A Course Module on Clickjacking

Lindsay Simpkins, Xiaohong Yuan, Junghee Kim
Computer Science Department
North Carolina A&T State University
1601 E Market St
Greensboro, NC 27411
USA
lsimpkin@aggies.ncat.edu, xhyan | jungkim @ncat.edu

ABSTRACT: **:** *Clickjacking is a form of UI-Redress where a victim thinks they are browsing the webpage they see, but click actions are actually on a hidden webpage. Typically, the victim must already be authenticated on the hidden page for the attack to work. There are several available methods to detect or prevent clickjacking attacks on both the server and client side. The main prevention methods are for the server side, and verify that the website is not being loaded inside an iFrame. If an attacker attempts to load a website with one of these methods in place, it will either "break" out of the frame, i.e. refresh the page directly to its URL, or not load the page in the first place. Currently it is important to increase the implementation rate of these prevention methods. This paper introduces a clickjacking course module which includes a tutorial of clickjacking, a hands-on lab, and a quiz. There is a discussion of the teaching experience with this course module. The module can be integrated into web security or network security courses introducing the topic of clickjacking.*

Keywords: Course module, Collaborative learning, Clickjacking, Cyber security, Web security

Received: 18 August 2014, Revised 21 September 2014, Accepted 28 September 2014

1.Introduction

A clickjacking attack does not use software vulnerabilities in a web application, but takes advantage of the HTML iFrame property and often the opacity property (Callegati, Ramilli 2009; Niemietz 2011). This makes clickjacking a fairly static attack. It can also be used to launch other attacks, such as Cross Site Request Forgery or Cross-Site Scripting (Braun, Heiderich 2013; Niemietz 2011; Stone 2010).

In 2008 Robert Hansen and Jeremiah Grossman demonstrated the first clickjacking Proof of Concept (POC), using the Adobe Flash Player Settings Manager to give an attacker access to a user's camera and microphone (Hansen, Grossman 2008). Since then, clickjacking has been used in several widespread attacks involving social media.

As a popular attack approach, it is important to introduce the concept of clickjacking to Computer Science or IT students in universities or colleges. There is an existing 2-hour student lab on clickjacking by SEED: Developing Instructional Laboratories for Computer Security Education created in 2011. This lab runs on an Ubuntu Virtual Machine available for use with the lab (Du 2014). Instructors may need to provide information to students on how to use Ubuntu, and how to develop an HTML

webpage. The SEED Project recently produced a new video with several demonstrations (Du 2010), including clickjacking and likejacking attacks, and how the prevention methods stop attacks.

The SEED lab on clickjacking provides students with a minimal introduction to the topic of clickjacking, and code for a basic webpage with a hidden iFrame. Students then have to create a clickjacking attack given a trusted website, but the attacker page must be built from scratch. Once the webpage is created, the lab instructs students to implement a prevention method using JavaScript FrameBusting code. The SEED lab may be difficult for students who lack experience with HTML and JavaScript. It seems best suited for a supervised lab environment, where an instructor or lab teaching assistant (TA) can provide guidance.

There are also two online training courses on clickjacking. The 15-minute course on Hacker Academy (Conway 2014) requires a paid subscription to their website. This course includes a hands-on lab that uses a virtual machine, and at least covers the implementation of a clickjacking attack. There is also a 3-question quiz. OpenSesame (2014) has a 15-minute series of short videos, and requires a paid subscription to their website or charges for the course itself. Their course includes a demonstration of clickjacking, a basic explanation of how clickjacking works, and covers the JavaScript and HTTP header prevention methods. There are 3 multiple-choice questions spread throughout the videos. Due to the requirement of paid subscriptions, these resources may not be accessible by college students.

Based on the SEED lab, we developed a course module which includes a tutorial of clickjacking and a hands-on lab. The students are required to complete reading materials before the lab. The reading includes a description of clickjacking, an implementation example with code, and examples of actual attacks, use on social networks, prevention methods, and some basic HTML. The lab asks students to first identify clickjacking attacks in three given websites, then create a clickjacking attack where both the attacker page and the trusted page were provided, and finally implement a JavaScript prevention method.

This lab is suited for either a supervised lab environment or an online class, and can be used by students with little programming experience. It can be easily downloaded and performed individually without the help from a lab TA or instructor.

The rest of the paper is organized as follows: Section 2 describes how and why clickjacking works, provides examples and data from actual attacks, and lists several prevention methods. Section 3 describes the hands-on lab. Section 4 discusses our teaching experience and lab improvement. This paper concludes with Section 5.

2. Clickjacking Tutorial

This course module includes a clickjacking tutorial, which explains how clickjacking works, and why it works.

2.1 How Clickjacking Works

The attack website loads the target website in an iFrame. To ensure correct alignment with the target website, the attacker can use a series of iFrames with absolute positioning (Callegati & Ramilli 2009; Niemietz 2011; Stone 2010) or JavaScript to follow mouse movement (Gall 2010; Niemietz 2011).

