UI Redressing: Attacks and Countermeasures Revisited

Marcus Niemietz
marcus.niemietz@rub.de

Seminar Work

at

Chair for Network and Data Security
Prof. Dr. Jörg Schwenk

advised through Dipl.-Ing. Mario Heiderich

12.01.2011

Horst-Görtz Institute Ruhr-University of Bochum

Contents

List of Figures, Tables and Listings

List of Figures

List of Tables

List of Listings

1. Introduction

The current era shows that a company like Google Inc. can generate a profit of over \$6.5 billion in 2009 inter alia by using web technologies [4]. This fact alone makes it especially interesting for commercial companies to offer web applications for doing online banking, shopping, or just to share status messages with friends via Twitter or other social networks like Facebook.

Particularly, with regard to the last few years, time has shown that web applications have become more complex to provide new functions, to have more usability features or just to create new eye-catchers. The higher complexity level does not say that the security level remains the same, as in the past. Therefore, software engineers have to pay attention to develop secure web applications. This thesis does not fit, for the reason that the experience of the software developer does not increase as fast as the complexity level. Thus, there are always new attacks available that can bypass existing protection mechanisms.

Robert Hansen and Jeremiah Grossman introduced such an attack in 2008 [5]. This technique is called "clickjacking" and it is also known under the previously used term "UI redressing" [6]. The relationship between both terms differs for the reason that clickjacking is a subset of UI redressing. So they are not treated equally in this paper.

A clickjacking attack can be used alone or in combination with other attacks. The combination part is particularly interesting, because it makes attacks such as CSRF possible, even though there are elements like nonces [1] available, which should prevent this kind of attack by providing freshness. A clickjacking attack forces a victim to unintentionally click on an obviously invisible web page, where such protection mechanisms are included. This is possible because the victim thinks that he or she is clicking on a seemingly harmless web page element. In reality, the victim is lured into doing exactly what the attacker wants the victim to do.

A usual prerequisite of the clickjacking attack is that the victim must be authenticated against an attackers target web page to perform actions on it. If these requirements are fulfilled, an attacker will ensnare the victim to perform at least one action on this user authenticated web page that the attacker is not allowed to perform. This consequently leads to the fact that the access control mechanism will fail and so not work as expected.

This seminar work focuses on UI redressing concerning different attack vectors and their counteractive measures. The primary goals are to understand how the attack and combinations of it work and to get knowledge about which safeguards are available. Regarding the safeguards, an automated detection system with statistics to scan web pages for clickjacking attacks will also be introduced. Last but not least, a conclusion is given with an outlook about how UI redressing can affect the future of web applications.

[1]A nonce, which is the abbreviation of "number used once", is a one-time token that can be used to assure that the corresponding event is unique. [7]

2. Attack vectors

This chapter provides information about the method of "basic clickjacking" and, particularly based on that, advanced attacks. Also introduced is the clickjacking virtualisation program "Clickjacking Tool".

2.1. Basic clickjacking

The following method is usually known under the name "clickjacking", but it is not the only type of click hijacking attack. For this reason, the attack will be called "basic clickjacking" [8].

From the perspective of an attacker, at first one has to create a web page that includes a large iframe [1] containing the target web page. The HTML code of the iframe, which is in the example inside the file "inner.html", is displayed in Listing 2.1.

Listing 2.1: Iframe code that opens the web site of "google.com" (filename: inner.html)

```
<iframe id="inner" src="http://www.google.com" width="2000" height="2000"
    scrolling="no" frameborder="none">
</iframe>
```

This iframe opens the web site of "google.com". The output of the web site interpreted by the web browser "Mozilla Firefox 3.6.3" [2] is displayed in Figure 2.1. It shows that the user is automatically authenticated against the web page with his or her name. This authentication method is a typical characteristic that an attacker is looking for.

In the second step, a new web page is created. It is exemplary called "clickjacking.html". This web page loads the web page "inner.html" with an iframe. From this it follows that a web page with an iframe is included, which also includes a web page with an iframe. However, there is a crucial difference in the HTML code of the "clickjacking.html" iframe, as shown in Listing 2.2.

Listing 2.2: Iframe and CSS code to open the web page of the file "inner.html" (filename: clickjacking.html)

```
<iframe id="inner" src="inner.html" width="2005" height="290" scrolling="no"
    frameborder="none">
</iframe>

<style type="text/css"><!--
#inner { position: absolute; left: -1955px; top: -14px;}
//--></style>
```

First of all it should be clear that the focus is on the text, or rather the link "Sign out", which is given at the top right of the web page of "google.com" or included in the "inner.html" document. A comparison

[1] An iframe element defines an inline frame for the inclusion of objects like HTML documents. It is an embedded, scrollable window that displays a web page inside another one. [9]

[2] The current version of the web browser Mozilla Firefox can be downloaded from the URL: http://www.mozilla-europe.org/en/firefox/

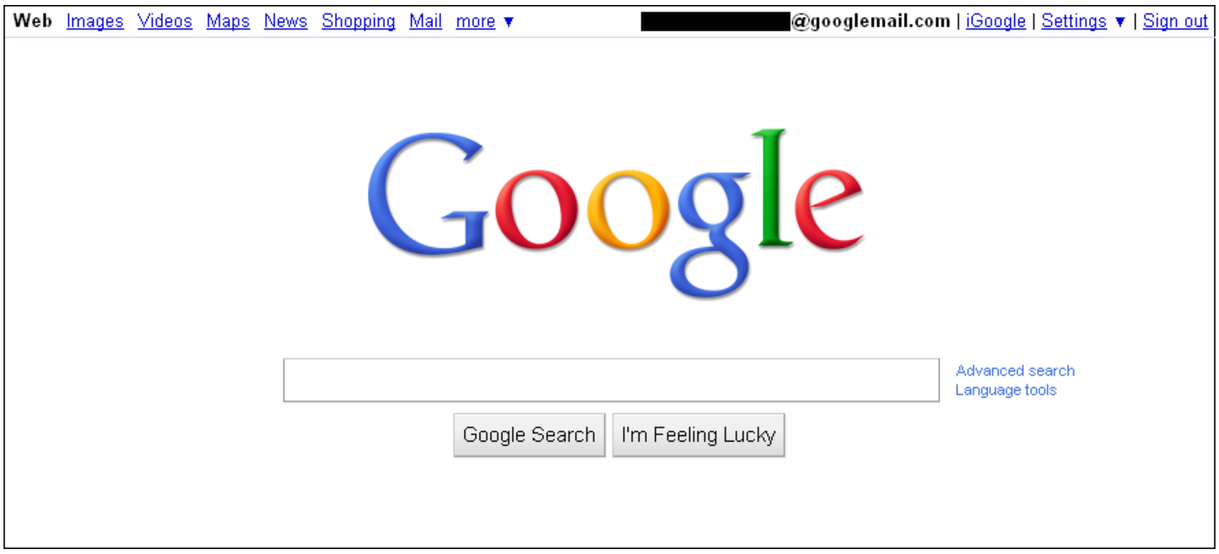

Web Images Videos Maps News Shopping Mail more ▾
[redacted]@googlemail.com | iGoogle | Settings ▾ | Sign out
Google
Advanced search
Language tools
Google Search
I'm Feeling Lucky

Figure 2.1.: Truncated and masked output of the web site “google.com” interpreted by the web browser Mozilla Firefox 3.6.3

of Listing 2.2 and Listing 2.1 shows that there is a new attribute “id” inside the “iframe” tag. The attribute “id” holds the value “inner”. This value is addressed by CSS [3] code given with the last three lines of Listing 2.2. With “#inner” is specified that the position of the element that holds the “id” value “inner” is absolute and that it should be moved 1,955 pixels to the left and 14 pixels up. Furthermore, there are new height and width specifications. These values are defined according to the layout of the web site and especially to the “Sign out” link. It should be noted that it may be an advantage to keep these values small to focus on the essential content, but in this case it does not play an important role. The web browser output of “Mozilla Firefox 3.6.3” is given in Figure 2.2.

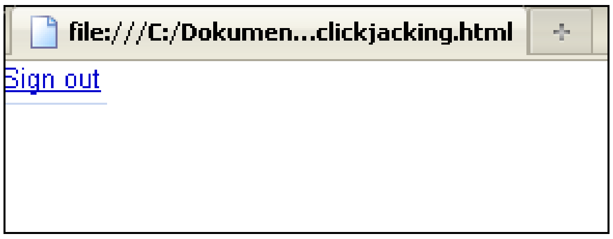


Figure 2.2.: Output of the file “clickjacking.html” interpreted by the web browser Mozilla Firefox 3.6.3

For demonstrative purposes, an illustration web page has been created. It allows, with one click on the “Go” button, to visit the web site of the “Chair for Network and Data Security”. It is used a form here to demonstrate the practical relevance. At this point it should be noted that the codes of the aforementioned Listings 2.1 and 2.2 include only the most essential HTML elements and do not correspond to a valid W3C [4] HTML structure as in Listing 2.3 [12].

Listing 2.3: HTML form web page (part of the file: trustedPage.html)

[3]Cascading Style Sheets (CSS) is style sheet language to separate a web page by its style and structure. With this it is possible to describe presentation semantics of HTML (or XML) documents. [10]

[4]The World Wide Web Consortium (W3C) is an international community that develops especially web-based standards [11]. More information can be found under the URL: http://www.w3.org

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
   <head>
      <title>Trusted web page</title>
   </head>
   <body>
    <h1>www.nds.rub.de</h1>
    <form action="http://www.nds.rub.de">
      <input type="submit" value="Go">
    </form>
   </body>
</html>
```

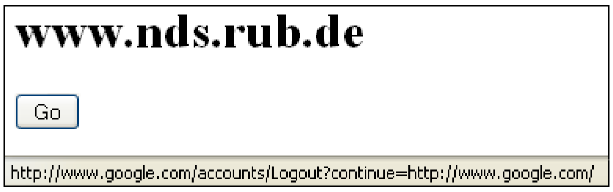
The generated HTML document is now supplemented with clickjacking code. Like Listing 2.2, Listing 2.4 also uses an iframe with CSS. This iframe loads the web page “clickjacking.html” in a 50 pixel width and 300 pixel height iframe. For this reason, a victim will click on the “Sign out” hyperlink by clicking on the “Go” button. This action is not visible to the user, because of the defined CSS property “opacity:0.0”. It makes the iframe transparent and thus hides it from the user [13]. This shows clearly that clicking on a link will forward to a “google.com” web page.

Listing 2.4: Iframe code with CSS for clickjacking (part of the file: trustedPage.html)

```
<iframe id="clickjacking" src="clickjacking.html" width="50" height="300"
    scrolling="no" frameborder="none">
</iframe>

