Survey on Web Browser and Their Extensions

PALAK JAIN[1]
[1]*CSE, Maulana Azad National Institute of Technology, Bhopal*

Abstract-- The Web today has become the most useful and popular platform for application development. In the beginnings of the Web, applications provided users just the ability to browse and read content. The expansion and adoption of new web technologies has led to a significant increase in development and, more importantly, usage of web applications that allow users to create their own content and impact their life. Almost every Internet user uses a web browser to access any content on the Internet. Each web application is designed and developed to be executed inside the web browser. Browser extensions are remarkably popular, with one in three users running at least one extension. Browser extensions allow for customization of the browser by adding functionality. The way these extensions are integrated strongly differs in the four major browsers i.e. Google Chrome, Mozilla Firefox, Apple Safari, and Internet Explorer. In this paper, there is an analysis of popular web browsers and their architecture. It also includes analysis of different browser extensions and their architecture with their security.

I. INTRODUCTION

Online banking, social networking and information retrieval are some of the terms which are confronted us every day. During the last years, several services of the Internet have become ubiquitous. Web browsers are the platform which allows users to browse web pages and other resources such as images, sound files, videos on the Internet. Web browsers are the intermediary applications between a user and the web server. Understanding of browser vulnerability requires the knowledge of architectural design of browsers. Basically a browser is a software or software application program which is used for retrieving or extracting information resources on World Wide Web. It consists of three main parts: i) controller ii) client program and iii) interpreter.

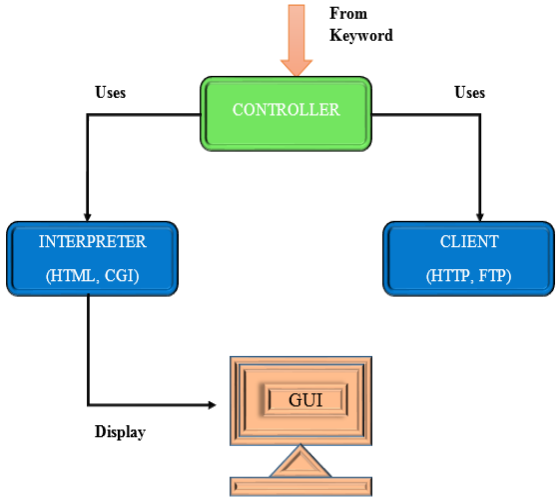

Figure 1: Functionality of Web Browser

The controller handles the other two parts i.e. interpreter and client program. A controller takes inputs from the standard input devices and uses a client program (http, ftp, telnet etc.) to access any document. As soon as the document is accessed, controller makes use of an interpreter (html, cgi or java etc.) to display it on the screen. Hence, it acts as an interface between a user and the World Wide Web. The major use case for web browsers is displaying web pages by rendering markup language content. When speaking about the Internet this markup language is the Hyper Text Markup Language (HTML) which is an international standard of W3C. Web Browser is a software application that resides on a computer and is used to display and locate web pages. Web user's access information from web servers by using a client program called Browser. A web browser is a software application for traversing, presenting, and retrieving information resources on the World Wide Web. The information resources are identified by using Uniform Resource Identifier (URI/URL) and may be a web page, image, video or another piece of content.

Browser Extensions (often called plug-ins or add-ons) are small pieces of code that let developers add additional functionality to the browser.

Offline content is stored in your computer in the form of web pages and to access them we need a browser to view it. The purpose of the browser is to take information which is sent to the computer and present it in a readable and useable format. The browser also connects you to different websites (web servers) and they reply with the requested information. If you are going to use the internet, firstly you have a browser on your desktop and if there is no browser then every operating system must have one built-in browser into it. The Internet Explorer for Windows OS, Mozilla Firefox for Linux and if you want Google Chrome then you need to install it.

II. CLASSIFICATION

The three varieties of classification browser are:

1. Standard Browser: The Standard browser displays the classification which schedules the links to other areas of classification and to the classification tables. When you click on a table link in the Standard browser, the data on the screen is replaced with the contents of the table. All classification number calculations and additions must be performed manually.

2. Enhanced Browser: The Enhanced browser is used to add support for a calculator that automatically merges classification table data into the main classification display. When you click on a table link in the Enhanced browser, the table is loaded in such a way that the table data seems to be a part of the schedules data. To avoid cluttering the screen, the Enhanced browser will not apply a table and it displays its content unless we click on a particular table link. Once we click on the link, the table will be merged into the current display with calculated classification numbers. The one time Class Web will automatically apply tables when you enter a classification number into the LC Class prompt at the top of the screen. In that case, any appropriate tables will be applied to get you to the most specific record possible for a given class number.

3. Hierarchy Browser: The Hierarchy browser is a browser which automatically takes table links as we move around in the data. We navigate hierarchically by moving down one level at a time and by moving up one or more levels at once. Like the Enhanced browser, when we typed a number into the LC Class prompt at the top of the screen, then the software will apply as many tables as necessary to get the most specific possible record. The Hierarchy browser keeps track of the current level and limits the display to the level of classified data. By clicking on a link that says Hide subtopics, Show subtopics or Apply table, through this we can change the maximum depth that the browser will go when creating the display.

III. BASIC ARCHITECTURE OF WEB BROWSER

1. User Interface: It is the space where interaction between users and the browser happens. Most of the browsers have common inputs for a user interface. Some of them are - an address bar, next and back buttons, and buttons for home, refresh and stop, choices to bookmark web pages, and so forth.

