IE2062 – Web Security

Year 2, Semester 2

TryHackMe Room

BSc (Hons) in Information Technology

Sri Lanka Institute of Information Technology

Malabe

Sri Lanka

Student Name	Registration Number
P T D Minipura	IT23298408

Table of Contents

Room Overview

1. Title: Cracking Weak CSP – Misconfigurations in Action

This room focuses on real-world misconfigurations in Content Security Policies (CSP) that fail to effectively safeguard web applications against Cross-Site Scripting (XSS) attacks. Learners will explore how poorly implemented CSP rules, rather than providing protection, can create a false sense of security and leave applications vulnerable. The room highlights common weaknesses such as the use of overly permissive directives like 'unsafe-inline', allowing the execution of malicious inline scripts. Participants will also encounter scenarios where CSP headers are missing crucial directives, improperly scoped or misused with wildcards, leading to policy bypasses. Through hands-on challenges, learners will practice analyzing flawed CSP configurations, crafting payloads that exploit these weaknesses and understanding how attackers can inject and execute arbitrary JavaScript despite the presence of CSP. By the end of the room, participants will gain practical skills in identifying CSP issues and understanding best practices for implementing strong, effective CSPs that genuinely enhance the security posture of web applications.

2. Key Concepts

1. CSP Headers
   - HTTP response headers that define which resources (scripts, styles, images, etc.) are allowed to load and execute.
   - Help prevent attacks like Cross-Site Scripting (XSS) by enforcing strict resource loading rules.

2. Inline Scripts
   - JavaScript code written directly in the HTML document inside <script> tags.
   - Often a security risk, as attackers can exploit them if input validation is weak.
   - CSP can control their execution by using directives like 'unsafe-inline', nonces or hashes.

3. Wildcards
   - The * character allows resources to be loaded from any origin.
   - Simplifies configuration but can dangerously weaken security.
   - Should be avoided in sensitive directives like script-src to maintain strong protection.

4. Secure Policy Crafting
    - Involves carefully specifying trusted sources and avoiding unsafe practices.
    - Prefer nonces (nonce-...) or hashes over allowing inline scripts.
    - Avoid wildcards and overly broad source definitions to minimize attack surfaces.
    - Regularly audit and update policies as the application evolves.

Learning Objectives

By completing this room, participants will develop a comprehensive set of skills and deepen their knowledge in the following areas:

1. **Understanding CSP's Role in XSS Prevention:**
   Gain a clear understanding of how Content Security Policy (CSP) is designed to reduce the risk of Cross-Site Scripting (XSS) attacks by restricting the sources from which content can be loaded and executed.

2. **Recognizing Misconfigured CSP Headers:**
   Learn to identify common mistakes in CSP implementation, such as overly permissive directives, missing critical restrictions or the improper use of wildcards and unsafe allowances.

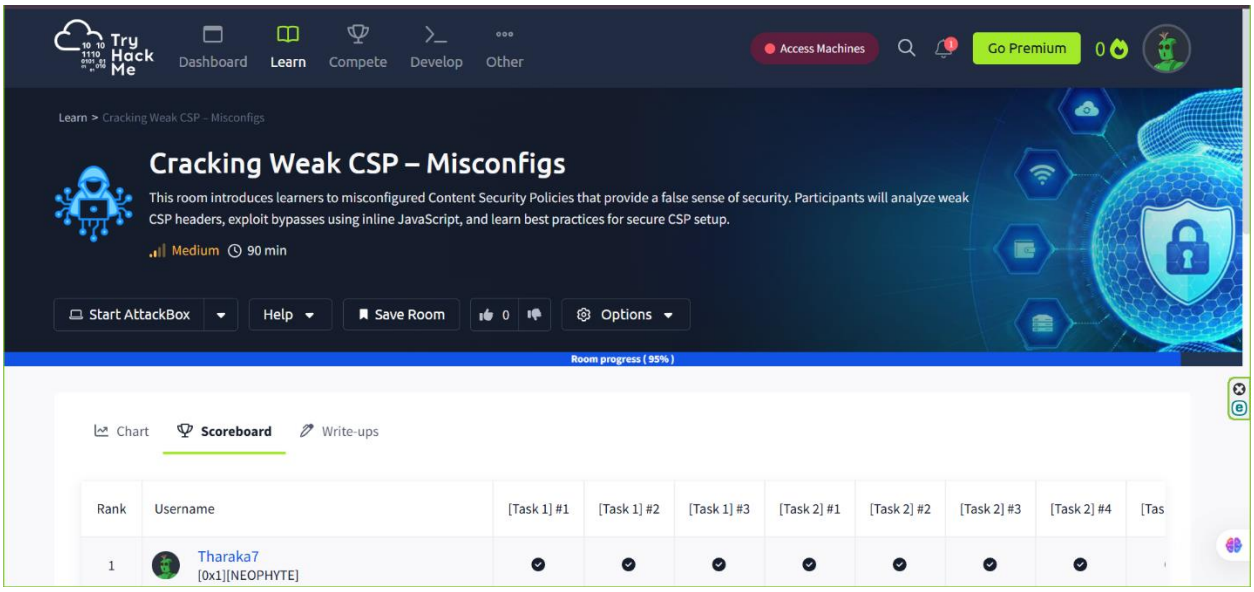
3. **Performing CSP Bypasses through Inline JavaScript and Input Injection:**
   Develop practical skills in exploiting weak CSP configurations by injecting malicious input or leveraging inline scripts to achieve script execution even when a CSP is present.

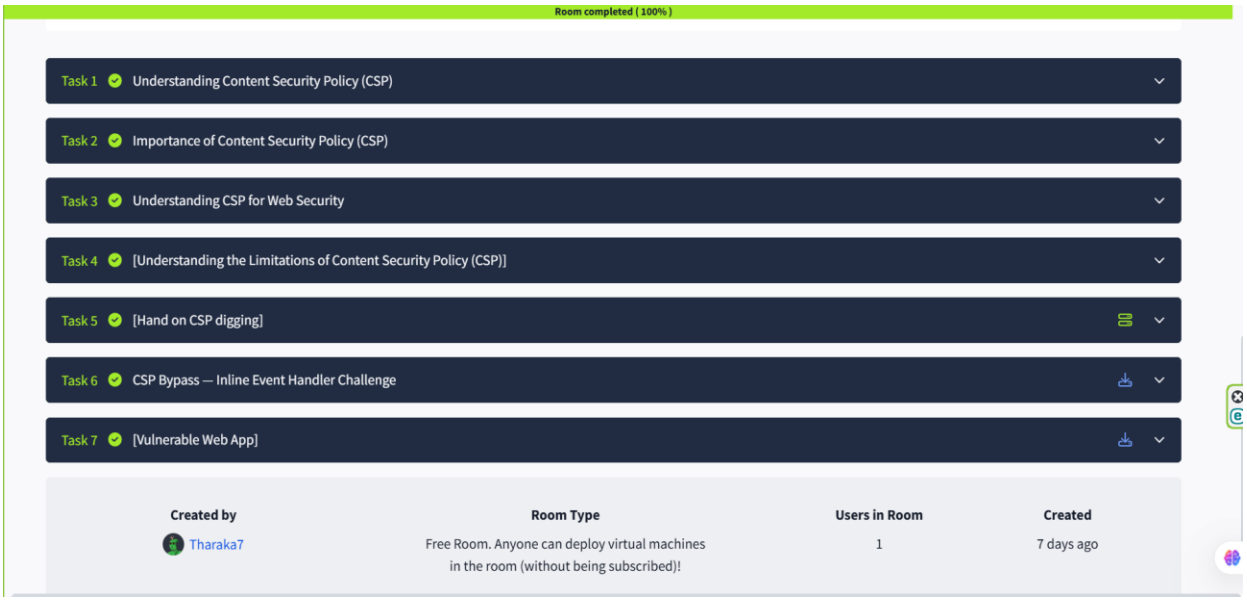
4. **Applying Strong CSP Mitigations:**
   Acquire the knowledge to design and implement robust CSP policies, using techniques like nonces, hashes and strict source whitelisting, to effectively defend against XSS and related attacks.

Room Structure

The following is the dashboard of my TryHackMe Room.

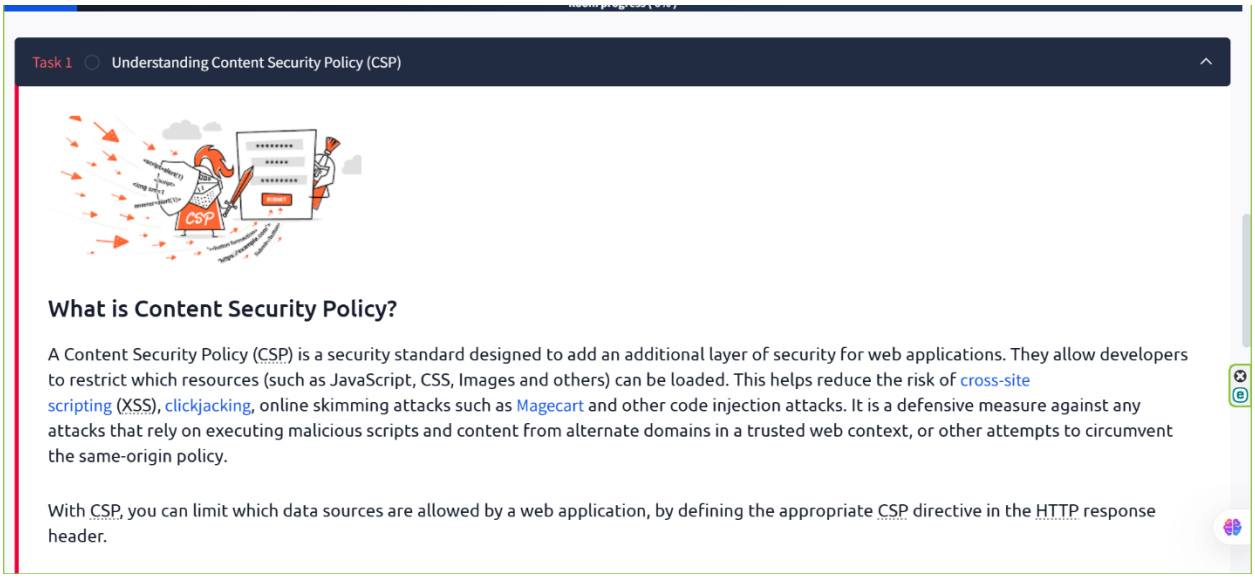
Room Link: https://tryhackme.com/jr/crackingweakcspmisconfigs

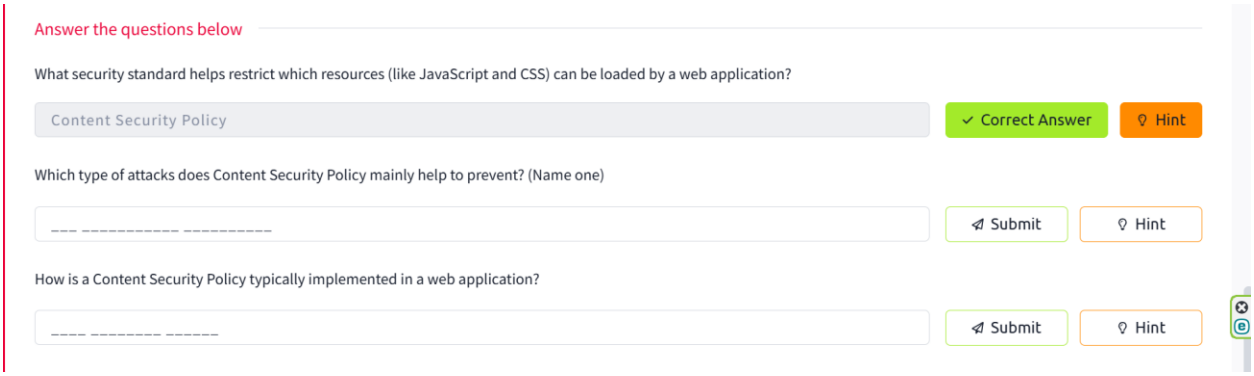


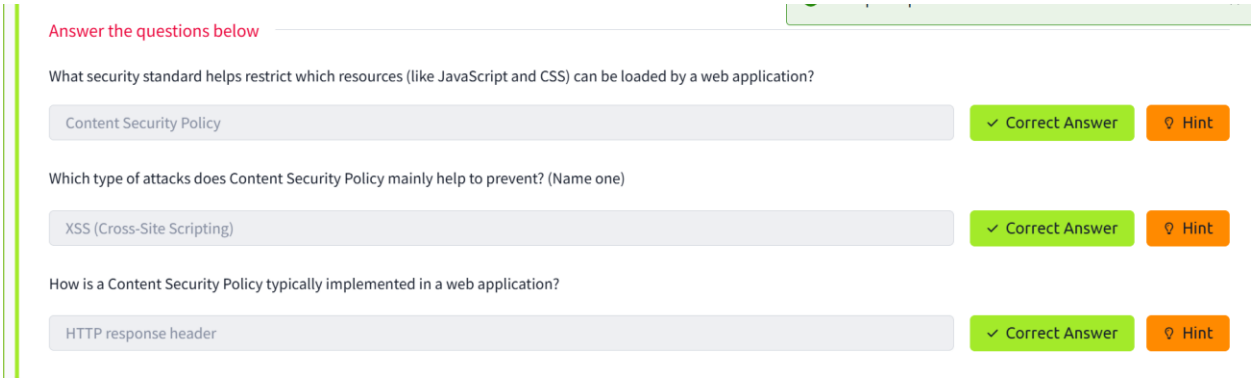


1. Task 1 – Understanding Content Security Policy (CSP)

- This Room introduces Content Security Policy, a user can gain basic knowledge on this vulnerability.
- Users can read the content below and test themselves by answering the following questions.

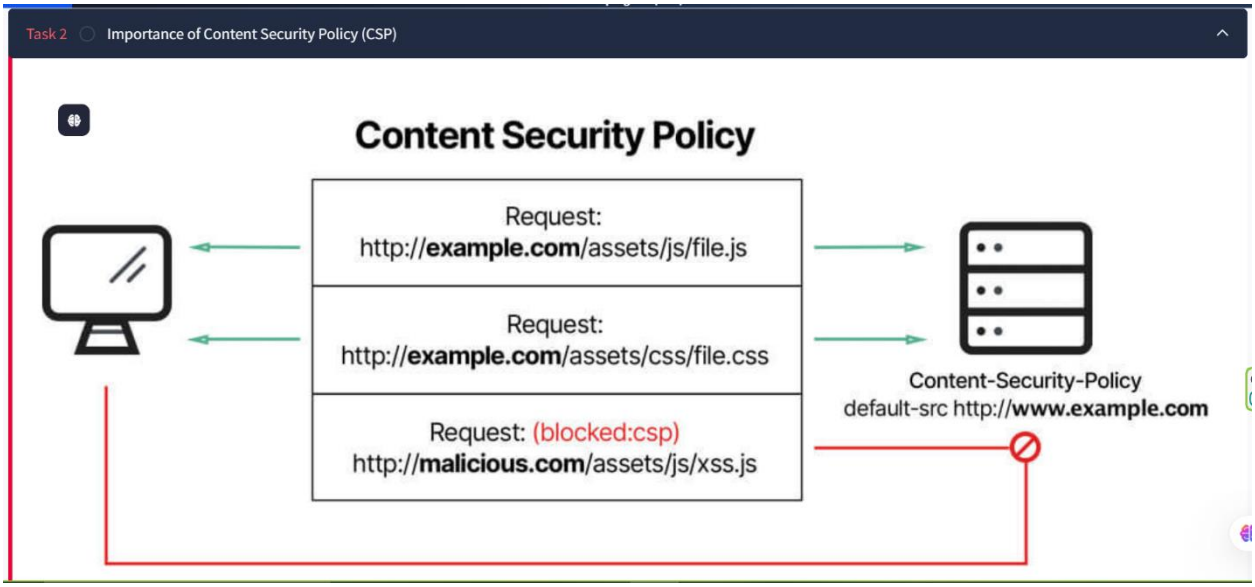






2. Task 2 – Importance of Content Security Policy (CSP)

- The importance of Content Security Policy is addressed here. This section understands the critical role of Content Security Policy (CSP) in mitigating Cross-Site Scripting (XSS) attacks and enforcing secure communication protocols like HTTPS.
- By answering the questions provided below the users can evaluate themselves.



Why is a Content Security Policy Important?

