromised [27]. An attacker is able to read the complete client database of a domain exploiting XSS vulnerability. Storing sensitive data is dangerous in this case, as there is a possibility that all the data of domain can be compromised and accessed by attacker.

XI. POSSIBLE PREVENTIONS

The following framework could be used in the development of IndexedDB web applications for the prevention of such attacks outlined above. The framework will be divide into parts as:

- Client side data encryption
- Code analysis
- Input validation
- SOP

The framework will be implemented to browser. This will add the required security for storing data in a database. The data will be stored as encrypted files and will encrypt data every time an application writes the data into a database. When the application reads the data back then the decryption process will be initiated. Client side encryption and decryption of data stored in IndexedDB, which will be a part of browser extension. This will be based in web browser as extension. The extension will be a third party encryption component as JavaScript Microsoft Exchange ActiveSync (EAS) or Secure Hash Algorithm (SHA)-256 implementation [28]. This method of encryption use verification hash, which ensure that the encryption is correct, without the decryption of data on the server. The data cannot be decrypted on the server, only in the web browser.

The data could be safely manipulated and only be retrieved by origin of the site that creates it.

The framework will protect the client database from various attacks. This is done be static and dynamic analysis of code. The code or data will be analyzed when the data will be written or read from a database. In some cases the data can include JavaScript code, which might be potentially dangerous.

The framework will consist of static and dynamic data validation.

The solution is to build a framework, which will check the data in IndexedDB. The data will be checked every time the application requires transaction to database. The transaction

might be read or written. The framework will consist of two parts:

- The first part is static analysis of code, which is going to be written to database.
- The second parts will be more complex in the dynamic code analysis, where the code will be analysed during run time.

Static analysis of code will aim to highlight any possible attacks that become apparent.

Dynamic analysis will be done when the application has been launched. The analysis will be based on checking the API call from web application to client side database and reverse. Before the actual action gets executed the code analysis checks the call and after the successful processing of the call, the action will be performed. Other method to secure the code and the client side data is to use the HTML5 sandbox. This method provides the functionality of locking down the content from third party content.

When sandbox is enabled it locks down the harmless content that behind the scenes attempts to access privileged information. This means that third party content couldn't access all the data in client side database.

Input and output validation to secure the client side application and database will be build on the top of IndexedDB API. The validation takes the string and returns true if the input is permitted by the input validation policy) otherwise returns false.

Possible solution to prevention against XSS attacks will be to secure the input validation. Other solutions might include using a different policy, as the current same origin-policy is not secure. Potential policies might include Content Security Policy (CSP). The CSP restricts common attack vectors in the client browser. The CSP employs a set of directives that define the security policy for all types of webpage content on the webpage [23].

XII. CONTRIBUTION

The main contribution of this paper will be the investigation of security mechanisms for IndexedDB, also the implementation of a security framework to address these issues.

This framework will be developed from an analysis of identified vulnerabilities in HTML5.

A series of experiments will be trialed by using static and dynamic code analysis will be used to test the proposed framework. The outcome of this work will aid in securing HTML5 and will be available to W3C and Web Hypertext Application Technology Working Group.

XIII. CONCLUSION

This paper has presented possible vulnerabilities and attacks in HTML5's IndexedDB. Although attacks are possible because the standard is not completed yet, but mostly because vulnerabilities such as XSS are a crucial part of today's web applications. Vulnerabilities exist in all web application, but securing client side, especially when the sensitive data is going to be stored is a crucial part. This paper

has point out vulnerabilities as XSS and the downfalls of same origin policy in HTML5's IndexedDB.

This paper also briefly presents a possible solution to input validation, where the data needs to be encrypted before it has been read or written. Possible solutions to XSS in HTML5's IndexedDB may include developing a new security policy that improves on the same origin policy.

- [1] Ryck, P. Desmet, L. Philippaerts, P. Piessens, F. (2011) A Security Analysis of Next Generation Web Standards, (ENISA). Tech. Rep.
- [2] 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
- [3] Sefton, P. (2009) Towards Scholarly HTML. Serials Review. Volume 45, Issue 3.p.154-158
- [4] West, W. and Pulimood, M. (2012) ANALYSIS OF PRIVACY AND SECURITY IN HTML5 WEB STORAGE. ACM digital library. Journal of Computing Sciences in College. Volume 27 Issue 3
- [5] Mitchell, E. (2010) Standards, Efficiency, and the Evolution of Design. Journal of Web Librarianship. Volume 4, Issue 4
- [6] Clark, J. (2010) HTML5. Online Journal. CINAHL database. ISSN:0146-5422 .Volume 34 Issue 6, p12
- [7] Harjono, J. Ng, G. Kong, D. Lo, J. (2011) Building smarter web applications with HTML5. Conference of the Center for Advanced Studies on Collaborative Research
- [8] Hilerio, I. (2011) Building offline access in Metro style apps and websites using HTML5. Available at: <http://channel9.msdn.com/Events/BUILD/BUILD2011/PLAT-376T>. Accessed on: 20 February 2012
- [9] Taivalseeri, A. Mikkonen, T. (2011) The Web as an Application Platform: The Saga Continues. Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference.
- [10] Nichols, V. (2010) Will HTML5 restandardize the Web? Computer. Volume: 43, Issue: 4. P. 13-15
- [11] Wisniewski, J. (2011) HTML5. Online journal. ISSN:01465422. Vol. 35 Issue 6, p53-56, 4p
- [12] Casario, M. Elst, P. Brown, Ch. Wormser, N. Hanguiez,C. (2011) HTML5 Solutions: Essential Techniques for HTML5 Developers. Publisher: FRIENDS OF ED; 1 edition ISBN: 1430233869
- [13] Windows development (2011) IDBDatabase. Available at: <http://msdn.microsoft.com/en-us/library/windows/apps/hh441231.aspx>. Accessed on: 24 March 2012
- [14] Gihan, D. Karunarathna, D.G.M ; Udantha, G.P.D.M ; Gunathilake, J.A.I.M ; Pathirathna, P.S.P ; Rathnayake, R.A.T.L (2011) Database based and RESTful email system with offline web based email client. Advances in ICT for Emerging Regions (ICTer), 2011 International Conference.
- [15] Ijtihadie, R.M. Chisaki, Y. Usagawa, T. Cahyo, H.B. Affandi, A. (2011) Offline web application and quiz synchronization for e-learning activity for mobile browser. TENCON 2010 - 2010 IEEE Region 10 Conference. On page(s): 2402 - 2405 ISBN: 978-1-4244-6889-8
- [16] W3Scool (2012) Like Operator. Available at: http://www.w3schools.com/sql/sql_like.asp. Accessed on: 17 April 2012
- [17] Tappenden, A. (2008) A Three-Tiered Testing Strategy for Cookies. Software Testing, Verification, and Validation, 2008 1st International Conference. p: 131 – 140
- [18] Lennon, J. (2010) Create modern Web sites using HTML5 and CSS3 Implementing the canvas and video elements in HTML5. Available at: <http://www.ibm.com/developerworks/web/tutorials/wa-html5/wa-html5-pdf.pdf>. Accessed on: 17 March 2012
- [19] Hsu, F., Chen, H.,(2009) Secure file system services for web 2.0 applications. CCSW '09 Proceedings of the 2009 ACM workshop on Cloud computing security. Pages 11-18

Appendix D: An Investigation into Possible Attacks on HTML5 IndexedDB and their Prevention

