**Cyber Security Project**

**SecureStream**

**(Your first step of defence for every downloads)**

A cybersecurity project that introduces a verification step in the browser before downloading files is an excellent initiative to enhance user safety.

A practical approach is to develop a browser extension that intercepts download requests, evaluates them against security criteria, and prompts the user for confirmation.

**🛠️ Implementation Steps**

**Create a folder name “Secure extension”.**

**Then create the following files in this folder.**

1. **Manifest.json : Create a manifest.json file to define the extension's metadata and permissions**

{

    "manifest\_version": 3,

    "name": "SecureDownload - Verify Before Download",

    "version": "1.0",

    "permissions": ["downloads", "downloads.shelf", "storage"],

    "host\_permissions": ["<all\_urls>"],

    "background": {

      "service\_worker": "background.js"

    },

    "action": {

      "default\_title": "SecureDownload"

    },

    "web\_accessible\_resources": [

      {

        "resources": ["verification.html"],

        "matches": ["<all\_urls>"]

      }

    ],

    "icons": {

      "48": "icon.png"

    }

  }

1. **background.js :**  In background.js, listen for download events and implement the verification logic. Use the chrome.downloads.onCreated event to intercept download requests

chrome.downloads.onCreated.addListener((downloadItem) => {

    chrome.downloads.pause(downloadItem.id, () => {

      chrome.windows.create({

        url: chrome.runtime.getURL(`verification.html?downloadId=${downloadItem.id}&url=${encodeURIComponent(downloadItem.url)}&filename=${encodeURIComponent(downloadItem.filename)}`),

        type: "popup",

        width: 420,

        height: 350

      });

    });

  });

  chrome.runtime.onMessage.addListener((message, sender, sendResponse) => {

    if (message.action === 'resume') {

      chrome.downloads.resume(message.downloadId);

    } else if (message.action === 'cancel') {

      chrome.downloads.cancel(message.downloadId);

    }

  });

1. **Verification. Html :** **Design a simple HTML interface for user interaction**

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <title>Download Verification</title>

  <link rel="stylesheet" href="styles.css">

</head>

<body>

  <div class="container">

    <h2>Verify File Download</h2>

    <pre id="file-info">Loading file scan...</pre>

    <div class="buttons">

      <button id="allow" disabled>Allow</button>

      <button id="block">Block</button>

    </div>

  </div>

  <script src="verification.js"></script>

</body>

</html>

1. **verification.js : Handle user interactions**

const API\_KEY = ''; // Replace with your real key

const urlParams = new URLSearchParams(window.location.search);

const downloadId = parseInt(urlParams.get('downloadId'));

const filename = decodeURIComponent(urlParams.get('filename'));

const url = decodeURIComponent(urlParams.get('url'));

const fileInfo = document.getElementById('file-info');

const allowBtn = document.getElementById('allow');

const blockBtn = document.getElementById('block');

allowBtn.disabled = true;

fileInfo.textContent = `Scanning: ${filename}\nSource: ${url}`;

// Step 1: Submit URL to VirusTotal

fetch("https://www.virustotal.com/api/v3/urls", {

  method: "POST",

  headers: {

    "x-apikey": API\_KEY,

    "content-type": "application/x-www-form-urlencoded"

  },

  body: `url=${encodeURIComponent(url)}`

})

.then(res => res.json())

.then(data => {

  const scanId = data.data.id;

  // Step 2: Poll for report

  return fetch(`https://www.virustotal.com/api/v3/analyses/${scanId}`, {

    headers: { "x-apikey": API\_KEY }

  });

})

.then(res => res.json())

.then(result => {

  const malicious = result.data.attributes.stats.malicious;

  const harmless = result.data.attributes.stats.harmless;

  if (malicious > 0) {

    fileInfo.textContent += `\n❌ Detected: ${malicious} malicious reports`;

  } else {

    fileInfo.textContent += `\n✅ File appears safe (${harmless} scanners marked it harmless)`;

  }

  allowBtn.disabled = false;

})

.catch(err => {

  fileInfo.textContent += `\n⚠️ Scan failed. Proceed with caution.`;

  allowBtn.disabled = false;

});