Mitigating Cross-Site Scripting

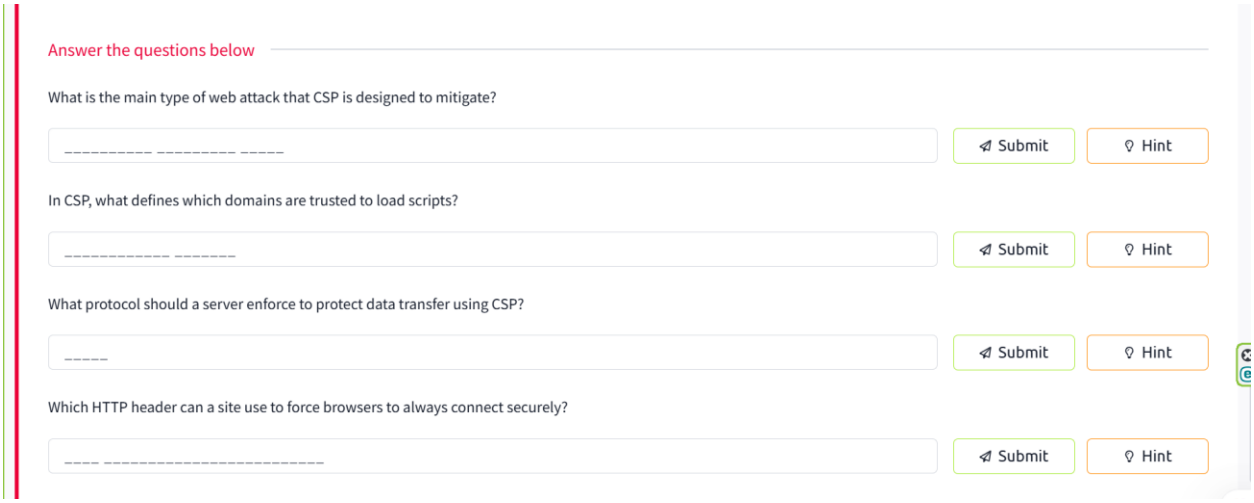
The main purpose of CSP is to mitigate and detect XSS attacks. XSS attacks exploit the browser's trust in the content received from the server. The victim's browser is exposed to execution of malicious scripts, because the browser trusts the source of the content.

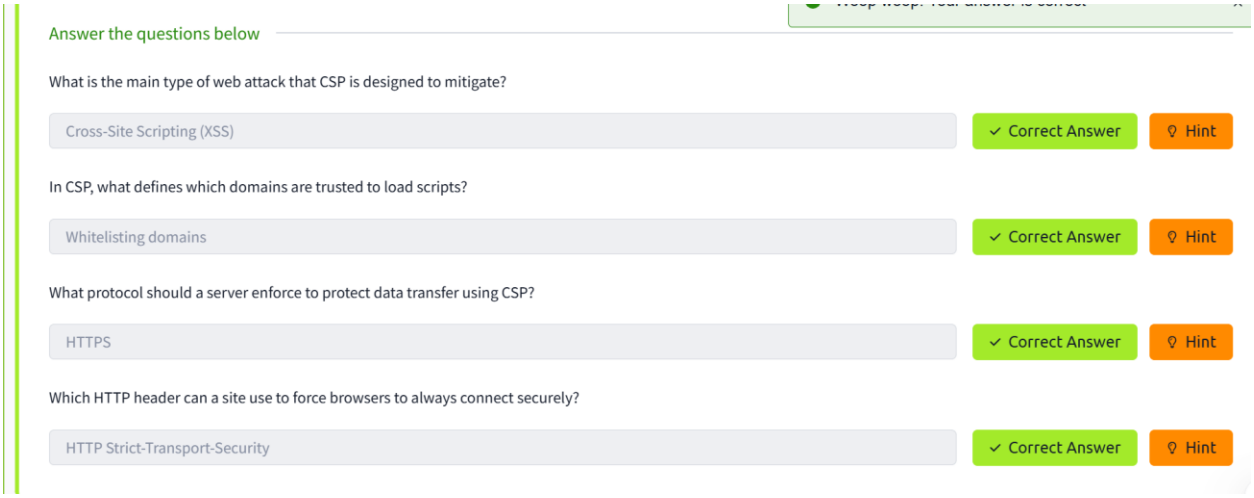
CSP allows server administrators to reduce or eliminate the ability of an attacker to trigger XSS, by specifying which Internet domains browsers should consider as legitimate sources of executable scripts. CSP-compliant browsers only run scripts contained source files that are retrieved from whitelisted domains, and ignore all other scripts (including inline script and HTML event handling attributes).

Mitigating Packet Sniffing and Enforcing HTTPS

In addition to whitelisting domains from which a browser may load content, servers can also specify the allowed protocols. For example, the server can specify that browsers must load content via HTTPS.

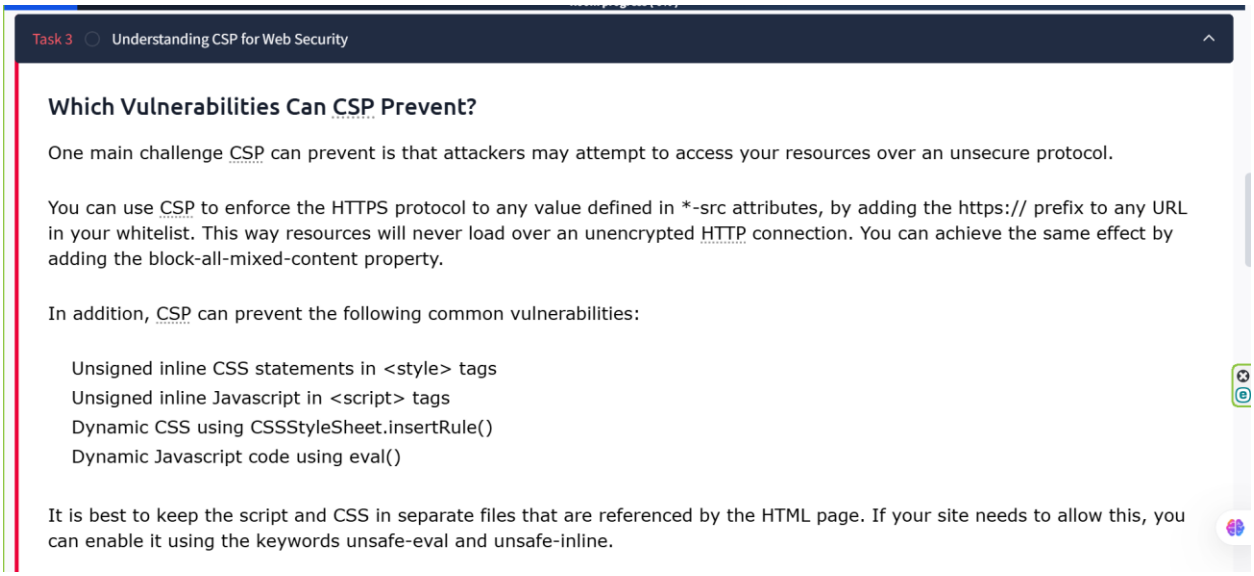
A comprehensive data transfer protection policy involves not only implementing HTTPS in data transfer, but also marking all cookies with the secure attribute and automatically redirecting HTTP pages to HTTPS. Additionally, sites may use HTTP Strict-Transport-Security headers to ensure that browsers only connect to the site via encrypted channels.


Answer the questions below
What is the main type of web attack that CSP is designed to mitigate?
Submit
Hint
In CSP, what defines which domains are trusted to load scripts?
Submit
Hint
What protocol should a server enforce to protect data transfer using CSP?
Submit
Hint
Which HTTP header can a site use to force browsers to always connect securely?
Submit
Hint


Answer the questions below
What is the main type of web attack that CSP is designed to mitigate?
Cross-Site Scripting (XSS)
Correct Answer
Hint
In CSP, what defines which domains are trusted to load scripts?
Whitelisting domains
Correct Answer
Hint
What protocol should a server enforce to protect data transfer using CSP?
HTTPS
Correct Answer
Hint
Which HTTP header can a site use to force browsers to always connect securely?
HTTP Strict-Transport-Security
Correct Answer
Hint

3. Task 3 – Understanding CSP for Web Security

- This task explains how Content Security Policy (CSP) can be used to prevent common vulnerabilities and its appropriate use cases in web applications.
- Users can assess themselves by answering the following questions finally.



Task 3 Understanding CSP for Web Security

Which Vulnerabilities Can CSP Prevent?

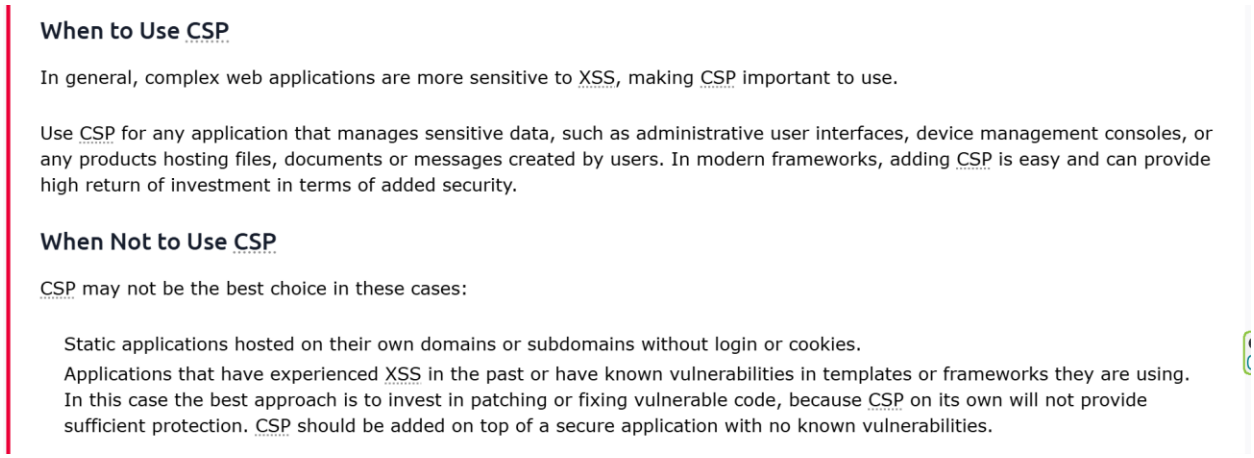
One main challenge CSP can prevent is that attackers may attempt to access your resources over an unsecure protocol.

You can use CSP to enforce the HTTPS protocol to any value defined in *-src attributes, by adding the https:// prefix to any URL in your whitelist. This way resources will never load over an unencrypted HTTP connection. You can achieve the same effect by adding the block-all-mixed-content property.

In addition, CSP can prevent the following common vulnerabilities:

- Unsigned inline CSS statements in <style> tags
- Unsigned inline Javascript in <script> tags
- Dynamic CSS using CSSStyleSheet.insertRule()
- Dynamic Javascript code using eval()

It is best to keep the script and CSS in separate files that are referenced by the HTML page. If your site needs to allow this, you can enable it using the keywords unsafe-eval and unsafe-inline.



When to Use CSP

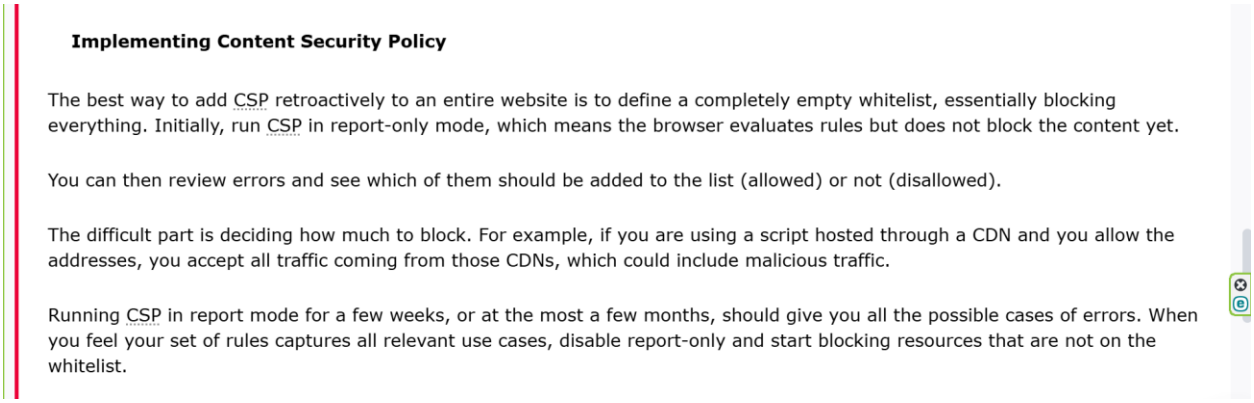
In general, complex web applications are more sensitive to XSS, making CSP important to use.

Use CSP for any application that manages sensitive data, such as administrative user interfaces, device management consoles, or any products hosting files, documents or messages created by users. In modern frameworks, adding CSP is easy and can provide high return of investment in terms of added security.

When Not to Use CSP

CSP may not be the best choice in these cases:

- Static applications hosted on their own domains or subdomains without login or cookies.
- Applications that have experienced XSS in the past or have known vulnerabilities in templates or frameworks they are using. In this case the best approach is to invest in patching or fixing vulnerable code, because CSP on its own will not provide sufficient protection. CSP should be added on top of a secure application with no known vulnerabilities.



Implementing Content Security Policy

The best way to add CSP retroactively to an entire website is to define a completely empty whitelist, essentially blocking everything. Initially, run CSP in report-only mode, which means the browser evaluates rules but does not block the content yet.

You can then review errors and see which of them should be added to the list (allowed) or not (disallowed).

The difficult part is deciding how much to block. For example, if you are using a script hosted through a CDN and you allow the addresses, you accept all traffic coming from those CDNs, which could include malicious traffic.

Running CSP in report mode for a few weeks, or at the most a few months, should give you all the possible cases of errors. When you feel your set of rules captures all relevant use cases, disable report-only and start blocking resources that are not on the whitelist.

Answer the questions below

What protocol can CSP enforce to ensure resources are loaded over a secure connection?

___ ___ _____ ___ _____ _____ __ _____ ___ _____:// _____ __ ____ __ ___ _____.

Submit Hint

Which of the following vulnerabilities can **CSP** prevent?
A) Unsigned inline CSS
B) SQL Injection
C) Buffer Overflow
D) Cross-Site Request Forgery (CSRF)

__ _____ _____ ___

Submit Hint

When should CSP NOT be used?

___ _____ ___ __ ____ ___ _____ _____ _____ _____ __ _____ __ ___ _____ ____ _____ ____

Submit Hint

What CSP keyword allows **inline scripts** and **dynamic code execution**?

___ _____ ___ _____ _____

Submit Hint

What is the purpose of running CSP in **report-only mode**?

__ _____ _____ _____ _____ _____ ___ _____ ___ _____.

Submit

Answer the questions below

What protocol can CSP enforce to ensure resources are loaded over a secure connection?

CSP can enforce the HTTPS protocol by adding the https:// prefix to URLs in the whitelist.

✓ Correct Answer Hint

Which of the following vulnerabilities can **CSP** prevent?
A) Unsigned inline CSS
B) SQL Injection
C) Buffer Overflow
D) Cross-Site Request Forgery (CSRF)

A) Unsigned inline CSS

✓ Correct Answer Hint

When should CSP NOT be used?

CSP should not be used for static applications without login or cookies or for applications with known vulnerabilities.

✓ Correct Answer Hint

What CSP keyword allows **inline scripts** and **dynamic code execution**?

The unsafe-inline and unsafe-eval keywords

✓ Correct Answer Hint

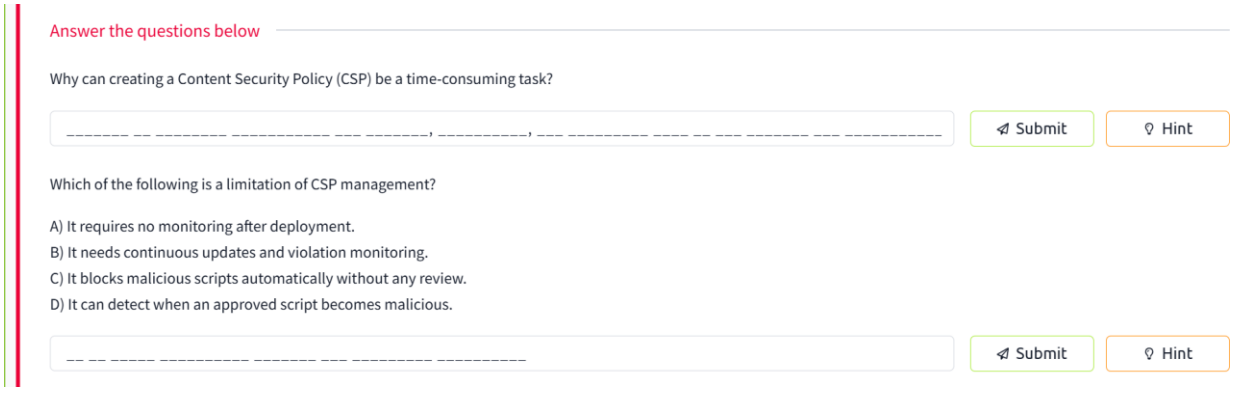
What is the purpose of running CSP in **report-only mode**?