The following is an example of clickjacking, using a series of iFrames with absolute positioning. The code is based on an example in UI Redressing: Attacks and Countermeasures Revisited (Niemietz 2011).
Table. 1 lists the source code for inner.html. It puts the target page inside an iFrame, shown in Figure 1.

```
1  <iframe src = "[target website]" width = "1000" height = "500"
scrolling = "no" frameborder = "none">
2  < / iframe>
```

Table 1. inner.html

Table. 2 lists the source code for clickjacking.html. It puts *inner.html* into an iFrame, which means the target page is now inside two levels of iFrames. The final result is that *clickjacking.html* only shows the Join Now button instead of the entire page, shown in Figure 2.

```
1 <iframe id = "inner" src = "inner.html" width = "200" height = "350"
scrolling = "no" frameborder = "none" style = "position: absolute; left: -
20px; top: - 305px;">
2 </iframe>
```

Table. 2 clickjacking.html

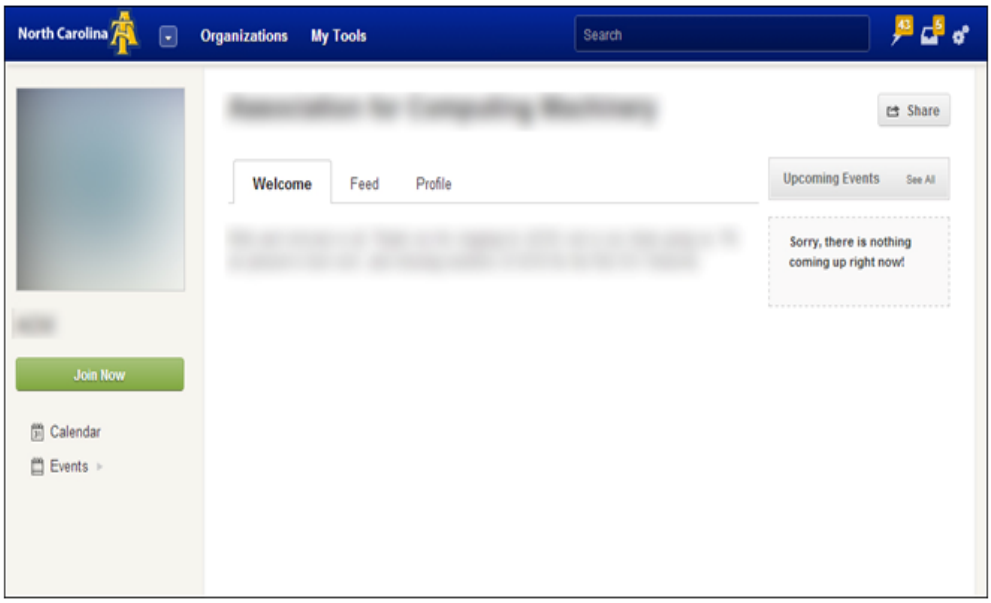

Figure. 1 inner.html screenshot

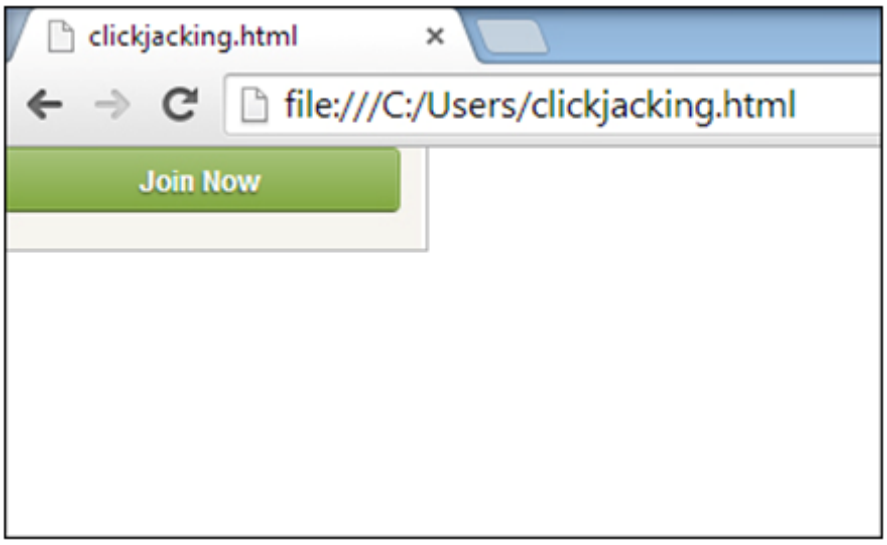

Figure 2. clickjacking.html screenshot

Table. 3 lists the source code for *t*rustedpage.html. It has an iFrame of clickjacking.html, which means the target page is now inside three levels of iFrames. The final result is that trustedpage.html has a button in the exact location as the Join Now button. However, the opacity of the iFrame was set to 0.0, which means the Join Now button on the target page is invisible, like in Figures 3. Figure 4 shows how the iFrame overlays the GO button. If a victim clicks on the GO button, they will actually click the Join Now button.

```
1 < ! DOCTYPE HTML PUBLIC "- // W3C//DTD HTML 4.01 // EN"
2 "http://www.w3.org/TR/html4/strict.dtd">
3 < html >
4 < head >
5 < title >Trusted Web Page < /t itle >
6 < / head >
7 < body >
8 < h1 >www.ncat.edu < / h1 >
9 < form action = "http://www.ncat.edu">
10 < input type = "submit" value = "GO" style = "width:100px;
height:30px; " >
11 < / form>
12
13 < iframe id = "clickjacking" src = "clickjacking.html" width
= "97" height= "25" scrolling = "no" frameborder ="none"
style = "position: absolute; left: 9px; top: 68px; opacity:0.0;">
14 </ iframe >
15
16 < / body >
17 < / html >
```

