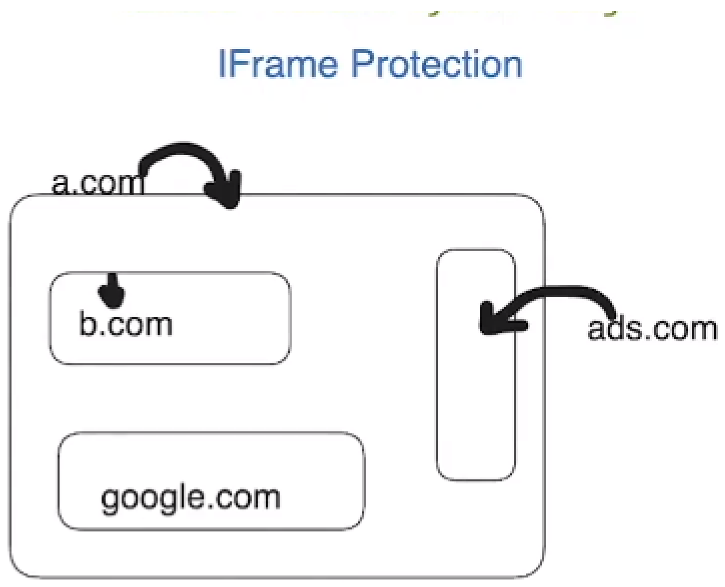
iFrame protection

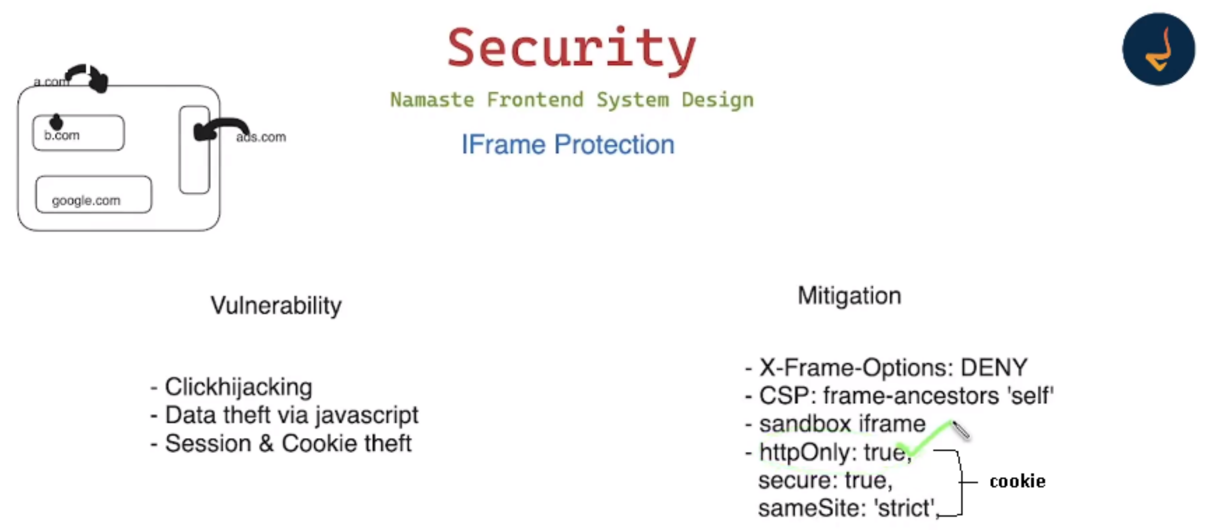
iFrames are widely used. We can embed different websites, ads, videos in our own website.



a.com is our website. We have embedded b.com , google.com and ads using iFrame in our website.

If not handled properly, b and ads can access different parts of our web application which we don’t want. It posses a security threat to us.

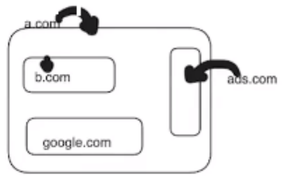
Take another example, we have embedded google and we are allowing users to do google search in our application. So basically we are using google’s all resources for free which shouldn’t be the case.



Security

Namaste Frontend System Design

IFrame Protection



Vulnerability

- Clickhijacking
- Data theft via javascript
- Session & Cookie theft

Mitigation

- X-Frame-Options: DENY
- CSP: frame-ancestors 'self'
- sandbox iframe
- httpOnly: true,
  secure: true,
  sameSite: 'strict', } cookie

VULNERABILITIES

1. Click Hijacking

We place an transparent iFrame on top of our website. User clicks on submit button. User thinks he has click on submit button but he/she clicks on transparent iFrame's button and ends up sending critical info

