Detection of Visual Clickjacking Vulnerabilities in Incomplete Defenses

Yusuke Takamatsu[†1] Kenji Kono[†1]

Abstract: Clickjacking is a new attack which exploits a vulnerability in web applications. It tricks victims into clicking on something different from what they perceive they are clicking on. The victims may reveal confidential information or start unintended online transactions. Clickjacking attacks compromise visual integrity (called visual clickjacking) or condition integrity (called switchover clickjacking) to deceive victims. We address visual clickjacking in this paper. Visual clickjacking can be prevented if appropriate countermeasures such as frame busting are implemented in web applications. However, the correct implementation is not easy. A trivial mistake in the implementation leads to evasion of the countermeasures. For the correct implementation, web developers must have intimate knowledge on evasion techniques of the countermeasures. In this paper, we propose *Clickjuggler*, an automated tool for checking defenses against visual clickjacking during the development. Clickjuggler generates some types of visual clickjacking attack, performs those attacks on web applications, and checks whether the attacks are successful or not. By automating the process of checking for the vulnerabilities, web developers are released from the burden of checking the correctness of their implementation. Unskillful developers can benefit from Clickjuggler since no special knowledge on a variety of visual clickjacking and evasion techniques is needed to use Clickjuggler. Our experimental results demonstrate that Clickjuggler can detect the visual clickjacking vulnerabilities in 4 real-world web applications and can detect the vulnerabilities in a shorter time than existing tools.

1. Introduction

Clickjacking is known as a new attack which exploits a vulnerability in web applications [1]. It tricks victims into clicking on something different from what they perceive they are clicking on. In principle, an attacker prepares a button, a link, or a form that the victims can not recognize (i.e, hidden from the victims), and induces them to manipulate the hidden elements of the web page. For example, an attacker overlays a *visible* button with an *invisible* button. Although the victims are clicking on the invisible button, they believe they are clicking on the visible button. To prepare invisible buttons, a vulnerable page is made transparent (the buttons on the vulnerable page become invisible) and embedded in an attacker's page (the buttons on the attacker's page are visible). As a result, an attacker can induce the victims to click on the buttons on the vulnerable page.

Clickjacking is a real threat. Sophos [2] reported a clickjacking worm spreads quickly over Facebook users. Using a clickjacking technique, the users are tricked into recommending a page to their Facebook friends. The Setting Manager of Adobe Flash Player was vulnerable to clickjacking [3]. The victims unknowingly click on the button of the access control dialog, and allow remote attackers to hijack the victims' cameras and microphones. According to [4], 30% of Alexa Top 10 web sites, 70% of Top 20 bank web sites, and 80% of 5 popular open-source web applications have no defense against clickjacking in 2012.

Attackers violate visual integrity (called visual clickjacking) or condition integrity (called switchover clickjacking) to deceive victims in clickjacking [5]. In visual clickjacking, attackers compromise the guarantee that victims can fully see and recognize an object on a browser. Visual clickjacking attacks are classified into two categories [5]. In the first category (called basic clickjacking), attackers manipulate an element on a vulnerable page. This is the category the example mentioned earlier falls in. In the second category (called cursorjacking), attackers manipulate cursor feedback to select locations for victims' input events. An attacker hides the real cursor and displays a fake cursor. In switchover clickjacking, attackers compromise the guarantee that victims have enough time to comprehend where they are clicking. This paper focuses on *visual clickjacking* because visual clickjacking is widely recognized as clickjacking and cursorjacking.

Frame busting [6] and X-Frame-Options [7] are well-known defenses against visual clickjacking. Since an attacker tries to embed a vulnerable page in an attacker's page, frame busting prevents a page from being embedded in attackers' pages. If an attempt is made to embed a page to be protected in an attacker's page, the code of frame busting redirects browsers to the protected page. X-Frame-Options controls whether a browser should be allowed or not to render a page in a frame or an iframe tag. To prevent a page from being embedded in attackers' pages, X-Frame-Options is set not to allow browsers to render framed pages. These defenses' policies are enforced on each page so that they are not enforced on each element on the page.

[†1] Presently with Keio University
A preliminary version of this paper appears in IEEE Twelfth Annual International Conference on Privacy, Security and Trust.

Unfortunately, it is quite difficult to correctly implement these defenses. A trivial mistake in the implementation leads to evasion of the defenses. As shown in Section 3.3, there are many subtle issues in the implementation of frame busting. If developers do not have intimate knowledge on evasion techniques of frame busting [8] [9] [10], it is almost impossible to correctly implement frame busting. In addition, X-Frame-Options is not free from incorrect implementations. Joomla 3.x, a content management framework, employs X-Frame-Options to defend against visual clickjacking. However, X-Frame-Options is not effective because of trivial typing errors [11].

Frame busting and X-Frame-Options can not be used to allow a page to be embedded in a third-party page. As shown in Section 3.1.3, some developers need to allow a page to be embedded in a third-party page and disable only sensitive buttons on the page in the third-party page. In this case, the developers must implement element-customized defenses to enforce different policies on different elements on the page. One approach to disable sensitive buttons does not display the buttons on a protected page if the protected page is embedded in the third-party page. Developers make mistakes in the implementation of element-customized defenses because they implement policies for each element and may not spend the time to test all elements.

We propose *Clickjuggler*, an automated tool that checks for incomplete defenses against visual clickjacking in the development phase. Clickjuggler checks for incomplete implementations of frame busting, X-Frame-Options, and the element-customized defenses. Clickjuggler crafts attacker's pages, generates events such as click to perform visual clickjacking attacks, and determines whether the attacks are successful or not.

Clickjuggler deals with a wide range of visual clickjacking attacks from the basic ones to advanced ones. In fact, it performs 8 different attacks including basic clickjacking one, cursorjacking one, and several ones based on evasion techniques of frame busting. Existing tools such as CJTool [12] and BeEF plug-in [13] help developers to craft attacker's pages. These tools focus only on basic clickjacking and do not cover such a wide range of evasion techniques. CJTool does not support any of evasion techniques and BeEF plug-in detects only one evasion technique.

Clickjuggler brings about several benefits by automating the process of checking for the visual clickjacking vulnerabilities. Web developers are released from the burden of checking the incorrectness of their implementation. Clickjuggler detects the vulnerabilities and/or insufficient implementations of the countermeasures, even if the developers are not familiar with a variety of visual clickjacking and evasion techniques. In addition, Clickjuggler shortens the time to test web applications and improves the coverage of the tests. Since modern web sites consist of a large number of buttons and pages, it is difficult for developers to spend the time to test the buttons and pages.

To demonstrate the usefulness of Clickjuggler, we have applied it to four real-world web applications including Joomla [14] (downloaded over 35 million times), WordPress [15] (over 70 million users), MediaWiki [16], and Roundcube [17]. Clickjuggler detects 15, 4, 5 and 2 visual clickjacking vulnerabilities in Joomla, WordPress, MediaWiki, and Roundcube, respectively. Clickjuggler does not cause false positives and false negatives in those web applications. Moreover, we measure the time to detect the vulnerabilities with Clickjuggler. The detection time of Clickjuggler is shorter than CJTool and BeEF plug-in.

The remainder of this paper is organized as follows. We describe the background of clickjacking in Section 2 & 3. Section 4 explains Clickjuggler. Section 5 presents our experimental results. Section 6 discusses related work. Finally, we conclude the paper in Section 7.