<style type="text/css"><!--
  #clickjacking { position: absolute; left: 7px; top: 81px; opacity:0.0}
//--></style>
```

The complete code, which includes Listing 2.3 and 2.4, is displayed in the appendix in Listing A.1. The output of the web page with “Mozilla Firefox 3.6.3” is displayed in Figure 2.3. Furthermore, the status bar is shown in the web browser. It holds information about the click, because the mouse pointer is placed over the link of the current web page.



http://www.google.com/accounts/Logout?continue=http://www.google.com/

Figure 2.3.: Output of the file “trustedPage.html” interpreted by the web browser Mozilla Firefox 3.6.3

2.1.1. UI redressing

The discussed “Sign out” link, or rather the iframe of section 2.1, is, without the CSS design specification “opacity”, visible. When the link is shown directly and in another context, the clickjacking attack is called “UI redressing” [14]. This user interface (UI) redressing method is especially useful when there are buttons with nonspecific text like “Go”, “Click here” or for example “Send”. Thus, it is not always necessary to make elements invisible to the user.

Last but not least, it is important to know, as described in the introduction of this paper, that “clickjacking” is regularly used as another term for “UI redressing” [15], so that the validity of the above definition of “UI redressing” depends on the view of the observer. This work treats clickjacking as a subset of UI Redressing.

2.1.2. Strokejacking

The term “strokejacking” was introduced by Michal Zalewski in 2010 [16]. He shows in a proof of concept (PoC) that keystrokes can be, in addition to clicks, also hijacked. The PoC is given in the Appendix in Listing A.2.

Zalewski used the fact that there are web pages available that make use of the focus functionality of web browsers. His example is the search engine of “google.com”. By loading the web page, the cursor will be automatically placed into the search field and thus it will be focussed. The “strokejacking” attack uses this property by showing an iframe, which includes the search engine web page, together with the text field of the attacker. This text field is also focussed automatically. Therefore, the web browser has to choose between these two elements, or more precisely, between the “iframe” and the “input” tag. This decision takes a JavaScript code of the attacker indirectly.

In the PoC, the highlighted word “opportunity” is shown in order to provide a human authentication service. This word should be entered by the user into the text field of the attacker’s web page. The victim types this word into the text field and at the same time some of the letters in the search engine text field of the iframe, too. As shown in Listing A.2, the term of the search field will be “porn”. This word consists of letters that are inside the word “opportunity”. Thus, by using the letters “p”, “o”, “r” and “n”, the JavaScript code jumps to the search engine text field, places the letter into the field, and goes back to the attacker’s text field. By pressing “return”, the search term will be submitted through the user with the help of the above-described code.

The interesting fact is that an attacker could make the iframe transparent, as it is in the case of “basic clickjacking”, and that the victim can be used to type in e.g. malicious code without getting any notice. Therefore, it is not surprising that a few weeks after the introduction of “strokejacking”, there is an attack available that combines XSS with “strokejacking” [17].

2.1.3. Likejacking

Likejacking is a special kind of “basic clickjacking”. This term was officially introduced by the security company “Sophos” in 2010 [18]. In essence, there is no difference between “basic clickjacking” and “likejacking”, because the same method with frames is used.

The special feature is that the characteristic of social networking is taken into account. The victim, in form of a user of Facebook, has to click on the button “Like”, which is offered by Facebook as a service. By clicking on this button, a status message will be printed on Facebook and also under the web page category “Like”. So one is able to hover web pages with malicious code by using clickjacking on the “Like” button, but the code is hidden to the user.

2.1.4. Adobe Flash Player security settings

Robert Hansen and Jeremiah Grossman introduced clickjacking inter alia with an example about the “Adobe Flash Player Settings Manager” [5]. By opening the URL

- http://www.macromedia.com/support/documentation/en/flashplayer/help/settings_manager04.html

a help web page about the Adobe Flash Player is displayed. Also, the "Settings Manager" with the register card "Global Security Settings" opens automatically, if the flash player is installed. A truncated screenshot of the web page is displayed in Figure 2.4.

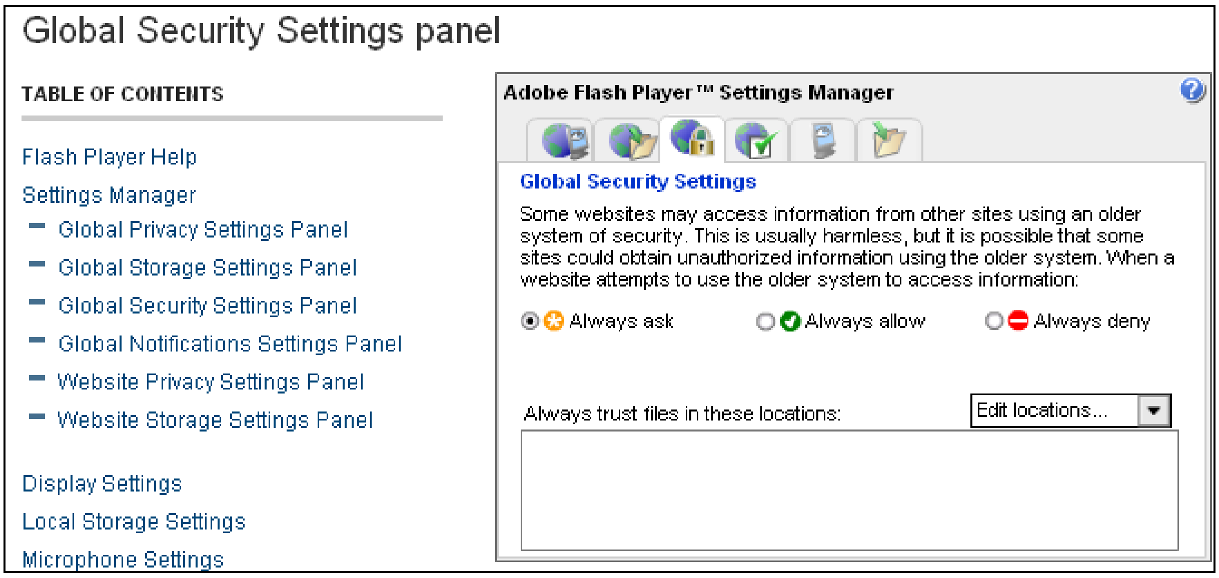

Global Security Settings panel

TABLE OF CONTENTS

Flash Player Help
Settings Manager
- Global Privacy Settings Panel
- Global Storage Settings Panel
- Global Security Settings Panel
- Global Notifications Settings Panel
- Website Privacy Settings Panel
- Website Storage Settings Panel

Display Settings
Local Storage Settings
Microphone Settings

Adobe Flash Player™ Settings Manager

Global Security Settings

Some websites may access information from other sites using an older system of security. This is usually harmless, but it is possible that some sites could obtain unauthorized information using the older system. When a website attempts to use the older system to access information:

[✓ Always ask] Always allow Always deny

Always trust files in these locations: Edit locations...

Figure 2.4.: Screenshot of the Adobe Security Settings panel

As it is described in the register card, a web page may access information from other web pages using an older system of security. For this reason, there are three radio buttons with the text "Always ask", "Always allow" and "Always deny" available. Therefore, an attacker can be interested in activating the radio button next to "Always allow". The method of this attack is analogue to "basic clickjacking", so it will not be described here.

Due to the critical gap, it is not surprising that this attack works any-more. There are protection mechanism now that break the frames. Such mechanisms will be explained in Chapter 3 in detail.

Clickjacking with AJAX and Flash

It is also noteworthy that there was published a PoC with AJAX [5], Flash and clickjacking in 2008 [6]. It does not use the settings manager, so that the above frame-busting techniques do not work. All it does is just redress an evil iframe with a preloaded malicious Flash file [19]. With this, the victim will be lured into clicking on the "Allow" button on an otherwise invisible "Flash Player Settings Manager" window, because the button is displayed in another context. In this instance it becomes clear that it is not necessary to visit the Adobe web site to activate the camera or microphone by doing clickjacking.

[5] AJAX stands for "Asynchronous JavaScript and XML" and can be used to send HTTP requests and responses without reloading the whole web page in the web browser.
[6] http://www.gnucitizen.org/blog/more-advanced-clickjacking-ui-redress-attacks/

2.1.5. JavaScript to follow the mouse cursor

Another variant to expand “basic clickjacking” is to use the object-oriented scripting language JavaScript. It can be used to make an invisible and small iframe follow the mouse cursor. For this reason, it is not important where a user moves his mouse, considering that just a click is needed, which regularly causes an action in the invisible iframe to follow the mouse cursor.

On the one hand, the victim is restricted to perform only one action with the first click, by virtue of that it is triggered by the iframe. In this case, a victim has no choice to choose another element to perform one more action by clicking somewhere on the web page. On the other hand, this kind of attack requires the usage of JavaScript, which can be e.g. disabled by protection mechanism of the web browser.

2.1.6. Implementation considerations

A detailed analysis of the explained attack “basic clickjacking” shows that there are a few deficits to cheat a victim. Some aspects are not solved optimally, and this will be discussed in the following.

To choose the right kind of attack

As described in Chapter 2.1, it is possible to lure a victim into clicking on the “Go” button. The target is that the victim clicks on the link “Log out” of the web site “google.com”. As it is presented in the status bar of Figure 2.3, the URL of the link “Log out” is

- http://www.google.com/accounts/Logout?continue=http://www.google.com

It is immediately obvious that a “Cross-Site Request Forgery” (CSRF) attack probably works here. The following code of Listing 2.5 illustrates this presumption. It is only required to load a web page of “google.com”, which holds parameters to log out the victim. Thus, in this case there is no need for a clickjacking attack. As already mentioned, CSRF only works here because there is no protection mechanism like a nonce available. Therefore, an attacker must always consider between which attacks can be carried out and how much time they take.

Listing 2.5: HTML “iframe” tag to execute a CSRF attack

```
<iframe src="http://www.google.com/accounts/Logout?continue=http://www.google.com
    " width="1" height="1" scrolling="no" border="0" frameborder="0">
</iframe>
```

Overall it becomes clear that the “basic clickjacking” attack of Chapter 2.1 is a suitable basis to expand the range of web vulnerabilities.

Hide information of the status bar

As mentioned above, the URL of “google.com” appears in the status bar. Regarding this, a careful user will notice that the functionality of the “Go” button has been manipulated. So an attacker must take care to hide this information, otherwise the victim can become suspicious.

A possible solution is, if it possible, to use an event handler in the HTML “a” tag with “document.location=” plus the URL and the instruction “return false”. Otherwise, more complex JavaScript code should be used. However, the instruction ensures that one will not be forwarded to the URL given in the “href” attribute. An example is illustrated in Listing 2.6. By clicking on the link, the victim will be forwarded to the URL of the event handler. In addition to this, there is just the URL “http://www.google.com” in the status bar and not the target web page “http://www.nds.rub.de”.

Listing 2.6: HTML “a” tag with the “onclick” event handler

```
<a href="http://www.google.com" onclick="document.location='http://www.nds.rub.de
    ';return false;">
  Go to google.com
</a>
```

Updated element positions

One of the problems with clickjacking is carrying it out, considering that the exact position on the target web page is required. A possible scenario is a new position of an element caused by an update on this web page. This update can consist of new content like announcements or advertisement banners with different height values.

Listing 2.7: HTML code to create an anchor

```
<a name=here>Target</a>
```

A possible solution for this positioning problem is to use HTML that can be set with the “a” tag, as given in Listing 2.7. As described on the W3C web page, the “id” attribute may be also used to create an anchor at the start tag of any element [20]. This fulfils exactly the requirement to jump automatically to the position of an element.

It should be noted that this property of using an anchor is important, and will be discussed in the following [21]. To give a concrete example, it is possible to directly jump to the menu of the web site

- http://www.nds.rub.de

by analyzing the source code. This is done to set the right anchor in the address bar of the web browser. A truncated HTML code of the web page is given in Listing 2.8.

Listing 2.8: Truncated HTML source code of the web page “www.nds.rub.de”

```
<div id="menu"><ul><li class="open"><a href="/chair/">Lehrstuhl</a><ul><li class
    =" heading"><span>Lehrstuhl</span></li><li class=""><a href="/chair/people/">
    People</a></li><li class=""><a href="/chair/hackpra/">HackerPraktikum</a></li
    ><li class=""><a
```

In the beginning of this Listing, the “div” tag contains the attribute “id” with the value “menu”. For this reason the anchor “#menu” is added at the end of the URL, such that the complete URL looks like

- http://www.nds.rub.de/chair/news/#menu

By opening the above URL, the web browser will automatically jump to this element. This “id” anchor works also with nested frames, so that it can be used to do relative positing of web page content with the help of JavaScript. A JavaScript code example is given in Listing 2.9 [21].

Listing 2.9: JavaScript code to do relative positing in nested frames

```
innerFrame.src=targetUrl+'#fragment';
outerFrame.scrollBy(100, 20);
```

Social engineering

In the case of social engineering, an attacker uses psychological tricks on a legitimate user of a computer system in order to gain information about the system or to perform unauthorised actions in general.

A good example is the Facebook clickjacking worm, which combines various human interests [22]. There was published an image with an almost naked woman and a link with the text “Want 2 C something hot? Click da button, baby!” in a Facebook profile message. When the user clicked on this link, a web page was loaded with a picture of the woman and the integrated text “Want 2 C something hot? Click da button, baby!”. Additionally, there was shown a big red button that the user should click on. Many user clicked on the button, which was deposited with clickjacking code. Therefore, the above Facebook message was in each user profile, too.

A similar example with respect to the in 2009 published “twitter.com” worm will be treated later in Chapter 2.2.1 [23].

2.2. Advanced Attacks

In the following, attacks will be discussed that are based on UI redressing and especially clickjacking. The attacks are described in detail by using examples to get a better understanding.

2.2.1. Clickjacking and CSRF

As generally known, a unique value in the form of a token can prevent a Cross-Site Request Forgery (CSRF) attack or, at least, it can be used to make it more difficult [24]. For this reason, such a token can be used in a hidden form field to transmit information with a form to a web server.

A modification of Listing 2.1 shows that there exists only one difference between a “basic clickjacking” attack and a clickjacking attack with CSRF. This difference is the changed value of the “src” attribute of the “iframe” tag with code to fill in information in a form automatically. The form can be filled in such a way if it is e.g. supported to transmit form values through variables with the HTTP “GET” method [25]. In this case it is required to open a web page with modified values in the address bar. Similarly, there can be used the “iframe” tag to open such a web page.

A well-known example for clickjacking with CSRF is the worm of “twitter.com”, published in February 2009 [23]. It was initiated and executed by a clickjacking attack. To understand this attack there should be some knowledge about parts of the basic functionality of the microblogging web site “twitter.com”.

An important basic functionality of the web site “twitter.com” is to send a short message to a web server of Twitter. This can be the current activity of a user or e.g. other important events. This kind of information is sent to the web server with the help of an HTML form. A short message can be typed in a text area and after that there is just a need to click on the send button. Another way to fill this text area with a short message is to use the address bar of a web browser and to type in the message as a value of the attribute “status”. This is possible because the form web page uses the GET method to fill out the text area automatically if there is a value of the attribute “status” existent. An example of a feasible URL typed into the address bar of a web browser is

- http://twitter.com/home?status=Don’t Click: http://tinyurl.com/amgzs6

After a request of the web page, the message “Don’t Click: http://tinyurl.com/amgzs6” will be pasted in the text area automatically. This message consists of the text “Don’t Click:“ and a short URL that refers to a web page of the attacker. When a visitor of the microblog clicks on the above URL, a web page will be loaded that holds a “Don’t Click” button as displayed in Listing 2.10.

Listing 2.10: HTML and CSS code to create a button and to set its position [1]

