Volume 3, Issue 3, March 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Review Study on Security Threat on HTML 5

Mr. Hemant Y. Kharbade ,
ME(CSE) 2nd Semester
G.H Raisoni College of Engineering,Amravati
India.

Mr. Avinash wadhe
M-Tech(CSE)
G.H Raisoni College of Engineering,Amravati
India.

Abstract- over the past 20 years web browsers have changed considerably from being a simple text display to now supporting complex multimedia applications [1]. 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 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 PC. 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. HTML5 standard in the light of how this feature is going to affect the privacy of its users. As a result one can see the need for additional standards for achieving better web privacy.

Keywords-component web security; HTML5; IndexedDB.

I. 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" [2]. Consequently HTML5 opens up entirely new security challenges and loopholes [3]. 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 presented.

II. Background

Hyper Text Markup Language (HTML) is the main programming language for web pages. Since it arrived in 1990 [4] the versions have evolved to allow web applications to act as desktop applications [5]. 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 [6].
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 [7].
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. 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 [8]. 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 [9].
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 [10]. 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 [1]. This will speed up application load time [11].

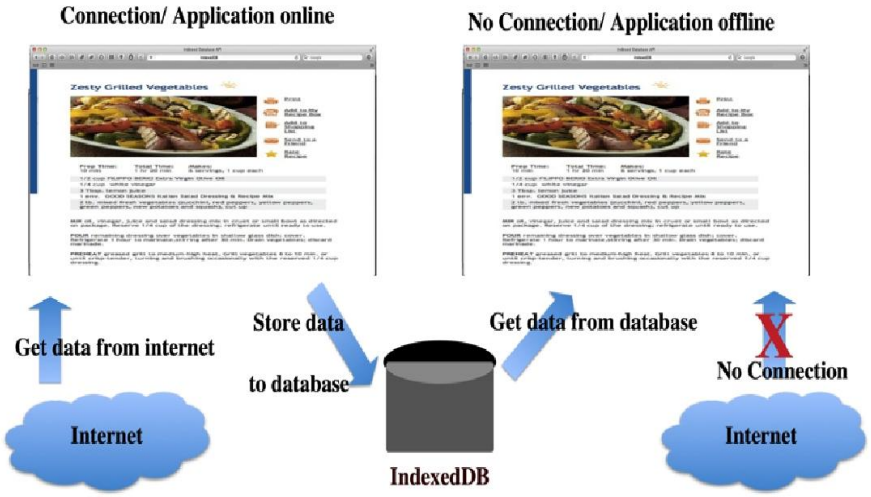

Figure 1. HTML5's IndexedDB functionality

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 [17].

The upcoming HTML5 standard [1] adds many new APIs and features allowing developers to create web applications responding to the today's needs. The new features include for example APIs for location-based services, cross-document messaging and local storage; each of which used abusively might lead to exposing personal information.

III Literature Survey

Stefan Kimak , Dr. Jeremy Ellman, Dr. Christopher Laing [1]
An investigation into possible attacks on HTML5 IndexedDB and their prevention
Harri Hämäläinen Aalto University, Department of Media Technology[2] How HTML5 Affects the Web Privacy

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. 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 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. Anyways an IndexedDB is a 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 (15). 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 [17]. Part of a local storage is indexedDB [18].