2. Clickjacking

2.1 An example of clickjacking

To understand the concept of clickjacking, we show a simple example of clickjacking. In this example, an attacker puts an unreasonably expensive item for sale at a shopping site (shopping.com). The attacker forces a victim to buy the item in the shopping site. Figure 1 illustrates the example of clickjacking. The shopping site is vulnerable to clickjacking. To exploit this shopping site, the attacker prepares the web site (malicious.com), called the *attacker's site*, and induces the victim to visit the attacker's site with, for instance, social engineering tricks (Step 1).

The page in the attacker's site, called the *attacker's page*, is constructed so that it is overlaid with the target page in the vulnerable site (shopping.com). The attacker's page obtains the target page and embeds it in the attacker's page (Step 2). To deceive the victim, the target page is made transparent by the attacker's page so that the victim cannot perceive the presence of the target page (Step 3). To force the victim to buy the item in shopping.com, the decoy button is shown on the attacker's page and precisely overlaid with the "Buy" button on the target page (Step 4). Since the target page is transparent, the victim cannot perceive the presence of the "Buy" button; he clicks on the "Buy" button although he believes he is clicking on the decoy button (Step 5). Consequently, the victim unintentionally buys the item in shopping.com (Step 6). The attacker can obtain a payment for the item.

2.2 Methods of clickjacking

Existing clickjacking attacks compromise visual integrity (called visual clickjacking) or condition integrity (called switchover clickjacking) to deceive victims [5]. In this paper, we address visual clickjacking.

(1) *Visual clickjacking*: the attackers compromise the guarantee that victims can fully see and recognize an object on a browser. Visual clickjacking attacks are classified into two categories based on the methods to deceive victims [5]. In the first category (called basic clickjacking), attackers manipulate an element on a target page. An approach is to overlay a web page with a transparent page as explained in the previous section. The victims click on buttons on the target page although they believe that they manipulate the attacker's page.

In the second category (called cursorjacking), attackers manipulate cursor feedback to select locations for victims' input events. The attackers display a fake cursor and hide the real one. Since the victims can not recognize correct locations of the cursor, they click on an unintended button on a target page. For example, an attacker's page displays a fake cursor at 200 pixels right of the

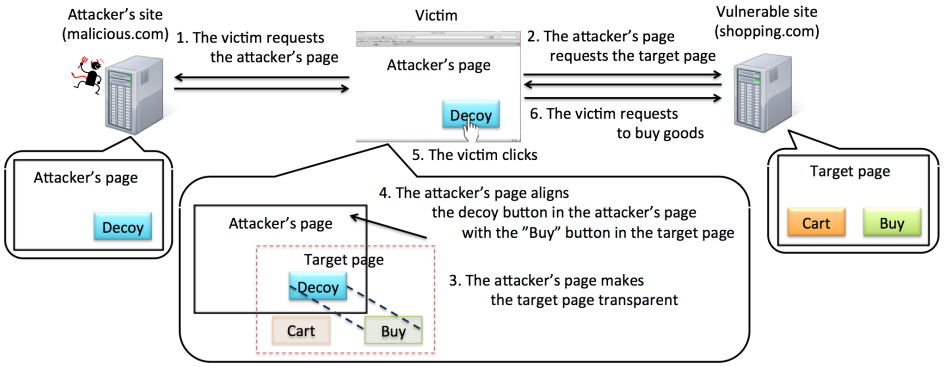

Fig. 1 Example of clickjacking. The victim accesses the attacker's page (Step 1). The attacker's page requests the target page (Step 2). To deceive the victim, the target page is made transparent by the attacker's page (Step 3). To force the victim to click on the "Buy" button, the decoy button on the attacker's page is precisely overlaid with the "Buy" button on the target page (Step 4). The victim clicks on the decoy button and sends the request (Steps 5 & 6).

real cursor and hides the real cursor to deceive a victim. To hide the real cursor, it uses the CSS cursor property which controls the type of cursor. To force the victim to click on a target button, it sets a decoy button at 200 pixels right of the target button. The victim clicks on the target button although he believes he clicks on the decoy button.

(2) *Switchover clickjacking*: the attackers compromise the guarantee that victims have enough time to comprehend where they are clicking. For example, the attackers move a button on a target page on top of a decoy button shortly after the victim hovers the cursor over the decoy button. The victim can not react to the condition change and clicks on the button.

3. Defenses and Evasion Techniques

This section introduces the well-known defenses against visual clickjacking: 1) frame busting and 2) X-Frame-Options. Since those defenses do not allow a page to be embedded in another page, the developers can not design a page that is allowed to be embedded but still prevents sensitive elements from being manipulated. This section also introduces 3) element-customized approaches to design such a page. Section 3.2 shows the difficulties that lie in the implementation of those defenses. Section 3.3 elaborates Section 3.2 from the viewpoint of frame busting.

3.1 Defenses against Visual Clickjacking

3.1.1 Frame Busting

Frame busting is a technique to prevent a page from being embedded in another page [6]. In visual clickjacking, a vulnerable page is embedded in an attacker's page to deceive victims. In frame busting, a small piece of code (usually written in JavaScript) is embedded in a page to be protected. If an attempt is made to embed a protected page in an attacker's page, the frame busting code redirects the browser to the original site so that the victims can see the protected page instead of the attacker's page.

Frame busting's policy is enforced on all buttons, links, and forms on a single web page and thus, can not be used if a web page is allowed to be embedded in a third-party page. As shown in Section 3.1.3, some web applications need to enforce different policies on different elements on a single web page.

3.1.2 X-Frame-Options

X-Frame-Options HTTP response header indicates whether a browser should be allowed or not to display a page in a frame or an iframe [7]. It can be used to avoid visual clickjacking because a page in which X-Frame-Options is set can not be embedded in other pages. Three attributes can be specified for X-Frame-Options: 1) `DENY`, 2) `SAMEORIGIN`, and 3) `ALLOW-FROM`. `DENY` prohibits a page from being displayed in an iframe. `SAMEORIGIN` allows only the pages that belong to the same origin to be displayed in an iframe. `ALLOW-FROM` allows the pages that originate from the pre-defined origins to be displayed in an iframe. As with frame busting, the policy of X-Frame-Options is not enforced on each element on a single page because it is enforced on the single web page.

There are web application frameworks which support X-Frame-Options, but developers need to check the correctness of X-Frame-Options in these frameworks. This is because the developers make mistakes in setting policies. For example, Ruby on Rails [18] and django [19] support X-Frame-Options. These frameworks set X-Frame-Options for all responses in sites by setting policies. However, the developers might make mistakes in setting the policies because they set the policies on every page.

3.1.3 Element-customized approaches

Frame busting and X-Frame-Options can not allow a page to be embedded in a third-party page. Suppose that there are the "Buy" and the "Cart" button on the vulnerable page in Fig. 1 and the page is allowed to be embedded in a third-party page. If it is embedded, the "Buy" button is not clickable but the "Cart" button is clickable since the "Cart" button does not start any critical transactions. Neither frame busting nor X-Frame-Options can be applied to this page.

One approach to dealing with this situation is to implement a defense customized for each element. A small piece of script code is associated with each element. If the page to be protected is embedded in a third-party page, the code, for instance, does not display security-critical buttons such as the "Buy" button. Another approach is to stop sending cookies if the security-critical buttons are clicked. If no cookies are sent, no critical transactions can be initiated.

3.2 Difficulties in Implementing Defenses

It is not easy to correctly implement these defenses. To implement frame busting, the developers must have intimate knowledge on evasion techniques of frame busting [8] [9] [10]. To understand subtle issues in the implementation of frame busting, Section 3.3 introduces seven evasion techniques of frame busting. X-Frame-Options is not free from trivial mistakes. Joomla 3.x (a widely used content management system) adopts X-Frame-Options to avoid visual clickjacking. However, X-Frame-Options is mis-spelled as X-Frames-Options ('s' following 'Frame' not needed) and SAMEORIGIN is mis-spelled as SAME-ORIGIN ('-' not needed) in `behavior.php`. This implies that the developers have not devoted time to check the correctness of the defense.