```
<BUTTON
  style={
    width: 120px; top: 10px; left: 10px;
  position: absolute; z-index: 1;
  }
>
Don’t Click
</BUTTON>
```

Regarding social engineering, a victim will be interested to know what information is behind the button, and might possibly click on it. This human characteristic will be used by an attacker. Similar to the “basic clickjacking” attack of section 2.1, there can be used a transparent “iframe” tag to abuse the “Don’t click” button as an eye catcher to click on an element of the iframe. The “iframe” tag of the worm is shown in Listing 2.11.

Listing 2.11: Iframe code of the Twitter worm [1]

```
<IFRAME
  style={
    width: 550px; height: 228px;
    top: -170px; left: -400px;
    position: absolute; z-index: 2;
    opacity: 0; filter: alpha(opacity=0);
  }
  scrolling="no"
  src="http://twitter.com/home?status=Don’t Click: http://tinyurl.com/amgzs6">
</IFRAME>
```

As in the “basic clickjacking” attack, the position of the transparent iframe can be set with CSS code. When a victim clicks on the button “Don’t click”, it clicks on the form button of “twitter.com”, loaded by the iframe. After a click by a victim, the defined message will be posted on the microblog automatically. Some readers of the microblog may follow the link to the malicious web page and the cycle of infection will go on.

2.2.2. Clickjacking and XSS

XSS is an abbreviation for “Cross-site scripting”. It is a kind of attack that can be typically found in web applications, as in the case of clickjacking. This type of vulnerability has the goal of injecting variable areas of dynamical web pages with the code of an attacker. The capability of XSS includes the theft of data, changes of the visual appearance of at least one web page and inter alia “Distributed Denial of Service" (DDoS) attacks to make a computer resource unavailable [26]. There are three basic types of XSS. They are known under the names "non-persistent", "persistent" and "DOM-based" cross-site scripting [27].

In the following the focus is exemplary on non-persistent XSS, also called "reflected XSS", to show what is possible with a combination of clickjacking and XSS. The attack starts usually with an attacker who sends input data to a web application of a web server via the HTTP GET method. The first step is successful if this input data is reflected by the server to the web browser. This data can be HTML code to create a headline, JavaScript code to output an alert window and so on. It is at this point that clickjacking comes into play. The main problem with this kind of XSS attack is, that an attacker has to find a vulnerability on a web page and to send the URL, which is used to send data with the GET method over the web browser, to a victim who opens this URL with the code of the attacker to perform actions on it.

Eventjacking

Regarding "clickjacking", "likejacking" and "strokejacking", two new invented term are introduced in this paper by the names "eventjacking" and "classjacking". At the end of this section, it will be addressed why they are called so.

It should be assumed in the following that there is a web page that loads an image. This image is defined by an "img" tag. It holds an "src" attribute with a value. This value is set by the parameter "pic" in the address bar of the web browser via the HTTP GET method. The URL should look like

- http://example.org/images/show.php?pic=flower.jpg

Thus, the HTML "img" tag code of the web page can look like

- <img src="flower.jpg">

It is also assumed that the value of the "pic" attribute will be, for the most part, filtered by the web server with a blacklist. Here, however, should be ignored by the programmer of the web page that an "onclick" event handler and quotation marks could also be dangerous. The modified URL, which has an alert window as an output by clicking on the picture, can look like

- http://example.org/images/show.php?pic=flower.jpg"%20onclick="alert(0)

and thus the HTML code of the "img" tag is

- <img src="flower.jpg" onclick="alert(0)">

The problem here is that the victim needs to perform a click on this image. This can be done with clickjacking by using transparent IFRAMEs, as described in section 2.1.

The conclusion of the example is that, depending on the XSS filtering, it is possible to append the "onclick" event handler up to all links or the whole document. That gives an attacker the option, beyond the "onclick" example, to perform actions between a click and a request of the next web page. As the name "eventjacking" says, this type of attack hijacks events to expand the functionality of XSS by clickjacking. This can be used to read the cookie, to get an CSRF token as discussed in section 2.2.1 or to perform many other dangerous actions by using the victim as a click generator.

Classjacking

CSS, as discussed in section 2.2.3 in detail, offers the attribute “class” as a selector to style a group of HTML elements [28]. Consequently, it is feasible to style e.g. “span” and “a” tags as illustrated in Listing 2.12. Here, the “span” tag has the value “foo” and the “a” tag the value “bar” inside the “class” attribute. This values can be used to define the font size or other CSS-specific properties and are not appended to Listing 2.2.3.

The first “script” tag holds an “src” attribute with the value “http://code.jquery.com/jquery-1.4.4.js”. It is a reference to a file of the “jQuery JavaScript Library v1.4.4”. The name “jQuery” stands for a JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions [29]. So it is ideally suited to deal with user interactions and to manipulate them, as required for complex UI redressing attacks. Thus, “jQuery” is given in the second “script” tag of Listing 2.12. At first, the “span” tag is selected, which holds the value “foo” in the “class” attribute. After that, “.click” is implemented. It can be used to bind an event handler to the “click” JavaScript event, or to trigger that event on an element [30].

Listing 2.12: Classjacking with jQuery (filename: classjacking.html)

```
<span class=foo>Some text</span>
<a class=bar href="http://www.nds.rub.de">www.nds.rub.de</a>

<script src="http://code.jquery.com/jquery-1.4.4.js"></script>
<script>
  $("span.foo").click(function() {
    alert('foo');
    $("a.bar").click();
  });
  $("a.bar").click(function() {
    alert('bar');
    location="http://www.example.org";
  });
</script>
```

In this case, an alert window will be executed with the text “foo” after clicking on the “Some text” value of the “span” tag. After closing the alert window, a click event is triggered on the “a” tag with the value “bar” inside the “class” attribute. Analogue to the first event, an alert window appears with the text “bar”. After closing the alert window, the web browser will redirect the web page to “http://www.example.org”. If there is a click on the link “www.nds.rub.de” and not on the text “Some text”, an alert window is displayed with the text “bar” followed by a redirection to “http://www.nds.rub.de” and not “http://www.example.org”. This behaviour follows from the “href” attribute.

In summary it can be said that it is possible to combine different actions as a consequence of a click on an element by using the CSS “class” attribute and “jQuery” as a JavaScript library. For this reason, the attack is called “classjacking”. The attack is, especially in combination with XSS, very powerful when it is possible to add JavaScript code, as shown in Listing 2.12, to handle elements by their class.

2.2.3. Clickjacking and CSS

CSS is the abbreviation for “Cascading Style Sheets”. It is a style sheet language to describe the formate and look of a document in a markup language [31].

Breaking pointer-events

In 2009 pointer-events were introduced for HTML in Firefox 3.6. The “pointer-events” CSS property allows for selecting between the two values “auto” and“none”. When “none” is used, pointer-events are passed through the target element, otherwise they will be handled normally [32]. This feature offers the possibility to place e.g. a DIV oder another DIV without blocking the click events addressed to the underlying DIV.

In the following the two Listings 2.13 and 2.14 are given [33]. Listing 2.13 shows a working “pointer-events” example in Firefox 3.6, 4.0 or inter alia Google Chrome 8.0. The web browser shows the text “XXX” and an alert window with the value “2” by clicking on the text, or more precisely the link. This is caused by “pointer-events” with the value “none”.

Listing 2.13: Working pointer-events:none code

```
<a style="pointer-events:none;position:absolute;" onclick="alert(1);">
    XXX
</a>
<a href="javascript:alert(2)">
  XXX
</a>
```

Listing 2.14 displays another “a” tag in the “a” tag, which holds the “pointer-events” property. The nested links ensures an alert window will appear with the value “1” by clicking on the “XXX” link and thus the feature of “pointer-events” breaks. This example illustrates that “a” tags should not be used with the pointer-event logic.

Listing 2.14: Breaking pointer-events:none code

```
<a style="pointer-events:none;position:absolute;">
  <a style="position:absolute;" onclick="alert(1);">
    XXX
  </a>
</a>
<a href="javascript:alert(2)">
  XXX
</a>
```

Whole-page clickjacking

CSS offers the option to use attribute selectors to select elements with specific attributes. One of four possible ways to define an attribute selector is given in Listing 2.15 [34]. The in the code listed property matches for every link that points to “http://www.example.org”.

Listing 2.15: CSS attribute selector code with a font-weight definition

```
a[href=http://www.example.org/] {
  font-weight:bold ;
}
```

The web browser Opera allows for breaking out of attribute selectors and other CSS constructs by using braces to define new attributes and their associated values. The example of Listing 2.16 illustrates how this can be done.

Listing 2.16: Whole-page clickjacking code [2]

```
<style>
  p[foo=bar{}*{-o-link:'javascript:alert(1)'}{}*{-o-link-source:current}]{
    color:red;
  }
</style>
```

"-o-link-source" is used to specify the source anchor for the element with the value "current" to use the current value of "-o-link". This "-o-link" is specified before "-o-link-source" with the value "javascript:alert(1)". This value outputs an alert window and shows exemplary, that JavaScript code can be inserted. This alert window will be shown after a click somewhere on the web page.

Opera uses this properties as CSS linking extensions to attach information to XML elements that can not be represented otherwise [35]. They allow for the inclusion of images, hyperlinks, and as it is shown in Listing 2.16 and certainly not intended, the possibility to execute malicious JavaScript code in XML documents.

2.2.4. Text injection by drag-and-drop

A possible solution to what can be done if CSRF and the combination with CSRF and clickjacking does not work is to use the drag-and-drop API. It is a part of HTML5 [36]. Data can be dragged across a domain and, regarding this, there is no need to care about the "Same Origin Policy" (SOP). Founded by Netscape Communications, the SOP restricts JavaScript on one web site from accessing data from another, so it is not surprising that it does not apply in the case when HTML5 is used [37].

An example is given in Listing 2.17. It shows code to create a DIV containing drag-and-drop code. An iframe is given to drop the heading "Drop me" into the text field of this iframe. Furthermore, the method "setData" allows, with the event handler "ondragstart" and the attribute "draggable" with the value "true", to drag the text "malicious code" and not "Drop me". The "src" attribute of the iframe holds the URL "http://www.google.com", so that it is possible to drop the text into the text field of the search engine of Google.

Listing 2.17: HTML code to display an iframe and a DIV to use the drag-and-drop API (filename: dragAndDrop.html)

```
<div draggable="true" ondragstart="event.dataTransfer.setData('text/plain','
     malicious code');">
  <h1>Drop me</h1>
</div>

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

Thus the example shows that the SOP does not apply as desired by the user concerning drag-and-drop operations. It also becomes clear that draggable text can be modified by an event handler. This is especially interesting if one wants to fill out different text fields by clicking on a text that places different text in each drag-and-drop operation.

This kind of attack is, in comparison to "basic clickjacking", much harder to realise, due to the fact that an attacker wants to get the user to drag something. Clicking on stuff like links is a normal action on web pages. For the reason that it is more complex, this can not be said about drag-and-drop operations. An important advantage regarding the SOP is that an attacker can bypass the SOP without using e.g. CSRF to inject text into targets like webmail or document text areas, text fields or e.g. WYSIWYG [7] editors.

