Table of contents

List of tables and figures

Abbreviations list

OWASP	Open Web Application Security Project
SQL	Structured Query Language
HTTP	Hypertext Transfer Protocol
URN	Uniform Resource Name
URL	Uniform Resource Locator
URI	Uniform Resource Identifier
GUI	Graphic User Interface
XSS	Cross Site Script
CSRF	Cross Site Request Forgery
TCP	Transmission Control Protocol
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
GMT	Greenwich Mean Time
US-ASCII	American Standard Code For Information Interchange
IE	Internet Explorer
ISO	International Organization For Standardization
XHR	Xml Http Request
IANA	Internet Assigned Numbers Authority

General Introduction

General introduction

Context of the study:

Today all the transactions and activities that happen online across the world, are results of web applications.

Now using web applications one can do anything right from entertainment, business information exchange, shopping, banking and so on.

Web applications have evolved from simple collections of static HTML documents to complex documents that containing hundreds of dynamically generated pages.

However, this rapid evolution in Internet usage, security concerns are also increased rapidly. Cyber-attacks are increasing day by day and Web applications are becoming more prone to worrisome vulnerabilities.

Recent researches [1] has shown that clickjacking is one of the common attacks which mainly targets social and financial sites, also shown that clickjacking is the primary source of different exploitations such as cross site request forgery and phishing.

Clickjacking is an attack that tricks victims into clicking on invisible elements of a web page to perform unintended actions that might be advantageous for the attacker.

Motivation:

According to the Cyber Statistics report presented in 2015, clickjacking represents 10.9 % of attacks which mainly targets social sites [1].

An empirical study in which it analyzes the feasibility of clickjacking attacks by testing the 500 most popular and all 36 financial institution websites in Korea [6]. Experimental results show that 524 out of 526 websites (about 99.6%) were still vulnerable to clickjacking attacks.

It also investigated the clickjacking possibility of the top 500 global websites and found that 390 of them (78%) were vulnerable to clickjacking attacks.

Moreover, in 2015 a comparison between governmental and commercial websites was done on Saudi Arabia's websites based on the numbers of vulnerabilities found. The results show that Saudi Arabia's websites suffer from high impact vulnerabilities, 61% of websites are vulnerable to clickjacking [10].

In this project we have made an experiment on 100 Algerian websites (banks, government, universities, institutes … etc.) by generating real clickjacking attacks. The results shows that 96% of Algerian websites are vulnerable on clickjacking attacks.

Statement of the Problem:

The Detection of Clickjacking attacks is a topic of active researches in the industry and academia. To achieve those purposes many defense techniques have been implemented. We divide defense techniques into two categories:

Server-Side: These mitigation techniques are those that are implemented on the website. The most widely deployed technique is the inclusion of Frame busting scripts to disable framing legitimate website in iframe.

Websites taking Server-side techniques suffer from the following shortcomings:

- This technique suffers from poor usability and incompatibility with webpages and widgets.
- This technique has all been circumvented by malicious users with Advanced Clickjacking attacks.

Client-Side: This category let researchers and developers to implement new mitigation techniques to combat clickjacking attacks. We divide client-side techniques into two categories:

- **Browser Extensions/ add-ons:** It refers to mitigation techniques that are implemented based upon the browser extensions and used by the users, example of such extensions are NoScript and ClickProtect.

The majority of these techniques have a small amount of attack vectors, with a high false positive.

- **Proxy level framework:** This category represents a client-side techniques that implemented on proxy level, example ProClick.

This technique requires a proxy installed on local host or on intranet, which affect the performance of the browser.

Objective:

In order to combat Clickjacking attacks, we propose a browser extension (ClickDetector) to defeat the attacker attempt to perform clickjacking attacks by **detecting all advanced attacks techniques reported by OWASP** to minimize the false negative detection rate.

We have also include users feedback, to avoid the false positive problem, where our extensions could report malicious websites in remote database (Blacklist), to make future interaction more informed for new users, and accelerates the detection process.

ClickDetector also exploits Google safe browsing service, to check every malicious website user are going to visit.

Report outline:

This report divided in four chapters:

The first chapter provides the basic concepts of Web Applications, Web Security, and clickjacking attack, we present a lot of recent researches that has shown that clickjacking is an emerging web application security issue today.

The second chapter provides a general overview of many defenses techniques have been suggested to combat clickjacking attacks.

The third chapter presents the conceptual design of our Browser extension (ClickDetector) with its different modules and methods for detecting advanced Clickjacking attacks, and stop the attacker's attempt.

The fourth chapter presents the implemented of ClickDetector, and discusses the obtained results from running the extension, to demonstrate the effectiveness of our extension.

Finally, we conclude this project by a general conclusion, recommendations and different perspectives.

CHAPTER 01

WEB APPLICATION AND WEB SECURITY

CHAPTER 1 WEB APPLICATION AND WEB SECURITY

1. What is web application?

By definition, it is something more than just a web site. It is a client/server application that uses a web browser as its client program, and performs an interactive service by connecting with servers over the Internet (or Intranet). A website simply delivers content from static files. A web application presents dynamically tailored content based on request parameters, tracked user behaviors, and security considerations. Browsers send requests to servers, and the servers generate responses and return them to the browsers [3].

2. Web application structure

Web applications now come in several varieties including 2-tier and 3-tier which refer to the number of levels of the applications. The three-tiered approach is most common at present and it has simple structure [15]:

- **1-tier is the client tier** which is deals with the presentation part of the system (user and system interfaces) such us Mozilla Firefox, Google Chrome …etc.
- **2-tier is the server** or the middle tier which is the most important part of the web application. It essentially plays the role of bringing together the three layers of the three tier web architecture. While technically processing the various inputs and selections received by the clients it plays the role of interaction with the vast database present in the third tier.
- **3-tier is the database tier**, it enables Web applications to store data and other content elements. By using the Structured Query Language (SQL), Web applications can interact with databases to create customized data for each user dynamically.

The 3-Tier system basically works as illustrated in figure 1.1. The client application (Web browser) receives input from the user and sends a request to the server tier. After the server application receives and processes the requests, passing it to the database tier (SQL server). Then the data base retrieves data and sends the data to the server application. The server application receives the data, executes the SQL-query and passes the result to the client application [3].

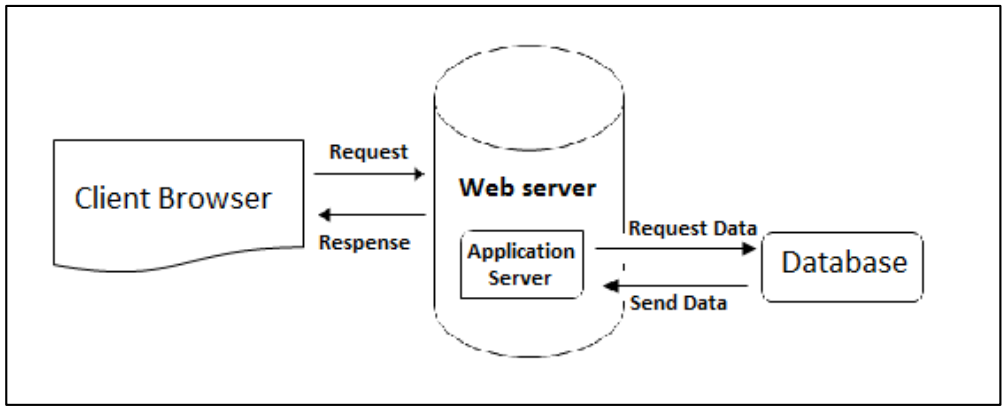


Figure 1.1 3-Tier Architecture.

3. HTTP (Hypertext Transfer protocol)

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. HTTP has been in use by the World-Wide Web global information initiative since 1990. HTTP is a generic and stateless protocol which can be used for other purposes as well using extensions of its request methods, error codes, and headers [4].

Basically, HTTP is a based communication protocol that is used to deliver data (HTML files, image files, query results, etc.) on the World Wide Web. The default port is TCP 80, but other ports can be used as well. It provides a standardized way for computers to communicate with each other. HTTP specification specifies how clients' request data will be constructed and sent to the server, and how the servers respond to these requests. [27]

3.1. Http parameters

3.1.1. HTTP version

HTTP uses a <major>.<minor> numbering scheme to indicate versions of the protocol. The version of an HTTP message is indicated by an HTTP-Version field in the first line. Here is the general syntax of specifying HTTP version number [27]:

```
HTTP-Version = "HTTP" "/" 1*DIGIT "." 1*DIGIT
```

The protocol versioning policy is intended to allow the sender to indicate the format of a message and its capacity for understanding further HTTP communication, rather than the features obtained via that communication. No change is made to the version number for the

addition of message components which do not affect communication behavior or which only add to extensible field values [4].

3.1.2. Uniform Resource Identifiers

URIs have been known by many names: WWW addresses, Universal Document Identifiers, Universal Resource Identifiers, and finally the combination of Uniform Resource Locators (URL) and Names (URN). As far as HTTP is concerned, Uniform Resource Identifiers are simply formatted strings which identify--via name, location, or any other characteristic--a resource [4].

URIs in HTTP can be represented in absolute form or relative to some known base URI, depending upon the context of their use. The two forms are differentiated by the fact that absolute URIs always begin with a scheme name followed by a colon [3]

The “http” scheme is used to locate network resources via the HTTP protocol. This section defines the scheme-specific syntax and semantics for http URLs.

```
http_URL = "http:" "//" host [ ":" port ] [ abs_path [ "?" query ]]
```

If the port is empty or not given, port 80 is assumed. The semantics are that the identified resource is located at the server listening for TCP connections on that port of that host, and the Request-URI for the resource is abs_path. If the abs_path is not present in the URL, it must be given as “/” when used as a Request-URI for a resource [4].

3.1.3. Date/Time Formats

All HTTP date/time stamps MUST be represented in Greenwich Mean Time (GMT), without exception. HTTP applications are allowed to use any of the following three representations of date/time stamps [27]:

```
Sun, 06 Nov 1994 08:49:37 GMT ; RFC 822, updated by RFC 1123
Sunday, 06-Nov-94 08:49:37 GMT ; RFC 850, obsoleted by RFC 1036
Sun Nov 6 08:49:37 1994 ; ANSI C's asctime() format
```

The first format is preferred as an Internet standard and represents a fixed-length subset of that defined by RFC 1123. The second format is based on the obsolete RFC 850 date format and lacks a four-digit year.

HTTP/1.1 clients and servers that parse the date value MUST accept all three formats (for compatibility with HTTP/1.0), though they MUST only generate the RFC 1123 format for representing HTTP-date values in header fields [4].

3.1.4. Character Sets

We use character sets to specify the character sets that the client prefers. Multiple character sets can be listed separated by commas. If a value is not specified, the default is the US-ASCII.

Following are the valid character sets [27]:

US-ASCII or ISO-8859-1 or ISO-8859-7

3.1.5. Content Encodings

A content encoding value indicates that an encoding algorithm has been used to encode the content before passing it over the network. Content coding are HTTP 5 primarily used to allow a document to be compressed or otherwise usefully transformed without lo sing the identity [3].

All content-coding values are case-insensitive. HTTP/1.1 uses content-coding values in the Accept-Encoding and Content-Encoding header fields [4].

Following are the valid encoding schemes:

Accept-encoding: gzip or Accept-encoding: compress or Accept-encoding: deflate

3.1.6. Media Types

HTTP uses Internet Media Types in the Content-Type and Accept header fields in order to provide open and extensible data typing and type negotiation. All the Media-type values are registered with the Internet Assigned Number Authority (IANA). The general syntax to specify media type is as follows [27]:

media-type = type "/" subtype *(";" parameter)

The type, subtype, and parameter attribute names are case-insensitive.