Table. 3 trustedpage.html

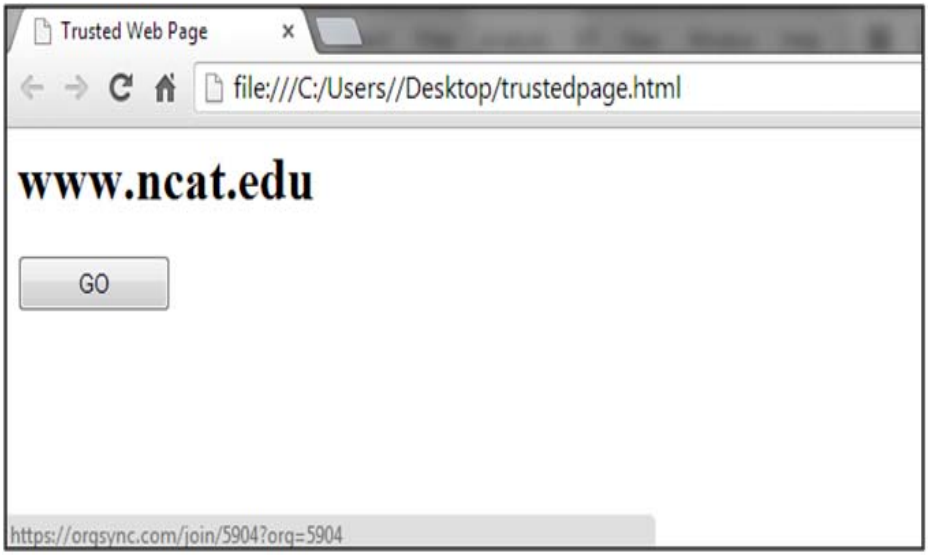

Figure. 3 trustedpage.html screenshot

2.2 Why Clickjacking Works

Most clickjacking only works if the victim is already logged into a website –such as a social networking site. This enables the attacker to trick the victim into performing actions on a trusted site, and the victim is already authenticated. Since the nature of social networking sites can be used to spread information or links quickly, as seen with viral videos, clickjacking often involves a social network in order to spread the attack.

Social media websites often provide buttons that can be easily integrated into any webpage, for example the Facebook 'Like' or 'Share' buttons or the Twitter 'Tweet' and 'Follow' buttons. A specific clickjacking attack that tricks the victim into clicking a Facebook 'Like' button was widespread enough that it was termed likejacking (SophosLabs 2010).

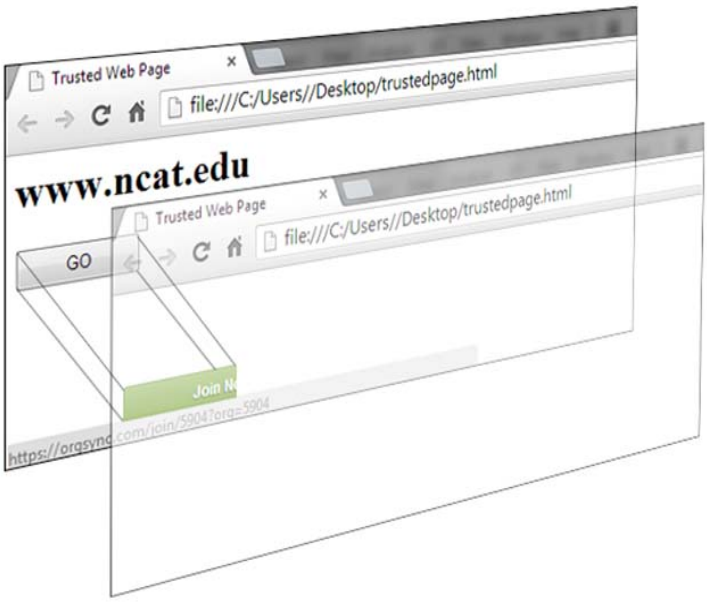

Figure. 4 trustedpage.html overlay

2.3 Actual Clickjacking Attacks
A number of real life clickjacking attacks have occurred in recent years. A few of the well-known ones are described below.

2.3.1 Facebook: Valentine Theme (Talampas 2012)
In Jan 2012, there was a Clickjacking attack on Facebook where users saw a post about installing a Valentines Theme on their Facebook account. If a victim followed the link, they were brought to a page where they could supposedly install the Facebook Valentine theme.

If the victim then clicked on the Install button, and they were using the Chrome browser, it prompted a download for file named FacebookChrome.crx and installed a browser extension named Facebook Improvement. It also automatically liked several Facebook pages as well as automatically posted a message on the affected users wall. The downloaded file was actually malware that displayed ads.