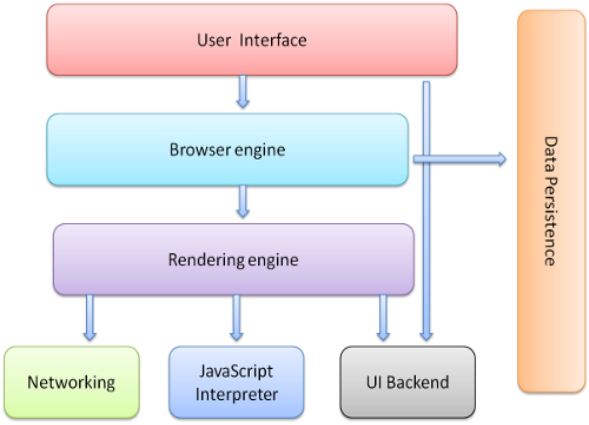

Figure 2: Architecture of Web Browser

2. Browser Engine: It is a bridge between a user interface and rendering engine. It is in charge of querying and manipulating the rendering engine according to the inputs from various user interfaces.
3. Rendering Engine: It is able to render the content available for the web representation of resources. It will start parsing the HTML document and turn the tags to DOM nodes in a tree called Content tree. It will also parse the style data both in external CSS files and in style elements.
4. Network: The fraction of the code written in the browser, responsible for sending various network calls. For example sending the HTTP requests to the server.
5. Java Script Interpreter: It is the part of the browser written to interpret the JavaScript code exhibited in a web page.
6. Display Backend: This draws basic widgets on the browser like combo boxes, windows, etc.
7. Data Persistence: It is the small database created on the local drive of the computer where the browser is installed. This database stores various files like cache, cookies, etc.

IV. FUNCTIONALITY OF WEB BROWSER

The ways in which users interact with browsers nowadays require some functionality features which modern browsers must provide. All of these features target on making surfing the Internet more convenient for users. Some of them are visible in which the features influence the way users can interact with the browser, while others stay invisible to users running in the background of the browser.

When we browse into our browser's address bar some communication starts between Browser and Server.

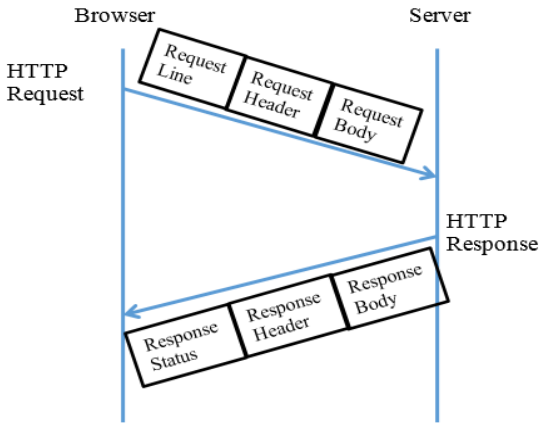

Figure 3: Functionality of Web Browser

- HTTP request contains 3 parts:
  - Request Line: It includes command, web page request, and HTTP version number.
  - Request Header: It includes browser in use, data and some other information.
  - Request Body (Optional): It contains information that was sent to the server.

- HTTP response contains 3 parts:
  - Response Status: It includes HTTP version number, status code and reason phrase i.e. description of status code.
  - Response Header: It includes optional information including server being used, date, URL of web page.
  - Response body: In includes the website (in HTML).

V. DETAILED STUDY ON WEB BROWSERS

Mainly we use four browsers i.e. Google Chrome, Mozilla Firefox Internet Explorer and Apple Safari.

1. Google Chrome: It is a web browser developed by Google that uses the WebKit layout engine and application framework. It was first released as beta version for Microsoft Windows on 2 September 2008, and the public stable release was on 11 December 2008. The latest stable major version is Goggle Chrome 15. Being

available for Windows, Mac, and Linux, Google Chrome is the third most widely used web browser having a usage share of 20.9% [23]. Chrome ships with the open-source WebKit layout engine. HTML5 is also supported by this browser.

Private browsing is possible in Incognito mode. After leaving Incognito mode, Chrome deletes cookies and undoes changes to the browsing history and download history made in private browsing mode. Changes made to the bookmarks and general settings are persistent.

Architecture of Google Chrome:[8]

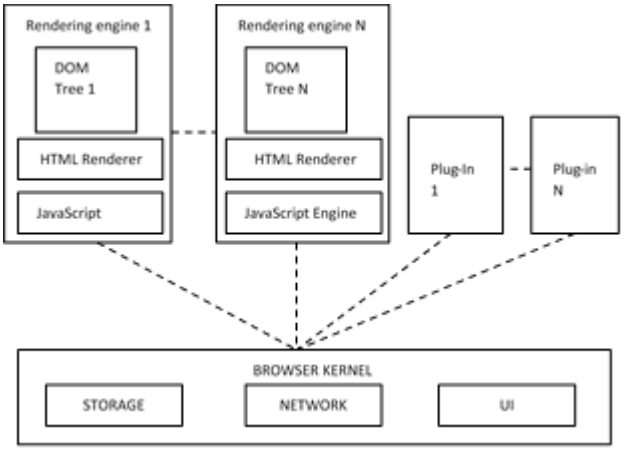

Figure 3: Architecture of Google Chrome Browser

Rendering engine: It converts HTTP responses into rendered bitmaps, browser kernel interacts with the OS, while the plug-ins module is responsible for execution of each plug-in. The rendering engine runs in a sandbox with restricted privileges, deprived of access to the OS resources. Each isolated web program in the browser is assigned to its own rendering engine. The rendering engine is responsible for parsing web content, creating DOM tree representation in the memory, manipulating the DOM tree while executing script instructions. Also, the rendering engine enforces SOP and interacts directly with untrusted web content.

Browser kernel: It runs with full user privileges on behalf of the user. It manages each instance of the rendering engine and implements browser kernel APIs. The browser kernel is responsible for storage management because such an activity requires file system access. By accessing the storage, browser kernel comes in contact with sensitive data, like cookies, bookmarks, passwords, etc. Furthermore, the browser kernel is responsible for executing network operations, e.g. downloading images sending them to a rendering engine for decoding. Also, browser kernel interacts with the OS, handles user inputs and forwards them to the rendering engine assigned to the focused window.