Example:

```
Accept: image/gif
```

3.1.7. Language Tags

HTTP uses language tags within the Accept-Language and Content- Language fields. A language tag is composed of one or more parts: a primary language tag and a possibly empty series of subtags [25]:

```
language-tag =primary-tag *("-" subtag )
```

White spaces are not allowed within the tags and all tags are case-insensitive.

Example tags include:

```
En, en-USA, en-cockney, i-cherokee, x-ping-latin
```

Where any two-letter primary-tag is an ISO-639 language abbreviation and any two-letter initial subtag is an ISO-3166 country code.

3.2. HTTP Message

HTTP messages consist of requests from client to server and responses from server to client.

HTTP-message = Request | Response; HTTP/1.1 messages.

Request and Response messages use the generic message format of RFC 822 [4] for transferring entities (the payload of the message). Both types of message consist of a start-line, zero or more header fields (also known as “headers”), an empty line (i.e., a line with nothing preceding the CRLF) indicating the end of the header fields, and possibly a message-body [4].

3.2.1. HTTP Requests

An HTTP client sends an HTTP request to a server in the form of a request message which includes following format:

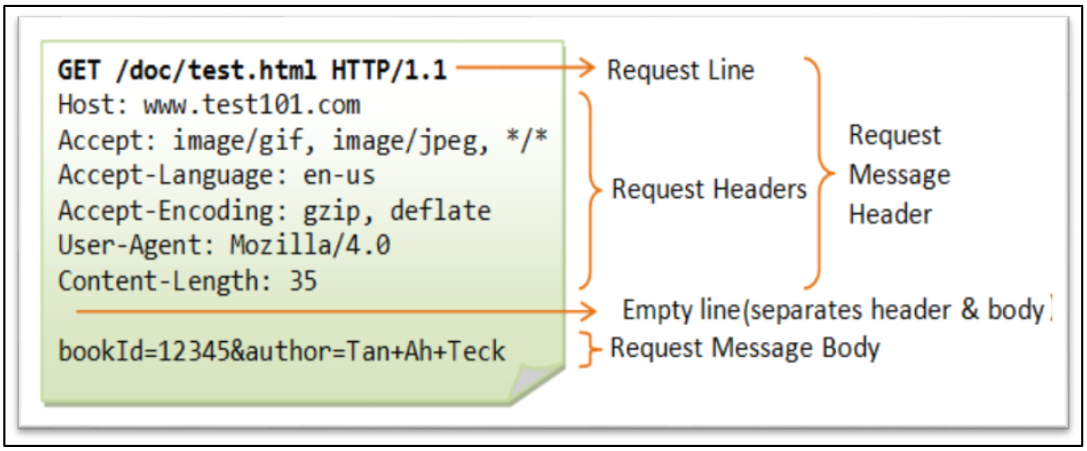


Figure 1.2 Sample HTTP request message [4].

3.2.2. HTTP Responses

After receiving and interpreting a request message, a server responds with an HTTP Response message.

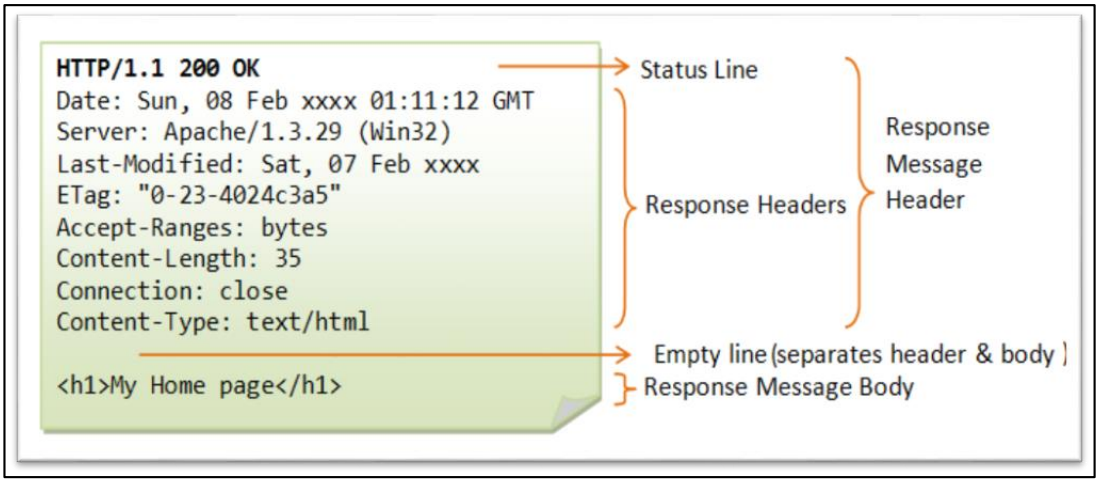


Figure 1.3 Sample HTTP response message [4].

4. Web security

As with any new class of technology, web applications have brought with them a new range of security vulnerabilities. The advantage of web technology, user unawareness and the flexible nature of its structure can easily make the malicious user to perform unintentional tasks without the user consent. Clickjacking [22] is a hijacking attack, which is considered as one of the topmost web attacks, where the attacker hijacks the user clicks in order to perform malicious actions.

According to the Cyber Statistics report presented in 2015[1], clickjacking represents 10.9 % of attacks which mainly targets social sites.

Attack Techniques	Percentage of exploitation in the year 2015 (%)
Unknown	26.5
Defacement	16.4
SQLi	14.3
Malicious iframe	10.9
Targeted attack	10.5
Account hijacking	10.4
Malware	6.9

Figure 1.4 Cyber Statistics report 2015 [1].

An empirical study in which it analyzes the feasibility of clickjacking attacks by testing the 500 most popular and all 36 financial institution websites in Korea [6]. Experimental results show that 524 out of 526 websites (about 99.6%) were still vulnerable to clickjacking attacks.
It also investigated the clickjacking possibility of the top 500 global websites and found that 390 of them (78%) were vulnerable to clickjacking attacks.

Moreover, in 2015 a comparison between governmental and commercial websites was done on Saudi Arabia's websites based on the numbers of vulnerabilities found. The results show that Saudi Arabia's websites suffer from high impact vulnerabilities, 61% of websites are vulnerable to clickjacking [10].

5. Clickjacking Attack

5.1. Definition

Clickjacking (which is a subset of “UI redressing”), is a malicious technique that consists of deceiving a web user into interacting (in most cases by clicking) with something different to what the user believes they are interacting with.[22]

This type of attack, that can be used alone or in combination with other attacks, could potentially send unauthorized commands or reveal confidential information while the victim is interacting with seemingly harmless web pages. The term “Clickjacking” was coined by Jeremiah Grossman and Robert Hansen in 2008 [22] and announced a talk on the topic at OWASP AppSec 2008.

Clickjacking attacks have been reported to be usable in practice to trick users into initiating money transfers, clicking on banner ads that are part of an advertising click fraud, posting blog

or forum messages, or, in general, to perform any action that can be triggered by a mouse click[22].

5.2. Basic Clickjacking attack

This basic attack type based on invisible iframe**.** An attacker first designs a malicious web page containing an iframe which may load a target web page or a GUI element.

Figure 1.5 Attacker framing victim webpage.

The iframe opacity level is set very low to make it barely visible to the user. The malicious web page allures the victim to click on a visible GUI element that is overlapped by the invisible web page(s) loaded in the iframe.

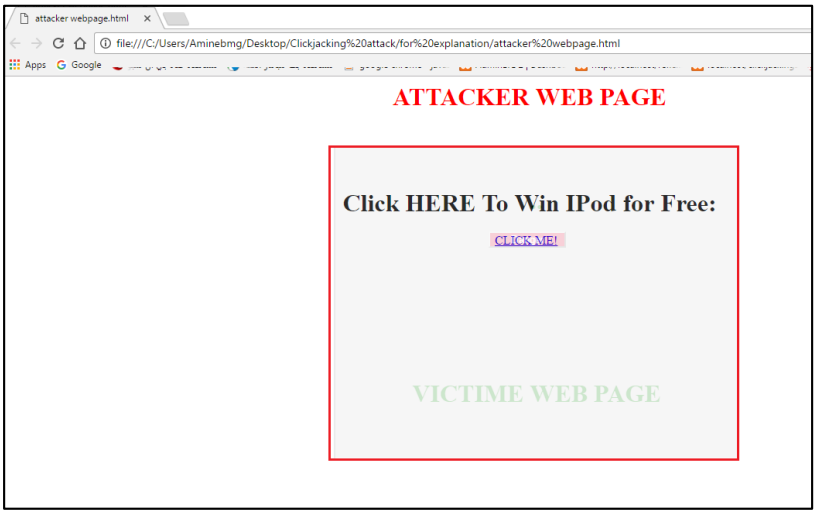


Figure 1.6 Attacker set iframe invisible.

Once the victim is surfing on the fictitious web page, he thinks that he is interacting with the visible user interface, but effectively he is performing actions on the hidden page. Since the

hidden page is an authentic page, the attacker can deceive users into performing actions they never intended to perform.

6. Existing clickjacking attacks

Existing clickjacking attacks are classified into three categories [9] illustrated in fig. 1.7:

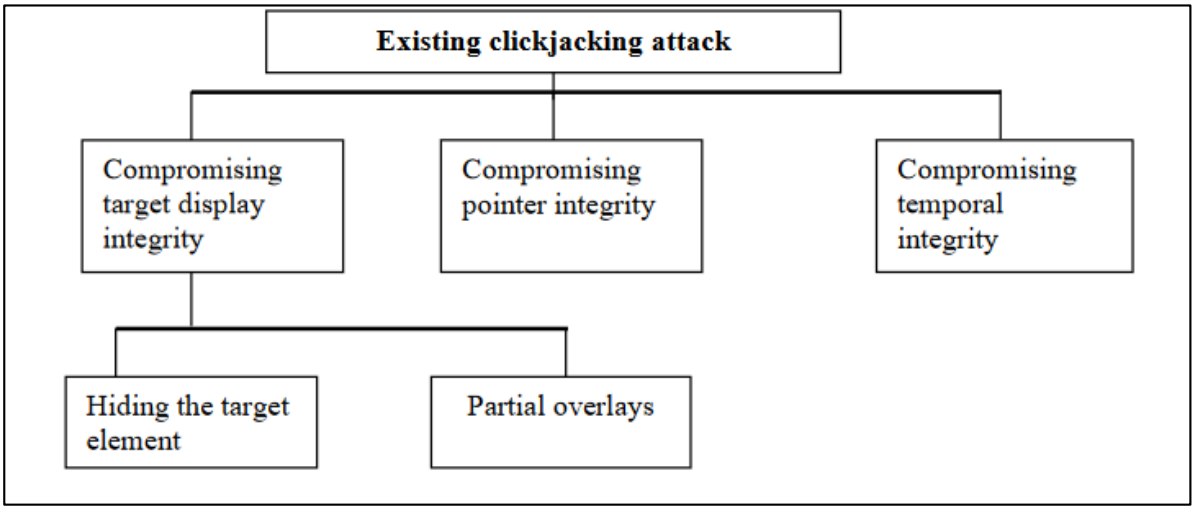


Figure 1.7 Existing clickjacking attack schema [13].

6.1. Compromising target display integrity

Hiding the target element: This can be done by making the target element transparent by wrapping that in a div container or iframe with opacity value zero and by using lower CSS z-index [9].

Partial overlays: It is possible to visually confuse a victim by obscuring only a part of the target element.