2. Data theft via javascript

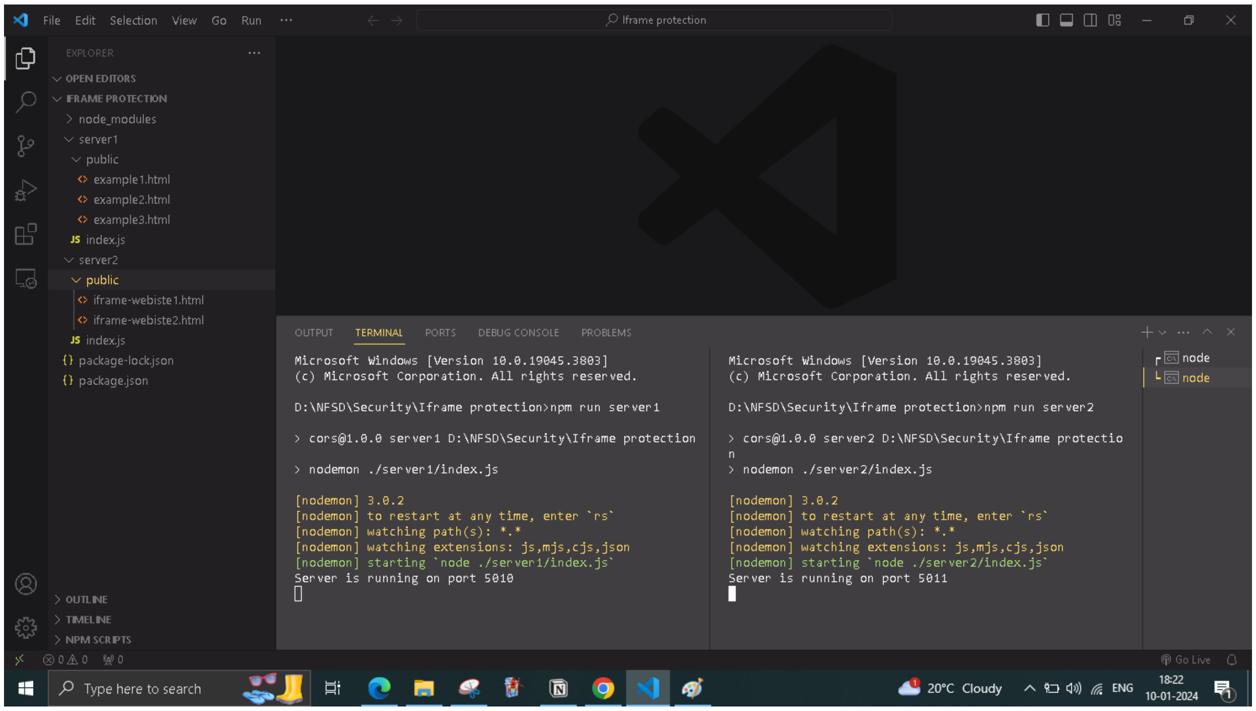
I am parent window(a)and I am able to access something in child window(b) or vice versa

3. Session and cookie theft

Parent to child or vice versa

Project folder code

https://github.com/namastedev/namaste-frontend-system-design/tree/master/Security/IframeProtection



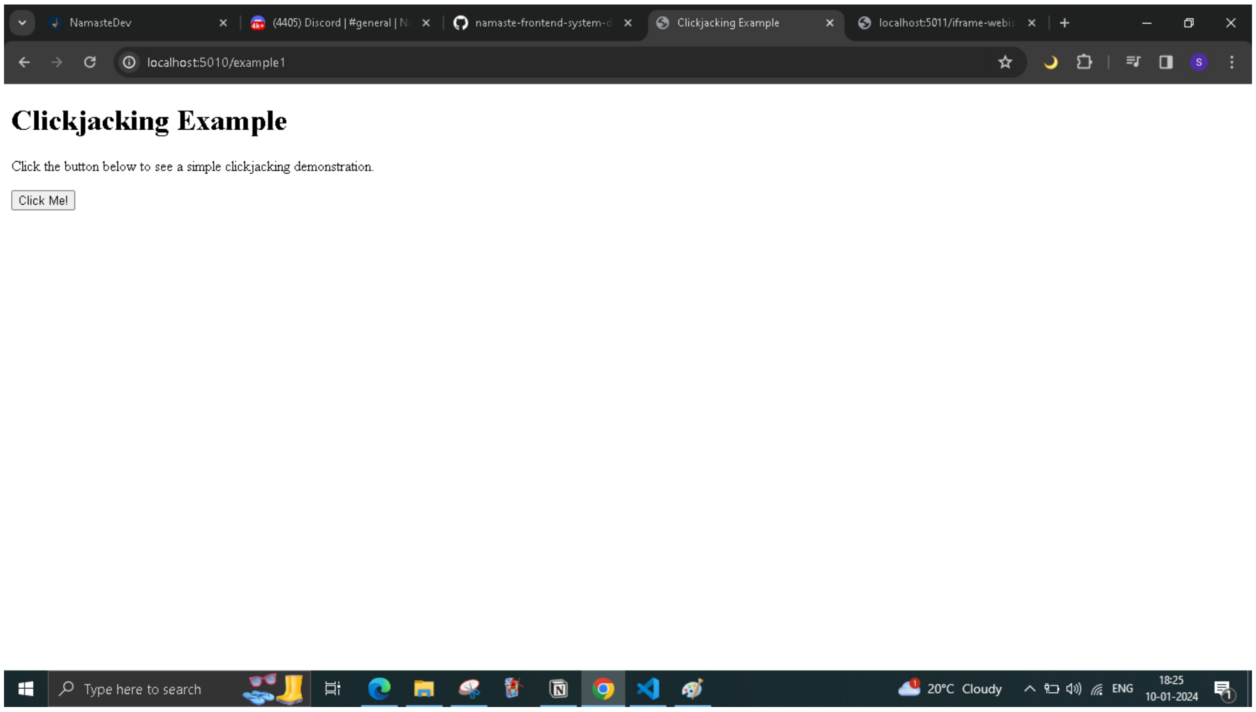
```
Microsoft Windows [Version 10.0.19045.3803]
(c) Microsoft Corporation. All rights reserved.

D:\NFSD\Security\Iframe protection>npm run server1

> cors@1.0.0 server1 D:\NFSD\Security\Iframe protection

> nodemon ./server1/index.js

[nodemon] 3.0.2
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node ./server1/index.js`
Server is running on port 5010
```

```
Microsoft Windows [Version 10.0.19045.3803]
(c) Microsoft Corporation. All rights reserved.

D:\NFSD\Security\Iframe protection>npm run server2

> cors@1.0.0 server2 D:\NFSD\Security\Iframe protectio
n
> nodemon ./server2/index.js

[nodemon] 3.0.2
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node ./server2/index.js`
Server is running on port 5011
```

example1 html

```html
<!DOCTYPE html>
<html>
  <head>
    <title>Clickjacking Example</title>
  </head>
  <body>
    <h1>Clickjacking Example</h1>
    <p>Click the button below to see a simple clickjacking de
monstration.</p>
    <button id="overlay-button">Click Me!</button>
    <script>
      // Simulate a clickjacking attack by overlaying the but
ton on top of the iframe
      document
        .getElementById("overlay-button")
```

```
         .addEventListener("click", () => {
           alert("Button clicked!");
         });
     </script>
   </body>
 </html>
```


NamasteDev | (4405) Discord | #general | N | namaste-frontend-system-d | Clickjacking Example | localhost:5011/iframe-webi

localhost:5010/example1

Clickjacking Example

Click the button below to see a simple clickjacking demonstration.