The element-customized defenses are headache. Since each button, link and form on every page must accompany a small piece of code that controls its behavior when embedded in a third-party page, the developers tend to make mistakes in the implementation or forget to write the code. It would be possible to extend existing frameworks to support the development of the element-customized defenses. In such a framework, however, the developers would be requested to set policies on each button, link and form on every web page. To check the correctness of the policy setting, we need another system that guarantees the web application is free from the visual clickjacking vulnerabilities.

3.3 Evasion Techniques of Frame Busting

3.3.1 double framing

As shown in Section 3.1.1, frame busting code redirects a browser to the original site. If `parent.location` is used to redirect a browser, the frame busting code can be evaded by *double framing*. In double framing, an attacker nests a target page in two frames of two attacker's pages. By nesting the target page in two frames, redirecting with `parent.location` in the frame busting code becomes security violation due to descendant policy and is disabled by browsers. In descendant policy a frame can navigate only its descendant frame to a different URI so that the descendant frame (the frame busting code in the protected page) can not redirect the parent frame.

3.3.2 onBeforeUnload Event

Frame busting can be evaded if frame busting code does not stop displaying a page to be protected when the redirection is canceled. An attacker prepares a page that registers an onBeforeUnload handler, which is invoked when the attacker's page is unloaded because of the redirection. The handler asks a user if the redirection should be canceled. If the user chooses the cancel, the browser cancels the redirection and the protected page remains being embedded into the attacker's page.

3.3.3 No-Content Flushing

Frame busting can be evaded if frame busting code is not placed in a head tag or the forefront of the body tag. If the frame busting code is placed at the end of the body tag, it is evaluated after the protected page is rendered, and then attempts to redirect the browser. To keep the protected page rendered, an attacker's page registers an onBeforeUnload handler. Recall that this handler is invoked when the frame busting code attempts to redirect the browser. The handler submits a request to a server which responds with a HTTP/1.1 204 No Content. On the receipt of No Content, the browser flushes the request pipeline and the redirection is canceled.

3.3.4 manipulating Referrer

Frame busting code checks `document.referrer` to allow a page to be framed in pre-defined third-party pages. If a URI of the pre-defined pages is in `document.referrer`, the frame busting code allows the page to be embedded. If this check is done by using simple string search, the frame busting code can be evaded since an attacker can embed a string that matches the search. For example, if a page to be protected is allowed to be embedded in a page originating from safe.com, an attacker can generate a URI of attacker.com/page?s=safe.com.

3.3.5 Browser-dependent Approaches

Three evasion techniques of frame busting exploit browser-specific behaviors. This is another reason it is hard to implement the code of frame busting.

clobbering `location` variable: To disable frame busting code, an attacker's page can make use of security violation caused by accessing a local variable in other pages. Frame busting code accesses a global variable, `top.location` (the origin of the parent page), to confirm the page is allowed to be embedded in the parent page. To disable this type of frame busting code, an attacker's page redefines the `location` variable as a local variable. When the frame busting code accesses `top.location` (now, it is a *local* variable), the frame busting code is disabled due to security violation. This evasion is specific to IE 7 or Safari 4.0.4.

restricting JavaScript: If JavaScript code is disabled, frame busting code can be evaded because it is usually written in JavaScript. If frame busting code is put in an `iframe` in Firefox and Chrome, it can be disabled by specifying a `sandbox` attribute in the `iframe` tag. Frame busting code can be also disabled by setting a `security` attribute to "restricted" in IE or turning on a `designMode` property in IE 8 and Firefox.

XSS filter: To disable frame busting code, an attacker's page can make use of XSS filters in IE 8 and Chrome. The XSS filters disable the script code included in a response of the HTTP request. If an attacker's page extracts the frame busting code and embeds it in a URI of an `iframe` tag, the XSS filters disable the frame busting code.

4. Clickjuggler

As shown in Sections 3.2 and 3.3, there are many words of caution to implement the defenses against visual clickjacking. We propose Clickjuggler, an automated tool to check the correctness of implementation of defenses against visual clickjacking. To check for the visual clickjacking vulnerabilities, Clickjuggler performs actual attacks on web applications. The current version of Clickjuggler covers basic clickjacking attack, cursorjacking attack, and attacks based on six evasion techniques of frame busting. As shown in Section 4.3, Clickjuggler can cover other types of visual clickjacking attack. Clickjuggler prepares some types of attacker's page to perform the visual clickjacking attacks, manipulates the attacker's pages as a victim, and determines whether the attacks are successful or not.

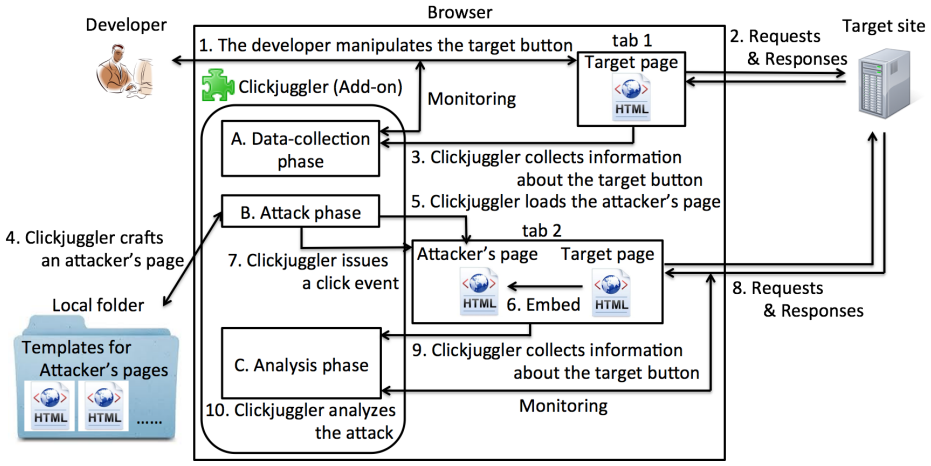

Fig. 2 Overview of Clickjuggler. In the data-collection phase, Clickjuggler collects information about the target button (Step 1 to Step 3). In the attack phase, Clickjuggler crafts some types of attacker's page (Step 4) and emulates actual attacks (Step 5 to Step 8). In the analysis phase, Clickjuggler determines whether the attack is successful or not (Steps 9 & 10).

4.1 Overview

Clickjuggler consists of three phases: 1) data-collection phase, 2) attack phase, and 3) analysis phase as shown in Figure 2. In the data-collection phase, Clickjuggler collects information about target buttons that will be needed to generate attacker's pages for clickjacking attacks. For example, a URI of a page which has a target button, coordinates, width, and height of the target button, and events to activate the button are collected. Clickjuggler collects these pieces of information while developers use target buttons during the test phase of a web application. During the test phase, the developer opens the target page and clicks on the target buttons to confirm the buttons work well (Steps 1 & 2). Clickjuggler collects necessary information through these operations (Step 3). The details are presented in the next section.

In the attack phase, Clickjuggler crafts some types of attacker's page (Step 4). Thereafter, Clickjuggler emulates actual attacks (Step 5 to Step 8). Clickjuggler has the browser load the attacker's page. The attacker's page uses the URI collected in the data-collection phase to embed the target page in the attacker's page. Clickjuggler issues events to emulate victims' behavior by using the coordinates of the target button and the events collected in the data-collection phase.