- [20] Mozilla developer network (2012) IndexedDB . Available at: <https://developer.mozilla.org/en/IndexedDB> Accessed on: 15 February 2012
- [21] Takesue, M. (2008) A Protection Scheme against the Attacks Deployed by Hiding the Violation of the Same Origin Policy. Emerging Security Information, Systems and Technologies, 2008. SECURWARE '08. Second International Conference p. 133 - 138
- [22] Stuttard, D. Pinto, M. (2011) Web Application Hacker's Handbook 2nd ed. Published by: Wiley Publishing, Indianapolis, Indiana
- [23] Saiedian, H. Broyle, D. (2011) Security Vulnerabilities in the Same-Origin Policy: Implications and Alternatives. Computer Journal. Volume: 44, Issue: 9, p29 – 36.
- [24] W3C (2010) IndexedDB . Available at: <http://www.w3.org/TR/IndexedDB/> Accessed on: 5 March 2012
- [25] Mayer, M. (2011) Computer (Same origin Policy) Published in: Computer, IEEE Journals & Magazines. Volume: 44, Issue: 12, p: 6 - 7
- [26] Lawton, G. (2007) Web 2.0 Creates Security Challenges. Computer. Volume 40, Issue:10, p. 13-16
- [27] Schmidt, M. (2011) HTML5 Web Security. Available at: http://media.hacking-lab.com/hlnews/HTML5_Web_Security_v1.0.pdf. Accessed on: 20 April 2012
- [28] Morse, R. Nadkarni, P. Schoenfeld, D. Finkelstein, D. (2011) Web-Browser encryption for personal health information. BMC Medical Informatics and Decision Making, Volume 11

Appendix E: Performance Testing and Comparison of Client Side Databases Versus Server Side

The 14th Annual Post-Graduate Symposium on The Convergence of Telecommunications, Networking and Broadcasting (PGNet 2013), Liverpool, UK.

Appendix E: Performance Testing and Comparison of Client Side Databases Versus Server Side

Stefan Kimak , Jeremy Ellman
Northumbria University, CEIS
Newcastle upon Tyne, UK
stefan.kimak@unn.ac.uk , jeremy.ellman@unn.ac.uk

Abstract- Databases are a crucial part of today's Internet based web applications. To date, almost all web applications have used server side databases. With the adaption of HTML5, which is currently in the process of being standardized by W3C, new client side databases are being introduced that will be embedded in the web browser. Client side databases have the advantage of reducing load on the web server, but the disadvantage that database performance will vary depending on the user's web browser and in particular how the browser's designers have chosen to implement the IndexedDB API. In this paper we describe appropriate database benchmarks and apply these to three current web browsers, Google Chrome 24, Firefox 17. We also compare these results with the popular server side database MySQL. The benchmarking is based on writing, reading and deleting database data. The comparison of benchmarks shows the suitability of client side versus server side databases.

Our findings are that there are significant performance differences between the indexedDB implementations. The main differences are discussed in relation to the benchmark results. Irrespective of browser differences, the results show that client side databases perform well in comparison to server side databases whilst reducing network latency concerns.

Keywords-component; HTML5; IndexedDB, Benchmarking

XIV. INTRODUCTION

The World Wide Web Consortium (W3C) is currently standardizing HTML5 for the next generation of web applications. With the new HTML5 standard come new functionalities. These include IndexedDB , which is a client side browser based database. The need for IndexedDB reflects requirements for more storage space, that persists beyond page refreshes whilst avoiding data transfers to the server. Storing the data on the client machine can resolve this issue. Traditional SQL relational databases have been used since 1976 (Chamberlin, 1976). The changes some companies made were to use object-oriented databases, as these have some advantages over SQL. The motivation for the changes and enhancements coming with HTML5 is that the web browser should be capable of running client side applications in the same way that it can run desktop applications. That is, client side process will be able to avoid the ineffectiveness and network connectivity issues found in server side applications. Consequently, major browsers now support the majority of the new HTML5 components and Application Programming Interfaces (API). Therefore an HTML5 browser client side

database may well contain stored data from online services that makes use of the new functionality of HTML5. Traditional desktop applications, like word processors and spreadsheets, might be used with Web applications. This means that client will not need to install any software on the computer, and will only need a Web browser (Stuttard, 2008). As Internet access is crucial for Web applications, and it should be possible to use these Web applications regardless of connection, so offline applications are an important part of this development.

The paper will compare various databases in different browser environments. Both client and server side databases are tested are. Client side databases include IndexedDB, LocalStorage and the deprecated WebSQL. Server side databases include MySQL with InnoDB and MyISAM. The benchmarking contains data from various testsets, which include the common create, read, update and delete (CRUD) data operations to and from a database. A comparison of databases will be discussed in the next section, where some background data will close up the details about particular database.

IndexedDB is implemented differently across browsers. Firefox uses SQLite and Chrome uses LevelDB (LevelDB is not a SQL database (W3C, 2011)). Like other NoSQL and Dbm stores, it does not have a relational data model, it does not support SQL queries, and it has no support for indexes, so even if IndexedDB is built into Firefox, a SQL-backed technology with SQL-like overhead is actually used. (MDN, 2011).

XV. BACKGROUND

A. Relational Database

A Relational Database Management Systems (RDMS) represent records organized in tables. The structure of tables consists of columns and rows. Columns represent data categories and row the data (Eaglestone, 1991). The structure of relational databases is good for managing large amount of structured data. The disadvantage is their inflexibility, because their only data structure is tables. They have a problem handling complex multimedia files, which is important for complex web applications (Harrington, 1998). Relational databases are computer programs used to store information in tables. These tables contain rows and columns used to sort and retrieve information. The rows and columns contain related information about the subject of the table. The

database administrator can define the relationships among the various types of data. Relational databases require data to be entered as integers, strings or real numbers. This data must then be accessed through SQL queries (Conolly and Begg, 2004).

An entity relationship diagram (ERD) is a useful technique for managing the development of a database information system. An ERD models data into logical and easy to understand graphical representations (Thalheim, 1998). Entity relationship diagrams illustrate the logical structure of databases. Boxes are used to represent entities, diamonds are normally used to represent relationships and ovals are used to represent attributes. The relationship between the boxes can be:

- One to one
- One to many
- Many to many

The relationships from the ERD are going to be used when creating the database tables to create relationships between them. So the diagram is showing the relationship between the objects.

B. *SQLite server Side Database*

SQLite is the most widely used database for web sites according to survey by Netcraft (Netcraft, 2006). The main advantage of SQLite is its availability (it is used by mobile browsers on Android and iOS, by PhoneGap for mobile applications and by Chrome 15 and Safari 5).

The main disadvantage of SQLite is that the W3C do not support SQLite anymore and some browsers, like Firefox, removed the SQLite support in their latest versions.

Embedding SQLite in web browsers has resulted in adding SQLite to the HTML5 Web Storage standard and after discussion inside the W3C Web Applications Working Group (Mozilla, 2009).

C. *NoSQL Client Side Database*

NoSQL (Not only SQL)(Strozzi, 1998) is the solution of database that is not relational or object oriented. It does store data in key/value format. The database can handle a large amount of data, where the relational model is not needed. They were really used only at the time when the designers of web services with very large number of users discovered that the traditional relational database management systems (RDBMS) are fit either for small but frequent read/write transactions or for large batch transactions with rare write accesses, and not for heavy read/write workloads (which is often the case for these large scale web services as Google, Amazon, Facebook, Yahoo and such)(Tudorica, 2011).

Advantages of NoSQL databases

NoSQL databases generally process data faster than relational databases (Leavitt, 2010). Relational databases are usually used by businesses and often for transactions that require great precision. Developers usually do not have their NoSQL databases support ACID (atomicity, consistency, isolation, durability) in order to increase performance, but this

can cause problems when used for applications that require great precision. NoSQL databases are also often faster because their data models are simpler (Banker, 2010). According to Leavitt (2010) there is a trade-off between speed and model complexity but it is frequently a trade off worth making.