2.3.2 Twitter: Don't Click (Constantin 2009; Mahemoff 2009)
One of the first widespread clickjacking attacks was the 'Don't Click' worm on Twitter. Originally the code was developed as a POC, but someone actually put it into practice. The attack itself was not malicious, but demonstrated how quickly a clickjacking attack can spread across social media networks (Johnson 2009). Victims would see a link to a shortened URL in their feeds, such as Figure 5, and if the link was followed it took the victim to a page with a button that said 'Don't Click' such as in Figure 6. Clicking on the button simply caused the victim to post the shortened URL to their own feed, so their friends will now see the link.

2.3.3 Chrome Vulnerability (Brook 2013)
There was a vulnerability found in January 2013 in the Chrome browser that could allow attackers to obtain information from victims such as Google, Amazon, Microsoft Live, and Yahoo! Profiles pages. The attack involved "*...a two-step drag and drop method that relies on users being tricked into letting Chrome publish their data publicly*."

2.4 Attack Trends
In August 2011 Symantec released a study on 3.5 million Facebook video posts, and "*..found that up to* 15 *percent of unique posts were identified as likejacking attacks*." (Protalinski 2011). Bitdefender reported a case study around the same time on the 'See who viewed your profile' scam, which revealed approximately 1.4 million clicks were generated each wave of the attack. It spiked at 34 hours, after the link had time to spread (Rowinski 2011). In 2012 Facebook announced a joint lawsuit

with Washington State against Adscend Media, LLC. The ad network was accused of using clickjacking to spread spam. "*At one point Adscend spam lined the defendants pockets with up to $1.2 million a month.*" (Facebook 2012).

Figure 5. Example Twitter post

Figure 6. Example Don't Click website

2.5 Detection and Prevention Methods

Overall, this online attack evolves slowly, so it is fairly simple to prevent a website from being part of a clickjacking attack. There are prevention methods for both the client side and the server side, and a detection method for the client side.

2.5.1 Detection and Prevention on the Client Side

The recommended method for detection and prevention on the client side is to use the Firefox extension NoScript (Maone 2014). It helps prevent JavaScript-based attacks by blocking JavaScript unless the user allows it to run on a given website, and includes a module called ClearClick (Maone 2013). ClearClick detects clicks on hidden elements overlapping visible elements and alerts the user, though a study in 2010 found it had many false positives (Balduzzi, Egele, Kirda, Balzarotti & Krugel 2010).

An extreme method to prevent clickjacking would be to completely disable JavaScript and iFrames in the browser, and disable all plugins (Lemos 2008).

Since clickjacking often relies on the victim being logged into a website, such as a social media network, it is recommended to always sign out when leaving a website (Lemos 2008).

2.5.2 Prevention on the Server Side

2.5.2.1 FrameBusting with JavaScript

On the server side, websites can include code to keep the website from being vulnerable to clickjacking, by preventing it from being frame-able. The most widely used method is to include some JavaScript to check if the website has been put into an iFrame, and if so "*break*" out of the frame. This is called FrameBusting.

This method has flaws and can often be circumvented, such as disabling JavaScript in the browser or iFrame (Microsoft 2014; Rydstedt, Burztein, Boneh & Jackson 2010). Downsides to this method include difficulty allowing your own website to frame itself (Rydstedt et al. 2010).

OWASP recommends the JavaScript code in Table 2 for FrameBusting. When the page first loads, everything is invisible. If JavaScript is enabled and the page is not inside an iFrame, it becomes visible. If JavaScript is enabled and the page is inside an iFrame, it "*breaks*" out of the frame by reloading the page directly to itself, and then allows content to be visible. This is not considered a strong prevention method, but is the best prevention method for legacy browsers (Kantor 2011; OWASP 2014; Rydstedt et al. 2010).

Add the following code inside the HEAD element:

```
<style id = "antiClickjack" >
body {  display:none !important;  }
</style>

< script type = "text / javascript">
if (self  = = = top) {
var antiClickjack  = document.getElementById
("antiClickjack");
 antiClickjack.parentNode.removeChild(antiClickjack);
 } else {
 top.location = self.location;
} </script >
```

Table 2. JavaScript FrameBusting code (OWASP 2014)

2.5.2.2 Require JavaScript to be Enabled

By requiring JavaScript for important action buttons on a website, such as form submit buttons, it helps prevent circumvention of JavaScript FrameBusting code (Lemos 2008; Rydstedt et al. 2010).

2.5.2.3 Out-of-Band Validation

Validating sensitive actions using an out-of-band form of communication, such as email or SMS, ensures the action will only be executed with the users knowledge and consent (Lemos 2008).

2.5.2.4 HTTP Header

The best method to prevent a website from being frame-able is to include the X-FRAME-OPTIONS HTTP header. When set to DENY, content will not load if the webpage is inside an iFrame. Originally debuted 2009 in Internet Explorer 8 (Lawrence 2009), it became a standard protocol header in October 2013 in RFC 7034 (Ross, Gonodrom & Stanley 2013).

