Clickjacking

1 Overview

Clickjacking, also known as a "UI redress attack," is an attack that tricks a user into clicking on something they do not intend to when visiting a webpage, thus "hijacking" the click. In this lab, we will explore a common attack vector for clickjacking: the attacker creates a webpage that loads the content of a legitimate page but overlays one or more of its buttons with invisible button(s) that trigger malicious actions. When a user attempts to click on the legitimate page's buttons, the browser registers a click on the invisible button instead, triggering the malicious action.

Example scenario. Suppose an attacker acquires the domain `starbux.com` and creates a website with that URL. The site first loads the legitimate target website `starbucks.com` in an iframe element spanning the entire webpage, so that the malicious `starbux.com` website looks identical to the legitimate `starbucks.com` website. The attacker's site then places an invisible button on top of the 'Menu' button on the displayed `starbucks` page; the button triggers a 1-click purchase of the attacker's product on Amazon. If the user is logged on to Amazon when they try to click the legitimate button, the inadvertent click on the invisible button will make the unintended purchase without the user's knowledge or consent.

Topic coverage. This lab covers the following topics:

- Clickjacking attack
- Countermeasures: frame busting and HTTP headers
- Iframes and sandboxes
- JavaScript

Lab environment. This lab has been tested on our pre-built Ubuntu 20.04 VM, which can be downloaded from the SEED website. Since we use containers to set up the lab environment, this lab does not depend much on the SEED VM. You can do this lab using other VMs, physical machines, or VMs on the cloud.

2 Lab Environment Setup

In this lab, we will use two websites. The first is the vulnerable homepage of the fictional business "Alice's Cupcakes", accessible at `http://www.cjlab.com`. The second is the attacker's malicious web site that is used for hijacking clicks intended for the Alice's Cupcakes page, accessible at `http://www.cjlab-attacker.com`. We use containers to set up the web servers.

2.1 Container Setup and Commands

Please download the `Labsetup.zip` file to your VM from the lab's website, unzip it, enter the `Labsetup` folder, and use the `docker-compose.yml` file to set up the lab environment. Detailed explanation of the content in this file and all the involved `Dockerfile` can be found from the user manual, which is linked to the website of this lab. If this is the first time you set up a SEED lab environment using containers, it is very important that you read the user manual.

In the following, we list some of the commonly used commands related to Docker and Compose. Since we are going to use these commands very frequently, we have created aliases for them in the `.bashrc` file (in our provided SEEDUbuntu 20.04 VM).

```
$ docker-compose build  # Build the container image
$ docker-compose up     # Start the container
$ docker-compose down   # Shut down the container

// Aliases for the Compose commands above
$ dcbuild       # Alias for: docker-compose build
$ dcup          # Alias for: docker-compose up
$ dcdown        # Alias for: docker-compose down
```

All the containers will be running in the background. To run commands on a container, we often need to get a shell on that container. We first need to use the "`docker ps`" command to find out the ID of the container, and then use "`docker exec`" to start a shell on that container. We have created aliases for them in the `.bashrc` file.

```
$ dockps        // Alias for: docker ps --format "{{.ID}}  {{.Names}}"
$ docksh <id>   // Alias for: docker exec -it <id> /bin/bash

// The following example shows how to get a shell inside hostC
$ dockps
b1004832e275  hostA-10.9.0.5
0af4ea7a3e2e  hostB-10.9.0.6
9652715c8e0a  hostC-10.9.0.7

$ docksh 96
root@9652715c8e0a:/#

// Note: If a docker command requires a container ID, you do not need to
//       type the entire ID string. Typing the first few characters will
//       be sufficient, as long as they are unique among all the containers.
```

If you encounter problems when setting up the lab environment, please read the "Common Problems" section of the manual for potential solutions.

2.2 Websites

We use two containers, one running the website for Alice's Cupcakes (the "defender", with IP address `10.9.0.5`) , and the other running the website for the attacker (with IP address `10.9.0.105`). The IP addresses for these two containers must be consistent in the `docker-compose.yml` file and the `/etc/hosts` file (see below), and we recommend not changing them from their default values.