Disadvantages of NoSQL databases

NoSQL databases face several challenges, which are overhead and complexity. They do not work with SQL queries, which means that they need to be manually programmed. In cases of simple tasks they perform fast, but is time consuming for complex queries (Leavitt, 2010).

Reliability- Relational databases natively support ACID (Conolly and Begg, 2004), while NoSQL databases do not. Therefore NoSQL databases do not offer reliability. For performing this functionality additional programming is required.

Consistency- The lack of support ACID transactions leads to compromising consistency. Banking sites are using Consistency in their applications; therefore usage of NoSQL databases might be a problem (Shashank, 2011). On the other hand they provide better performance and scalability. Most organizations are unfamiliar with NoSQL databases and thus may not feel knowledgeable enough to choose one or even to determine that the approach might be better for their purposes (Stonebraker, 2010). Unlike commercial relational databases, many open source NoSQL applications do not yet come with customer support or management tools. Each NoSQL database has its own set of APIs, libraries and preferred languages for interacting with the data they contain.

Few examples document-oriented NoSQL database include MongoDB, LevelDB, BerkeleyDB. The first two databases store the data on HDD. The BerkeleyDB uses ordered key/value store.

D. *LevelDB Client Side Database*

LevelDB is a fast key/value storage library written at Google that provides an ordered mapping from string keys to string values. The stored data is sorted by key and it provides an ordered mapping from string keys to string values (Dean, 2011).

E. *WebSQL Client Side Database*

W3C (2010) wrote that the WebSQL database API is off active maintenance. They cited lack of independent implementations as being the reason because most of the browser relied on SQLite as the underlying database. WebSQL database brought real relational database implementation onto browsers. Data could be stored in a very structured way.

F. *IndexedDB Client Side Database*

HTML5's IndexedDB is a client side database integrated into the client's web browser. The application uses local data

Appendix E: Performance Testing and Comparison of Client Side Databases Versus Server Side

stored on a client system (Casario, 2011). It caches large data from server to client side using JavaScript Object Stores, equivalent to tables in relational databases (Windows, 2011). An application stores JavaScript objects into IndexedDB when the application is connected to the Internet. When the connection terminates for any reason the application can fetch data from the IndexedDB and the application may then be run offline (Gihan, 2011). The application runs as if it were a desktop application. This will be beneficial to mobile clients as they can use the application even if the connection is lost due to poor signal for example, however it can be run on many platforms such as tablets or smart phones (Ijtihadie, 2011).

IndexedDB databases store key/value pairs. The values can be complex structured objects, and keys can be properties of those objects. Indexes use property of the objects for quick searching and sorted enumeration. A key is a data value by which stored values are organized and retrieved in the object store.

IndexedDB is built on a transactional database model.

Everything done in IndexedDB always happens in the context of a transaction. A transaction is an atomic and durable set of data-access and data-modification operations on a particular database. It is how a browser interacts with the data in a database. Any reading or changing of data in the database must happen in a transaction (MSDN, 2012). The IndexedDB API provides lots of objects that represent indexes, tables, cursors, but each of these is tied to a particular transaction. A command cannot be executed or cursor opened outside a transaction. Transactions have a defined lifetime, so attempting to use a transaction after it has completed throws exceptions (W3C, 2011). IndexedDB does not use SQL; it uses queries on an index that produces a cursor, which is used to iterate across the result set. Index is a data structure (a way of storing and organizing data) that improves the retrieval of data from database. The structure of an IndexedDB database can only be modified during a versionchange transaction. This means that the only time ObjectStores or indexes can be created or removed is during the versionchange transaction. Basically, the IndexedDB API automatically creates a versionchange transaction anytime a database is opened through the open method and one of the following two conditions occur:

- The requested database does not exist.
- The requested database version number is greater than the version number of the database on the client machine.

New implementation of open database

Chrome currently still implements the old specification rather than the new one. Similarly it still has the prefixed `webkitIndexedDB` property even if the unprefixed `indexedDB` is present.

New IndexedDB projects:

PouchDB - An implementation of CouchDB on top of IndexedDB. One of the premises is to offer the same

synchronization (master-to-master) decentralized capabilities of CouchDB on the browser.

BrowserCouch - A similar project but using WebSQL as browser storage.

Html5sql is a light JavaScript module (jQuery library) that simplifies working with IndexedDB. Its primary function is to provide a structure for the SEQUENTIAL processing of SQL statements within a single transaction. This alone greatly simplifies the interaction with the database. Many other smaller features have been included to make things easier, more natural and more convenient for the programmers. (<http://html5sql.com/index.html>). Table with 11000 entries created in: 1.405s.

G. Differences Between NoSQL and SQL Database

The main difference between NoSQL and a SQL database is how the data is stored. NoSQL uses key/value as the main storage. On the other hand SQL is a relational database, which means that it uses relations (called tables). The databases are different in scalability and performance. NoSQL database has advantages over SQL database because it allows scaling an application to new levels. The new data services require scalable structures, which can work in the cloud. In comparison the NoSQL database does not need a database administrator, or complicated SQL queries and still is considered faster in managing high amount of data. NoSQL does not however support SQL Joins, and relations between tables need to be manually programmed. The differences between relational and IndexedDB are in storing the data. Relational databases store tables with rows and columns of types of data. IndexedDB requires creating an object store for type of data and saving JavaScript objects to that store. Each object can have collection of indexes that make it faster to query and search across. IndexedDB does not support joins, where relational database does. The comparison of the query results using joins shows, that IndexedDB performs the query and renders the data faster. On the other hand the code in IndexedDB is much more complicated, as all the code needs to be manually done in JavaScript that is otherwise provided natively by SQL. IndexedDB can split array in chunks of small pieces and using `setTimeout`, instead of loop inserted the data faster in database.

XVI. TEST SUITE ENVIRONMENT

IndexedDB is the client side database. The database supports blobs and JavaScript objects. The applications were tested in Firefox (v.15) and Chrome (v.22). Both of these browsers fully support the IndexedDB API. The application is setup on the server, because IndexedDB does not support local servers. While Firefox uses the latest W3C specifications an `onupgradeneeded` event to determine if a database should be created or upgraded, Chrome still uses the older and now obsolete `setVersion` method.

Appendix E: Performance Testing and Comparison of Client Side Databases Versus Server Side

A. Benchmark for Client Side Database

The benchmark consists of functionality that adds records to database. The records are randomly generated as follow example: `ssn:"111-11-1115",name:"Donna",age:12,email:"donna123@gmail.org"` The records are objects of size 151 Bytes. The ssn is the key of the object stored in the database, which is also randomly generated. The generator is a JavaScript random number generator. Purpose of the ssn key is a keyPath that is the property that makes an individual object in the store unique. The benchmark measures the time needed to generate the values and the time needed for insertion. For the benchmarking and comparison the insertion is the only important part. The benchmark firstly opens a database connection, and creates an object store to store the generated records.

B. Benchmark for Server Side Database

For server side database a JavaScript function will generate a random name and surname and insert these data into database. The JavaScript function consists of array of names, which will be randomly put together and inserted as one array into database. The scenario is based on a real application, where the data of people is stored and retrieved from/to a database. This scenario is based for benchmarking; it does not consider security of browser side databases. The code calls the onSuccess function for every record, as the cursor points the records retrieve it one by one and shows it. All the retrieved data is stored in memory and from there output to the browser. Since there is no other way to get all records from database, this way might be slow in some cases, where the database contains a large quantity of data.

For insert- The benchmark is measures the time or request for insertion to time of response.

For retrieve- For the retrieve the benchmark measures time of request to actual data appearing on the screen. This code structure is not optimized, as the record retrieval to actual output of the screen might take longer than just measuring the time of the response.