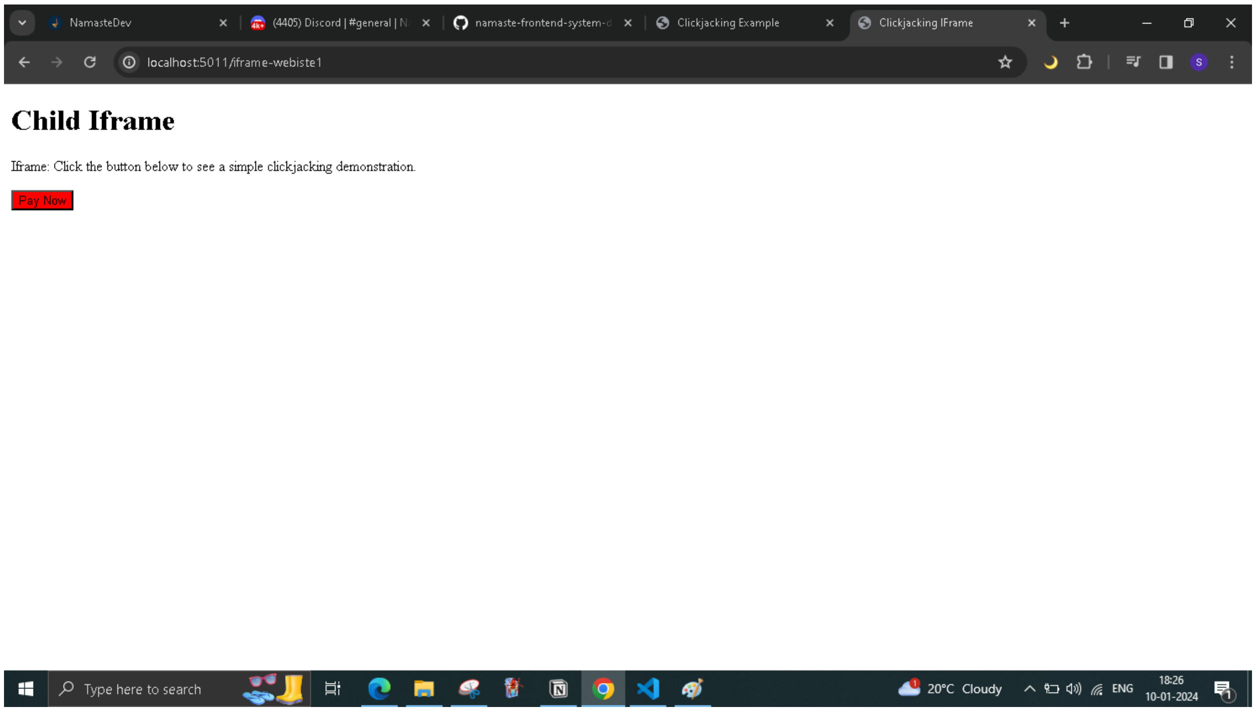
Click Me!

Type here to search | 20°C Cloudy | ENG | 18:25 10-01-2024

iframe-webiste1 html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Clickjacking IFrame</title>
  </head>
  <body>
    <h1>Child Iframe</h1>
    <p>Iframe: Click the button below to see a simple clickja
```

```
cking demonstration.</p>

    <button style="background-color: red;" onclick="alert('cl
icked iframe button')">Pay Now</button>
  </body>
</html>
```



Child Iframe

Iframe: Click the button below to see a simple clickjacking demonstration.

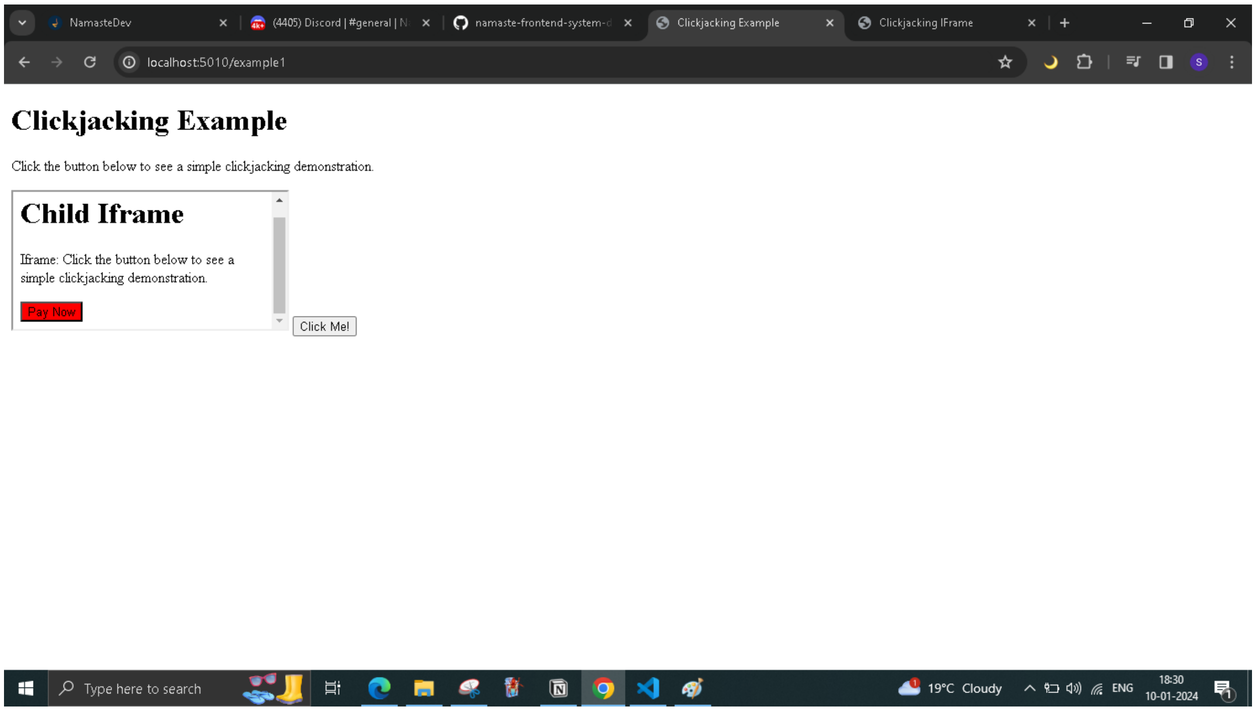
1. CLICK HIJACKING example

Adding iframe of iframe-website in example1

example1