The Defender container. The website for Alice's Cupcakes is hosted on an Apache web server. The website setup is included in `apache_defender.conf` inside the defender image folder. The configuration specifies the URL for the website and the folder where the web application code is stored. The configuration also contains placeholders for HTTP response headers returned by the server (commented out), which will be filled in during the course of the lab as a defense countermeasure.

```
<VirtualHost *:80>
     DocumentRoot /var/www/defender
     ServerName www.cjlab.com
#    Header set <Header-name> "<value>";
#    Header set Content-Security-Policy " \
#             <directive> '<value>'; \
#           "
</VirtualHost>
```

Because we need to modify the defender's web page inside this container, for convenience as well as to allow the modified files to persist beyond the containers, we have mounted a folder (`Labsetup/defender` on the hosting VM) to the container's `/var/www/defender` folder, which is the `DocumentRoot` folder in our Apache configuration. Therefore any files we modify inside the `defender` folder on the VM will be seen as modified by the defender's web server on the container.

The Attacker container. The attacker's website is also hosted on an Apache web server. The website setup is included in `apache_attacker.conf` inside the attacker image folder. The Apache configuration for this website is as follows:

```
<VirtualHost *:80>
    ServerName www.cjlab-attacker.com
    DocumentRoot "/var/www/attacker"
    DirectoryIndex attacker.html
</VirtualHost>
```

Because we need to modify the attacker's web page inside this container, for convenience as well as to allow the modified files to persist beyond the containers, we have mounted a folder (`Labsetup/attacker` on the hosting VM) to the container's `/var/www/attacker` folder, which is the `DocumentRoot` folder in our Apache configuration. Therefore any files we modify inside the `attacker` folder on the VM will be seen as modified by the attacker's web server on the container.

Important note. Any time you make updates to the websites, you may need to clear the browser's cache and/or rebuild and restart the containers for the change to be visible, depending on the scope of the change.

DNS configuration. We access the defender website and the attacker website using their respective URLs. To enable these hostnames to be mapped to their corresponding IP addresses, we need to add the following entries to the `/etc/hosts` file on the VM. You need to use the root privilege to change this file (using `sudo`).

```
10.9.0.5        www.cjlab.com
10.9.0.105      www.cjlab-attacker.com
```

Test: defender. Build and start the containers, then navigate to the page `http://www.cjlab.com` on the VM. You should see a page for Alice's Cupcakes, a fictional local bakery. If the whole site does not fit in your window, use `Ctrl + (minus key)` and `Ctrl + (plus key)` to zoom out and in respectively until the site fits. Note that the header and footer buttons on the site are just placeholders and do not contain live links.

Test: attacker. Next, navigate to the page `http://www.cjlab-attacker.com`. You should see a single button which, when clicked, takes you to a page that tells you you've been hacked. In a real attack, this button could perform a variety of malicious actions.

The goal of the attacker is to overlay this button onto a view of the defender's webpage displayed on the attacker's site, so that a victim user will inadvertently click the malicious button when they think they are clicking a button on the defender's webpage.

3 Lab Tasks

3.1 Task 1: Copy that site!

In the `defender` folder, you will find the files comprising the website for Alice's Cupcakes: `index.html` and `defender.css`. In the `attacker` folder, you will find the files comprising the attacker's website: `attacker.html` and `attacker.css`. You will be making changes to all of these files except `defender.css` throughout the lab.

Our first step as the attacker is to add code to `attacker.html` so that it mimics the Alice's Cupcakes website as closely as possible. A common way to do this is with an HTML Inline Frame element ("iframe"). An iframe enables embedding one HTML page within another. The `src` attribute of the iframe specifies the site to be embedded, and when the iframe code is executed on a page, the embedded site is loaded into the iframe.

Embed the defender's site into the attacker's site.

- Add an iframe HTML element in `attacker.html` that pulls from `http://www.cjlab.com`.
- Modify the CSS in `attacker.css` using the `height`, `width`, and `position` attributes to make the iframe cover the whole page and the button overlay the iframe.
- Hints:
  - Explicitly set the iframe to have no border.
  - Investigate the 'absolute' and 'relative' settings of the `position` attribute to determine which should be used.
- Test your changes by navigating to the attacker's website. (Remember that you may need to clear the browser's cache and reload the page to see changes made after the initial load.)

