TryHackMe — JavaScript Essentials | Cyber Security 101 (THM)

Z3pH7 · Follow
12 min read · Nov 1, 2024

Listen Share More

Hey everyone! TryHackMe just announced the **NEW Cyber Security 101** learning path, and there are tons of giveaways this time! This article might help you out, but I've kept the summary short for easy understanding. Enjoy hacking!

Introduction

JavaScript (JS) is a popular scripting language that allows web developers to add interactive features to websites containing HTML and CSS (styling). Once the HTML elements are created, you can add interactiveness like validation, onClick actions, animations, etc, through JS. Learning the language is equally important as that of HTML and CSS. The JS scripts are used primarily with HTML.

This room is designed as an introductory overview of JS, specifically tailored for beginners with limited JS experience. The primary focus is on teaching the fundamentals of JS from a cyber perspective and how hackers utilise legitimate functionalities to achieve malicious results.

Room Prerequisites

- Linux Fundamentals Module
- Web Application Basics
- How Websites Work

Learning Objectives

- Understand the basics of JS
- Integrating JS in HTML
- Abusing Dialogue Function
- Bypassing Control Flow Statements
- Exploring Minified Files

Answer the questions below

> *I have successfully started the attached VM.*

Answer: No answer needed

Essential Concepts

JavaScript (JS) is a core technology of web development, running on the client-side to add interactivity, handle user input, and manipulate the DOM (Document Object

Model). JS is versatile and can be used both as a procedural and object-oriented language. Some essential concepts include:

- **Variables:** Store data values and can be declared using `var`, `let`, or `const`.
- **Data Types:** Include strings, numbers, booleans, objects, and arrays. JavaScript is dynamically typed, meaning the type is determined at runtime.
- **Functions:** Blocks of code defined to perform specific tasks and can be reused with different inputs.

Example Code:

```
let age = 25;
let name = "Alice";
function greet(person) {
    console.log("Hello, " + person + "!");
}
greet(name); // Outputs: "Hello, Alice!"
```

Functions

A function represents a block of code designed to perform a specific task. Inside a function, you group code that needs to perform a similar task. For example, you are developing a web application in which you need to print students' results on the web page. The ideal case would be to create a function `PrintResult(rollNum)` that would accept the roll number of the user as an argument.

```
<script>
        function PrintResult(rollNum) {
            alert("Username with roll number " + rollNum + " has passed the exa
            // any other logic to display the result
        }

for (let i = 0; i < 100; i++) {
            PrintResult(rollNumbers[i]);
        }
    </script>
```

So, instead of writing the same print code for all the students, we will use a simple function to print the result.

Loops

Loops allow you to run a code block multiple times as long as a condition is `true`. Common loops in JS are `for`, `while`, and `do...while`, which are used to repeat tasks, like going through a list of items. For example, if we want to print the results of 100 students, we can call the PrintResult(rollNum) function 100 times by writing it 100 times, or we can create a loop that will be iterated through 1 to 100 and will call the PrintResult(rollNum) function as shown below.

```
// Function to print student result
        function PrintResult(rollNum) {
            alert("Username with roll number " + rollNum + " has passed the exa
            // any other logic to the display result
        }

for (let i = 0; i < 100; i++) {
            PrintResult(rollNumbers[i]); // this will be called 100 times
        }
```

Request-Response Cycle

In web development, the request-response cycle is when a user's browser (the client) sends a request to a web server, and the server responds with the requested information. This could be a webpage, data, or other resources. You can learn more about it here.

Answer the questions below

> *What term allows you to run a code block multiple times as long as it is a condition?*

Answer: Loop

JavaScript Overview

JavaScript is typically executed directly in web browsers, which allows developers to inspect and manipulate code in real-time. Using Google Chrome's **Console** is one way to quickly test and debug JavaScript code:

1. Open Chrome's Developer Tools (`Ctrl + Shift + I` on Windows or `Cmd + Option + I` on Mac).

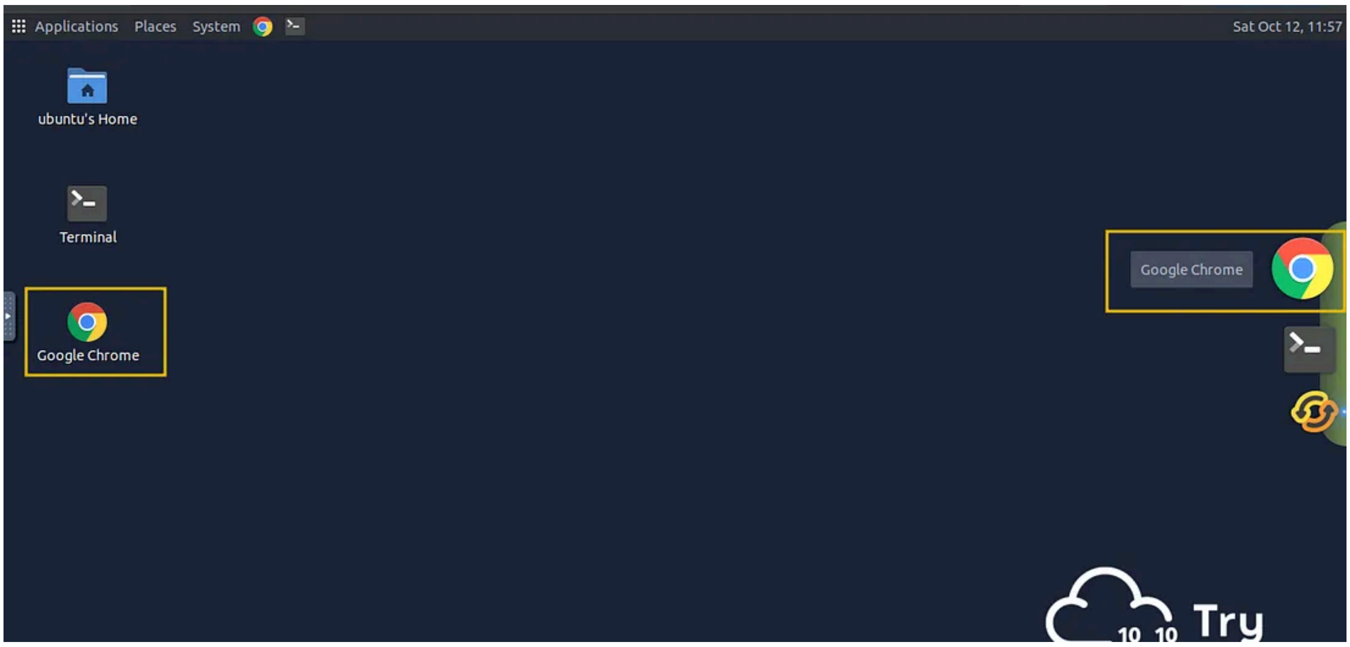
2. Navigate to the **Console** tab to enter JavaScript commands directly.

Example Command:

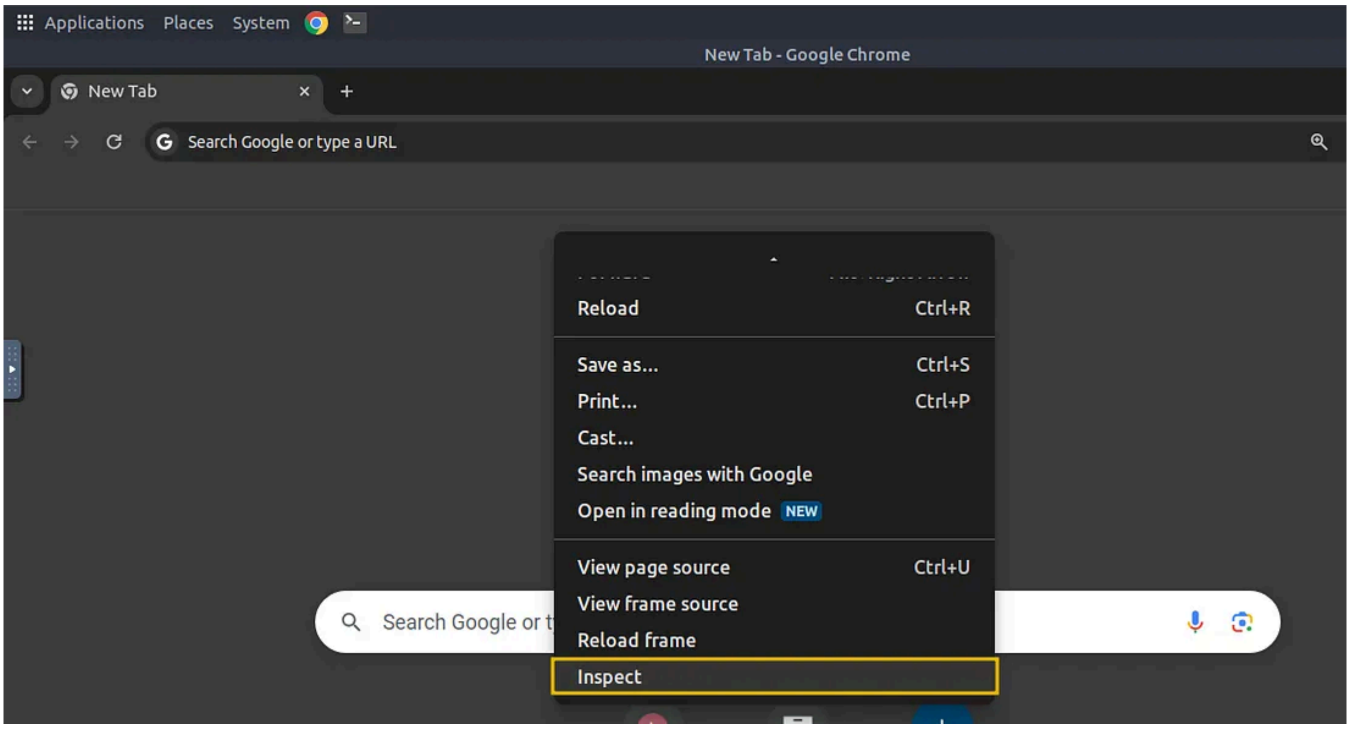
```
console.log("Hello, World!"); // Prints "Hello, World!" to the console
```