// Allow or block

allowBtn.onclick = () => {

  chrome.runtime.sendMessage({ action: 'resume', downloadId });

  window.close();

};

blockBtn.onclick = () => {

  chrome.runtime.sendMessage({ action: 'cancel', downloadId });

  window.close();

};

1. **styles.css**

body {

    font-family: Arial, sans-serif;

    background-color: #f0f2f5;

    margin: 0;

    padding: 0;

    display: flex;

    height: 100vh;

    align-items: center;

    justify-content: center;

  }

  .container {

    background-color: #fff;

    padding: 20px 30px;

    border-radius: 12px;

    box-shadow: 0 4px 12px rgba(0, 0, 0, 0.1);

    width: 350px;

    text-align: center;

  }

  h2 {

    color: #333;

    margin-bottom: 15px;

  }

  #file-info {

    font-size: 14px;

    background: #f9f9f9;

    padding: 10px;

    border-radius: 6px;

    text-align: left;

    white-space: pre-wrap;

    margin-bottom: 20px;

    border: 1px solid #ddd;

    color: #444;

  }

  .buttons {

    display: flex;

    justify-content: space-around;

  }

  button {

    padding: 10px 20px;

    font-size: 16px;

    border-radius: 8px;

    border: none;

    cursor: pointer;

    transition: background-color 0.3s ease;

  }

  #allow {

    background-color: #4CAF50;

    color: white;

  }

  #allow:disabled {

    background-color: #a5d6a7;

    cursor: not-allowed;

  }

  #block {

    background-color: #f44336;

    color: white;

  }

  #block:hover {

    background-color: #d32f2f;

