We are still vulnerable to clickjacking attacks: about 99% of Korean websites are dangerous

Daehyun Kim and Hyoungshick Kim

Department of Computer Science and Engineering,
Sungkyunkwan University, Republic of Korea
{dan.0901, hyoung}@skku.edu

Abstract. *Clickjacking* is an attack that tricks victims into clicking on invisible elements of a web page to perform an unintended action that is advantageous for an attacker. To defend against clickjacking, many techniques have already been proposed, but it is still unclear whether they are effectively deployed in practice. We study how vulnerable Korean websites are to clickjacking attacks by performing real attacks on top 100 popular Korean websites as well as all the financial websites. Our results are quite significant: almost all Korean websites (about 99.2%) that we looked at are vulnerable to clickjacking attacks. Extending our observation to mobile websites, we can also obtain similar results.

1 Introduction

A web framing attack, called *clickjacking* [4], uses a transparent `iframe`[1] to hijack users' clicks. In a typical clickjacking attack scenario, a malicious web page is constructed by an attacker so that the attacker tricks victims into clicking on elements of the web page within an invisible iframe to perform an unintended action that is advantageous for the attacker.

Recently, clickjacking attacks have been considerable interest and many prevention techniques (e.g., *frame busting* [12]) have been proposed [6]. However, it is still questionable whether the defence mechanisms are indeed effectively deployed in practice. Our work is originally motivated by this question.

In this paper, we present an empirical study on analyzing the feasibility of clickjacking attacks by intensively testing the 100 most popular and all 36 financial institution websites, respectively, in Korea (the total of 130 unique Korean websites). Our experimental results show that 129 out of 130 websites (about 99.2%) are vulnerable to clickjacking attacks. The `Citi Bank` website (`http://www.citibank.co.kr/`) is the only website that was not vulnerable to clickjacking attacks. We also extend our analysis to mobile web pages and then obtain the almost same results: all of the 100 most popular websites, when browsed through mobile phones, were also vulnerable to clickjacking attacks. Our key contributions can be summarized as follows:

[1] `iframe` is the HTML tag to specify an inline frame which is used to embed another document within the current HTML document.

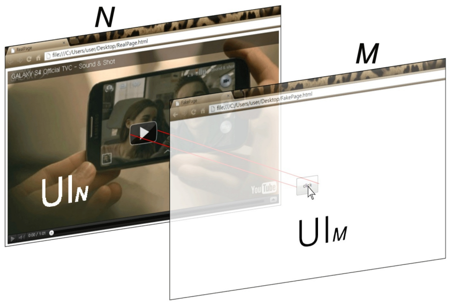

Fig. 1. An example of clickjacking attacks. An invisible web page (M) with a button (UI$_M$) to run a malicious action is put on top of what appears to be a normal web page (N) with a video clip (UI$_N$). A victim can see the play button on the video clip, but cannot see the button in a transparent web page. When the victim click on the play button, the victim actually click on the button in the transparent web page to trigger a malicious action such as downloading malware.

– As far as we are aware, this is the first empirical study of clickjacking attacks for the regional websites in a country. Our results show that the most popular Korean websites (96 out of 96 websites – 100%) are much more vulnerable to clickjacking attacks compared with global websites (62 out of 100 websites – 62%). We also extend our analysis to mobile web pages and then obtained the same results (read Section 3.1 and 3.2).
– Second, we examine the feasibility of clickjacking attacks for all the financial institution web pages in Korea and demonstrate that they are also vulnerable to clickjacking attacks (35 out of 36 websites – about 97.2%). Five websites have used defence codes against clickjacking, but three websites among them can be easily bypassed by disabling JavaScript with a simple HTML tag; one website can be vulnerable to an attacker who controls a domain with the victim's server domain as a substring (read Section 3.3).
– Third, we discuss the possible reasons why Korean websites are still vulnerable to clickjacking attacks and suggest reasonable solutions to fix the problems (read Section 4).

