JavaScript Essentials

Essential Concepts

Variables

Variables are containers that allow you to store data values in them

Data Types

include `string` (text), `number`, `boolean` (true/false), null, undefined, and object (for more complex data like arrays or objects).

Functions

block of code designed to perform a specific task

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

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.

Integrating JavaScript in HTML

Internal JavaScript

Internal JS refers to embedding the JS code directly within an HTML document. This method is preferable for beginners because it allows them to see how the script interacts with the HTML

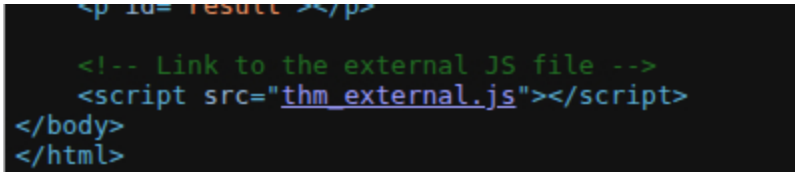
External JavaScript

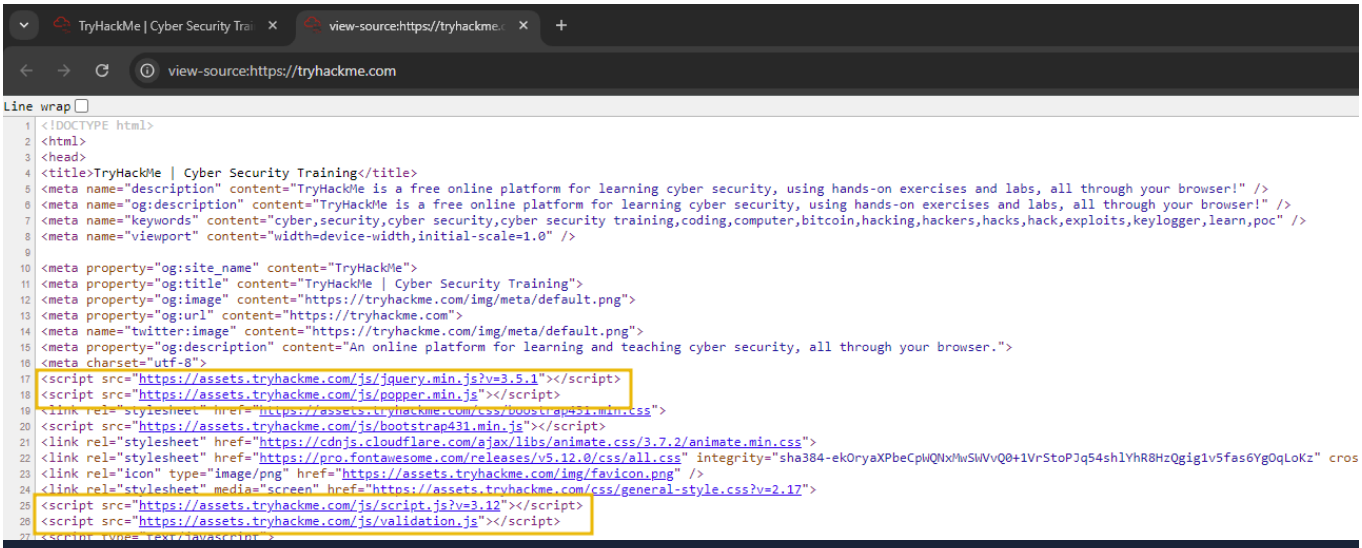
External JS involves creating and storing JS code in a separate file ending with a `.js` file extension. This method helps developers keep the HTML document clean and organised