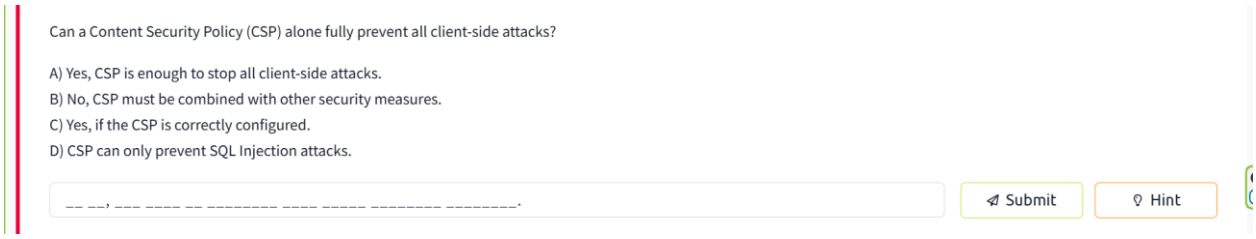
To evaluate errors without blocking content and refine the whitelist.

✓ Correct Answer

4. Task 4 - Understanding the Limitations of Content Security Policy (CSP)

- This section elaborates on learning the practical challenges and limitations involved in creating, managing and relying on CSP for securing web applications.
- By answering the following questions users can clarify the content well.





Answer the questions below

Why can creating a Content Security Policy (CSP) be a time-consuming task?

Because it requires identifying all domains, subdomains, and resources used by the website and understanding their functionality ✓ Correct Answer Hint

Which of the following is a limitation of CSP management?

A) It requires no monitoring after deployment.
B) It needs continuous updates and violation monitoring.
C) It blocks malicious scripts automatically without any review.
D) It can detect when an approved script becomes malicious.

B) It needs continuous updates and violation monitoring ✓ Correct Answer Hint

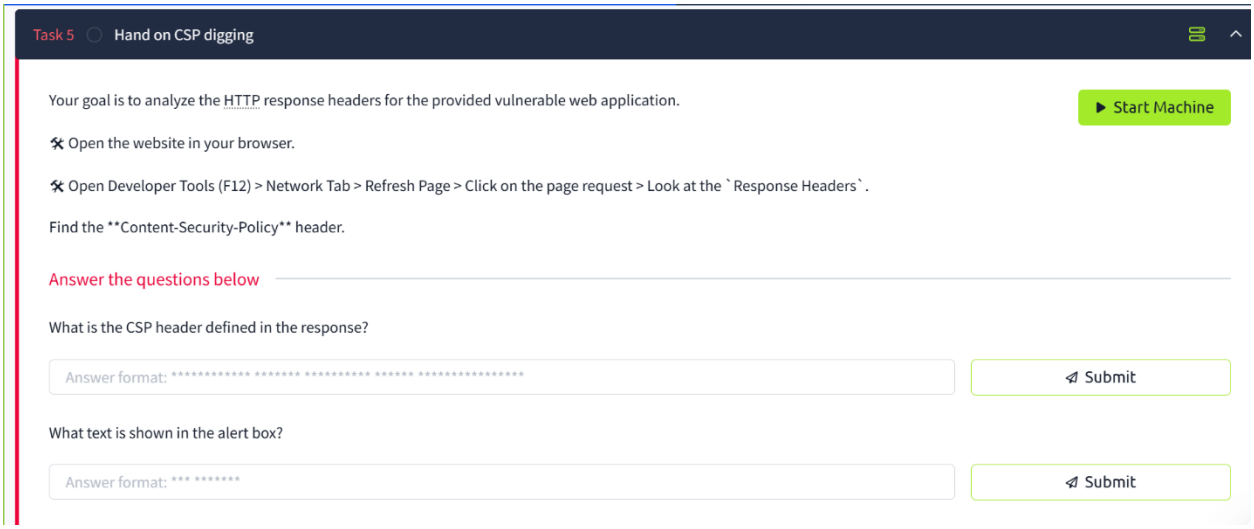
Can a Content Security Policy (CSP) alone fully prevent all client-side attacks?

A) Yes, CSP is enough to stop all client-side attacks.
B) No, CSP must be combined with other security measures.
C) Yes, if the CSP is correctly configured.
D) CSP can only prevent SQL Injection attacks.

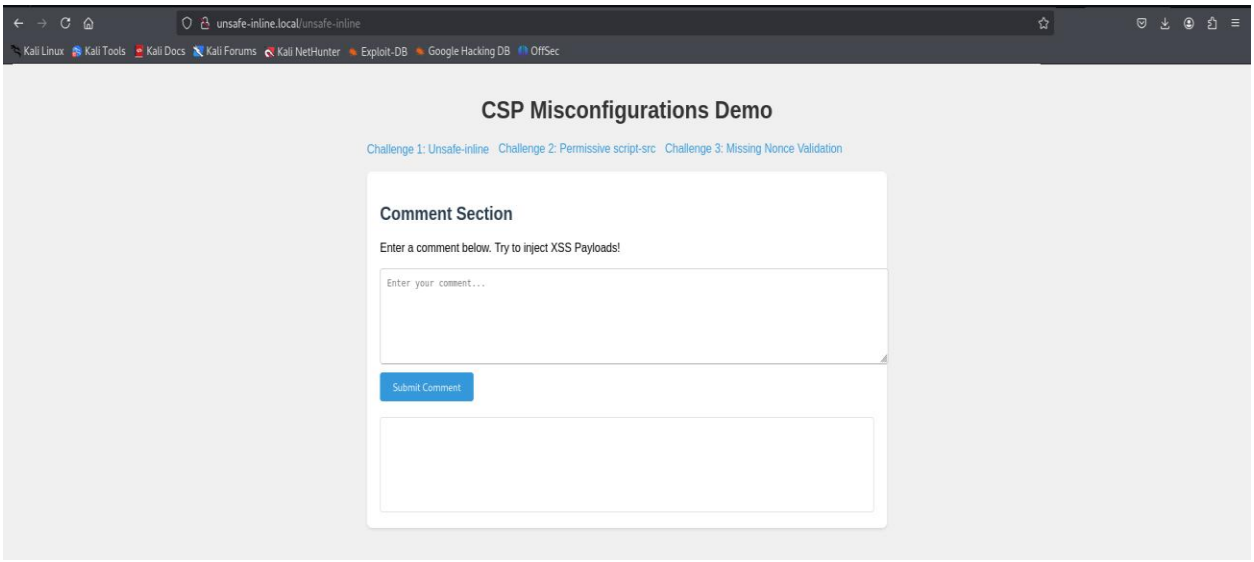
B) No, CSP must be combined with other security measures. ✓ Correct Answer Hint

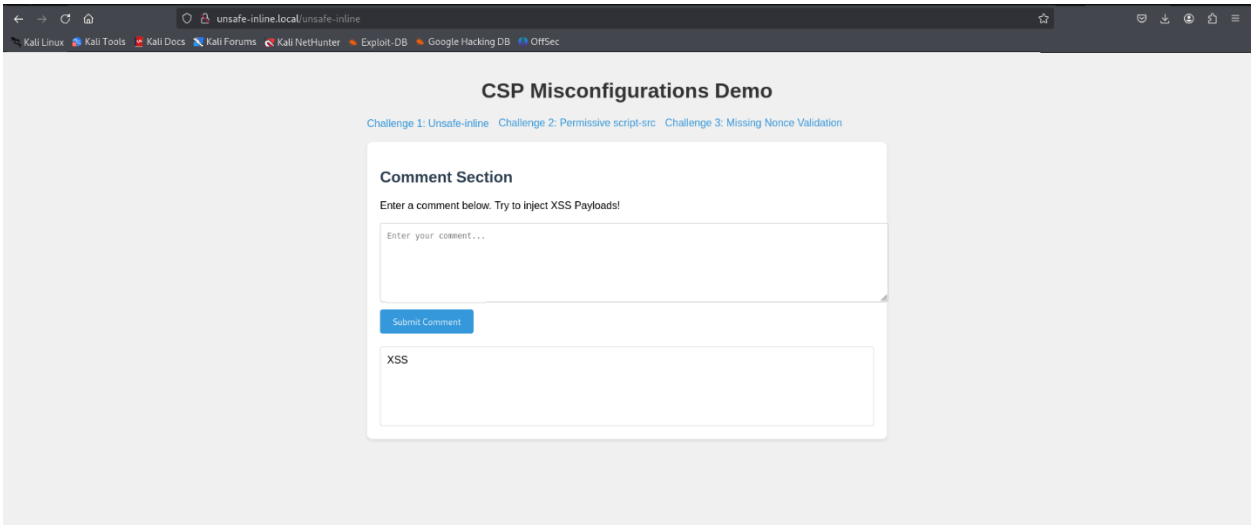
5. Task 5 - Hand on CSP digging

- This task gives users gain some hand on experience related to Content Security Policy Misconfigurations identification.



- As instructed on the task go to the developer tools and look for the response headers, identify the CSP header defined there.





Answer the questions below

What is the CSP header defined in the response?

"default-src 'self'; script-src 'self' 'unsafe-inline'" ✓ Correct Answer

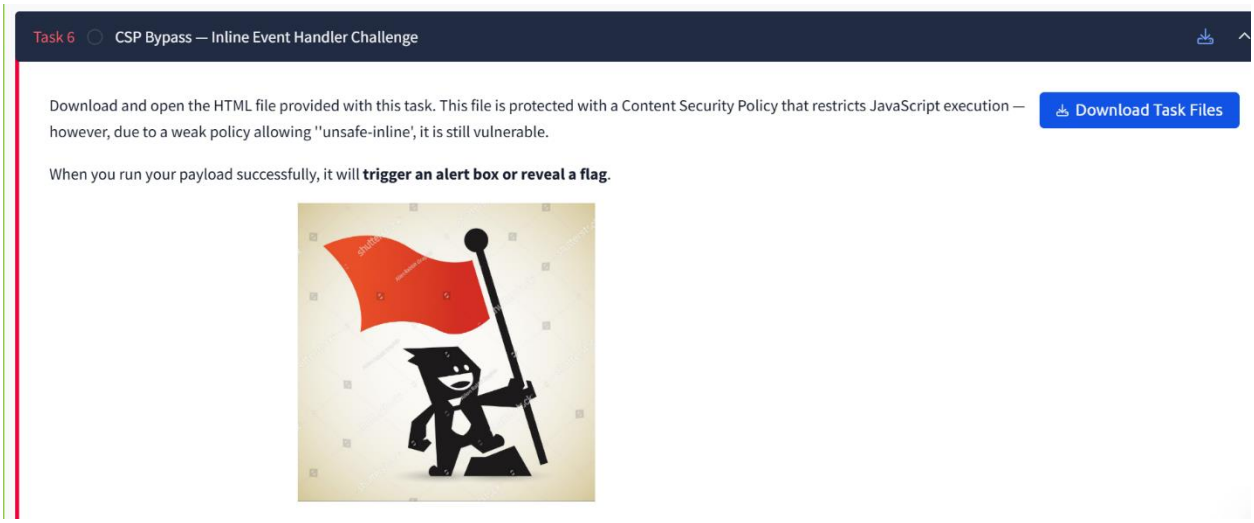
What text is shown in the alert box?

XSS Bypass! ✓ Correct Answer

6. Task 6 - CSP Bypass — Inline Event Handler Challenge

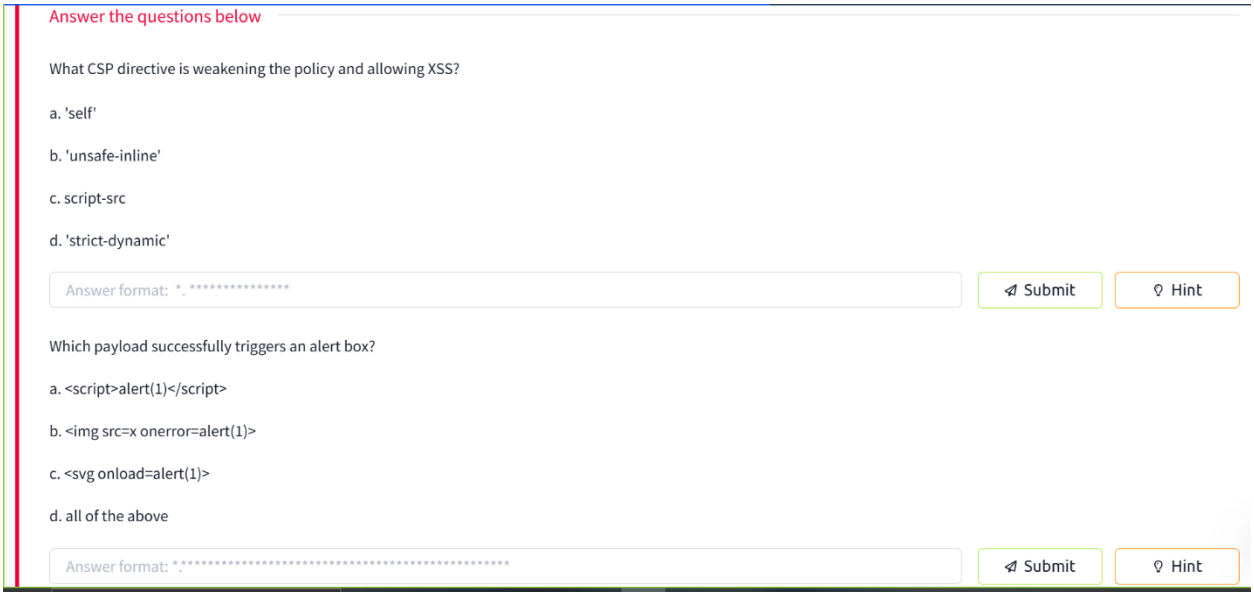
- This task shows though it is protected by a Content Security Policy that restricts JavaScript execution, how it is still vulnerable due to a weak policy allowing 'unsafe-inline'.
- Task Instructions:

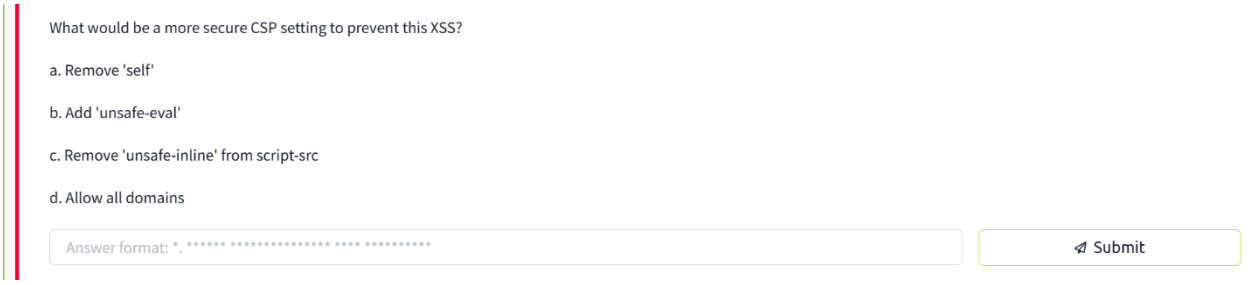
1. Download and open the HTML file in your browser.
2. You will see an input field and a button labeled **Run Payload**.
3. Inject a payload that abuses the CSP misconfiguration.
4. Your goal is to make an alert box appear or retrieve the hidden flag.