If possible, Java Applets can also be used to perform a text injection attack. It is more complex, but it offers more possibilities to the attacker for the reason that the Java API has many more functionalities than a default web browser [8]. All in all, the text injection attack can be seen as very powerful and dangerous.

2.2.5. Content extraction

With the help of text injection, it is possible to bypass the SOP, as described in section 2.2.4. Regarding this, it is also possible to get data of a web page of another web site. This technique is called "content extraction" [8]. It makes use of reverse drag-and-drop operations, so that one can drag content out of an iframe.

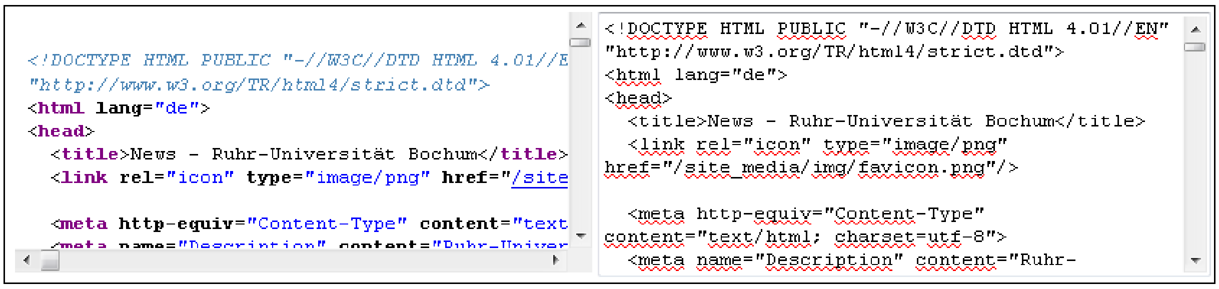
First of all it should be clear that links and images are draggable by default as URLs. Each URL can contain sensitive information like session identifier. Normaly it is much more interesting to get the content in the form of source code of a web page, so an attacker can e.g. grab nonces inside hidden text fields or the account balance in the case of an online banking application. Another possible scenario is to get data of intranet web pages by knowing the URL.

Listing 2.18: HTML "iframe" and "textarea" tag to perform content extraction (filename: sourceCode.html)

```
<iframe src="view-source:http://www.nds.rub.de/chair/news/" frameborder="0" style
    ="width:400px;height:180px">
</iframe>
<textarea type="text" cols="50" rows="10"></textarea>
```

To show the source code of a web page inside the web browser Mozilla Firefox or Google Chrome, there can be used "view-source:" as a prefix in the web page address. A decisive factor is that only Firefox allows the source code in an iframe to be displayed.

With the combination of a "textarea" tag, just two drags to perform this attack are needed, as in the case of elements like images. The first drag is to select an element and the second to drag an element out of the iframe into the text area. The element is, in the case of "view-source:", the source code of the web page. Listing 2.18 gives the discussed text area and iframe. The iframe loads the web page of the "Chair of network and data security". A screenshot of the created web page and a successful drag-and-drop operation is given in Figure 2.5.



```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//E
"http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
<head>
  <title>News - Ruhr-Universität Bochum</title>
  <link rel="icon" type="image/png" href="/site

  <meta http-equiv="Content-Type" content="text
  <meta name="Description" content="Ruhr-Univer
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
<head>
  <title>News - Ruhr-Universität Bochum</title>
  <link rel="icon" type="image/png"
href="/site_media/img/favicon.png"/>

  <meta http-equiv="Content-Type"
content="text/html; charset=utf-8">
  <meta name="Description" content="Ruhr-
```

Figure 2.5.: Content extraction code of Listing 2.18 displayed with Mozilla Firefox 3.6.3

[7] A "what you see is what you get" (WYSIWYG) editor is a visual program in which one can see on-screen what one will get.

2.2.6. Pop-up blocker bypass

Modern web browsers include blocking software to protect a user against pop-up windows, which hold e.g. advertisement messages. A warning message is regularly displayed to tell the user that a pop-up window has been blocked. To give an example, a highlighted warning message of Mozilla Firefox occurred when blocking two pop-up windows is “Firefox prevented this site from opening 2 pop-up windows.”.

A web browser like Firefox distinguishes between trusted and not trusted events, depending on the situation [38]. User interactions like a click will be trusted for the reason that they are made explicitly by the user. If a web page initiates an event like opening a pop-up window automatically, the event is not trusted and therefore blocked. Tests of this work have shown that other browsers like Google Chrome or Opera behave similarly.

With the use of clickjacking techniques, an attacker can get its victim to create a trusted event by clicking on a link that opens one or more pop-up windows. An example is given in Listing 2.19.

Listing 2.19: HTML and JavaScript code to open pop-up windows (filename: popup.html)

```
<script>
function makePopups(){
  for (i=1;i<6;i++) {
    window.open('popup.html','spam'+i,'width=50,height=50');
  }
}
</script>
<body>
<a href="#" onclick="makePopups()">Spam</a>
```

First of all the web page includes a link with the help of an “a” tag. The “src” attribute refers to a hash and thus it is a non-existing anchor. The last attribute “onclick” will open the JavaScript function “makePopups” by clicking on the link “Spam”. Regarding this the web browser will flag this event as a trusted action. The function takes this property and generates, with a “for” loop, five pop-up windows with the method “window.open”. The method can consist of the three parameters “URL”, “windowName” and “windowFeatures” [39]. Thus, the method opens its own web page “popup.html”. It also gives each window the name “spam” plus the round number of the variable “i”, so that each pop-up window has its own name to be not blocked. Finally, the method determines the width and the height of the pop-up windows.

The conclusion is that an attacker can get the victim to unknowingly trigger a trusted event by doing a click. This event can be recycled by an attacker for later usage or directly used to generate, as in the example of Listing 2.19, pop-up windows that the user does not desire.

2.2.7. SVG masking

“Scalable Vector Graphics”, or “SVG”, is a specification of the W3C for describing two-dimensional vector graphics [40]. This specification offers many functionalities such as creating visual effects on text, triggering events with a following reaction, creating animations, and last but not least, masking.

Listing 2.20: Truncated code of the SVG demo with a “body” and “iframe” tag (filename: svg.xhtml)

```
<body style="background:gray">
```

```
  <iframe src="http://www.ruhr-uni-bochum.de/suche/index_en.htm" style="width:800
      px; height:350px; border:none; mask: url(#maskForClickjacking)"/>
```

Masking elements can greatly simplify a clickjacking attack, so a feasible scenario of it will be shown [41]. As shown in Listing 2.20, a “body” tag with the “style” attribute “background:gray” is given. As the name suggests, the background of the web page will have the color gray. The “iframe” tag holds the attributes “src” and “style”. As already known, the URL of the target web page is the value of the “src” attribute. Inside the “style” attribute there is information to the width, the height, and the border of the web page. Finally, there is the property “mask” with “url(#maskForClickjacking)”. This “url” points to an SVG with the “id” value “maskForClickjacking”.

Listing 2.21: Truncated code of the SVG demo with the “svg” tag (filename: svg.xhtml)

```
<svg:svg>
  <svg:mask id="maskForClickjacking" maskUnits="objectBoundingBox"
      maskContentUnits="objectBoundingBox">
    <svg:rect x="0.0" y="0.0" width="0.373" height="0.3" fill="white"/>
    <svg:circle cx="0.45" cy="0.7" r="0.075" fill="white"/>
  </svg:mask>
</svg:svg>
```

The code of the SVG is given in Listing 2.21. First of all, an “svg” tag with the namespace “svg” is defined. After that, a “mask” tag with the attributes “id”, “maskUnits” and “maskContentUnits” is inside the “svg” tag. The attribute “id” holds the value “maskForClickjacking”, which is exactly the value inside the “url” of Listing 2.20. The attribute “maskUnits” defines the coordinate system for the data of “x”, “y”, “width” and “height”. The second attribute “maskContentUnits” defines the coordinate system for the contents of the “mask” with “objectBoundingBox” [42]. Inside the “mask” tag, there are two tags called “rect” and “circle”. Each tag holds information to the position and is determined by the geometric shape the width and height or radius. The attribute “fill”, with the value “white”, ensures that the viewing whole in the mask is visible.

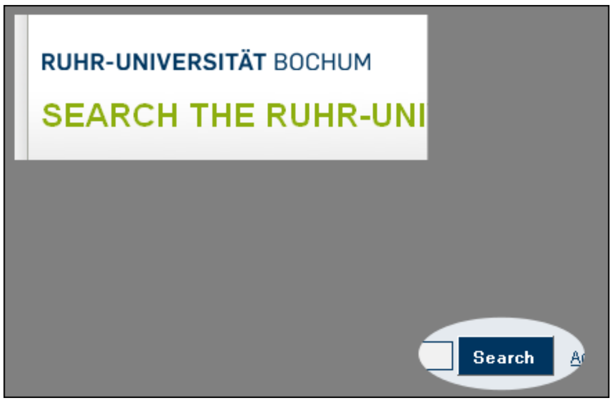


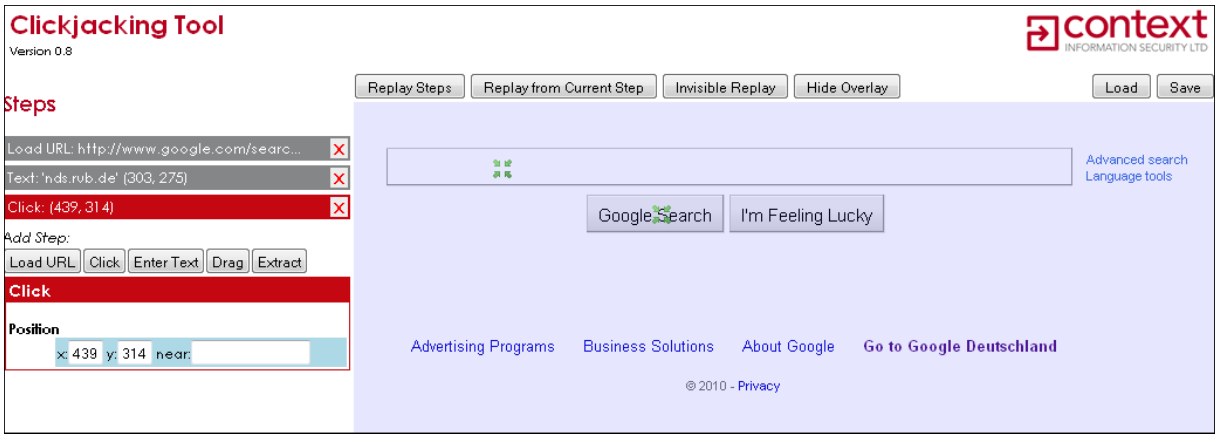
Figure 2.6.: Iframe with an SVG mask displayed with the web browser Mozilla Firefox 3.6.3

The complete XHTML code as a combination of Listing 2.20 and Listing 2.21 is given in the appendix in Listing A.3. A screenshot of the code of Listing A.1.2 is given in Figure 2.6.

2.3. Clickjacking Tool

For the reason that most web sites still have no implemented security mechanism against clickjacking, which is discussed later in chapter 3, the company “Context Information Security Limited” published a program for clickjacking called “Clickjacking Tool” [8]. This tool was introduced with a talk by Paul Stone at the Black Hat Europe in 2010 [43]. It was primarily built due to the fact that it seems to be difficult to visualise clickjacking techniques in practice.

It can be run in a web browser like Mozilla Firefox and allows for experimentation and trying out different techniques with point-and-click operations without a detailed knowledge of JavaScript or e.g. CSS [44]. So it is possible to use point-and-click operations to select areas of a web page to be targeted just by using the mouse. It also supports techniques that are introduced in this paper that allow it to be easy used to try out the attacks. Functionalities like a “Visible mode” for replaying attacks to see how they work and the “hidden mode” for simulating a real clickjacking attack are helpful to get a better understanding of clickjacking.



Load URL: http://www.google.com/searc...
Text: 'nds.rub.de' (303, 275)
Click: (439, 314)

Click
Position
x: 439 y: 314 near:

Figure 2.7.: Truncated screenshot of the “Clickjacking Tool” displayed with the web browser Mozilla Firefox 3.6.3

A screenshot of the “Clickjacking Tool” is given in Figure 2.7. There are three steps defined. First, the web browser opens the “google.com” web page, after that it types “nds.rub.de” into the search engine input field, and finally it clicks on the button “Google Search”. Moreover, the condition for these steps to work is that the user must disable the automatic AJAX-based search function of “google.com”.