In the analysis phase, Clickjuggler determines whether the attack is successful or not (Steps 9 & 10). To conclude the basic clickjacking is successful, Clickjuggler confirms that the target button is not displayed and the target button is clicked when a click event is issued.

Clickjuggler requires developers to perform three operations because it executes three phases. First, they manipulate the target buttons in data-collection phase as shown in Section 4.2. Second, they provide a special keyword in data-collection phase as shown in Section 4.4. Third, they manipulate the confirmation dialog in the attack phase as shown in Section 4.5.3. The details of three operations are presented in each section.

Although the developers are required to perform three operations, the manual detection of visual clickjacking vulnerabilities is automated by Clickjuggler. They need not to collect information about the target buttons to craft attacker's pages. They need not to craft some types of attacker's page for visual clickjacking and perform actual attacks. They need not to conclude whether each attack is successful or not.

4.2 Data-collection phase

Clickjuggler collects information about the target buttons while a developer manipulates the target buttons. When a developer opens a target page to check for whether buttons on the target page work well or not, Clickjuggler records a URI of the opened page. Figure 3 illustrates what kind of information Clickjuggler collects. The example used in this figure is the same as that in Fig 1. In this example, when the developer opens the purchase page, Clickjuggler records the URI "http://shopping.com/purchase" (Steps 1 & 2).

After opening the web page, the developer clicks on a target button or inputs data to a target form. Clickjuggler obtains the coordinates, the width and the height of the clicked button or the clicked form, and events generated to perform the operations. In Fig. 3, the developer clicks on the "Buy" button (Step 3). Clickjuggler records the coordinates of the "Buy" button (Step 4). To reproduce the developer's behavior, Clickjuggler maintains the order in which objects are manipulated. For example, if an item in a checkbox menu is clicked on, the order of the manipulated objects is significant.

During the data-collection phase, Clickjuggler takes screen shots of the clicked button or the clicked form. These screen shots are later used to determine whether the attack is successful or not. In Fig. 3, Clickjuggler takes a screen shot of the "Buy" button (Step 5).

4.3 Attack Phase

Clickjuggler crafts some types of attacker's page for visual clickjacking attacks. Clickjuggler prepares some template HTML files to craft the attacker's pages. In those templates the methods to deceive victims or to evade the frame busting are implemented. Clickjuggler fills in *holes* of the template HTML files, making use

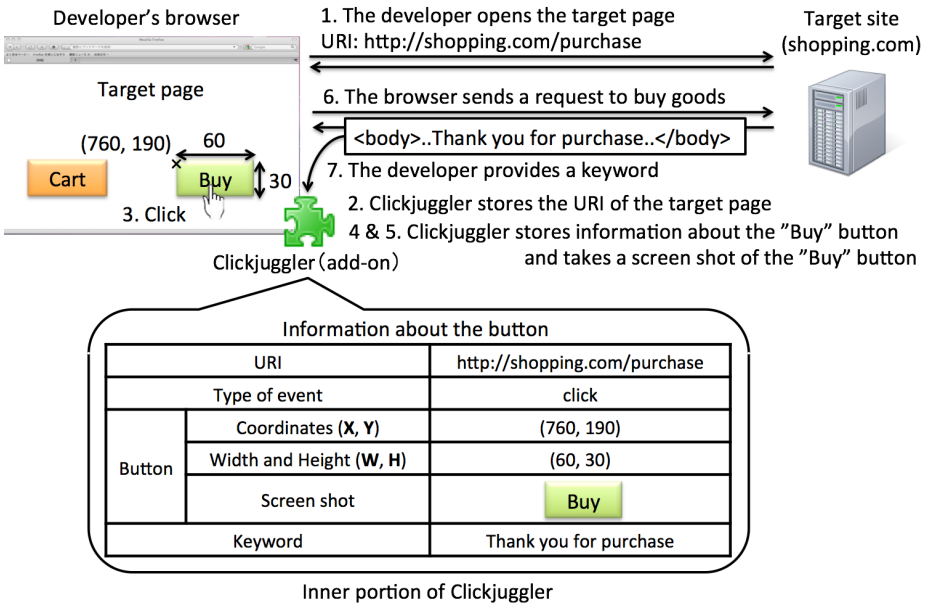

Fig. 3 Data-collection phase. The developer accesses the target page (Step 1) and clicks on the target button on the target page (Steps 3 & 6). Clickjuggler stores the URI of the target page (Step 2), information about the target button (Step 4), and takes the screen shot of the target button (Step 5). Finally, the developer provides the keyword to Clickjuggler (Step7).

of the information collected in the data-collection phase.

Clickjuggler prepares two template HTML files for basic clickjacking and cursorjacking and six template HTML files for the evasion techniques. The templates for basic clickjacking and cursorjacking generate attacker's pages that emulate basic clickjacking and cursorjacking introduced in Section 2.1 and Section 2.2, respectively. Each of the six templates generates an attacker's page that emulates one of the evasion techniques introduced in Section 3.3. We show detail of the templates in Section 4.5.

These templates detect basic clickjacking vulnerability, cursorjacking vulnerability, or incomplete countermeasures. If a web page is not equipped with any defenses, the templates for basic clickjacking and cursorjacking can detect the vulnerabilities. If a web page implements incomplete frame busting code, at least one of six templates can detect the vulnerability. If a web page implements invalid X-Frame-Options or incomplete element-customized defenses, the templates for basic clickjacking and cursorjacking can detect the vulnerability.

Clickjuggler can cover other and new types of visual clickjacking attacks by preparing new templates for the attacks. This is because the methods to deceive victims and to evade the frame busting are implemented in the templates. The method is the difference between visual clickjacking attacks.

The attack results of templates provide information to modify the incomplete countermeasure to developers. This is because the developers can identify the cause that the incomplete countermeasure is evaded from the results. For example, if a template of "double framing" as shown in Section 3.3 judges incomplete frame busting code is vulnerable, a developer can identify that the cause is the redirection with `parent.location` in the code. The developer can correct `parent.location` in the code.

To initiate visual clickjacking attacks, Clickjuggler has the browser connect to the attacker's page generated from the template and starts emulating the behavior of victims by replaying the events recorded during the data-collection phase. Clickjuggler lets the browser emulate the behavior (e.g., a click event and a mousemove event) to the attacker's page.

Note that an attacker's page must have an origin different from that of a target page. This is important because some defenses (e.g., X-Frame-Options) rely on the origins of web pages. Clickjuggler saves the attacker's page as local files so that the origin of the attacker's page differs from the target page.

4.4 Analysis phase

To conclude visual clickjacking attacks are successful, Clickjuggler confirms conditions are met for each template. The conditions for basic clickjacking differ from the ones for cursorjacking. This is because basic clickjacking manipulates the target element on the target page and cursorjacking manipulates cursor feedback. We describe the conditions for basic clickjacking in next section and the ones for cursorjacking in Section 4.4.2.

4.4.1 Basic clickjacking

To conclude basic clickjacking is successful, Clickjuggler confirms two conditions are met. First, Clickjuggler confirms the target button becomes transparent when it is embedded in the attacker's page. To confirm the first condition, Clickjuggler compares the screen shots of the target button before and after the button is loaded into the attacker's page.

Second, Clickjuggler confirms the target button is clicked on when a click event is sent to the location where a decoy button is displayed. We consider three design alternatives to determine whether the target button is clicked on or not.

Design 1: Clickjuggler checks to which site a request is sent after the button is clicked on. If the request is sent to the site to be framed, Clickjuggler concludes the button is clicked on. This approach misjudges a button is vulnerable if the button im-

plements the defense in which cookies are deleted as shown in Section 3.1.3.