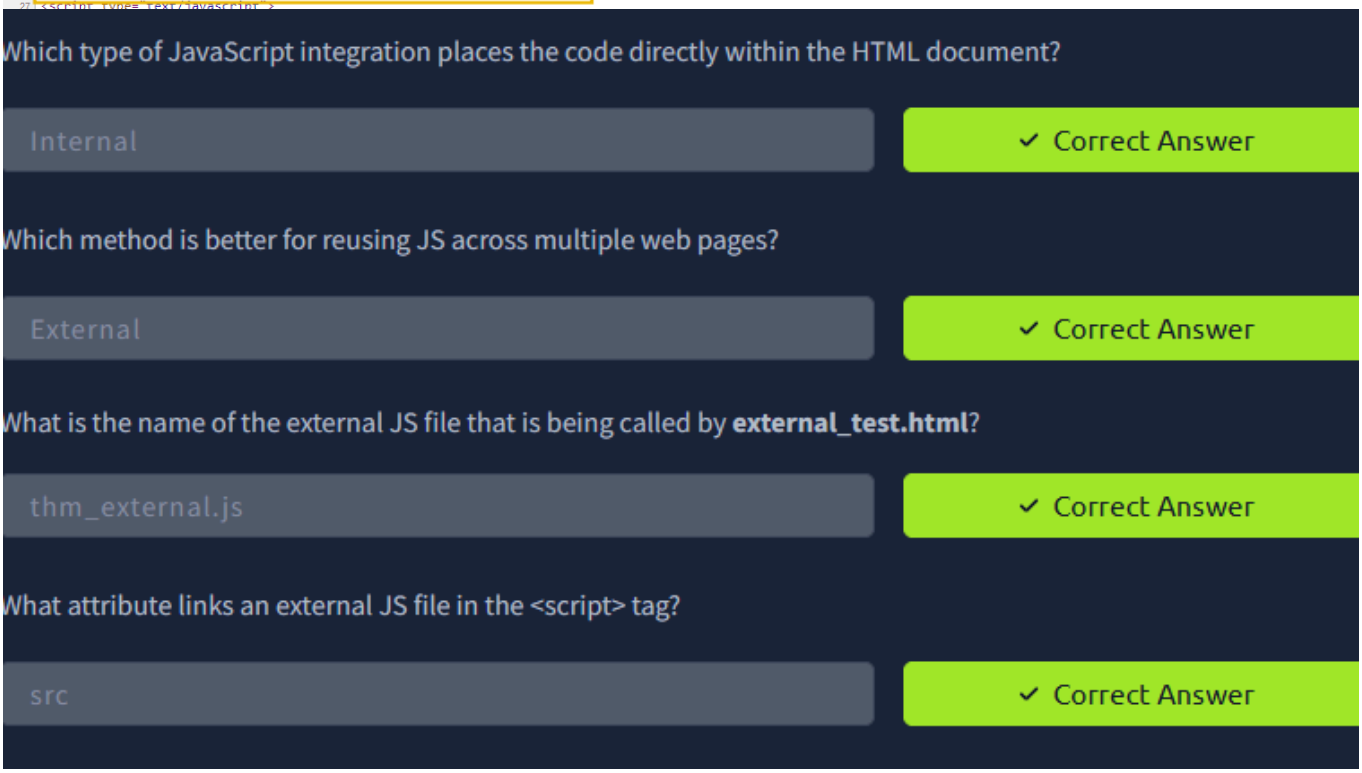
Verifying Internal or External JS

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`.