There are also things, or rather functionalities, that the program does not offer. The first is that it is not possible to export a demo page to get malicious code to perform the attack directly. Secondly, the source code of the simulated clickjacking web page can not be edited to see new effects concerning each attack variant by changing some parts of it. However, it is possible to check quickly whether a web page can be attacked or not. It would be also desirable to integrate SVG masking features so that one can, for example, place the mask over an element with the mouse directly.

[8]The “Clickjacking Tool” can be downloaded from the URL: http://contextis.com/resources/tools/clickjacking-tool/

3. Counteractive measures

This chapter focuses on the topics frame busting and busting frame busting. A detection system to scan web pages for possible clickjacking vulnerabilities is also discussed.

3.1. Frame busting

As discussed in Chapter 2, HTML "iframe" tags are required to execute clickjacking or e.g. to show specific content in another context. Especially some certification authorities that offer the signing of SSL/TLS certificates provide products like logos to show that the current web page of the user is secure and trustworthy. Usually, these logos are used to encourage the visitor to make commercial transactions, for example to buy something in an online shop. If it is possible to place such logos on a malicious web page, a victim could have, at least in the first moment, a wrong impression about the trustworthiness of this web page.

A randomly selected test has shown, that providers of such logos have no frame busting mechanism on their web pages, on which their own logos are included. These logos can be used to show them in another context, as displayed in Figure 3.1. The HTML and CSS code used in Figure 3.1 is given in the appendix with the Listings A.4, A.5, A.6 and A.7.

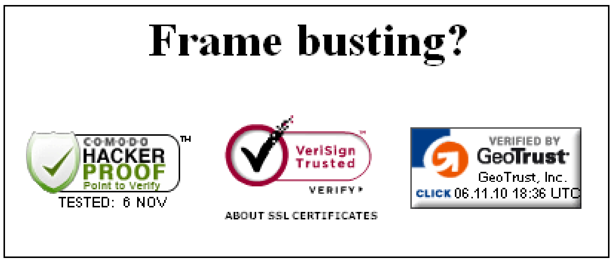


Figure 3.1.: Logos of the certification authorities Comodo, Verisign and Geotrust loaded by iframes displayed with the web browser Google Chrome 8.0 beta

Regarding this, there are protection measures required to prevent attacks with "iframe" tags. This will be discussed in the following subsections.

3.1.1. JavaScript

In section 2.1.4 the vulnerability concerning the Flash Player Global Security Settings panel not working any-more was explained. A prominent reason for this is the used JavaScript frame busting code. This code is displayed in Listing 3.1. It shows the structure of frame busting code, which typically consists of a "conditional statement" and a "counter-action" statement. The conditional statement consists in this case of the code "top!=self". The conditional statement is true if the own window, described through "self", is not simultaneously the window on the "top". This is the case when a web page is framed. If so, the counter-action forwards the user to the framed page with "top.location.href=self.location.href".

Listing 3.1: Frame busting code of the Flash Player Global Security Settings panel

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

In the paper "Busting Frame Busting: a Study of Clickjacking Vulnerabilities on Popular Sites", published in 2010, the Top-500 Alexa web sites were analysed [3]. This was used to create tables that consist of the most popular frame busting codes. They are split into the conditional and counter-action statement. The first four entries of each table are listed in Table 3.1 and 3.2.

Unique sites	Conditional statement
38%	if (top != self)
22.5%	if (top.location != self.location)
13.5%	if (top.location != location)
8%	if (parent.frames.length > 0)

Table 3.1.: Conditional statements of frame busting code

Unique sites	Counter-action
7	top.location = self.location
4	top.location.href = document.location.href
3	top.location.href = self.location.href
3	top.location.replace(self.location)

Table 3.2.: Counter-actions of frame busting code

The authors of the above-mentioned paper proposed their own frame busting code, too. This code is displayed in Listing 3.2. At first there is a "style" tag, which hides the whole body of the web page. After that, there is a "script" tag with the conditional statement "self==top". If it is true, the web page is not framed. If so, the code inside the conditional statement makes the body visible. Otherwise, one will be forwarded to the framed web page.

Listing 3.2: Proposed Frame Busting code of the paper "Busting Frame Busting"

```
<style>
body { display: none; }
</style>

<script>
  if (self == top) {
    document.getElementsByTagName(``body'')[0].style.display = 'block';
  } else {
    top.location = self.location;
  }
</script>
```

It is immediately apparent that the “script” tag code only works if JavaScript is enabled. Otherwise, the body will not be displayed and is blank. However, if a web site administrator only wants to rely on JavaScript as a protection mechanism, the specified code is useful to protect the user against clickjacking attacks.

3.1.2. X-Frame-Options

The X-Frame-Options HTTP response header was introduced by Microsoft in 2008. It can be used to indicate if the web page can be loaded in a “frame” or “iframe” tag. There are two possible values for X-Frame-Options. They are called “DENY” and “SAMEORIGIN”. By using “DENY” the web page can not be loaded by a frame. In contrast, “SAMEORIGIN” can be used to display the web page in a frame when the origin of the top-level browsing context is not different [45].

Listing 3.3: PHP header to use X-Frame-Options

```
<?php
header("X-Frame-Options: DENY");
?>
```

Listing 3.4: Apache configuration entry to use X-Frame-Options

```
Header always append X-Frame-Options DENY
```

A possible implementation in the server-side language PHP and the web server Apache is given in Listing 3.3 and 3.4. A list of the supported web browsers is given in Table 3.3 [46]. It should be noted for Mozilla Firefox, that the plugin NoScript had experimental X-FRAME-OPTIONS compatibility support in version “1.8.9.9” [47]. This support has been adopted in Firefox since version “3.6.9”, as it is displayed in the mentioned table.

Browser	Lowest version
Internet Explorer	8.0
Firefox (Gecko)	3.6.9 (1.9.2.9)
Opera	10.50
Safari	4.0
Chrome	4.1.249.1042

Table 3.3.: X-Frame-Options web browser compatibility

The problem with the necessity of JavaScript does not exists in this case. This is because there is no need to use JavaScript code. Possible problems can be to use X-Frame-Options with proxies, which strip the HTTP header, or multi-domain sites [3]. Nevertheless, the advantages outweigh the disadvantages. One should therefore decide for this variant of the frame busting.

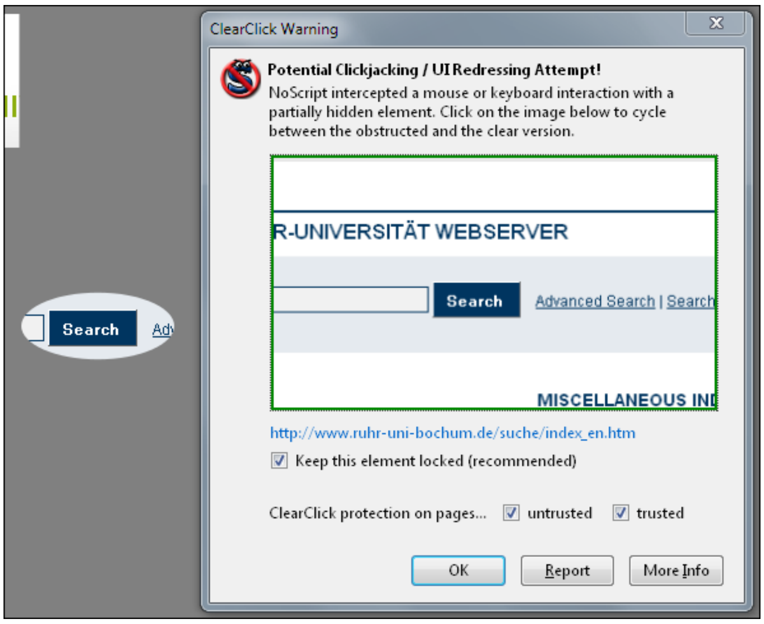
3.1.3. NoScript

NoScript [1] is an extension for Mozilla-based web browsers like Firefox. It allows JavaScript, Java, Flash and other plugins to be executed only by a trusted web sites of the user’s choice. In addition, XSS and

[1]The NoScript extension can be downloaded from the URL: http://noscript.net/

clickjacking protection is integrated [48].

Concerning clickjacking, the extension has provided a plugin called “ClearClick” since version 1.8.2. This feature recognises clicking and typing on hidden elements and clearly reveals the element by showing a “ClearClick Warning” window as it is displayed in Figure 3.2.


ClearClick Warning

Potential Clickjacking / UI Redressing Attempt!
NoScript intercepted a mouse or keyboard interaction with a partially hidden element. Click on the image below to cycle between the obstructed and the clear version.

http://www.ruhr-uni-bochum.de/suche/index_en.htm

[✓] Keep this element locked (recommended)

ClearClick protection on pages... [✓] untrusted [✓] trusted

Figure 3.2.: Clickjacking SVG example of section 2.2.7 with a “ClearClick Warning” window displayed in Mozilla Firefox 3.6.12

Furthermore, the extension protects the user inter alia against the usage of IFRAMEs. It blocks such frames in untrusted web pages and also trusted web pages, which try to load content from untrusted web pages [49]. This extension is highly recommended as a client-side solution for clickjacking with a Mozilla-based web browser.

At this point it should be supplemented prophylactically, that the web browser Google Chrome has a plugin such as “NoScript”, too. This plugin is called “NotScripts” and it does not provide advanced protection for clickjacking [50]. However, the clickjacking mask of Figure 3.2 does not appear by using this plugin.

3.2. Busting frame busting

Regarding frame busting there are techniques available that can bust frame busting in the case that JavaScript protection mechanisms are used. These techniques are discussed in the following.

3.2.1. Mobile versus non-mobile applications

Mobile applications are today, especially through the use of smart phones, used frequently. Many web site owners are following this trend and offering mobile applications. They are usually smaller, faster, and less complex than the main applications.

An examination in mobile applications has shown in e.g. “mail.google.com” or in general on web sites like “mobile.example.org” that mobile variants are, in contrast to the main applications, sometimes not protected against framing attacks. Probably there is the assumption, that an attacker does not use clickjacking against mobile web pages, because of their often limited functionality. However, this assumption is fundamentally wrong, because a powerful attacker can fake the origin and other environmental variables given by a web browser, such that a non-mobile victim is able to for example visit an application made just for mobile users [51].

The result is that an attacker can use clickjacking techniques here. From this it follows that it is not necessary to bust frame busting when there are alternatives like unprotected mobile web applications, which allow for using the same attack vectors.

3.2.2. Double framing

As discussed in section 3.1.1, frame busting code consists of a conditional statement and a counter-action. In Table 3.2 are listed four counter actions. They are typically used in frame busting code.

Due to the negligible, the counter-action table does not hold counter-actions with "parent.location". Such actions are, for example

- self.parent.location = document.location
- parent.location.href = self.location
- parent.location = self.location

The “parent” property refers always to the window, if it is available that is one level higher. Otherwise, the property refers to its own window. Over the object “location”, which lies in the JavaScript object hierarchy below the window object, the full URI of the currently displayed web page can be accessed. The combination of the two objects allow therefore, if possible, access to the URI of the upper window. From this it follows that a frame will be busted if it loads a web page with such a counter-action.

The irony here is the behave of the web browser, when an attacker uses a frame to load a frame that loads the target web page. In this case this frame, which loads a protected web page like “http://www.example.org”, will be busted and the web page will be loaded in the frame that holds the busted frame. Actually, the second frame should be also broken, but the security violation of the frame navigation policy prevents it. It is discussed detail in the article “Securing frame communication in browsers” by Adam Barth, Collin Jackson, and John C. Mitchell [52]. Thus, it is possible to frame a web page protected by “parent.location” as a part of the frame busting code by using the double-framing method.

3.2.3. onBeforeUnload event

The “onbeforeunload” event can be used to invoke actions like closing of the current web browser window, clicking on “Back”, “Forward”, “Refresh”, or the “Home” button and so on [53]. When this event is executed, the user gets a message window. It asks if the user really wants to leave the page or not.

Default case

The following example to the above-described default case includes the “onbeforeunload” event. It is given in Listing 3.5.

First of all a headline with the “h1” tag and the text “www.example.org” is defined. This should illustrate that the user is on a web page of “http://www.example.org”. After that, JavaScript code and an “iframe” tag are given. The “src” attribute of the iframe refers to the web site of Paypal. This web site is protected by JavaScript-based frame busting code. To prevent the destruction of the iframe, JavaScript code is given with an “onbeforeunload” event. It is activated for example when the frame busting code wants to destroy the iframe by loading the URL in the whole web page and not only in the iframe. The event creates a warning message, which includes the in-the-function-defined “Do you want to leave example.org?” question. The user can choose, in the case of Mozilla Firefox, between the prompt buttons “OK” or “Cancel”. Another variant is to close the prompt, which is equal to a click on “Cancel”.