This method is more difficult to circumvent, though the header can be removed with a proxy tool (OWASP 2014). There are options to let a website easily frame itself, or allow one domain to load the page inside an iFrame (OWASP 2014; Ross et al. 2013) (Protalinski 2011; Ross et al. 2013). This is the recommended prevention method (Braun & Heidrich 2013; Constantin 2013; Kantor 2011; Niemietz 2011).

3. Hands-on Lab

Students were given a document to read prior to the lab, which included a tutorial of clickjacking, and some basic HTML tags

and style properties, such as opacity. This lab used the Ubuntu virtual machine available for the SEED Clickjacking Attack Lab (Du 2014). It includes three parts and a quiz after the lab is conducted.

3.1 Part 1

Students were given three example websites that have a clickjacking attack, representing a contest website (Figure 7), a livestream (Figure 8), and the Twitter "*Don't Click*" attack (Figure 6).

The websites were clicked in order to see the result of attack. Figures 9, 10, and 11 show the results of clickjacking on the three websites respectively. Figure 9 shows an alert box representing the victims' e-mail being deleted. Figure 10 shows an alert box representing the victim purchasing a laptop on an e-commerce website. Figure 11 represents the Twitter Don't Click attack, where the user would post a link to their feed.

Students should view the source code to identify where the clickjacking attack occurs.

3.2 Part 2

Students were asked to create a clickjacking attack. They were given both the "*legitimate*" website (Figure 12) and the "*attacker*" website (Figure 13), and had to add, align, and hide an iFrame on the "*attacker*" website. When the victim user clicks on the link for "*PLAY A FREE ONLINE DEMO HERE*" on the "*attacker*" website, the "*Confirm*" button on the "*legitimate*" website should be clicked.

3.3 Part 3

Students were asked to implement server-side clickjacking prevention, using the JavaScript FrameBusting method. The JavaScript code provided was the same code used for the SEED lab.

3.4 Quiz

After students completed the lab assignments, they were given an individual quiz. The questions asked students to describe how an attack could be identified, why clickjacking works, and why the prevention methods work.

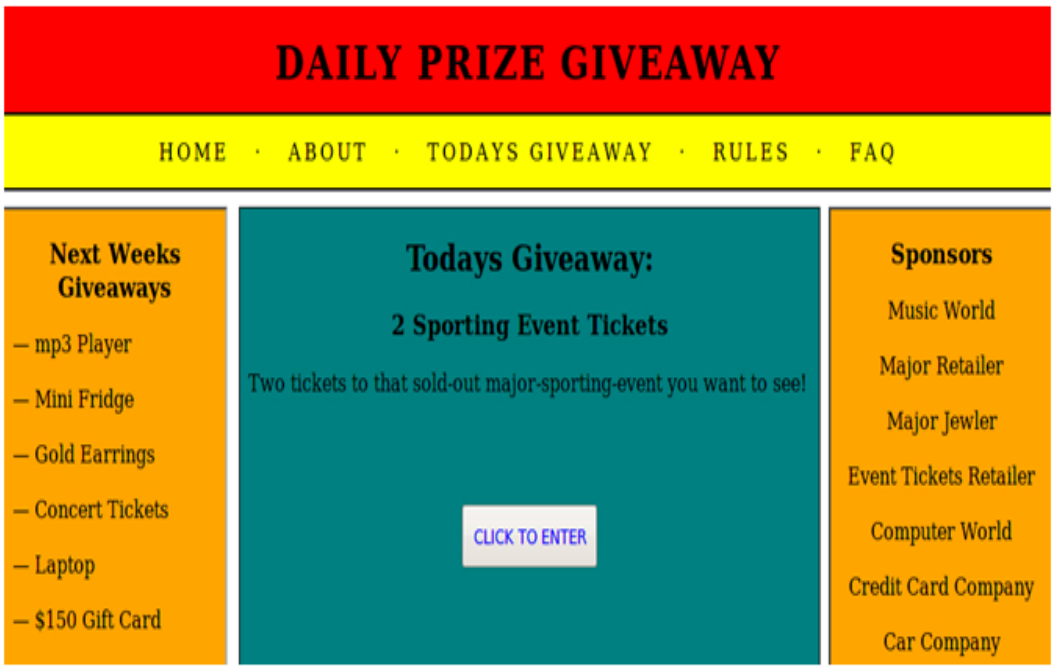

Figure 7. Example Contest website

4. Teaching Experience

This section discusses our experience teaching this course module and the initial assessment result.

4.1 Collaborative Learning Based Lab Session

While this lab can be performed by students individually, it was taught in COMP725 Software Security Testing in Fall 2013 using collaborative learning. Students were assigned to groups of 2-3, and used an online chat program to communicate. Each group posted their answers to part 1, and code from parts 2 and 3 into the chat as they completed each section of the lab. They also had to confirm their answer with the instructor or lab TA before moving to the next section. The chat logs were then used for analyzing student behavior in collaborative learning.

After the students completed the lab, they were given an individual quiz. The final grade for the lab was based on the average quiz grade from each student in the group. This was to help ensure that during the lab, the entire group would comprehend the answer, instead of one group member finding the answer and simply moving on.