Design 2: Clickjuggler checks contents of the HTML file returned after the button is clicked on. If the contents of the HTML file is the same as that obtained in the data-collection phase, Clickjuggler concludes the target button is clicked on. This is because the site returns the HTML file obtained in the data-collection phase by clicking on the target button. This approach does not work well if the response differs time to time.

Design 3: As with the design 2, Clickjuggler checks the contents of the HTML file returned after the button is clicked on. To deal with the buttons that return different pages, the developer specifies a special keyword during the data-collection phase that identifies the page to be returned (Step 7 in Fig 3). Clickjuggler employs the design 3 because the design 1 and 2 cause false positives in some defenses and pages.

4.4.2 Cursorjacking

To conclude cursorjacking is successful, Clickjuggler confirms two conditions. First, Clickjuggler confirms a fake cursor is displayed to force victims not to recognize the correct position of the real cursor. Clickjuggler compares the screen shots of the decoy button on the attacker's page before and after the fake cursor is moved. If the screen shots differ, the fake cursor is displayed.

It is possible that Clickjuggler causes false positive when the target pages change dynamically. This is because the change in the target pages affects the screen shots of the decoy button. To prevent the screen shots from changing, Clickjuggler covers the target page with an element and sets the decoy button on the element as shown in Section 4.5.2.

Second, Clickjuggler confirms the target button on the attacker's page is clicked when the target button is issued a click event to in a condition that the attacker's page visually deceives a victim. To confirm the condition Clickjuggler uses the same way as basic clickjacking.

Clickjuggler does not confirm the real cursor is hidden to force victims not to recognize the real cursor. This is because it is possible to succeed cursorjacking even if the real cursor is displayed. The victims click on the target button if they focus on the fake cursor when the real cursor and the fake cursor are displayed.

There are techniques [5] to let users pay attention to the correct position of the real cursor. These techniques do not prevent cursorjacking completely. Clickjuggler determines that a target button which implements these techniques is vulnerable. This is because it is possible to succeed cursorjacking.

4.5 Implementation

Clickjuggler is implemented as a plug-in for Firefox 20.0.1 and 3.6.8. Our implementation makes use of Firefox plug-in interface, but we believe Clickjuggler can be ported easily to other browsers such as IE, Chrome, and Safari because our implementation uses only the common API functions. Table 1 lists the API that Clickjuggler uses.

Attackers can abuse Clickjuggler to scan vulnerable web applications. We can consider some approaches to prevent attackers from abusing Clickjuggler. For example, Clickjuggler performs authentication between the developers and the target sites. However, Clickjuggler does not incorporate such approaches because the purpose of this paper is to detect some types of visual clickjacking vulnerability.

Table 1 Web API interfaces

Web API interface	Summary [20]
window.content	Returns a reference to the content element in the current window
document.URL	Returns the string URL of the HTML document
Element.getBoundingClientRect	Returns a text rectangle object that encloses a group of text rectangles
document.createEvent	Creates an event of the type specified
event.initMouseEvent	Initializes the value of a mouse event once it's been created
EventTarget.dispatchEvent	Dispatches an Event at the specified EventTarget, invoking the affected EventListeners in the appropriate order
EventTarget.addEventListener	Registers the specified listener on the EventTarget it's called on

```
1 <IFRAME style="opacity:0;" src="URI">
2 </IFRAME>
3 <BUTTON style="left:X; top:Y; width:WIDTH; height:HEIGHT;
4   position:absolute; z-index:-1;">Decoy
5 </BUTTON>
```

Fig. 4 Template for basic clickjacking

Clickjuggler uses the templates for basic clickjacking and cursorjacking, and the six templates to craft the attacker's pages as shown in Section 4.3. Clickjuggler generates all attacks introduced in Section 3.3 except for the evasion techniques of restricting JavaScript with security attribute, clobbering location variable, and XSS filter. These evasion techniques are not supported because these evasion techniques are the attack specific to IE, Chrome, and Safari. We introduce the templates for basic clickjacking and cursorjacking, and the six templates, and discuss the evasion techniques for IE, Chrome, and Safari.

4.5.1 Template for basic clickjacking

As shown in Figure 4, attacker's pages, crafted from the template for basic clickjacking, display a decoy button that is overlaid with a target page which is made transparent. In the template, an `iframe` tag is used to load a target page from a *URI* (line 1). To make the loaded page transparent, an `opacity` is set in the style attribute of the `iframe` tag. A decoy button is displayed at coordinates (X, Y) with height *HEIGHT* and width *WIDTH* (line 3); i.e. precisely at the location of the target button on the target page. To craft an attacker's page, Clickjuggler fills in *URI*, *X*, *Y*, *HEIGHT*, and *WIDTH*, using the information obtained in the data-collection phase.

4.5.2 Template for cursorjacking

As shown in Figure 5, attacker's pages, crafted from this template for cursorjacking, hides a real cursor, displays a fake cursor, and covers a target page to prevent false positive. To hide a real cursor, `cursor:none` is set in the `style` attribute of the `body` tag (line 1). The `Img` tag is used to load a bitmap of a fake cursor to be displayed (line 2). JavaScript code in the `script` tag moves the fake cursor at coordinates (x coordinate of the real cursor +200, y coordinate of the real cursor) (line 3 to line 11). The `Button` tag sets a decoy button at coordinates $(X + 200, Y)$ with width $WIDTH$ and height $HEIGHT$ (line 12 to line 14). The `iframe` tag is used to load a target page from "URI" in the iframe (line 15). The `Div` tags overlay the target page without covering the

```
1  <BODY style="cursor: none;">
2  <img src="cursor.png"/>
3  <SCRIPT>
4    var move = function(e){
5      a = coordinate x of the real cursor + 200;
6      b = coordinate y of the real cursor;
7      // move a fake cursor on coordinates (a, b);
8    };
9    //Continuously catch mousemove event
10   document.body.addEventListener('mousemove', move, true);
11 </SCRIPT>
12 <BUTTON style="left:X+200; top:Y;
13                width:WIDTH; height:HEIGHT;
14                position:absolute; z-index:-1;">Decoy<BUTTON>
15 <IFRAME src="URI"></IFRAME>
16 <DIV style="..."></DIV>...<DIV style="..."></DIV>
17 </BODY>
```

Fig. 5 Template for cursorjacking

```
1  var prevent_bust = 0;
2  // Event handler to catch execution of redirection
3  window.onbeforeunload = function(){
4    prevent_bust++
5  };
6  // Continuously monitor whether redirection is
7  // executed or not
8  setInterval(function(){
9    if(prevent_bust > 0){
10     prevent_bust -= 2;
11     // Get "No Content"
12     window.top.location = 'http://no-content-204.com/';
13   }
14 }, 1);
```

Fig. 6 Template for *No-Content Flushing*

target button (line 16).

4.5.3 Six templates for evasion techniques

Each template extends the template for basic clickjacking to generate each of the evasion techniques. The first template, *double framing*, consists of two template files because a target page is nested in two attacker's pages. The one template loads the other template in a frame, the other template is the same as the template for basic clickjacking.

The second template, *the onBeforeUnload event*, cancels redirection requested by the frame busting code. To cancel the redirection an attacker's page registers an onBeforeUnload handler. This handler asks the developer to cancel the redirection with a confirmation dialog. Clickjuggler requires developers to select the cancel of the redirection in attack phase. Clickjuggler can not select the cancel of the redirection because Clickjuggler implemented in JavaScript can not manipulate the confirmation dialog. In the second template, the code is added as follows.

```
window.onbeforeunload=function(){return "Do you want to exit?";}
```

The third template, *No-Content flushing*, cancels redirection requested by the frame busting code as shown in Figure 6. To cancel the redirection an attacker's page registers an onBeforeUnload handler. This handler catches the redirection requested by the frame busting code (line 3 to line 5). This template continuously monitors whether the redirection is executed or not and issues a request to a server which responds with a HTTP/1.1 204 No Content (line 8 to line 14).

