23. Clickjacking/User Interface (UI) Attacks

23.1. Clickjacking Attacks

Many of the web attacks we’ve seen involve forcing a victim to click on an attacker-generated link (reflected XSS), or forcing a victim to visit an attacker-controlled website (CSRF). How might an attacker achieve this?

UI attacks (or *clickjacking attacks*) are a category of attacks that try to fool a victim into inadvertently clicking on an attacker-supplied input. The end goal of these attacks is to “steal” a click from the user, so that the user loads something controlled by the attacker (possibly for a further attack). Many UI attacks rely on clever visual tricks to deceive the user.

Download buttons are a classic example of clickjacking. When you visit a website to download a file, you might see many different download buttons with different shapes and colors. One of these is the true download button, and the others are malicious download buttons that actually take you to attacker-controlled websites or perform other malicious actions in your browser. An unwitting user might click on the wrong download button and be sent to the attacker website. The malicious download buttons could be added to the website through a different web exploit (e.g. stored XSS) or as a paid advertisement.

Depending on how much control the attacker has over the page, more sophisticated clickjacking attacks are possible:

- The attacker could manipulate an HTML form so that the user sees a payment of $5, but the underlying form will actually submit a payment of $50.
- The attacker could draw a fake cursor on the page. The user sees the fake cursor over a legitimate button and clicks, but their real cursor has actually clicked on a malicious link.
- The attacker could draw an entire browser on the page. The user sees an address bar and clicks, but they have actually clicked on a fake address bar generated by the attacker (with a malicious link behind the address bar).

23.2. Clickjacking Defenses

There are many ways to defend against clickjacking attacks. The general idea is to force the user to make sure that they're clicking on what they intended to click.

Confirmation pop-ups: If the user clicks on a link or button that will perform some potentially dangerous activity (e.g. opening a website, executing Javascript, downloading a file), display a pop-up asking the user to confirm that this is their intended action. However, users might still click on the pop-up without reading it, especially if they're too frequent. Remember to consider human factors!

UI randomization: Randomize the location of certain elements on a website. For example, a submit button could alternately be located at the left side of the screen and the right side of the screen. This makes it harder for attackers to draw a fake submit button over the real submit button, because they won't know where it's located. However, webpages that look different every time could pose usability problems.

Direct the user's attention to their click: This can be done by freezing the rest of the screen besides the area directly around the user's cursor, or by highlighting the user's cursor. This will make the user less likely to be fooled by a fake cursor and force them to focus on where their real cursor is pointing. The user's clicks can also be invalidated if the user tries to click outside of a relevant portion of the screen.

Delay the click: Force the user to hover over the desired button for some amount of time before allowing the user to click the button. This forces the user to spend some time looking at where they're clicking before they actually perform the click.