Joshua Trzcienski
Comp 435
Lab 4 Report

Clickjacking Lab:

Clickjacking is a type of UI redress attack that generally happens when an attack has created a webpage that tricks the user into clicking something that they either could not see, or did not intend to which hijacks the click. One common way to do this is to have some sort of invisible button or page over some other webpage that can be taken through an iframe. The person getting attacked sees a normal webpage, be secretly over this webpage they see, there could be some sort of button or something else that when clicked would do something malicious.

Task 1: Copy That Site!

The first task involves modifying the html of the attacker's website. Without any modifications, there is just a singular blue button. By inserting an iframe, seen in Image Two, the html of the defender's website can be overlaid onto the attacker's website. Once the size was adjusted in the CSS file, seen in Image One, the new attacker's website looks very similar to the defender's website, with the only difference being that the blue button that was originally there is still there, and it is displayed on top of the defender's website in the top right, which can be seen in Image Three.

Image One:

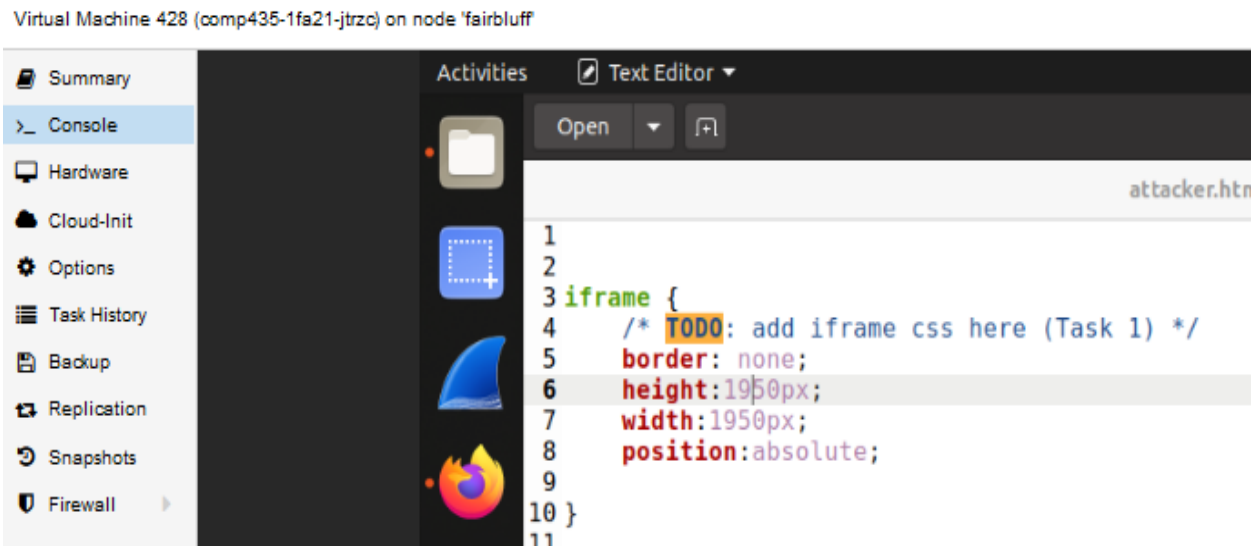


Image Two:

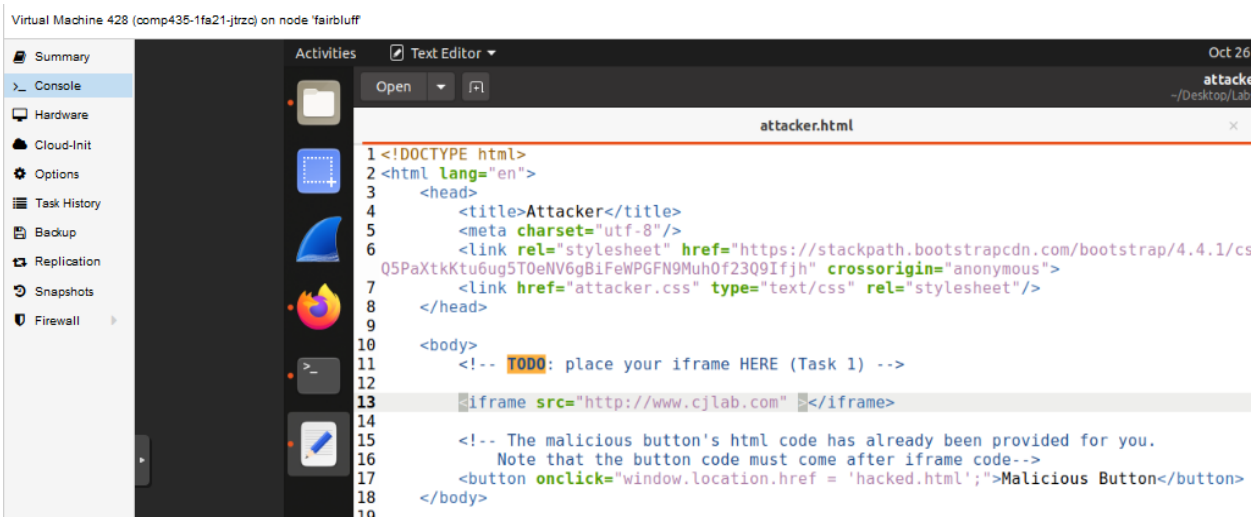
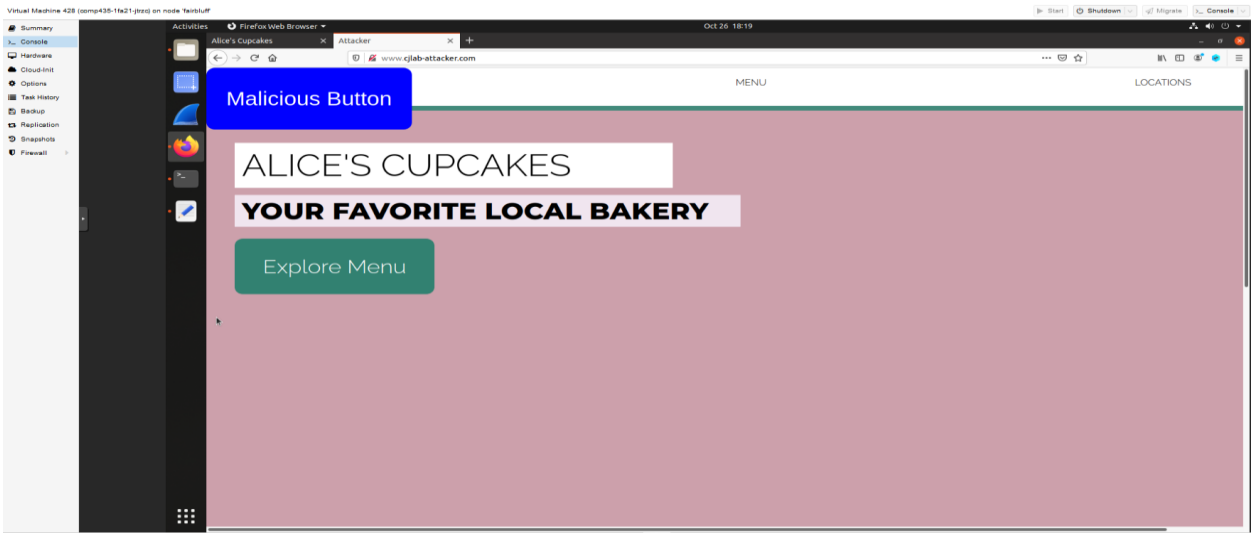


Image Three:



Task 2: Let's Get Clickjacking

In the last task, although the attacker's website looks similar to the defender's website, you can still see the malicious button, so it would be unlikely someone would click on it. In this task, the goal is to take the button and place it over the Explore Menu button on the defender's website, but have the malicious button not able to be seen, and all of this was done in the CSS file and can be seen in Image Five. Once this is done, and if someone tries to click on the Explore Menu button when on the attacker's website, but they are actually clicking the malicious button. The make the button invisible, the opacity was set to 0, therefore it could not be seen. In Image Five, this is what the button looked like before the

opacity was changed, and Image Six shows what it looks like after the opacity was set to 0. Now when you try to click on the Explore Menu Button, what happens can be seen in Image Seven, and what happens is that you actually click the malicious button and it brings you to a page saying “You Have Been Hacked!!” simulating some sort of malicious behavior that could happen. There are a vast number of different things that could happen when the malicious button is pressed, but one example is that a bank transfer is executed, especially if the user is already logged into their bank account online. For example, the person being attacked goes to this attacker’s page and thinks it is the normal defender’s page. The attacker, in the background, checks if the user is signed into their back account, and if they are, they start executing some script to start a transfer of funds to their account. Now when the person being attacked clicks the Explore Menu button, which is really the malicious button, they are actually authorizing the transfer of funds.