XVII. RESULTS AND ANALYSIS

The performance testing of IndexedDB has been compared to others alternatives as local storage and WebSQL (which has been deprecated). The tests performed were inserting data in client side database to show the time of actual insertion. The IndexedDB has shown that it can insert the data very quickly, in most cases faster than the others alternatives. This demonstrates why IndexedDB was chosen by W3C as client side database, due to the potential of fast inserting and reading the data. The tables below show insertion of records into MySQL with MyISAM and InnoDB types. Comparison between SQL and IndexedDB databases are visibly different

and the results are showing the big potential of client side storage.

Databases are tested on web server and local machine. These show data insertion time. Results in tables show the storage size of data in databases and it can be seen that IndexedDB uses more storage than a traditional relational SQL database.

Test	Amount of data	Time for insertion 1	Time for insertion 2	Size of file	Type
Test1	1000	1.09s	1.04s	48KB	MYISAM
Test2	2000	2.05s	2.04s	94KB	MYISAM
Test3	5000	5.58s	5.14s	232KB	MYISAM
Test4	10000	12.178s	11.175s	461KB	MYISAM
Test5	20000	23.47s	20.31s	923KB	MYISAM
Test6	30000	31.84s	36.24	1.4MB	MYISAM

Table 1. MySQL database performance testing with MYISAM

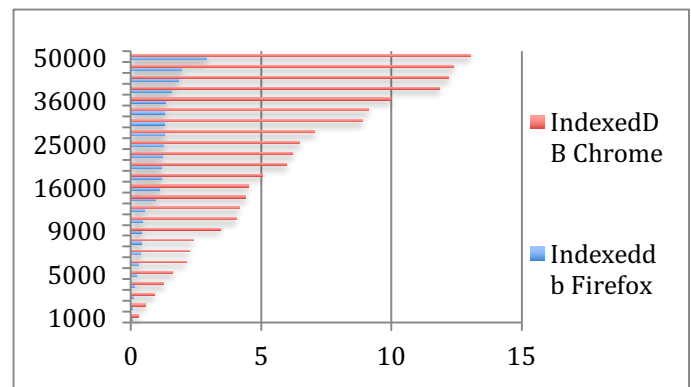


Fig. 1 Performance testing: Insertion of records into database (IndexedDB in Firefox and Chrome)

From the results can be seen that IndexedDB perform faster in Firefox. The reason is that Firefox implements and maintains the newest code architecture, as Chrome is still behind. The results show a visible difference in time, where the possible reason is usage of backend database.

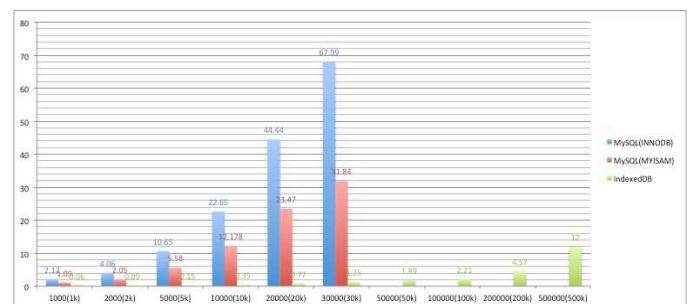


Fig. 2 Performance testing: Insertion of records into database (Firefox)

Test	Amount of data	Time for insertion	Type / Browser
------	----------------	--------------------	----------------

Appendix E: Performance Testing and Comparison of Client Side Databases Versus Server Side

Test1	1000	0.05s	LocalStorage/ Chrome
Test2	2000	0.2s	LocalStorage / Chrome
Test3	5000	0.75s	LocalStorage / Chrome
Test4	10000	1.38s	LocalStorage / Chrome
Test5	20000	2.93s	LocalStorage / Chrome
Test6	25000	4.61	LocalStorage / Chrome
Test7	29000	5.1s	LocalStorage / Chrome
Test8	30000	Failed	LocalStorage / Chrome
Test9	200000		LocalStorage / Chrome

Table 2. LocalStorage database performance testing in Chrome

Test	Amount of data	Time for insertion	Type / Browser
Test1	1000	0.09s	WebSQL/Chrome
Test2	2000	0.17s	WebSQL/Chrome
Test3	5000	0.37s	WebSQL/Chrome
Test4	10000	0.81s	WebSQL/Chrome
Test5	20000	1.73s	WebSQL/Chrome
Test6	30000	2.24s	WebSQL/Chrome
Test7	50000	3.45s	WebSQL/Chrome
Test8	100000	7.35s	WebSQL/Chrome
Test9	200000	20.45s	WebSQL/Chrome
Test10	500000	50s	WebSQL/Chrome
Test11	600000	76s	WebSQL/Chrome
Test12	700000	80s	WebSQL/Chrome
Test13	800000	91s	WebSQL/Chrome
Test14	1000000	Failed	WebSQL/Chrome

Table 3. WebSQL database performance testing in Chrome

Test	Amount of data	Time for insertion	Size of file	Type / Browser
Test1	1000	0.06s	918KB	IndexedDB/FF
Test2	2000	0.09s	1,4MB	IndexedDB/FF
Test3	5000	0.15s	2,8MB	IndexedDB/FF
Test4	10000	0.35s	4,8MB	IndexedDB/FF
Test5	20000	0.77s	9MB	IndexedDB/FF
Test6	30000	1.25s	13,3MB	IndexedDB/FF
Test7	50000	1.89s	22.2MB	IndexedDB/FF
Test8	100000	2,21s	43,8MB	IndexedDB/FF
Test9	200000	4,57s	87,5MB	IndexedDB/FF
Test10	500000	12s	136MB	IndexedDB/FF
Test11	1000000	113s	218MB	IndexedDB/FF
Test12	2000000	127s	350MB	IndexedDB/FF
Test13	5000000	460s	665MB	

Table 4. IndexedDB database performance testing in Firefox

The tables are showing results of various databases in different browsers.

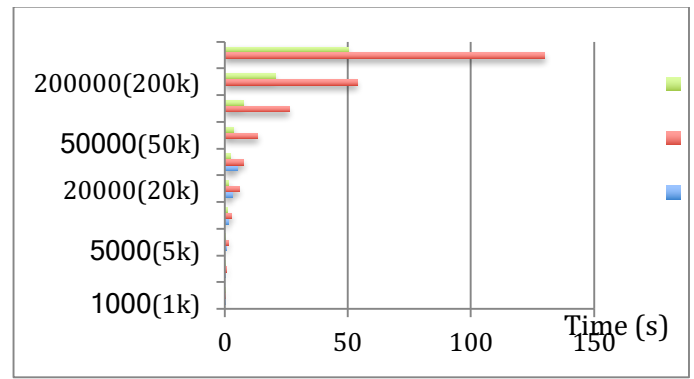


Fig. 3 Performance testing: Insertion of records into database (Firefox, WebSQL, Local Storage, IndexedDB)

XVIII. DISCUSSION OF ANALYSIS

A Comparison of the results of inserting the records into database in Chrome and Firefox can be seen in table 8. This shows that IndexedDB in Firefox handles the insertion of data faster than in Chrome. In many forums and support discussion groups, developers note that Chrome performance is worse because the support and code architecture is not updated to support fully IndexedDB. From the results it can be seen that IndexedDB performs faster than the MySQL database. The insertion size of object data into database starts at 1K, up to 500K. The insertion of the objects comparing between the databases is shown in the table.

Network Latency

Client side database process the data on the client side, where the network latency is minimal. All of the data is stored and retrieved from the client machine disk, based on the web application coding. Comparing the results of benchmarking the client side database handle the data much faster.

Scalability

Data scalability is very important for any web-based business. The web applications are becoming more scalable to fit the current market, so there is need for a database, which will handle this requirement.

Client side databases may have an advantage as they avoid communications costs and other overheads that server side databases may incur.