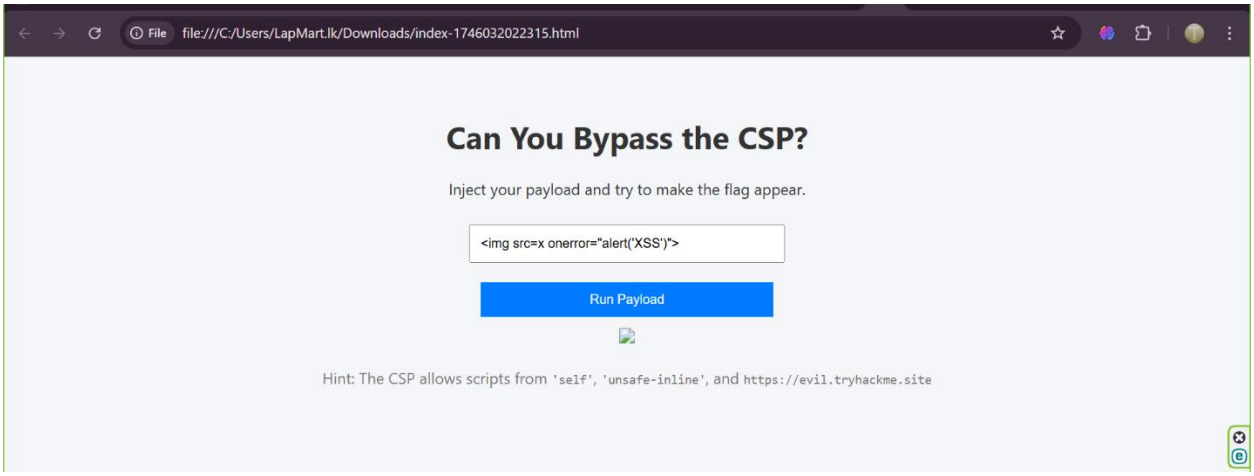
Task Instructions:

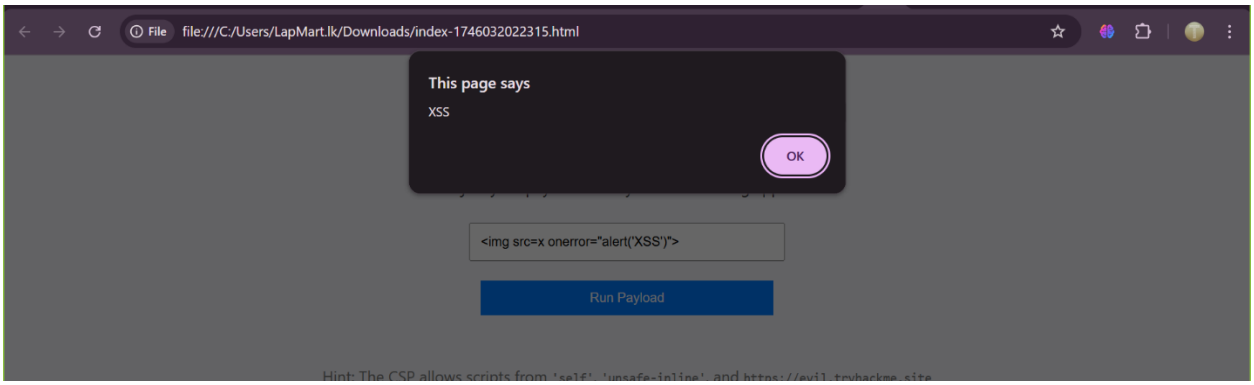
1. Download and open the HTML file in your browser.
2. You will see an input field and a button labeled **Run Payload**.
3. Inject a payload that abuses the CSP misconfiguration.
4. Your goal is to make an alert box appear or retrieve the hidden flag.

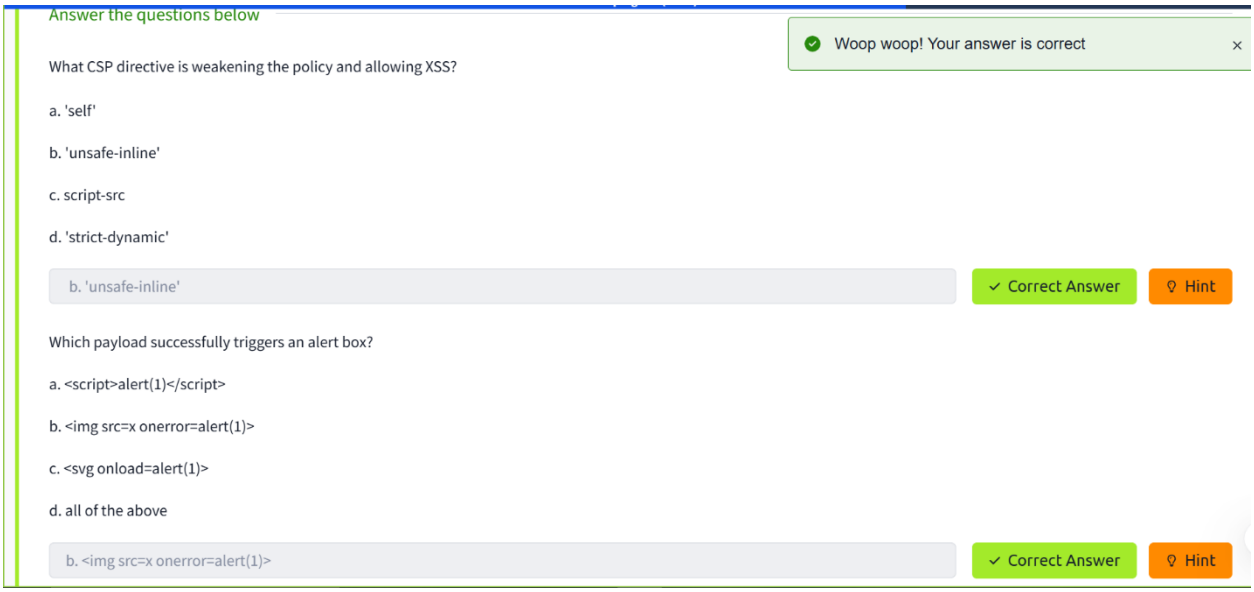


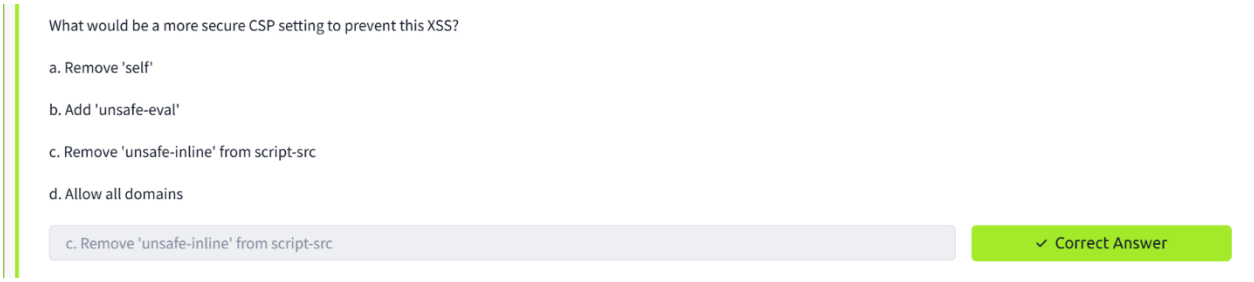


- When you download the file, you can open it through a browser and visit this simple page. Then give a payload to identify the inline CSP vulnerability.



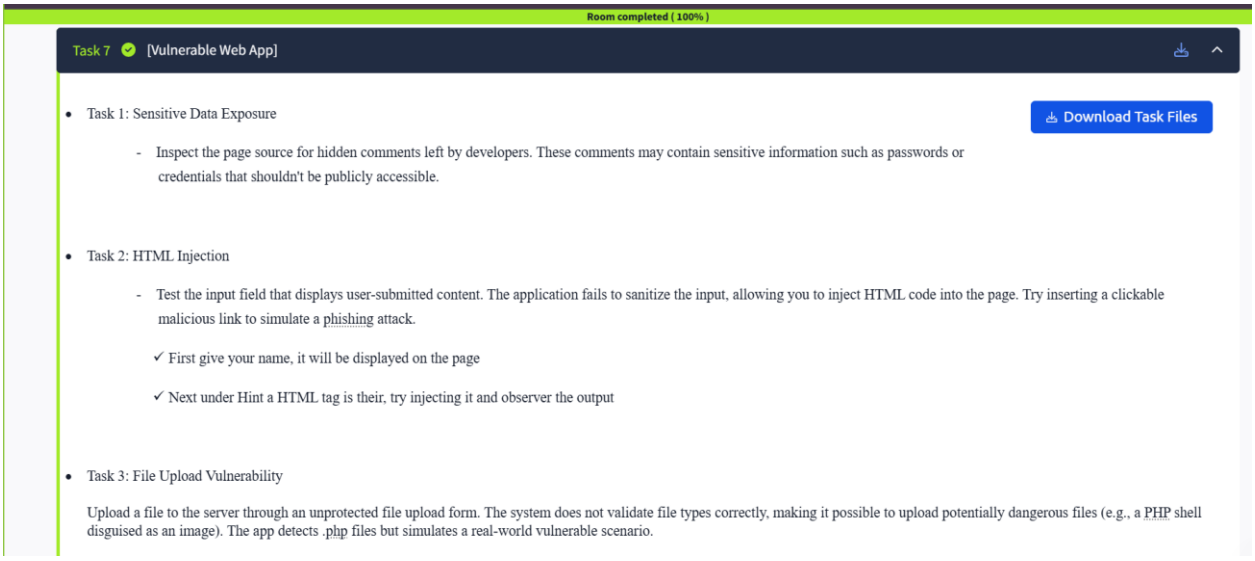

File file:///C:/Users/LapMart.lk/Downloads/index-1746032022315.html
This page says
XSS
OK
<img src=x onerror="alert('XSS')">
Run Payload

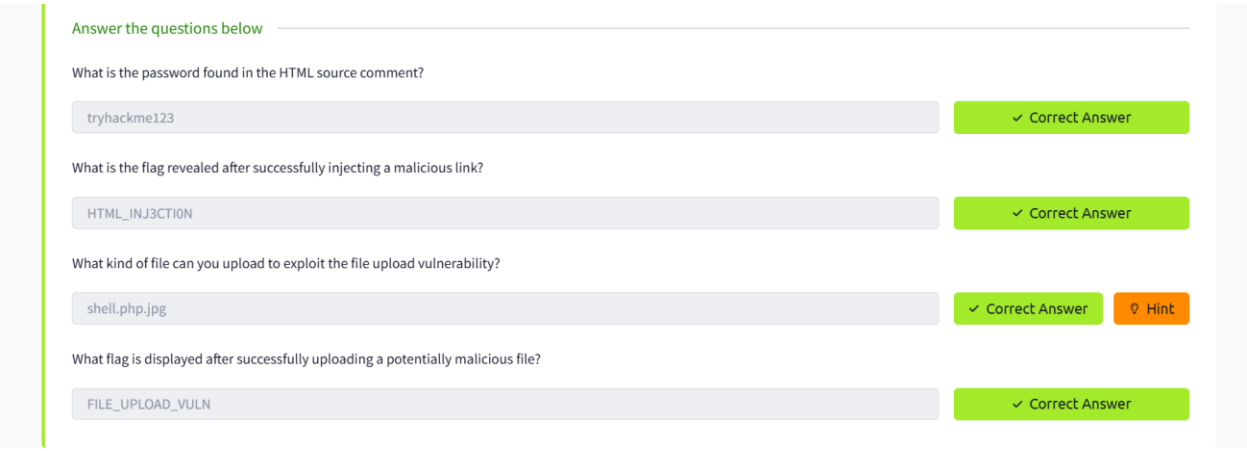

Answer the questions below
Woop woop! Your answer is correct
What CSP directive is weakening the policy and allowing XSS?
a. 'self'
b. 'unsafe-inline'
c. script-src
d. 'strict-dynamic'
b. 'unsafe-inline'
Correct Answer
Hint
Which payload successfully triggers an alert box?
a. <script>alert(1)</script>
b. <img src=x onerror=alert(1)>
c. <svg onload=alert(1)>
d. all of the above
b. <img src=x onerror=alert(1)>
Correct Answer
Hint

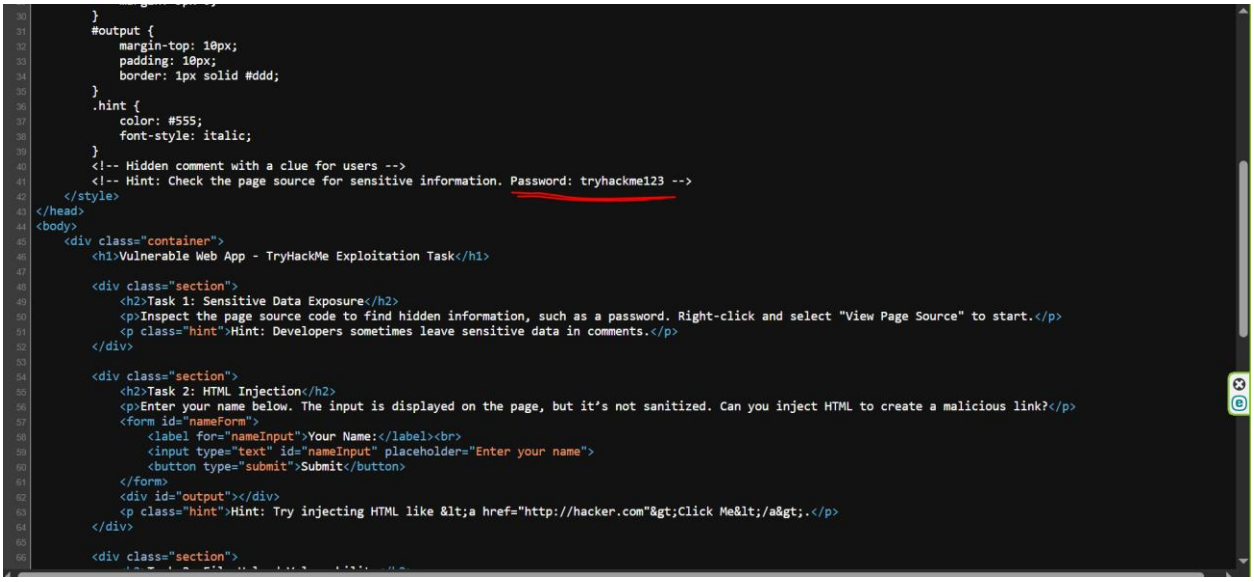

What would be a more secure CSP setting to prevent this XSS?
a. Remove 'self'
b. Add 'unsafe-eval'
c. Remove 'unsafe-inline' from script-src
d. Allow all domains
c. Remove 'unsafe-inline' from script-src
Correct Answer

7. Task 7 – [Vulnerable Web App]

- Welcome to the Vulnerable Web App designed to simulate common web vulnerabilities in a safe, sandboxed environment. In this task, you'll explore and exploit three common flaws found in insecure web applications. Each challenge demonstrates a different vulnerability and includes hints to guide your testing process.







```
        }
        #output {
            margin-top: 10px;
            padding: 10px;
            border: 1px solid #ddd;
        }
        .hint {
            color: #555;
            font-style: italic;
        }
        <!-- Hidden comment with a clue for users -->
        <!-- Hint: Check the page source for sensitive information. Password: tryhackme123 -->
    </style>
</head>
<body>
    <div class="container">
        <h1>Vulnerable Web App - TryHackMe Exploitation Task</h1>

        <div class="section">
            <h2>Task 1: Sensitive Data Exposure</h2>
            <p>Inspect the page source code to find hidden information, such as a password. Right-click and select "View Page Source" to start.</p>
            <p class="hint">Hint: Developers sometimes leave sensitive data in comments.</p>
        </div>

        <div class="section">
            <h2>Task 2: HTML Injection</h2>
            <p>Enter your name below. The input is displayed on the page, but it’s not sanitized. Can you inject HTML to create a malicious link?</p>
            <form id="nameForm">
                <label for="nameInput">Your Name:</label><br>
                <input type="text" id="nameInput" placeholder="Enter your name">
                <button type="submit">Submit</button>
            </form>
            <div id="output"></div>
            <p class="hint">Hint: Try injecting HTML like &lt;a href="http://hacker.com"&gt;Click Me&lt;/a&gt;.</p>
        </div>

        <div class="section">
```

Task 2: HTML Injection

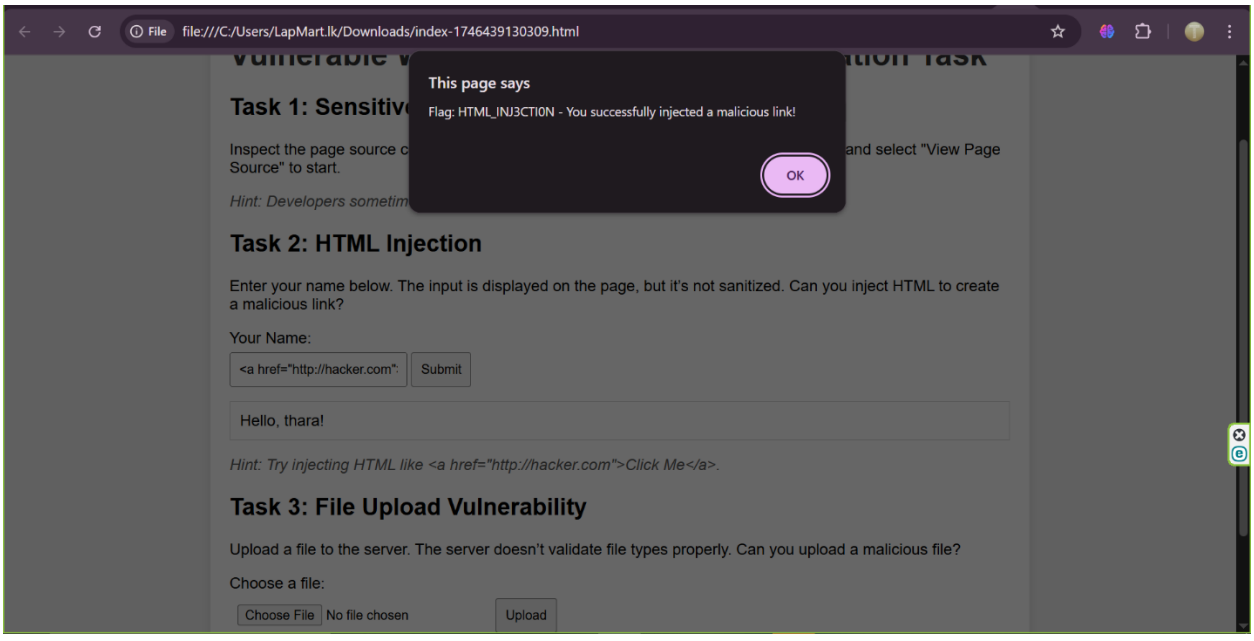
Enter your name below. The input is displayed on the page, but it’s not sanitized. Can you inject HTML to create a malicious link?