Image Four:

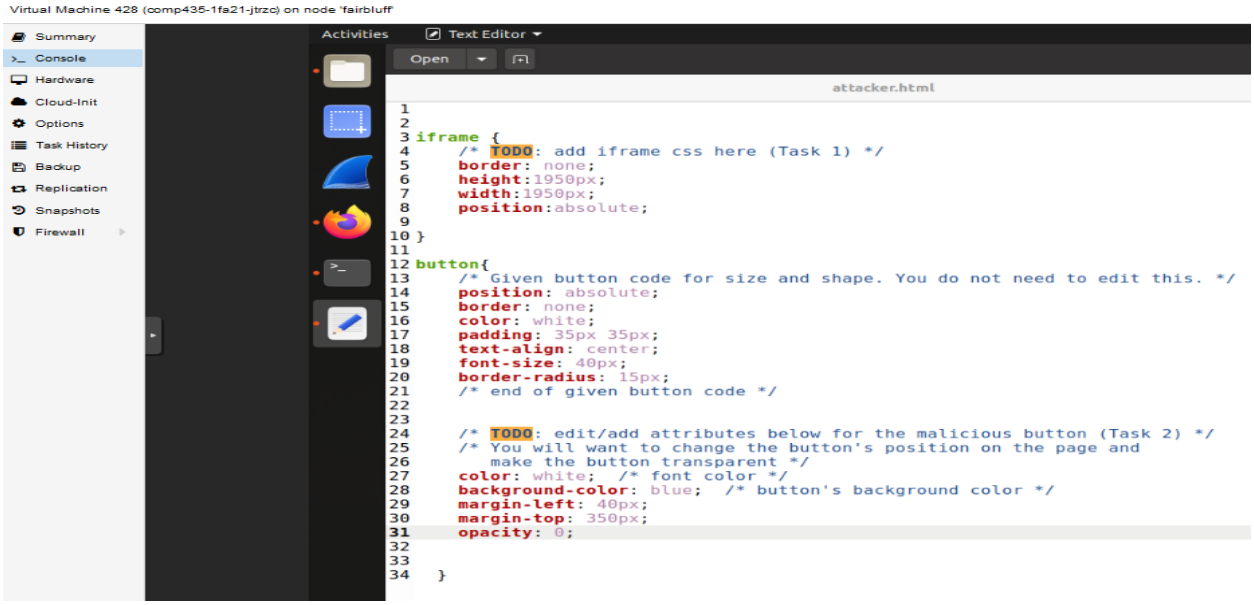


Image Five:

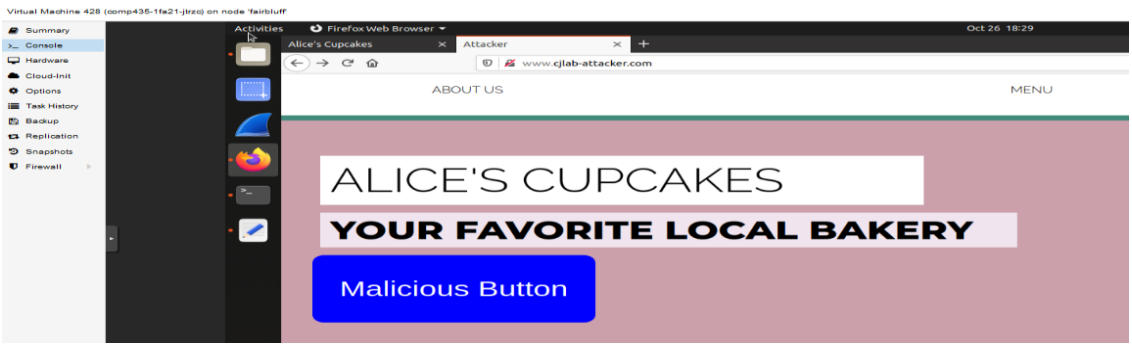


Image Six:

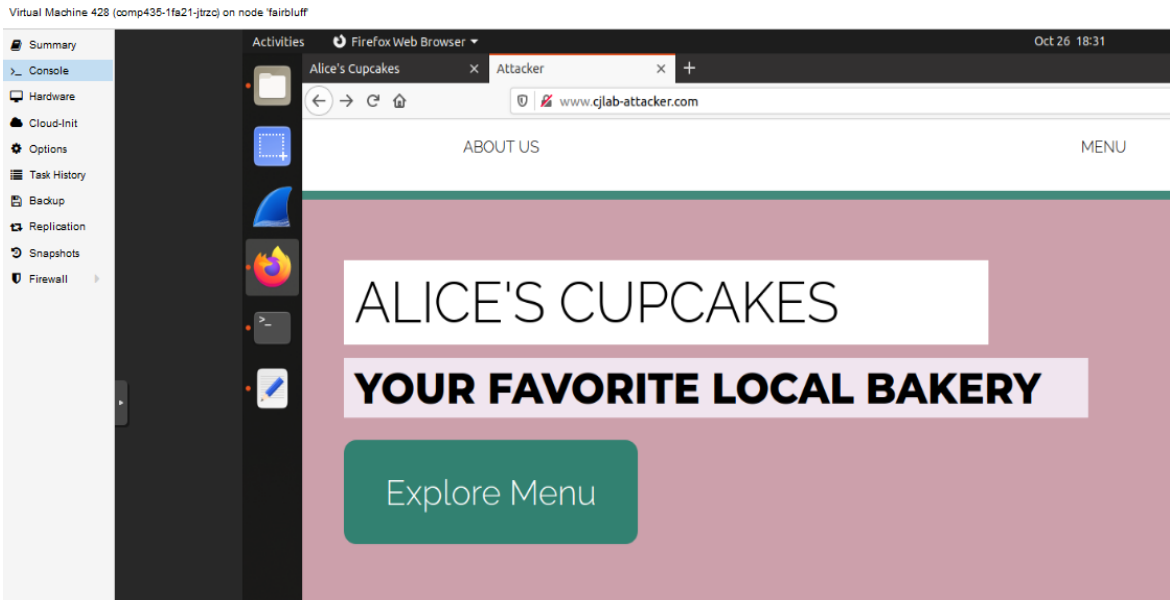
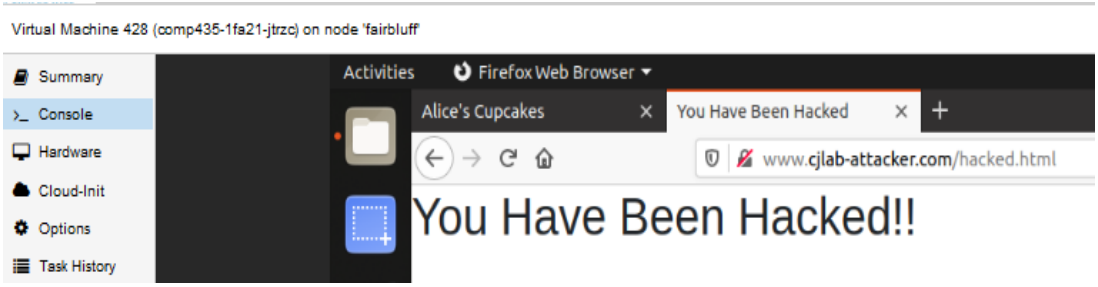


Image Seven:



Task 3: Bust That Frame!

Now that the malicious button cannot be seen, it is a viable attack if someone goes to the site and tries to click on the Explore Menu button, the malicious code would be run. In this task, the html of the defender's site is going to be modified so that it cannot be used in a clickjacking attack with an iframe. This is done by inserting some JavaScript code to ensure that it cannot be opened in any sort of frame. This is done by looking if the top window and the current window are the same window, and if they are not, then the current site's window should be set as the current window, which can be seen in Image Eight. Now when you navigate to the attacker's website it will just load the defender's website, and when you click the Explore Menu button, it does not click the malicious button anymore. Unlike before, the iframe was not loaded was the defender's website was not loaded in the attacker's website

where the malicious button was hidden over the Explore Menu button. This prevents the attack from being possible, and Image Nine shows the defender's website being loaded after trying to go to the attacker's website.