In the rest of this paper, we will present the above results in detail. First, we will explain how clickjacking attacks work in practice to provide a better understanding of our study and then demonstrate this empirically on Korean websites.

2 What is clickjacking?

Clickjacking is a web-based attack that was reported by Jeremiah Grossman and Robert Hansen in 2008 [4]. It is a technique to attract users to click an element of a web page which is designed by an attacker [12]. The attacker uses the HTML tag called `iframe` to specify an inline frame which is used to embed an HTML document inside another HTML document. Fig. 1 illustrates an example

```
<html>
<title>An example page</title>
<body>

<style>
iframe { filter:alpha(opacity=0); opacity:0; }
</style>

<iframe src="./FakePage.html" border="0" scrolling="no">
</iframe>

</body>
</html>
```

Fig. 2. A HTML document with a transparent inline frame. An element in the HTML document can be hidden by making the inline frame's CSS opacity value zero. In addition, an attack can specify the option for scrolling to remove the scroll bar from the inline frame. This document shows that clickjacking attacks can be easily implemented.

of clickjacking attacks. Fig. 2 also shows how to create such a transparent inline frame. We can see that an invisible `iframe` can be simply made by setting the Cascading Style Sheet (CSS) property to control the opacity of HTML elements.

To generalize clickjacking attacks, we use N to represent a normal web page (e.g., web-portal, banking, social networking services) with an innocuous UI element UI_N. An attacker creates a malicious web page M with an element UI_M to perform a malicious action (e.g., downloading malware, sending an email message to the attacker, liking the attacker's website) by transparently overlaying UI_M on top of UI_N. Hence, when a victim tries to click on UI_N within N, the victim indeed clicks on UI_M within M to trigger an unintended action instead of UI_N. Clickjacking attacks can also be more sophisticated than simply hiding the target element as follows:

– Using the CSS cursor property, attackers can hide the default cursor and programmatically draw a fake cursor elsewhere [7], or alternatively set a custom mouse cursor icon to a deceptive image that has a cursor icon shifted several pixels off the original position [3].
– Attackers can use JavaScript – a single click can be changed into a double click which can click on UI_N as well as UI_M at the same time. Unless the intended action is performed by clicking on UI_N, careful users suspect that the visiting web page might be strange or deceptive.
– Clickjacking can be used with another attack such as Cross-Site Request Forgery (CSRF) which is a widely exploited website vulnerability whereby unauthorized commands are transmitted from a user that the website trusts [10] (see, for example, `http://seclab.skku.edu/csrf_daum.avi`).

Note that clickjacking attacks don't make use of bugs (e.g. a failure to properly sanitize the user input) in web browsers or applications unlike other common

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

Fig. 3. A simple frame busting code. This code is typically used to for preventing framing by another website.

web vulnerabilities such as SQL injection and cross site scripting. They are consequences of a misuse of some HTML/CSS features (e.g., the ability to manipulate opacity of inline frames), combined with the way in which the web browser allows users to interact with invisible, or barely visible, elements [1] and thus anyone who knows basic HTML/CSS can easily implement clickjacking attacks.

3 Feasibility of clickjacking attacks for Korean websites

We evaluated the feasibility of clickjacking attacks for the 100 most popular websites in Korea and their corresponding mobile websites. We also examined all Korean financial institutions since these are obviously high-risk targets.

In our adversary model, the goal of an adversary is to embed a target website within the adversary's website so that the embedded website can be used to lure victims to trigger an action (e.g., downloading malware) that is advantageous for an attacker. We here assume that the target website is not compromised.

3.1 Results for the most popular websites in Korea

We used the top 100 websites (including web portals, online shopping, search engines, social networks, banks, online media, and games listed on Rankey (`http://www.rankey.com/`) which provides information about websites ranking to show the feasibility of clickjacking attacks for Korean websites.

To test whether clickjacking attacks can be successfully achieved, we used the five different web browsers (Internet Explorer, Chrome, Firefox, Safari, and Opera) which run on a Windows PC since the attack and defence implementations might perform differently on some web browsers. The reason why we chose these browsers is that they are the most popular in Korea[2]; we also used Internet Explorer 8 between different Internet Explorer versions for the same reason.