```
<!DOCTYPE html>
<html>
```

```html
  <head>
    <title>Clickjacking Example</title>
  </head>
  <body>
    <h1>Clickjacking Example</h1>
    <p>Click the button below to see a simple clickjacking demonstration.</p>

    <iframe src="http://localhost:5011/iframe-webiste1" id="legit-iframe"></iframe>


    <button id="overlay-button">Click Me!</button>
    <!-- <style>
        #legit-iframe {
            background-color: #ccc;
            opacity: 0;
            position: absolute;
            top: 0px;
            left: 0;
            width: 100%;
            height: 100%;
            z-index: 1;
          }
    </style> -->

    <script>
      // Simulate a clickjacking attack by overlaying the button on top of the iframe
      document
        .getElementById("overlay-button")
        .addEventListener("click", () => {
          alert("Button clicked!");
        });
    </script>
  </body>
</html>
```



Clickjacking Example

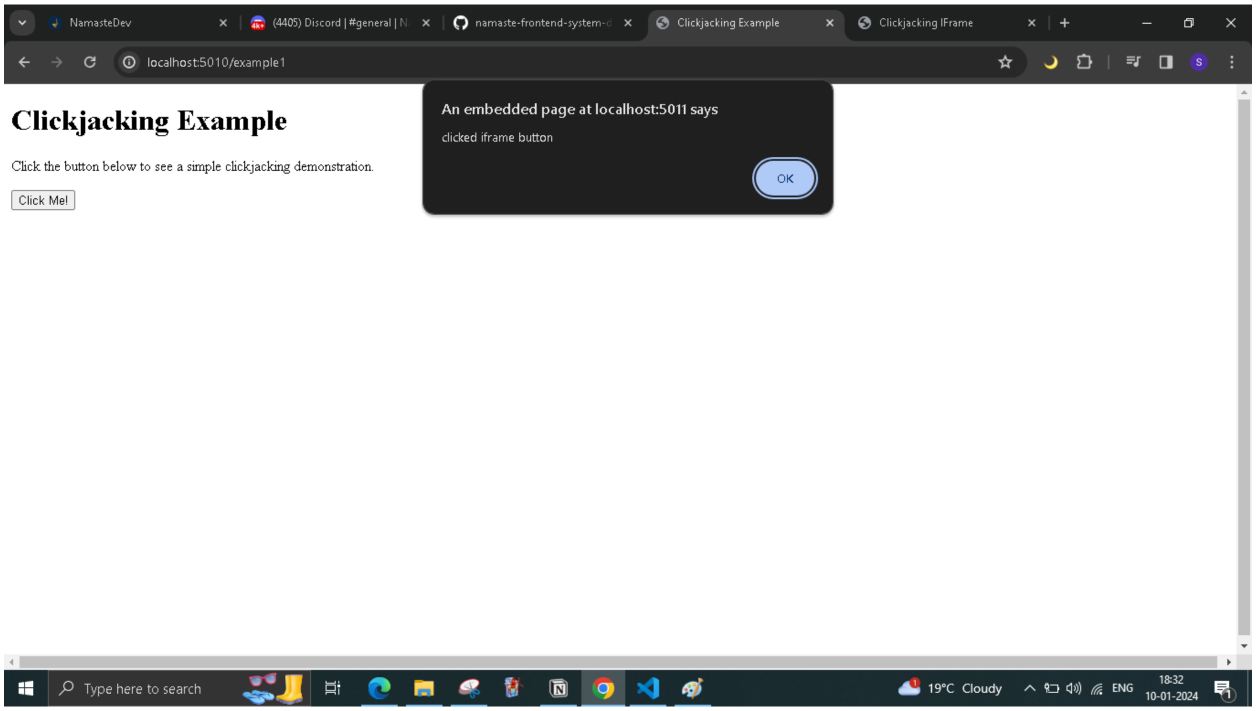
Click the button below to see a simple clickjacking demonstration.

Child Iframe

Iframe: Click the button below to see a simple clickjacking demonstration.

Uncommenting style in above code

Styling child iFrame such that Pay now button overlaps Click me button but opacity is 0(transparent)


Clickjacking Example

Click the button below to see a simple clickjacking demonstration.

An embedded page at localhost:5011 says

clicked iframe button

When user clicks on Click Me, he has actually clicked on Pay now button of iFrame. This is known as click hijacking.

2. Data theft | session theft | cookie theft

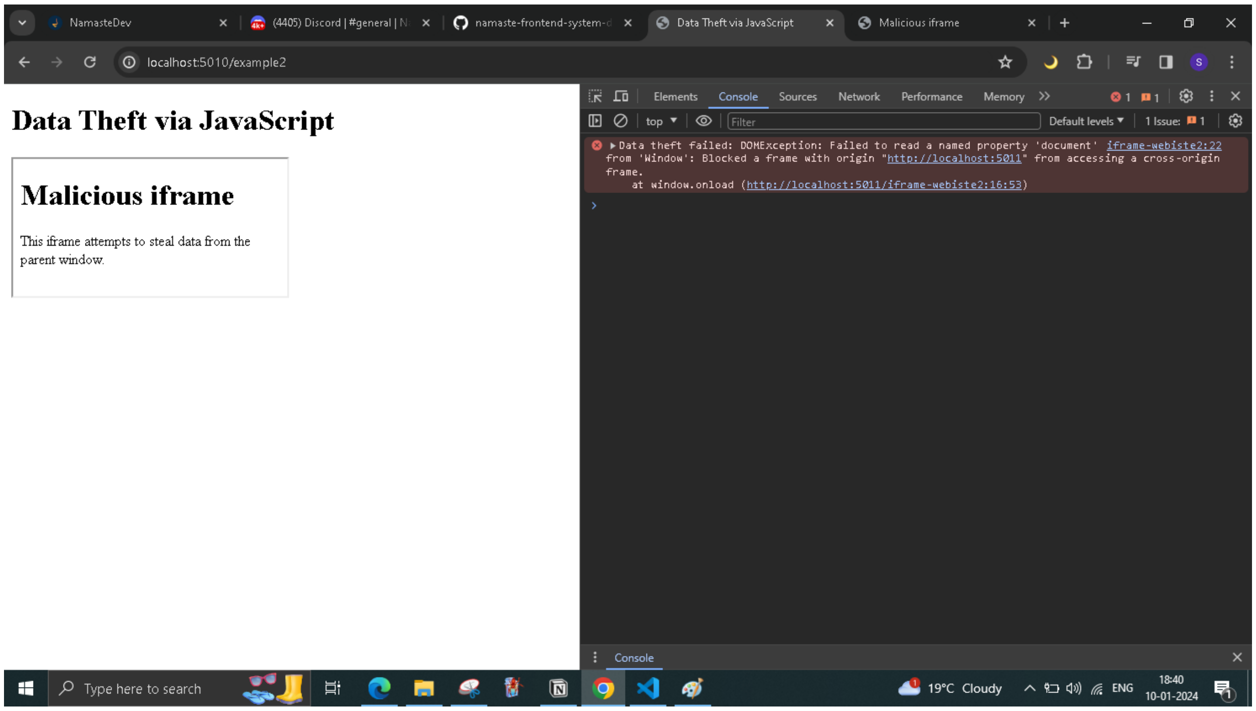
example2 html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Data Theft via JavaScript</title>
    <script>
      function setCookie(name, value, days) {
        var expires = "";

        if (days) {
          var date = new Date();
          date.setTime(date.getTime() + days * 24 * 60 * 60 *
1000);
          expires = "; expires=" + date.toUTCString();
        }

        document.cookie = name + "=" + value + expires + "; p
ath=/";
      }

      setCookie("parentCookie", "secret", 1);
    </script>
  </head>
  <body>
    <h1>Data Theft via JavaScript</h1>

    <iframe
      src="http://localhost:5011/iframe-webiste2"
      sandbox="allow-same-origin allow-scripts allow-modals"
    ></iframe>
  </body>
</html>
```