Plug-ins: It runs in their own process, independent from the rendering engine and browser kernel. Web compatibility requires plug-ins to run outside the sandbox, because plug-ins may require access to a microphone, web cam or local file system. Thus, plug-ins cannot be placed inside the rendering engine since rendering engine runs in a sandbox. Plug-ins could be placed within the browser kernel, but in this case, a crash in plug-ins would take down the entire browser.

2. Mozilla Firefox: It is a free and open source Web Browser developed for Windows, OS X and Linux with a mobile version for Android, by the Mozilla Foundation and its subsidiary, the Mozilla Corporation. Firefox uses the Gecko layout engine to render web pages and the Jager Monkey JavaScript engine, which implements current and anticipated web standards. Firefox does not provide a feature to display the users' most popular websites like the other browsers do. Instead of that, Mozilla's browser provides a feature called Panorama that allows users to organize open tabs by grouping them. Firefox also provides auto-completion for online forms and an intelligent address bar called Awesome Bar. The Awesome Bar looks for possible matches to user requests in the browsing history, bookmarks, and the opened tabs.

A mode for private browsing is available. Mozilla simply calls it Private Browsing. Private Browsing prevents data from being stored on the user's machine while activated. Nevertheless, bookmarks and downloaded files will not be deleted after leaving the private browsing mode.

Architecture of Mozilla Firefox:[7]

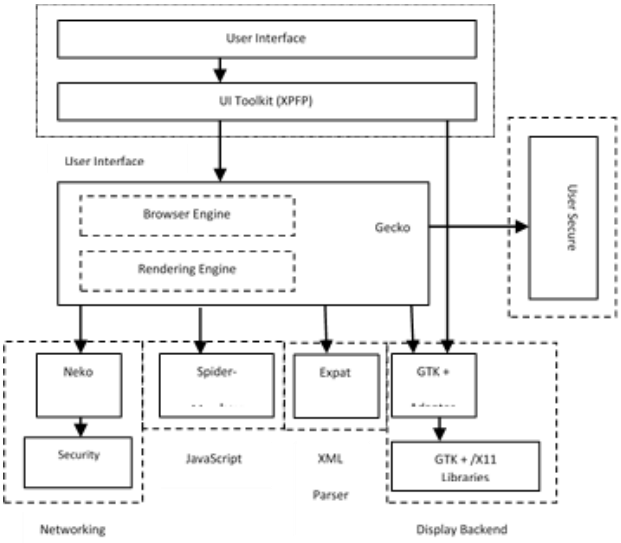

Figure 4: Architecture of Mozilla Firefox Browser

- User Interface: The User Interface layer is the upper layer of the browser which gives setting up the configuration of the browser, handling the visualization of the web pages, web page bookmark and saving options. The User Interface consists of two sub-layers User Interface and Cross Platform Front End (XPFE).XPFE is a development tool based on XML and allows to develop different Mozilla application such as Firefox, Thunderbird. Most parts of Mozilla Firefox is written in XUL (XML User interface Language), HTML and CSS.

- Gecko: Gecko consists of a browser engine and rendering engine. The browser engine goes about as a high level interface to the rendering engine, provides different browser action like Back, Forward, Reload and Stop along with a different error message.

- HTML Parser: It parses the HTML document and generates the layout for web pages.

- XML Parser: It parses the XML document which is responsible for displaying in the user interface.

- Content Model: It arranges parsed web page data based on Document Object Model.

- JavaScript Interpreter: This component executes the JavaScript code embedded in a webpage. It includes Spider Monkey which is a C implementation of JavaScript. In Mozilla Firefox JavaScript interpreter is strongly included in Gecko.

- Data persistence: The Data Persistence component manages user data in a persistent and secure manner.

3. Internet Explorer: Internet Explorer is a standout amongst the most generally utilized web browsers, attaining a peak of 95% during 2002 and 2003. Its usage share has since declined with the launch of Firefox and Chrome. Its latest stable version, Windows Internet Explorer 9, was released in March 2011 and is available for Windows operating systems only. The MSHTML layout engine and the Chakra JavaScript engine form the backbone of the Windows Internet Explorer. When opening a new tab Internet Explorer displays the most popular websites of the user on that page. The features for auto-completion, the intelligent address bar and the integrated search tool are incorporated into a combined search and address bar. The auto completion mechanism is called AutoComplete and is responsible for auto-completion of the address bar, Internet form fields, as well as usernames and passwords. Internet Explorer's private browsing mode is called InPrivate. InPrivate mode focuses on the threat model of a local attacker and thus only prevents browsing data from being stored locally. When enabling InPrivate browsing, Internet Explorer opens a new browser window for which private browsing is enabled. Such windows can be recognized by the “InPrivate" label in the address bar.

Architecture of Internet Explorer:[7]

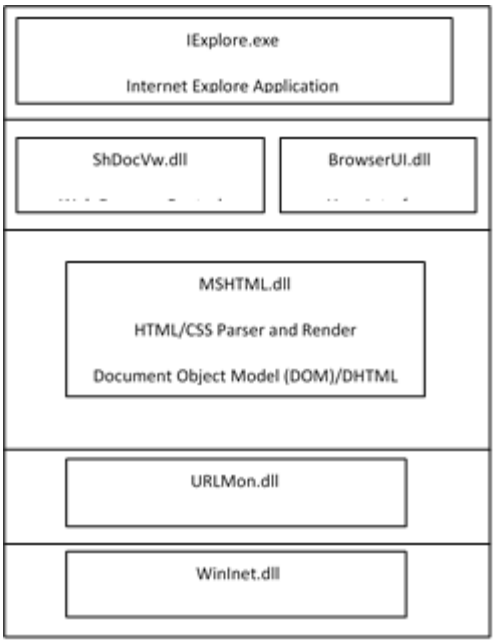

Figure 5: Architecture of Internet Explorer Browser

- IExplore.exe: It is a small component that is reliant on the other main components of IE. The main job of this component is rendering, navigation, protocol implementation, and so on.