JS is primarily executed on the client side, which makes it easy to inspect and interact with HTML directly within the browser. We'll use the `Google Chrome Console` feature to run our first JS program, allowing us to write and execute JS code easily without additional tools. Follow these steps to get started:

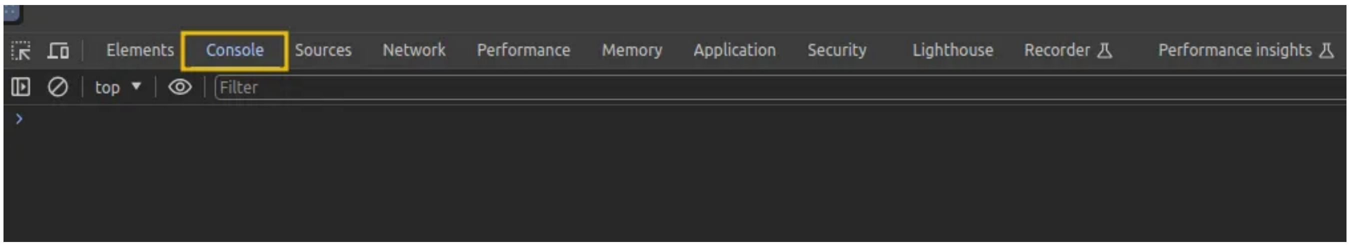
- Open `Google Chrome` by clicking the `Google Chrome` icon on the Desktop of the VM.



- Once Chrome is open, press `Ctrl + Shift + I` to open the `Console` or right-click anywhere on the page and select `Inspect` .



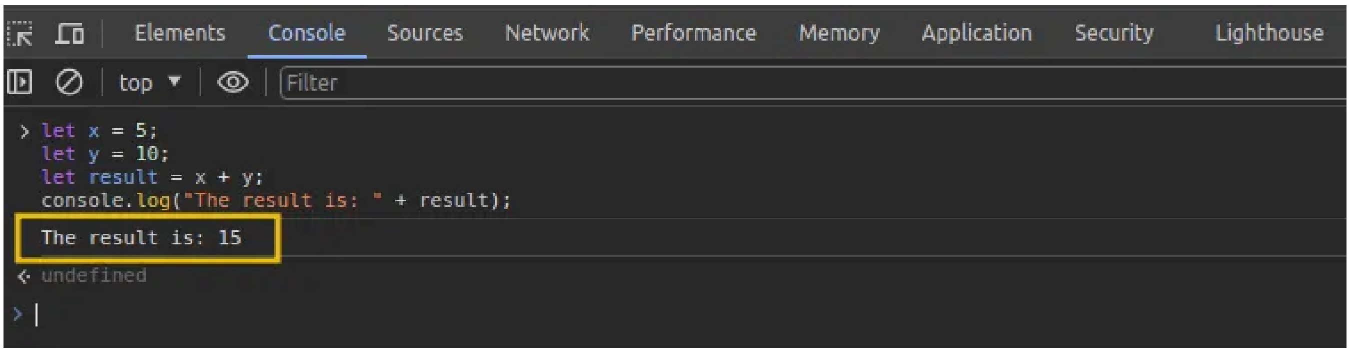
- Then, click on the `Console` tab. This console allows you to run JS code directly in the browser without installing additional software.



- Let's create a simple JS program that adds two numbers and displays the result. Below is the code:

```
let x = 5;
let y = 10;
let result = x + y;
console.log("The result is: " + result);
```

- In the code above, `x` and `y` are variables holding the numbers. `x + y` is an expression that adds the two numbers together, whereas `console.log` is a function used to print the result to the console.
- Copy the above code and paste it into the console by pressing the key `Ctrl + V`. Once pasted, press `Enter`. You should see the result displayed as:



```
> let x = 5;
  let y = 10;
  let result = x + y;
  console.log("The result is: " + result);
  The result is: 15
<· undefined
>
```

Answer the questions below

> *What is the code output if the value of x is changed to 10?*

Answer: The result is: 20

> *Is JavaScript a compiled or interpreted language?*

Answer: Interpreted

Integrating JavaScript in HTML

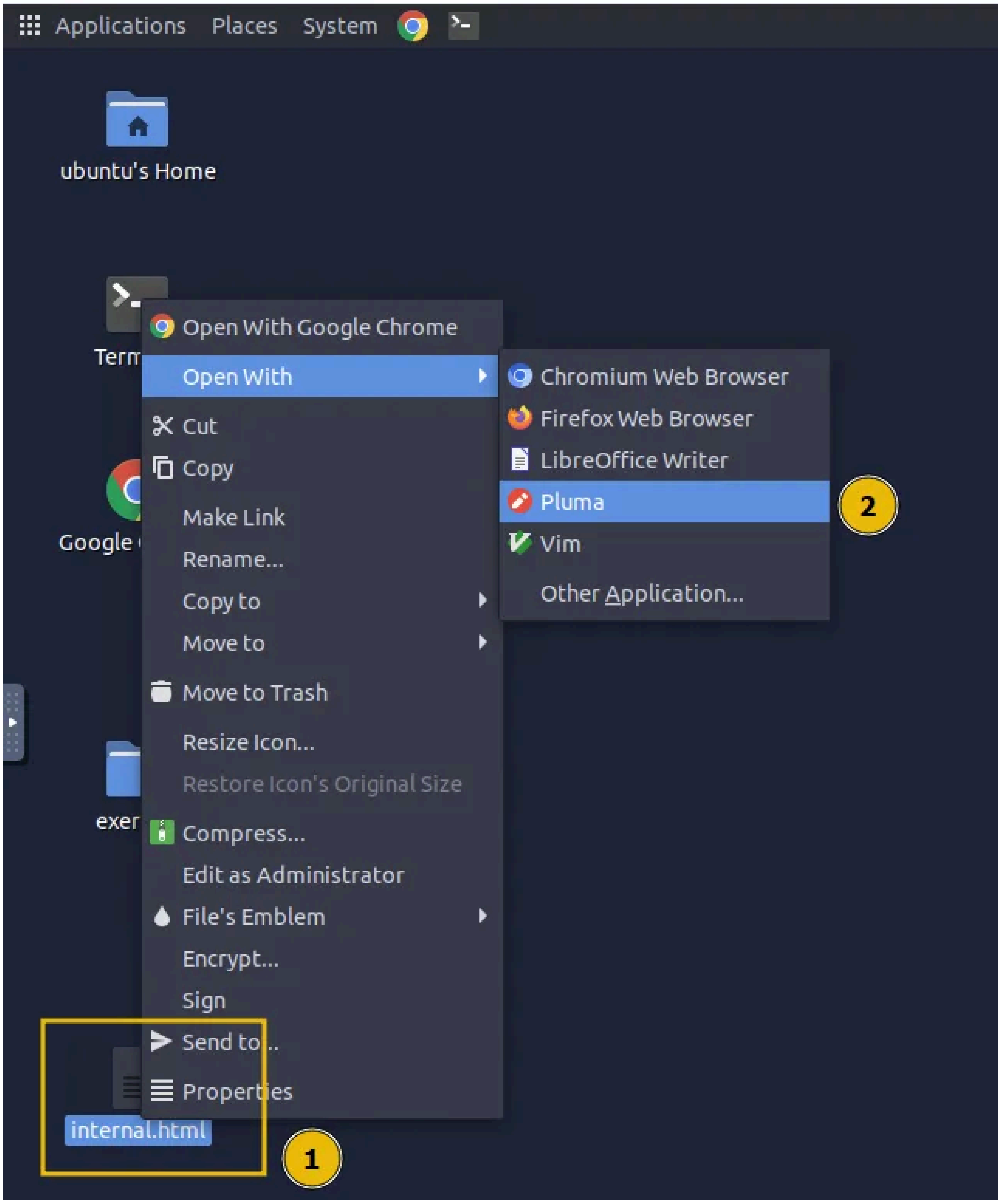
JavaScript can be integrated into HTML to add interactivity directly into web pages, using either:

- **Internal JavaScript**: Placed within `<script>` tags in the HTML file.
- **External JavaScript**: Linked as a separate `.js` file, which improves code organization and maintainability.

Internal JavaScript

Written directly within `<script>` tags in the HTML document.