iFrame-webiste2.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Malicious iframe</title>
</head>
<body>
    <h1>Malicious iframe</h1>
    <p>This iframe attempts to steal data from the parent win
dow.</p>
```

```
    <script>
        window.alert('Hi');
        // Malicious JavaScript code in the iframe
        window.onload = function () {
            try {
                const parentWindow = window.parent;
                const parentDocument = parentWindow.document;

                // Attempt to read html
                const stolenData = parentDocument.innerHtml;
                alert("Stolen Data: " + stolenData);
            } catch (error) {
                console.error("Data theft failed:", error);
            }
        };
    </script>
</body>
</html>
```



Data Theft via JavaScript

Malicious iframe

This iframe attempts to steal data from the parent window.

```
Data theft failed: DOMException: Failed to read a named property 'document'   iframe-webiste2:22
from 'Window': Blocked a frame with origin "http://localhost:5011" from accessing a cross-origin
frame.
    at window.onload (http://localhost:5011/iframe-webiste2:16:53)
```

As you can see, modern browsers doesn’t allow embedded iFrame to access parent dom but issue still exists in older browsers

We can similarly access cookie data as well in older browsers.

MITIGATIONS

1. X-Frame-Options: DENY(older way)

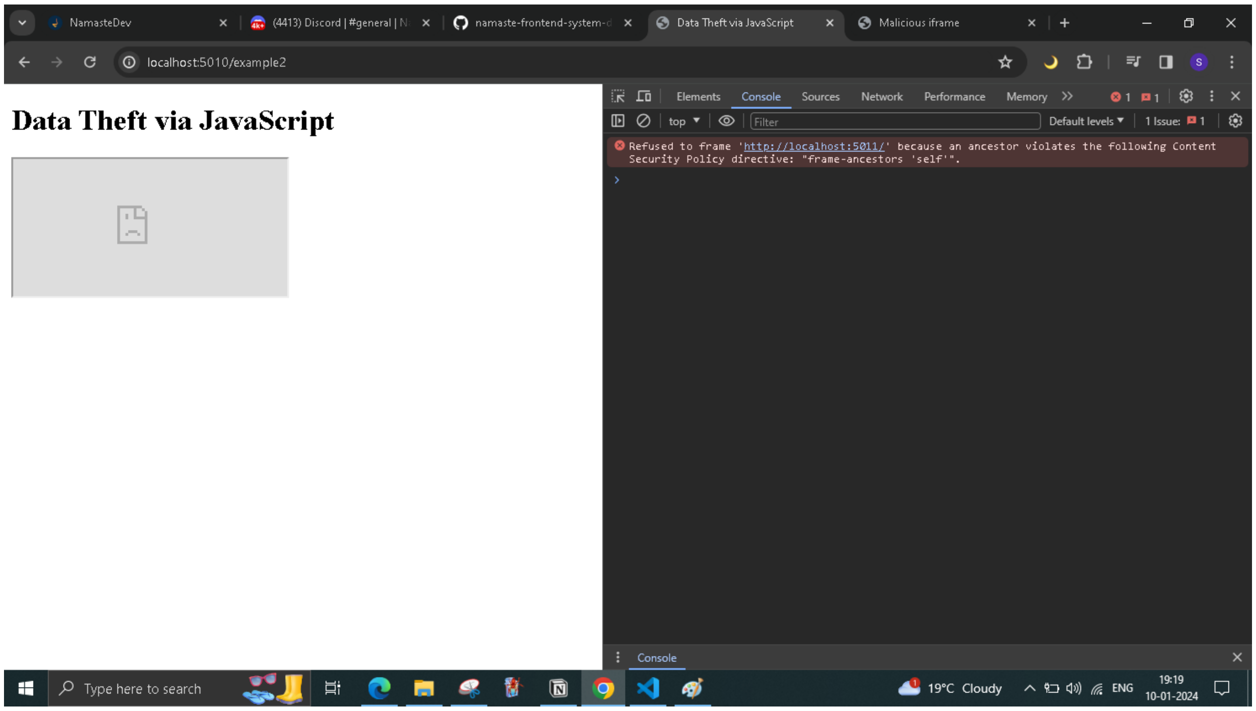
   CSP- frame ancestors ‘self’ (we can use none as well)

   If we don’t want our website to be iframed we can use both the ways above

   If we add below code in index file of server2