  }

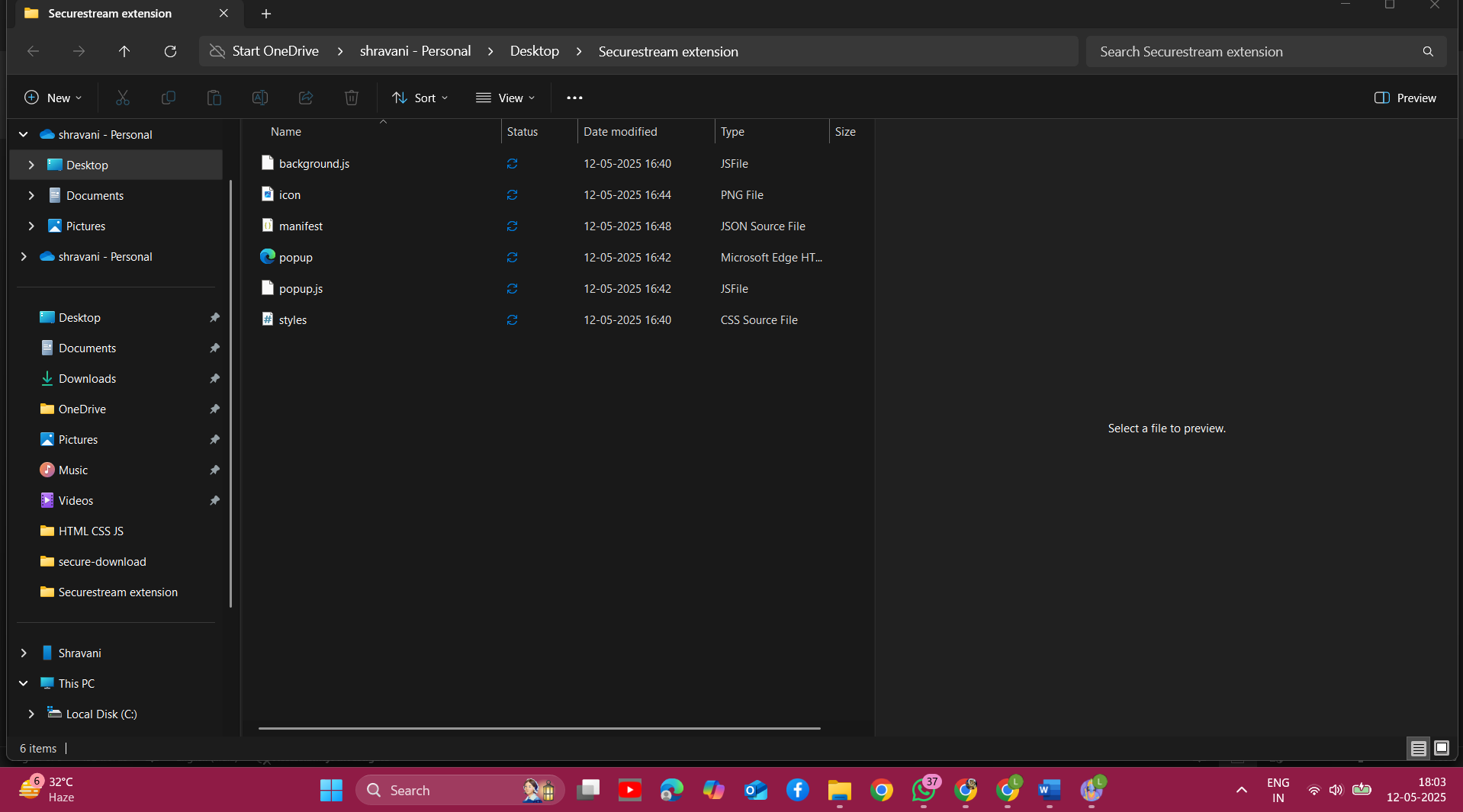
  #allow:hover:enabled {