The fourth template, *manipulating Referrer*, manipulates `document.referrer`. Clickjuggler checks for whether the frame busting code uses simple string search to check `document.referrer` or not. To check the frame busting code Clickjuggler manipulates a URI of the fourth template because the URI is assigned to `document.referrer`. Clickjuggler embeds a URI of a target page into the URI of the fourth template. This is because we assume that developers allow pages in the target site to embed pages in the target site.

The fifth template, *restricting JavaScript with sandbox*, prohibits the execution of JavaScript code. An attacker's page specifies a `sandbox` attribute in an `iframe` tag to prevent the JavaScript code from running. In the fifth template, the `sandbox` attribute is added as follows. "allow-forms" allows form submission in the iframe.

```
<IFRAME style="opacity:0;"
        sandbox="allow-forms" src="URI"></IFRAME>
```

The sixth template, *restricting JavaScript with designMode*, prohibits the execution of JavaScript code. An attacker's page can prevent the frame busting code in an iframe from working by setting "on" as a value of the `designMode` property of the `iframe` tag. In the sixth template, the `designMode` property is added as follows.

```
<IFRAME style="opacity:0;" id="tgt" src="URI"></IFRAME>
document.getElementById("tgt").contentDocument.designMode="on";
```

4.5.4 Evasion techniques for Chrome, IE, or Safari

We believe Clickjuggler can check for the visual clickjacking vulnerabilities with two evasion techniques (clobbering `location` variable and restricting JavaScript with `security` attribute) by making Clickjuggler as Chrome, IE, or Safari plug-in. Clickjuggler can craft the attacker's pages which make use of two techniques from two templates.

Two templates extend the template for basic clickjacking as with the six templates. The template, *clobbering `location` variable*, manipulates the `location` variable. An attacker's page redefines the `location` variable as a local variable to disable the frame busting code due to security violation. In this template, the script code is added as follows.

```
<SCRIPT> var location="clobbered" </SCRIPT>
```

The template, *restricting JavaScript with `security` attribute*, prohibits the execution of JavaScript code in an iframe. An attacker's page prevents the frame busting code in the iframe from working by setting "restricted" as a value of `security` attribute of the `iframe` tag. In this template, the `security` attribute is added as follows.

```
<IFRAME style="opacity:0;"
        security="restricted" src="URI"></IFRAME>
```

Clickjuggler which is implemented as Chrome and IE plug-in can not check for the visual clickjacking vulnerabilities with the evasion technique of XSS filter because the current version of Clickjuggler can not craft the attacker's pages which make use of XSS filters. We discuss the reason in Section 5.3.

5. Experiments

We evaluate the accuracy and the performance of Clickjuggler to demonstrate the usefulness of Clickjuggler. To evaluate the accuracy, we confirm that Clickjuggler can detect correctly visual clickjacking vulnerabilities in real-world web applications in next section. To evaluate the performance, we compare the detection time of Clickjuggler with existing tools in Section 5.2. Moreover, we discuss about limitation of Clickjuggler in Section 5.3.

Table 2 Detection result of visual clickjacking

Web Application	target button	Vulnerability	Detection results of Clickjuggler								
			Template*								Result
			basic	1	2	3	4	5	6	cursor	
Joomla 1.6.1	Normal link	vul.	vul.	vul.	vul.	vul.	vul.	vul.	vul.	vul.	vul.
	Edit profile	vul.	vul.	vul.	vul.	vul.	vul.	vul.	vul.	-	vul.
	Normal link (admin page)	vul.	vul.	vul.	vul.	vul.	vul.	vul.	vul.	vul.	vul.
	Edit article (admin page)	vul.	vul.	vul.	vul.	vul.	vul.	No vul.	No vul.	-	vul.
	Edit user's info (admin page)	vul.	vul.	vul.	vul.	vul.	vul.	No vul.	No vul.	-	vul.
Joomla 2.5.7	Normal link	vul.	vul.	vul.	vul.	vul.	vul.	vul.	vul.	vul.	vul.
	Edit profile	vul.	vul.	vul.	vul.	vul.	vul.	vul.	vul.	-	vul.
	Normal link (admin page)	vul.	vul.	vul.	vul.	vul.	vul.	vul.	vul.	vul.	vul.
	Edit article (admin page)	vul.	vul.	vul.	vul.	vul.	vul.	No vul.	No vul.	-	vul.
	Edit user's info (admin page)	vul.	vul.	vul.	vul.	vul.	vul.	No vul.	No vul.	-	vul.
Joomla 3.0.2	Normal link	vul.	vul.	vul.	vul.	vul.	vul.	vul.	vul.	vul.	vul.
	Make article	vul.	vul.	vul.	vul.	vul.	vul.	No vul.	No vul.	-	vul.
	Edit profile	No vul.	No vul.	No vul.	No vul.	No vul.	No vul.	No vul.	No vul.	-	No vul.
	Normal link (admin page)	vul.	vul.	vul.	vul.	vul.	vul.	vul.	vul.	vul.	vul.
	Edit article (admin page)	vul.	vul.	vul.	vul.	vul.	vul.	No vul.	No vul.	-	vul.
	Edit user's info (admin page)	vul.	vul.	vul.	vul.	vul.	vul.	No vul.	No vul.	-	vul.
Roundcube 0.4.1	Normal link	vul.	vul.	vul.	vul.	vul.	vul.	No vul.	No vul.	vul.	vul.
	Edit setting	vul.	vul.	vul.	vul.	vul.	vul.	No vul.	No vul.	-	vul.
Roundcube 0.7.0	Normal link	No vul.	No vul.	No vul.	No vul.	No vul.	No vul.	No vul.	No vul.	No vul.	No vul.
	Edit setting	No vul.	No vul.	No vul.	No vul.	No vul.	No vul.	No vul.	No vul.	-	No vul.
Roundcube 0.7.0 (Firefox 3.6.8)	Normal link	No vul.	No vul.	No vul.	No vul.	No vul.	No vul.	No vul.	No vul.	No vul.	No vul.
	Edit setting	No vul.	No vul.	No vul.	No vul.	No vul.	No vul.	No vul.	No vul.	-	No vul.
MediaWiki 1.16.0	Normal link	vul.	vul.	vul.	vul.	vul.	vul.	vul.	vul.	vul.	vul.
	Edit article	vul.	vul.	vul.	vul.	vul.	vul.	No vul.	No vul.	-	vul.
	Normal link (admin page)	vul.	vul.	vul.	vul.	vul.	vul.	vul.	vul.	vul.	vul.
	Edit article (admin page)	vul.	vul.	vul.	vul.	vul.	vul.	No vul.	No vul.	-	vul.
	Edit user's info (admin page)	vul.	vul.	vul.	vul.	vul.	vul.	No vul.	No vul.	-	vul.
WordPress 3.1.2	Normal link	vul.	vul.	vul.	vul.	vul.	vul.	vul.	vul.	vul.	vul.
	Post comment	vul.	vul.	vul.	vul.	vul.	vul.	vul.	vul.	-	vul.
	Normal link (admin page)	vul.	vul.	vul.	vul.	vul.	vul.	vul.	vul.	vul.	vul.
	Add user (admin page)	vul.	vul.	vul.	vul.	vul.	vul.	No vul.	No vul.	-	vul.