- BrowseUI.dll: This is referred as the “chrome" and provides the user interface to IE. It includes the IE address bar, status bar, menus, and so on.

- ShDocVW.dll: It is a core component of IE and is of 32bit, protected by the OS. Since IE is integrated with Windows OS, ActiveX Control interfaces are hosted by this dll. It provides navigation and history. Microsoft Excel, Microsoft Word, Microsoft Visio, and many non-Microsoft applications also expose active document interfaces so that they can be hosted by it.

- MSHTML.dll: It takes care of HTML and Cascading Style Sheets (CSS) parsing i.e., it is responsible for rendering web pages. It is also 32bit dll. MSHTML.dll exposes interfaces to host, as an active document. MSHTML.dll may be called upon to host other components depending on the HTML document's content.

- UrlMon.dll: It provides functionality for MIME handling and code download.

- WinInet.dll: Windows Internet Protocol handler. It implements the HTTP and FTP protocols along with cache management.

4. Apple Safari: It is a web browser developed by Apple Inc. and included with the OS and iOS operating systems. Apple's Safari web browser was first released exclusively for Mac OS users in January 2001. In June 2007 the first version for Windows was released. Apple's browser is the fourth most widely used web browser. Safari employs the WebKit layout engine and the Nitro JavaScript engine. A feature called Top Sites displays the most popular sites of a user. This feature also allows to browse the browsing history in the cover flow design known from iTunes. Safari provides an auto-completion function that automatically fills incomplete webforms. This functionality is called AutoFill and can be configured in the preferences. Furthermore, there is an intelligent address bar that tries to find matches to user requests in the bookmarks and the browsing history. Safari private browsing provides the option to manually clear the browsing history and other data that websites could use to track users. Privacy protection also includes blocking third-party tracking cookies by default.

Architecture of Safari:

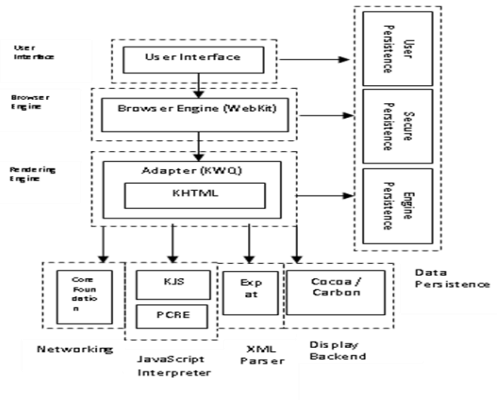

Figure 6: Architecture of Safari Browser

- Rendering Engine: It is composed of the KHTML core engine wrapped in the KWQ adapter. KWQ is written in objective C++, allowing it to present an objective C API to KHTML, which is written in C++. This was needed for integrating Safari into OS.

- XML Parser: It is provided by the Expat XML parser, used in place of the XML Parser provided by the Qt toolkit.

- Display Backend: It is composed of two libraries: Carbon and Cocoa. Carbon provides a lower-level C API for display routines, while Cocoa provides a higher-level Objective C API.

- Data Persistence: It is handled by three separated system that are built into OS: Keychains, Preferences, and Caches. The use of these services allows Safari to integrate smoothly with other OS applications.

VI. VULNERABILITIES IN WEB BROWSER

Vulnerability is the weakness or design flaw of a software program that can be used by an attacker to reduce the system performance or to get the unauthorized access by exploiting it.

1. Cross-site Scripting Vulnerability: Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. The main cause of cross site scripting vulnerability is dynamic web pages because pages are generated by the web server, it is up to the client browser to interpret the page. If it is a static web page it will not be an easy job for the attacker to inject something malicious in the page because the server will have the full control over how the client browser will interpret it. But in case of dynamic pages server does not have full control over it. So, it leaves behind an opportunity for the attacker to inject some malicious code which can be detected neither by the server nor by the client browser interpreter.

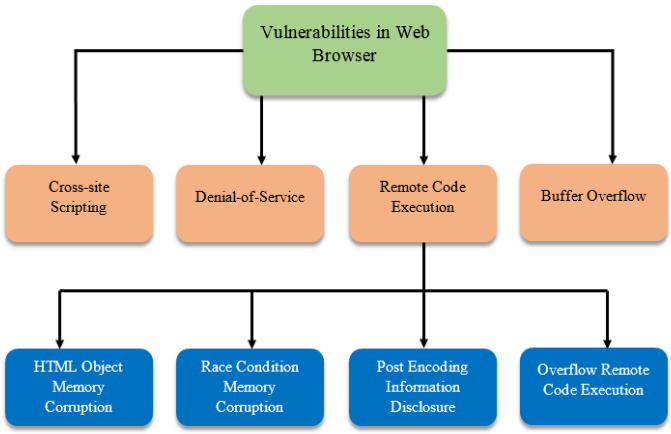

Figure 7: Hierarchy diagram of vulnerabilities

2. Denial of Service Vulnerability: Denial-of-Service attack is a cyber-attack where the committer seeks to make a machine or network resource unavailable to its intended users by temporarily or indefinitely disrupting services of a host connected to the Internet. The main cause of DoS vulnerability in web browsers is infinite looping in JavaScript. And as there is no limitation on windows a JavaScript can open on the monitor. Taking advantage of this feature, a hacker can inject malicious code to open the window repeatedly. It creates a DOS attack on the victim machine. This attack prevents legitimate users from accessing information from a server or from some other machine.