To create an HTML document with internal JS, right-click on the `Desktop` and select `Create Document > Empty File`. Name the file `internal.html`. Next, right-click the `internal.html` file and `choose Open with Pluma` to open it in a text editor.

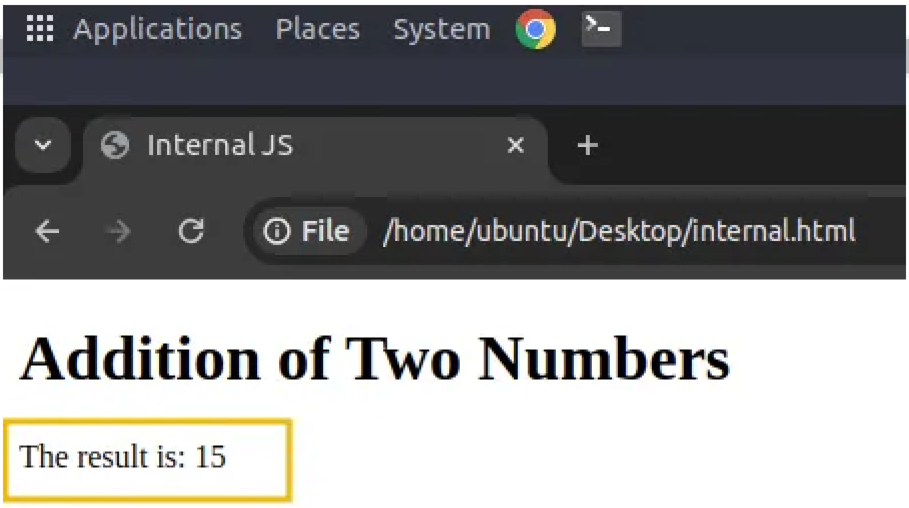


Once the editor is open, paste the following code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Internal JS</title>
</head>
<body>
    <h1>Addition of Two Numbers</h1>
```

```
        <p id="result"></p>
    <script>
            let x = 5;
            let y = 10;
            let result = x + y;
            document.getElementById("result").innerHTML = "The result is: " + resul
        </script>
    </body>
    </html>
```

After pasting the code, click `File` and select `Save`, which will save the file to `internal.html`.Double-click the file to open it in Chrome browser, where you will see the following output:



Addition of Two Numbers

The result is: 15

In this HTML document, we are using internal JS, meaning the code is placed directly inside the HTML file within the **<script>** tag. The script performs a simple task: it adds two numbers (x and y) and then displays the result on the web page. The JS interacts with the HTML by selecting an element (**<p> with id="result"**) and updating its content using `document.getElementById("result").innerHTML`. This internal JS is executed when the browser loads the HTML file.

External JavaScript: Placed in a separate `.js` file and linked in HTML using the `src` attribute in the `<script>` tag.

We will use the same example for external JS but separate the JS code into a different file.

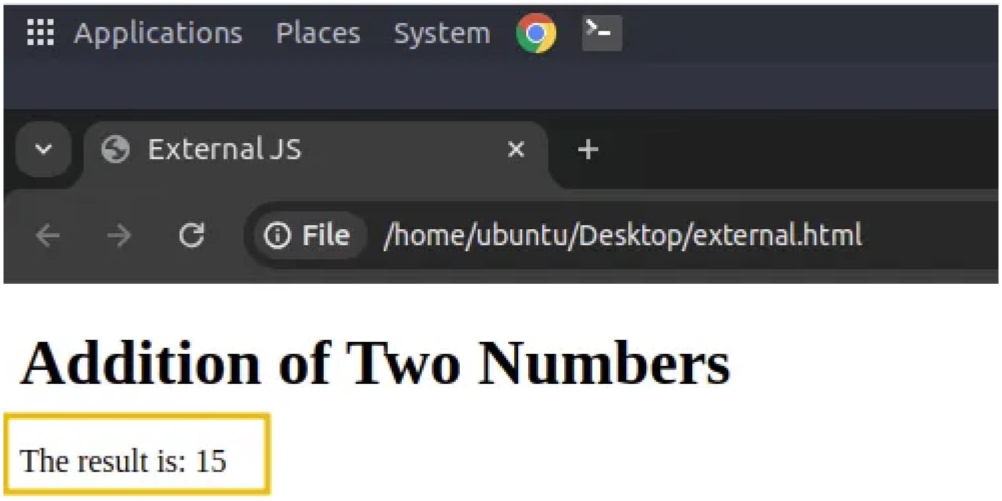
First, create a new file named `script.js` and save it on the `Desktop` with the following code:

```
let x = 5;
let y = 10;
let result = x + y;
document.getElementById("result").innerHTML = "The result is: " + result;
```

Next, create a new file named `external.html` and paste the following code (notice that the HTML code is the same as that of the previous example):

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>External JS</title>
</head>
<body>
    <h1>Addition of Two Numbers</h1>
    <p id="result"></p>
<!-- Link to the external JS file -->
    <script src="script.js"></script>
</body>
</html>
```

Now, double-click the external.html file and check the results. Do you see any difference? No, the output remains the same as in the previous example.



Addition of Two Numbers

The result is: 15

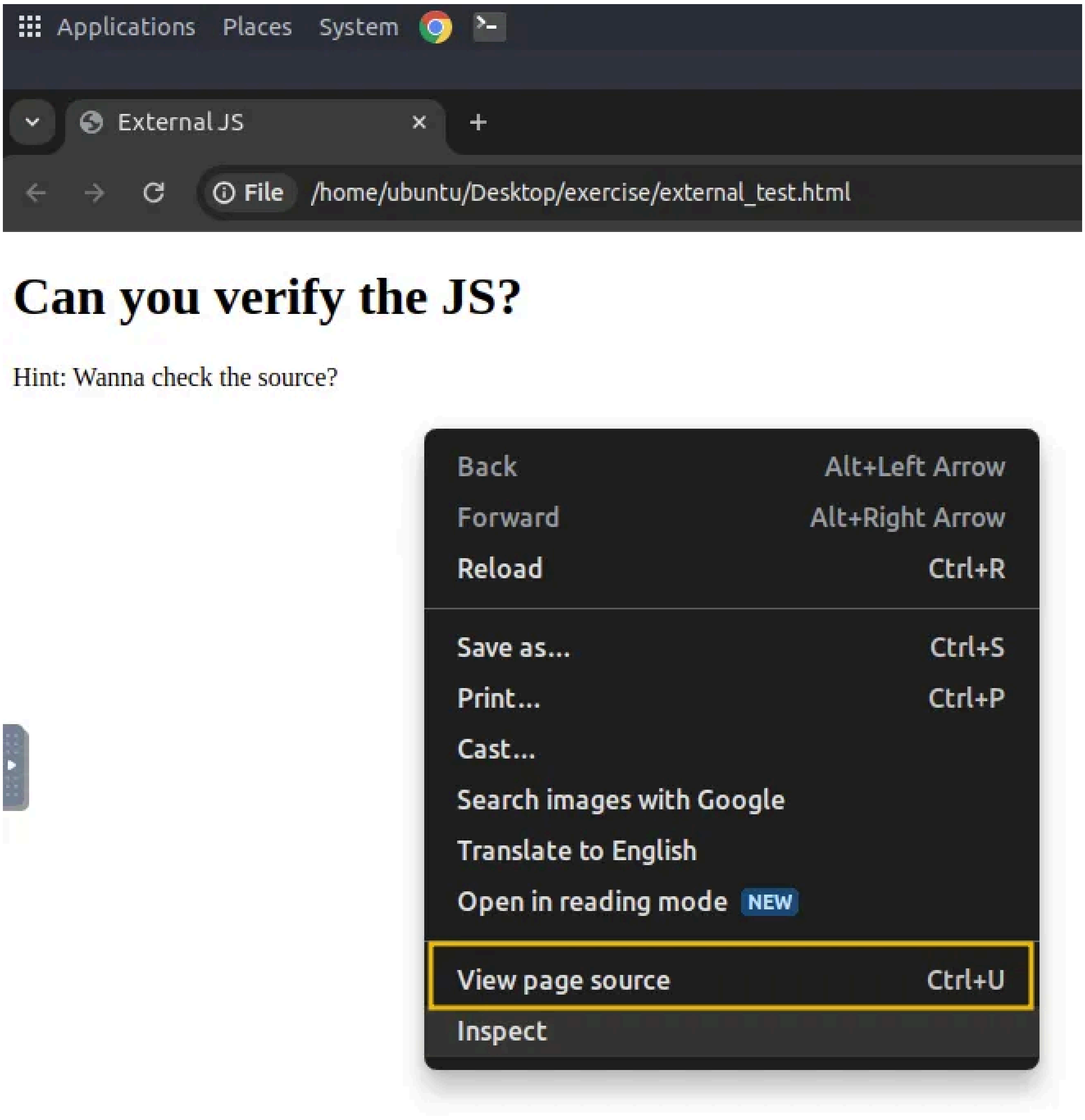
What we did differently is use the `src` attribute in the **<script>** tag to load the JS from an external file. When the browser loads the page, it looks for the `script.js` file and loads its content into the HTML document. This approach allows us to keep the JS code separate from the HTML, making the code more organised and easier to maintain, especially when working on larger projects.