4.2 Problems Encountered

Students ran into several issues with lab instructions, and there were technical issues with the chat program. In part 1 of the lab, the instructions were not clear that to discover the attack, they should look for the hidden "*legitimate*" website in the source code.

In part 2, students did not understand they were being asked to perform the attacker role when implementing the clickjacking attack. It was also not clear that the "*legitimate*" website was provided in addition to the "*attacker*" website.

In part 3, some students were confused on the concept and had thought the "*legitimate*" website was actually the "*attacker*" website, so they didn't understand why the prevention method was implemented on the provided "*legitimate*" website. There were misunderstandings about the instructions for implementing the prevention method, and several issues with the prevention method not working after the new code was saved and the webpage was refreshed. After several attempts to refresh the page, including clearing the browser cache, it was found that reloading the browser fixed the issue.

At one point during the lab, the server hosting the chat program had to be restarted.

4.3 Lab Improvement Based on Teaching Experience

The instructions were modified to address issues found when the lab was taught. For part 1, it was clarified that students should look for the "*legitimate*" website the attacker was trying to get victims to click. Part 2 specified that students were playing the role of the attacker, that the "*legitimate*" website already existed, and where it could be found. Some tips were provided to help students correctly align the iFrame of the "*legitimate*" website over the "*attacker*" website. Part 3 specified that students were playing the role of the website administrator on the side of the "*legitimate*" website. The instructions for implementing the prevention method were clarified, and it recommended reloading the browser, if refreshing the webpage did not work.

Figure 8. Example Livestream website

4.4 Course Module Assessment

Right after the lab session, the students were given an individual quiz on what they learned. 17 students attend the collaborative learning based lab session. 5 students did a make-up lab individually, with the updated instructions. Figure 14 shows a box-and-whisker plot of students individual quiz scores for both the in-class group and the make-up group. The quiz was out of 15 points.

From the responses, it was clear that most of the students could identify an attack from the source code, and describe clickjacking and how to prevent it. The students expressed that the collaborative learning based lab session and course modules were very interesting.