6.2. Compromising pointer integrity

An attacker can easily hide the default cursor and programmatically draw a fake cursor several pixels away from the original position by using the CSS cursor property. This will make the victim to click on the fake cursor it known as Cursorjacking. [9].

6.3. Compromising temporal integrity

The UI element is manipulated after the user decided to click on the element, but before the actual click occurs. Humans will take few hundred milliseconds to react to changes, attacker will use this time to change the element [9].

7. Advanced Clickjacking attacks

The six types of attack that we will present that are designed to avoid the execution of Frame Busting code which is the most popular way to defend against Clickjacking:

7.1. Double framing

If the attacker encloses the victim in one frame inside another (a double frame), then accessing parent.location becomes a security violation in all popular browsers, due to the **descendant frame navigation policy**. This security violation disables the counter-action navigation. [23].

7.2. OnBeforeUnload Event

A user can manually cancel any navigation request submitted by framed page. So attacker can register an OnBeforeUnload handler at the top of his malicious web page which is called whenever the framing page is about to be unloaded due to navigation.

The handler function returns a message to warn a victim of leaving legitimate website for example “Do you want to exit www.xyz.com website?” If the user choose to not leave the website, the action is loading the legitimate web page in iframe. Thus the frame busting code is defeated successfully [8].

7.3. No-Content flushing

The previous attack requires user interaction, the same attack can be done without promoting the user. This approach is to repeatedly submit navigation requests to a website and receiving HTTP response of 204 (no-content) from the legitimate website. Receiving a no-content response flushes the request and cancels the original request forcing the original URL to load in an iframe [24].

7.4. Reflective XSS filtering

An attacker can arbitrarily supply a portion of JavaScript code to effectively eliminate the frame busting code present in the victim’s web page. As an example, an attacker may load the victim’s web page as follows:

```
<iframe src="http://www.xyz.com/?v=<script>if">
```

The XSS filter will match that parameter "<script>if" to the beginning of the frame busting script on the victim and will consequently disable all inline scripts in the victim's page, including the frame busting script. [24]

7.5. **Clobbering top.location**

Some older browsers (IE7 and Safari 4.0.4) consider specific built-in objects (relevant to frame busting code such as top, self, and location) as mutable attributes. This opens up the opportunity to circumvent frame busting by redefining objects at the beginning of a malicious web page [8].

7.6. Restricting JavaScript

JavaScript can be disabled inside an iframe. So, the victim's web page being loaded in the iframe cannot execute to prevent clickjacking attack. There are unfortunately several ways of restricting JavaScript code: **In IE 8** [24]:

```
<iframe src="http://www.victim.com" security="restricted"></iframe>
```

In Chrome:

```
<iframe src="http://www.victim.com" sandbox></iframe>
```

In Firefox and IE:

Activate designMode in parent page.

8. Other Clickjacking Attack

This attacks that do not intend to circumvent the execution of frame busting code. It rely on CSS property and JavaScript features:

8.1. Cursorjacking

An attacker can easily hide the default cursor and programmatically draw a fake cursor several pixels away from the original position. So the victim assumes that he clicked on the object that he sees on. In fact, the clicked object is located at different place.

CHAPTER 02

CLICKJACKING DEFENSE TECHNIQUES

CHAPTER 2 CLICKJACKING DEFENSE TECHNIQUES

1 Introduction

Several clickjacking defense techniques have been proposed, The Existing defense techniques is classified[13] into two categories based on how the mitigation is being done to combat the clickjacking attack[8] .First category is client-side approaches where actually the mitigation were done on proxy level or with a browser Add-on/Browser based. The second category is a Server-sides approaches which is Website code/script, the most widely defense is the inclusion of frame busting code and X-frame-option header.

In this chapter we will see the previous Clickjacking defense techniques that have been proposed and its limitations.

2. Existing anti-clickjacking defenses

Existing defense techniques are classified into two sides [9] (client side and server side), for the client side there are two categories alternative browser design and proxy level framework.

Figure 2.1 shown the two sides of defense techniques against clickjacking attacks.

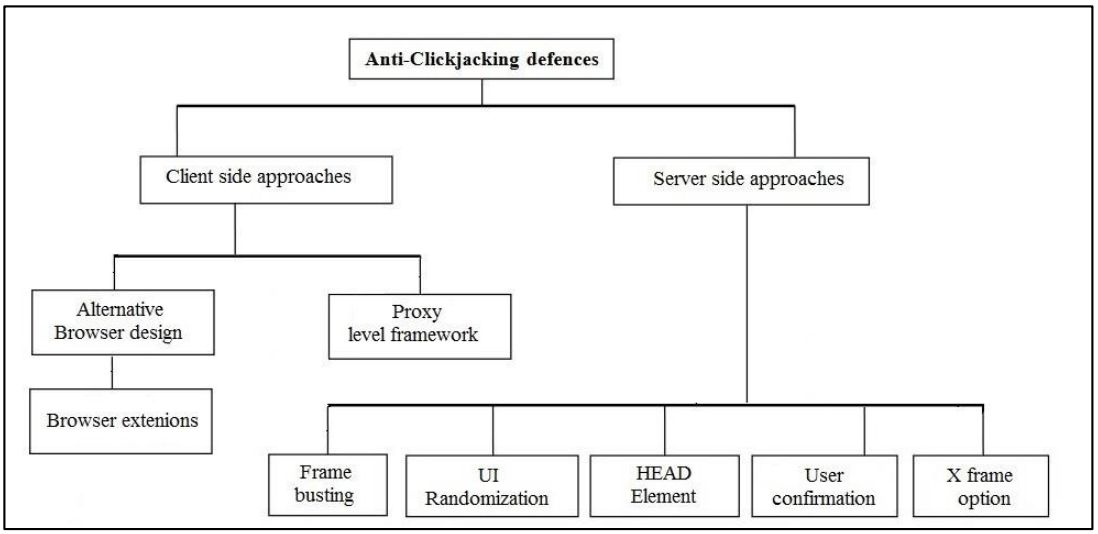


Figure 2.1 existing anti-clickjacking defenses techniques.

3. Server-side (website code/script) approaches

3.1. Frame busting

Website owners can protect their users against clickjacking attack by including JavaScript code snippets in pages that need protection against clickjacking (Rydstedt et al., 2010). The code snippet bellow checks if a loaded web page is at the top window or not. If the current page is not in the top window, then it is loaded on the top window, thereby stopping the page loading in the iframe [8].

```
if (top.location != self.location) {
parent.Location = self.location;
}
```

Limits:

Unfortunately, this approach has been defeated, the advanced clickjacking attack such us multiple iframe, Reflected XSS ….etc. that are designed to avoid the execution of frame busting code.

Also, a number of recent studies [16] have found that most websites do not deploy frame busting code. Research conducted in 2010 showed that at that time, only about 14 percent of the Alexa Top 500 web sites and 37 percent of Alexa Top 100 web sites were using frame busters to secure their websites.

3.2. User Confirmation

One straight forward way is to provide authentication for a click i.e. sending conformation popup after the click has been done on the target element. Facebook currently using this approach for like button in order to avoid clickjacking (likejacking) [9].

Limits:

First of all, this approach is not used by every website, because it degrades user experience and victim may be circumvent to click both target element and confirm popup.

3.3. UI Randomization

Another technique to protect the target element is to randomize its UI layout. For example, PayPal could randomize the position of the Pay button on its express checkout dialog to make it harder for the attacker to cover it with a decoy button [9].

Limits:

This is not robust, since the attacker may ask the victim to keep clicking until successfully guessing the Pay button's location.

3.4. HEAD Element

A workaround is to apply a special HTML HEAD element containing a special style id with no display for a framed page. The page is displayed only if it is located as the top window (OWASP, 2015b) [17].

3.5. X-Frame-Options headers (HTTP header)

This approach introduced by Microsoft. It similar to frame busting it also avoid unauthorized framing [13].

The X-Frame-Options HTTP response header (RFC7034, 2013) can be used to indicate whether a page can be rendered in an iframe or not.

The header can have three possible values [8]:

- Deny (prevents any domain from framing the content of a URL).
- Same-origin (only allows framing of pages from the current website).
- Allow-from (a set of whitelisted domains that can frame a given URL).

Limits:

However, this technique requires sending HTTP header from each of the pages that need protection against clickjacking attacks [23].

In particular, many legacy web applications may not include HTTP Headers (to be generated by the server side code).

Another difficulty is the lack of support at the browser side to understand the header and act appropriately (e.g. older versions of IE and Firefox will ignore the header) [20].

4. Client-Side approaches

4.1. Alternative browser designs (Browser Add-on/Browser extension)

4.1.1. NoScript

NoScript [21] is a Firefox add-on that provides protection against common security vulnerabilities such as cross-site scripting. It also features a URL access-control mechanism that filters browser-side executable contents such as Java, Adobe Flash, and Microsoft Silverlight. Anti-Clickjacking feature was integrated into NoScript after October 2008 as ClearClick and ClickIDS that we will explain in this chapter.

4.1.2. ClearClick

ClearClick [2] developed in 2008, module of Firefox extension to the NoScript add-on that especially catered to clickjacking. Whenever a user interacts with an embedded element which is partially obstructed, transparent or otherwise disguised, ClearClick intercepts the user click and make the user know about the hidden embedded element. It supports desktop and mobile versions via the NoScript add-on.

ClearClick combat invisible iframe clickjacking attack, also it was the first defense technique that have been proposed after clickjacking attack was discovered in 2008.

Limits:

ClearClick exhibited a large number of false positives in experiments that have been done by ClickIDS developers [5].

4.1.3. ClickIDS

ClickIDS [9] is a Firefox extension developed by Balduzzi et al 2010. ClickIDS was proposed to reduce the false positives of ClearClick by alerting users only when the clicked element overlaps with other clickable elements.

Figure 2.2 shown the architecture system of ClickIDS extension [9], which consist of two main parts: **A detection unit** is responsible for identifying possible clickjacking attempts on the web page under analysis where it combines two browser plug-ins (ClickIDS that they developed and NoScript) that operate in parallel to analyze the automated clicks, **a testing**

unit is in charge of performing the clicks which contains a Xclick plug-in.

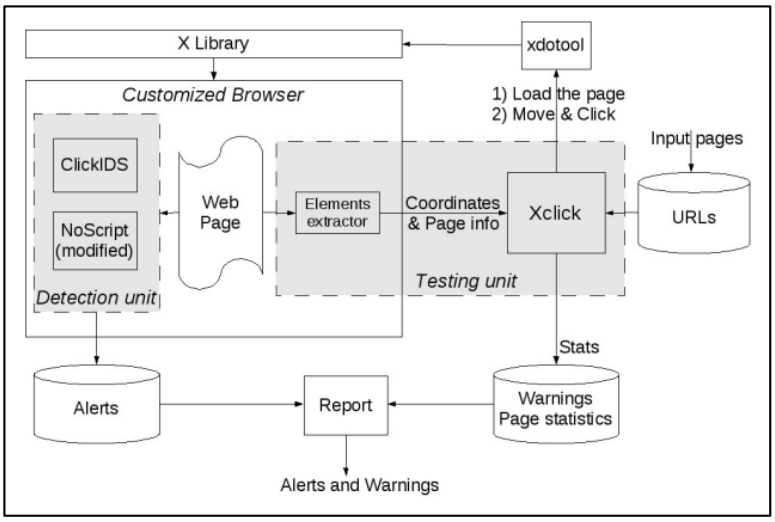


Figure 2.2 System architecture ClickIDS [9].

Limits:

Unfortunately, ClickIDS it not requirement for mounting Clickjacking attack, because cannot detect attacks based on partial overlays [9].