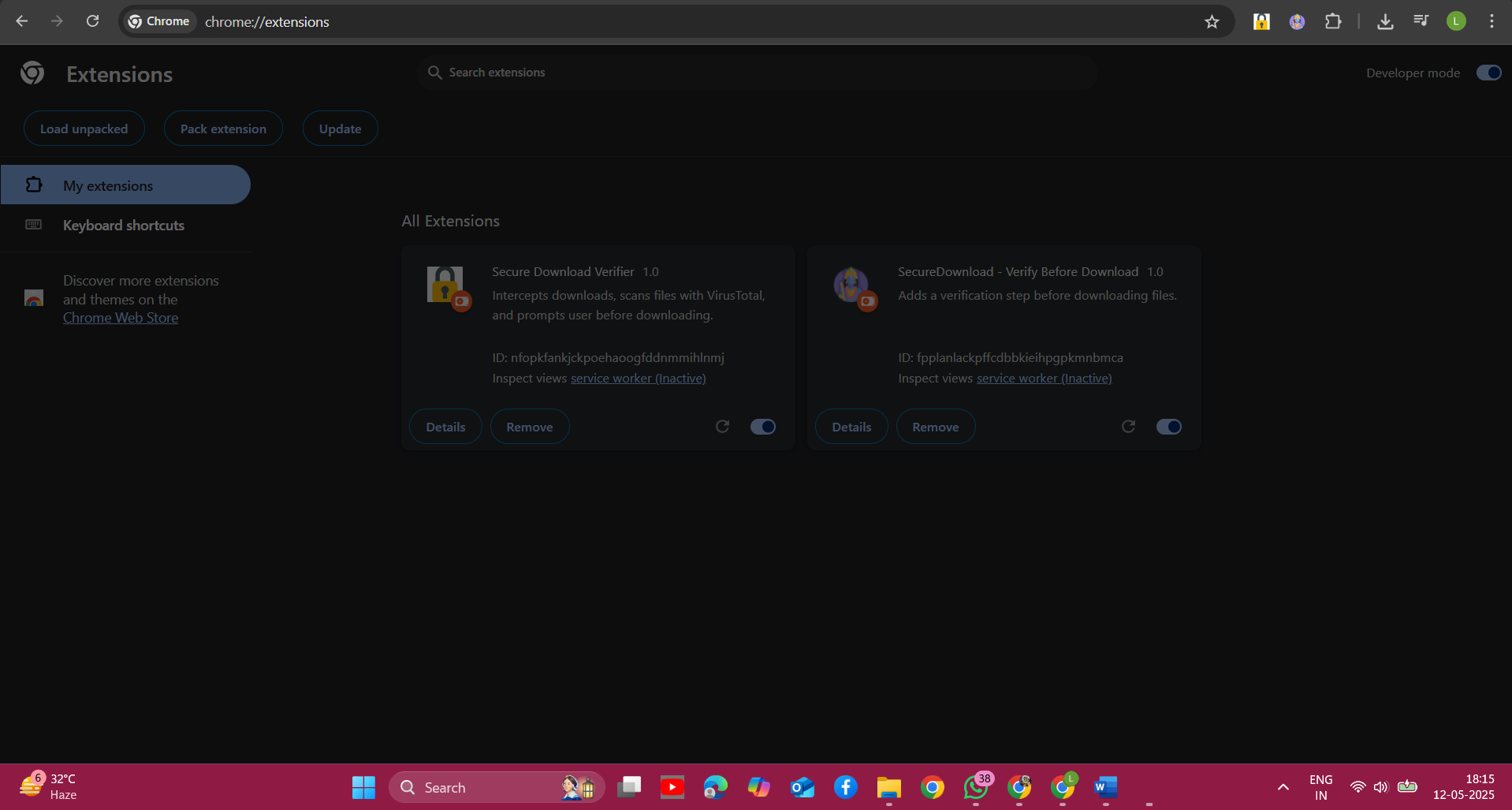
    background-color: #388e3c;