```
app.use((req, res, next) => {
    res.setHeader('Content-Security-Policy', "frame-ancestors 
'self'")
    next();
})
```

Try to access example2 as above

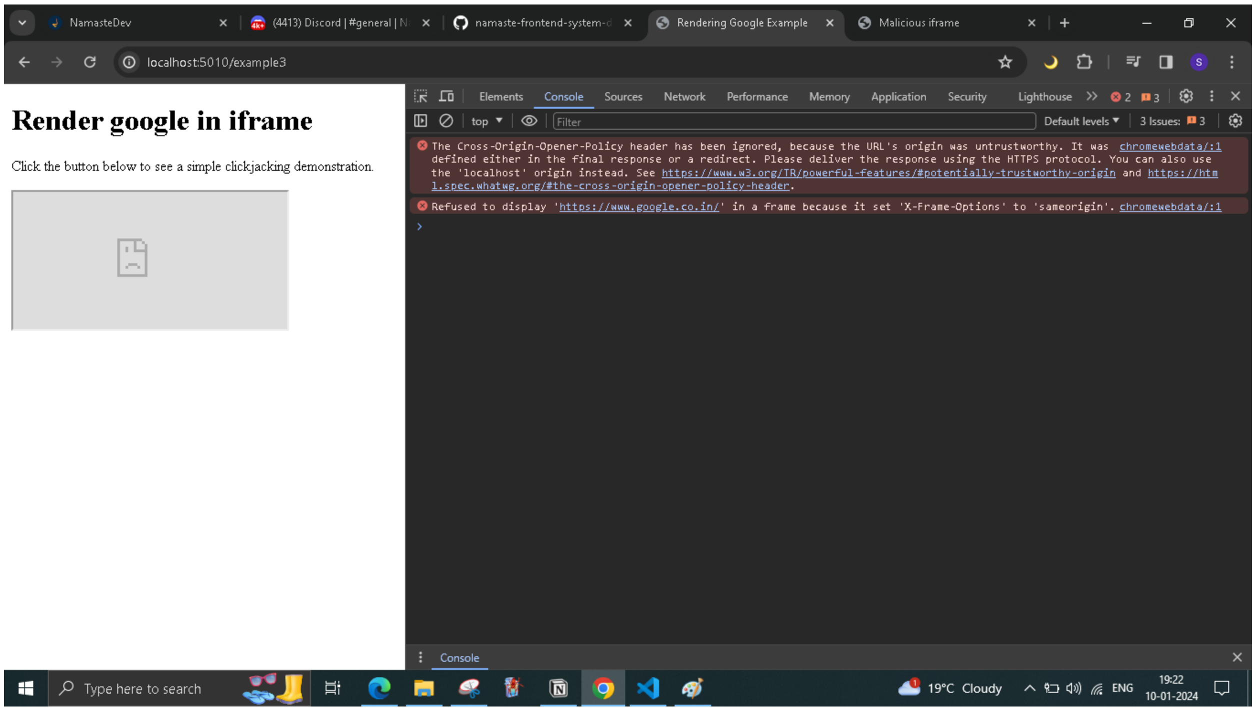


```
Refused to frame 'http://localhost:5011/' because an ancestor violates the following Content
Security Policy directive: "frame-ancestors 'self'".
```

Another example, let’s try to access google as iFrame in our website

example3 html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Clickjacking Example</title>
  </head>
  <body>
    <h1>Render google in iframe</h1>
    <p>Click the button below to see a simple clickjacking de
monstration.</p>

    <iframe src="http://google.co.in"></iframe>
  </body>
</html>
```



Render google in iframe

Click the button below to see a simple clickjacking demonstration.

```
The Cross-Origin-Opener-Policy header has been ignored, because the URL's origin was untrustworthy. It was  chromewebdata/:1
defined either in the final response or a redirect. Please deliver the response using the HTTPS protocol. You can also use
the 'localhost' origin instead. See https://www.w3.org/TR/powerful-features/#potentially-trustworthy-origin and https://htm
l.spec.whatwg.org/#the-cross-origin-opener-policy-header.
Refused to display 'https://www.google.co.in/' in a frame because it set 'X-Frame-Options' to 'sameorigin'. chromewebdata/:1
```

2. sandbox

The `sandbox` attribute enables an extra set of restrictions for the content in the iframe.