Your Name:

thara Submit

Hello, thara!

Hint: Try injecting HTML like Click Me.



Task 3: File Upload Vulnerability

Upload a file to the server. The server doesn't validate file types properly. Can you upload a malicious file?

Choose a file:

Choose File shell.php.jpg Upload

File uploaded: shell.php.jpg
Flag: FILE_UPLOAD_VULN - You uploaded a potentially malicious file!

Hint: Try uploading a file with a .php extension disguised as an image (e.g., shell.php.jpg).

Room Link

Room Link: https://tryhackme.com/jr/crackingweakcspmisconfigs

Reflection

1. Personal Insights and Lessons Learned While Designing the Room

- Understanding the Limits of CSP:
  While designing the CSP challenge, I realized that many developers overestimate the protection offered by CSP. Simply applying a policy is not enough — if it includes overly permissive directives or allows scripts from compromised or untrusted domains, it can still leave the application exposed. This experience taught me how critical it is to implement strict, minimal policies.
- Creating Realistic Bypass Scenarios:
  Building a lab to demonstrate CSP bypasses helped me understand the subtle ways attackers can exploit relaxed directives, such as unsafe-inline or script-src with overly broad whitelists. Crafting scenarios where a supposedly secure CSP could still be circumvented was both challenging and rewarding.
- Thinking Like an Attacker:
  Developing the lab deepened my attacker mindset. I had to consider how a user might interact with the page, how a malicious actor could inject or manipulate content, and how different browser behaviors might influence exploitation. This perspective is essential not only for breaking things but also for building more secure systems.
- Technical Deployment Skills:
  Setting up the environment required configuring a Flask-based web app with a vulnerable CSP policy, ensuring it served the correct headers, and testing it across browsers. I also refined my skills in deploying services on Ubuntu, using systemd for autostart and managing port binding and access within TryHackMe's VM infrastructure.

2. How This Room Contributes to the Learning Community

- Hands-On CSP Misconfiguration Exploration:
  This room offers a practical way to explore how small missteps in CSP implementation can lead to significant vulnerabilities. It moves beyond theory and allows learners to experiment in a safe, controlled environment.

- Promoting Secure Development Practices:
  By highlighting the consequences of weak CSP rules, the room reinforces the need for careful configuration. Learners gain a better appreciation for why directives should be as restrictive as possible and how misconfigurations can backfire.

- Progressive Learning Experience:
  The challenge is structured to guide users through discovery and exploitation without being overwhelming. This makes it accessible to beginners while still engaging for those with intermediate skills.

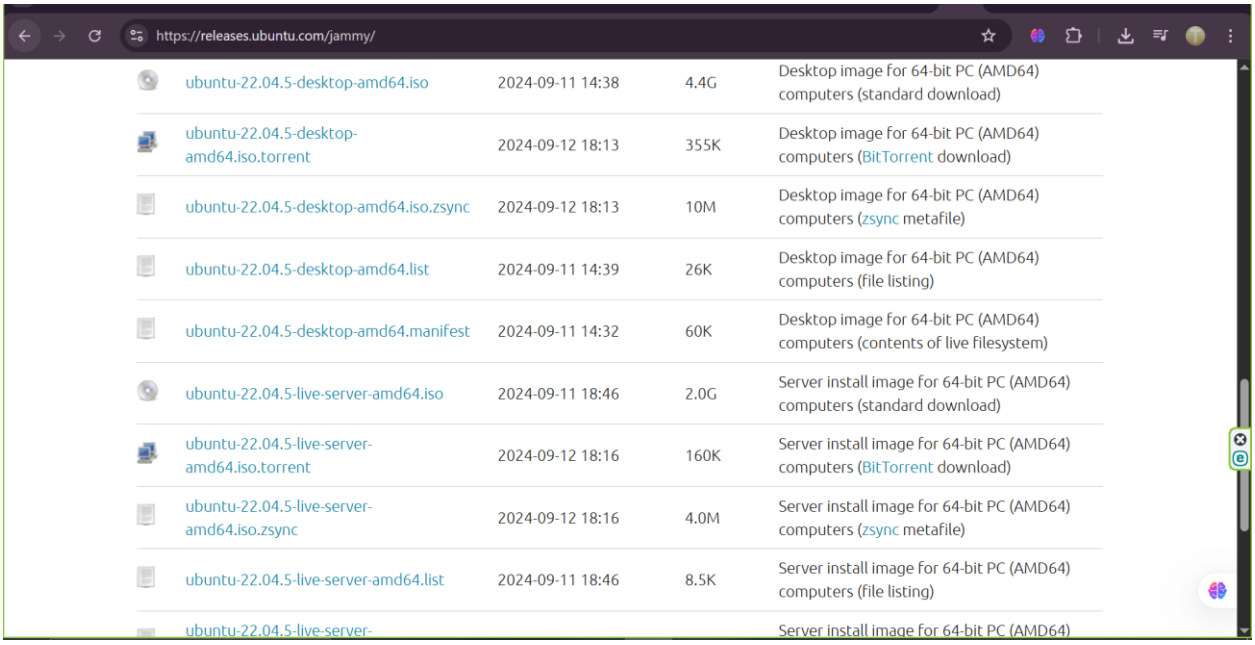
- Encouraging Creative Problem Solving:
  Users are prompted to find non-obvious paths, such as leveraging whitelisted domains or crafting specific payloads that slip through loose policies. This nurtures critical thinking and creative exploitation techniques.

- Real-World Relevance:
  CSP misconfigurations are a common issue in real-world bug bounty programs and pentests. By practicing in this environment, learners develop transferable skills for recognizing and responsibly reporting similar issues in production systems.

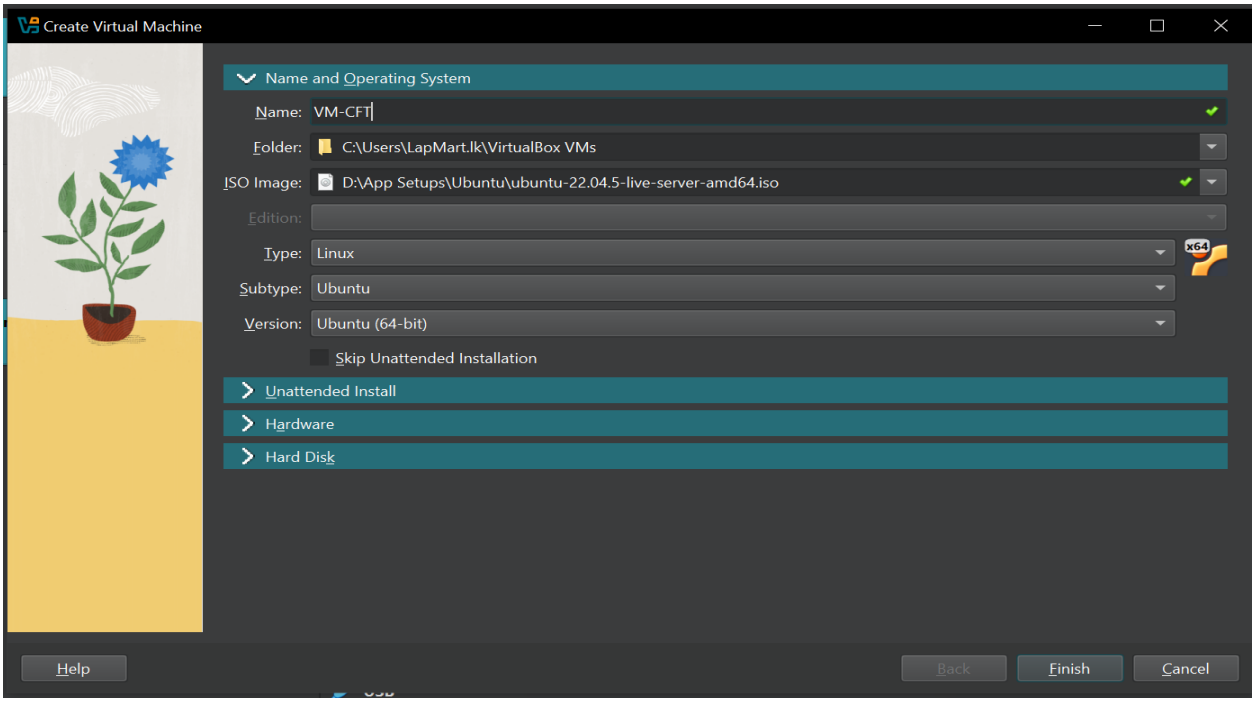
Workings

Create Virtual Machine

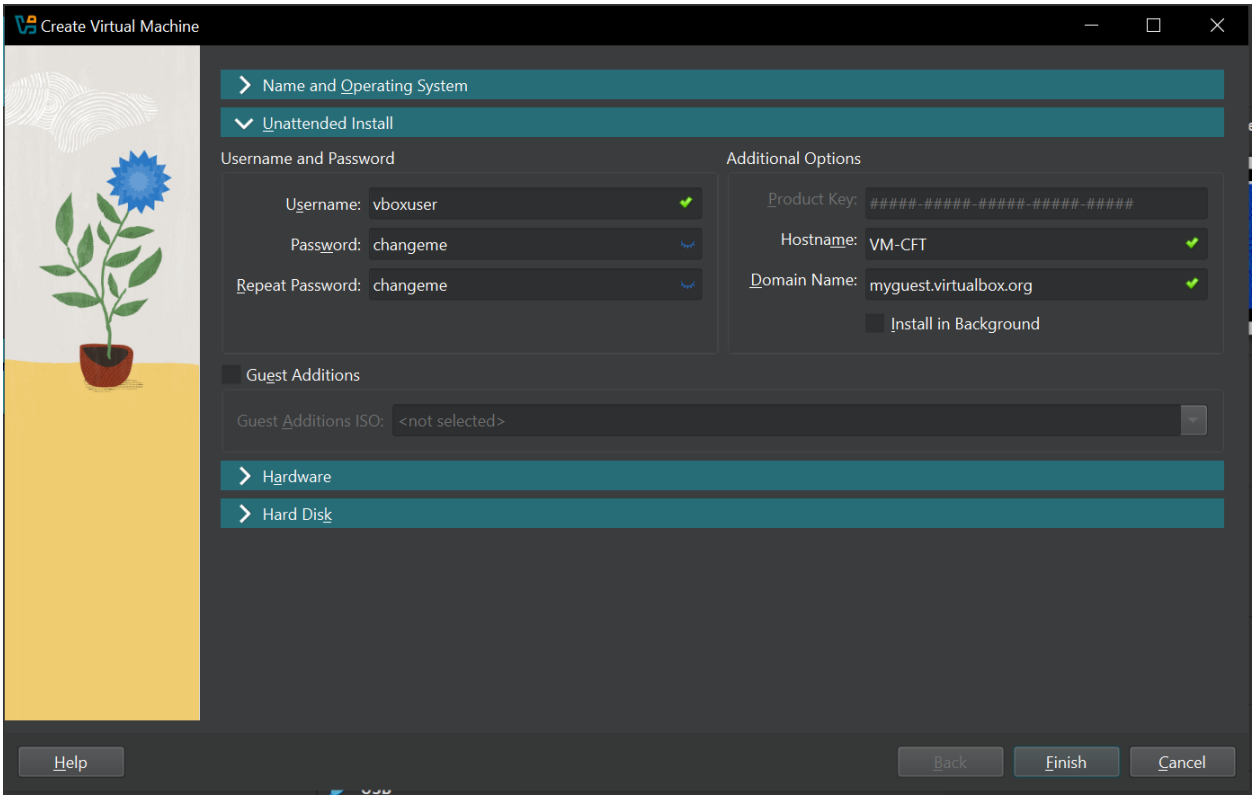
- Select the correct type and version, here I select *ubuntu-22.04.5-live-server-amd64.iso.*



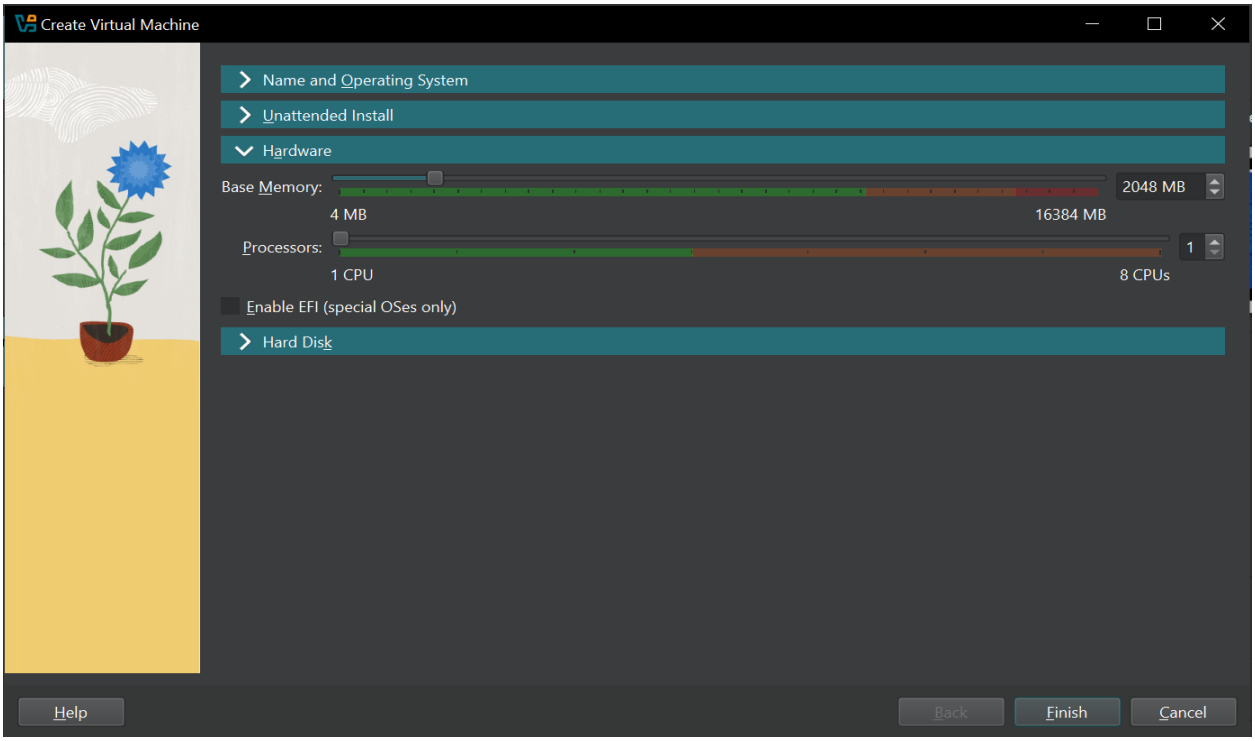
- Click ‘New’ button to open a dialog. Type a name for the new virtual machine.