Template*: basic is *Template for basic clickjacking*, 1 is *double framing*, 2 is *The onBeforeUnload event*, 3 is *No-Content flushing*, 4 is *manipulating Referrer*, 5 is *restricting JavaScript with sandbox*, 6 is *restricting JavaScript with designMode*, and cursor is *Template for cursorjacking* as shown in Section 4.5.
"-" means "not checked by Clickjuggler because the template for the cursorjacking targets normal links and buttons."

Table 3 Detection time of Clickjuggler, CJTool, and BeEF plug-in

Web Application	Clickjuggler			CJTool			BeEF plug-in		
	preparatory (s)	test (s)	total (s)	preparatory (s)	test (s)	total (s)	preparatory (s)	test (s)	total (s)
Joomla 1.6.1	9.78	10.7	20.5	11.5	15.9	27.4	11.5	26.3	37.8
Joomla 2.5.7	9.44	10.7	20.1	11.5	15.9	27.4	11.5	25.4	36.9
Joomla 3.0.2	9.41	10.7	20.1	10.5	15.4	25.9	11.3	24.7	36.0
Roundcube 0.4.1	12.6	14.6	27.2	19.7	28.9	48.6	19.7	36.2	55.9
Roundcube 0.7.0	12.8	14.4	27.2	19.7	21.2	40.9	19.7	28.0	47.7
Roundcube 0.7.0 (Firefox 3.6.8)	12.8	14.5	27.3	19.5	31.3	50.8	19.5	36.1	55.6
MediaWiki 1.16.0	8.14	10.7	18.8	10.8	15.0	25.8	10.8	24.3	35.1
WordPress 3.1.2	9.18	10.6	19.8	10.3	15.0	25.3	10.3	22.8	33.1

5.1 Accuracy of detection results

We evaluate Clickjuggler using four real-world web applications: Joomla [14], Roundcube [17], MediaWiki [16], and WordPress [15]. All of these applications are widely used. For example, Joomla is downloaded over 35 million times, Roundcube is downloaded over 2 million times, WordPress is used in over 70 million users. To check for the clickjacking vulnerabilities, we use two versions of Firefox (20.0.1 and 3.6.8) because the former (20.0.1) supports X-Frame-Options HTTP response header but the letter (3.6.8) does not. We select normal links in the four web applications as target buttons to check for the cursorjacking vulnerabilities. This is because the template for the cursorjacking targets normal links and buttons.

Table 2 lists the summary of those experiments. The names and versions of the tested web applications are listed in (Web Application). (target button) lists target links, buttons, or forms in the experiments. (Vulnerability) shows if the target links, buttons, or forms are vulnerable or not. We manually investigate each link, button, or form to find out the visual clickjacking vulnerabilities. We have confirmed the results of our manual investigation by searching for the clickjacking vulnerabilities of tested web applications in vulnerability repositories such as [21] and [22], wiki [23], and release note [24]. (Result) shows the test results obtained from Clickjuggler.

From Table 2, we can say that Clickjuggler detects visual clickjacking vulnerabilities without any false positives and any false negatives. For all the vulnerable elements (vul. in the Vulnerability column), at least one of the Clickjuggler templates judges they are vulnerable. For all the non-vulnerable elements (no vul. in the Vulnerability column), all the Clickjuggler templates judge they are not vulnerable. (basic), (1) to (6), and (cursor) in (Template) of Table 2 show the Clickjuggler templates' results.

Clickjuggler can check the defenses against visual clickjacking attacks introduced in Section 3. Clickjuggler concludes that X-Frame-Options and the element-customized defense in Roundcube 0.7.0 are not vulnerable. As you can see from Table 2, Roundcube 0.7.0 is not vulnerable in Firefox 20.0.1 (supporting X-Frame-Options) because Roundcube 0.7.0 implements

X-Frame-Options correctly. In addition to X-Frame-Options, Roundcube 0.7.0 implements another element-customized defense so that Firefox 3.6.8 (not supporting X-Frame-Options) can not be compromised as shown Table 2. If a page is embedded in another page, a code of the element-customized defense examines the origin of the page in which the page is embedded. If the origin of the page differs, this code disables all the form elements.

As shown in Table 2, Clickjuggler concludes that a page to edit user's profile in Joomla 3.0.2 is not vulnerable to basic clickjacking and evasion techniques of frame busting. This is because this page exceptionally implements the frame busting code and its implementation is perfect. The frame busting code is in the head tag of the page. Even if JavaScript is disabled, it does not display the page. Subtle variables such as `parent.location` and `document.referrer` are not used in the code.

5.2 Performance

We compare the detection time using CJTool [12] and BeEF plug-in [13]. These tools help the developers to craft the attacker's pages of visual clickjacking. In those experiments Clickjuggler and these tools detect basic clickjacking vulnerability in normal links of four web applications used in Section 5. This is because CJTool targets only basic clickjacking vulnerability and BeEF plug-in does not target forms which require users to input data. We repeatedly measure the time to test the normal links with each tool 5 times and calculate the average time.

Table 3 lists the summary of those experiments. The names and versions of the targeted web applications are listed in (Web Application). (total) shows the total time of the (preparatory) time and the (test) time. (preparatory) shows the time to prepare information to check for vulnerabilities and perform operations for each tool. (test) shows the time to craft the attacker's pages and test the normal buttons.

Table 3 shows that Clickjuggler detects the vulnerabilities in a shorter time than the existing tools in all cases. The test time is the cause of the difference between the total time of Clickjuggler and the existing tools. This is because Clickjuggler automates the processes to craft the attacker's pages and to perform visual clickjacking attacks. CJTool and BeEF plug-in do not automate those processes as shown in Section 6.

5.3 Limitation

Clickjuggler causes false positives when the defense uses CSS because even if a mouse event is issued to a button for which a CSS effect is written, this event does not trigger the CSS effect. Developers discuss that the CSS effect is not triggered with JavaScript [25]. When Clickjuggler issues a mouse event to a button for which a defense is written in CSS, this defense does not work. Clickjuggler determines that this button is vulnerable because all attacks are successful. We believe that Clickjuggler can detect the defense written in CSS by modifying the browsers. This is because JavaScript does not trigger the CSS effect due to the browsers. However, we do not modify Firefox because as far as we know, there is not the defense written in CSS.

Clickjuggler can not craft attacker's pages which make use of XSS filters in IE and Chrome. To make use of XSS filters, Clickjuggler must embed part of the frame busting code in the URI of the protected page. The current version of Clickjuggler cannot identify the frame busting code in the protected page. This is because each developer can write different frame busting code. Clickjuggler can identify the frame busting code by a slight extension. For example, Clickjuggler requires the developers to provide the frame busting code. However, this extension is not implemented because Clickjuggler is implemented as a Firefox plug-in.

6. Related work

Some countermeasures against visual clickjacking have been proposed. Frame busting [6] and X-Frame-Options [7] have been already discussed in Section 3.1. In this section, we address other defenses against visual clickjacking.

Several existing tools help developers to check for a clickjacking vulnerability in web applications. CJTool [12] and a plug-in [13] for BeEF (the Browser Exploitation Framework [26]) help to craft an attacker's page for basic clickjacking. The vulnerabilities which these tools focus on and the detection time of these tools are discussed in Section 1 and Section 5.2, respectively. We address the architecture of these tools in this section.

It is difficult that these tools craft always exhaustively the attacker's pages. These tools craft the attacker's page according to the developers' setting. This is because these tools do not use templates to craft the attacker's page. Clickjuggler crafts exhaustively the attacker's pages because it uses templates.