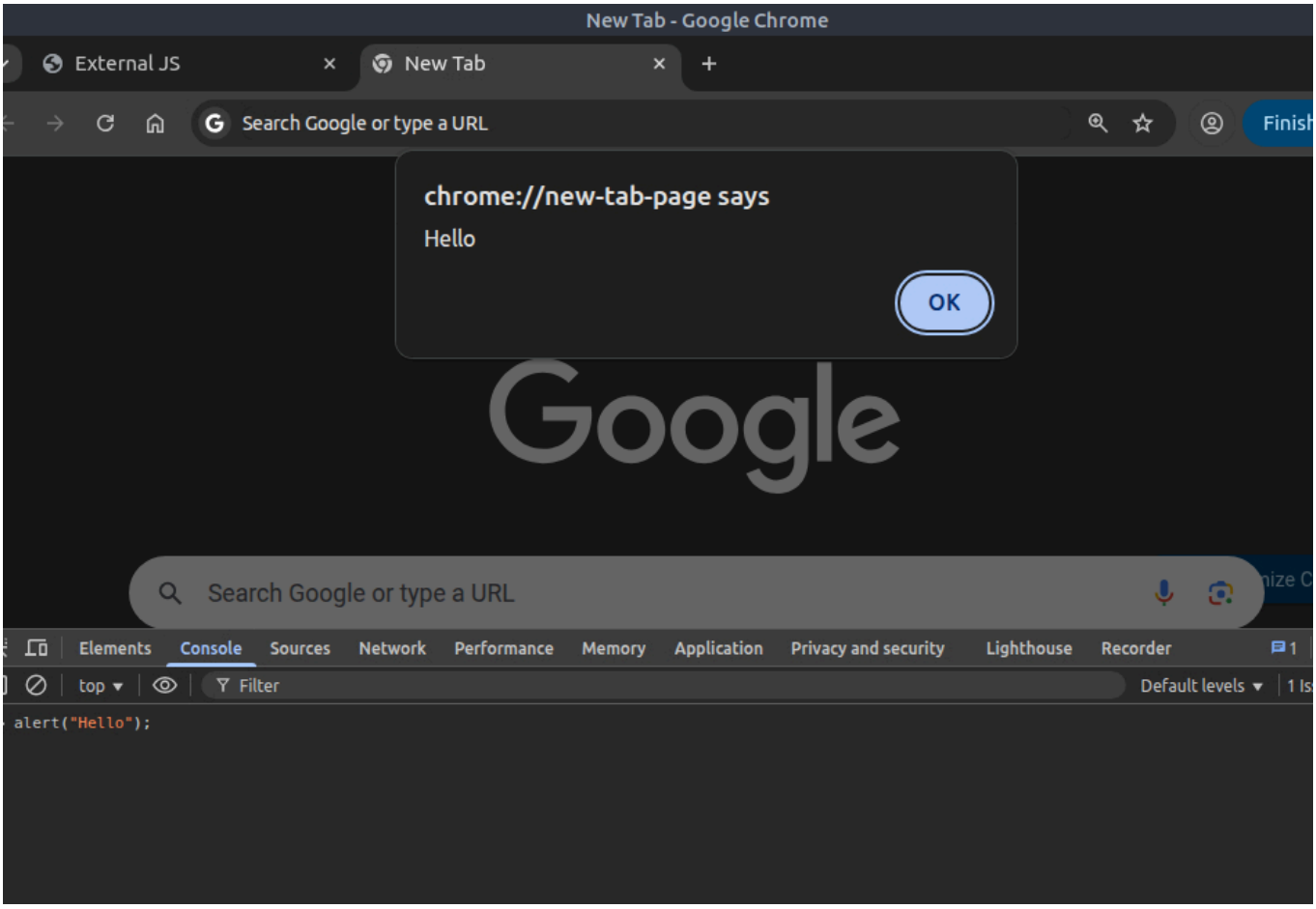


Abusing Dialogue Functions

One of the main objectives of JS is to provide dialogue boxes for interaction with users and dynamically update content on web pages. JS provides built-in functions like `alert`, `prompt`, and `confirm` to facilitate this interaction