Example Internal JavaScript:

```
<!DOCTYPE html>
<html>
<body>
    <h1>Result of Addition</h1>
    <p id="result"></p>
    <script>
        let x = 5, y = 10;
        document.getElementById("result").innerHTML = "The result is: " + (x +
    </script>
</body>
</html>
```

Verifying Internal or External JS

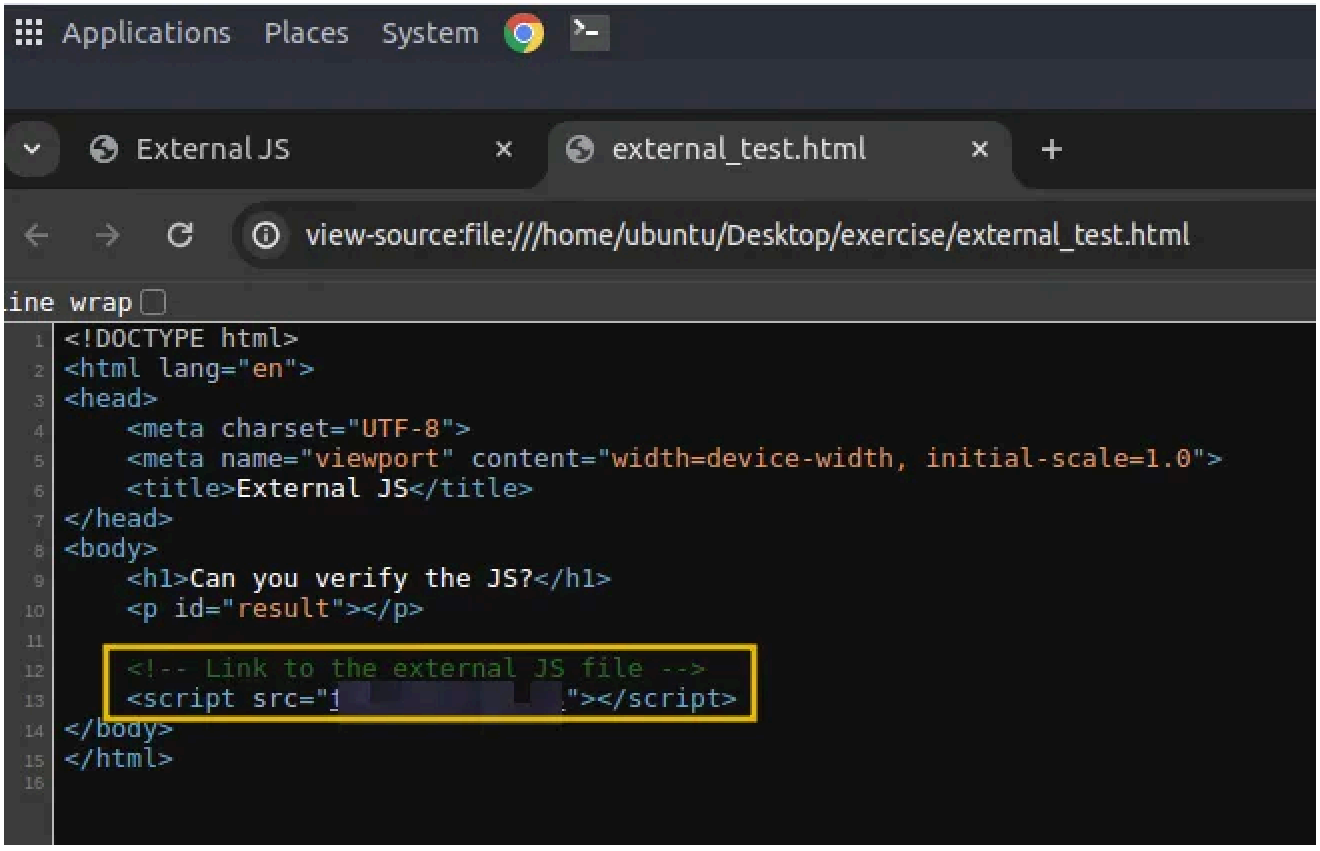
When pen-testing a web application, it is important to check whether the website uses internal or external JS. This can be easily verified by viewing the page's source code. To do this, open the page `external_test.html` located in the `exercise` folder in `Chrome`, right-click anywhere on the page, and select `View Page Source`.



Can you verify the JS?

Hint: Wanna check the source?

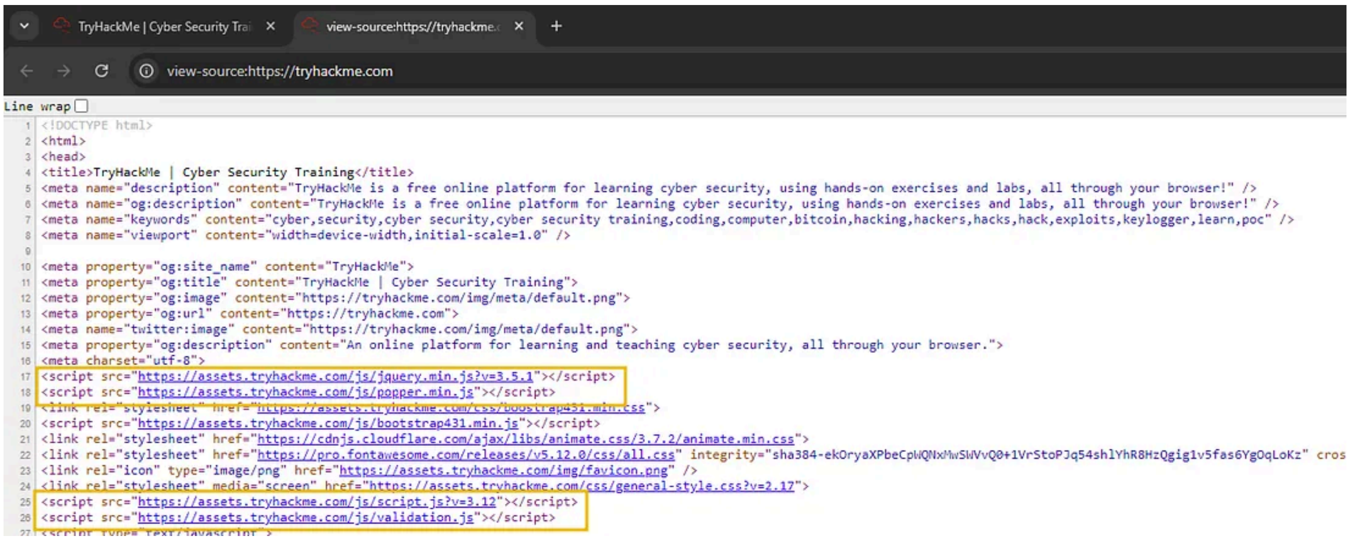
This will display the HTML code of the rendered page. Inside the source code, any JS written directly on the page will appear between **<script>** tags without the `src` attribute. If you see a **<script>** tag with a **src** attribute, it indicates that the page is loading external JS from a separate file.


Applications
Places
System
External JS
external_test.html
view-source:file:///home/ubuntu/Desktop/exercise/external_test.html
Line wrap

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>External JS</title>
</head>
<body>
    <h1>Can you verify the JS?</h1>
    <p id="result"></p>

    <!-- Link to the external JS file -->
    <script src="j[redacted]"></script>
</body>
</html>
```

For a practical example, visit https://tryhackme.com in your browser and inspect the source code to identify how the website loads the JS internally and from external sources.


TryHackMe | Cyber Security Trai
view-source:https://tryhackme.c
view-source:https://tryhackme.com
Line wrap

```
<!DOCTYPE html>
<html>
<head>
<title>TryHackMe | Cyber Security Training</title>
<meta name="description" content="TryHackMe is a free online platform for learning cyber security, using hands-on exercises and labs, all through your browser!" />
<meta name="og:description" content="TryHackMe is a free online platform for learning cyber security, using hands-on exercises and labs, all through your browser!" />
<meta name="keywords" content="cyber,security,cyber security,cyber security training,coding,computer,bitcoin,hacking,hackers,hacks,hack,exploits,keylogger,learn,poc" />
<meta name="viewport" content="width=device-width,initial-scale=1.0" />