4.1.4. ClickSafe

ClickSafe [5] developed by Shamsi J, Hameed S, Rahman W, Zuberi F, Kaiser A, Amjad in 2014[5], to provide increased security and reliability against clickjacking attacks. It depend on user's feedback and websites rating.

In Figure 2.3, illustrate the architecture system of ClickSafe extension. It composed of three main parts: **the Detection Unit** which is responsible for detecting any insecure elements that contain implicit redirection code, **the mitigation unit** where the extension intercept the user clicks and make the user aware of the potential threat of his click, **the feedback unit** which is the most critical part in that system where records user's actions, convert them into ratings [5].

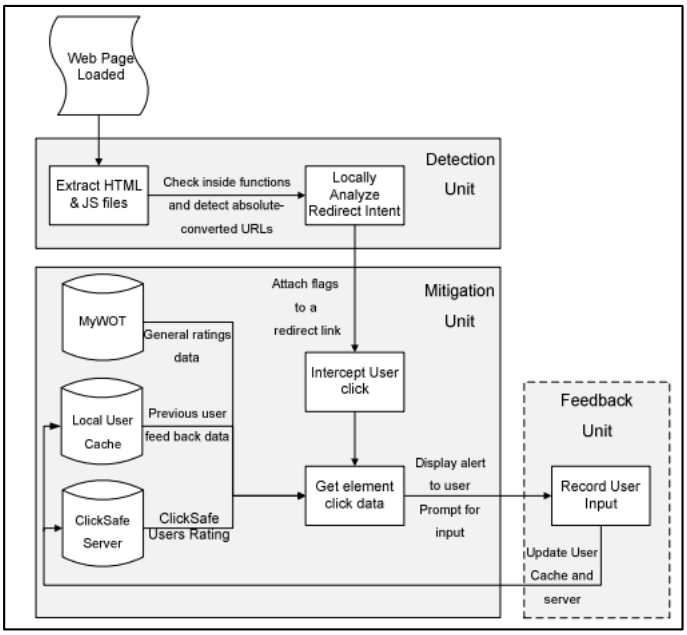


Figure 2.3 ClickSafe System architecture [5].

Limits:

Unfortunately, First limit is traced to the fragility of user trust and the unreliability of the user's response (feedback). Thus attackers or malicious users could give false rating.

4.1.5. ClickProtect

ClickProtect [12] developed in 2016, it is envisioned to detect clickjacking attacks by warning the user before proceeding with the unsafe actions. The most common types of clickjacking attack that have been considered by the developers is the invisible iframe attack plus the fake cursor (Cursorjacking) attack.

Figure 2.4 illustrate the architecture system of ClickProtect extension. The system based on two major component: **The revealing unit** which the extension get all the source code and scan it for vulnerable elements then it will check its visibility proprieties, **the extenuation unit** which notify user about the detection attack then the user can choose to continue or not.

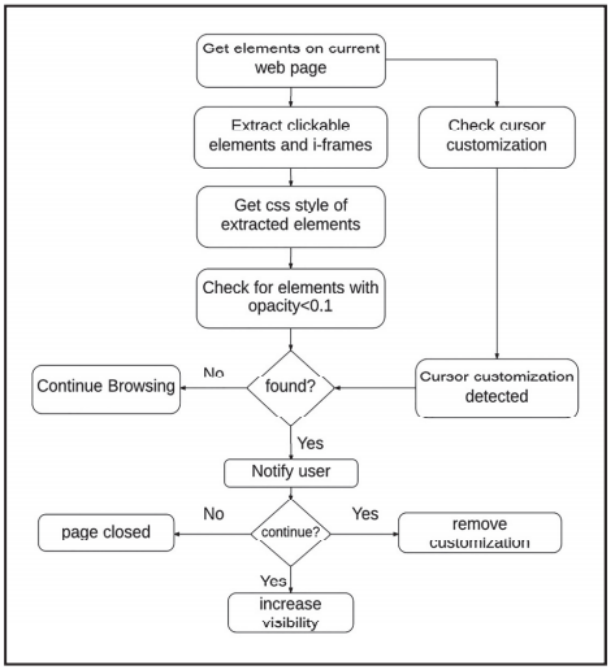


Figure 2.4 Architecture system of ClickProtect [12].

Limits:

Unfortunately, increasing visibility and user notification not enough to safe users of that hiding attacks because not every user notified are aware to what he is see.

4.2. Proxy level framework

4.2.1. ProClick

ProClick [8] is developed by Shahriar H, Devendran V, and Haddad H in 2013, it designed to check for the symptoms of clickjacking attacks. ProClick is a client-side framework located at the proxy level. So it can intercept and analyze requests and response pages between browser and remote server as it shown in figure 2.5. ProClick performs a number of checks based on the specified policy.

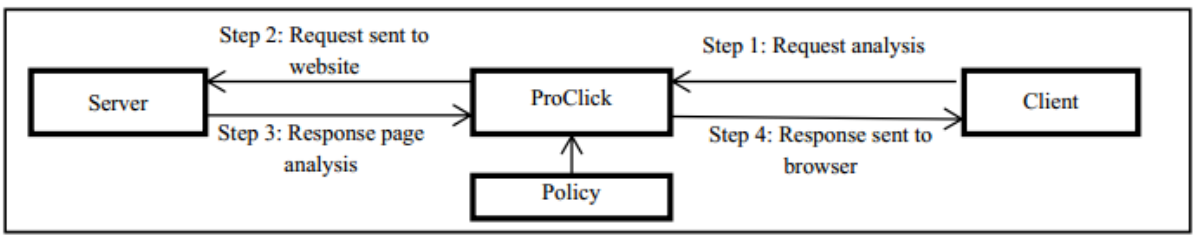


Figure 2.5 ProClick architecture system [8].

Limits:

Unfortunately, this framework requires a proxy installed on local host or on intranet, which affect the performance of the browser. Also not every user is related over proxy.

5. Conclusion

In this chapter we talked about many defense techniques that have been proposed to protect web users against clickjacking attacks. The techniques in server side are defeated with advanced clickjacking attacks, also recent research shown [16] that the most existing web sites is vulnerable to clickjacking attacks.

We have seen several mitigations techniques are implemented to combat clickjacking attacks in client side and we show its limits.

So we confirm the weakness of the websites protection against clickjacking attacks (server side techniques) and limits of every proposed tool on client side such us a high false positive.

Finally, we propose a browser extension “ClickDetector”, a client side tool. ClickDetector will defeat the attacker attempt to perform clickjacking attacks by **detecting all advanced attacks techniques reported by OWASP** to minimize the false negative detection rate.

CHAPTER 03

CLICKDETECTOR EXTENSION

CHAPTER 3 CLICKDETECTOR EXTENSION

1. Introduction

After we saw the anterior works that have been done to combat clickjacking attacks in the previous chapter. In this chapter we will talk about our proposed tool to defeat attackers attempt to perform clickjacking attacks.

2. The proposed tool

In the previous server-side defense techniques they all been circumvented by malicious users and advanced clickjacking attacks. For the client-side we have notice that the mitigation to combat clickjacking attack take several ways such as detect and analyze user click elements which mean after the user click or by making the hidden iframe visible and only notify users, also there is a mitigation based only for the users feedback which it is suffer of false positive …etc., moreover using this tools it may limit browsers functionality. Also mostly, they have a high false positive with a small amount of attack vectors.

In order to combat Clickjacking attacks and avoid high false positive detections. We propose a browser extension (ClickDetector) to defeat the attacker attempt to perform clickjacking attacks by **analyzing requests** and **responses** from websites to **detect all advanced attacks techniques reported by OWASP.**

We have also include Extension feedback, where our extensions could report malicious websites in remote database (Suspiciouslist), to make future interaction more informed for new users.

We also adopted **Google Safe Browsing service** to identify unsafe websites and notify users to protect themselves from harm (malwares, phishing …).

3. ClickDetector system

Figure 3.1 shows the flowchart of the act of ClickDetector between the interactions of server and google chrome browser. ClickDetector **analyzing requests** and **responses** to **detect all advanced attacks.**

Also it illustrate that our system performs a number of checks, it checks local user cache and blacklist table for the malicious websites that contain clickjacking attacks, and send feedback to

report malicious websites in suspiciouslist table, plus the system also adopted **google safe browsing service** to check in really big global database offered by google that contain the most **malicious websites** (Malware, fishing, social engineering).

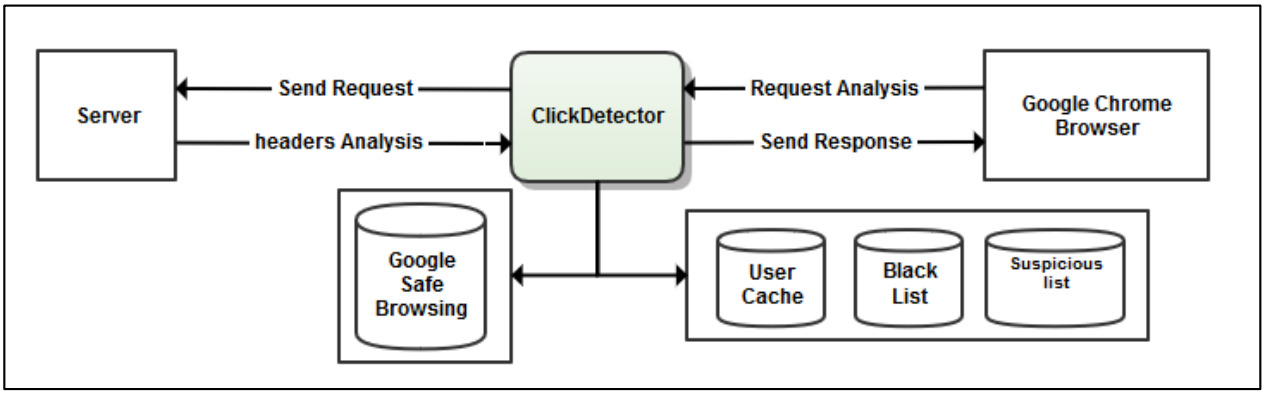


Figure 3.1 ClickDetector working system flowchart.

ClickDetector has three parts:

1- **Request analysis**: ClickDetector will intercept the request sent by the browser to check for:
   - Check in local user cache.
   - Check in remote databases (Blacklist).
   - Check in google safe browsing database.
   - Check for Reflective XSS Attacks.

2- **headers Analysis** (OnheadersReceived): ClickDetector will Intercept headers to check for unloading page to check for:
   - OnBeforeUnload attack.
   - No-Content attack.

3- **webpage analysis** (windowload): when page is loaded ClickDetector will check for:
   - Invisible frames.
   - Restricting JavaScript attack.
   - Cursorjacking attack.