XIX. FUTURE WORK AND MOTIVATION

Internet Explorer 10 (IE10) benchmarking remains to be done, since currently IE10 does not support IndexedDB completely. Further experiments will look at various optimization of retrieving the data from database in a faster manner.

Appendix E: Performance Testing and Comparison of Client Side Databases Versus Server Side

New features of web browsers and new technologies such as HTML5 bring databases to web browsers. I believe that these features are the future of upcoming technologies, where the performance for the end user is important. The motivation is a investigation into performance of client side databases.

Future work will include discussion on how to properly merge those two to get good features from both sides or how do we need to extend one side or the other to avoid certain overheads.

XX. CONTRIBUTION

The main contribution of this paper is a benchmark that has the potential to compare performances of client side databases such as IndexedDB in different browser implementations.

The implementation of the benchmarking tool to perform various tests as inserting and reading data has been described and the results were compared and analyzed.

Future development will include a tool to delete data and measure the deletion time.

XXI. CONCLUSION

HTML5 IndexedDB performance has been examined in multiple client browsers. From the results it can be seen that IndexedDB performs faster in Firefox, but in Chrome the performance is lacking. Chrome still uses WebSQL, notwithstanding its deprecation since its performance is fast when compared to IndexedDB. In conclusion the backend technology in Firefox uses SQLite to implement IndexedDB, which supports SQL queries and indexes for search optimisation. On another hand Chrome is using its own backend LevelDB, which does not support SQL queries and indexes. We conclude then that the Firefox implementation of IndexedDB is a better solution. The performance depends on the browser as Firefox implementation of IndexedDB API is much more developed than Chrome or IE. Firefox uses SQLite as a backend database, and IndexedDB is implemented on the top of it. Researchers and developers note that IndexedDB performs faster with SQL as a backend. Comparing to Chrome implementation where IndexedDB is implemented on the top of LevelDB (which is NoSQL) is much slower than Firefox. On the other end WebSQL (deprecated) performs well in Chrome, whilst Firefox support for WebSQL has ceased.

REFERENCES

1. Alaric Snell-Pym (2010) NoSQL vs SQL, Why Not Both? Cloudbook Journal Vol 1 Issue
2. Banker, K. (2010) MongoDB and e-commerce. Available at: <http://kylebanker.com/blog/2010/04/30/mongodb-and-e-commerce/>
3. Connolly, T. Begg, C. (2004) Database Systems: A Practical Approach to Design, Implementation and Management. 4th

Edition. Published by Addison Wesley Publishing Company. ISBN 0321210255

4. Dean, J. Ghemawat, S. (2011) LevelDB: A Fast Persistent Key-Value Store Available at: <http://google-opensource.blogspot.co.uk/2011/07/leveldb-fast-persistent-key-value-store.html>
5. Eaglestone, B. (1991) Relational databases. Published Cheltenham : Stanley Thornes. ISBN: 0748711767, 9780748711765
6. Harrington, J. L. (1998) Relational database design clearly explained. Published San Diego ; London : AP Professional 1998. ISBN: 0123264251, 9780123264251
7. Leavitt, N. (2010) Will NoSQL Databases Live Up to Their Promise? Computer. Volume: 43, Issue: 2. Page(s): 12 - 14
8. MDN (2011) IndexedDB development. Available at: https://developer.mozilla.org/en/IndexedDB/Basic_Concepts_Behind_IndexedDB
9. Netcraft (2006) November 2006 web server survey. Available at: http://news.netcraft.com/archives/2006/11/01/november_2006_web_server_survey.html
10. Okman, L., Gal-Oz, N., Gonen, Y., Gudes, E., & Abramov, J. (2011). Security Issues in NoSQL Databases. *2011 IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, 541-547. Ieee. doi:10.1109/TrustCom.2011.70
11. Shashank, T. (2011) Professional NoSQL. Published Indianapolis, IN : Wiley 2011. ISBN: 6613258156, 9786613258151, 9780470942246
12. Stonebraker, M. (2010) SQL Databases v. NoSQL Databases. *Communications of the ACM*. Vol. 53 Issue 4, p10-11, 2p
13. Strozzi, C. (1998) NoSQL A Relational Database Management System. Available at: http://www.strozzi.it/cgi-bin/CSA/tw7/1/en_US/nosql/Home%20Page
14. Thalheim, B. (1998) Entity-Relationship Modeling: Foundations of Database Technology. Published by Springer, New York, USA. ISBN: 3-540-65470-4
15. Tudorica, B.G. Bucur, C. (2011) A comparison between several NoSQL databases with comments and notes. *Roedunet International Conference (RoEduNet), 2011 10th*. Page(s): 1 - 5
16. W3C (2011) IndexedDB transactions. Available at: <http://lists.w3.org/Archives/Public-public-webapps/2011JulSep/0614.html>

Appendix F: Some Potential Issues with the Security of HTML5 IndexedDB

9th IET System Safety and Cyber Security, Manchester, UK

Appendix F: Some Potential Issues with the Security of HTML5 IndexedDB

Stefan Kimak , Jeremy Ellman, Christopher Laing

Northumbria University, Faculty of Engineering and Environment
Newcastle upon Tyne, UK

stefan.kimak@northumbria.ac.uk , jeremy.ellman@northumbria.ac.uk, christopher.laing@northumbria.ac.uk

Keywords: Component web security; IndexedDB, Security, Forensic Test, Encase

Abstract

The new HTML5 standard provides much more access to client resources, such as user location and local data storage etc. Unfortunately, this greater access may create new security risks that potentially can yield new threats to user privacy and web attacks. One of these security risks lies with the HTML5 client-side database. It appears that data stored on the client file system is unencrypted. Therefore, any stored data might be at risk of exposure. This paper explains and performs a security investigation into how the data is stored on client local file systems. The investigation was undertaken using Firefox and Chrome web browsers, and Encase (a computer forensic tool), was used to examine the stored data. This paper describes how the data can be retrieved after an application deletes the client side database. Finally, based on our findings, we propose a solution to correct any potential issues and security risks, and recommend ways to store data securely on local file systems.

1 Introduction

While HTML5 is still in the process of being standardized by the W3C [1], its adoption will greatly help developers resolve recognized problems, such as media and online data handling; thereby providing a more robust method for handling data [20]. Furthermore, the enhanced functionalities of HTML5, such as a client-side database called IndexedDB (which will be embedded within the web browser), will provide additional benefits, such as reducing the web server load. However, while client-side databases have the advantage of reducing load on the web server, their performance will be dependent on the user's web browser; particularly, how the browser implements the new client side database API; otherwise known as the IndexedDB API.

This paper will focus on the security of this new browser-based storage capability, and a series of experiments will show how vulnerable the IndexedDB API is to attacks. These attacks will be described in more detail later, after which we will propose methods of protecting against such attacks. This paper will also investigate how the web application will store the data in the client-side database, and a series of tests will be conducted to retrieve the deleted database files. A possible

solution for storing and retrieving data in secure manner will be proposed and described in further detail.

The testing will use Firefox and Chrome browsers, as they currently support the IndexedDB client-side database. The investigation will focus on the data storage mechanism of the client-side database. To help us analyse the results, a forensic tool called Encase was used; Encase is an industry standard computer forensics tool, used in the majority of criminal cases involving the collection and presentation of digital evidence [5]. Encase is a software to access raw data, and provide the functionality to create disk images.

2 Background

The development of new Web technologies faces a trade-off between stronger security (thereby protecting the user), and increased functionality (thereby helping the user). Unfortunately, this trade-off may have resulted in the development and implementation of an insecure API, namely IndexedDB API. It should be noted that the implementation of the IndexedDB into Web browsers is not yet fully completed; consequently these some of the security risks may no longer exist in future implementations of the IndexedDB API. The security issue with the unencrypted data stored into IndexedDB is considerably a structure flaw. This means that the database is designed to store all of the data in unencrypted state.