These tools can not perform three processes because they are not implemented in the browsers. First, they can not obtain information about the target buttons from the browsers. Second, they can not issue the event via the browsers to perform visual clickjacking. Third, they can not determine wether the attack is successful or not because they can not obtain information from the browsers in the attack. Clickjuggler automates those processes.

InContext [5], CSP [27], and [28] offer the defenses against visual clickjacking in which the client and the server cooperate. The server indicates behavior of each element on a page in InContext and behavior of each content and page in CSP and [28] on the client's browser. Clickjuggler is useful to check for whether or not the behavior indicated by the server are complete and correct.

ProClick [29] is a client-side defense to detect basic clickjacking attacks in a proxy-level framework. To identify symptoms of basic clickjacking, ProClick examines parameters of requests and responses according to a policy crafted by users. To set up the policy, ProClick users need to have intimate knowledge about the evasion techniques of frame busting. Clickjuggler does not require users to have the intimate knowledge.

ClearClick [30], Clicksafe [31], and CSCP [32] are browser's extensions to defend against the basic clickjacking. To detect a button to be disguised, ClearClick compares the bitmap of the clicked element with that rendered without inheriting properties from the parent element. If the bitmaps differ, ClearClick concludes the button suffers from basic clickjacking and alerts users. The users choose between sending the request or not. The choice is not easy for the users which do not have knowledge about clickjacking. Clickjuggler users do not need the knowledge.

To safely choose between sending the request or not, Clicksafe provides the user with other users' ratings about the choice. For example, the user obtains a percentage of the users that chose to send the request. The ratings are created from feedback on the action which users of Clicksafe choose. Clicksafe does not ensure that the ratings are correct because malicious users can give incorrect feedback on the choice. Detection results of Clickjuggler is not affected by the malicious users.

CSCP employs an existing method to detect hidden Facebook widgets and warns users. CSCP does not detect basic clickjacking on web applications other than Facebook. In contrast, Clickjuggler can test these web applications.

[33] is a technique for automatically searching for basic clickjacking attacks in the wild. This technique automatically clicks on buttons on a target page, analyzes the page, and determines whether or not the page is an attacker's page. This technique can not confirm the correctness of the defenses.

7. Conclusion

This paper has presented Clickjuggler, an automated tool for checking defenses against visual clickjacking during the development phase. Web developers must carefully write the code for defending against visual clickjacking, but it is not easy to implement the defending code correctly. Clickjuggler releases the developers from the burden of checking the correctness of the implementation. Our experimental results demonstrate that Clickjuggler can detect the visual clickjacking vulnerabilities in 4 real-word web applications and in a shorter time than the existing tools.

References

[1] Hansen, R. and Grossman, J.: Clickjacking, http://www.sectheory.com/clickjacking.htm (2008).
[2] Sophos: Viral clickjacking 'Like' worm hits Facebook users, http://nakedsecurity.sophos.com/2010/05/31/viral-clickjacking-like-worm-hits-facebook-users/.
[3] US-CERT: CVE-2008-4503: Adobe Flash Player Clickjacking Vulnerability, http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4503 (2008).
[4] Yang, D.: Clickjacking: An Overlooked Web Security Hole, https://community.qualys.com/blogs/securitylabs/2012/11/29/clickjacking-an-overlooked-web-security-hole.
[5] Huang, L.-S., Moshchuk, A., Wang, H. J., Schechter, S. and Jackson, C.: Clickjacking: attacks and defenses, *Proc. of USENIX Security Symp.*, pp. 22–22 (2012).
[6] OWASP: Clickjacking Defense Cheat Sheet, https://www.owasp.org/index.php/Clickjacking_Defense_Cheat_Sheet.
[7] Microsoft: IE8 Clickjacking Defense, http://blogs.msdn.com/b/ie/archive/2009/01/27/ie8-security-part-vii-clickjacking-defenses.aspx.
[8] Rydstedt, G., Bursztein, E., Boneh, D. and Jackson, C.: Busting frame busting: a study of clickjacking vulnerabilities at popular sites, *in IEEE Oakland Web 2.0 Security and Privacy* (2010).
[9] OWASP: Clickjacking Defense Cheat Sheet, https://www.owasp.org/index.php/Clickjacking_Defense_Cheat_Sheet.
[10] Lekies, S., Heiderich, M., Appelt, D., Holz, T. and Johns, M.: On the Fragility and Limitations of Current Browser-provided Clickjacking Protection Schemes, *Proc. of USENIX Conf, on Offensive Technologies*, pp. 6–6 (2012).
[11] Brackebusch, T.: Typo in header makes header useless, http://joomlacode.org/gf/project/joomla/tracker/?action=TrackerItemEdit&tracker_item_id=30790.
[12] Stone, P.: Clickjacking Tool, http://www.contextis.com/research/tools/clickjacking-tool/.
[13] Lundeen, B. and Alves-Foss, J.: Practical clickjacking with BeEF, *IEEE Conf. on Technologies for Homeland Security*, pp. 614–619 (2012).
[14] Joomla: Joomla, http://www.joomla.org/.
[15] WordPress: WordPress, http://wordpress.org/.
[16] MediaWiki: MediaWiki, http://www.mediawiki.org/wiki/MediaWiki.
[17] Roundcube: Roundcube, http://roundcube.net/.
[18] Ruby on Rails: Ruby on Rails Security Guide, http://guides.rubyonrails.org/security.html.
[19] django: Clickjacking Protection, https://docs.djangoproject.com/en/1.6/ref/clickjacking/.
[20] Mozilla: Mozilla Developer Network "Web API interfaces", https://developer.mozilla.org/en-US/docs/Web/API.
[21] National Institute of Standards and Technology: National Vulnerability Database, http://web.nvd.nist.gov/.
[22] SECLISTS: SECLISTS.ORG, http://seclists.org/.
[23] Roundcube: Roundcube(wiki), http://trac.roundcube.net/wiki/Changelog.
[24] Joomla: Joomla 3.0.2 Released, http://www.joomla.org/announcements/release-news/5471-joomla-3-0-2-released.html.
[25] stackoverflow: Trigger css hover with JS, http://stackoverflow.com/questions/4347116/.
[26] BeEF: BeEF: The Browser Exploitation Framework Project, http://beefproject.com/.
[27] Stamm, S., Sterne, B. and Markham, G.: Reining in the web with content security policy, *Proc. of Int'l Conf. on World Wide Web*, pp. 921–930 (2010).
[28] Nepomnyashy, M.: Protecting applications against Clickjacking with F5 LTM, *SANS Institute InfoSec Reading Room* (2013).
[29] Shahriar, H., Devendran, V. K. and Haddad, H.: ProClick: A Framework for Testing Clickjacking Attacks in Web Applications, *Proc. of Int'l Conf. on Security of Information and Networks*, pp. 144–151 (2013).
[30] Maone, G.: Hello ClearClick, Goodbye Clickjacking!, *In Black Hat Europe* (2012).
[31] Shamsi, J. A., Hameed, S., Rahman, W., Zuberi, F., Altaf, K. and Amjad, A.: Clicksafe: Providing Security against Clickjacking Attacks, *Proc. of Int'l Symp. on High-Assurance Systems Engineering*, pp. 206–210 (2014).
[32] Rehman, U., Khan, W., Saqib, N. and Kaleem, M.: On Detection and Prevention of Clickjacking Attack for OSNs, *Proc. of Int'l Conf. on Frontiers of Information Technology*, pp. 160–165 (2013).
[33] Balduzzi, M., Egele, M., Kirda, E., Balzarotti, D. and Kruegel, C.: A solution for the automated detection of clickjacking attacks, *Proc. of ACM Symp. on Information, Computer and Communications Security*, pp. 135–144 (2010).