3.1. Request Analysis

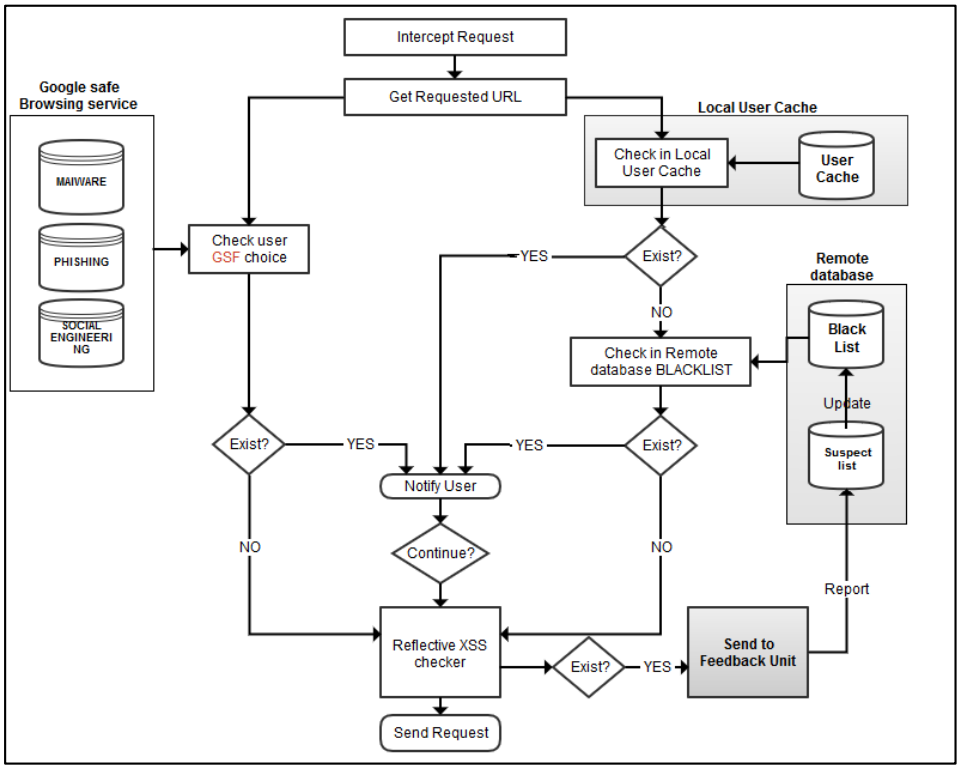


Figure 3.2 Request analysis flowchart.

In figure 3.2 we ilustrate how ClickDetector intercept the request sent by the browser. First of all ClickDetector will get requested url then it will check in:

- ✓ **User Local cache:** if the url exist in user local cache that mean user visited that website before and it is contain clickjacking attacks, then user must be notified. Else if the url is not exist in local cache it will check in blackList table.
- ✓ **Remote BlackList:** this table contains list of malicious websites that is vulnerable to clickjacking attacks (malicious URLs) reported by extension feedback after a number **K** of URL reported on the suspiciouslist table.
- ✓ **Google safe browsing:** if the user enable google safe browsing service for more protection, then clickDetector will sent the url for check in google safe brossing databses.

In case where the user have been notified and he choose to contiue, ClickDetector will check for Refelected XSS attacks using Reflective XSS checker.

3.1.1. Reflective XSS checker

In this module will check for reflective XSS in requested URL for all iframes exist in webpage. Reflective XSS is classify into five types (OWASP, 2015c) [23]:

- Partial script tag with partial frame busting code:

http://www.xyz.com/index.php?q=<script>if

- Partial script tag with complete frame busting code:

http://www.xyz.com/index.php?q=<script>if(top.location!=self.location) {top.locationn=self.location;}

- Full script tag with partial frame busting code:

http://www.xyz.com/index.php?q=<script>if </script>

- Full script tag with complete frame busting code:

http://www.xyz.com/index.php?q=<script>if(top.location!=self.location){ top.location=self.location;}</script>

- Full script with non-frame busting code:

http://www.xyz.com/index.php?q="><script>alert(document.cookie)</script>

After we review all types of relective xss [23] plus the most comon counter-action statement [7] that can elmienate the excution of frame busting code in victim website, here is the most keywords it may exist in url to detected as reflective xss attack:

{"script", "if", "self", "top", "location", "parent"}

In case This modele match this keywords in frames url, it will remove the malicious iframe plus send a message to feedback unit (notify user+ insert into user local cache and remote suspiciouslist table).

3.2. Headers analysis

3.2.1. OnBeforeUnload checker

In this module ClickDetector will check for OnBeforeUnload and No-Content clickjacking attacks. Figure 3.3 shows the flowchart of OnBeforeUnload checker.

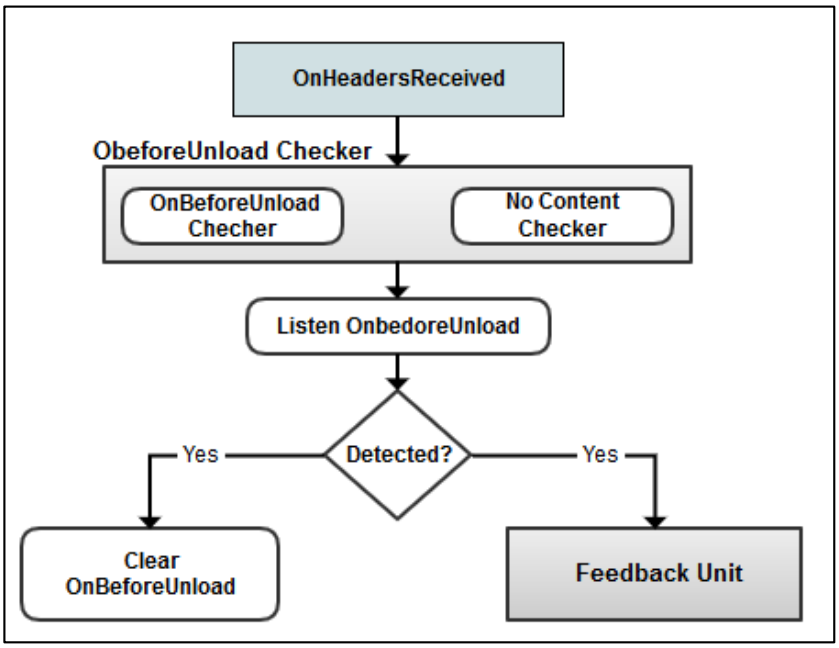


Figure 3.3 OnBeforeUnload checker flowchart.

OnBeforeunaload and No-Content of advanced clickjacking attacks we notice both of them are based on OnBeforeUnload JavaScript object (see chapter 01).

OnBeforeUnload checker will listen then clear OnBeforeUnload object while the page is loading, if it detected send message to feedback unit to notify user and report the malicious webpage.

3.3. Webpage analysis

Figure 3.5 shows the flowchart demonstrating working when the webpage is loaded. This modules will check for Invisible frames (Multiple iframe + basic clickjacking attack), Restricting JavaScript attack, and Cursorjacking attack.

Also it demonstrating working of feedback unit (extension feedback), by notify user and how it report the malicious webpage in local user cache and suspiciouslist.

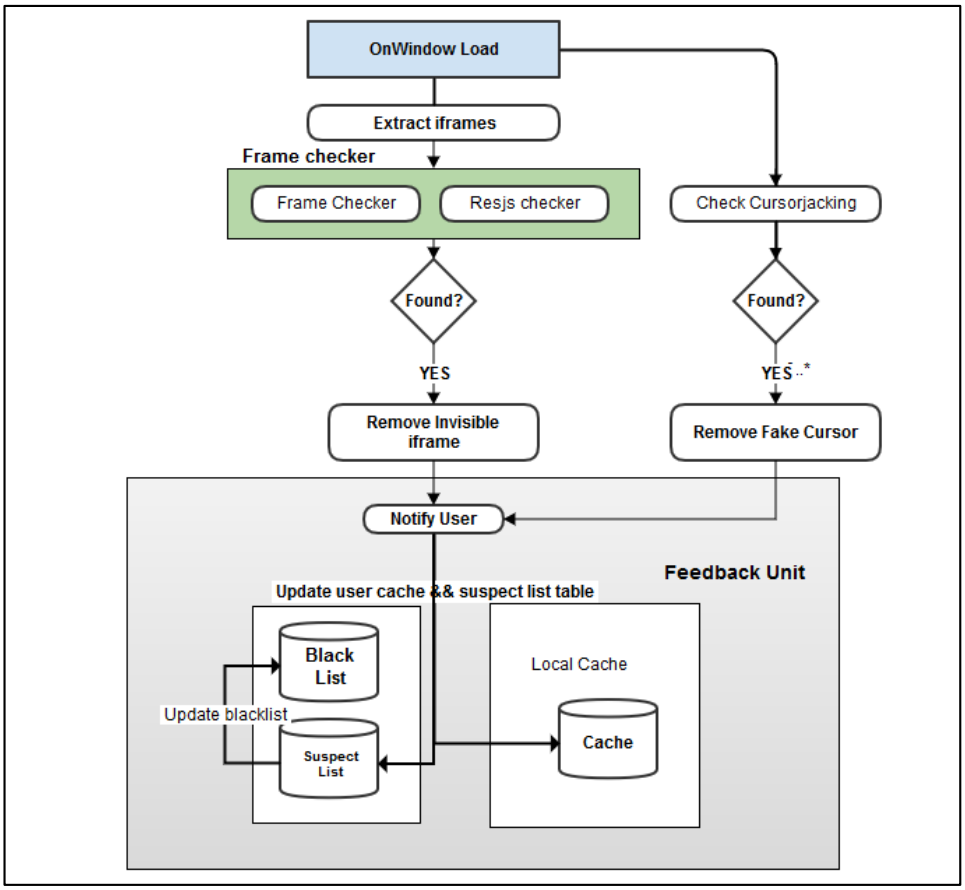


Figure 3.4 webpage analysis flowchart.

3.3.1. Frame checker

Figure 3.5 shows frame checker flowchart. This module will extract all frames exist in webpage to check its domains with webpage domain:

If were not the same, ClickDetector will examine the CSS features, by examine the CSS style of the frames or in div tag surrounding the iframe, either in inline style attribute, then get the opacity for check:

If opacity < 0.1, the page is considered to have symptoms of clickjacking attack (invisible iframe), ClickDetector will remove the invisible iframe.

The iframe tag is also analyzed for the presence of restriction zone on JavaScript code. This step detects restricting JavaScript attacks.

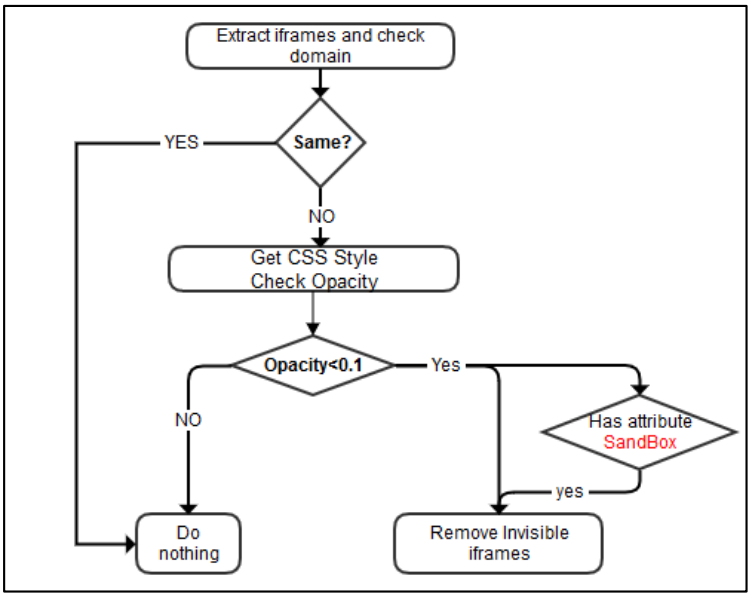


Figure 3.5 Frame checker flowchart

3.3.2. Cursorjacking checker

This module analyzes JavaScript and CSS code to identify mouse properties to identify the symptom of Cursorjacking attack. The checker illustrated in Algorithm bellow:

- Get cursor properties
- If cursor style is none.
- Create element ‘style’.
- Set innerText to 'body {cursor: default ;}'.
- Notify user.

3.4. Databases

3.4.1. Remote suspiciouslist table:

The suspiciouslist is a table that contains suspicious URLs addresses and their notification number. Extension can add new URLs to suspiciouslist when our ClickDetector reports the URL as suspicious, after K notifications of the same URLs, the URL is reported in the blacklist, and its corresponding entry in the suspiciouslist is deleted.