2.1 Problem identification

IndexedDB is storing data in unencrypted state. This information might not be sensitive, such as usernames or password, but can include client name, address, place of birth or date of birth.

If some of the information is put together, then this can lead to identity theft.

To prevent leaking of data all over the place, we propose an algorithm to secure this information, and prevent the end user from identity theft.

IndexedDB also works on mobile devices, where the data is stored into internal phone memory. Therefore, the problem also exists on mobile device, which is more serious compared to desktop. The deleted data can be retrieved from any mobile device. As the data is unencrypted, the security issue is much higher. Considering the scenario where the mobile phone is lost or stolen, it will be possible to retrieve the deleted data. This risk of data exposure is much higher in this case.

Appendix F: Some Potential Issues with the Security of HTML5 IndexedDB

Compared to storing data on server, where data is not available to recover after deletion, storing on client side is going to be more insecure.

2.2 IndexedDB Structure

Files and data stored by the browser are retained on the file storage system, on the computer's hard drive. The client-side database, IndexedDB, is a persistent client-side database, consequently the files reside on the user file system and can be recovered until they are overwritten by other files.

IndexedDB treats file data just like any other type of data. An application can write a file (or Blob), into IndexedDB, as well as storing strings, numbers and JavaScript objects [6]. This is detailed in the IndexedDB specifications and, so far, implemented in both the Firefox and Chrome applications of IndexedDB. In Firefox and Chrome's IndexedDB implementation, the files are stored transparently, externally to the actual database; the performance of storing a file in IndexedDB is just as good as storing it in a filesystem. It does not bloat the database and slow down other operations. Moreover, reading from the file means that the implementation reads from an OS file; therefore, it is just as fast as a filesystem.

The Firefox IndexedDB implementation will, if it is storing the same Blob in multiple files, create 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; it writes data faster while using fewer resources.

2.3 Value in Database

Each record has a value, which could include anything that can be expressed in JavaScript, including: Boolean, number, string, date, object, array, regexp, undefined, and null. IndexedDB enables the storage of structured data, and unlike cookies and DOM Storage, IndexedDB provides features that enable to group, iterate, search, and filter JavaScript objects [18]. Each record consists of a key path and a matching value. These can be a simple type, such as string or date, or more advanced, such as JavaScript objects and arrays. It can include indexes for faster retrieval of records and can store large amount of objects.

IndexedDB is a key-value store in the same way as Local storage. However, Local storage just retains strings only key; therefore, the usual approach to local storage is to `JSON.stringify` it. While this is suitable for finding the object with key `uniq`, the only way to retrieve the properties of `myObject` from local storage is to `JSON.parse` the object and examine it. IndexedDB can store data other than strings in the value, including simple types such as `DOMString` and `Date` as well as `Object` and `Array` instances [16].

Furthermore, it can create indexes on object properties containing a specific value. So while IndexedDB can hold the same one-thousand objects, it can also create an index on the `b` property and use that to retrieve only the objects where `b==2` without having to scan every object in the store. Furthermore, IndexedDB is aware of ranges; therefore, it can search and retrieve all records beginning with 'ab' and ending with 'abd' in order to find 'abc' etc.

IndexedDB is implemented differently across browsers. Firefox uses SQLite and Chrome LevelDB. It should be noted that LevelDB is not a SQL database. Like other NoSQL and Dbm stores, it does not have a relational data model, it does not support SQL queries, and it has no support for indexes.

IndexedDB is implemented in the browser on top of another database. This mean that it does not work on its own, as it is an API layer. IndexedDB is storing the value in local filesystem, which means that the limit of storage is limited to space on user hard drive. When compared to other databases, IndexedDB is updating the whole data rather that just the bits of specific data values.

3 Potential attack vector

This section is considering an unauthorized physical access attack to IndexedDB file, from outside the user local machine.

3.1 CORS (Cross-origin resource sharing) Attack

CORS is a mechanism that allows JavaScript on a web page to make XMLHttpRequests to another domain, not the domain the JavaScript originated from. 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 [23]. 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, in some CORS should not be allowed, to protect the privacy of the end user.

Scenario 1: Unauthorized physical access to the OS file system, where the data from the browser database (IndexedDB) is stored unencrypted.

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 cannot 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 [21].

Scenario 2 (Data Breach): Unauthorized access from an external machine (bypassing the Same Origin Policy (SOP)) to read the data and retrieve the information stored in the IndexedDB files.

But why is the ability to read and retrieve data stored in the IndexedDB files such an issue. In order to demonstrate the problem, we will firstly conduct an analysis of the IndexedDB database file.

4 Analysis of indexedDB database file

The first step in conducting this analysis is to build a ‘clean’ Hard Disk Drive (HDD) on a PC [HP Z400 6-DIMM with 12GB RAM and XEON 4 physical cores (8logical cores)], that will include the operating system (Windows 7, 64-bit) and web browsers (Firefox v.20.0.1; Chrome v.29.0.1547.620. after which some initial Internet browsing will be conducted. The HDD will then be ‘acquired’ using Encase v.6.11.1. This will be the starting point for each investigation. After each experiment, the image needs to be restored to a ‘clean’ state, and following each experiment, the disk will be forensically wiped, and then same component (operating system, web browsers) will be installed. The aim of these experiments is to investigate and show how the data is deleted from IndexedDB (local file) and also if the data is in unencrypted state. We will also perform an reuse of recovered file and show, if it can be successfully achieved.

EXPERIMENT 1: RECOVERY OF DELETED INDEXEDDB SQLITE DATABASE FILE

In this experiment, the SQLite database file will be deleted from a Hard Disk Drive (HDD), in a PC [HP Modified to i5 processors and 16GB Ram] running a Windows 7 64-bit Operating System. Then using Encase v.6.11.1, locate the deleted data and perform a data recovery. The structure of the web browsers (Firefox v.20.0.1 and Chrome v.29.0.1547.62) will also be examined to assess how the data is stored.

EXPERIMENT 1: RESULTS

Firefox stores all data in a temporary table (SQLite database) from where the data is copied into an Object Store, complete with key/value link. After the data has been copied successfully, the temporary table is dropped. The browser always stored the SQL file in the same location in the file system. On Firefox the location is C:\Users\[user-name]\Application Data\Mozilla\Firefox\Profiles\[profile name.default]\indexedDB\[domain-name]\[database-name] where on Chrome C:\Users[user-name]\AppData\Local\Google\Chrome\User Data\Default\IndexedDB. Consequently, and previously stored data is always overwritten. Interestingly, when the data is deleted from the application (using delete function), the location within the file system is reserved for that deleted file. It is keeping the reserved location, because the deleted file still persists on the HDD. So when running the application again, the browser always allocates a different location for the newly created Object Store.

Allocation of file storage in Chrome is slightly different; all of the databases are stored in the same file. Consequently, it is assumed that Chrome is using compression for storing browsing data.

In Encase we choose the option for Copy/UnErase the deleted file. This exported the deleted file with all the data. While the deleted file data can be read from Encase, we choose to export the file and opened with SQLite Manager (Figure 1). In this way all the data in table was visible, and the field values in the BLOB could be exported unencrypted.

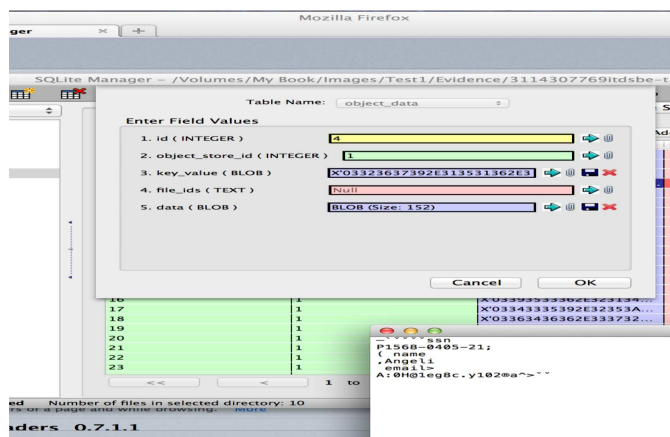


Figure 1: Exported deleted database file.