To defend against clickjacking attacks, some websites have used frame busting codes which intend to prevent the web pages from being loaded within an inline frame. A simple example code is shown in Fig. 3. A common frame busting code is made up of a conditional statement (e.g., `top.location != self.location`) and a counter action (e.g., `top.location = self.location`) [12].

We found that seven websites have only used frame busting codes which rely on JavaScript to detect framing and prevent it, but can easily be bypassed by disabling JavaScript HTML attributes; if JavaScript is disabled in the context of `iframe`, their frame busting codes might not be working properly against

[2] `http://gs.statcounter.com/#browser-KR-monthly-201204-201304`

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

Fig. 4. An example code to bypass the frame busting code used for Korean websites: KB bank (`http://www.kbstar.com`). We can also bypass the codes for Nonghyup (`http://www.nonghyup.com`) and Hana bank (`http://www.hanabank.com`) in the same manner.

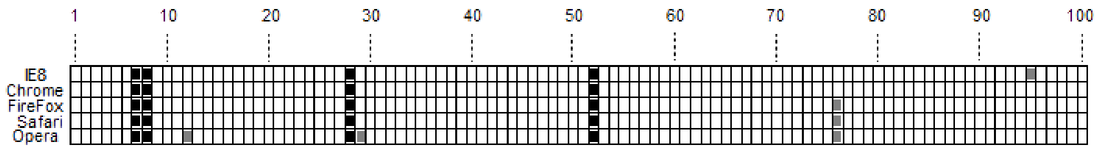

Fig. 5. Experimental results for the 100 most popular websites with five web browsers (Internet Explorer 8, Chrome, Firefox, Safari, and Opera – which are sorted in descending order of their traffics in Korea) on a Windows PC. The numbers represent the ranks of websites based on their traffics. Black boxes indicate that the websites are secure against our clickjacking attacks; gray boxes indicate that the websites do not support the tested web browser.

clickjacking attacks. Therefore we tried to restrict use of the JavaScript with the attribute `security`=“`restricted`” for Internet Explorer 8 and `sandbox` for Chrome, Firefox, and Safari. Fig. 4 shows how to bypass the frame busting codes used for the three Korean websites (KB bank, Nonghyup, and Hana bank). Their frame busting codes can be easily framed by using simple HTML tags.

For each combination of a website S in the list and a web browser B, we tested whether the website S can be successfully framed by our website which can be assumed as the attacker's malicious website M and then still normally work. The visualization in Fig. 5 provides an overall view of the clickjacking distributions for the tested web browsers. In this figure, black boxes indicate that the websites are secure against our clickjacking attacks; gray boxes indicate that the websites do not support the tested web browser.

Surprisingly, in Fig. 5, we can see that all websites are vulnerable to our clickjacking attacks except for the four ones (Google, Facebook, YouTube, and Twitter). Interestingly, all these are not Korean websites. This is because global websites such as Google and Facebook might have already experience such attacks [13] more intensively than the Korean websites. To see the difference between Korean websites and global websites, we also evaluated the top 100 global websites selected from the Alexa Top-500 Global Sites (`http://www.alexa.com/topsites`) in the same manner. 62 out of the tested global websites can be effectively framed by our clickjacking attacks; this proportion (62%) is significantly less than the 100% attack success with the most popular Korean websites. To verify this statistically, we performed the one-tailed, two-proportion Z test with 95% confidence and then obtained the results of $z = 6.7271$ and $p < 0.05$. This shows that Korean websites are significantly weaker against clickjacking attacks compared with global websites.