Question:

1. With the iframe inserted, what does the attacker's website look like?

3.2 Task 2: Let's Get Clickjacking!

Basic clickjacking attack. Add code to the CSS specification of a "button" object given in `attacker.css` to make the malicious button in `attacker.html` invisible. Position the button so that it covers the "Explore Menu" button within the iframe added in the previous Task. There are a variety of ways to accomplish this Task; we recommend using the CSS attributes `margin-left`, `margin-top`, `color`, and `background-color`. When this Task is complete, you will have a functioning clickjacking attack.

Questions:

2. How does the appearance of the attacker's site compare to that of the defender's site?
3. What happens when you click on the "Explore Menu" button on the attacker's site?
4. Describe an attack scenario in which the style of clickjacking implemented for this Task leads to undesirable consequences for a victim user.

3.3 Task 3: Bust That Frame!

"Frame busting" is the practice of preventing a web page from being displayed within a frame, thus defending against the type of attack implemented in the previous Task. One way to bust frames is to include script code in the webpage source that prevents the site from being framed – that is, it prevents other sites from opening the webpage in an iframe. In this Task we will add script code to the defender's webpage that ensures 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.

Write the frame-busting script. Open the file `defender/index.html`, which contains code for the Alice's Cupcakes homepage. We would like to protect the homepage from clickjacking. Your task is to fill in the Javascript method called $makeThisFrameOnTop()$. Your code should 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). If not, 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. This should be a simple method and take no more than a few lines of code. Test it out and confirm that your script successfully stops the clickjacker.

Reminder. Remember that any time you make changes to one of the websites, you may need to clear the browser's cache and reload the page for the changes to take effect.

Questions:

5. What happens when you navigate to the attacker's site now?
6. What happens when you click the button?

3.4 Task 4: Attacker Countermeasure (Bust the Buster)

Disable the frame-busting script. Now let's explore how an attacker can create a workaround for front-end clickjacking defenses like frame busting. There are multiple workarounds, but one of the simplest in the current scenario is to add the `sandbox` attribute to the malicious iframe. Read more about the `sandbox` attribute on this page about iframes, then add the `sandbox` attribute to the iframe in `attacker.html` and answer the following questions.

Questions:

7. What does the `sandbox` attribute do? Why does this prevent the frame buster from working?
8. What happens when you navigate to the attacker's site after updating the iframe to use the `sandbox` attribute?
9. What happens when you click the button on the attacker's site?

3.5 Task 5: The Ultimate Bust

As we saw in the previous Task, 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. To prevent clickjacking attacks more comprehensively, we need to set up back-end (i.e. server-side) defenses. Modern websites can cooperate with common browsers to provide such defenses.

The front-end attacks presented in previous Tasks 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. By providing one of these HTTP headers with its response to a request for content, the website can instruct browsers to only display the content according to specific matching rules designed to exclude clickjacking attack scenarios. These headers are not part of the official HTTP specification, but are processed by many common browsers.

One such header is called "X-Frame-Options", and a newer, more popular one is called "Content-Security-Policy". This header can include a diverse set of key-value directives to implement a site's Content Security Policy (CSP) with the intention of stopping many common attacks while perserving the desired content-sharing behavior of the site. The CSP header directive relevant for preventing clickjacking is "frame-ancestors" , which specifies the valid parents that may embed a page in a frame.

You can read more about the XFO attribute here, and more about CSPs here.

Modify the defender's response headers. Open the Apache configuration file for the defender's website (`apache_defender.conf` in the defender's image folder). Uncomment the lines that specify the HTTP response headers served with the page, and substitute appropriate text in order to prevent the clickjacking attack. Specifically, set the X-Frame-Options (XFO) header to the value `"DENY"` and the Content-Security-Policy (CSP) header to contain the directive `"frame-ancestors 'none' "`.

Hint. Because you are making a change to the server's configuration, you will need to rebuild and restart the containers for the change to take effect.

Questions:

10. What is the X-Frame-Options HTTP header attribute, and why is it set to "DENY" to prevent the attack?
11. What is the Content-Security-Policy header attribute, and why is it set to "frame-ancestors 'none' " to prevent the attack?
12. What happens when you navigate to the attacker's site after modifying each response header (one at a time)? What do you see when you click the button?

Learn more. There are other ways to perform clickjacking besides the one explored in this lab, and many possible malicious consequences beyond the ones suggested here. To learn more about clickjacking, visit the Open Web Application Security Project (OWASP) page on Clickjacking here.

4 Submission

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. You also need to provide explanation to the observations that are interesting or surprising. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not receive credits.