<meta property="og:site_name" content="TryHackMe">
<meta property="og:title" content="TryHackMe | Cyber Security Training">
<meta property="og:image" content="https://tryhackme.com/img/meta/default.png">
<meta property="og:url" content="https://tryhackme.com">
<meta name="twitter:image" content="https://tryhackme.com/img/meta/default.png">
<meta property="og:description" content="An online platform for learning and teaching cyber security, all through your browser.">
<meta charset="utf-8">
<script src="https://assets.tryhackme.com/js/jquery.min.js?v=3.5.1"></script>
<script src="https://assets.tryhackme.com/js/popper.min.js"></script>
<link rel="stylesheet" href="https://assets.tryhackme.com/css/bootstrap431.min.css">
<script src="https://assets.tryhackme.com/js/bootstrap431.min.js"></script>
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/animate.css/3.7.2/animate.min.css">
<link rel="stylesheet" href="https://pro.fontawesome.com/releases/v5.12.0/css/all.css" integrity="sha384-ekOryaXPbeCpWQNxMwSWVvQ0+1VrStoPJq54shlYhR8HzQgig1v5fas6YgOqLoKz" cros
<link rel="icon" type="image/png" href="https://assets.tryhackme.com/img/favicon.png" />
<link rel="stylesheet" media="screen" href="https://assets.tryhackme.com/css/general-style.css?v=2.17">
<script src="https://assets.tryhackme.com/js/script.js?v=3.12"></script>
<script src="https://assets.tryhackme.com/js/validation.js"></script>
<script type="text/javascript">
```

Answer the questions below

> *Which type of JavaScript integration places the code directly within the HTML document?*

Answer: Internal

> *Which method is better for reusing JS across multiple web pages?*

Answer: External

> *What is the name of the external JS file that is being called by* ***external_test.html?***

Answer: thm_external.js

> *What attribute links an external JS file in the <script> tag?*

Answer: src

Abusing Dialogue Functions

JavaScript provides functions like `alert`, `prompt`, and `confirm` for displaying messages and gathering input from users. These functions are sometimes exploited in phishing or social engineering tactics, making users interact with untrusted dialogs.

- `alert("Hello!");` : Opens an alert dialog with the text "Hello!".
- `prompt("Enter your name:");` : Prompts the user to enter information.
- `confirm("Are you sure?");` : Displays a dialog with "OK" and "Cancel" options, often used for confirmations.

Command Examples:

Alert:

```
alert("Hello, THM"); // Displays an alert box with the message
```

Prompt:

```
let name = prompt("Enter your name:");
alert("Hello " + name); // Displays a personalized greeting
```

Confirm:

```
let isSure = confirm("Proceed?");
console.log(isSure); // Logs true if OK, false if Cancel
```

Example Use of Dialogue Functions:

```
alert("Warning: You are entering a secure area."); // Simple alert box
let userName = prompt("Please enter your name:");
if (userName) {
    alert("Welcome, " + userName);
}
let confirmAction = confirm("Do you agree to the terms?");
if (confirmAction) {
    alert("Thank you for agreeing.");
} else {
    alert("You did not agree.");
}
```

Answer the questions below

> *In the file **invoice.html**, how many times does the code show the alert Hacked?*

Answer: 5

> *Which of the JS interactive elements should be used to display a dialogue box that asks the user for input?*

Answer: prompt

> *If the user enters Tesla, what value is stored in the carName= prompt("What is your car name?")? in the carName variable?*

Answer: Tesla

Bypassing Control Flow Statements

Control flow in JavaScript can be manipulated using statements like `if-else`, `switch`, and various types of loops (`for`, `while`). By controlling the flow, developers can create conditional behaviors based on user input or other conditions.

If-Else Statement:

```
let age = prompt("Enter your age:");
if (age >= 18) {
    console.log("Adult");
} else {
    console.log("Minor");
}
```

For Loop:

```
for (let i = 0; i < 5; i++) {
    console.log("Count: " + i); // Prints "Count: 0" to "Count: 4"
}
```

Switch Statement Example:

```
let day = prompt("Enter a day (1-7):");
switch(day) {
    case '1':
        alert("It's Monday!");
        break;
    case '2':
        alert("It's Tuesday!");
        break;
    // Additional cases can go here
    default:
        alert("Not a valid day.");
}
```

Answer the questions below

> *What is the message displayed if you enter the age less than 18?*

Answer: You are a minor.

> *What is the password for the user admin?*

Answer: ComplexPassword

Exploring Minified Files

JavaScript code is often **minified** to reduce file size by removing all unnecessary characters (e.g., spaces and comments) without affecting functionality. Minification makes files load faster but harder to read. **Obfuscation** goes further, making code very difficult to interpret, often used in production environments to protect intellectual property or deter code manipulation.

Practical Example

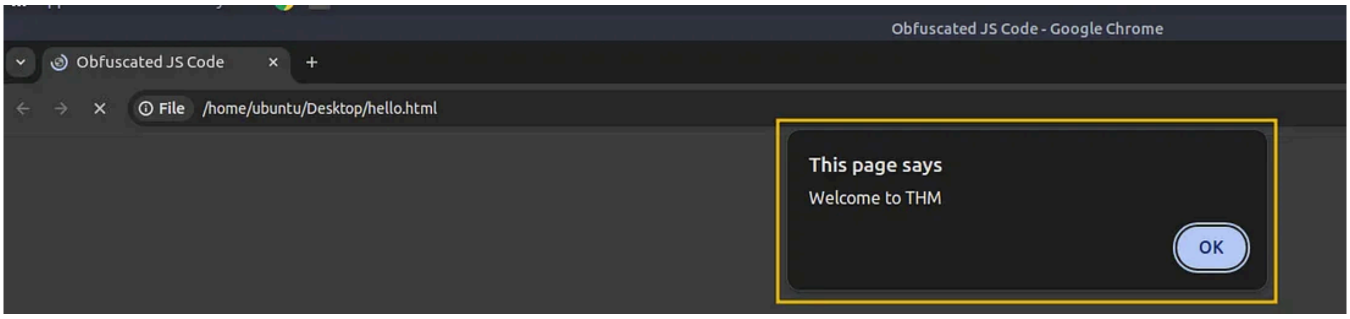
Create a file on the Desktop of the attached VM with the name `hello.html` and paste the following HTML code:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Obfuscated JS Code</title>
</head>
<body>
    <h1>Obfuscated JS Code</h1>
    <script src="hello.js"></script>
</body>
</html>
```

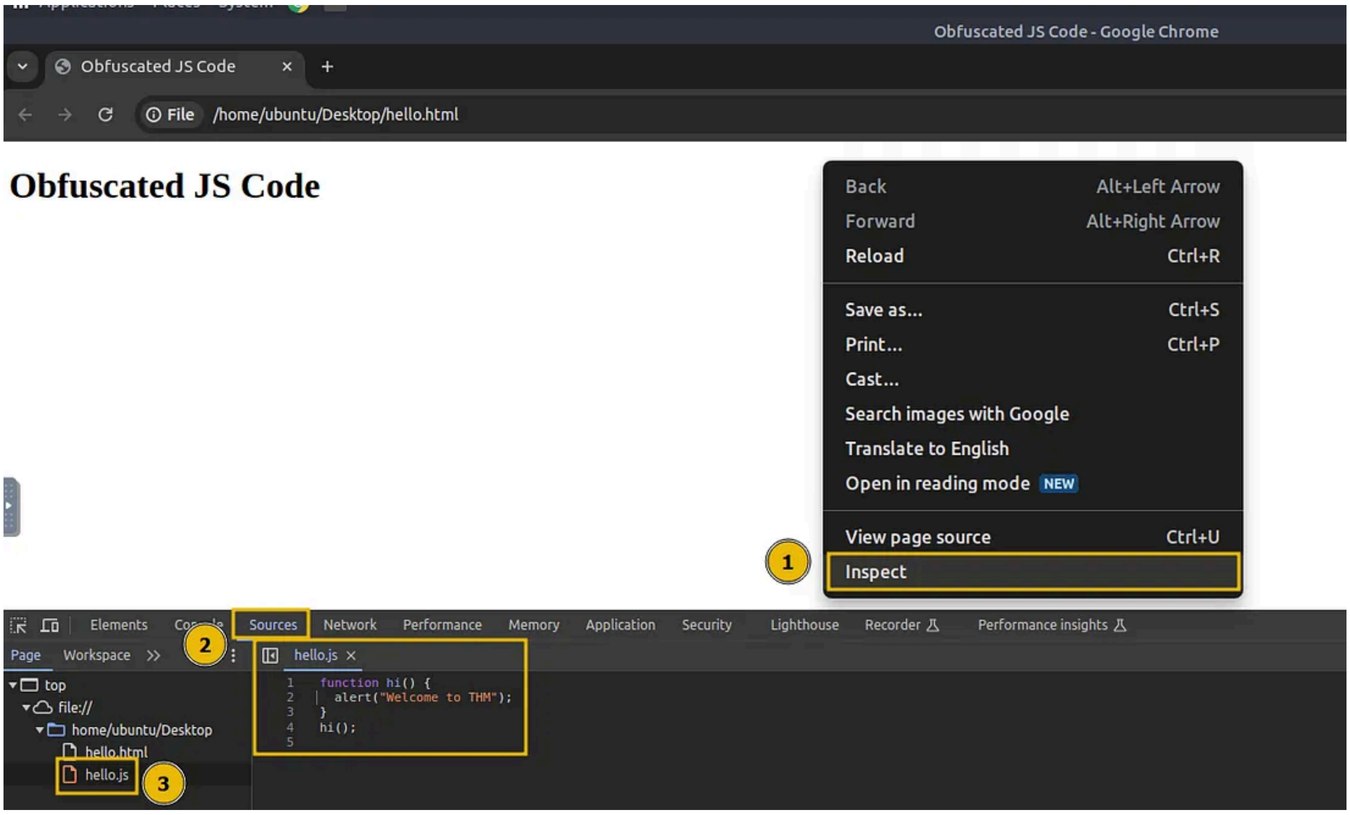
Then, create another file with the name `hello.js` and add the following code:

```javascript
function hi() {
  alert("Welcome to THM");
}
hi();
```

Now, double-click the `hello.html` file to open it in Google Chrome. Once the file is opened, you will see an alert greeting you with "`Welcome to THM`".