3. Buffer Overflow Vulnerability: Buffer overflow vulnerability occurs due to boundary checking error. If the buffer takes the user supplied input which is greater than the buffer size, there will be a buffer overflow vulnerability. In IE this bug takes advantage of the way it handles long string written in JavaScript code. As a result the browser crashes, potentially compromising malicious code.

4. Remote Code Execution or Memory Corruption Vulnerability: Remote code execution is a security vulnerability that allows an attacker to execute codes from a remote server. Most of the

browsers are vulnerable to remote code execution and memory corruption. Some of the recent vulnerabilities of this type that are exist in these browsers are listed below:

a. Html Object Memory Corruption Vulnerabilities: HTML object memory corruption vulnerability is associated with a pointer of a deleted HTML object. Intruder can use the pointers of deleted objects to run arbitrary code due to incorrectly initialized memory and improper handling of objects in memory.

b. Race Condition Memory Corruption Vulnerability: The cause of Race condition memory corruption vulnerability is a bit different. The way Internet Explorer accesses an object that may have been corrupted due to a race condition may invoke its existence. It's Exploitation and the consequences are similar to the HTML Object or Uninitialized memory corruption vulnerability.

c. Post Encoding Information Disclosure Vulnerability: An information disclosure vulnerability leaks sensitive information. It occurs while submitting data to the server. Exploitation may occur if a user visits a web page which is specifically crafted to take advantage of these vulnerabilities. Successful exploitation of this vulnerability could result in an attacker viewing content from the local computer or another browser window in another domain or Internet Explorer zone.

d. Overflow Remote Code Execution Vulnerability: Overflow Remote Code Execution vulnerability is due to an integer overflow error in Web Open Font Format (WOFF) decoder which is the abbreviation for Mozilla Web Open Fonts Format. WOFF is a simple compressed file format for fonts. The WOFF decoder handles the size of tables which are specified in the font file an integer overflow vulnerability may exist. This error could result in a buffer overflow vulnerability on a subsequent memory allocation.

VII. ATTACKS AGAINST WEB BROWSER

1. Browser bugs exploitation: It takes advantage of programming errors in the browser or in the internet protocols properties. Among the most famous feats we find the user's home page hacking by replacing it with that of the hacker and complicate the reset by the user. Generally, codes attacks are based on vulnerabilities already present within browsers, usually a design error or unexpected behavior. On another hand, there are some methods based on the browser and extensions interactions that require high level privileges for functioning. However, these transactions make the browser more sensitive to attacks.

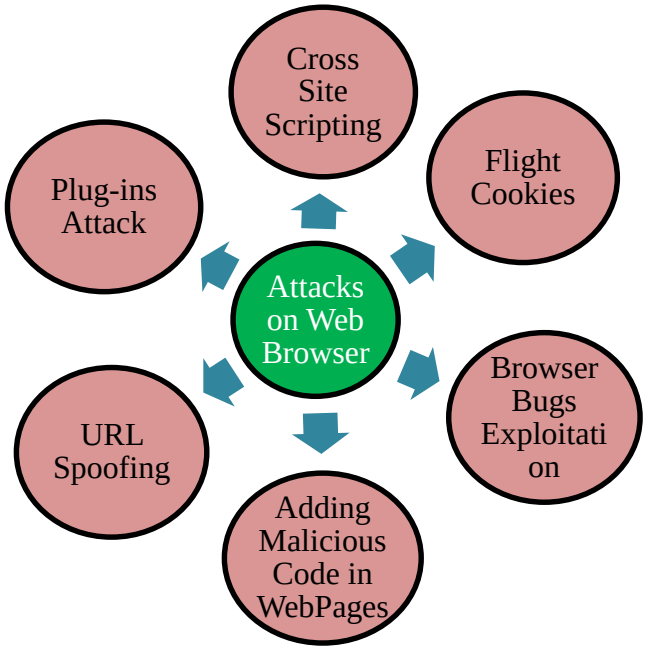

Figure 8: Attacks on Web Browser

2. URL spoofing: URL spoofing is used by phishers mainly aiming the theft of users' identities through the collection of their confidential data. This attack previously involves creating a website similar to that of the victim. Therefore, spoofing attacks use usurped URLs, usually those of e-commerce, banks, etc., and encourage

victims to enter their confidential information (passwords, accounts numbers etc.).

3. Adding malicious code in web pages: This type of attacks is based on including executable codes or scripts on a previously compromised web site. Some interpreted codes or scripts can truly improve the navigation ergonomics. Their use by an unscrupulous user is a significant flow. An HTML or JavaScript code combined with malicious ActiveX or Java code can potentially crash the browser.

4. Flight cookies attacks: As files written on the user's computer by a remote web server in order to back up a connection context, and seen their transitions usually clear on the network; the cybercriminals' choice of using cookies gives a free access to the user's machine present on the network. Cookies can be considered as a breach in the confidentiality of communication. Theft of stored data affects the privacy of its victims and allows retrieval of the required information for a web site authentication.

5. Plug-ins attacks: The cybercriminals do not only look for the browser vulnerabilities, but they are also interested in the bugs of the browser plug-ins to help them to carry out drive-by downloads and click jacking attacks. Frequently, users enable scripting when the site does not load correctly. Therefore, this attack is based essentially on a known vulnerability. Particularly, companies should be wary of Java which is considered as the most susceptible language for carrying out an attack and one of the preferred cyber criminal's languages.

6. Cross-Site scripting: A Cross-Site Scripting (XSS) attack consists in injecting arbitrary script in a web page to cause a malicious action. XSS attacks aim to run a script allowing data transmitting from a website to another. The XSS vulnerabilities are divided into two types: standing XSS attacks correspond to cases where the malicious script is stored on the remote server. This implies the script running at any time by all the users of the website. The second type corresponds to non-standing XSS which consists in injecting the malicious script in the URL. The use of these XSS attacks can be interpreted by sessions and cookies flight or delivering a website unreachable. In the following, we give a simple example of a XSS attack which consists in creating a fake image in JavaScript and which allows the user's cookies collection.