Suspiciouslist table contain four columns and trigger:

ID: every page has a unique id.

DOMAIN: it contain the domain of the suspicious URL.

PAGE: which is the URL of the vulnerable webpage reported.

NMB: it count how many extension feedback reported the existing URL. It incremented after report the same URL from different ClickDetector extensions.

Trigger:

Is used to insert the most malicious webpage URLs that have been reported in suspiciouslist in blacklist table.

It executed after suspiciouslist are updated, if the new NMB of the existing updated URL are passed to 10, trigger will insert that webpage URL in blacklist table to report it as serious clickjacking attack, trigger algorithm is on figure 3.6.

```
CREATE [ OR REPLACE ] TRIGGER Report
BEFORE UPDATE
ON suspiciouslist
BEGIN
        IF NEW.NMB=10 THEN
                INSERT INTO blacklist (ID, Dm, PG) values (OLD.ID,
                OLD.DOMAIN,OLD.PAGE);
        END IF
END;
```

Figure 3.6 Report trigger algorithm.

3.4.2. Remote blacklist table:

Blacklist is a table that contains suspicious URLs. This URLs is reported automatically from suspiciouslist (using trigger).

4. Conclusion

we have shown the different parts of ClickDetector system and it's techniques to defeat the attacks attempt to perform the advanced clickjacking attacks, in the next chapter we will show how we implement ClickDetector , the tool we use and we will discuss the results of the experiments we have made.

CHAPTER 04

IMPLEMENTATION AND EXPERIMENTATION

CHAPTER 4 IMPLEMENTATION AND EXPERIMENTATION

1. Introduction

This chapter describes the experimental study of our implemented extension. In which our goal is implementing a browser extension which detect advanced clickjacking attacks.

We conducted a number of experiments to study and measure the performance of our extension. A series of different advanced attacks has been tested, to achieve this objective, our extension (ClickDetector) is tested with 200 pages of white list Websites, extracted from Alexa top 100 websites to identify false positive generating by ClickDetector.

To measure the false negative detection rate of our extension, ClickDetector is tested with malicious Websites that we developed which contain advanced clickjacking attacks, and with Burp Clickbandit tool to generate clickjacking attacks on online websites.

2. Language and tools used to develop

2.1. Google chrome Browser:

Chrome is freeware web browser developed by google in 2008. Released for windows then it ported to Linux, macOS, ISO, Android.

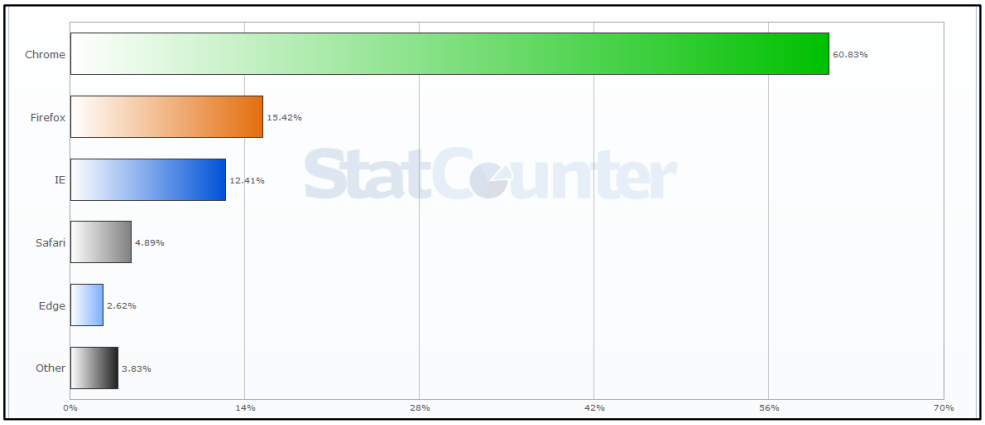


Figure 4.1 Most popular web browser (NOV 2015- MARCH 2017) [24].

Currently, Google Chrome is the most popular and widely used browser, In March 2017, StatCounter [24] estimates that Google Chrome has a 60.03% worldwide usage share of web

browsers as a desktop browser. As it illustrated in figure 4.1 below google chrome browser in the first place then Firefox and internet explorer in the second and the fourth place.

2.2. What is browser extension or add-on?

Extensions are small software programs that can modify and enhance the functionality of your browser. You write them using web technologies such as HTML, JavaScript and CSS [18].

Chrome extensions have a strong focus on extending the browser's functionality. It is used for different reasons and extension types are assigned different tasks, for example to customize the visual appearance of the browser or add new languages and location-specific display information to the UI.

2.3. Extension Architecture

Chrome extension has different parts as it demonstrated in figure 4.2.

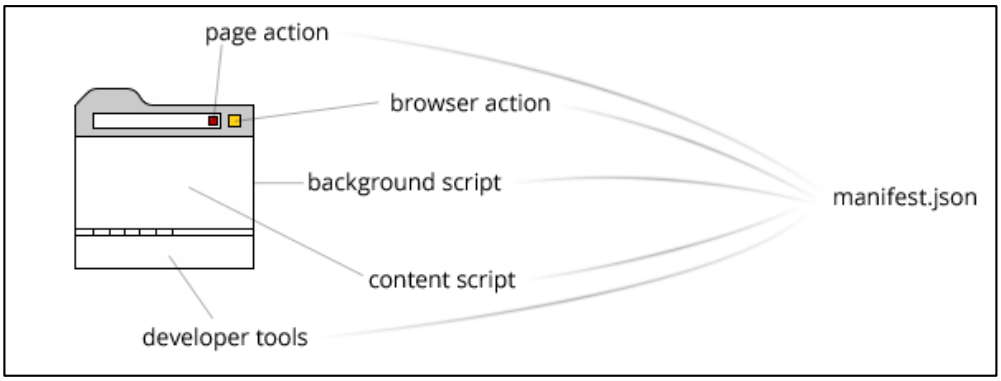


Figure 4.2 Google chrome extension architecture system [18].

2.3.1. Manifest file

The manifest file, called **manifest.json**, gives information about the extension, such as the most important files and the capabilities that the extension might use. It should at least contain bellow JSON Objects in figure 4.3:

```
{
   "name": "Extensionname",
   "version": "0.0.1",
   "manifest_version": 2,

"browser_action": {
       "default_title": "That's it",
       "default_popup": "popup.html"
   }
   "background": {
   "scripts": ["background.js"],
   "persistent": false/true
   },
   "content_scripts": [
   {
       "matches": ["http://*/*", "https://*/*"],
       "js": ["content.js"]
   }
   ]
}
```

Figure 4.3 Manifest file format.

The manifest could contain other properties depending on what kind of extension you need.

2.3.2. Background PAGE:

Every extension has an invisible background page which is run by the browser. It plays the role of bridge between the other parts of the extension which is has two type:

- **Persistent background pages:** it active all of the time, and the persistent attribute is true.
- **Event page:** is active only when it needed, and persistent attribute should be false in that case.

2.3.3. Content script:

Content script is the only script could access and made changes to the current page's DOM. The code is run with the current page and executed with every refresh.

2.4. Google safe browsing:

Safe Browsing is a Google service that lets client applications check URLs against Google's constantly updated lists of unsafe web resources [19]. Examples of unsafe web resources are

social engineering sites (phishing and deceptive sites) and sites that host malware or unwanted software.

2.4.1. Google safe browsing advantage

With Safe Browsing you can:

- Check pages against our Safe Browsing lists based on platform and threat types.
- Warn users before they click links in your site that may lead to infected pages.
- Prevent users from posting links to known infected pages from your site.

Clients can check if a URL is safe or unsafe using either Lookup API (v4) or the Update API.

2.4.2. Lookup API (v4)

The Lookup API [19] is an experimental API that enables applications to send URLs to the Safe Browsing service and check their status (e.g. phishing, malware, unwanted software).

In ClickDetector we used the Lookup API because:

- It is Simple to implement: we can send a HTTP GET or POST request with the URLs, and the server responds with the state of the URLs.

2.4.3. Update API (v4)

The Update API lets client applications download encrypted versions of the Safe Browsing lists for local. The Update API is designed for clients that require high frequency [19], low-latency verdicts. Several web browsers and software platforms use this API to protect large sets of users.

In ClickDetector we have used Lookup API (v4), it send the requested URL to the Safe Browsing service for the check. In this explanation we will illustrate the steps to how we implement this service with ClickDetector extension:

2.4.4. Set up an API key

To access to the safe browser APIs we set up and activate our own API key to authenticate as an API user to allow us interact with the APIs. The API key is passed as a URL parameter in the HTTP request to the Safe Browsing service. Here the URL with our API key:

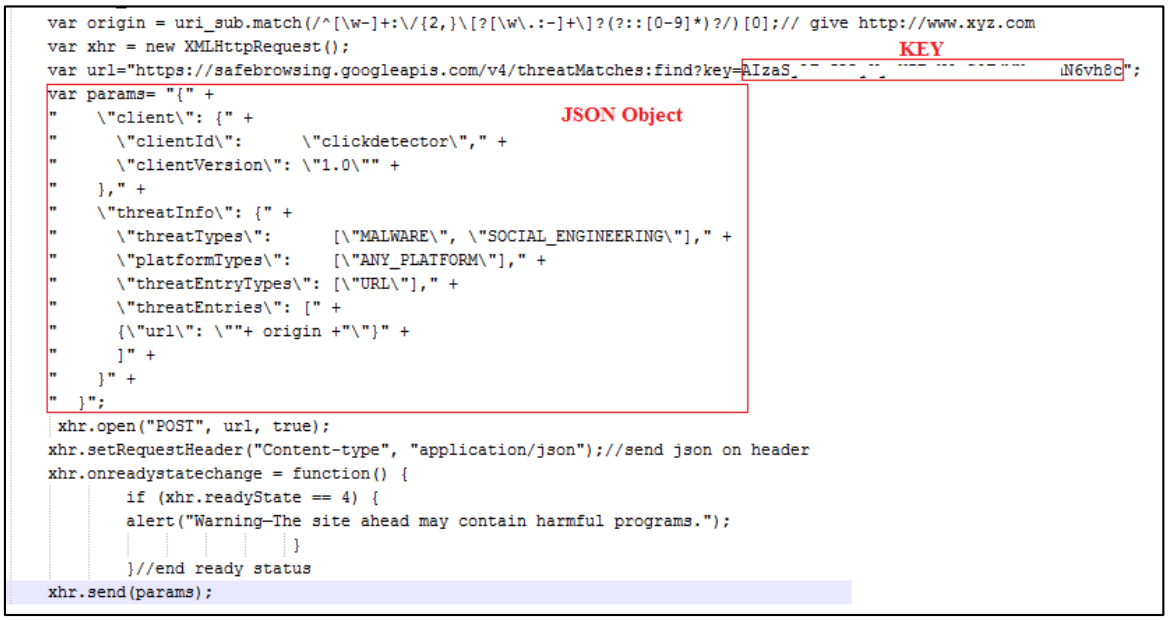
https://safebrowsing.googleapis.com/v4/threatMatches:find?key=AIz*******ycYI

2.4.5. Checking URL

To check URL on Safe Browsing list, we have send an HTTP **POST** request to the Safe Browsing server. We have used XHR (XmlHttpRequest) object to request data From Safe Browsing server.

The HTTP POST request header includes the request URL and the content type which is a JSON format.