Obfuscated JS Code - Google Chrome
Obfuscated JS Code
File /home/ubuntu/Desktop/hello.html
This page says
Welcome to THM
OK

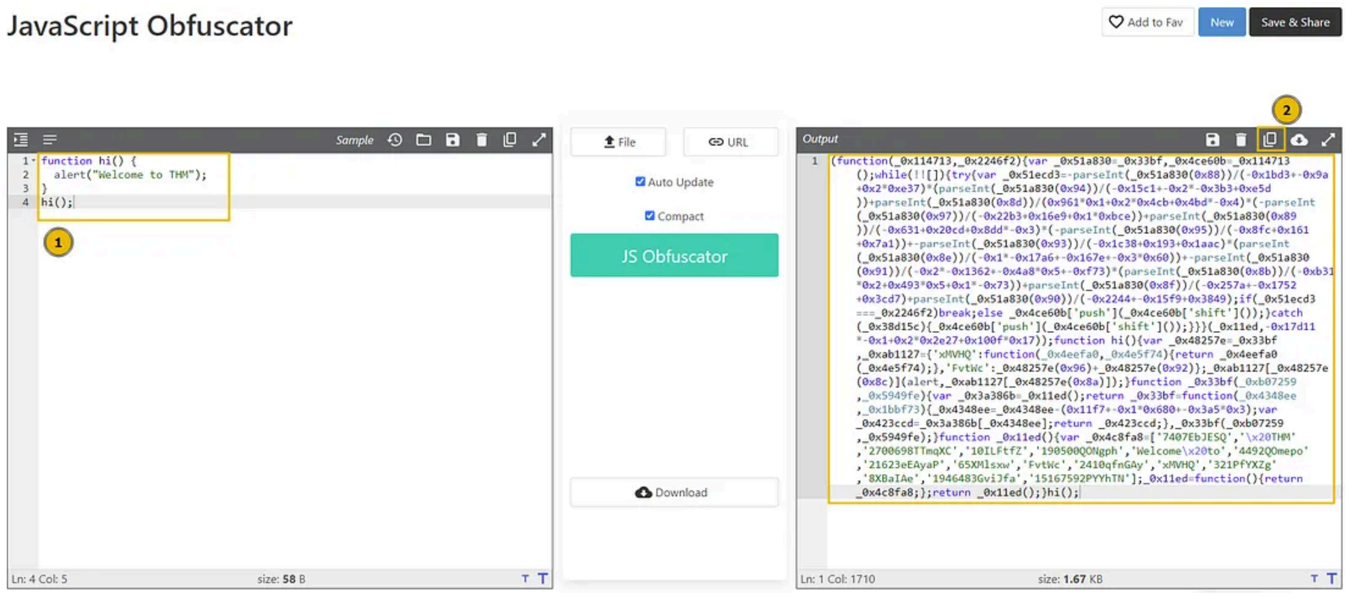
Click `OK` to close the alert dialogue box. Right-click anywhere on the page and click on `Inspect` to open the developer tools. In the developer tools, navigate to the `Sources` tab and click on the `hello.js` file to view the source code. You will see that the JS code is easily accessible and viewable, as shown below:



```
function hi() {
  alert("Welcome to THM");
}
hi();
```

Obfuscation in Action

Now, we will try to minify and obfuscate the JS code using an online tool. Visit the website and copy the contents of `hello.js`, and paste them into the dialogue box on the website. The tool will minify and obfuscate the code, turning it into a string of gibberish characters shown below:



```
function hi() {
  alert("Welcome to THM");
}
hi();
```

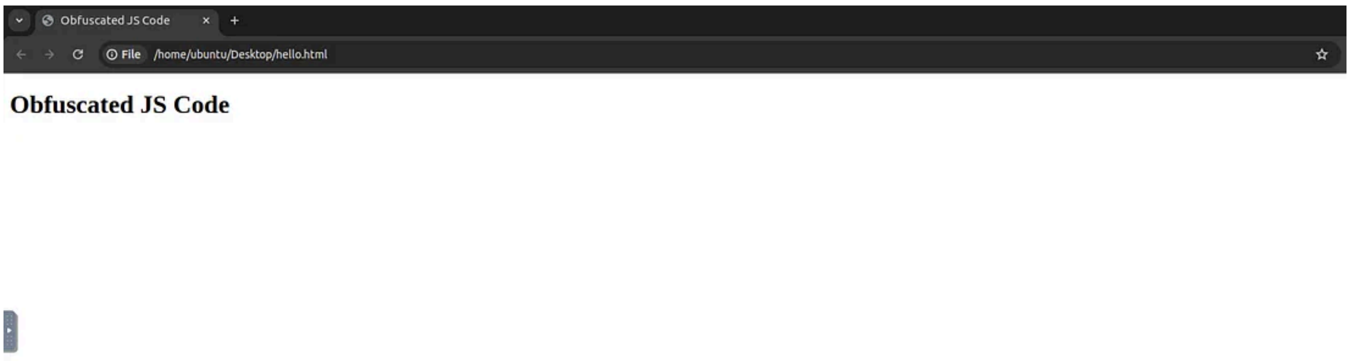
```
(function(_0x114713,_0x2246f2){var _0x51a830=_0x33bf,_0x4ce60b=_0x114713
    ();while(!![]){try{var _0x51ecd3=-parseInt(_0x51a830(0x88))/(-0x1bd3+-0x9a
    +0x2*0xe37)*(parseInt(_0x51a830(0x94))/(-0x15c1+-0x2*-0x3b3+0xe5d
    ))+parseInt(_0x51a830(0x8d))/(0x961*0x1+0x2*0x4cb+0x4bd*-0x4)*(-parseInt
    (_0x51a830(0x97))/(-0x22b3+0x16e9+0x1*0xbce))+parseInt(_0x51a830(0x89
    ))/(-0x631+0x20cd+0x8dd*-0x3)*(-parseInt(_0x51a830(0x95))/(-0x8fc+0x161
    +0x7a1))+-parseInt(_0x51a830(0x93))/(-0x1c38+0x193+0x1aac)*(parseInt
    (_0x51a830(0x8e))/(-0x1*-0x17a6+-0x167e+-0x3*0x60))+-parseInt(_0x51a830
    (0x91))/(-0x2*-0x1362+-0x4a8*0x5+-0xf73)*(parseInt(_0x51a830(0x8b))/(-0xb31
    *0x2+0x493*0x5+0x1*-0x73))+parseInt(_0x51a830(0x8f))/(-0x257a+-0x1752
    +0x3cd7)+parseInt(_0x51a830(0x90))/(-0x2244+-0x15f9+0x3849);if(_0x51ecd3
    ===_0x2246f2)break;else _0x4ce60b['push'](_0x4ce60b['shift']());}catch
    (_0x38d15c){_0x4ce60b['push'](_0x4ce60b['shift']());}}}(_0x11ed,-0x17d11
    *-0x1+0x2*0x2e27+0x100f*0x17));function hi(){var _0x48257e=_0x33bf
    ,_0xab1127={'xMVHQ':function(_0x4eefa0,_0x4e5f74){return _0x4eefa0
    (_0x4e5f74);},'FvtWc':_0x48257e(0x96)+_0x48257e(0x92)};_0xab1127[_0x48257e
    (0x8c)](alert,_0xab1127[_0x48257e(0x8a)]);}function _0x33bf(_0xb07259
    ,_0x5949fe){var _0x3a386b=_0x11ed();return _0x33bf=function(_0x4348ee
    ,_0x1bbf73){_0x4348ee=_0x4348ee-(0x11f7+-0x1*0x680+-0x3a5*0x3);var
    _0x423ccd=_0x3a386b[_0x4348ee];return _0x423ccd;},_0x33bf(_0xb07259
    ,_0x5949fe);}function _0x11ed(){var _0x4c8fa8=['7407EbJESQ','\x20THM'
    ,'2700698TTmqXC','10ILFtfZ','190500QONgph','Welcome\x20to','4492QOmepo'
    ,'21623eEAyaP','65XMlsxw','FvtWc','2410qfnGAy','xMVHQ','321PfYXZg'
    ,'8XBaIAe','1946483GviJfa','15167592PYYhTN'];_0x11ed=function(){return
    _0x4c8fa8;};return _0x11ed();}hi();
```

But what if we tell you that these gibberish characters are still fully functional code? The only difference is that they are not human-readable, but the browser can still execute them correctly. The website converted our JS code to this one:

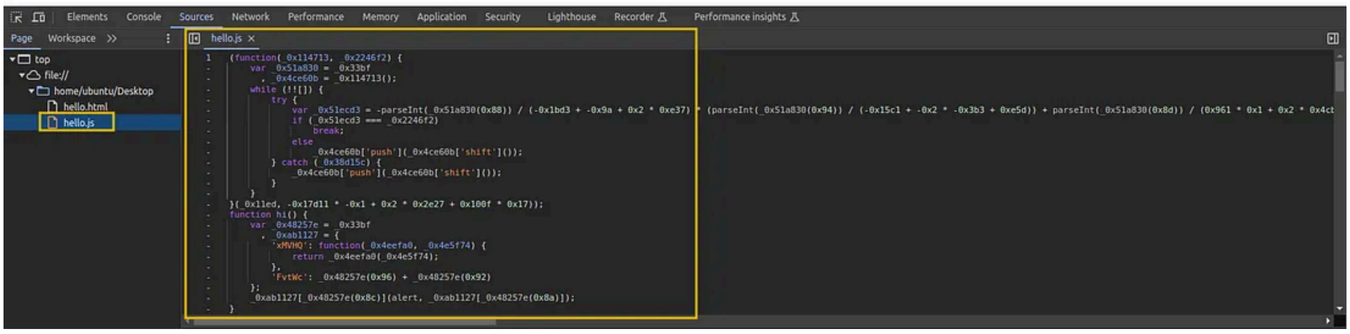
```
(function(_0x114713,_0x2246f2){var _0x51a830=_0x33bf,_0x4ce60b=_0x114713();whil
{var _0x51ecd3=-parseInt(_0x51a830(0x88))/(-0x1bd3+-0x9a+0x2*0xe37)*(parseInt(_
(-0x15c1+-0x2*-0x3b3+0xe5d))+parseInt(_0x51a830(0x8d))/(0x961*0x1+0x2*0x4cb+0x4
(-parseInt(_0x51a830(0x97))/(-0x22b3+0x16e9+0x1*0xbce))+parseInt(_0x51a830(0x89
(-0x631+0x20cd+0x8dd*-0x3)*(-parseInt(_0x51a830(0x95))/(-0x8fc+0x161+0x7a1))+-
parseInt(_0x51a830(0x93))/(-0x1c38+0x193+0x1aac)*(parseInt(_0x51a830(0x8e))/
(-0x1*-0x17a6+-0x167e+-0x3*0x60))+-parseInt(_0x51a830(0x91))/(-0x2*-0x1362+-0x4
(parseInt(_0x51a830(0x8b))/(-0xb31*0x2+0x493*0x5+0x1*-0x73))+parseInt(_0x51a830
(-0x257a+-0x1752+0x3cd7)+parseInt(_0x51a830(0x90))/(-0x2244+-0x15f9+0x3849);if(
===_0x2246f2)break;else _0x4ce60b['push'](_0x4ce60b['shift']());}catch(_0x38d15
{_0x4ce60b['push'](_0x4ce60b['shift']());}}}(_0x11ed,-0x17d11*-0x1+0x2*0x2e27+0
function hi(){var _0x48257e=_0x33bf,_0xab1127={'xMVHQ':function(_0x4eefa0,_0x4e
{return _0x4eefa0(_0x4e5f74);},'FvtWc':_0x48257e(0x96)+_0x48257e(0x92)};_0xab11
](alert,_0xab1127[_0x48257e(0x8a)]);}function _0x33bf(_0xb07259,_0x5949fe){var
=_0x11ed();return _0x33bf=function(_0x4348ee,_0x1bbf73){_0x4348ee=_0x4348ee-(0x
0x1*0x680+-0x3a5*0x3);var _0x423ccd=_0x3a386b[_0x4348ee];return _0x423ccd;},_0x
(_0xb07259,_0x5949fe);}function _0x11ed(){var _0x4c8fa8=['7407EbJESQ','\x20THM'
'2700698TTmqXC','10ILFtfZ','190500QONgph','Welcome\x20to',
'4492QOmepo','21623eEAyaP','65XMlsxw','FvtWc','2410qfnGAy','xMVHQ','321PfYXZg',
'8XBaIAe','1946483GviJfa','15167592PYYhTN'];_0x11ed=function(){return _0x4c8fa8
```

Click the `Copy to Clipboard` (highlighted as **2** in the above image) button as shown on the website. Then, remove the current content of `hello.js` on the attached VM and paste the obfuscated content into the file.

Reload the `hello.html` file in Google Chrome and inspect the source code again under the `Sources` tab. You will notice that the code is now obfuscated but still functions exactly the same as before.



Obfuscated JS Code



```
(function(_0x114713, _0x2246f2) {
    var _0x51a830 = _0x33bf
      , _0x4ce60b = _0x114713();
    while (!![]) {
        try {
            var _0x51ecd3 = -parseInt(_0x51a830(0x88)) / (-0x1bd3 + -0x9a + 0x2 * 0xe37) * (parseInt(_0x51a830(0x94)) / (-0x15c1 + -0x2 * -0x3b3 + 0xe5d)) + parseInt(_0x51a830(0x8d)) / (0x961 * 0x1 + 0x2 * 0x4c
            if (_0x51ecd3 === _0x2246f2)
                break;
            else
                _0x4ce60b['push'](_0x4ce60b['shift']());
        } catch (_0x38d15c) {
            _0x4ce60b['push'](_0x4ce60b['shift']());
        }
    }
}(_0x11ed, -0x17d11 * -0x1 + 0x2 * 0x2e27 + 0x100f * 0x17));
function hi() {
    var _0x48257e = _0x33bf
      , _0xab1127 = {
        'xMVHQ': function(_0x4eefa0, _0x4e5f74) {
            return _0x4eefa0(_0x4e5f74);
        },
        'FvtWc': _0x48257e(0x96) + _0x48257e(0x92)
    };
    _0xab1127[_0x48257e(0x8c)](alert, _0xab1127[_0x48257e(0x8a)]);
}
```

Deobfuscation Tools: Tools like JS Beautifier can help deobfuscate code for analysis.

Answer the questions below

> *What is the alert message shown after running the file **hello.html**?*

Answer: Welcome to THM

> *What is the value of the **age** variable in the following obfuscated code snippet?*
>
> *age=0x1*0x247e+0x35*-0x2e+-0x1ae3;*

Answer: 21

Best Practices

Following best practices in JavaScript can prevent common security vulnerabilities and improve code quality:

- **Server-Side Validation:** Avoid relying only on client-side validation to prevent data tampering.
- **Sanitize User Inputs:** This is essential to avoid XSS (Cross-Site Scripting) attacks.

- **Use Secure External Libraries:** Only include trusted and regularly updated libraries.
- **Minification & Obfuscation:** For production, minify and obfuscate code to enhance performance and security.
- **Do Not Hardcode Sensitive Data:** Keep API keys, secrets, or other sensitive data out of public code.

Example of Avoiding Sensitive Data Hardcoding:

```
// Instead of hardcoding API keys in JavaScript:
const API_KEY = "your-api-key"; // Avoid this practice
// Consider storing it securely on the server side
```

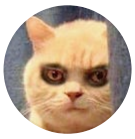
Answer the questions below

> *Is it a good practice to blindly include JS in your code from any source (yea/nay)?*

Answer: nay

Thank You!

Tryhackme Tryhackme Walkthrough Cyber Security 101 JavaScript



Follow

Written by Z3pH7

234 Followers · 7 Following

Cybersecurity | Pentester | Student

Responses (1)

What are your thoughts?

Respond

hi
2 months ago

```
3
```

5*

1 1 reply Reply

More from Z3pH7

Z3pH7

TryHackMe—Shells Overview | Cyber Security 101 (THM)

Hey everyone! TryHackMe just announced the NEW Cyber Security 101 learning path, and there are tons of giveaways this time! This article...

Nov 1, 2024 52

Z3pH7

How to Install and Uninstall OpenVAS completely—Kali linux

OpenVAS, an application used to scan endpoints and web applications to identify and detect vulnerabilities. It is commonly used by...

Sep 19, 2024 3

Z3pH7

TryHackMe — Basic Pentesting | Write-up (THM)

Hello, everyone! This CTF is an entry-level path toward becoming a penetration tester, taking your first step. This challenge is very easy…

Aug 26, 2024

Z3pH7

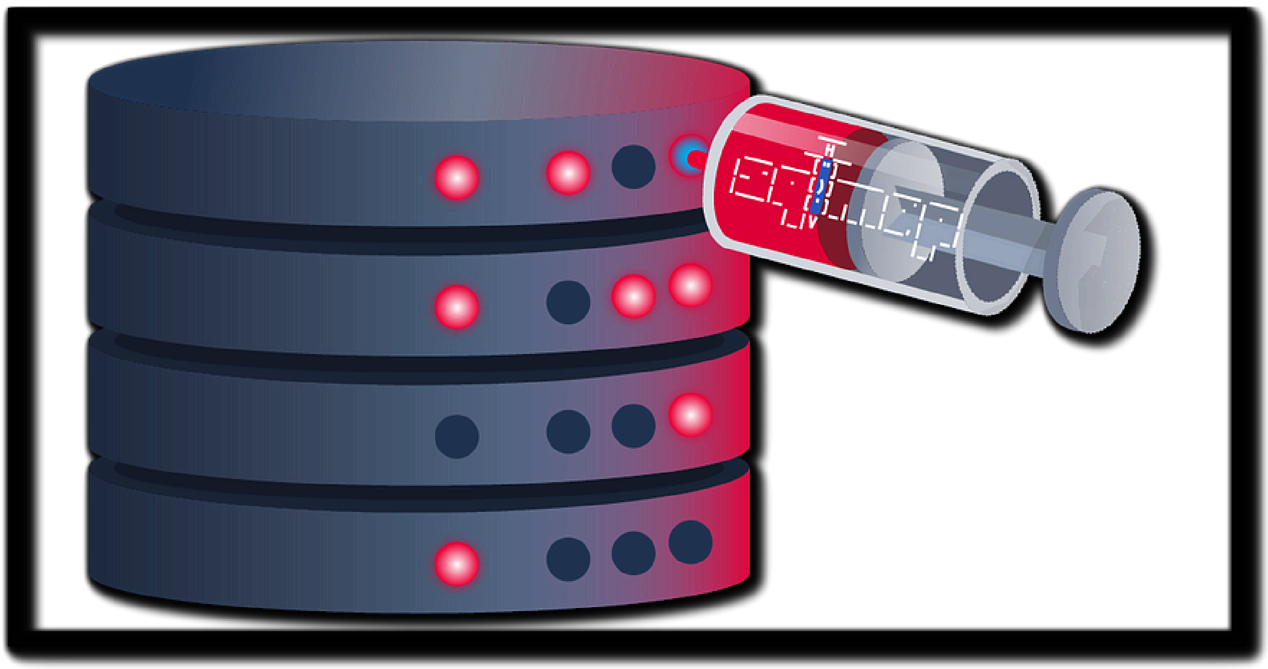
TryHackMe — Digital Forensics Fundamentals | Cyber Security 101 (THM)