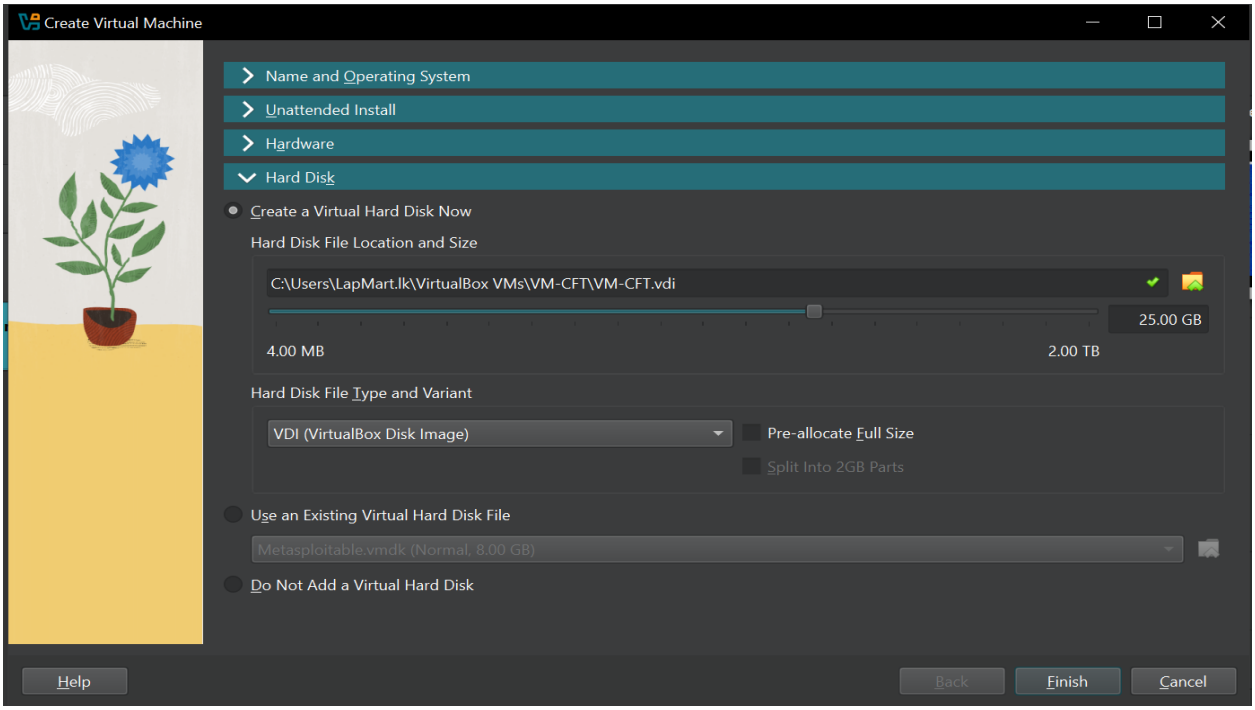
- Give username password and next



- Allocate the memory and CPU (2GB,2cpu recommended).

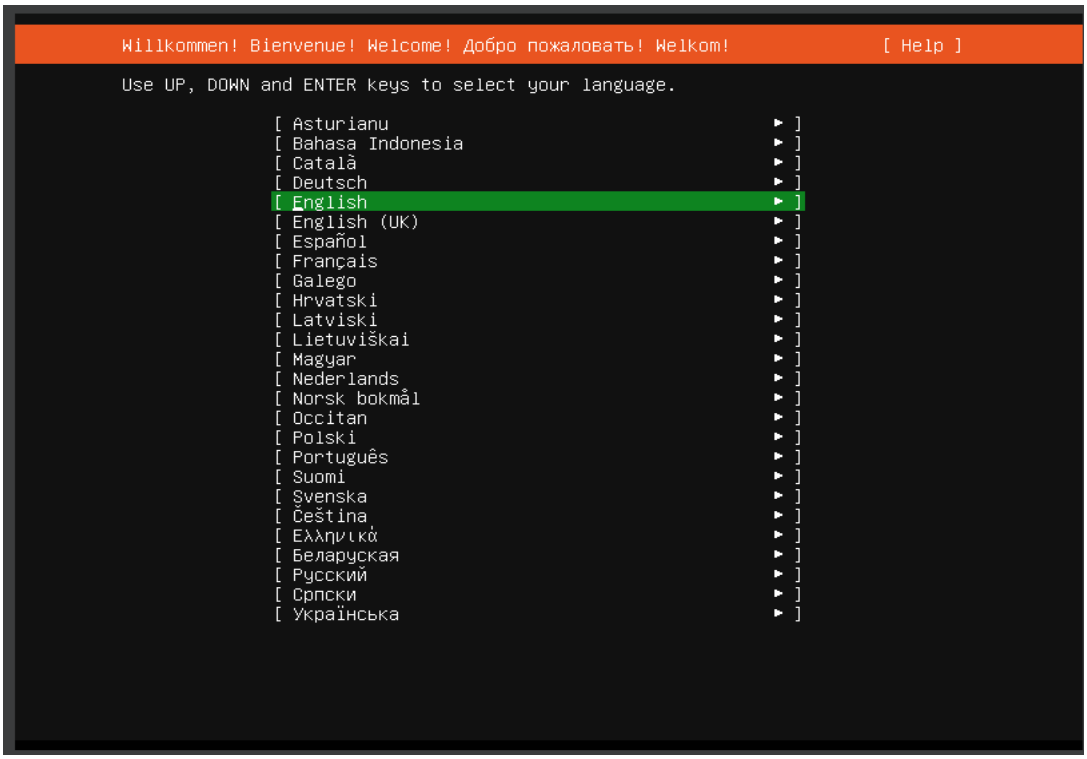


- Give Hard space as 20 GB and finish.

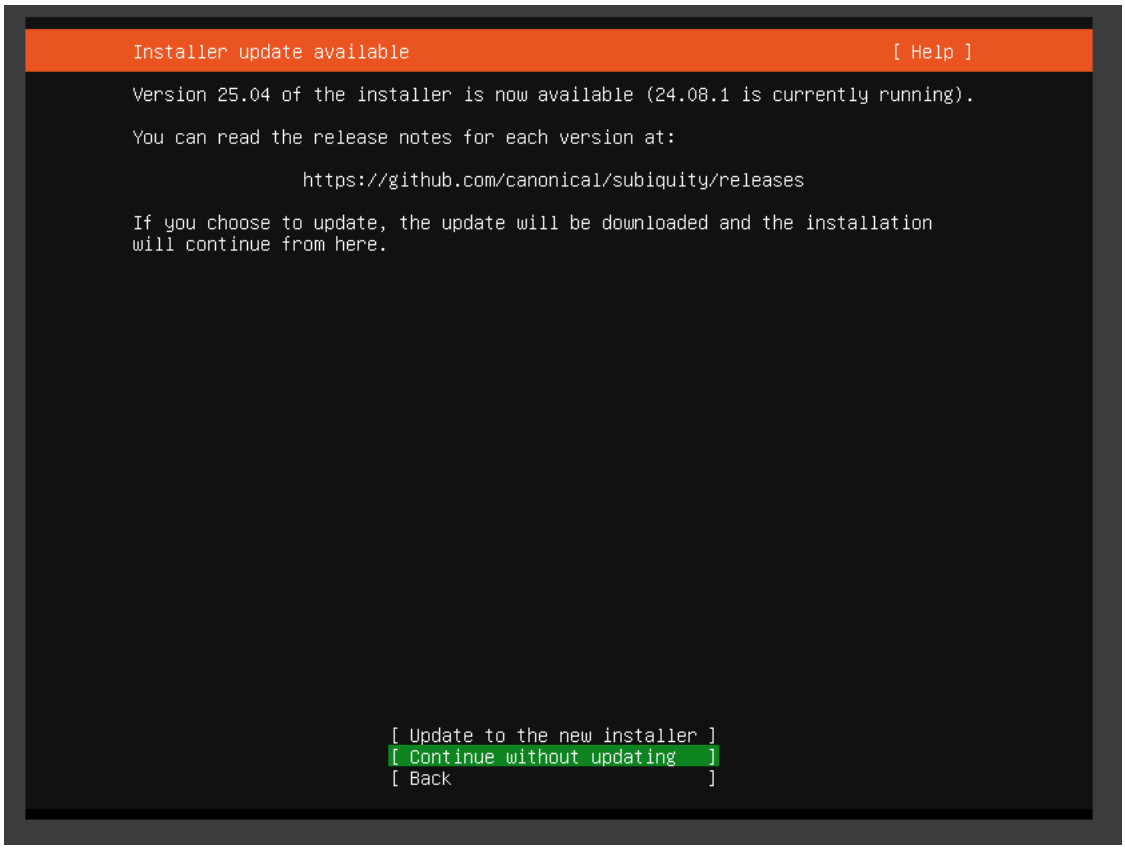


Ubuntu Server Installation

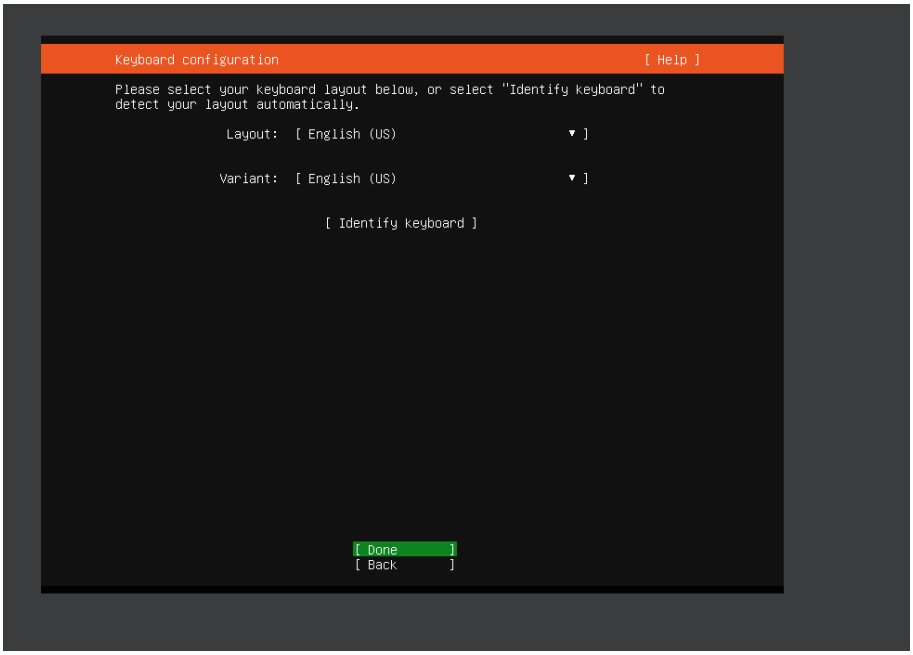
- We will first see this window where a language needs to be selected.

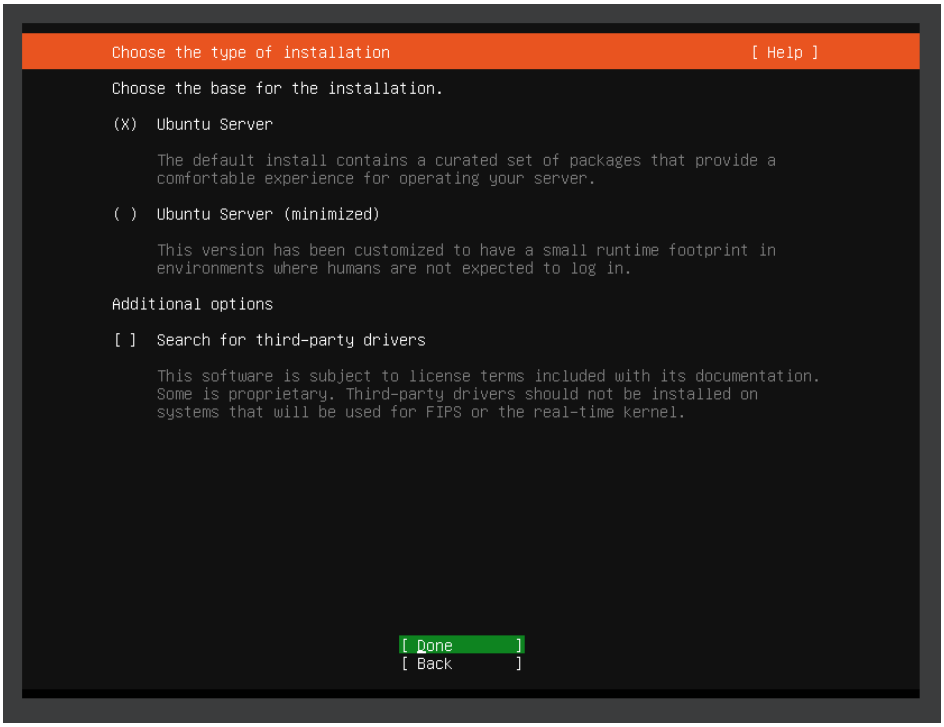


- Press Enter and we will see the installer update available. But let's continue without updating it.

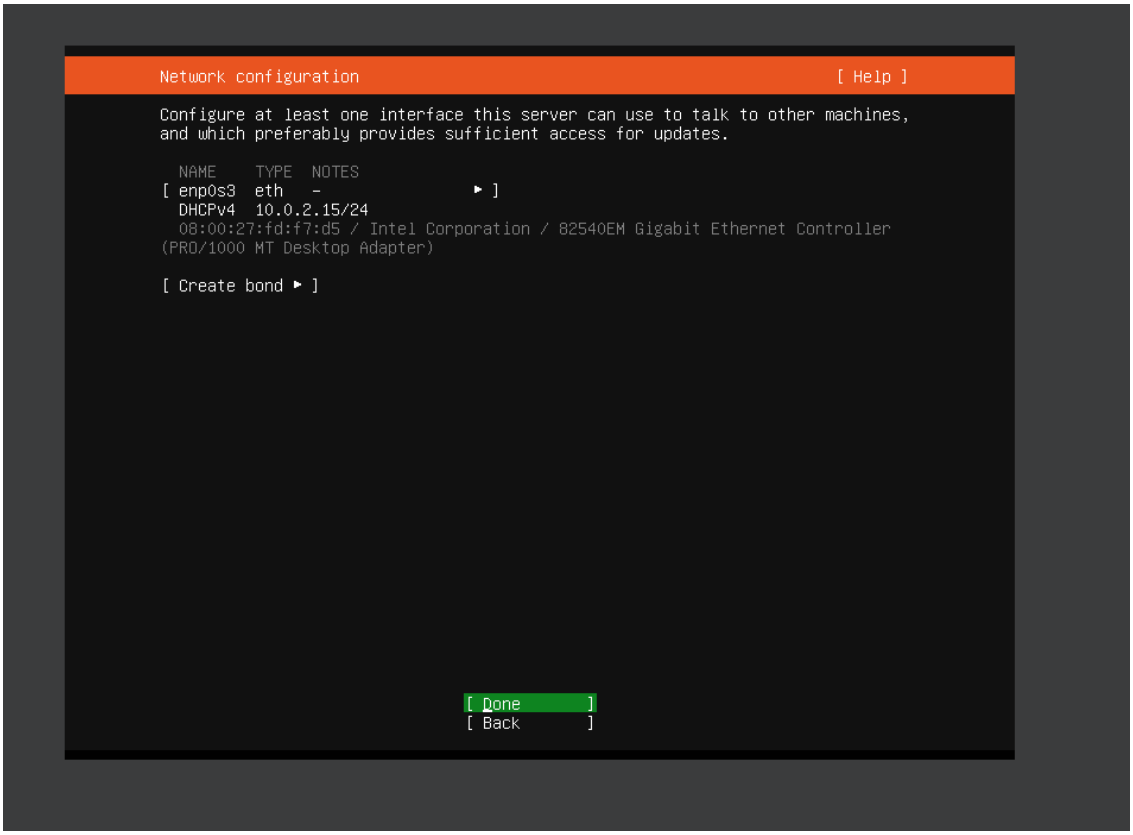


- Next will come Keyboard Configuration. Leave it as it is.