EXPERIMENT 2: CLEARING THE BROWSER CACHE

Experiments in Firefox will include deleting the data by clearing the browser cache (deleting the offline data option). Each experiment will store 300K records with a file size of

Appendix F: Some Potential Issues with the Security of HTML5 IndexedDB

127MB. Experiments in chrome will include deleting the data by clearing the browser cache (clearing browsing data-Hosted app data). Each experiment will store 300K records with a file size of 128MB.

EXPERIMENT 2: RESULTS

Clearing the browser cache in Chrome clears the databases and deletes the file where it is stored. In Firefox the clearing the cache does not delete the database file from local file system.

EXPERIMENT 3: REUSE OF RECOVERED INDEXEDDB DATABASE

In this experiment the possibility of reusing a recovered IndexedDB database in a different web browser was investigated. This involved identifying the location (physical address on the HDD), of the file after it had been deleted. In addition, this experiment also considered if the database name is changed after it has been deleted; to see if the Web application can read deleted file with different filename. When deleting a database from the application, everything in the folders is deleted; including those that can be stored locally (image, document, video, audio). SQLite is not a typed database, which means that any data type can be put into any cell, regardless of the type declared for the column, and the database will attempt to convert it. Similarly, a different type than the column type is requested, SQLite will also convert this value.

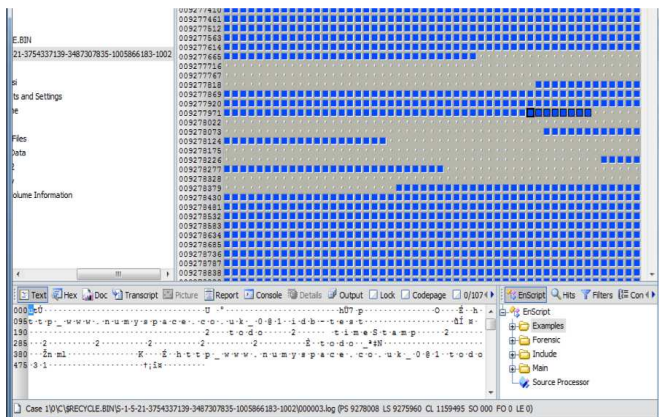


Figure 2: The physical address, and data in database file.

EXPERIMENT 3: RESULTS

Figure 2 displays the physical address of the file before and after deletion. Also the physical address of the newly created database file persists on the same location. The deleted files

are marked with red cross. This file was restored with Encase and exported to another hard drive. The file was restored into database folder, and application was run to check if the data could be accessed. The result is that the application read the file and all of the data in unencrypted state was available for us to see.

5 Analyses and Possible Solution

The results were as expected; the deleted data has been marked as deleted, but it can be exported and all the information inside the database could be read. Moreover, exported data that has been imported to another PC running Windows 7 can be accessed and re-used. However a possible solution to this security issue is presented below.

5.1 A Proposed Solution to Security issue in IndexedDB

In this section we are going to propose a solution to IndexedDB storage security issue. The prevention against such scenarios might include encryption of the files stored by the browser on the file system. All the data stored by the browser will be encrypted and stored to the file system. When retrieving the data, a secure key will be required to read the data from the file system. An encryption library will generate this key, which will permit access to read the data. Otherwise, the data remains encrypted and impossible to read. The encryption key will be downloaded dynamically and the key (password) will be stored in session key. When the key is secure, then it will encrypt data. When a user closes browser, then the key is overwritten in RAM. This will help to prevent attacker getting access to secure key when reading data from RAM.

The algorithm to secure saving of data could be a JavaScript library (proposed Stanford JS Encryption library), which will help us to prevent saving data in unencrypted state. We going to explain steps to write, update and read the data with algorithm in pseudo code.

The following steps are described writing or updating data to database.

1. Ensure we have established the secure connection through OAuth - The first step is to provide a secure login functionality, which can be provided by web application. The web application will use the login to authenticate a user and securely logged the user into system.

Appendix F: Some Potential Issues with the Security of HTML5 IndexedDB

2. Open a connection to database - When an application requests a new transaction for IndexedDB to open 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 not readable to others.
3. Encryption library generates public and private key - When the data is encrypted a key will be generated and stored with the user information on server. Client-side encrypts sensitive data using the public key, which will be generated and stored on sever side. This public key is used when encrypting information using the JavaScript library.
Public and private key are created simultaneously using the same algorithm (RSA- Rivest-Shamir-Adleman).
4. Encrypt data - When Client-side encryption is enabled, an RSA keypair is generated and 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 [15]. The private key, however, is never revealed to user or anyone else. Once the data servers, the data is decrypted using the keypair's private key [16]. Private key is used to decrypt text that has been encrypted with public key. It uses the industry-standard AES algorithm at 128, 192 or 256 bits; the SHA256 hash function; the HMAC authentication code.
5. Save the file and close the connection

When reading the data, the following steps needs to be fulfilled.

1. Checking user credentials - When the user request the read the data from database, the web application will check user credentials (if the session is active) and get the key from server to allow decryption of data.
2. Get the key to decrypt data - Upon successful authentication user will be given a public key, which will be used for decryption of the data. This private key will be stored on 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 private key to server.
3. Decrypt Data – Encryption library will check for matching combination of private and public key, and perform decryption of data.
4. Show the decrypted data to user
5. Close the connection

For a secure authentication with server we are going to consider OAuth. This will provide authentication between the application and web server using a security token. We do

not consider security issues with OAuth, because this will be done in later stage, when the implementation is done.

The data is stored unencrypted to file system, which can be accessed by the web application. When an application send a request to web browser to store the data on local file system, the cryptography library will encrypt the data, to be stored secure. A secure key will be also generated and stored on web server. Reading the data from local file system will be possible only when a secure key is provided and the authentication between web application and server is established. Considering all of the points are made and connection is securely established, the data is decrypted by cryptography library and displayed trough web browser to user. In fig. 3 we highlight the proposed solution showing how the cryptography library will be implemented. The library will be implemented on top of web browser API. It should be noted, that at this stage this solution is only theoretical, however further work will be undertaken to prove this hypothesis.

The algorithm will consist of the following components, which are build into browser (Figure 4).

- Mechanism for generating private and public key
- Mechanism for checking the combination of keys
 - Encryption
 - Decryption

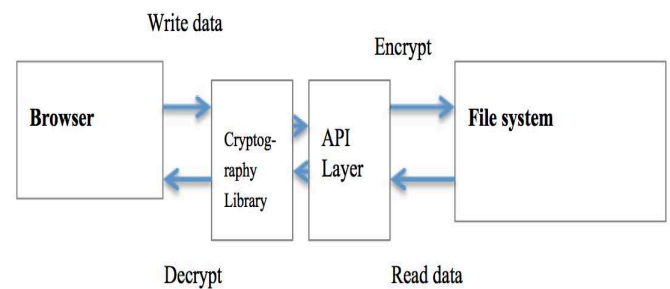


Figure 3: Proposed Encryption Library

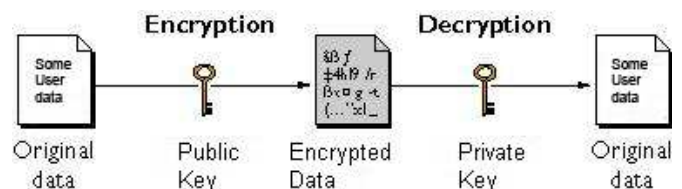


Figure 4: Encryption and decryption using keys

Appendix F: Some Potential Issues with the Security of HTML5 IndexedDB

Encryption/Cryptography library is a piece of software or code, which encrypts readable text into unreadable data. This data can be accessed by using an encryption key. Some examples of encryption libraries are listed below. These are just few encryption libraries, which are considered for implementation into browser. These libraries were chosen, because they provide the functionality to encrypt on client side, and also are available as open source.