Hey everyone! TryHackMe just announced the NEW Cyber Security 101 learning path, and there are tons of giveaways this time! This article…

Nov 1, 2024 10 1

See all from Z3pH7

Recommended from Medium

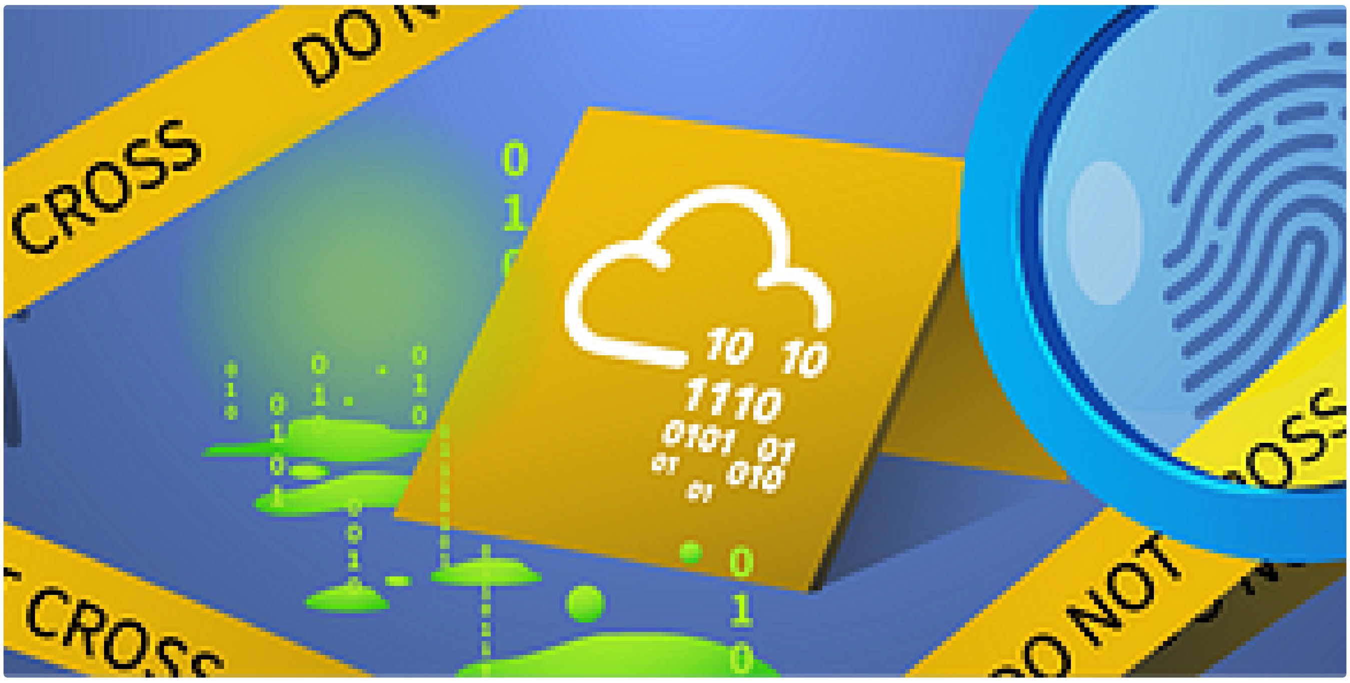


Sunny Singh Verma [SuNnY]

SQLMap: The Basics [Cyber Security 101] TryHackMe Writeup | Detailed Walkthrough | THM Premium...

Kudos To the Creators of this Room 😎

Oct 23, 2024 172 2



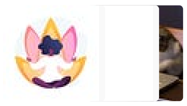
In T3CH by Axoloth

TryHackMe | SOC Fundamentals | WriteUp

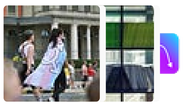
Learn about the SOC team and their processes

Oct 25, 2024 51

Lists


Stories to Help You Grow as a Software Developer
19 stories · 1537 saves

General Coding Knowledge
20 stories · 1841 saves


Generative AI Recommended Reading
52 stories · 1580 saves

Visual Storytellers Playlist
61 stories · 574 saves

Options
Room completed (100%)
Task 1 Introduction
Task 2 Accessing the Tool
Task 3 Navigating the Interface

Jawstar

CyberChef: The Basics Tryhackme Write up

Tryhackme

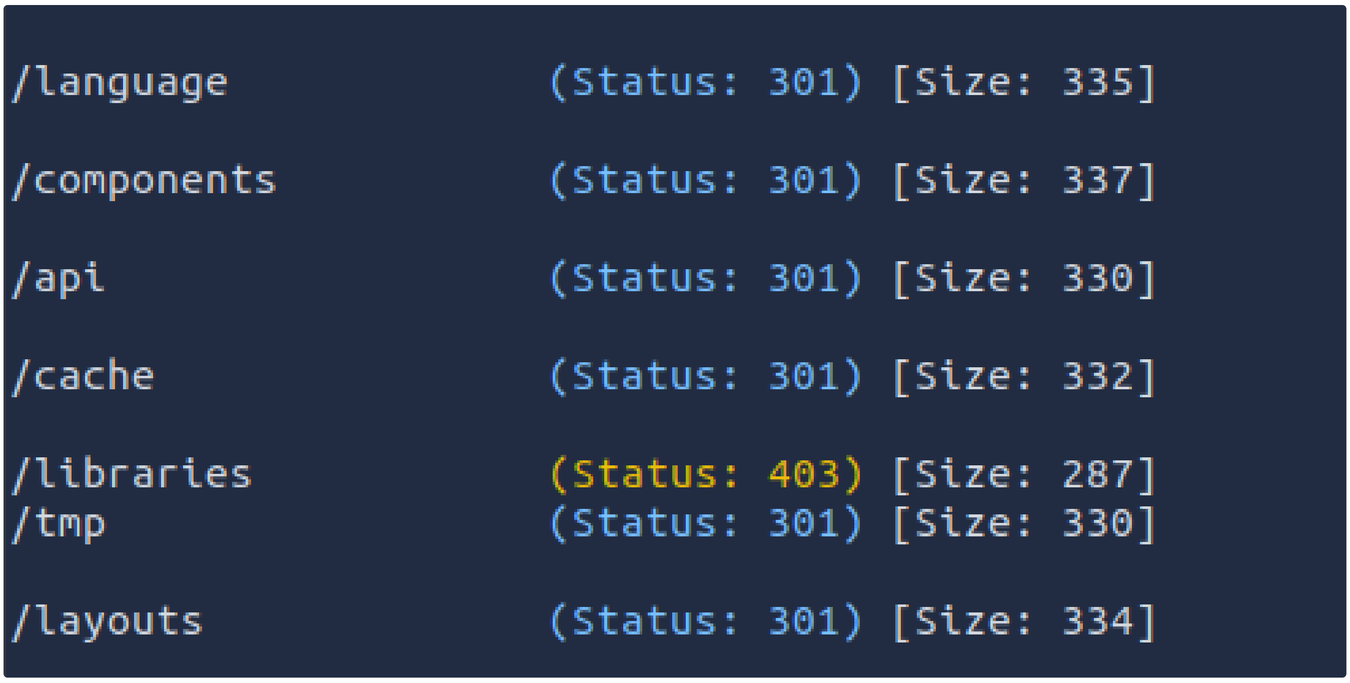
Nov 7, 2024 8

rutbar

TryHackMe—Shells Overview | Cyber Security 101 (THM)

Shell Overview

Oct 23, 2024 1 1



```
/language             (Status: 301) [Size: 335]

/components           (Status: 301) [Size: 337]

/api                  (Status: 301) [Size: 330]

/cache                (Status: 301) [Size: 332]

/libraries            (Status: 403) [Size: 287]
/tmp                  (Status: 301) [Size: 330]

/layouts              (Status: 301) [Size: 334]
```

embossdotar

TryHackMe — Gobuster: The Basics — Writeup

Key points: Recon | Enumeration | Gobuster. Gobuster: The Basics by awesome TryHackMe! 🎉

Oct 23, 2024 1

rutbar

TryHackMe — CAPA: The Basics | Cyber Security 101 (THM)

Tool Overview: How CAPA Works

Oct 23, 2024 13

See more recommendations