Listing 3.5: HTML and JavaScript code to execute an onBeforeUnload event to bust frame busting code [3] (filename: onBeforeUnload.html)

```
<h1>www.example.org</h1>
<script>
  window.onbeforeunload = function() {
    return "Do you want to leave example.org?";
  }
</script>
<iframe src="http://www.paypal.com">
```

204 - HTTP header

Another variant to eliminate some JavaScript frame busting counter-actions is to use a “204 - No Content” HTTP header. The special feature of this case is that a prompt will not be displayed in the web browser. A way to generate a file with this header is given in Listing 3.6. It shows a PHP header function and the header value “HTTP/1.1 204 No Content” [54].

Listing 3.6: PHP code to create a file with an “HTTP/1.1 204” header (filename: 204.php)

```
<?php
  header("HTTP/1.1 204  No Content");
?>
```

Listing 3.7 displays an exemplary web page of a victim that is protected by frame busting code with the counter action “top.location.href = self.location.href;”. It also consists of a “body” tag to make the background red and an “h1” tag to create a headline with the text “Victim”.

Listing 3.7: JavaScript frame busting code on a victim’s web page [3] (filename: VictimOnBeforeUnload.html)

```
<script>
try {
  if (top.location.hostname != self.location.hostname) throw 1;
} catch (e) {
  top.location.href = self.location.href;
}
</script>
<body bgcolor=red>
<h1>Victim</h1>
```

A way to use this header is given in Listing 3.8 [55]. It holds JavaScript code and an “iframe” tag. The iframe loads the web page of the victim of Listing 3.7. At first, the JavaScript code defines a variable called “prevent_bust” and set it to “0”. After that, the variable will be incremented if an “onbeforeunload” occurs. After that, “setInterval” is used to call a function that repeats its contents every “1” millisecond [56]. Inside the function there is an if-condition that checks if the variable is bigger than “0”. This is the case when there is an “onbeforeunload” event. When the condition is true, the variable will be subtracted with the number “2”. Finally, a web page with the “HTTP/1.1 204 No Content” header will be loaded.

Listing 3.8: JavaScript code to eliminate “top.location” frame busting code (filename: onBeforeUnloadWithout-Prompt.html)

```
<script>
var prevent_bust = 0;
window.onbeforeunload = function() {
  prevent_bust++;
};
setInterval(
  function() {
    if (prevent_bust > 0) {
      prevent_bust -= 2;
      window.top.location = "http://localhost/204.php";
    }
  }
, 1);
</script>
<iframe src="VictimOnBeforeUnload.html">
```

Therefore, a web page will be loaded every millisecond that has no content whenever frame busting code with “top.location” inside the counter-action is run, as displayed in Listing 3.7. So there will always be the “HTTP/1.1 204 No Content” header in the request pipeline.

3.2.4. XSS filter

Cross-site scripting (XSS) filter is introduced by browser vendors like Microsoft or Google to provide a security mechanism on the client side. This section describes how the filter can be used to deactivate frame busting code by faking it as malicious code.

IE8 XSS filter

The XSS filter of the web browser Internet Explorer 8 is a component with visibility into all requests and responses flowing through the web browser [57]. By discovering XSS, the attack will be identified and the code will be deactivated.

Listing 3.9: JavaScript frame busting code (filename: frameBustingCode.html)

```
<script type="text/javascript">
if (parent.frames.length > 0){
  top.location.replace(document.location);
}
</script>
<h1>Victim</h1>
```

Listing 3.9 shows an exemplary frame busting code. This code consists of a “script” tag with the attribute “type” and the value “text/javascript”. There is also an if-condition inside the “script” tag. As given in

Listing 3.10, a web page that holds this frame busting code can be loaded with an “iframe” tag. However, the special feature is that there is added a parameter “xyz” with the value “http://www.example.org/” plus the above-described JavaScript code. This code is the beginning of the script and consists of a “script” tag and an “if” from the if-condition.

Listing 3.10: HTML “iframe” tag with JavaScript code to activate the IE8 XSS filter (filename: XSSFilterFrame-BustingCode.html)

```
<iframe src="http://www.example.org/?xyz=%3Cscript%20type=%22text/javascript%22%3
    Eif"></iframe>
```

The XSS filter evaluates this request as an attack, shows a small yellow notification bar as an XSS warning to the user, and disables all inline scripts. Thus, the frame busting code will not be interpreted by the web browser any-more.

Google Chrome XSS-Auditor

In the web browser Google Chrome 4.0 an experimental anti-XSS feature called “XSS-Auditor” was introduced. It should protect the user against cross-site scripting attacks. It can cause compatibility problems with some web pages and it significantly slows down the web browser performance in comparison to the previous version, Google Chrome 3.0. For this reason, by default the XSS auditor has been disabled since version 4.1 [58]. It should be mentioned that Google released in Chrome 8.0 the configuration web page “about:flags”, which allows for enabling experimental features like the XSS-Auditor.

However, in Google Chrome 4.0 it is possible to deactivate a “script” tag with its code of a separate origin by using it as a URL parameter, as shown in Listing 3.11. The blocked frame busting code is given in Listing 3.12 [3].

Listing 3.11: HTML “iframe” tag with a parameter that holds encoded JavaScript frame busting code (filename: GoogleXSSAuditor.html)

```
<iframe src=''http://www.example.org/?xyz=if(top+!%3D+self)+%7B+top.location%3
    Dself.location%3B+%7D''>
```

Listing 3.12: Frame busting code example for the XSS-Auditor

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

3.2.5. Disabling JavaScript

As given in section 3.1, there are different methods available to protect a user against clickjacking attacks. A popular protection method is to use JavaScript frame busting code. If it is possible for an attacker to disable JavaScript code, a web page that only relies on JavaScript frame busting code will not have any protection mechanism against clickjacking. Three deactivation procedures regarding JavaScript and frames are described in the following subsections.

Restricted frames with Internet Explorer

Since Internet Explorer (IE) 6, a frame can have the “security” attribute with the value “restricted” [59]. An “iframe” tag with this attribute is given in Listing 3.13.

Listing 3.13: Restricted frames in IE with the “security” attribute

```
<iframe src=''http://www.example.org'' security=''restricted''>
</iframe>
```

By using the “security” attribute, Internet Explorer will automatically handle the contents of the frame. This is done by a rendering in the “Restricted Sites Security Zone” [60]. It ensures that JavaScript code, ActiveX controls, and inter alia re-directs to other sites do not work in the frame any-more. The background to the introduction of this zone was to create isolated frames to display content like e-mails, which can include malicious code. This code will no longer run after the restriction, but it applies to JavaScript frame busting code, too.

Sandbox attribute

In response to the Internet Explorer “security” attribute, there is also an attribute called “sandbox” specified in HTML5. It enables a set of restrictions on content hosted by the iframe. Allowed optional values of this attribute are “allow-same-origin”, “allow-top-navigation”, “allow-forms”, and “allow-scripts”. The value “allow-same-origin” allows the loaded content of the iframe to be treated as if it were from the same origin and not from a unique origin. To navigate at the top-level browsing context, the value “allow-top-navigation” can be used. Last but not least, the values “allow-forms” and “allow-scripts” re-enable forms and scripts respectively [61].

As described, “allow-scripts” should not be used to bust frame busting. However, if the attribute “sandbox” as a part of the HTML5 development is maintained, there is a similar or even more powerful functionality in comparison to the IE “security” attribute.

Design mode

To edit the current document by bringing its objects into a UI-activated state, the property “designMode” can be used. The support has been successfully tested in Mozilla Firefox, IE8, Google Chrome, and other modern web browsers. It allows for dragging pictures and e.g. modifying and adding text elements in the document.

Paul Stone posted a security issue concerning the “designMode” in the bug-tracking software of Mozilla in 2009 [62]. The problem will be described with the code of Listing 3.14. It is derived from the testcase of Paul Stone and simplifies the issue.

Listing 3.14: HTML and JavaScript code to disable JavaScript in an “iframe” tag by using the “designMode” (filename: designMode.html)

```
<script>
function disableJavaScript() {
  CJiframe = document.getElementById('clickjacking').contentWindow.document;
  CJiframe.designMode='on';
  CJiframe.body.innerHTML = "<iframe id=victim width=100%></iframe>";
  CJiframe.getElementById('victim').src = "http://www.isjavascriptenabled.com";
}
</script>
<iframe id="clickjacking" onload="disableJavaScript()">
</iframe>
```

First of all, an “iframe” tag that holds the attributes “id” and “onload” is given. The attribute “id” is used to address the element. The event handler “onload” executes the function “disableJavaScript” when the “iframe” tag loads. Inside the function, the “iframe” tag is linked to the object “CJiframe” by using the “id” value “clickjacking”. After that, the iframe is set to “designMode=’on”’. Thus, only the iframe is in the design mode and not the whole document. After that a new “iframe” tag with the “id” value “victim” is loaded inside the “iframe”. Last but not least, the last line of the JavaScript code is used to set the “src” attribute of the “victim” iframe.

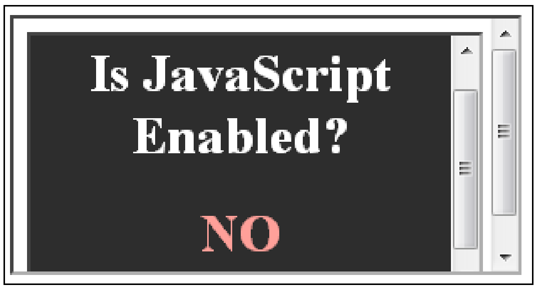


Figure 3.3.: Screenshot of the “designMode.html” file of Listing 3.14 displayed in Mozilla Firefox 3.6.12

After executing the JavaScript code, an iframe in the designMode includes an iframe that loads the web page “http://www.isjavascriptenabled.com”. The result is displayed in Figure 3.3. The difference between Mozilla Firefox and other web browsers is that Firefox disables JavaScript inside the iframe and that the functionality of the design mode is not enabled. So the code can be used to deactivate JavaScript frame busting code in Mozilla Firefox without getting in an enabled “designMode”.

3.2.6. Redefining location

With the web browsers Safari and Internet Explorer, it is possible to redefine the “location” as a variable or function. Therefore, it is not an immutable attribute, as it is in other web browsers [3]. This can be exploited to bust frame busting code.

It is assumed in the following that “http://www.example.org” holds the code of Listing 3.9. This code consists, in order to mention it explicitly, inter alia of “top.location”.

Internet Explorer

In IE7, also successfully tested in IE8, it is possible to redefine “location” as it is illustrated in Listing 3.15. By defining “location” as a variable, a reading or navigation by assigning “top.location” will fail, due to a security violation [63]. Thus, such a frame busting code is suspended.

Listing 3.15: Redefining “location” to deactivate frame busting code with “top.location” checks (filename: locationIE.html)

```
<script>
  var location = "dummy";
```

```
</script>
<iframe src="http://www.example.org">
</iframe>
```

Safari

To bind an object property to a function, the method "__defineSetter__" allows for specifying a string containing the name of the property and a function to be called [64]. To bust frame busting code with "top.location", the code of Listing 3.16 can be used in case "Safari 4.0.4" is used. It binds "location" to a function, so that an attempt to read or navigate to the "top.location" will fail [3].

Listing 3.16: Binding "location" to a function with "defineSetter" (filename: locationSafari.html)

```
<script>
  window.__defineSetter__("location", function(){});
</script>
<iframe src="http://www.example.org">
</iframe>
```

3.2.7. Referrer checking

Allowing framing by the web pages can be done by using the "document.referrer" property. It is supported by all major web browsers and it returns the referrer of the current document.

The authors of the paper "Busting Frame Busting" showed in July 2010 that a correct implementation of "document.referrer" can fail, as in the case of the web site "walmart.com" [3]. The frame busting code of Listing 3.17 was used.