When the `sandbox` attribute is present, and it will:

- treat the content as being from a unique origin
- block form submission
- block script execution
- disable APIs
- prevent links from targeting other browsing contexts

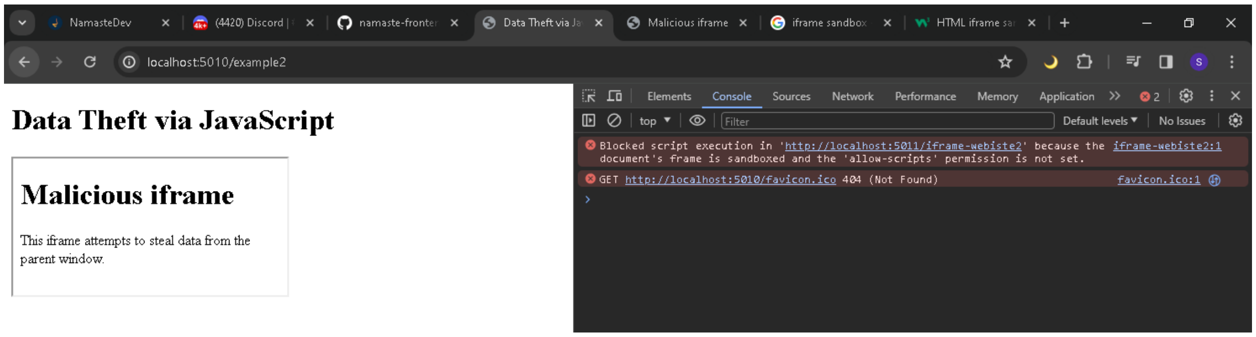
- prevent content from using plugins (through `<embed>`, `<object>`, `<applet>`, or other)
- prevent the content to navigate its top-level browsing context
- block automatically triggered features (such as automatically playing a video or automatically focusing a form control)

Value	Description
(*no value*)	Applies all restrictions
allow-forms	Allows form submission
allow-modals	Allows to open modal windows
allow-orientation-lock	Allows to lock the screen orientation
allow-pointer-lock	Allows to use the Pointer Lock API
allow-popups	Allows popups
allow-popups-to-escape-sandbox	Allows popups to open new windows without inheriting the sandboxing
allow-presentation	Allows to start a presentation session
allow-same-origin	Allows the iframe content to be treated as being from the same origin
allow-scripts	Allows to run scripts
allow-top-navigation	Allows the iframe content to navigate its top-level browsing context
allow-top-navigation-by-user-activation	Allows the iframe content to navigate its top-level browsing context, but only if initiated by user

example2 html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Data Theft via JavaScript</title>
    <script>
      function setCookie(name, value, days) {
        var expires = "";
        if (days) {
          var date = new Date();
          date.setTime(date.getTime() + days * 24 * 60 * 60 *
1000);
          expires = "; expires=" + date.toUTCString();
        }

        document.cookie = name + "=" + value + expires + "; p
ath=/";
      }

      setCookie("parentCookie", "secret", 1);
    </script>
  </head>
  <body>
    <h1>Data Theft via JavaScript</h1>

    <iframe
      src="http://localhost:5011/iframe-webiste2"
      sandbox
    ></iframe>
  </body>
</html>
```

Remove CSP header from index.js of server 2



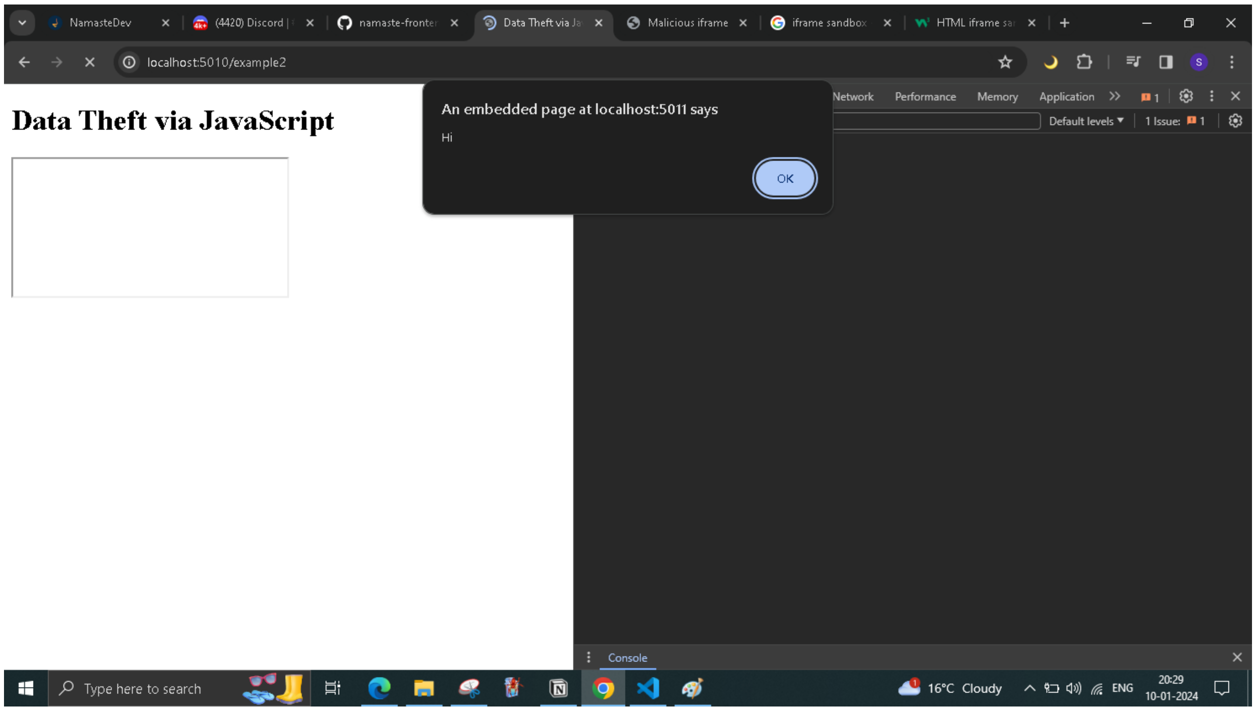
Data Theft via JavaScript

Malicious iframe

This iframe attempts to steal data from the parent window.