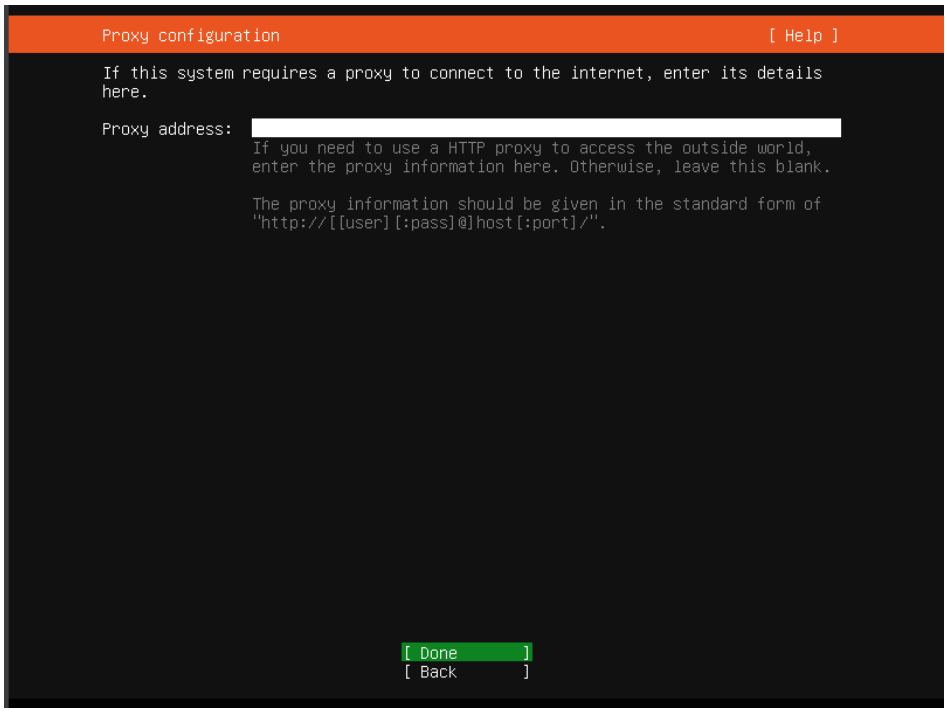


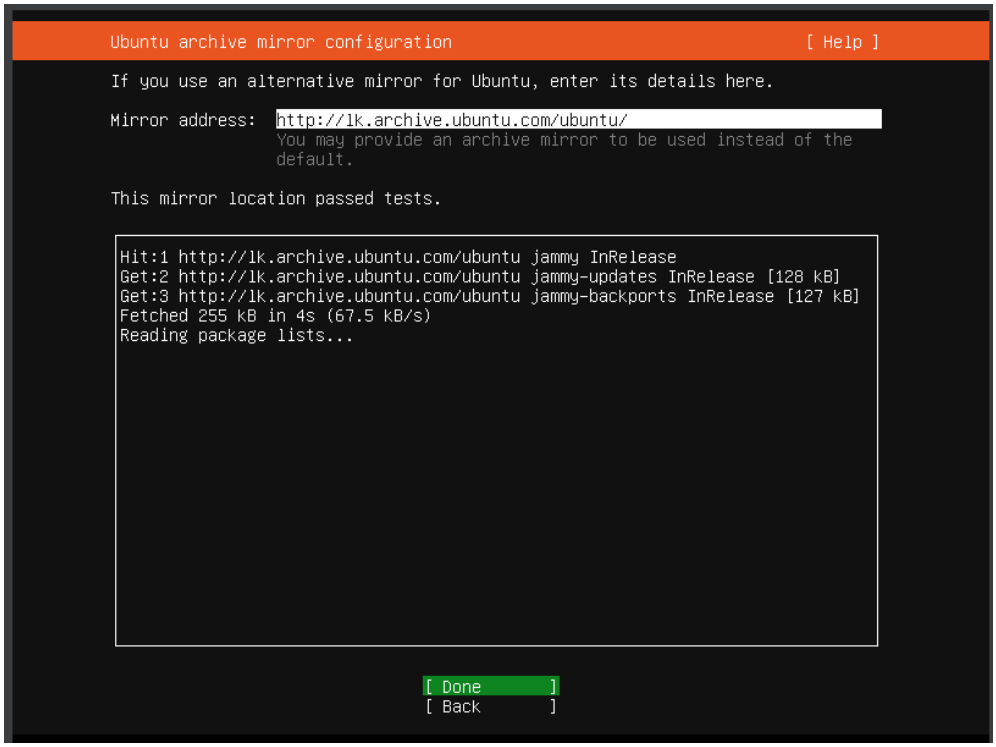


- Next, we will see the Network connections window. Let the server use DHCP to self-assign IP address to the machine.

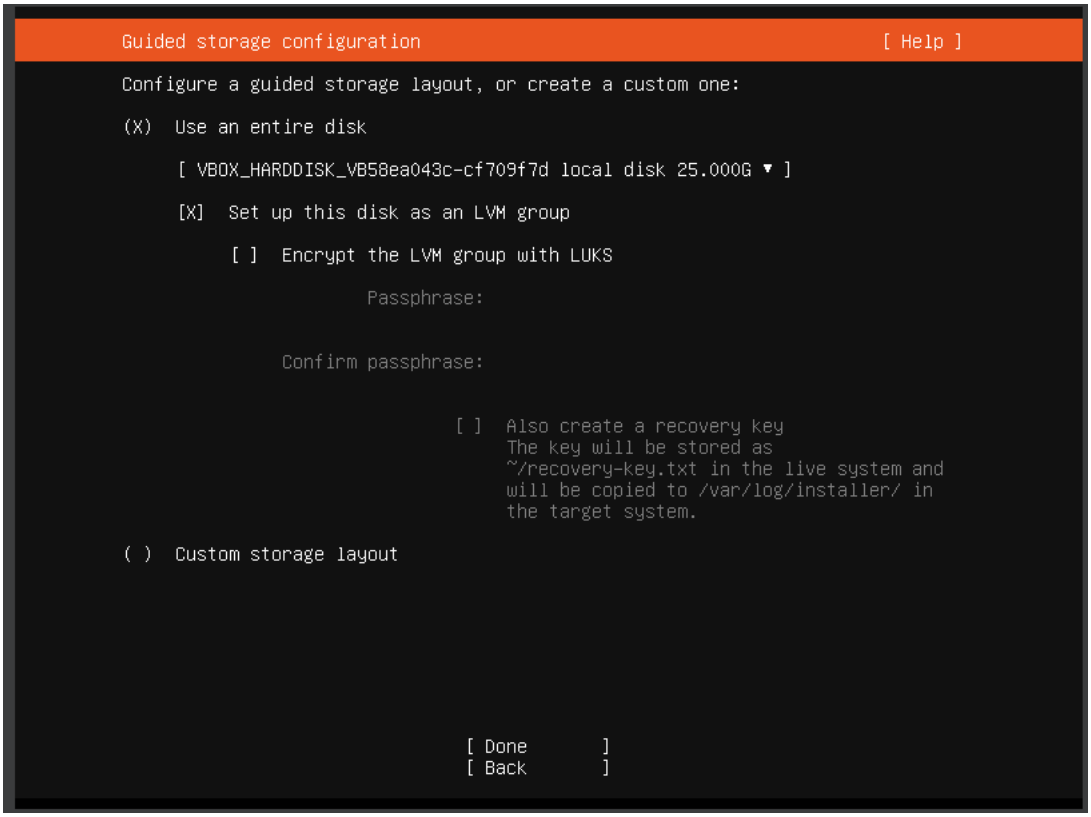


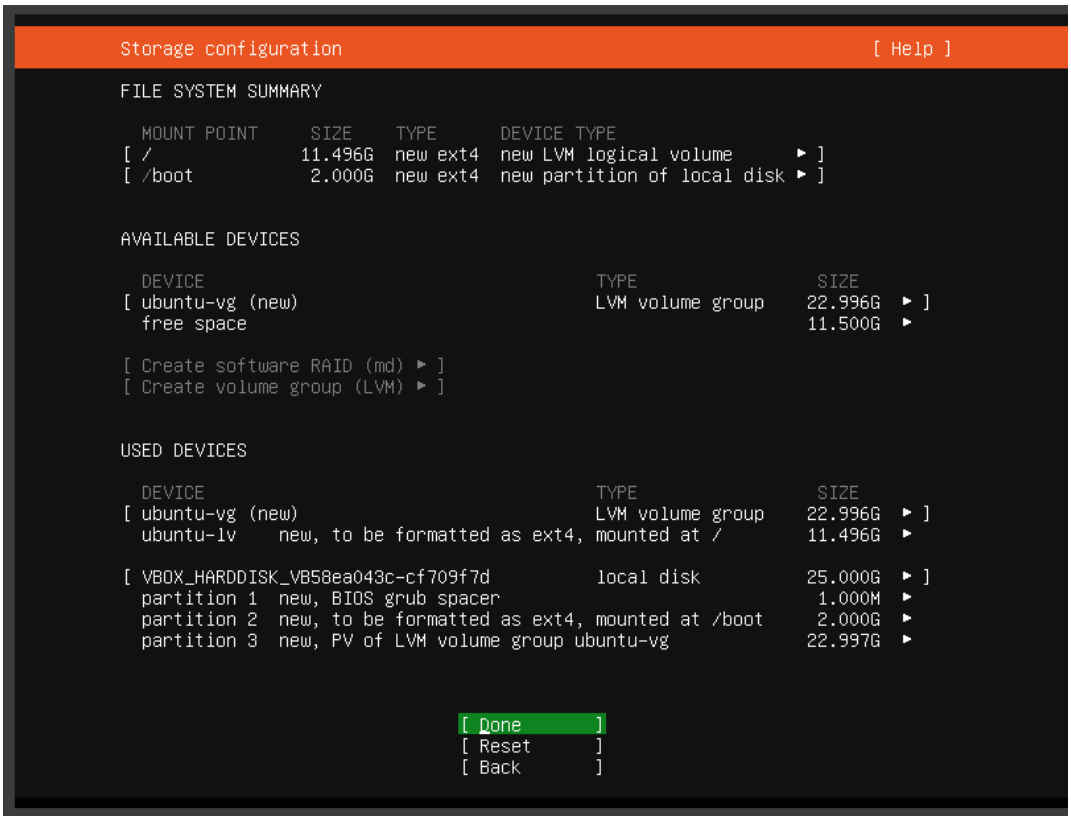
- Press Enter and now we will be presented to configure proxy but we'll skip it by pressing enter



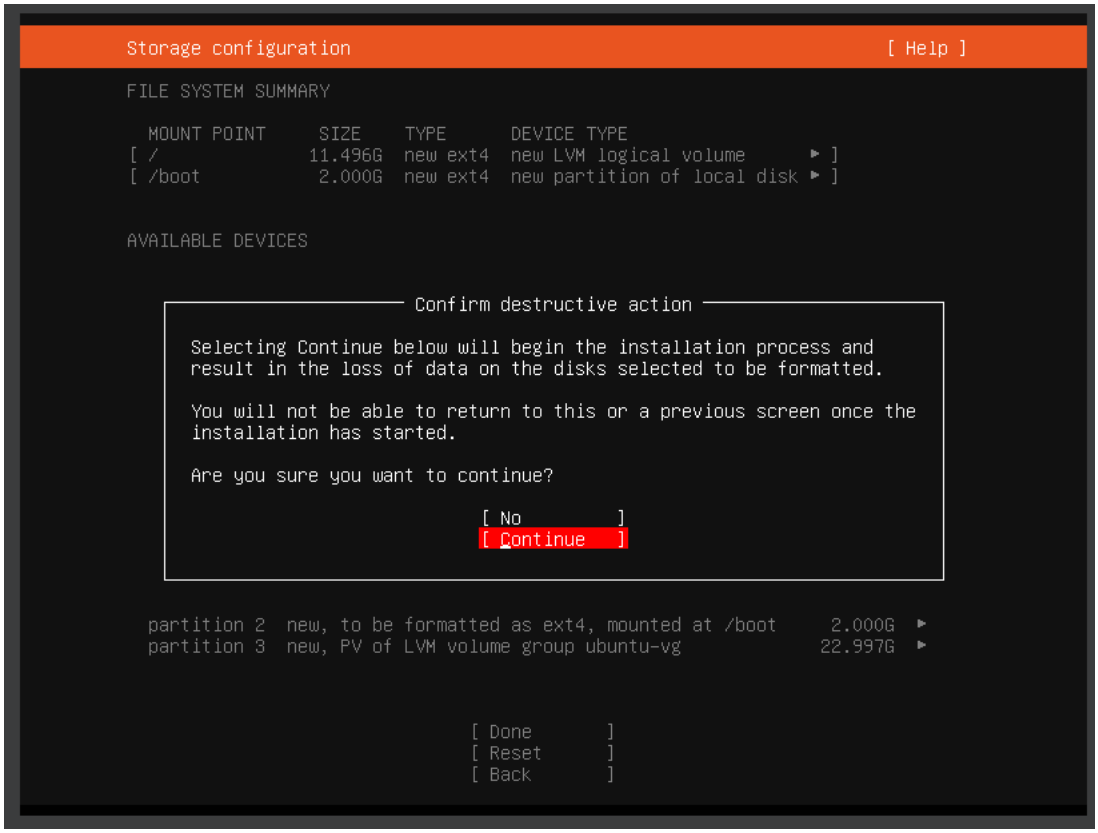


- Now, we will be presented with Storage configs. Use the entire disk for our box.

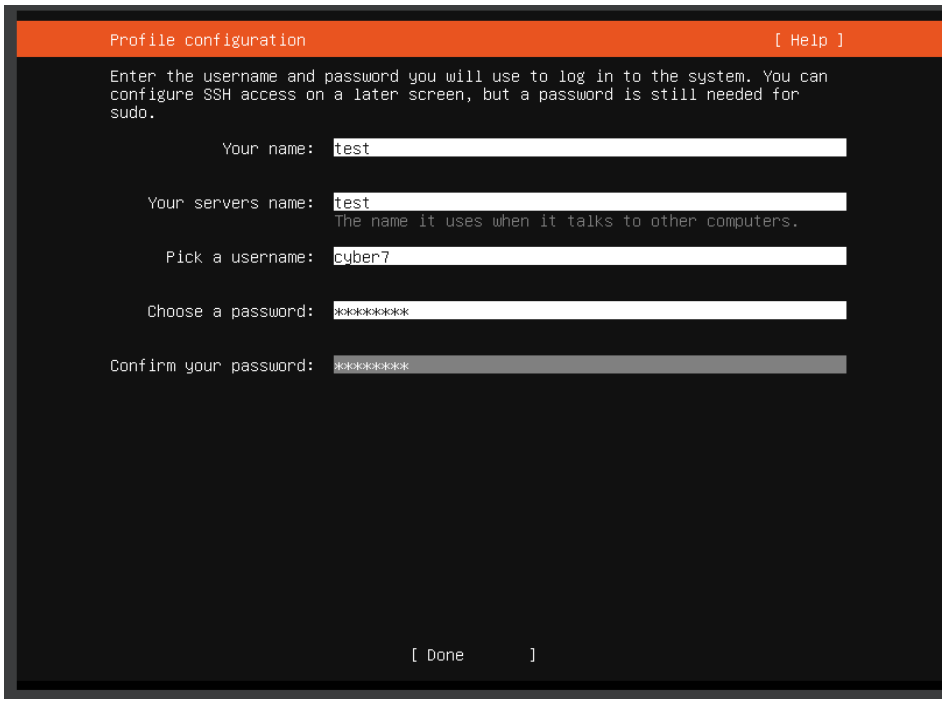




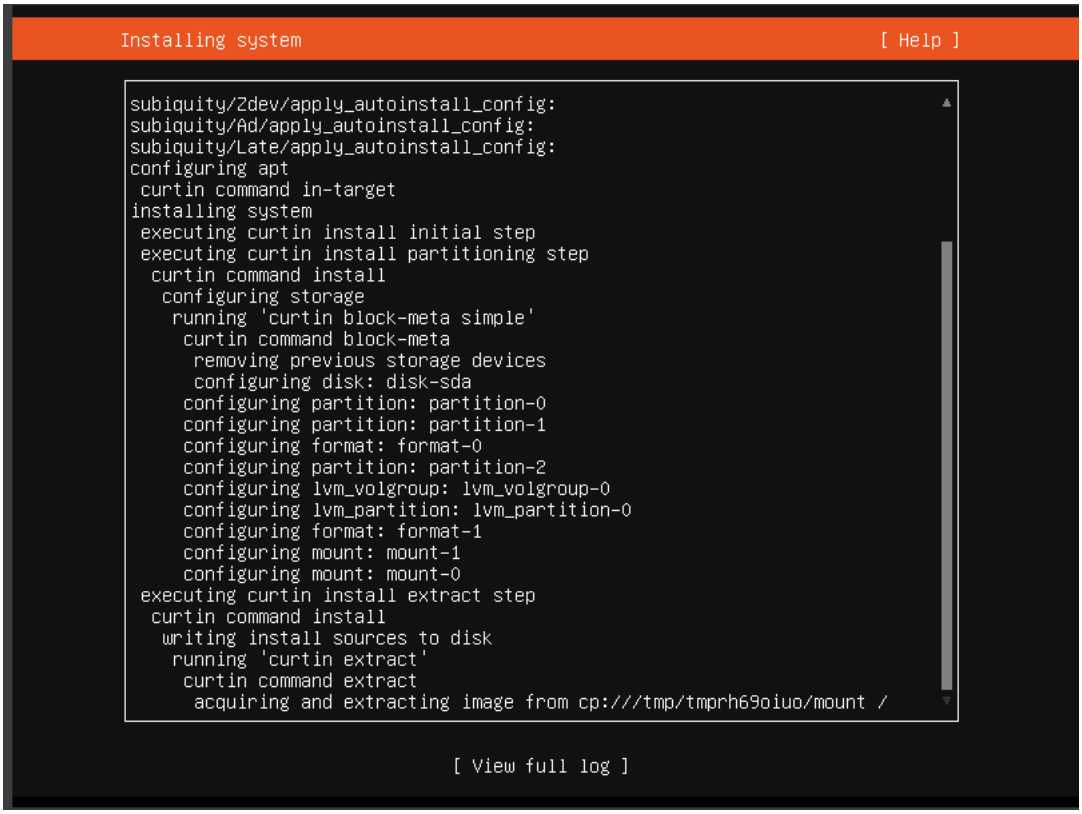
- Press Continue

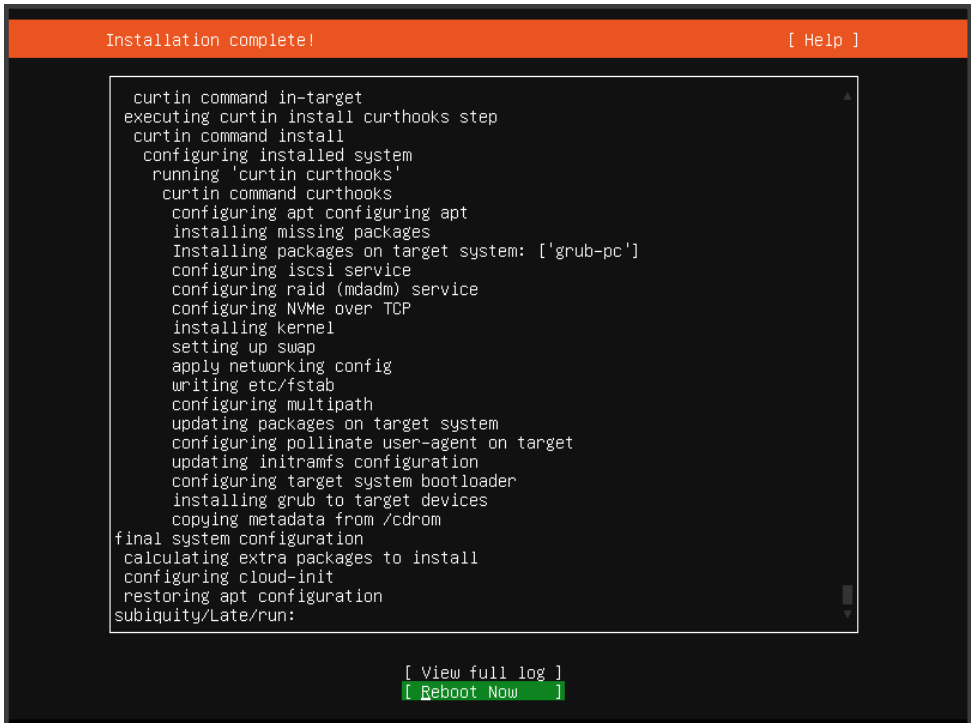


- Next it will ask to enter the device name and a username and it's corresponding password.