The HTTP POST request body includes our information (ID and version) and the URL want to check for. The code is shown in figure 4.4.




```
var origin = uri_sub.match(/^[\w-]+:\/{2,}\[?[\w\.:-]+\]?(?::[0-9]*)?/)[0];// give http://www.xyz.com
var xhr = new XMLHttpRequest();
var url="https://safebrowsing.googleapis.com/v4/threatMatches:find?key=AIzaS_ -- --- - - --- -- --- ---  iN6vh8c";
var params= "{" +
"    \"client\": {" +
"      \"clientId\":      \"clickdetector\"," +
"      \"clientVersion\": \"1.0\"" +
"    }," +
"    \"threatInfo\": {" +
"      \"threatTypes\":      [\"MALWARE\", \"SOCIAL_ENGINEERING\"]," +
"      \"platformTypes\":    [\"ANY_PLATFORM\"]," +
"      \"threatEntryTypes\": [\"URL\"]," +
"      \"threatEntries\": [" +
"      {\"url\": \""+ origin +"\"}" +
"      ]" +
"    }" +
"  }";
 xhr.open("POST", url, true);
xhr.setRequestHeader("Content-type", "application/json");//send json on header
xhr.onreadystatechange = function() {
        if (xhr.readyState == 4) {
        alert("Warning—The site ahead may contain harmful programs.");
                }
        }//end ready status
xhr.send(params);
```

Figure 4.4 Lookup API HTTP post request code.

2.5. Local user cache:

We have used localStorage API in google chrome browser to store extension data locally (store reported URLs). To make future interaction faster, because data are stored locally which is easy to access and faster.

The following figures illustrate the code used to store and retrieve from local user cache using local chrome storage using JavaScript:

```
function insertintocache(){
        var dd=true;
        chrome.tabs.getSelected(null,function(tab) {
                var pg=tab.url;
                var dom = pg.match(/^[\w-]+:\/{2,}\[?[\w\.:-]+\]?(?::[0-9]*)?/)[0];// give http://www.xyz.co
                chrome.storage.sync.get('urllist', function(items) {
                if(items.urllist== undefined){
                                var urltab=[];
                                urltab.push({domain: dom, page: pg});
                                chrome.storage.sync.set({'urllist': urltab}, function() {// callback body
                                });//end set
                                //insert into db
                                insert(dom, pg);
                }else{
                     for(i=0;i<items.urllist.length; i++)  {
                        if(items.urllist[i].page== pg) {
                            dd=false;
                        }
                }
```

Figure 4.5 Insertion into local cache function code.

Figure 4.5 illustrate the code used to insert into browser local cache. We have create a table called **urltab** that could store objects ({domain: domain, page, page}) page and its domain, using chrome storage synchronization commands (**set** and **get)** to store and retrieve data from the local cache.

2.6. Remote database

ClickDetector remote database contain tow tables, suspicious list that contain the reported URLs from feedback extension and the blacklist table that contain the top reported URLs.

We have used phpMyAdmin to create ClickDetector database and deploy it on Apache server, so we used php to insert and retrieve from the remote database.

3. Interfaces

3.1. ClickDetector user interface

ClickDetector is an extension that works on browser background with simple interface. We only need user's choice for using google safe browsing service or not as input (checkbox), as it illustrated in figure 4.6.

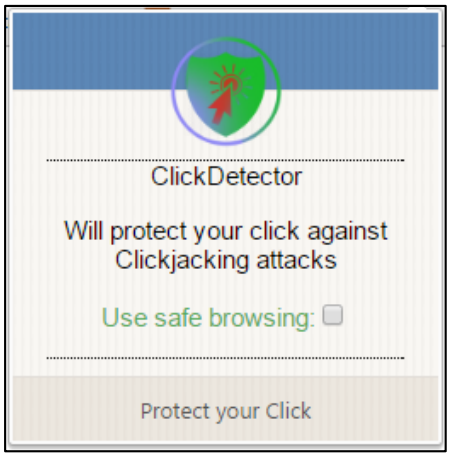


Figure 4.6 ClickDetector user interface.

3.2. ClickDetector Notifications module:

Is used to notify user, that the malicious element is removed successfully.

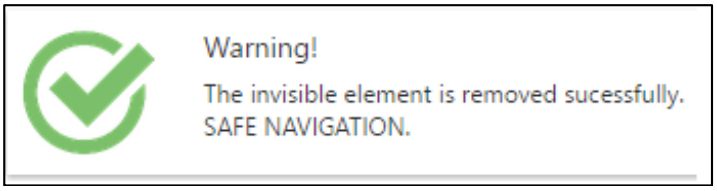


Figure 4.7 ClickDetector notification module.

3.3. ClickDetector alerts

ClickDetector generate alerts and promoting users interaction to close or continue, this three cases when the users will be alerted:

- **Blacklist alert:** it alert user when the requested URL exist in blacklist table:

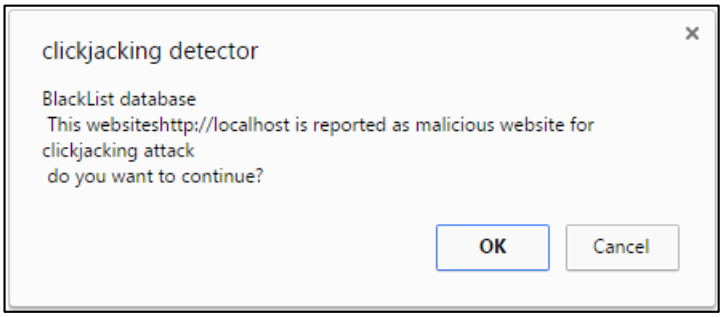


Figure 4.8 Alert user (blacklist).

- **Cache alert:** it alert user when the request URL exist in local user cache:

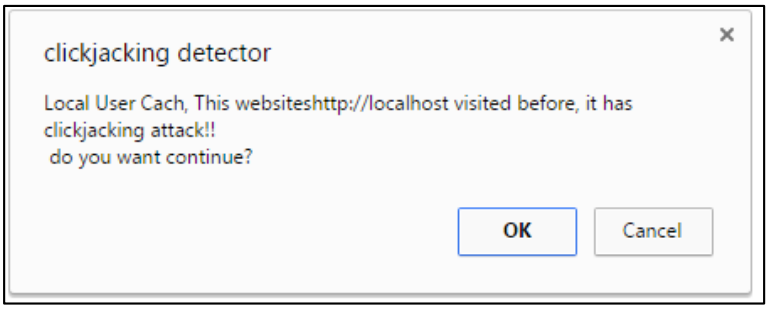


Figure 4.9 Alert user (local user cache).

- **Reflective XSS alert:** it alert user while the requested URL contain symptom of clickjacking reflective XSS attack:

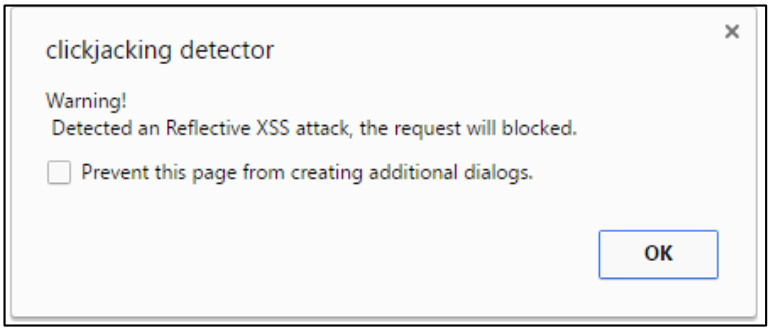


Figure 4.10 Alert on reflective XSS attack.

4. Experimentation and discussion:

4.1. Testing with white list Websites:

We used websites from Alexa[14](a well-known source of legitimate website domain names used in other related research Lekies et al 2012, Hossain Shahriar 2015[8]). We have choose top 100 websites from Alexa in April – May 2017, for every website we visit top page and randomly another inner page which make in total 200 visited pages.

We have used that white list to check if our proposed tool will generate any warning on possible clickjacking attack to identify its false positive rate.

The results shown that we got **no false positive** over 200 pages visited.

Here is characteristics of top 10 websites visited from white list:

Websites	Presence iframe	Number of iframes	Minimum opacity	Frame busting	Restricting JavaScript	OnBeforeUnload	Cursor change
Twitter.com	Yes	8	0.8	No	No	No	No
Amazon.com	Yes	3	0.3	Yes	No	Yes	No
Yahoo.com	Yes	6	0.4	No	Yes	Yes	No
Microsoft.com	Yes	4	0.8	No	No	No	No
paypal.com	No	0	No	Yes	No	No	No
Imdb.com	Yes	9	0.55	No	No	Yes	No
chase.com	Yes	2	0	No	No	No	No
Qq.com	Yes	7	1	No	Yes	Yes	No
Youtube.com	Yes	4	1	No	No	No	No
Wordpress.com	No	No	No	No	No	No	No

Table 1 top 10 white list sites.

4.2. Testing with black list (malicious) Websites

Burp Clickbandit [25] is a tool used for generating clickjacking attacks on online websites, we randomly testing on 100 Algerian websites (banks, government, universities, institutes … etc.). The test is done by injecting script generated by Burp on visited sites.

The results show that ClickDetector successfully detect every attack generated by Burp Clickbandit. We also got 96 of 100 Algerian websites are vulnerable to clickjacking attacks.

Burp Clickbandit can't generate all advanced clickjacking attack , for this reason we developed webpages contain advanced clickjacking attacks where each of them includes just one malicious web page ,and deployed them in an Apache web server, enabled ClickDetector, and then visited each of the pages from the seven websites containing malicious iframes.

In figure 4.11 below shows the webpage that contain links to the six malicious webpages contain advanced clickjacking attacks we developed.

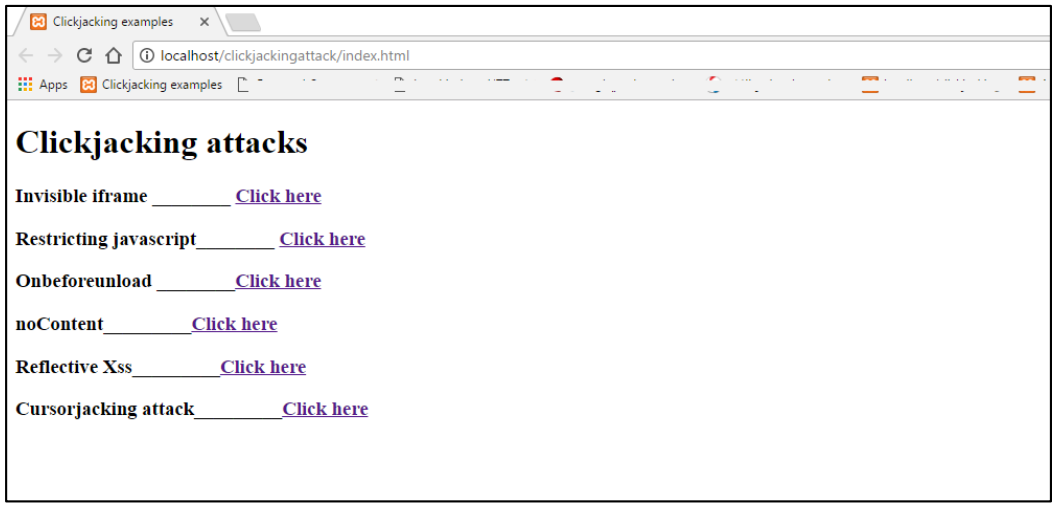


Figure 4.11 Advanced clickjacking attacks webpage.

ClickDetector successfully detected clickjacking attacks in all malicious web pages. Which means that the **false negative** rate is zero.

5. Performance:

We visited several webpages from white list sites with and without using ClickDetector extension then we have used google chrome inspect to measure webpage loading time.