OpenSSL: Open Source toolkit implementing the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) protocols as well as a full-strength general purpose cryptography library (Engelschall, 1999).

BeeCrypt: A C++ API cryptography library (Doxygen, 2009).

Crypto++: A free C++ library for cryptography: includes ciphers, message authentication codes, one-way hash functions, public-key cryptosystems and key agreement schemes (Dai, 2004).

GPGME: (GnuPG Made Easy) a C language library that allows support for cryptography to be added to a programme. It is designed to provide easier access to public key crypto engines like GnuPG or GpgSM. GPGME provides a high-level crypto API for encryption, decryption, signing, signature verification and key management (Koch, 1999)

Libgcrypt: GNU's basic cryptographic library (Werner, 2007).

Of course, another possible solution to the problem might include usage of an external device to store data from a browser. For example, a user could specify a location to which any IndexedDB files should be stored when browsing the web or running some applications. This includes an option where the data could be written and read from an external source, such as USB. The USB key will need to be secured with access encryption and restricted to access data when the master password is entered.

6 Conclusion

In this paper, we have demonstrated security related flaws within IndexedDB. While the browser can delete IndexedDB files stored on the local filesystem, they can be retrieved by Encase. Unfortunately, the retrieved data is in an unencrypted format, and given the nature of the data held within the IndexedDB API, a potential security issue exists. All the data in IndexedDB is exposed. We have demonstrated a solution for this security issue, which includes a secure 'library', located between the browser and the filesystem. All data stored by the Indexed DB application will be encrypted

and saved to the library. Therefore, if the application needs to read the data, an encryption key will be required. Without a key, the data will not be decrypted and reading the data will not be possible. This will help to secure the data stored on the client side and prevent retrieval in an unencrypted state. Future work will focus on implementing the cryptography library into web browser and testing for possible attacks.

References

- [1] Berjon, R. (2014) W3C HTML5 Specification. Available at: <http://www.w3.org/html/wg/drafts/html/master/>
- [2] Bunting, S. Wei, W. (2006) Encase computer forensics: the official EnCE EnCase certified examiner study guide. Published San Francisco, Calif.: SYBEX / Wiley 2006
- [3] Dai, W. (2004) Crypto++ Library. Available at: <http://www.cryptopp.com> Last Accessed :20 May 2014
- [4] Doxygen (2009) BeeCrypt C++ API Documentation. Available at: <http://beecrypt.sourceforge.net/doxygen/c++/index.html> Last Accessed :20 May 2014
- [5] Engelschall, R. S. (1999). About the OpenSSL Project Available at: <https://www.openssl.org/about/>
- [6] Encase (2004) EnCase Forensic Edition User Manual. v.4 Available at: <http://www.guidancesoftware.com/> Accessed on: 5th January 2014
- [7] Flanagan, D. (2011) JavaScript: The Definitive Guide Activate Your Web Pages. 6th edition, Publisher: O'Reilly
- [8] Koch, W. (1999) GPGME – The GNU Privacy Guard. Available at: <https://www.gnupg.org/index.html> Last Accessed : 20 May 2014
- [9] Koch, W. (2003) Libgcrypt. Available at: <http://www.gnu.org/software/libgcrypt/> Last Accessed : 20 May 2014
- [10] Mehta, N. Sicking, J. Graff, E. Popescu, A. Orlow, J. (2012) Indexed Database API Available at: <http://www.w3.org/TR/IndexedDB/> Accessed on: 5th January 2014
- [11] MSDN (2012) IndexedDB. Available at: [http://msdn.microsoft.com/en-us/library/ie/hh673548\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/ie/hh673548(v=vs.85).aspx). Accessed on: 5th January 2014
- [12] Sarris, S. (2013) HTML5 Unleashed. Published by: Sams. **Print ISBN-10:** 0-672-33627-8
- [13] Stuttard, D. Pinco, M. (2007) Web Application Hacker's Handbook. Published by: Wiley Publishing, Indianapolis, Indiana
- [14] Zakas, C. N. (2010) Cross-domain Ajax with Cross-Origin Resource Sharing". NCZOnline. Available at: <http://www.nczonline.net/blog/2010/05/25/cross-domain-ajax-with-cross-origin-resource-sharing>. Last Accessed: 20 February 2014
- [15] Bennett, S. Paine, S. (2001) RSA Security's Official Guide to Cryptography. Osborne/McGraw-Hill
- [16] Mollin, R. (2003) RSA and Public-Key Cryptography. CHAPMAN & HALL/CRC A CRC Press Company ISBN 1-58488-338-3

Appendix G: The role of HTML5 IndexedDB, the past, present and future

**10th International Conference for Internet Technology and Secured Transactions
(ICITST-2015), London, UK**

Appendix G: The role of HTML5 IndexedDB, the past, present and future

Stefan Kimak

Faculty of Engineering and Environment
Northumbria University
Newcastle Upon Tyne, UK
stefan.kimak@northumbria.ac.uk

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.

Keywords—component; Cookies; HTML5; IndexedDB

XXII. 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.

XXIII. INDEXEDDB – THE PAST

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.

A. 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 retail 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].

B. 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.

C. 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 IndexedDB.

XXIV. INDEXEDDB - PRESENT

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 (Navigator), 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 IndexedDB, which was most influenced by Oracle's Berkeley 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

Appendix G: The role of HTML5 IndexedDB, the past, present and future

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.

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.

XXV. INDEXEDDB - FUTURE

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 client side encryption, which would mean that browser based storage is at least as secure as that on the server.

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.

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. On top of the encryption library a multifactor authentication (MFA) or two factors authentication (2FA) has been implemented.

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.

XXVI. 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.

Our future work includes more details of the security mechanism in [21]. 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.

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*

Appendix G: The role of HTML5 IndexedDB, the past, present and future

2014 (IET Conference), 14-16th October 2014, The Midland Hotel, Manchester, UK.

- [20] Casario, M. Elst, P. Brown, Ch. Wormser, N. Hanguéz, C. (2011) HTML5 Solutions: Essential Techniques for HTML5 Developers. Publisher: FRIENDS OF ED; 1 edition ISBN: 1430233869
- [21] Kimak S, forthcoming, 2016 'Phd Thesis' Northumbria University

Appendix H: HTML5 IndexedDB encryption: Prevention against potential attacks

International Journal of Intelligent Computing Research (IJICR), Volume 6, Issue 4, 2015

Appendix H: HTML5 IndexedDB encryption: Prevention against potential attacks

Stefan Kimak

*Faculty of Engineering and Environment
Northumbria University
Newcastle Upon Tyne, UK
stefan.kimak@northumbria.ac.uk*

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 IndexedDB.

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 reopened, 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 (Navigator), 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 IndexedDB, which was most influenced by Oracle's Berkeley 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:

[21] 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.

[21] 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.

[21] 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.

[21] 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.

[21] 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 through XSS attacks. First we will perform an 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/tw7/1/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.

Appendix H: HTML5 IndexedDB encryption: Prevention against potential attacks

- [20] Casario, M. Elst, P. Brown, Ch. Wormser, N. Hanguéz, C. (2011) HTML5 Solutions: Essential Techniques for HTML5 Developers. Publisher: FRIENDS OF ED; 1 edition ISBN: 14302338