D. Traditional web attacks

The purpose of this section is to summarize the Cross-site request forgery (CSRF) and the Cross-site scripting (XSS) vulnerabilities, which are often exploited in web applications.A vulnerability is called CSRF vulnerability if it allows an attacker to trick the user agent to perform an action on trusted site [3]. Because the action itself is performed by the user agent it has the same context to trusted site as any request to that site. Commonly used protection mechanism against CSRF attacks is to add a special synchronizing token into forms to be submitted. A web server receiving POST request with the token present can then assume, with a high probability, that the submission was actually intended. The problem with this approach is that, due to the nature of synchronizing tokens, one token must only be used once. This effectively breaks the user agent's back button functionality in scenarios where user has submitted the form but wants to
return and modify the form before resubmitting the form. Other solutions are also used like checking the referrer from request headers, which is not a completely working solution as referrer header is not usually sent when using HTTPS. Also some user agents are offering a way to hide the referrer headers because of the privacy concerns. With Cross-site scripting (XSS) an attacker can inject malicious script content into the trusted site [4]. XSS vulnerabilities are serious security holes because these allows an attacker to run malicious scripts on trusted site which can leak privateinformation or access and modify the DOM of the trusted site among the other things. Accessing the DOM of the trusted site can then again be used to defeat any synchronizing token based CSRF protection mechanisms. The primary protection mechanism against XSS attacks is to apply an input filtering for any untrusted data entering the system. The problem is that there are multiple ways to mount XSS attacks which are not always well known by the application developers [5].

E. Click-jacking

The term click-jacking [6] refers to an attack scenario where a malicious content on a web page renders a with some user interface elements trying to get the user to act with these elements. However the malicious content also loads an iframe with the targeted site of attack on top of the rendered page and by using Cascading Style Sheet (CSS) features sets this top page to be transparent. While the user, who cannot see the top page has no idea of its existence, tries to interact with the page controlled by the attacker she is actually interacting with the attacked site. As the outcome of click-jacking attack the user can be tricked to perform unwanted actions on the targeted trusted site. The famous example of click-jacking attack is the "don't click" attack [7], where by clicking a harmless looking button on attacker's site causes the user to post a Twitter 1 update in case she has an active Twitter session. To protect against click-jacking attacks several sites are trying to detect when it is loaded into iframe and then redirect to the original site. Several click-jacking protection mechanisms, and the click-jacking problem in general, are studied in depth by
Rydstedt et.al. [8].

F .Third-party cookies and user tracking

HTTP Cookies [9] are primarily used to save the web application state on user agent. The state is saved in special file called HTTP Cookie. When an origin sets a cookies it's sent back to this origin whenever the user agent visits the same origin again, allowing
server side application to restore the state for user agent. The controlled state also allows more extensive user tracking as the server can assign a cookie with an unique session ID to every new request without a cookie with session ID present. Later on any request from client who already has received its session ID is supposed to sent this information back to the server. The user tracking becomes even more problematic when the discussion turns to so called third-party cookies. As we know web documents might include resources from an origin of different author. Each of these are retrieved with separated HTTP requests allowing also these to set their own cookies, which are called third-party cookies as these were

not sent from the origin of the original document. If we think some author who is able to set some content like ads into multiple popular sites this author is going to be able to track the behavior and usage trends of users in web. Usually browsers allow user to disable these third-party cookies while by default the option is often enabled.

IV. 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 Cross Site scripting (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. 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), cross-site scripting (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 Cross Site Scripting (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.

V. 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 EAS or 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].

VI. 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.

VII. 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 todays 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.

References

[1] Nichols, V. (2010) Will HTML5 restandardize the Web? Computer. Volume: 43, Issue: 4. P. 13-15

[2] Ryck, P. Desmet, L. Philippaerts, P. Piessens, F. (2011) A Security Analysis of Next Generation Web Standards, (ENISA). Tech. Rep.

[3] 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

[4] Sefton, P. (2009) Towards Scholarly HTML. Serials Review. Volume 45, Issue 3.p.154-158

[5] 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

[6] Mitchell, E. (2010) Standards, Efficiency, and the Evolution of Design. Journal of Web Libibrarianship. Volume 4, Issue 4

[7] Clark, J. (2010) HTML5. Online Journal. CINAHL database. ISSN:0146-5422 .Volume 34 Issue 6, p12

[8] 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

[9] 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

[10] Taivalseeri, A. Mikkonen, T. (2011) The Web as an Application Platform: The Saga Continues. Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference.

[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. Hanguez,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] Tappenden, A. (2008) A Three-Tiered Testing Strategy for Cookies. Software Testing, Verification, and Validation, 2008 1st International Conference. p: 131 – 140