Image Eight:

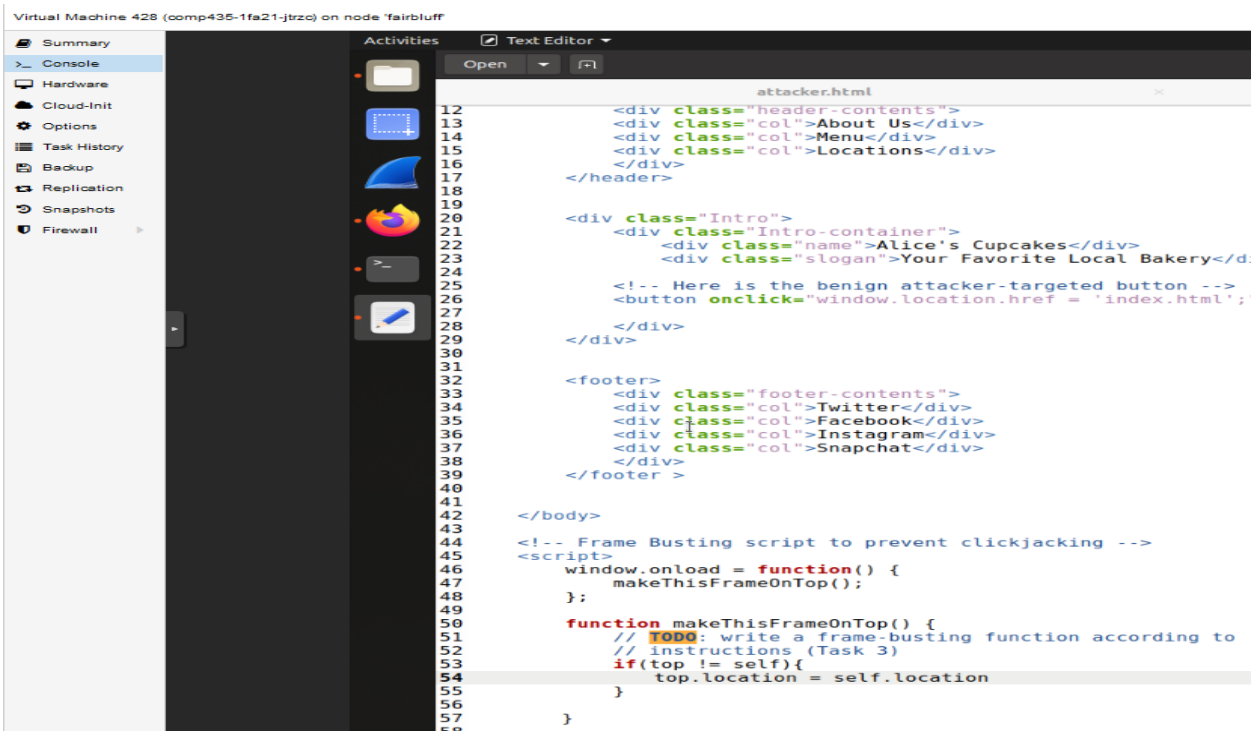
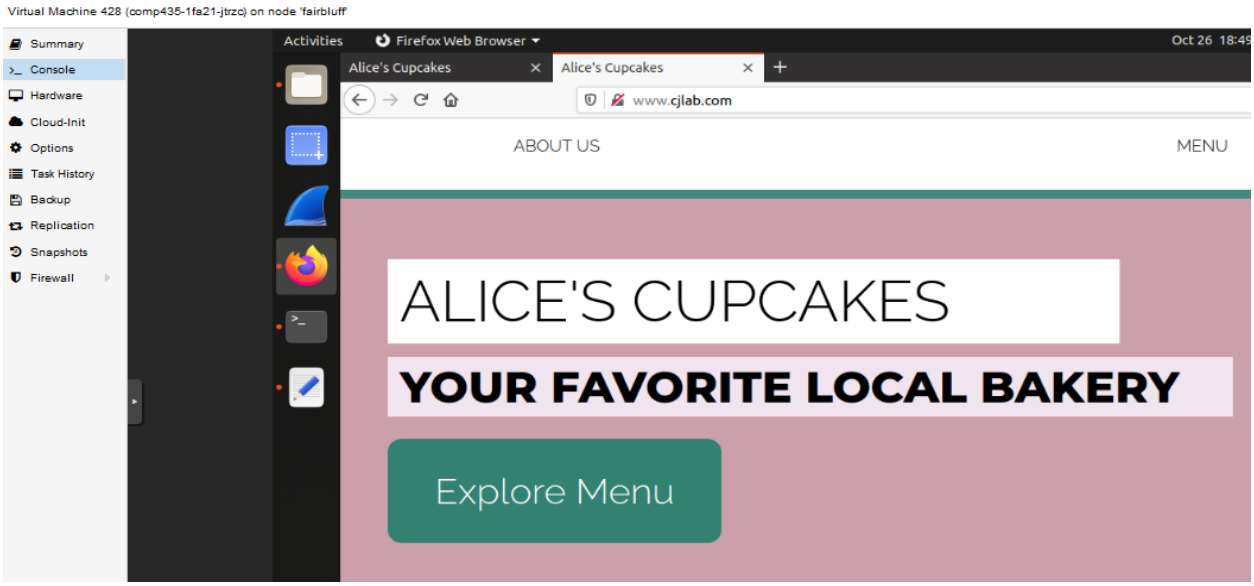