Listing 3.17: JavaScript frame busting code of "walmart.com" (filename: walmart.html)

```
if (top.location != location) {
 if(document.referrer.indexOf
 (''walmart.com'') == -1)
 {
  top.location.replace
 (document.location.href);
 }
}
```

The method "indexOf" returns the position of the first occurrence of the string "walmart.com". If the result is "-1", the value to search for never occurs [65]. As it is displayed in Listing 3.17, the second if-condition is not true and thus does not return the counter-action, if there is a document referrer available and when it holds the value "walmart.com". Therefore, the frame busting code will not be executed when an attacker uses a domain like:

- walmart.com.example.org

Similar problems may arise in the implementation even with regular expressions. It is explained in the above "Busting Frame Busting" paper in detail. Generally, it follows that a developer must pay attention by defining frame busting code exceptions.

3.3. Clickjacking detection system

In 2010 there was released a scientific paper called “A Solution for the Automated Detection of Clickjacking Attacks”. In this work an automated system to analyse clickjacking attacks was designed and developed. A clickjacking study of over one million unique web pages is also given.

Based on Figure 3.4, the system architecture of the mentioned detection system is shown [1]. It displays a “Customized Browser” section with a “Detection unit”. It is used to search for clickjacking attacks with the help of the Mozilla-based web browser plugins “ClickIDS” and “NoScript” (modified).

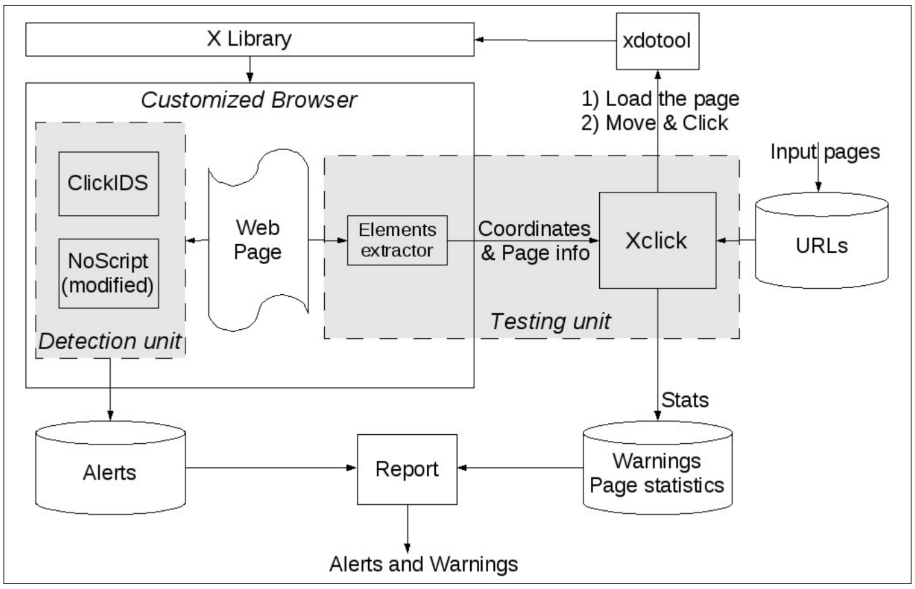


Figure 3.4.: System architecture of an automated detection system for clickjacking [1]

The ClickIDS plugin is implemented by the authors of the detection system. It intercepts click events, scans for interactions and detects clickjacking attacks. The scan for clickable elements concentrates on links, buttons, form input fields, and Adobe flash content. As already discussed in section 3.1.3, there comes a clickjacking warning, denoted “ClearClick”, with NoScript. The modified version NoScript 1.9.0.5 of the system architecture replaces this visual alert. Both plugins together produce an entry for each warning in the “Alerts” database. This database is outside the “Customized Browser” section.

The next interesting section is the “Testing unit”. It simulates a human user to create mouse clicks and other characteristic actions. This testing unit was designed for the reason that there are no solutions to do this with tools like Selenium IDE [2]. The reason why this does not work in the case of clickjacking is that it is not possible to tell a web browser to click on an area just by setting display coordinates with the available tools. In the case of Selenium, one has to select elements like an “iframe” tag directly to perform interactions.

Therefore, the “Testing unit” to correspond with the tool “xdotool” is given [67]. It lets the architecture simulate keyboard inputs and mouse activities, move and resize windows, and to do over actions. Basically it is a wrapper around the X11 testing library, which is displayed in the described Figure 3.4 with

[2]Selenium IDE is a part of three major software tools of Selenium and used as a Mozilla-based web browser plugin as a web application testing system. With the help of this tool it is possible to, for example, record and execute interactions of peripheral devices such as a mouse or a keyboard [66].

"X Library". This script will not be discussed here, because it is explained in the paper "A Solution for the Automated Detection of Clickjacking Attacks" in detail. In general, the "Xclick" testing unit exists to load URLs from the "URLs" database outside the "Testing unit". It holds input pages to send it to the "Xclick" script. The coordinates of each element, which should be checked by the "Detection Unit", are given by the "Elements extractor". The coordinates will be sent to the "Xclick" script.

Warning messages of e.g. transparent "iframe" tags, which are not detected as tools for clickjacking, and page statistics are sent to the "Warnings and page statics" database. Together with the "Alerts" database, a "Report" that consists of Alerts and Warnings is built. The authors run the system for about two months. At this time 1,065,482 unique web pages have been visited. As input pages the top 17 Alexa web pages of each category and the top 10,000 web pages of Alexa [68] were used. Statistics on the visited pages are given in Table 3.4.

	Value	Rate
Visited Pages	1,065,482	100 %
Unreachable or Empty	86,799	8.15%
Valid Pages	978,683	91.85%
With IFRAMEs	368,963	31,70%
With FRAMEs	32,296	3.30%
Transparent (I)FRAMEs	1,557	0.16%
Clickable Elements	143,701,194	146.83 el./page
Speed Performance	71 days	15,006 pages/day

Table 3.4.: Statistics of the visited web pages of the automated clickjacking detection system

The reason why two plugins in the "Detection unit" are used is to minimise the large number of false positives generated by NoScript. As illustrated in Table 3.5, there are just 130 false positives by using ClickIDS and 502 false positives by using the modified NoScript version. The combination of both plugins shrunk the number to 4.

	Total	True Positives	Borderlines	False Positives
ClickIDS	137	2	5	130
NoScript	535	2	31	502
Both	6	2	0	4

Table 3.5.: Results of the automated clickjacking detection system

All in all the test shows that just 352 web sites (3.8%) have protection mechanism against clickjacking by using frame busting techniques. Just one web page [3] was using the discussed X-FRAME-OPTIONS header of section 3.1.2.

[3] http://flashgot.net

4. Conclusion and outlook

First of all, the sections “Basic clickjacking”, “Advanced Attacks”, and “Clickjacking Tool” are introduced in the chapter “Attack vectors”. The section “Basic clickjacking” is used to show how a clickjacking attack works in general by giving a practical example with the web site “google.com”. Beside that, secondary attacks like “Likejacking” and “Strokejacking” are introduced. Finally, topics about an attack on the “Adobe Flash Player Settings Manager”, the usage of JavaScript, and implementation considerations based on clickjacking are also discussed. The section “Advanced Attacks” shows possibilities to combine clickjacking with techniques of CSRF, XSS, and CSS. Furthermore, in the context of UI redressing, opportunities to perform text injection by drag-and-drop operations and to perform content extraction are identified. Ultimately, it holds information to bypass pop-up blockers and to perform SVG masking, too. The section “Clickjacking Tool” describes a tool to visualise clickjacking techniques in practice.

The next chapter, “Counteractive measures”, focuses on the sections “Frame busting”, “Busting frame busting”, and “Clickjacking detection system”. The section “Frame busting” deals with the topics “JavaScript”, “X-Frame-Options”, and “NoScript”. Each topic shows a way to protect a user against framing attacks, as it is in the case of clickjacking. The “JavaScript” protection mechanism of this section are busted in the next section “Busting frame busting”. It shows techniques to disable frame busting code by using inter alia double framing or the “onBeforeUnload” event. Finally, in the last section, “Clickjacking detection system”, an automated system to analyse clickjacking attacks is described and a study of over one million unique web pages is based on it.

On the one hand, both chapters show that UI redressing is a serious attack that can have terrible effects. On the other hand, there are protection mechanisms like frame busting to provide a certain degree of client-side security. However, in the case of JavaScript there are protection mechanism available to disable frame busting code. Thus, JavaScript should not be the only protection mechanism for protecting a user against UI redressing. It is better to use a combination of different measures like JavaScript frame busting code, the “X-Frame-Options” header and, if a Mozilla-based web browser is used, the plugin NoScript.

In the outlook the question is raised how UI redressing will be treated in the next few years. Generally, it is expected that especially JavaScript frame busting code will be used more often than it is today. It would be also advisable to use “X-Frame-Options” as a protection mechanism, but it is questionable whether the majority will use it. Nevertheless, if these methods are used, there is a useful protection available. In order to identify trends, it would be advisable to use the discussed clickjacking detection system again. Last but not least, the increasing popularity of UI redressing, or specifically clickjacking, determines very likely that there will be some new attacks that are based on UI redressing in the future.

A. Appendix

A.1. Attack vectors

A.1.1. Basic clickjacking

Listing A.1: HTML web page with clickjacking (file: trustedPage.html)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">
<html>
   <head>
      <title>Trusted web page</title>
   </head>
   <body>
    <h1>www.nds.rub.de</h1>
    <form action="http://www.nds.rub.de">
      <input type="submit" value="Go">
    </form>

    <!-- Iframe Code -->
    <iframe id="clickjacking" src="clickjacking.html" width="50" height="300"
        scrolling="no" frameborder="none">
    </iframe>

    <style type="text/css"><!--
      #clickjacking { position: absolute; left: 7px; top: 81px; opacity:0.0}
    //--></style>
   </body>
</html>
```

Listing A.2: Strokejacking HTML, CSS and JavaScript code (file: strokejacking.html)

```
<!-- June 2009 -->
<body onload="document.getElementById('dummy').focus()">
<h3>Totally legitimate CAPTCHA page</h3>
<iframe src="http://www.google.com/" id=victim name=victim style="opacity: 0.2"
    height="20%" width="80%">
</iframe>
<script>
var need = [ 80, 79, 82, 78, 13 ];
var text = [ 'p', 'o', 'r', 'n', '' ];

var cur_pos = 0;

function maybe_redirect(e) {
  var evt = window.event ? event : e;
  var cc = evt.charCode ? evt.charCode : evt.keyCode;

  if (cur_pos >= need.length || need[cur_pos] != cc) return;

  if (window.netscape == undefined) {
    document.getElementById('victim').focus();
  } else {
    frames['victim'].focus();
  }

  document.getElementById('dummy').value += text[cur_pos];
  cur_pos++;

  setTimeout('focus();document.getElementById("dummy").focus()',1);

}
</script>
<p>
<span style="border: 1px solid red; background-color: #FFFFC0; font-size: 20pt;
    padding: 5px">opportunity</span>
<p>
Retype text from the image to authenticate, then hit RETURN:<br>
<input type=text onkeydown="maybe_redirect(event)" id=dummy>

<p>
<font color=gray size=-1>PS. Can we call it "strokejacking"?</font><br>
<font color=gray size=-1>PPS. This version should work in Firefox 3.x, too. <a
    href=shark.jpg>Enjoy</a>.</font>
```

A.1.2. Clickjacking with SVG

Listing A.3: XHTML code to load an iframe and a SVG mask (filename: svg.xhtml)

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:svg="http://www.w3.org/2000/svg">
<body style="background:gray">
  <iframe src="http://www.ruhr-uni-bochum.de/suche/index_en.htm" style="width:800
      px; height:350px; border:none; mask: url(#maskForClickjacking);"/>
  <svg:svg>
    <svg:mask id="maskForClickjacking" maskUnits="objectBoundingBox"
        maskContentUnits="objectBoundingBox">
      <svg:rect x="0.0" y="0.0" width="0.373" height="0.3" fill="white"/>
    <svg:circle cx="0.45" cy="0.7" r="0.075" fill="white"/>
    </svg:mask>
  </svg:svg>
</body>
</html>
```

A.2. Counteractive measures

A.2.1. Frame busting

Listing A.4: HTML code to load logos of the certification authorities Comodo, Verisign and Geotrust by iframes (filename: frameBusting.html)

```
<center>
  <h1>Frame busting?</h1>
  <iframe src="comodo.html" width="120" height="68" scrolling="no" frameborder="none"></iframe>
  <iframe src="verisign.html" width="130" height="88" scrolling="no" frameborder="none"></iframe>
  <iframe src="geotrust.html" width="125" height="70" scrolling="no" frameborder="none"></iframe>
</center>
```