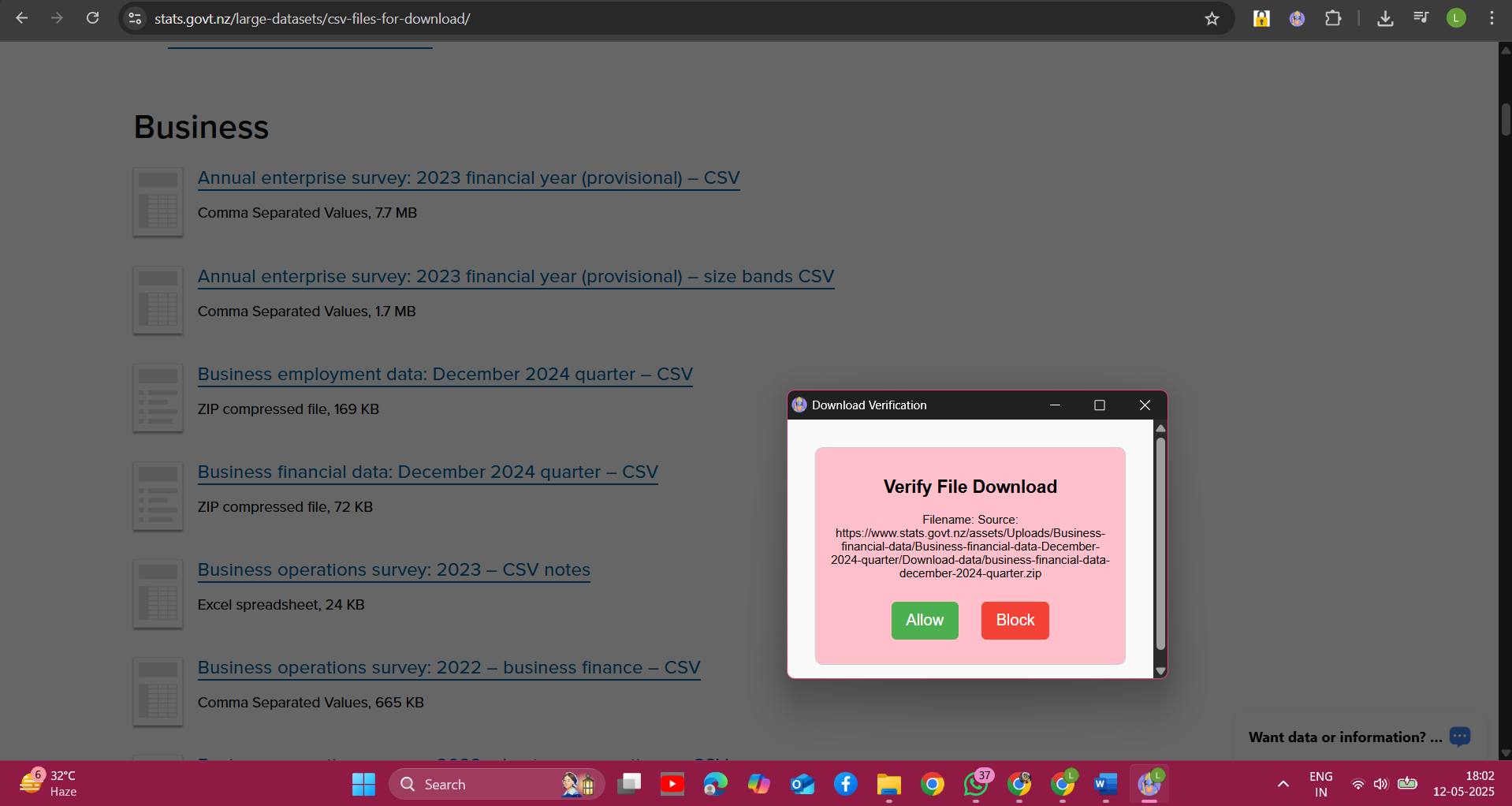
  }

**6.icon.png : Download any image from chrome and save it as ‘icon.png’ in our main folder.**

**As the folder is complete with all files go to chrome -> add extensions -> unpacked folder -> Select the folder and load it. -> solve the errors if any and use the extension.**

****

****

****

**🔐 Security Best Practices**

* **Least Privilege Principle: Request only the necessary permissions in your manifest.json to minimize potential attack vectors.**
* **Content Security Policy (CSP): Define a strict CSP in the manifest to prevent the execution of untrusted scripts.**
* **Secure Communication: Ensure all external communications use HTTPS to protect data integrity and confidentiality.**
* **Regular Updates: Keep the extension updated to patch vulnerabilities and adapt to new security threats**

**🔍 Key Research Papers & Resources**

**1. "A Study on Malicious Browser Extensions in 2025"\***

**This paper examines the evolving threat landscape of malicious browser extensions in 2025, focusing on Mozilla Firefox and Chrome. It highlights the persisting weaknesses in browser security frameworks and the risks associated with browser extensions.&#x20;**

**2. "Effective Detection of Vulnerable and Malicious Browser Extensions"\***

**This study discusses methods for detecting unsafe browser extensions, emphasizing the importance of analyzing download patterns and behaviors to identify potential threats.&#x20;**

**3. "Protecting Browsers from Extension Vulnerabilities"**

**This paper analyzes popular Firefox extensions and finds that a significant percentage require fewer privileges than they request, suggesting a need for more stringent permission controls to prevent exploitation.&#x20;**

**4."A Systematic Review of Security Measures for Web Browser Extension Vulnerabilities"**

**This review presents an overview of research on browser extension vulnerabilities, noting that most studies focus on Firefox and Chrome. It emphasizes the need for tools to evaluate and eliminate vulnerabilities in web browser extensions.&#x20;**

**5. "DOMtegrity: Ensuring Web Page Integrity against Malicious Browser Extensions"**

**This paper addresses the problem of ensuring the integrity of web content in browsers in the presence of malicious extensions. It proposes a cryptographic protocol called DOMtegrity to protect the Document Object Model (DOM) structure of web pages.&#x20;**

**🧠 Additional Resources**

**"Code Verify: An Open Source Browser Extension for Verifying Code Integrity"\***

**This resource discusses the development of a browser extension that verifies the authenticity of web application code, ensuring that users receive unaltered content. ([Engineering at Meta][1])**

**"Introduction to Chrome Browser Extension Security Testing"**

**This blog post provides an overview of Chrome browser extension security, including permissions, testing for vulnerabilities, and mitigation methods. ([Cobalt][2])**

**DownloadSentinel – Smart Download Verification System\***

**This project is a browser-based cybersecurity tool designed to intercept and verify file downloads in real time. When a user initiates a download, the extension halts the process and displays a verification prompt. If the user chooses to proceed, the file's URL is sent to a malware scanning service (like VirusTotal). The system checks for known threats, and only if the file is deemed safe is the download allowed to continue. Malicious files are automatically blocked, preventing potential infections.**

* **Key Features:**

**Real-time download interception**

**User verification step before download**

**Integration with VirusTotal API for threat scanning**

**Blocks known malicious files before reaching the system**

**Easy-to-use browser extension interface**

* **This project aims to reduce the risk of accidental malware infections and empower users with more control over file downloads, offering an added layer of protection beyond traditional antivirus tools.**