- Press Enter after providing name, username and password. Then it'll ask us if we want to install **OpenSSH Server**, and since we want to, so we'll select it by taking cursor to the area and pressing ***spacebar***,





- Pressing Enter for the last time to finish with this configuration and ubuntu will start the installation process.
- After getting to this point, you will get an option to **Reboot Now**. Go ahead and reboot this machine and now our machine is ready.

Implementation of Challenges

- Open the Ubuntu virtual machine
- Create the directories

```
cyber7@test:~$ sudo mkdir -p /var/www/csp-lab/templates_
```

```
cyber7@test:~$ tree /var/www/csp-lab/
/var/www/csp-lab/
├── app.py
└── templates
    ├── index.html
    └── static

2 directories, 2 files
cyber7@test:~$ _
```

- Install Python3, Flask

```
cyber7@test:~$ sudo apt update
[sudo] password for cyber7:
Hit:1 http://archive.ubuntu.com/ubuntu jammy InRelease
Get:2 http://archive.ubuntu.com/ubuntu jammy-updates InRelease [128 kB]
Get:3 http://security.ubuntu.com/ubuntu jammy-security InRelease [129 kB]
Get:4 http://archive.ubuntu.com/ubuntu jammy-backports InRelease [127 kB]
Get:5 http://archive.ubuntu.com/ubuntu jammy/main amd64 c-n-f Metadata [30.3 kB]
Get:6 http://archive.ubuntu.com/ubuntu jammy/restricted amd64 c-n-f Metadata [488 B]
Get:7 http://archive.ubuntu.com/ubuntu jammy/universe amd64 c-n-f Metadata [286 kB]
Get:8 http://security.ubuntu.com/ubuntu jammy-security/main amd64 Packages [2,266 kB]
Get:9 http://archive.ubuntu.com/ubuntu jammy/multiverse amd64 c-n-f Metadata [8,372 B]
Get:10 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 Packages [2,511 kB]
Get:11 http://archive.ubuntu.com/ubuntu jammy-updates/main Translation-en [410 kB]
35% [10 Packages store 0 B] [11 Translation-en 95.6 kB/410 kB 23%] [8 Packages 1,300 kB/2,266 kB 57_
```

```
cyber7@test:~$ python3 --version
Python 3.10.12
cyber7@test:~$
```

```
No VM guests are running outdated hypervisor (qemu) binaries on this host.
^[[Bcyber7@test:~$ pip3 install flask_
```

```
 systemctl restart unattended-upgrades.service
 systemctl restart user@1000.service

No containers need to be restarted.

No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.
^[[Bcyber7@test:~$ pip3 install flask
Defaulting to user installation because normal site-packages is not writeable
Collecting flask
  Downloading flask-3.1.0-py3-none-any.whl (102 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 103.0/103.0 KB 40.7 kB/s eta 0:00:00
Collecting Werkzeug>=3.1
  Downloading werkzeug-3.1.3-py3-none-any.whl (224 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 224.5/224.5 KB 380.0 kB/s eta 0:00:00
Collecting blinker>=1.9
  Downloading blinker-1.9.0-py3-none-any.whl (8.5 kB)
Collecting Jinja2>=3.1.2
  Downloading jinja2-3.1.6-py3-none-any.whl (134 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 134.9/134.9 KB 517.5 kB/s eta 0:00:00
Collecting itsdangerous>=2.2
  Downloading itsdangerous-2.2.0-py3-none-any.whl (16 kB)
Collecting click>=8.1.3
  Downloading click-8.1.8-py3-none-any.whl (98 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 98.2/98.2 KB 574.2 kB/s eta 0:00:00
Requirement already satisfied: MarkupSafe>=2.0 in /usr/lib/python3/dist-packages (from Jinja2>=3.1.2
->flask) (2.0.1)
Collecting MarkupSafe>=2.0
  Downloading MarkupSafe-3.0.2-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (20 kB)
Installing collected packages: MarkupSafe, itsdangerous, click, blinker, Werkzeug, Jinja2, flask
  WARNING: The script flask is installed in '/home/cyber7/.local/bin' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-s
cript-location.
Successfully installed Jinja2-3.1.6 MarkupSafe-3.0.2 Werkzeug-3.1.3 blinker-1.9.0 click-8.1.8 flask-
3.1.0 itsdangerous-2.2.0
cyber7@test:~$ _
```

```
cyber7@test:~$ cd /var/www/csp-lab/
cyber7@test:/var/www/csp-lab$ sudo python3 app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WS
GI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:80
 * Running on http://192.168.8.135:80
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 380-534-068
192.168.8.181 - - [27/Apr/2025 23:54:25] "GET / HTTP/1.1" 200 -
192.168.8.181 - - [27/Apr/2025 23:54:25] "GET /favicon.ico HTTP/1.1" 404 -
```

```
cyber7@test:~$ cd /var/www/csp-lab/
cyber7@test:/var/www/csp-lab$ sudo nano /var/www/csp-lab/app.py
```

```
  GNU nano 6.2                          /var/www/csp-lab/app.py
from flask import Flask, request, render_template, Response

app = Flask(_name_)

@app.route("/", methods=["GET", "POST"])
def home():
        return Response(render_template("index.html"), headers={
                "Content-Security-Policy": "default-src 'self'; script-src 'self' 'unsafe-inline';
        })

if _name_ == "_main_":
        app.run(host="0.0.0.0", port=80)
```

- Developing the vulnerable site

```
cyber7@test:~$ cd /var/www/csp-lab/templates/
cyber7@test:/var/www/csp-lab/templates$
```

```
cyber7@test:/var/www/csp-lab/templates$ sudo nano index.html
```

```
  GNU nano 6.2                          index.html *
<!DOCTYPE html>
<html lang="en">
<head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, intial-scale=1.0">
        <title>CSP Misconfigurations Demo</title>
        <style>
                body{
                        font-family: Arial, sans-serif;
                        max-width: 800px;
                        margin: 0 auto;
                        padding: 20px;
                        background-color: #f0f0f0;
                }
                h1{
                        color: #333;
                        text-align: center;
                }
                .challenge{
                        background-color: white;
                        padding: 20px;
                        margin-bottom: 20px;
                        border-radius: 8px;
                        box-shadow: 0 2px 4px rgba(0,0,0,0.1);
                }
                .challenge h2{
                        color: #2c3e50;
                }
                textarea{
                        width: 100%;
                        height: 100px;
                        margin-bottom: 10px;
                        padding: 10px;

^G Help      ^O Write Out  ^W Where Is   ^K Cut       ^T Execute   ^C Location  M-U Undo
^X Exit      ^R Read File  ^\ Replace    ^U Paste     ^J Justify   ^/ Go To Line M-E Redo
```

```
 GNU nano 6.2                                  index.html *
                textarea{
                        width: 100%;
                        height: 100px;
                        margin-bottom: 10px;
                        padding: 10px;
                        border: 1px solid #ccc;
                        border-radius: 4px;
                }
                button{
                        padding: 10px 20px;
                        background-color: #3498db;
                        color: white;
                        border: none;
                        border-radius: 4px;
                        cursor: pointer;
                }
                #comments{
                        margin-top: 20px;
                        padding: 10px;
                        border: 1px solid #ddd;
                        border-radius: 4px;
                        min-height: 100px;
                }
                .config-links{
                        margin-bottom: 20px;
                }
                .config-links a{
                        margin-right: 10px;
                        text-decoration: none;
                        color: #3498db;
                }
                .config-links a:hover{
                        color: #2980b9;

^G Help       ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo
^X Exit       ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^/ Go To Line M-E Redo
```

```
 GNU nano 6.2                                  index.html *
                .config-links a:hover{
                        color: #2980b9;
                }
        </style>
</head>
<body>
        <h1>CSP Misconfigurations Demo</h1>
        <div class="config-links">
                <a href="/unsafe-inline">Challenge 1: Unsafe-inline</a>
                <a href="/permissive">Challenge 2: Permissive script-src</a>
                <a href="/nonce">Challenge 3: Missing Nonce Validation</a>
        </div>
        <div class="challenge">
                <h2>Comment Section</h2>
                <p>Enter a comment below. Try to inject XSS Payloads!</p>
                <textarea id="commentInput" placeholder="Enter your comment..."></textarea>
                <button onclick="submitComment()">Submit Comment</button>
                <div id="comments"></div>
        </div>

        <script>
                function submitComment(){
                        const input = document.getElementById('commentInput').value;
                        const commentsDiv = document.getElementById('comments');
                        const commentElement = document.createElement('div');
                        commentElement.innerHTML = input;
                        commentsDiv.appendChild(commentElement);
                        document.getElementById('commentInput').value = '';
                }
        </script>
</body>
</html>
_

^G Help       ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo
^X Exit       ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^/ Go To Line M-E Redo
```

Downloaded File Challenge 1

- Create a project file

```
┌──(kali㉿kali)-[~]
└─$ mkdir csp-misconfig-challenge
cd csp-misconfig-challenge
```

- Create a nano file to create a web page

```
┌──(kali㉿kali)-[~/csp-misconfig-challenge]
└─$ nano index.html
```

```
  GNU nano 8.2                                                    index.html *
  </style>
</head>
<body>
  <h1>Can You Bypass the CSP?</h1>
  <p>Inject your payload and try to make the flag appear.</p>
  <input type="text" id="payload" placeholder="Enter your CSP bypass payload">
  <br>
  <button onclick="runPayload()">Run Payload</button>

  <div id="result"></div>

  <script>
    function runPayload() {
      const input = document.getElementById("payload").value;
      document.getElementById("result").innerHTML = input;
    }
  </script>

  <div class="hint">
    Hint: The CSP allows scripts from <code>https://evil.tryhackme.site</code>
  </div>
</body>
</html>
```

- Create a JavaScript file that would simulate an attack-controlled domain

```
┌──(kali㉿kali)-[~/csp-misconfig-challenge]
└─$ nano payload.js
```

```
File  Actions  Edit  View  Help
  GNU nano 8.2                                                    payload.js *
alert("THM{csp_bypassed_flag}");
```

Downloaded File Challenge 2

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Vulnerable Web App - TryHackMe Task</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 40px;
            background-color: #f0f0f0;
        }
        .container {
            max-width: 800px;
            margin: 0 auto;
            background-color: white;
            padding: 20px;
            border-radius: 8px;
            box-shadow: 0 0 10px rgba(0,0,0,0.1);
        }
        h1 {
            color: #333;
        }
        .section {
            margin-bottom: 20px;
        }
```

htdocs/index.html

```
            margin-bottom: 20px;
        }
        input, button {
            padding: 8px;
            margin: 5px 0;
        }
        #output {
            margin-top: 10px;
            padding: 10px;
            border: 1px solid #ddd;
        }
        .hint {
            color: #555;
            font-style: italic;
        }
        <!-- Hidden comment with a clue for users -->
        <!-- Hint: Check the page source for sensitive information. Password: tryhackme123 -->
    </style>
</head>
<body>
    <div class="container">
        <h1>Vulnerable Web App - TryHackMe Exploitation Task</h1>

        <div class="section">
            <h2>Task 1: Sensitive Data Exposure</h2>
            <p>Inspect the page source code to find hidden information, such as a password. Right-click and select "View Page Source" to start.</p>
```

htdocs/index.html

```
            <p class="hint">Hint: Developers sometimes leave sensitive data in comments.</p>
        </div>

        <div class="section">
            <h2>Task 2: HTML Injection</h2>
            <p>Enter your name below. The input is displayed on the page, but it's not sanitized. Can you inject HTML to create a malicious link?</p>
            <form id="nameForm">
                <label for="nameInput">Your Name:</label><br>
                <input type="text" id="nameInput" placeholder="Enter your name">
                <button type="submit">Submit</button>
            </form>
            <div id="output"></div>
            <p class="hint">Hint: Try injecting HTML like &lt;a href="http://hacker.com"&gt;Click Me&lt;/a&gt;.</p>
        </div>

        <div class="section">
            <h2>Task 3: File Upload Vulnerability</h2>
            <p>Upload a file to the server. The server doesn't validate file types properly. Can you upload a malicious file?</p>
            <form id="uploadForm" enctype="multipart/form-data">
                <label for="fileInput">Choose a file:</label><br>
                <input type="file" id="fileInput">
                <button type="submit">Upload</button>
            </form>
            <div id="uploadOutput"></div>
            <p class="hint">Hint: Try uploading a file with a .php extension disguised as an image (e.g., shell.php.jpg).</p>
        </div>
```

htdocs/index.html

```
72                  <button type="submit">Upload</button>
73              </form>
74              <div id="uploadOutput"></div>
75              <p class="hint">Hint: Try uploading a file with a .php extension disguised as an image (e.g., shell.php.jpg).</p>
76          </div>
77      </div>
78
79      <script>
80          // Task 2: HTML Injection
81          document.getElementById('nameForm').addEventListener('submit', function(e) {
82              e.preventDefault();
83              const name = document.getElementById('nameInput').value;
84              // Vulnerable: Directly injecting user input without sanitization
85              document.getElementById('output').innerHTML = `Hello, ${name}!`;
86              // Check for successful HTML injection
87              if (name.includes('hacker.com')) {
88                  alert('Flag: HTML_INJ3CTI0N - You successfully injected a malicious link!');
89              }
90          });
91
92          // Task 3: File Upload Simulation
93          document.getElementById('uploadForm').addEventListener('submit', function(e) {
94              e.preventDefault();
95              const fileInput = document.getElementById('fileInput');
96              if (fileInput.files.length > 0) {
97                  const fileName = fileInput.files[0].name.toLowerCase();
```

htdocs/index.html

References

1. "How to make our own CTF Challenge with ease.", Medium, 2022. [Online]. Available: https://infosecwriteups.com/how-to-make-our-own-ctf-challenge-with-ease-6b15f76865b5.
[Accessed: 5-Desc- 2022].

2. "Create a CTF: Capture the Flag box!", Reddit.com, 2022. [Online]. Available: https://www.reddit.com/r/cybersecurity/comments/fqjkg1/create_a_ctf_capture_the_flag_box/.
[Accessed: 5-Desc- 2022].

3. "OWASP Top 10:2021", Owasp.org, 2022. [Online]. Available: https://owasp.org/Top10/.
[Accessed: 5-Desc- 2022].

4. https://www.imperva.com/learn/application-security/content-security-policy-csp-header/