Listing A.5: HTML and CSS code to load the logo of the certification authority Comodo by an iframe (filename: comodo.html)

```
<iframe id="inner" src="https://www.comodo.com/hackerproof/" width="2000" height="2000" scrolling="no" frameborder="none"></iframe>

<style type="text/css"><!--
#inner { position: absolute; left: -1885px; top: -1945px;}
//--></style>
```

Listing A.6: HTML and CSS code to load the logo of the certification authority Verisign by an iframe (filename: verisign.html)

```
<iframe id="inner" src="http://www.verisign.com/trust-seal/index.html" width="2000" height="2000" scrolling="no" frameborder="none"></iframe>

<style type="text/css"><!--
#inner { position: absolute; left: -1350px; top: -998px;}
//--></style>
```

Listing A.7: HTML and CSS code to load the logo of the certification authority Geotrust by an iframe (filename: geotrust.html)

```
<iframe id="inner" src="http://www.geotrust.com/" width="2000" height="2000" scrolling="no" frameborder="none"></iframe>

<style type="text/css"><!--
#inner { position: absolute; left: -1305px; top: -420px;}
//--></style>
```

Bibliography

[1] Marco Balduzzi, Manuel Egele, Engin Kirda, Davide Balzarotti, Christopher Kruegel, “A Solution for the Automated Detection of Clickjacking Attacks,” p. 3, Apr. 2010. [Online]. Available: http://www.iseclab.org/papers/asiaccs122-balduzzi.pdf

[2] “HTML5 Security Cheatsheet - Opera whole-page click hijacking via CSS,” 2010. [Online]. Available: http://heideri.ch/jso/?-link

[3] G. Rydstedt, E. Bursztein, D. Boneh, and C. Jackson, “Busting frame busting: a study of clickjacking vulnerabilities at popular sites,” in *in IEEE Oakland Web 2.0 Security and Privacy (W2SP 2010)*, 2010. [Online]. Available: http://seclab.stanford.edu/websec/framebusting/framebust.pdf

[4] nasdaq.com, “Google Company Financials,” Dec. 2010. [Online]. Available: http://www.nasdaq.com/asp/ExtendFund.asp?symbol=GOOG&selected=GOOG

[5] Robert Hansen, Jeremiah Grossman, “Clickjacking,” Dec. 2008. [Online]. Available: http://www.sectheory.com/clickjacking.htm

[6] Michal Zalewski, “Arbitrary page mashups (UI redressing),” Nov. 2010. [Online]. Available: http://code.google.com/p/browsersec/wiki/Part2#Arbitrary_page_mashups_(UI_redressing)

[7] Himanshu Dwivedi, Chris Clark, David Thiel, *Mobile Application Security.* Osborne, 2010, p. 268.

[8] Paul Stone, “Next Generation Clickjacking,” Apr. 2010. [Online]. Available: http://www.contextis.co.uk/resources/white-papers/clickjacking/Context-Clickjacking_white_paper.pdf

[9] Matthew MacDonald, *Creating a Web Site: The Missing Manual.* O’Reilly Media, 2009, p. 549.

[10] David Flanagan, *JavaScript, The Definitive Guide (Definitive Guides).* O’Reilly Media, 2001, p. 314.

[11] “World Wide Web Consortium (W3C),” World Wide Web Consortium, 2010. [Online]. Available: http://www.w3.org

[12] “Markup Validation Service,” World Wide Web Consortium, 2010. [Online]. Available: http://validator.w3.org

[13] Jason Cranford Teague, *CSS, DHTML, & Ajax: Visual QuickStart Guide (Visual QuickStart Guides).* Peachpit Pr, 2006, p. 226.

[14] Paul Stone, “Next Generation Clickjacking (Video),” Black Hat Europe, 2010. [Online]. Available: https://media.blackhat.com/bh-eu-10/video/Stone/BlackHat-EU-2010-Stone-Next-Generation-Clickjacking.m4v

[15] Himanshu Dwivedi, Chris Clark, David Thiel, *Professional Penetration Testing: Creating and Operating a Formal Hacking Lab.* Syngress Media, 2009, p. 353.

[16] “Strokejacking,” seclists.org, 2010. [Online]. Available: http://seclists.org/fulldisclosure/2010/Mar/232

[17] "Stroke triggered XSS and StrokeJacking," andlabs.org, 2010. [Online]. Available: http://blog.andlabs.org/2010/04/stroke-triggered-xss-and-strokejacking_06.html

[18] "Facebook Worm - Likejacking," Sophos, 2010. [Online]. Available: http://nakedsecurity.sophos.com/2010/05/31/facebook-likejacking-worm/

[19] "Clickjacking with AJAX and Flash," Blog of gnucitizen.org, 2008. [Online]. Available: http://www.gnucitizen.org/blog/more-advanced-clickjacking-ui-redress-attacks/

[20] "Introduction to links and anchors," World Wide Web Consortium, 2010. [Online]. Available: http://www.w3.org/TR/REC-html40/struct/links.html#anchors

[21] Paul Stone, "Next Generation Clickjacking (Slides)," Black Hat Europe, 2010. [Online]. Available: https://media.blackhat.com/bh-eu-10/presentations/Stone/BlackHat-EU-2010-Stone-Next-Generation-Clickjacking-slides.pdf

[22] "Facebookers hit with steamy clickjacking exploit," The Register, 2010. [Online]. Available: http://www.theregister.co.uk/2009/11/23/facebook_clickjacking_exploit/

[23] twitter.com, "Clickjacking Blocked," Feb. 2009. [Online]. Available: http://blog.twitter.com/2009/02/clickjacking-blocked.html

[24] Larry Chaffin, Anton Chuvakin, Champ, III Clark, *Infosecurity 2008 Threat Analysis*. Syngress Media, 2008, pp. 109–111.

[25] Jukka Korpela, "Methods GET and POST in HTML forms," andlabs.org, 2001. [Online]. Available: http://www.cs.tut.fi/~jkorpela/forms/methods.html

[26] Marcus Niemietz, *Authentication Web Pages with Selenium: Vulnerability Analysis and Exploitation of Authentication Web Pages with Selenium*. AVM, 2010, pp. 22–23.

[27] Paul Sebastian Ziegler, *Cross-Site Scripting*. O'Reilly, 2008.

[28] "CSS Id and Class," w3schools.com, 2010. [Online]. Available: http://www.w3schools.com/Css/css_id_class.asp

[29] "jQuery," jquery.com, 2010. [Online]. Available: http://jquery.com/

[30] "jQuery .click()," jquery.com, 2010. [Online]. Available: http://api.jquery.com/click/

[31] "Cascading Style Sheets," W3C, 2010. [Online]. Available: http://www.w3.org/Style/CSS/

[32] "pointer-events for HTML in Firefox 3.6," Mozilla, 2009. [Online]. Available: http://hacks.mozilla.org/2009/12/pointer-events-for-html-in-firefox-3-6/

[33] "HTML5 Security Cheatsheet - Breaking pointer-events:none with nested links," 2010. [Online]. Available: http://heideri.ch/jso/?pointer

[34] "What is a CSS Attribute Selector?" 2010. [Online]. Available: http://webdesign.about.com/od/cssselectors/qt/cssselattribute.htm

[35] "Opera CSS linking extensions," 2010. [Online]. Available: http://www.opera.com/docs/specs/opera9/#xml-css-link

[36] "Drag and drop," World Wide Web Consortium, 2010. [Online]. Available: http://www.w3.org/TR/html5/dnd.html#dnd

[37] "Same origin policy for JavaScript," Mozilla, 2010. [Online]. Available: https://developer.mozilla.org/En/Same_origin_policy_for_JavaScript

[38] "isTrusted property (event)," dottoro.com, 2010. [Online]. Available: http://help.dottoro.com/ljoljvsn.php

[39] “Using the window.open method,” javascript-coder.com, 2010. [Online]. Available: http://www.javascript-coder.com/window-popup/javascript-window-open.phtml

[40] “Scalable Vector Graphics (SVG),” World Wide Web Consortium, 2010. [Online]. Available: http://www.w3.org/Graphics/SVG/

[41] “Even more advanced clickjacking,” GNUCITIZEN, 2008. [Online]. Available: http://www.gnucitizen.org/blog/even-more-advanced-clickjacking/

[42] “Clipping, Masking and Compositing,” World Wide Web Consortium, 2010. [Online]. Available: http://www.w3.org/TR/SVG/masking.html

[43] “//speakers & topics,” Black Hat Europe 2010, 2010. [Online]. Available: http://www.blackhat.com/html/bh-eu-10/bh-eu-10-briefings.html

[44] “Clickjacking Tool,” Context Information Security Limited, 2010. [Online]. Available: http://contextis.com/resources/tools/clickjacking-tool/

[45] “IE8 Security Part VII: ClickJacking Defenses,” Microsoft Corporation, 2009. [Online]. Available: http://blogs.msdn.com/b/ie/archive/2009/01/27/ie8-security-part-vii-clickjacking-defenses.aspx

[46] “The X-Frame-Options response header,” Mozilla Corporation, 2010. [Online]. Available: https://developer.mozilla.org/en/the_x-frame-options_response_header

[47] “NoScript Changelog,” InformAction, 2010. [Online]. Available: http://noscript.net/changelog

[48] “NoScript Firefox extension,” InformAction, 2010. [Online]. Available: http://noscript.net

[49] “ClearClick and Clickjacking,” InformAction, 2010. [Online]. Available: http://noscript.net/faq#faqsec7

[50] “NotScripts Limitations,” 2010. [Online]. Available: http://optimalcycling.com/other-projects/notscripts/limitations/

[51] “Using Environment Variables,” http://oreilly.com, 1996. [Online]. Available: http://oreilly.com/openbook/cgi/ch02_02.html

[52] Adam Barth, Collin Jackson, John C. Mitchell, “Securing frame communication in browsers,” 2009. [Online]. Available: http://www.adambarth.com/papers/2009/barth-jackson-mitchell-cacm.pdf

[53] “onbeforeunload Event,” Microsoft, 2010. [Online]. Available: http://msdn.microsoft.com/en-us/library/ms536907(VS.85).aspx

[54] “PHP header function,” php.net, 2010. [Online]. Available: http://www.php.net/manual/en/function.header.php

[55] “Preventing Frame Busting and Click Jacking (UI Redressing),” 2009. [Online]. Available: http://coderrr.wordpress.com/2009/02/13/preventing-frame-busting-and-click-jacking-ui-redressing/

[56] “window.setInterval,” Mozilla, 2010. [Online]. Available: https://developer.mozilla.org/en/DOM/window.setInterval

[57] “IE8 Security Part IV: The XSS Filter,” Microsoft, 2008. [Online]. Available: http://blogs.msdn.com/b/ie/archive/2008/07/02/ie8-security-part-iv-the-xss-filter.aspx

[58] “Google Chrome - Stable Channel Update,” Google, 2010. [Online]. Available: http://googlechromereleases.blogspot.com/2010/03/stable-channel-update.html

[59] “Using Frames More Securely,” Microsoft, 2008. [Online]. Available: http://blogs.msdn.com/b/ie/archive/2008/01/18/using-frames-more-securely.aspx

[60] “Restricted Sites Zone,” Microsoft, 2010. [Online]. Available: http://msdn.microsoft.com/en-us/

library/ms537183.aspx#restricted

[61] “HTML5 sandbox attribute,” W3C, 2010. [Online]. Available: http://dev.w3.org/html5/spec-author-view/the-iframe-element.html#attr-iframe-sandbox

[62] “designMode - Bugzilla Bug 519928,” Mozilla, 2010. [Online]. Available: https://bugzilla.mozilla.org/show_bug.cgi?id=519928

[63] Michal Zalewski, “Arbitrary page mashups (UI redressing),” Google, 2010. [Online]. Available: http://code.google.com/p/browsersec/wiki/Part2#Arbitrary_page_mashups_(UI_redressing)

[64] “defineSetter,” Mozilla, 2010. [Online]. Available: https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/Object/defineSetter

[65] “JavaScript indexOf() Method,” w3schools.com, 2010. [Online]. Available: http://www.w3schools.com/jsref/jsref_indexOf.asp

[66] Marcus Niemietz, *Authentication Web Pages with Selenium: Vulnerability Analysis and Exploitation of Authentication Web Pages with Selenium.* AVM, 2010, p. 35.

[67] “xdotool - fake keyboard/mouse input, window management, and more,” semicomplete.com, 2010. [Online]. Available: http://www.semicomplete.com/projects/xdotool/

[68] “Alexa Top Sites,” Alexa, 2010. [Online]. Available: http://www.alexa.com/topsites/category/Top