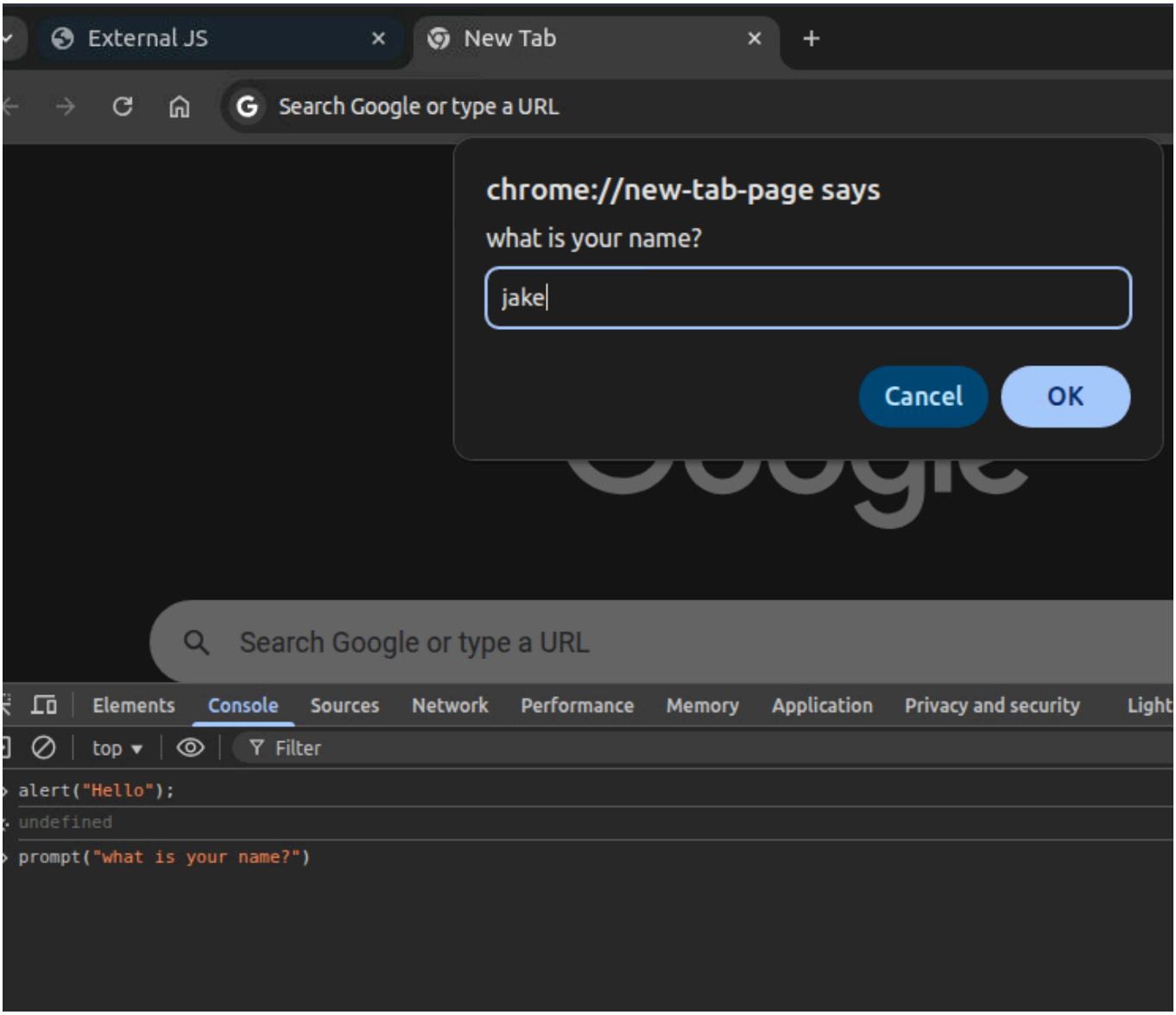
Alert

The alert function displays a message in a dialogue box with an "`OK`" button, typically used to convey information or warnings to users



Prompt

The prompt function displays a dialogue box that asks the user for input. It returns the entered value when the user clicks "`OK`", or null if the user clicks "`Cancel`". For example, to ask the user for their name, we would use `prompt("What is your name?");`



Confirm

The confirm function displays a dialogue box with a message and two buttons: "`OK`" and "`Cancel`". It returns true if the user clicks "`OK`" and false if the user clicks "`Cancel`"

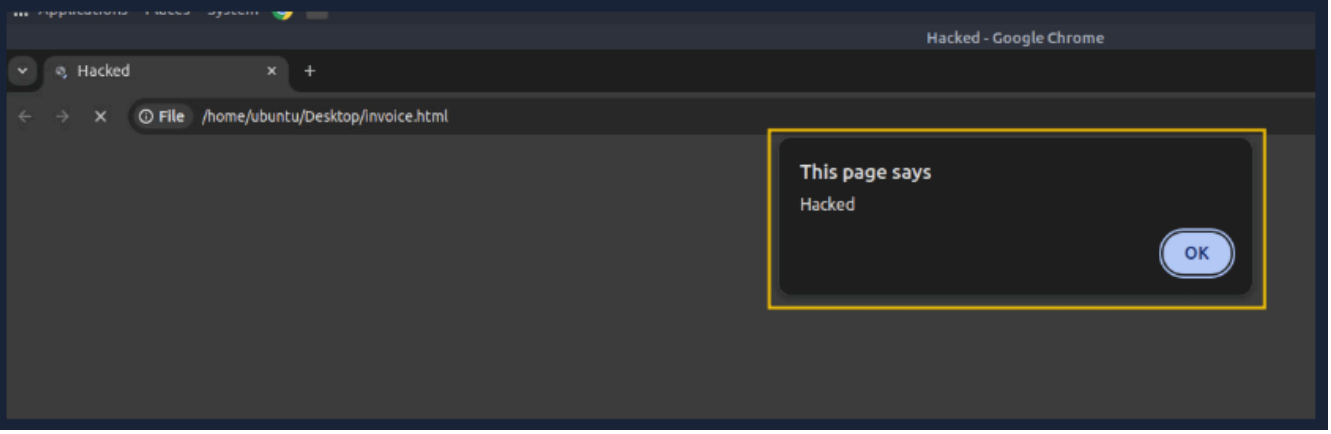
How Hackers Exploit the Functionality

Imagine receiving an email from a stranger with an attached HTML file. The file looks harmless, but when you open it, it contains JS that disrupts your browsing experience.