VIII. BROWSER EXTENSION

It is a small piece of code that let developers to add an additional functionality to the browser. It is also called Plug-in or Add-ons. We have to create a set of rules to maintain the principle of least privileges in the browser. The extension allows user to modify the browser behavior on certain websites. While usage of extension is widespread on the desktop but it is still in the native stage on mobile browsers. Only a few mobile browsers such as Mozilla Firefox for android and Dolphin support third party extension.

Role of Extensions in Web Browser:

Browser extensions allow for customization of the browser by adding functionality. The way these extensions are integrated strongly differs in the four major browsers. One example for this diversity is the possibility to add themes to the browser. Firefox and Chrome allow this feature while Internet Explorer and Safari do not. Although this example clearly shows the difference in the extensibility of the four browsers, themes stay out of the focus of this work. Browser extensions modify the core browser user experience by changing the browser's user interface and interacting with websites.

Internet Explorer's extension model:[10]

Internet Explorer supports several extension mechanisms out of which browser helper objects i.e. BHOs are probably the most commonly used. BHOs have virtually unrestricted access to IE's event model and have been used by malware writers in the past to create password capturing programs and key loggers. This is especially true because some BHOs run without making any change in the user interface.

For instance, the ClSpring Trojan [4] uses BHOs to install scripts which provide a number of instructions to be performed such as adding and deleting registry

values and downloading additional executable files which is all completely transparent to the user. Even if the BHO is completely benign, but buggy, its presence might be enough to open up exploits in another fully patched browser.

Firefox's extension model:[10]

Firefox extensions are typically written in JavaScript and can modify Firefox browser in sort of unrestricted ways. This flexibility comes with few security guarantees. Extensions run with the same privilege as the browser process, so malicious extension can cause random damage. Firefox extensions often employ highly dynamic programming techniques that make it difficult to reason about their behavior. To protect end-users, Firefox relies on a community review process to determine which extensions are safe. Only extensions deemed safe are added to Mozilla's curated extension gallery. Firefox usually refuses to install extensions that do not originate from this gallery. Users are thus protected from unreviewed extensions, but reviews themselves as error-prone and sometimes malicious extensions are accidentally added to the gallery. An example of this is Mozilla Sniffer, an extension which was downloaded close to 2,000 times, before being removed from the gallery after it was deemed malicious.

Architecture of Firefox Extension:[13]

1. Extension Privileges: To improve the browser functionality and get customizable features, Firefox extensions execute with the full chrome privileges by invoking XPCOM interface. The XPCOM interface includes services such as file system access, process launching, network access, Browser components and APIs access.
2. JavaScript: The JavaScript functions can be used for code injection and privilege escalation attacks. The Firefox automatically wraps the object to prevent malicious script from accessing the properties and methods of the document object.
3. XPCOM: The Firefox extension can use XPCOM interface to interact with low layer libraries, like network, I/O, file system, etc.

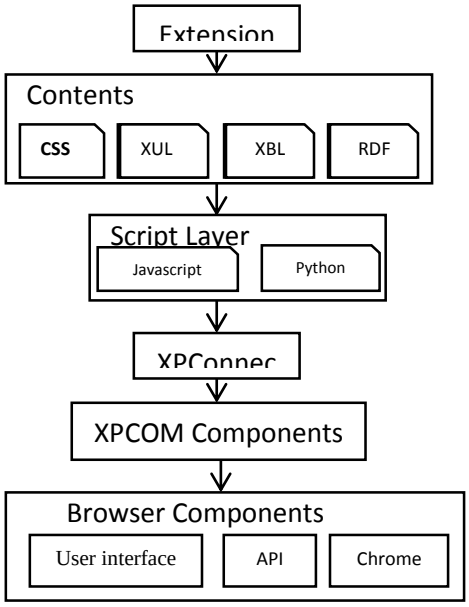

Figure 7: Architecture of Mozilla Firefox Extension

4. API's: XMLHttpRequest is a JavaScript API that allows a client side JavaScript code to communicate over HTTP and HTTPS channel. It can be used to send same domain as well as cross-domain HTTP/S requests.

Chrome's Extension Model:

Chrome extensions are written in JavaScript and hosted on extension pages, but they have access to APIs that are not available to web pages. An extension page run in the context of the extension process, different from the browser processes and has the ability to both access and augment the browser UI. Extension pages can register to listen to special browser events such as tab switching, window closing, etc.

Extension manifests: Extensions specify their resources and the capabilities they require in an extension manifest file. When a user tries to install an extension, Chrome reads the extension manifest and displays a warning and the warning raised by Chrome before the extension is installed.

Over-privileged extensions: Chrome's model also allows extensions to request rights over other resources, including, the privilege to access "your data on all websites". Many simple, seemingly benign operations require extensions to request access to this very coarse privilege. In all the cases, manifests are uninformative and the extensions require manual code review.

Extension study: We conducted a simple analysis of the manifests for over 1,139 popular Chrome extensions, to determine how many require the capability to read and write to all websites.

Architecture of Chrome's Extension:

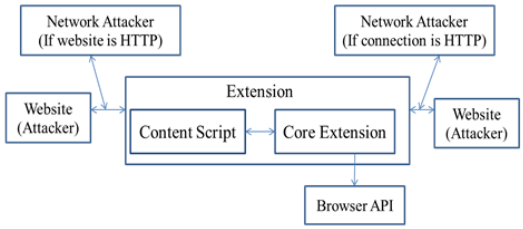

Figure 8: Architecture of Google Chrome's Extension

Architecture of Google Chrome Extension is divided into three parts i.e.