Figure 9. Example Contest website: Attack Result

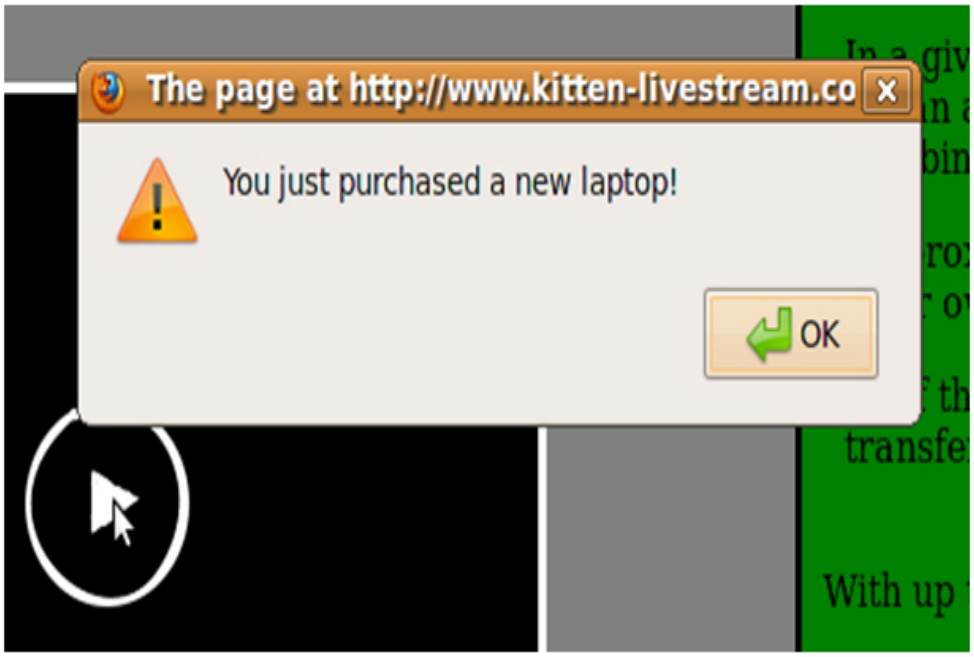

Figure 10. Example Livestream website: Attack Result

Figure. 11 Example Don't Click website: Attack Result

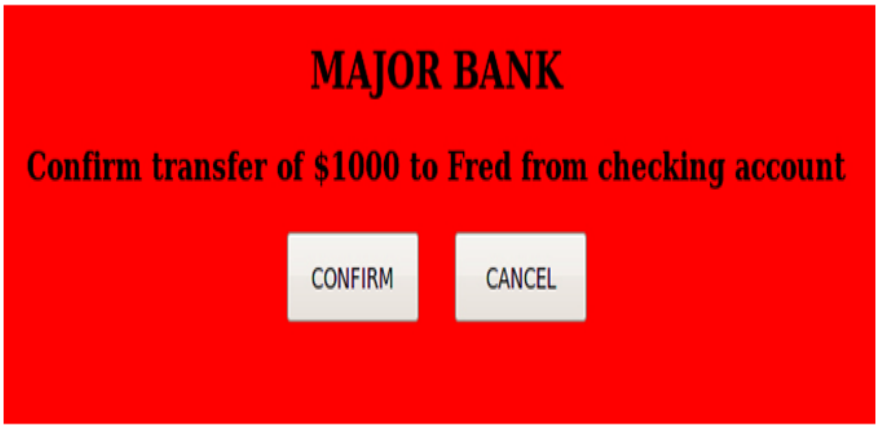

Figure 12. Provided "*Legitimate*" website

Figure 13. Provided "*Attacker*" website

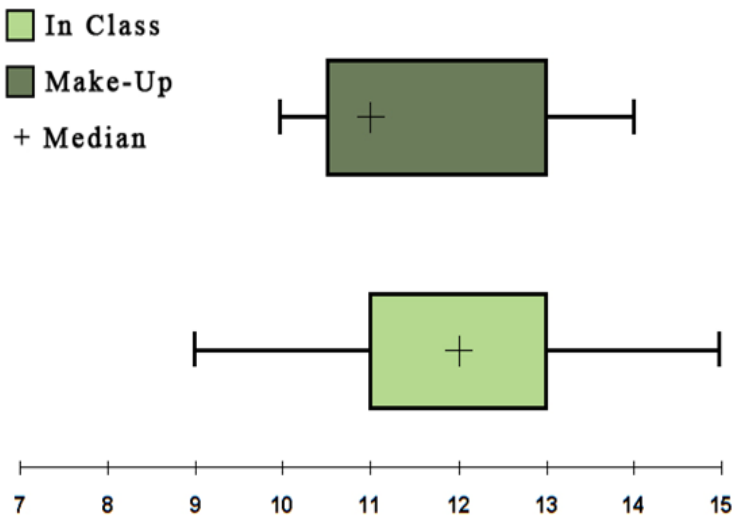

Figure 14. Quiz Scores

5. Conclusion

In the past few years, clickjacking attacks have run rampant on social media networks. The method of attack is fairly static, and is easy to prevent on the server side. To teach students about clickjacking and how to prevent it, we created a course module on clickjacking which includes a hands-on lab that can be done individually or in a supervised lab environment. Before the lab, students had required reading introducing them to the topic. The lab teaches students to identify, implement, and prevent a clickjacking attack. Students should not need prior programming experience in order to perform the lab. Course module assessment shows most students mastered the material taught and enjoyed the collaborative learning based hands-on experience. Our future work includes analyzing the online chat logs to gain insight on the characteristics of collaborative learning, and to continue teaching and assessing this course module.

6. Acknowledgements

This work is partially supported by NSF under the grant DUE-1318695. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of NSF.

Citations and References

[1] Balduzzi, M., Egele, M., Kirda, E., Balzarotti, D., Krugel, C. (2010). A Solution for the Automated Detection of Clickjacking Attacks. *In:* Proc. of the 5th ACM Symposium on Information, Computer and Communications Security (ASIACCS '10). New York, NY, USA: ACM, 135-144. DOI: http://doi.acm.org/10.1145/1755688.1755706.

[2] Braun, F., Heiderich, M (2013). X-Frame-Options: All about Clickjacking?. White Paper. Cure53. Retrieved June 24, 2014, from: https://cure53.de/xfo-clickjacking.pdf.

[3] Brook, C. (2013). Chrome Clickjacking Vulnerability Could Expose User Information on Google, Amazon. Threat Post. Retrieved June 24, 2014, from: http://threatpost.com/chrome-clickjacking-vulnerability-could-expose-user-information-google-amazon-010213/77358.

[4] Callegati, F., Ramilli, M. (2009). Frightened by Links. *IEEE Security & Privacy,* 7(6) 147-152. DOI: http://doi.ieeecomputersociety.org/10.1109/MSP.2009.177.

[5] Coderrr. (2009). Preventing Frame Busting and Click Jacking (UI Redressing). coderrr. Retrieved June 24, 2014, from: http://coderrr.wordpress.com/2009/02/13/preventing-frame-busting-and-click-jacking-ui-redressing.

[6] Constantin, L. (2009). Clickjacking Attack Launched on Twitter. Softpedia. Retrieved June 24, 2014, from: http://news.softpedia.com/news/Clickjacking-Attack-Launched-on-Twitter-104456.shtml.

[7] Constantin, L. (2013). Mozilla advises webmasters to implement X-Frame-Options security header. PCWorld. Retrieved June 24, 2014, from: http://www.pcworld.com/article/2079921/mozilla-advises-webmasters-to-implement-xframeoptions-security-header.html.

[8] Conway, J. (2014). Click Jacking. The Hacker Academy & MAD Security. Retrieved June 24, 2014, from: http://hackeracademy.com/module/clickjacking.

[9] Du, W. (2010). ClickJacking Lab. Syracuse University. Retrieved June 24, 2014, from: http://www.cis.syr.edu/~wedu/seed/Labs/Vulnerability/ClickJacking.