In the analysis for the top 100 global websites, we found that 18 out of the 38 websites that cannot be framed by clickjacking are in the US (the total of 53 websites); 4 out of the 38 websites are in Russia (the total of 5 websites). This shows that clickjacking attacks may appear to be relatively popular in some countries such as US and Russia rather than the other countries. Also, we surmise that companies with higher rewards for finding security bugs are more secure against clickjacking; 25 out of the 38 websites (about 66%) are owned by the companies (Google, Facebook, and PayPal) which run a bug bounty program for cash (see the list of representative bug bounty programs in `http://bugcrowd.com/list-of-bug-bounty-programs/`).

All web browsers show almost the same results except for a few websites that do not support Internet Explorer, Firefox, Safari, and Opera, respectively. For example, `Hyundai card` (`http://www.hyundaicard.com/`) with the 95th rank requires a plug-in but Internet Explorer 8 has failed to install the required plug-in unlike the other web browsers. Since the Opera web browser doesn't support the `sandbox` attribute, the approach to disable JavaScript is not effective in the 12th and 29th websites with the Opera web browser.

3.2 Results for mobile web pages

We extend our empirical analysis to also investigate the feasibility of clickjacking attacks on mobile web pages. We visited the 100 most popular Korean websites again with the two mobile devices: Samsung Galaxy Note 2 which runs Android Jelly Bean 4.2 and Apple iPad which runs iOS 6.1.3. We tested each website with the ten different web browsers (Android: Android default, Chrome, Opera, Dolfin, Firefox, and iOS: Safari, Chrome, Dolfin, Opera, Mecury), respectively, which are selected based on their popularity (see `http://statcounter.com/`).

When we visited the websites via the mobile devices, 92 websites served mobile alternatives instead of their original web pages. However, the results for mobile devices are not changed from those which are designed for desktop screens. We cannot observe any obvious difference between browsers and/or platforms as the same as for the desktop system.

The six mobile web pages are only designed to make users to download an app offering services rather than to explore the websites themselves. However, this practice might lead to security risks for mobile users; clickjacking attacks will be a serious concern for such web pages since clickjackers can effectively try to lure users to download malware instead of legitimate apps.

3.3 Results for Korean banking websites

We also examined the feasibility of clickjacking attacks for 36 financial institution websites since these are obviously high-risk targets. We classified the financial institutions as monetary and non-monetary financial institutions to see whether monetary financial institutions have much concerned about security compared with non-monetary financial institutions.

```
<style>
html{display:none;}
<style>

<script>
try{
    if(top.location.host == self.location.host)
    {
        document.documentElement.style.display='block';
    }else{
        top.location = self.location;
    }
}catch(e){ top.location = self.location; }
</script>
```

Fig. 6. The frame busting code used for `Citi Bank`. This code is almost similar to Rysdstedt's recommendation [12].

For the 18 *monetary* financial institutions, 17 banking websites can be successfully framed by using clickjacking. Although the four banking websites have used codes to detect and disable clickjacking, the three codes among them can be easily bypassed with the HTML attributes of `sandbox` or `security`="`restricted`". There exists only one monetary financial institution `Citi Bank` (`http://www.citibank.co.kr/`) using a sophisticated defence code (see Fig. 6) against clickjacking attack, which is similar to Rysdstedt's recommendation [12]: When a web page is first loaded, the style tag hides all contents on the web page (`html{display:none;}`). If JavaScript is disabled, the web page will remain blank. Similarly, if the web page is framed, it will either remain blank or it will attempt to frame bust. The script only reveals the document's contents if the web page is not running in a frame.

For the 18 *non-monetary* financial institution websites, all the websites can be attacked by using clickjacking. Since the `SinHyup Bank` website (`http://www.cu.co.kr/`) that only uses a defence code to block clickjacking attacks, all the remaining websites can be easily framed with a simple HTML tag.

The defence code (see Fig. 7) used in the `SinHyup Bank` website seems secure against clickjacking at first glance in that this code uses the property of `document.domain` which returns the domain name of the server that loaded the current document. However, this web page can be framed by an attacker who controls a domain with the substring "`openbank.cu.co.kr`" (e.g., `kopenbank.cu.co.kr`). It shows the limitation in checking the domain name of a server to prevent clickjacking attacks.

