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

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

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

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

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

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

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

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

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

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

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

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

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

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

4. Potential attacks

4.1. CORS

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

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

4.2. XSS

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

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

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

4.3. Social engineering attacks

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