1. Content Script: Content scripts incorporate any sort of JavaScript file that runs in the context of web pages and allow for direct interaction with web pages. Each content script can directly access the DOM of a single web page. However, content scripts cannot use variables and functions defined by web pages.

2. Core Extension: The extension core holds the user interface of an extension and can access the APIs requested in the manifest file. This part of the extension is implemented in HTML and JavaScript. Although holding the main logic of the extension, the extension core cannot directly interact with web content. The extension core needs to communicate with a content script through the message APIs or execute a XML Http Request.

3. Browser API: It can be integrated to extensions via NPAPI plug-ins and make the only possibility for an extension to execute arbitrary code and to access the user's file system outside the extension's folder. By default, it can only interact with the extension core but developers can expose browser API's directly to web content.

Safari Extension Model:[14]

Safari implements the principle of least privilege by strongly restricting extensions in their privileges. Safari extensions can neither access the file system outside the extension's folder, nor access user-related data such as cookies, bookmarks, or the browsing history. Safari extensions cannot execute native code and furthermore, extensions lack the ability to manage proxy settings, to add themes to the browser, to push notifications, to communicate with other extensions, to access the clipboard, to access the application cache of the browser, and to access functions and variables defined in web page scripts.

Architecture of Safari Extension:[14]

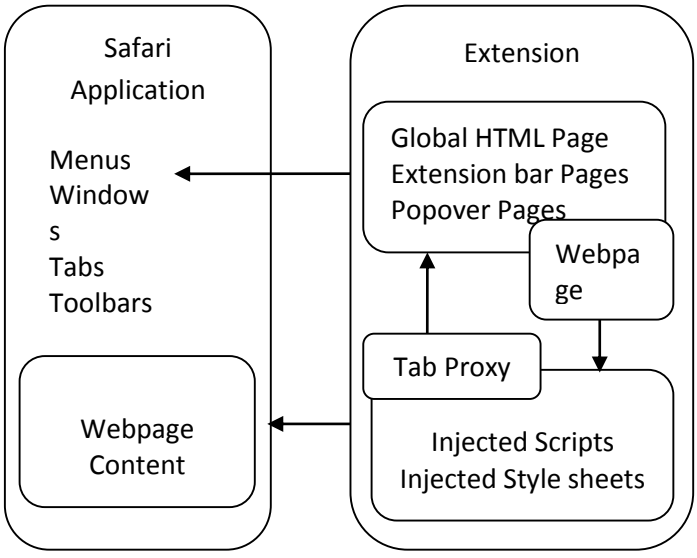

Figure 9: Architecture of Safari Extension

Architecture of Safari is divided into two parts i.e.

1. Application Part: It can hold any global page or extension bar. Application part interacts with Safari applications and can access Safari application and Safari Extension classes.
2. Content Part: It can hold injected scripts and injected style sheets. Content part interacts with web content and has access to Safari Content Extension class. In this extension parts can interact with each other by sending messages over message proxies. There are two message proxies i.e.
    a) Tab Proxy: It is responsible for forwarding messages from the content part to the application part.
    b) Webpage Proxy: It forwards messages from the content part to the application parts and vice versa.

IX. RELATED WORK

A. Existing javascript security methods are inadequate for preventing javascript injection attacks that can exploit vulnerable extensions. Anton et.al. [5] present a runtime protection mechanism based on code randomization technique which is coupled with a static analysis technique to protect browser extensions from javascript attacks. The protection is applied at runtime by separating malicious code from the randomization extension code. The protection mechanism is evaluated on the set of vulnerable and non-vulnerable firefox extensions. Their results indicate that the approach would be a viable extension. Their approach were also be able to reduce false positives and achieve maximum compatibility with existing extensions and the burden of rewriting an extension for a new API from developers is evacuated.

B. Man-in-the-Browser is a Trojan horse that infects a web browser and has the ability to tamper the contents of web pages and transactions. This attack is a serious threat for online services. Sampsa Rauti et.al. [12] explains that the problem is raised by the powerful browser extensions and viable attack surface of internet applications. Browser extension is not only the way to realize man-in-the-browser attack. Techniques like Modifying payload, Modifying DOM tree, Modifying Ajax transmission mechanism, Modifying Ajax application functionality have flaws as well, because these are implemented on the target site in javascript which can be overwritten by the attacker. But the dynamical changes in implementation limits the attacker's time frame and make this job harder.

C. Colluding browser extension attack is an attack in which one extension interacts with another installed extension and share their object or information. Anil Saini et.al. [14] extends the concept of colluding extension and demonstrate a new attack that can leverage the concept and causes the privacy leakage in a web browser. In this paper, object reference sharing, event notification and preference overriding is identified. There is also a proof-of-concept explaining how multiple extensions can collude with each other for adjusting the browser for data leakage. In this paper, possible approaches are there to mitigate the colluding browser extension attack.

D. SABRE [11] tracks the flow of JavaScript objects from sensitive sources to sinks inside the Mozilla Firefox browser by employing a dynamic taint analysis technique. White listing is used to separate benign extension flows from malicious ones. However, the whitelist approach essentially delegates the responsibility of deciding the maliciousness of an extension to a user. Similarly, a dynamic taint analysis based approach detects vulnerable extensions. This approach attempts to prevent unprivileged data from being compiled into privileged bytecode. It also identifies and prevents privileged caller functions from accidentally calling unprivileged code.

E. IBEX [10] is a general purpose browser extension development system that provides verifiable security guarantees. It provides an API that lets extensions to use common browser functionalities. Privileges of an extension are specified in a custom policy language. The

extension code can then be formally verified against the specified policy. A cross-compiler is available to deploy the same extension in Internet Explorer, Google Chrome, Mozilla Firefox and C3 (an experimental browser). However, to use IBEX, extension developers need to write their extensions in a verifiable language other than JavaScript. Moreover, each browser provides unique APIs for its extension system that is constantly updated with new features, making it difficult to develop extensions using a general purpose system like IBEX.