Figure 4.12 shows graph of average delay time of loading webpage (in seconds) measured by webpage size (bytes).

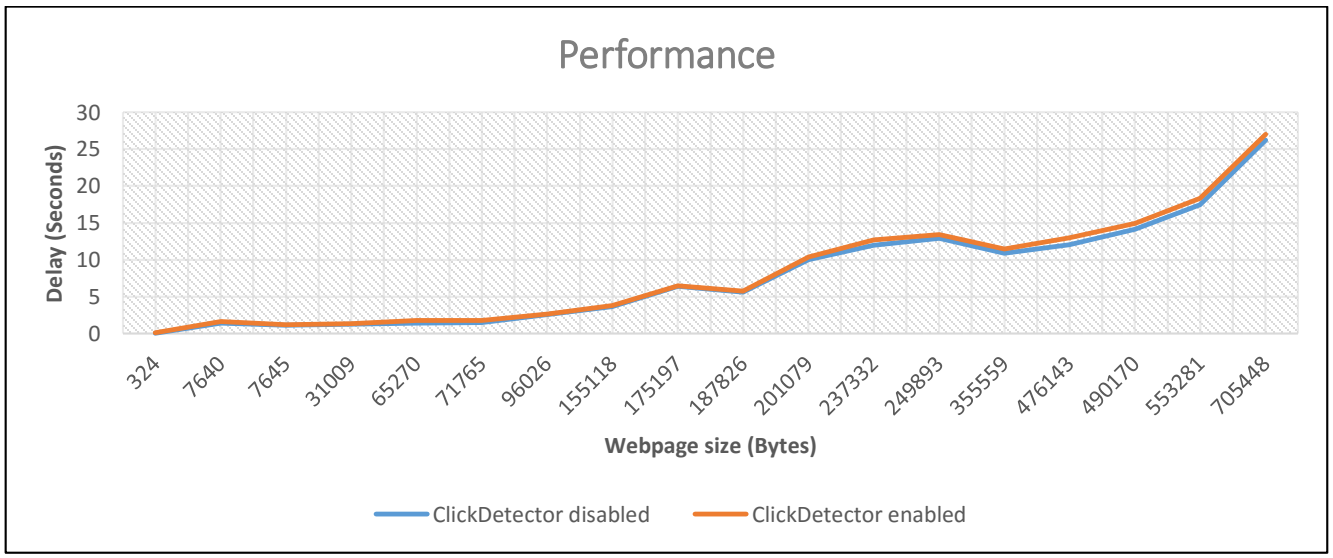


Figure 4.12 Performance graph.

There are two graphs in the chart. The blue graph shows loading delay time when ClickDetector is disabled, rather the orange one when ClickDetector is enabled. We can say that the rise of the average loading time is simultaneously, and the highest difference time between them is 0.88 seconds.

So we can say that ClickDetector effect on browser performance is negligible.

6. Conclusion:

In this chapter we present the implementation and experimentation of our ClickDetector extension.

The extension was tested by determining how well it picks up a large variety of advanced clickjacking attacks.

The experimental study clearly shows the efficiency of our extension, the results proves the high detection of advanced clickjacking attacks with low false positive rate.

General Conclusion

General conclusion

Clickjacking is an attack that tricks the victims into clicking on invisible elements of a web page to perform unwanted action which are beneficial for the attacker.

We investigated how Algerian websites are vulnerable to clickjacking attacks by performing real attacks on 100 Algerian websites (Universities, banks, governments, companies …). The results are quite significant that almost all Algerian websites (96%) are vulnerable to clickjacking attacks.

In this project, we developed a google chrome extension "ClickDetector" to defeat the attacker attempt to perform clickjacking attacks by **analyzing requests** and **responses** from websites to detect all the advanced clickjacking attacks techniques reported by OWSAP.

ClickDetector have three basic phases to detect this advanced attacks:

- **Request analysis**: Intercept the request to check the requested URL in databases to warn the users of the malicious web pages and Reflective XSS attacks.
- **Headers Analysis:** Intercept header to check for unloading page to detect OnBeforeUnload and No-Content attacks.
- **Webpage analysis:** Once the page is loaded, ClickDetector will check for the hiding iframes (Multiple iframes attack, Restricting JavaScript attack), also to protect against Cursorjacking attack.

ClickDetector also based on extensions feedback to make future interaction to be more informed for new users.

In the experimentation phase, we have tested ClickDetector with 200 benign web pages from Alexa (white list), and 100 malicious webpage (using Click Burp to generate online clickjacking attacks on 100 Algerian websites).

Burp Clickbandit can't generate all advanced clickjacking attacks, for this reason we developed webpages contain advanced clickjacking attacks where each of them includes just one malicious web page.

The obtained experimental results demonstrate that our ClickDetector extension successfully detects all the generated attacks with no false positive , this proves the effectiveness of our extension to detect advanced clickjacking attacks and protect users from harm without effect on browser performance.

References

References

Books:

[1] D. Kavitha, S. Chandrasekaran and S.K. Rani, HDTCV: Hybrid Detection Technique for Clickjacking Vulnerability, Springer India, 2016.

[2] G. Maone. ClearClick: Effective Client-Side Protection against UI Redressing Attacks, May 3, 2012.

[3] Leon, S and Rechard, R, Web Application Architecture: Principles, protocols and Practices, England, 2003.

[4] R. Fielding, U. Irvine, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, Hypertext Transfer Protocol -- HTTP/1.1, The Internet Society, June 1999.

Articles:

[5] A. Shamsi, S. Hameed, W. Rahman, F. Zuberi, K. Altaf, A. Amjad, Clicksafe: providing security against clickjacking attacks. IEEE 15th International Symposium on High-Assurance Systems Engineering. C.S Dept., National University of Computer and Emerging Sciences – Karachi Pakistan. 2014.

[6] Daehyun. Kim, Hyoungshick Kim, Performing clickjacking attacks in the wild: 99% are still vulnerable, IEEE, Department of Computer Science and Engineering Sungkyunkwan University, Republic of Korea, 2016.

[7] G. Rydstedt, E. Bursztein, D. Boneh, C. Jackson, Busting Frame Busting: a Study of Clickjacking Vulnerabilities on Popular Sites. Stanford University, Procceding of the IEEE Web 2.0 Security and Privacy Workshop, W2SP 2010.

[8] H. Shahriar, H. Haddad, V. Devendran, A framework for testing clickjacking attacks in web applications, International Journal of Secure Software Engineering, Volume 6, July 2015.

[9] Huang LS, Moshchuk A, Wang HJ, Schecter S, Jackson C. Clickjacking: Attacks and Defenses. USENIX Security Symposium 2012 Aug 8 (pp. 413-428).

[10] M. AL-Sanea, A. Al-Daraiseh, Security Evaluation of Saudi Arabia's Websites Using Open Source Tools, IEEE, College of Computer Science and Information King Saud University Riyadh, Saudi Arabia, 2015.

[11] M. Balduzzi, M. Egele, E. Kirda, D. Balzarotti, and C. Kruegel, "A solution for the automated detection of clickjacking attacks," Proc. of the 5th ACM Symposium on Information, Computer and Communications Security, Beijing, China, April 2010, pp. 135-144.

[12] Prof. D. Pawade, A. Lahigude, D. Reja, Prof. E Johri, Implementation of Extension for Browser to Detect Vulnerable Elements on Web Pages and Avoid Clickjacking, IEEE. 2016.

[13] R. P. Seenivasan, K. Suresh Joseph, A Survey of Clickjacking Attack and Countermeasures in Web Environment, International Journal of Advanced Research in Computer Science and Software Engineering, Volume 6. December 2016.

Websites:

[14] Alexa, www.alexa.com/topsites, Accessed: 03/05/2017.

[15] Arokiait, http://www.arokiait.com/3-tier-web-architecture.htm, Accessed : 22/02/2017.

[16] Blog.qualys,https://blog.qualys.com/securitylabs/2012/11/29/clickjacking-an-overlooked-web-security-hole, Accessed 03/11/2016.

[17] Codemagi, https://www.codemagi.com/blog/post/194, Accessed : 14/01/2017.

[18] Developer. Chrome, https://developer.chrome.com/extensions, Accessed 28/04/2017.

[19] Developers. Google, https://developers.google.com/safe-browsing, Accessed: 02/03/2017.

[20] Erlend.oftedal.no,http://erlend.oftedal.no/blog/tools/xframeoptions. Accessed: 27/05/2017.

[21] NoScript, https://noscript.net, Accessed: 05/02/2017.

[22] OWASP, https://www.owasp.org/index.php/Clickjacking, Accessed: 17/03/2017.

[23] OWASP,https://www.owasp.org/index.php/Clickjacking_Defense_Cheat_Sheet,Accessed 24/02/2017.

[24] OWASP,https://www.owasp.org/index.php/Testing_for_Clickjacking_(OTG-CLIENT009)# Client_side_protection:_Frame_Busting, Accessed: 03/03/2017.

[25] Portswigger,https://portswigger.net/burp/help/suite_functions_clickbandit.html, Accessed 22/04/2017.

[26] StatCounter,http://gs.statcounter.com/#all-browser-ww-monthly-201612-201612-bar, Accessed 22/05/2017.

[27] Tutorials Point. https://www.tutorialspoint.com/HTTP-hypertext_transfer_protocol, Accessed 08/05/2017.

الملخص:

هجوم خطف النقرات هو هجوم لخداع الضحية للنقر على شئ مخفي في صفحة الويب لتنفيد أوامر غير مرغوب فيها و التي تصب لصالح منفذ هذا الهجوم. العديد من الأبحاث بينت أن هجوم خطف النقرات يعد مصدر أساسي لعدة هجومات نذكر منها : الـ Cross Site Scripting او ما يُختصر بـXSS و هجوم CSRF. في هذه الدراسة نقترح ClickDetector, و التي هي تطبيق على متصفح كروم تعمل على إبطال محاولة الهاكر لتنفيد هجوم خطف النقرات و ذلك بكشف جميع أنواع الهجومات المتقدمة المذكورة في موقع OWASP من خلال تحليل طلبات و الردود صفحات الواب, بالإضافة إلى ذلك ClickDetector قائم على التغذية الرجعية (Feedback) لجعل المستخدمين الجدد أكثر اطلاعا على مدى سلامة الصفحات المطلوبة. المستخدمين أيضا قادرون على اختيار خدمة جوجل التصفح الآمن google safe browsing لمزيد من الحماية. النتائج التجريبية التي تم الحصول عليها تثبت أن تطبيق ClickDetector يكشف بنجاح كل الهجمات التي تم إنشاؤها وهذا يثبت فعالية التمديد دون أي تأثير على أداء المتصفح.

الكلمات المفتاحية:Clickjacking, Frame Busting, iframe, ClickDetector, Cursorjacking.

Abstract:

Clickjacking is an attack that tricks the victims into clicking on invisible elements of a web page to perform unwanted action which are beneficial for the attacker. Many recent researches have shown that clickjacking is the primary source of different exploitations such as cross site request forgery and phishing. In this project we proposed ClickDetector, a chrome extension to defeat the attacker attempt to perform clickjacking attacks by **detecting all advanced attacks techniques reported by OWASP** by analyzing requests and responses**,** also ClickDetector based on extension feedback to make future interaction to be more informed for new users. Users also are able to choose google safe browsing service for more protection. The obtained experimental results demonstrate that our ClickDetector extension successfully detects all the generated attacks with no false positive, this proves the effectiveness of our extension without any effect on browser performance.

Keywords: Clickjacking, Frame busting, iframe, ClickDetector, Cursorjacking.