Image Nine:



Task 4: Attacker Countermeasure (Bust the Buster)

In the previous attack, the HTML of the defender's webpage was altered to prevent the iframe from being loaded. In this task the iframe will be altered to get around that change. By adding the sandbox attribute, Image Ten, to the iframe, it prevents the JavaScript from being run from the defender.html file when it is loaded into the iframe. The sandbox attribute restricts the whatever is in the frame, and by default blocks different actions like preventing scripts, downloads, popups, etc. In this case, the blocking of the JavaScript is what allows the attack to work again, as now when you go to the attacker's website, Image Eleven, it stays there and clicking the Explore Menu button once again executes the attack.

Image Ten:

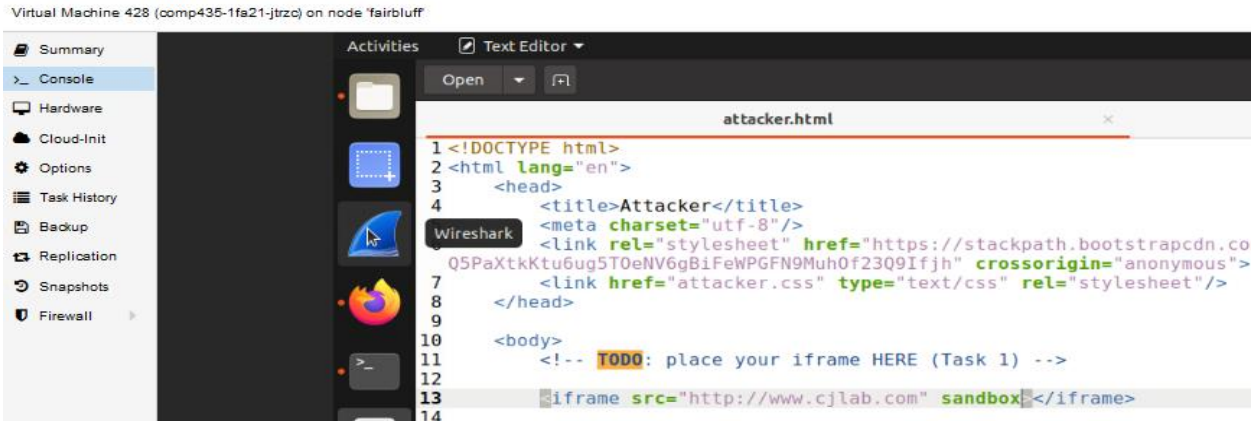
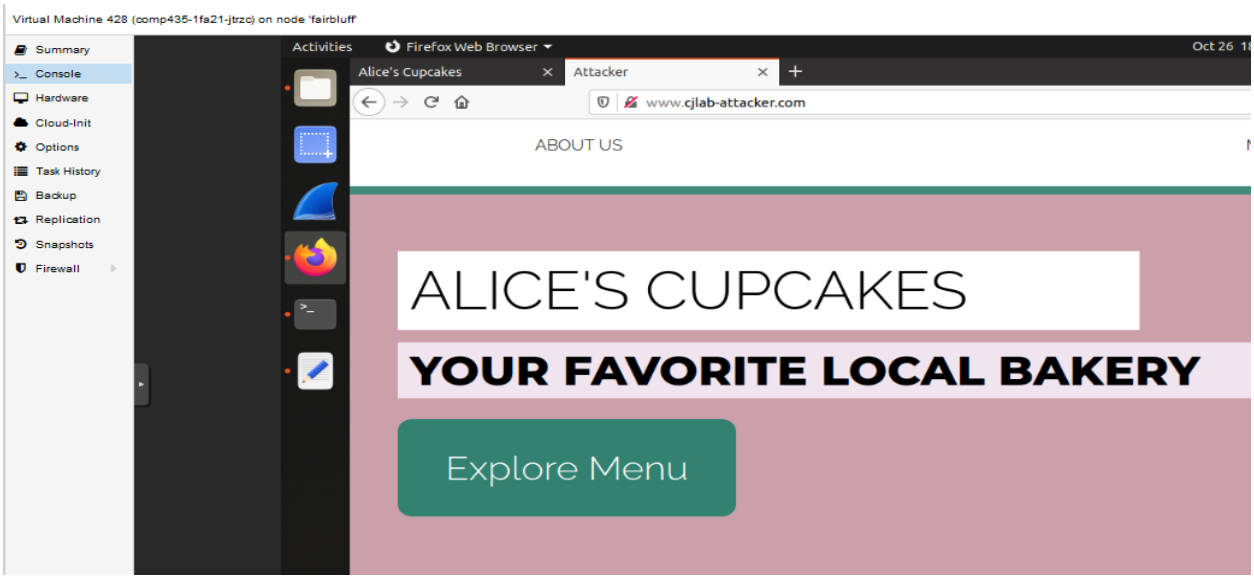