Although one *monetary* financial institution website `Citi Bank` is secure against clickjacking, it is still unclear that *monetary* financial institution websites are overall more secure against clickjacking than *non-monetary* financial institution websites. To verify this statistically, we performed the one-tailed, two-proportion Z test with 95% confidence and then obtained the results of

```
<script>
if(document.domain.indexOf("openbank.cu.co.kr") > -1) {
      location.href = "https://" + document.domain;
  }else{
      alert("Access Deny");
  }
}
</script>
```

Fig. 7. The frame busting code used for `SinHyup Bank`. This code checks whether the domain name of the server includes the substring "`openbank.cu.co.kr`".

$z = 1.0142$ and $p = 0.1563$. From this test result, we conclude that they are not significantly different at $p < 0.05$.

4 Discussion

Clickjacking attacks and defences have been intensively studied [12,6] since it was already reported almost five years ago [4]. Interestingly, however, almost Korean websites (about 99.2%) are still vulnerable to these attacks.

Probably, clickjacking is unpopular in Korea. We can see that the small number (6 out of 130 – about 4.6%) of websites have only tried to prevent clickjacking attacks. This is because victims from clickjacking attacks have been highly concentrated until now. For example, there have currently been two kinds of widespread clickjacking attacks in the wild: Likejaking [13] and Tweetbomb [9]. Fortunately, Korean websites don't seem attractive targets yet for clickjackers.

A few of Korean websites have used defence codes to detect framing and prevent it, but they are naively implemented and thus impractical. Unlike the global websites (e.g., Google), the Korean websites don't use code obfuscation techniques for their defence codes to make the codes themselves difficult to analyze. To make matters worse, almost frame busting codes used in the Korean websites can be easily bypassed with a simple HTML tag. Perhaps this is another example of "security theatre", which tackles the feeling but not the reality – vendors (or engineers) may demonstrate this perfunctory function to assure their customers (or managers) that their websites are really secure against clickjacking attacks. We have to understand what defences are practically effective. At the technical level, we recommend that Korean websites have to implement a secure code like the frame busting script used in `Citi Bank` (see Fig. 6) and also use obfuscation techniques to protect their codes. Some manual labor will be required to de-obfuscate clickjacking defences when they are properly obfuscated.

Surely, it is not easy to trace new vulnerabilities in dynamically updated websites over time. The existence of a bounty program for security bugs seems helpful to solve this problem. For example, when a clickjacking vulnerability in Google Docs (`http://docs.google.com`) was found [8], it was fixed soon within two weeks. In general, companies with higher rewards for finding bugs will become more secure and sustainable.

5 Related work

The possibility of attacks using transparent frames was first mentioned in a Mozilla bug report [11]. Hansen and Grossman [4] coined the term “clickjacking” in 2008. Clickjacking attacks can be used alone or in combination with other attacks such as Cross-Site Request Forgery (CSRF) [2] which is a technique that allows the attacker to trick a user into performing an action, using her authority and credentials. Targeted attacks were made on Facebook [13] and Twitter [9]. Rydstedt et al. [12] showed that the most frame-busting practices implemented by the Alexa top 500 sites can be circumvented by a simple way such as JavaScript disabling.

There are several clickjacking defence techniques to provide the visual integrity of a web page. One straightforward mitigation is to present a confirmation prompt to users when the target element has been clicked. Unfortunately, this approach gives a poor user experience and is still vulnerable to double-click timing attacks [6]. Another popular approach is to use the UI layout randomization technique [5]; however, this is not the ultimate solution to this problem since the attacker may try to ask the victim to keep clicking until successfully guessing the location of a target UI element [6]. Huang et al. [6] proposed a new defence called “InContext”, where websites mark UI elements that are sensitive, and browsers enforce context integrity of user actions on these sensitive UI elements. This technique is very effective against clickjacking attacks but requires the client browsers (or OSes) modification. Rydstedt et al. [12] presented a simple but effective frame-busting code to mitigate clickjacking attacks.

Many client-side defence solutions were also proposed: (1) The removal of all transparency from all cross-origin elements is surely effective to detect clickjacing attacks; however, it can incur a significant visual penalty on benign websites [6]. (2) Heuristic-based techniques (e.g., ClickIDS [1]) decide when to allow rendering transparent frames by detecting whether the clicked cross-origin frame is not fully visible. But, these solutions have not been widely deployed since heuristics will inherently incur false positives and false negatives, and there is always a room for attackers to design their websites to bypass the heuristics that are being checked. They also introduced some compatibility costs for legacy websites, which may hinder browser vendor adoption.

6 Conclusion

We analyzed the feasibility of clickjacking attacks by testing the 100 most popular and all 36 financial institution websites in Korea (the total of 130 unique websites). Our experiments show that 129 out of 130 websites (about 99.2%) are vulnerable to clickjacking attacks; even Korean websites with clickjacking defences could be easily defeated with a simple HTML tag. A statistical test (the one-tailed, two-proportion Z test with 95% confidence) shows that Korean websites are significantly more vulnerable compared with global websites (62 out of 100 websites – 62%).

When we consider that clickjacking defence techniques are clearly understood and fairly easy to implement, the reason why Korean websites are still vulnerable to clickjacking seems rather clear. This is because Korean websites might be unattractive targets for clickjackers until now; the small number (6 out of 130 – about 4.6%) of websites have only tried to prevent clickjacking attacks but 5 of them failed in practice. However, we need to prepare for forthcoming attacks. So we recommend a secure implementation of frame busting codes and the use of code obfuscation techniques to make the codes themselves difficult to analyze at the technical level. Also, the introduction of a bounty program will be helpful to fix security bugs such as clickjacking quickly; we can see that about 66% of the secure global websites against clickjacking are owned by the companies (Google, Facebook, and PayPal) which run a bug bounty program.

References

1. Balduzzi, M., Egele, M., Kirda, E., Balzarotti, D., Kruegel, C.: A solution for the automated detection of clickjacking attacks. In: Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security (2010)
2. Barth, A., Jackson, C., Mitchell, J.C.: Robust defenses for cross-site request forgery. In: Proceedings of the 15th ACM conference on Computer and communications security (2008)
3. Bordi, E.: Proof of concept - cursorjacking (noscript), `http://static.vulnerability.fr/noscript-cursorjacking.html`
4. Hansen, R.: Clickjacking, `http://ha.ckers.org/blog/20080915/clickjacking/`
5. Hill, B.: Adaptive user interface randomization as an anti-clickjacking strategy (2012), `http://www.thesecuritypractice.com/the_security_practice/papers/AdaptiveUserInterfaceRandomization.pdf`
6. Huang, L.S., Moshchuk, A., Wang, H.J., Schechter, S., Jackson, C.: Clickjacking: attacks and defenses. In: Proceedings of the 21st USENIX conference on Security symposium (2012)
7. Kotowicz, K.: Cursorjacking again, `http://blog.kotowicz.net/2012/01/cursorjacking-again.html`
8. Kumar, M.: Hacking google users with google's goopass phishing attack (2013), `http://thehackernews.com/2013/03/hacking-google-users-with-googles.html`
9. Mahemoff, M.: Explaining the "don't click" clickjacking tweetbom (2009), `http://softwareas.com/explaining-the-dont-click-clickjacking-tweetbomb`
10. Ristic, I.: Apache Security. O'Reilly Media (2005)
11. Ruderman, J.: Bug 154957 - iframe content background defaults to transparent (2002), `https://bugzilla.mozilla.org/show_bug.cgi?id=154957`
12. Rydstedt, G., Bursztein, E., Boneh, D., Jackson, C.: Busting frame busting: a study of clickjacking vulnerabilities at popular sites. In: IEEE Oakland Web 2.0 Security and Privacy (W2SP 2010) (2010)
13. SophosLabs: Facebook worm - "likejacking" (2010), `http://nakedsecurity.sophos.com/2010/05/31/facebook-likejacking-worm/`