[10] Du, W. (2014). Vulnerability and Attack Labs. Syracuse University. Retrieved June 24, 2014, from: http://www.cis.syr.edu/~wedu/seed/all_labs.html.

[11] Facebook (2012). Facebook, Washington State AG Target Clickjackers. Facebook. Retrieved June 24, 2014, from: https://www.facebook.com/notes/facebook-security/facebook-washington-state-ag-target-clickjackers/10150494427000766.

[12] Gall, M. (2010). Facebook/Flattr/… Clickjacking Examples And How To Avoid It. digitalbreed. Retrieved June 24, 2014, from: http://digitalbreed.com/2010/facebook-flattr-clickjacking-and-how-to-avoid-it.

[13] Hansen, R., Grossman, J. (2008). Clickjacking. SecTheory. Retrieved June 24, 2014, from: http://www.sectheory.com/clickjacking.htm.

[14] Johnson, C. (2009). What is this Don't Click business?. Sunlight Foundation. Retrieved June 24, 2014, from: http://sunlightfoundation.com/blog/2009/02/12/what-dont-click-business.

[15] Kantor, I. (2011). The Clickjacking attack, X-Frame-Options. JavaScript Tutorial. Retrieved June 24, 2014, from: http://javascript.info/tutorial/clickjacking.

[16] Lawrence, E. (2009). IE8 Security Part VII: ClickJacking Defenses. Microsoft. Retrieved June 24, 2014, from: http://blogs.msdn.com/b/ie/archive/2009/01/27/ie8-security-part-vii-clickjacking-defenses.aspx.

[17] Lemos, R . (2008). You don't know (click)jack. SecurityFocus. Retrieved June 24, 2014, from: http://www.securityfocus.com/news/11535/2.

[18] Mahemoff, M. (2009). Explaining the, Don't Click, Clickjacking Tweetbomb. Software As She's Developed. Retrieved June 24, 2014, from: http://softwareas.com/explaining-the-dont-click-clickjacking-tweetbomb.

[19] Maone, G. (2013). ClearClick: Effective Client-Side Protection Against UI Redressing Attacks. W3C. Retrieved June 24, 2014, from: http://lists.w3.org/Archives/Public/public-webappsec/2012May/att-0030/ClearClick_WAS2012_rv2.pdf.

[20] Maone, G. (2014). Noscript FAQ. InformAction. Retrieved June 24, 2014, from: http://noscript.net/faq.

[21] Microsoft (2014). Security attribute. Microsoft. Retrieved June 24, 2014, from: http://msdn.microsoft.com/en-us/library/%20ms534622%28VS.85%29.aspx.

[22] Niemietz, M. (2011). UI Redressing: Attacks and Countermeasures Revisited. Seminar Work. Ruhr-University of Bochum, Bochum, Germany. http://ui-redressing.mniemietz.de/uiRedressing.pdf.

[23] OpenSesame (2014). Clickjacking Training Course Video. OpenSesame. Retrieved June 24, 2014, from: https://www.opensesame.com/c/clickjacking-training-course.

[24] OWASP (2014). Clickjacking Defense Cheat Sheet. OWASP. Retrieved June 24, 2014, from: https://www.owasp.org/index.php/Clickjacking_Defense_Cheat_Sheet.

[25] Protalinski, E. (2011). Symantec finds 15% of Facebook videos are likejacking attacks. ZDNet. Retrieved June 24, 2014, from: http://www.zdnet.com/blog/facebook/symantec-finds-15-of-facebook-videos-are-likejacking-attacks/3316.

[26] Ross, D., Gondrom, T., Stanley T. (2013). HTTP Header Field X-Frame-Options. RFC Editor. Retrieved June 24, 2014, from: https://www.rfc-editor.org/rfc/rfc7034.txt.

[27] Rowinski, D. (2011). Report: Nearly 15% of Videos on Facebook Are Likejacking Attempts [Updated]. ReadWrite. Retrieved June 24, 2014, from: http://readwrite.com/2011/09/06/report_nearly_15_of_videos_on_facebook_are_lifejac.

[28] Rydstedt, G., Bursztein, E., Boneh, D., Jackson, C. (2010). Busting Frame Busting: a Study of Clickjacking Vulnerabilities on Popular Sites. *In*: Proc. of the IEEE Oakland Web 2.0 Security and Privacy (W2SP 2010). http://w2spconf.com/2010/papers/p27.pdf

[29] SophosLabs. (2010). Facebook Worm - , Likejacking. Sophos. Retrieved June 24, 2014, from: http://nakedsecurity.sophos.com/2010/05/31/facebook-likejacking-worm.

[30] Stone, P. (2010). Next Generation Clickjacking: New attacks against framed web pages. *In*: Proc. of the Black Hat Europe. https://media.blackhat.com/bh-eu-10/presentations/Stone/BlackHat-EU-2010-Stone-Next-Generation-Clickjacking-slides.pdf.

[31] Talampas, C. (2012). Report: Facebook Valentine's Theme Leads to Malware. TrendMicro. Retrieved June 24, 2014, from: http://blog.trendmicro.com/trendlabs-security-intelligence/facebook-valentines-theme-leads-to-malware.