```
Blocked script execution in 'http://localhost:5011/iframe-webiste2' because the   iframe-webiste2:1
document's frame is sandboxed and the 'allow-scripts' permission is not set.
GET http://localhost:5010/favicon.ico 404 (Not Found)                              favicon.ico:1
```

```
<iframe
      src="http://localhost:5011/iframe-webiste2"
      sandbox="allow-same-origin allow-scripts allow-modals"
    ></iframe>
```


localhost:5010/example2

Data Theft via JavaScript

An embedded page at localhost:5011 says

Hi

3. If parent tries to steal something from child. You can mention below code in child script

```
<script>
        if (top != self) {
            top.location = self.location;
        }
</script>
```

We have added this in iframe-webiste1 html as we are adding its iFrame in example 1 html. This is prevent example 1 to access anything from iframe-webiste1.

4. Setting extra headers in cookie

httpOnly: true - it means you can only access cookies only on the server. You cannot access cookie using JavaScript(cannot use document.cookie etc)

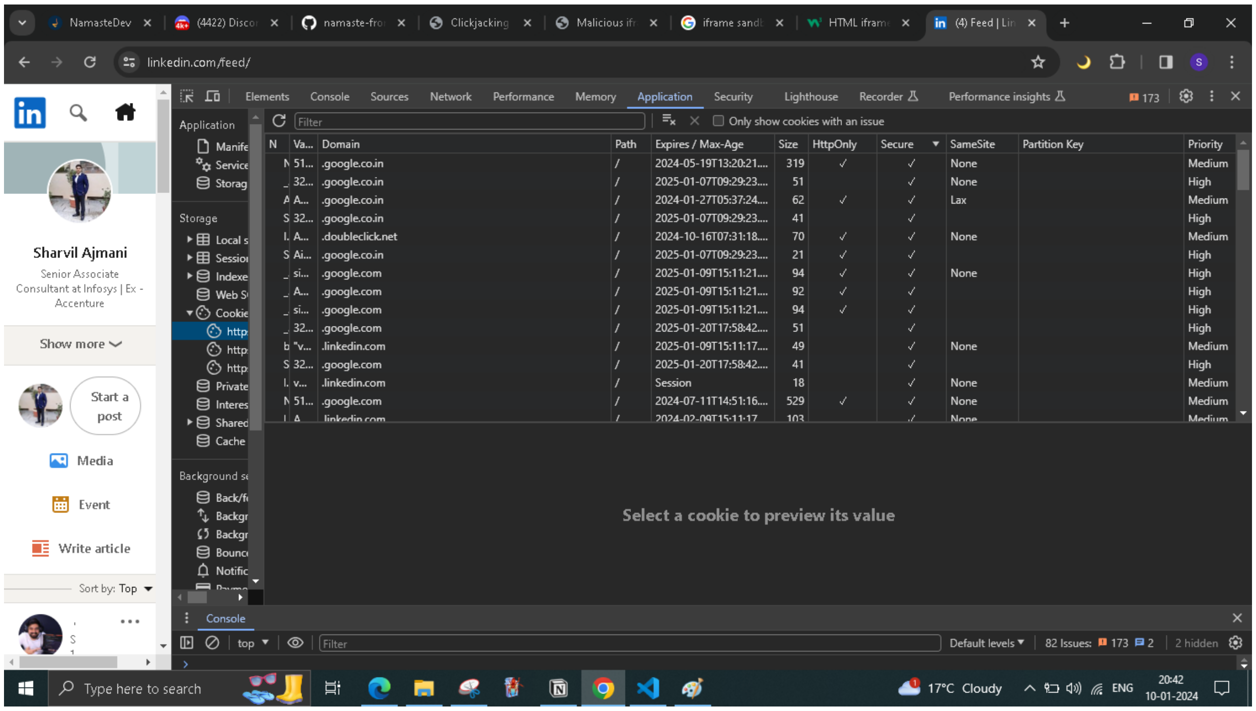
secure: true - cookie will be sent to the client if it is https

sameSite: ‘strict’ - if api’s are being called to some other domain, your cookies will not travel along with them

```
app.use((req, res, next) => {
    res.setHeader('Content-Security-Policy', "frame-ancestors
'self'")

    res.cookie('sessionID', '12345', {
        httpOnly: true,
        secure: true,
        sameSite: 'strict',
      });
    next();
})
```

This code is present in server 2 - index.js

Example - linkedn inspect



linkedin.com/feed/

Select a cookie to preview its value

N	Va...	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Partition Key	Priority
N	51...	.google.co.in	/	2024-05-19T13:20:21....	319	✓	✓	None		Medium
_	32...	.google.co.in	/	2025-01-07T09:29:23....	51		✓	None		High
A	A...	.google.co.in	/	2024-01-27T05:37:24....	62	✓	✓	Lax		Medium
S	32...	.google.co.in	/	2025-01-07T09:29:23....	41		✓			High
I.	A...	.doubleclick.net	/	2024-10-16T07:31:18....	70	✓	✓	None		Medium
S	Ai...	.google.co.in	/	2025-01-07T09:29:23....	21	✓	✓			High
_	si...	.google.com	/	2025-01-09T15:11:21....	94	✓	✓	None		High
_	A...	.google.com	/	2025-01-09T15:11:21....	92	✓	✓			High
_	si...	.google.com	/	2025-01-09T15:11:21....	94	✓	✓			High
_	32...	.google.com	/	2025-01-20T17:58:42....	51		✓			High
b	"v...	.linkedin.com	/	2025-01-09T15:11:17....	49		✓	None		Medium
S	32...	.google.com	/	2025-01-20T17:58:42....	41		✓			High
I.	v...	.linkedin.com	/	Session	18		✓	None		Medium
N	51...	.google.com	/	2024-07-11T14:51:16....	529	✓	✓	None		Medium
I	A	.linkedin.com	/	2024-02-09T15:11:17	103		✓	None		Medium