Image Eleven:



Task 5: The Ultimate Bust

As seen in the last task, there are ways to get around some of the front-end countermeasures that can be implemented. In this task, some back-end countermeasures will be added to prevent the attack from being able to occur. This can be done by adding some HTTP headers, specifically X-Frame-Options and Content-Security-Policy. X-Frame-Options is an HTTP header that blocks rendering of the page in any sort of frame like an iframe. With this on, and while using a supported browser, the loading of the attacker's website will be blocked. The X-Frame-Options being set to DENY does not allow the page to be displayed in any frame, which is something the attacker uses. The other header is the Content-Security-Policy which with frame-ancestors set to none, it accomplishes the same thing as the X-Frame-Options, but is slightly different. The CSP sets what pages can use an iframe or another frame to embed the page on a website, but when this is set to none, the attacker can no longer use the iframe to embed the defender's webpage in their own. Using a compatible browser, it also blocks the page from being loaded, and gives a warning message. Whether one or the other of the HTTP headers is turned on, or they both are, the attacker's website is blocked from being loaded.

Image Twelve:

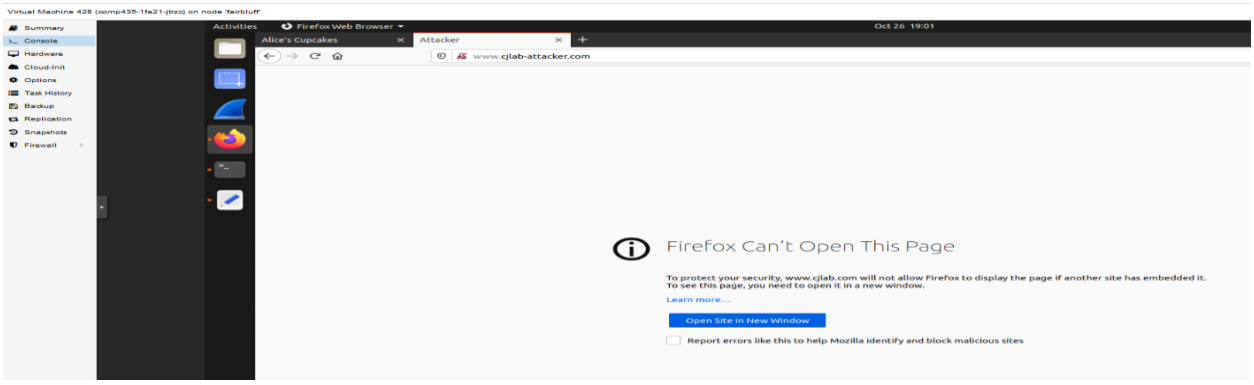


Image Thirteen: