CLICKJACKING ATTACK SCANNING AND DEFENCING

IT20023614 WIJESINGHA W.M.P.M
Department of Computer Systems Engineering
Sri Lanka Institute of Information Technology,
New Kandy Rd,
Malabe 10115, Sri Lanka.
it20023614@my.sliit.lk

***Abstract* - New forms of web-based extortion are on the rise, such as clickjacking. Current clickjacking countermeasures are shown to be ineffective in this study by creating new attack variations based on existing tactics. A user's private webcam, email, or other private data may be compromised as a result of our clickjacking assaults, as well as online browsing anonymity. Several clickjacking protections for web browsers have been developed and implemented, but all have flaws. The most extensively used protections now depend on frame-busting, which prevents a vulnerable page from being framed. This research project mainly focuses on identifying clickjacking attacks and countermeasures for those kinds of attacks. This solution uses Python for the backend. The methodology behind the solution is first, to scan the vulnerability using the python program. and after that, patch the vulnerability, As an example attack, attempts to display the target site in an iframe and then install another iframe on top of it. Inspired by OWASP's proof-of-concept HTML boilerplate.**

Keywords— iframe, threat, threat agent,

I. INTRODUCTION

Clickjacking is a new kind of online assault that has gained popularity in recent years. Current clickjacking countermeasures are shown to be ineffective in this study by creating new attack variations based on existing tactics. A user's private webcam, email, or other private data may be compromised through clickjacking, as well as the user's ability to remain anonymous while online.

Multiple apps and websites that share a graphical display are vulnerable to "clickjacking" assaults, which deceive users into interacting with UI components of a different principal (e.g., by clicking, touching, or using voice input) in ways they did not intend. If, for example, an attacker web page overlays on top of a benign user interface element like as a "claim your free iPad" button, the user is tricked into clicking on the "Like" button. Therefore, Facebook friends see a tale about the user "liking" the attacker's website when she "claims" an iPad. Web browsers will be used as a case study for the sake of clarity. All client operating systems that share a display and use mutually distrusting principles may, however, benefit from the strategies outlined here. [1]

The term "frame busting" refers to a method of preventing a web page from loading in a sub-frame. To protect against clickjacking, frame-busting is the preferred method, and it is also essential for image-based authentication, such as Yahoo's Sign-in Seal. The Yahoo! login page is verified by the Sign-in Seal, which shows a picture picked by the user. Even if the top page isn't the genuine Yahoo login page, the login page might be accessed in a sub-frame without frame breaking. For example, new developments in the use of drag-and-drop to extract and inject data into frames further show the need for safe frame busting. [2]

Figure 1: Visualization of a clickjacking attack

A clickjacking assault is seen in Figure 1: the target site is framed in a transparent iframe that is placed on top of what looks to be a regular page. Users unintentionally engage with the victim site when they interact with the usual website. Web sites typically utilize the following basic frame busting code to fight against clickjacking attacks.

An attacker provides a user with a sensitive UI element of a target app while the user is unaware of the application's context, which causes the user to do an action they would not normally perform. Likejacking, for example, is a kind of online assault in which an attacker hides the sensitive "Like" button on top of the button that says, "claim your free iPad."
Web sites are able to designate their sensitive UI components and have browsers enforce the context integrity of user activities on these elements with our protection, dubbed In Context. The visual and temporal contexts of a user's behavior form the basis of the user's context. [1] [3]

- *Visual context:* What a user sees just before performing a sensitive action is referred to as the "visual context." Users require complete visibility of all pointer feedback, including the sensitive UI element (such as cursors, touch, or NUI input feedback), in order to preserve visual context integrity. For the first, we use the term "target display integrity," whereas for the second, we use the term "pointer integrity." [1]
- *Temporal context:* The timeliness of a user action is referred to here. Temporal integrity assures that a user's activity at a certain moment is intended by the user and not a fluke. Using a bait-and-switch attack, for example, an attack website may violate users' temporal integrity by inserting a sensitive UX element just before the user is expected to click on a "claim your free iPad" button. [1]

II. THREAT MODEL

In Context's main line of defense is against clickjacking attackers. As a web thief, a clickjacked has all the same powers. When a victim visits an [4] [5]attacker's site, the attacker's material is shown in their browser because
(1) they hold a domain name
(2) they can make them visit their site. [1]

In order to entice the user into doing unwanted UI actions, the website conceals a sensitive UI element either visually or temporarily
We don't do anything to guard against social engineering assaults, which can succeed even if the system is flawlessly conceived and developed from the beginning. An attacker may dupe users into clicking on a Facebook Like button by putting deceptive content, such as photographs from an organization dedicated to helping the underprivileged, around it. It is possible for a victim to misunderstand the Like button as expressing a preference for charity work rather than for the attacker's website, and hence intend to click on it. A clickjacking attacker, on the other hand, takes advantage of a system's inability to preserve context integrity for user activities and may therefore change the sensitive element visually or temporally to deceive customers.

III. BACKGROUND

This section compares known clickjacking attacks and responses to our own efforts. We presume that a victim user is accessing a clickjacking attacker's website, which embeds and manipulates the target element, such as Facebook's Like button or PayPal's checkout dialog, from a separate domain. [1] [4]

For the first time in 2002, the ability to load a translucent layer over a web page and have the user's input effect it was discovered. It wasn't until 2008 that this became a big concern. Adobe Flash Player was clickjacked in 2008 by Jeremiah Grossman and Robert Hansen, who determined that an attacker may get access to the computer without the user's awareness. Jeremiah Grossman and Robert Hansen invented the phrase "clickjacking," which is a mashup of the terms "click" and "hijacking." [7] There have been many similar assaults, hence the phrase "UI redressing" has been used to characterize these attacks rather than simply clickjacking itself.

Clickjacking categories

- Classic: works mostly through a web browser
- Likejacking: utilizes Facebook's social media capabilities
- Nested: clickjacking tailored to affect Google+
- Cursor jacking: manipulates the cursor's appearance and location
- Mouse Jacking: inject keyboard or mouse input via remote RF link
- Browser less: does not use a browser

- Cookie jacking: acquires cookies from browsers
- File jacking: capable of setting up the affected device as a file server
- Password manager attack: clickjacking that utilizes a vulnerability in the autofill capability of browsers

IV. CLICKJACKING VULNERABILITY IDENTIFICATION TOOL

To determine if a website is susceptible or not, I developed this tool in the Python programming language. This tool has the capability to check webpages for clickjacking attack vulnerabilities.

How this program works?

First, it will generate an iframe template (a file with the extension.html), and then it will attempt to load the specified website underneath the iframe template. When a webpage loads behind the iframe, we know that it is susceptible to clickjacking because of this. If the content is not loaded underneath the iframe, then it is not susceptible to clickjacking. This tool may examine a certain website to determine whether its content security policy and x-frame options are enabled.

File Stack

Figure 2

Clickjack.py

Python code included in this file serves as the primary application for determining whether a website is insecure. It receives data from the terminal and produces two separate outputs.

1. Terminal Output
2. Web base Output

Solution.html
In the event that a user visits a website that has a vulnerability, we are able to route them to this solution.html page. This file contains a variety of preventative and defensive strategies against clickjacking assaults.

Readme.md
This file includes how to run this program and more information about the

Program Demo

Run the program:

python3 clickjack.py <url>

```
python3 clickjack.py https://msclubsliit.org
```

Figure 3

After the run this program, it makes two file call

1. cj-target.html: this html file loaded our main output with targeted website.
2. cj-attacker.html: this html loads our iframe template. This is the web site we trying to load on top of targeted web site.

After running this program, it makes additional above two files.

Figure 4

Outputs
Here we have two web base outputs and terminal outputs.

1. Terminal
2. cj-target.html
3. cj-attacker.html

Terminal output

Figure 5

```
[+] Test Complete!
[+] Target URL :  https://msclubsliit.org
[+] https://msclubsliit.org is   [ VULNERABLE !!! ]
[+] Thanks for using Clickjacking Tester ;-)
```

Figure 6

Terminal output shows

- Test Status (Complete or Not)
- Target URL
- Vulnerable Status
- Thank You message
- Creator name

Web Base Output

cj-target.html

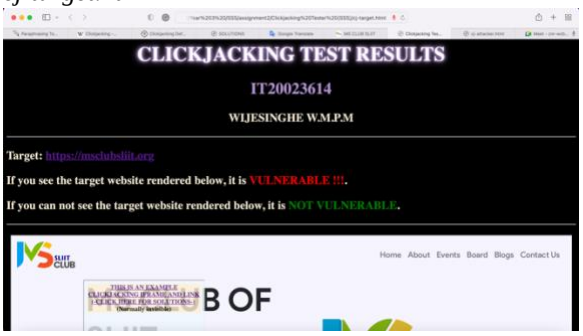
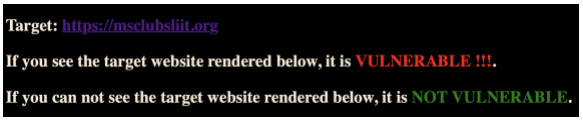


Figure 7



This file webpage includes Target URL and Vulnerable status with vulnerable status.

cj-attacker.html

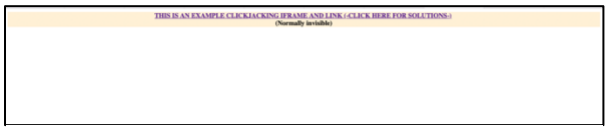


Figure 8

This is the iframe created from clickjack.py

Solution page

In main page there is a link to solution page

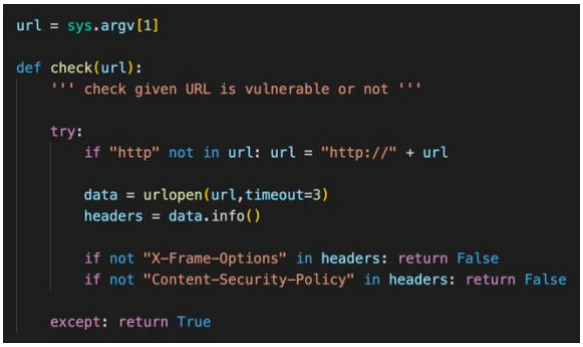
Figure 9

Figure 10

This page have how to mitigate Clickjacking attack and prevent from it.

Code base explanation:

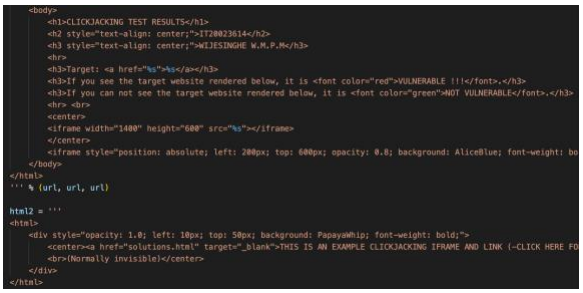
Codes are written from Python programming language and Web base outputs are create from HTML and CSS.



```
url = sys.argv[1]

def check(url):
    ''' check given URL is vulnerable or not '''

    try:
        if "http" not in url: url = "http://" + url

        data = urlopen(url,timeout=3)
        headers = data.info()

        if not "X-Frame-Options" in headers: return False
        if not "Content-Security-Policy" in headers: return False

    except: return True
```

Figure 11

Fig 11 shows how to detect weather x-Frame -Options and Content-Security-Policy enable or not if they are enabling it returns Not Vulnerable and if it is not, it returns It is Vulnerable.



```
    <body>
        <h1>CLICKJACKING TEST RESULTS</h1>
        <h2 style="text-align: center;">IT20023614</h2>
        <h3 style="text-align: center;">WIJESINGHE W.M.P.M</h3>
        <hr>
        <h3>Target: <a href="%s">%s</a></h3>
        <h3>If you see the target website rendered below, it is <font color="red">VULNERABLE !!!</font>.</h3>
        <h3>If you can not see the target website rendered below, it is <font color="green">NOT VULNERABLE</font>.</h3>
        <hr> <br>
        <center>
        <iframe width="1400" height="600" src="%s"></iframe>
        </center>
        <iframe style="position: absolute; left: 200px; top: 600px; opacity: 0.8; background: AliceBlue; font-weight: bo
    </body>
</html>
''' % (url, url, url)

html2 = '''
<html>
    <div style="opacity: 1.0; left: 10px; top: 50px; background: PapayaWhip; font-weight: bold;">
        <center><a href="solutions.html" target="_blank">THIS IS AN EXAMPLE CLICKJACKING IFRAME AND LINK (-CLICK HERE FO
        <br>(Normally invisible)</center>
    </div>
</html>
```

Figure 12

Fig 12 shows HTML which make web-based output.

This entire program takes (262 KB on disk) and easy to use. It works for most every most popular browsers in the industry as an example: Safari, Fire Fox, Chrome, Brave etc.

V. MITIGATION

To avoid becoming a victim of clickjacking, follow these three steps:

- Forbidding framing from other domains by sending the correct Content Security Policy (CSP) frame-ancestor directive response headers. Older HTTP headers, such as X-Frame-Options, are utilized for gradual degradation and older browser support. [8]

- Authentication cookies should be set with SameSite=Strict (or Lax) unless none are required explicitly (which is rare). [8]

- Use code to make sure that your current frame is always your highest-level window.

Defending with Content Security Policy (CSP) frame-ancestors directive

A browser's ability to display a website as an iframe or a frame may be indicated using the frame-ancestors directive in a Content-Security-Policy HTTP response header. By ensuring that their material is not integrated onto other websites, sites may prevent Clickjacking assaults.

VI. CONCLUSION

A cyber exploit known as clickjacking has lately gained a lot of media attention. Numerous media outlets and blogs have covered the issue. In the meanwhile, it isn't known how widespread clickjacking is in the real world, and how important it is to Internet users' security. System that can automatically identify clickjacking attempts on web pages was described in this study. In order to determine the incidence of such assaults, we tested more than one million web sites that are likely to contain malicious information and are often accessed by Internet users to verify our technology. We were able to scan up to 15,000 web pages each day by distributing the analysis over numerous virtual computers. Along with the ClearClick protection given by the NoScript plug-in, we created a new detection method called ClickIDS. All components were incorporated into a web application testing system that was au- tomated. [3]

We were able to find two examples of clickjacking attacks used for spamming and fraud on the websites we visited. The pages containing these clickjacking assaults were provided as examples on security-related websites, yet we were able to find them without any effort. Additionally, we discovered a number of "borderline at-tacks" throughout our investigation. False positives are difficult to distinguish from genuine threats in this kind of assault. In our research, we found that clickjacking isn't the most popular attack method used by cybercriminals nowadays. As a result of our research, Jeremiah Grossman predicted that clickjacking would only become a major issue in the next five to six years.

VII. ACKNOWLAGEMENT

I give my special thanks to Dr. Lakmal Rupasinghe, head of web security (IE2062) module. And I want to give my gratitude to Ms. Chethana Liyanapathirana, our lecturer of Web Security Module. Both of you gave us most valuable knowledge, information and also study resources. Ms. Eshandi Aththanayake for their immense support and guidance on this web audit. I really appreciate all your support.

VIII. REFERANCE

[1] Lin-Shung Huang, Alex Moshchuk, Helen J. Wang, Stuart Schechter, Collin Jackson, "Clickjacking: Attacks and Defenses," USENIX Security, 2012.

[2] Gustav Rydstedt, Elie Bursztein, Dan Boneh, Collin Jackson, "Busting Frame Busting: a Study of Clickjacking Vulnerabilities on Popular Sites," amazonNews, 2010.

[3] Marco Balduzzi, Manuel Egele, Engin Kirda, Davide Balzarotti, Christopher Kruegel, "A Solution for the Automated Detection of Clickjacking Attacks".

[4] G. Rydstedt, "OWASP," [Online]. Available: https://owasp.org/www-community/attacks/Clickjacking.

[5] A. Chiarelli, "Auth0," 30 October 2020. [Online]. Available: https://auth0.com/blog/preventing-clickjacking-attacks/.

[6] Jawwad A. Shamsi, Sufian Hameed, Waleed Rahman, Farooq Zuberi, Kaiser Altaf, Ammar Amjad, "CLICKSAFE: PROVIDING SECURITY AGAINST CLICKJACKING ATTACKS," High-Assurance Systems Engineering, 2014.

[7] t. f. e. Wikipedia, " Clickjacking Wikipedia, the free encyclopedia," [Online]. Available: https://en.wikipedia.org/wiki/Clickjacking.

[8] O. org, "OWASP Cheat Sheet Series," [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Clickjacking_Defense_Cheat_Sheet.html.

[9] Devdatta Akhawe, Warren He, Zhiwei Li, Reza Moazzezi, Dawn Song UC Berkeley, "Clickjacking Revisited A Perceptual View of UI Security," usenix, 2014.

[10] A. o. C. A. a. A. E. D. S. f. A. Devices.

[11] Longfei Wu, Benjamin Brandt, Xiaojiang Du, and Bo Ji, "Analysis of Clickjacking Attacks and An Effective Defense Scheme for Android Devices," Department of Computer and Information Sciences Temple University Philadelphia, Pennsylvania , 2016.