X. ATTACKS ON BROWSER EXTENSION

Two possible attacks on extensions in web browsers are:

1. Malicious Extension: These types of extensions are written by well-meaning developers who are not security experts. We generally do not consider malicious extensions because preventing malicious extensions require completely different tactics such as warnings, user education, security scans of the market, feedback and rating, etc.

2. Non-Malicious Extension: They are of two types:

a) Network Attackers: People who use insecure networks may encounter network attackers. Network attacker's goal is to collect personal information or credentials from a target user. To achieve this goal, network attacker will read and alter HTTP traffic to mount man-in-middle attack.

b) Web Attackers: Users may visit websites which are having advertisements. The website can launch a cross-site scripting attack on an extension if the extension treats the website's data or functions as trusted. The goal of web attacker is to gain access to browser user data (such as history) or another site's password. For example NPAPI (Netscape plug-in application programming interface), Adobe flash player, PPAPI (Pepper plug-in application programming interface), Coincidence detector, etc.

XI. BROWSER SECURITY

- Some software features which provide functionality to a web browser, such as Java, ActiveX, Scripting (JavaScript, VBScript, etc.), may also introduce vulnerabilities to the computer system.

- These vulnerabilities may be introduced because of poor design, poor implementation, or an insecure configuration.

- For these reasons, we should understand which browser support which features and the risks they introduce.

- Some web browsers permit us fully disable the use of these technologies, while others may permit you to enable features on a per-site basis.

Six ways through which security can be enhanced:

1. Configure your browser's security and privacy settings.
2. Keep your browser updated.
3. Sign up for alerts.
4. Be cautious when installing plug-ins.
5. Make sure you have an Antivirus installed.
6. Install security plug-ins.

Security concepts for keeping extension secured:

1. Isolated Worlds: The isolated world's mechanism is intended to protect content scripts from web attackers. Extension content script can't access the direct document object module of the current running page instead of this it can access a copy of it.

2. Privilege Separation: Chrome's Extension run in two different privilege modes. One is Content Script and the other one is Core Extension Script. Content Script can access API's by using a message passing interface to talk to the core extension script. While Core Extension Script has an access to the chrome native API's but content script does not have an access on it.

3. Sandboxing: Running code or programs in a sandbox means running the code or program in a virtual, isolated environment. Sandboxing prevents negative impacts of untrusted code to the host machine.

4. Private Browsing: Browser extensions should generally not be able to circumvent the goals of private browsing by collecting data or sending data to locations excluded by private browsing.

5. Permission Model: By default, extensions are not able to use parts of the browser API which impact users' security or privacy. In order to gain access to the APIs, developer must specify the desired permissions in a file which is packaged with the extension. Chrome extensions already have a privilege model, where extensions are required to pre-declare their needed privileges and are limited to the browser.
   If vulnerability is found in the Core extension, the attacker will be limited by the privileges which are attained by the extension.

REFERENCES

[1] A. Grosskurth and M. W. Godfrey, "A reference architecture for Web browsers," 21st IEEE International Conference on Software Maintenance (ICSM'05), 2005, pp. 661-664.

[2] Dormann, Will, and Jason Rafail. "Securing your web browser." CERT, 2006.

[3] Barth, A., Felt, A. P., Saxena, P., & Boodman, A. "Protecting Browsers from Extension Vulnerabilities". In NDSS, 2010

[4] A. Barua, M. Zulkernine and K. Weldemariam, "Protecting Web Browser Extensions from JavaScript Injection Attacks," 2013 18th International Conference on Engineering of Complex Computer Systems, Singapore, 2013, pp. 188-197.

[5] Nicholas Carlini, Adrienne Porter Felt, and David Wagner. 2012. An evaluation of the Google Chrome extension security architecture. In Proceedings of the 21st USENIX conference on Security symposium (Security'12). USENIX Association, Berkeley, CA, USA, 7-7.

[6] Dhruwajita Devi, Dhrubajyoti Pathak, and Sukumar Nandi. 2010. Vulnerabilities in Web Browsers

[7] M. Šilić, J. Krolo and G. Delač, "Security vulnerabilities in modern web browser architecture," The 33rd International Convention MIPRO, Opatija, Croatia, 2010, pp. 1240-1245.

[8] A. Zammouri and A. A. Moussa, "SafeBrowse: A new tool for strengthening and monitoring the security configuration of web browsers," 2016 International Conference on Information Technology for Organizations Development (IT4OD), Fez, 2016, pp. 1-5.

[9] A. Guha, M. Fredrikson, B. Livshits and N. Swamy, "Verified Security for Browser Extensions," 2011 IEEE Symposium on Security and Privacy, Berkeley, CA, 2011, pp. 115-130.

[10] M. Dhawan and V. Ganapathy, "Analyzing Information Flow in JavaScript-Based Browser Extensions," 2009 Annual Computer Security Applications Conference, Honolulu, HI, 2009, pp. 382-391.

[11] Sampsa Rauti and Ville Leppänen. 2012. Browser extension-based man-in-the-browser attacks against Ajax applications with countermeasures. In Proceedings of the 13th International Conference on Computer Systems and Technologies (CompSysTech '12), Boris Rachev and Angel Smrikarov (Eds.). ACM, New York, NY, USA, 251-258.

[12] Anil Saini, Manoj Singh Gaur, and Vijay Laxmi. 2013. The darker side of Firefox extension. In Proceedings of the 6th International Conference on Security of

Information and Networks (SIN '13). ACM, New York, NY, USA, 316-320.

[13] Anil Saini, Manoj Singh Gaur, Vijay Laxmi, Mauro Conti, Colluding browser extension attack on user privacy and its implication for web browsers, Computers & Security, Volume 63, November 2016, Pages 14-28