"**Hacked**" three times:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Hacked</title>
</head>
<body>
    <script>
        for (let i = 0; i < 3; i++) {
            alert("Hacked");
        }
    </script>
</body>
</html>
```

On the Desktop of the attached VM, create a file called `invoice.html` and paste the above code. Double-click the file to open it, and the alert message will pop up three times, causing an undesired experience.



Imagine if a bad actor sent you a similar file, but instead of displaying the alert three times, the number is set to **500**. You would be forced to keep closing the alert boxes one after another

Bypassing Control Flow Statements

Control flow in JS refers to the order in which statements and code blocks are executed based on certain conditions. JS provides several control flow structures such as `if-else`, `switch` statements to make decisions, and loops like `for`, `while`, and `do...while` to repeat actions

Conditional Statements in Action

One of the most used conditional statements is the `if-else` statements, which allows you to execute different blocks of code depending on whether a condition evaluates to `true` or `false`

Bypassing Login Forms

Suppose a developer has implemented authentication functionality in JS, where only users with the username "`admin`" and passwords matching a specific value are allowed to log in

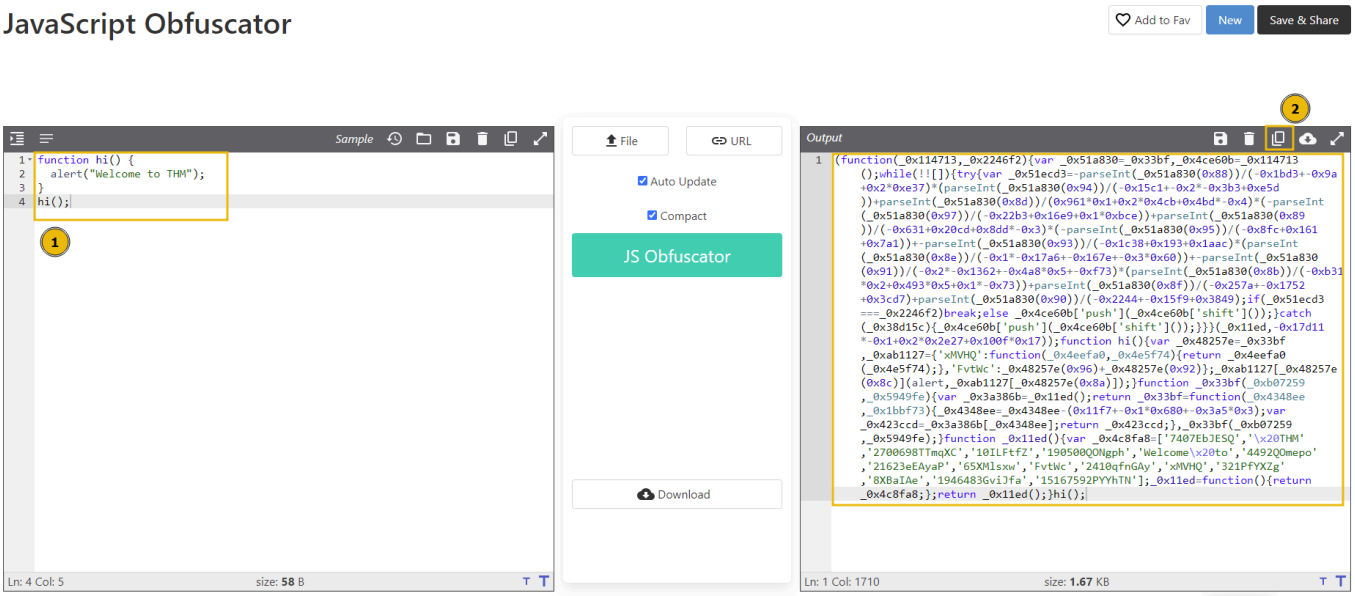
Exploring Minified Files

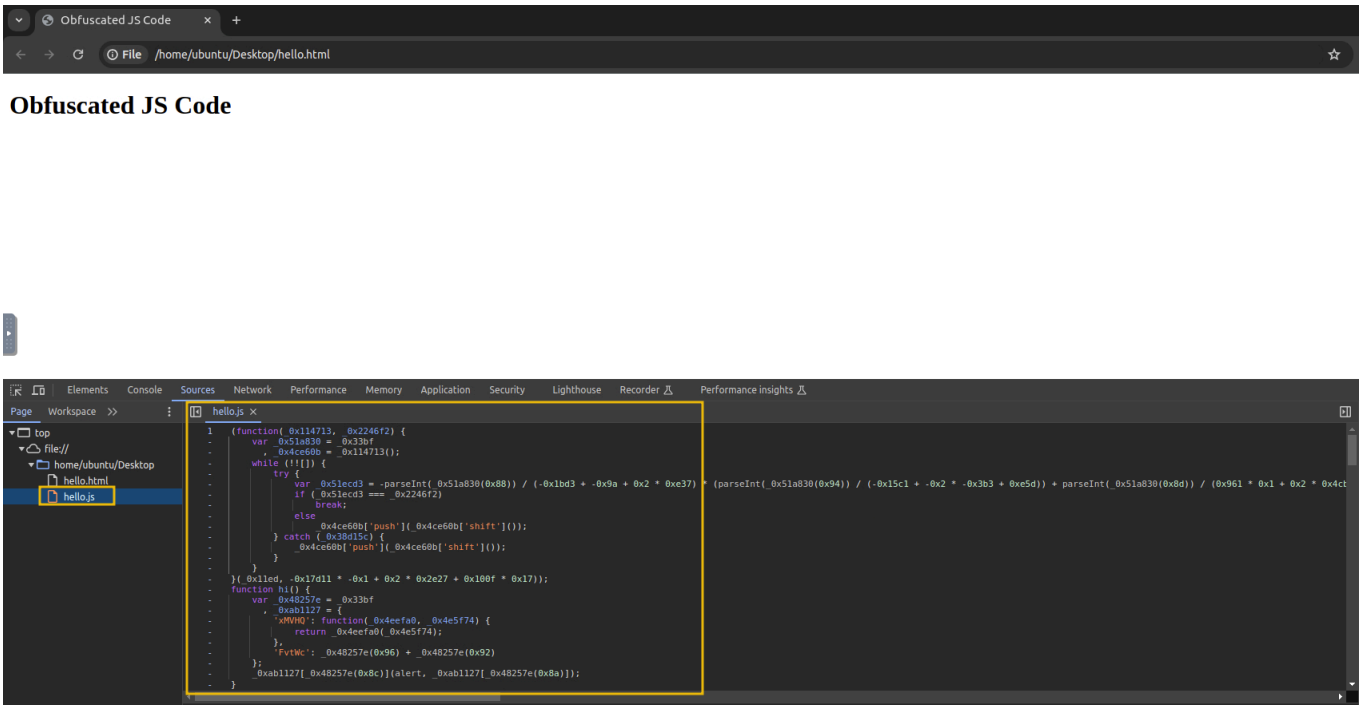
Minification in JS is the process of compressing JS files by removing all unnecessary characters, such as spaces, line breaks, comments, and even shortening variable names

Similarly, **obfuscation** is often used to make JS harder to understand by adding undesired code, renaming variables and functions to meaningless names, and even inserting dummy code.

Obfuscation in Action

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:





Deobfuscating a Code

We can also deobfuscate an obfuscated code using an online tool. Visit the website, then paste the obfuscated code into the provided dialogue box

Best Practices

Avoid Relying on Client-Side Validation Only

Since a user can **disable/manipulate** JS on the client side, performing validation on the server side is also essential.

Refrain from Adding Untrusted Libraries

Bad actors have uploaded a bundle of libraries on the internet with names that resemble legitimate ones. So, if you blindly include a malicious library, you will expose your web application to threats.

Avoid Hardcoded Secrets

Never hardcode sensitive data like **API keys**, **access tokens**, or **credentials** into your JS code, as the user can easily check the source code and get the password.

Minify and Obfuscate Your JavaScript Code

Minifying and obfuscating JS code reduces its size, improves load time, and makes it harder for attackers to understand the logic of the code. Therefore, always **minify** and **obfuscate** the code when using code in production.