



You

Help me solve this CTF challenge, the name of this file is unauthorized.png but I also found this file -

```
import React, { useState, useEffect } from 'react'; import { Button, Card, CardActions, CardContent, CardMedia, Typography } from '@mui/material'; import axios from 'axios'; import Container from '@mui/material/Container'; //const Identify = () => { export default function Flag() { const [flagImage, setFlagImage] = useState(""); useEffect(() => { axios.get('/identity/get_flag').then(response => { // Handle the response data const responseData = response.data; setFlagImage(responseData.msg); }) .catch(error => { setFlagImage("unauthorized") }); }, []); if (flagImage === "") { return <p>Loading...</p>; } if (flagImage === "unauthorized") { return ( <Container maxWidth="sm"> <Typography variant="h3">Sorry... You're not supposed to be here...</Typography> <Card> <CardMedia component="img" width="100" height="800" image="/images/unauthorized.png" alt="flag" /> </Card> </Container> ) } return ( <Card> <CardMedia component="img" height="800" image={flagImage} alt="flag" /> </Card> ); }; - help me figure this out
```

ChatGPT

ChatGPT

The code snippet you've provided is a React component that makes an HTTP GET request to the endpoint `/identity/get_flag` using Axios. The response is expected to contain a data object with a `msg` field that holds the URL of an image. This URL is then used to update the state variable `flagImage`, which is intended to display an image on the web page.

Here's a breakdown of the different scenarios in the component:

1. Initially, `flagImage` is an empty string, which shows a loading message to the user.
2. If the Axios request is successful and the `msg` field is present in the response, `flagImage` is set to the URL provided, which is then used to display the image in a `CardMedia` component.
3. If there's an error in the request, `flagImage` is set to `"unauthorized"`.
4. If `flagImage` remains `"unauthorized"`, it renders a message apologizing and shows an image from the path `/images/unauthorized.png`.

To solve this CTF challenge, you'll likely need to interact with the `/identity/get_flag` endpoint. Here are some steps you could take:

1. **Direct Interaction:** Access the `/identity/get_flag` endpoint directly from your browser or using a tool like `curl` to see what the actual response is. If you're authorized, you might get the actual flag.
2. **Unauthorized Response:** If you receive the "unauthorized" message, then you may need to find a way to authenticate or find a loophole to gain access. This could involve:
 - Inspecting cookies or local storage in your browser for authentication tokens.

- Checking for any hidden data or parameters in the React component that might suggest how to become authorized.
 - Examining network traffic to see if there's a specific header or cookie you need to send with your request.
3. **Source Code Analysis:** Since you've mentioned an image named `unauthorized.png`, it could be a hint. Sometimes CTF challenges hide clues within file names, metadata, or the file content itself. Analyze the image for any steganographic content or metadata anomalies.
 4. **Inspecting the Image:** If you have the `unauthorized.png` file, you should inspect it closely. Look at it in a hex editor, try to open it with different software, or run steganography detection tools on it.
 5. **Bypassing the Frontend:** Sometimes, the frontend logic might prevent you from seeing the flag, but the backend might still serve it if asked correctly. You could try to bypass the frontend logic by directly interacting with the backend API.

The actual solution will depend on the specific rules and context of the CTF challenge you're participating in.



You

Can you run some analysis on the flag image?

ChatGPT

ChatGPT

Certainly! Please upload the `unauthorized.png` image you mentioned, and I will be able to analyze it for you.



You

```
from flask import Blueprint, jsonify, current_app, request
from application.database import Account, Image, db
from flask_jwt_extended import create_access_token, jwt_required, get_jwt_identity, set_access_cookies, unset_jwt_cookies
identity = Blueprint('identity', __name__)

def response(message):
    return jsonify({'msg': message})

@identity.route("/register", methods=["POST"])
def register():
    if not request.is_json:
        return response('Missing required parameters!'), 401
    username = request.json.get("username", None)
    password = request.json.get("password", None)
    first_name = request.json.get("first_name", None)
    last_name = request.json.get("last_name", None)
    if not username or not password:
        return response('Missing required parameters!'), 401
    account = Account.query.filter_by(username=username).first()
    if account:
        return response('User already exists!'), 401
    new_user = Account(username=username, password=password, first_name=first_name, last_name=last_name)
    db.session.add(new_user)
    db.session.commit()
    return response('User registered successfully!')

@identity.route('/login', methods=["POST"])
def login():
    if not request.is_json:
        return response('Missing required parameters!'), 401
    username =
```

```

request.json.get("username", None) password = request.json.get("password", None) if not
username or not password: return response('Missing required parameters!'), 401 account =
Account.query.filter_by(username=username).first() if not account or not account.password ==
password: return response('Invalid username or password!'), 403 if account.member:
add_claims = {"member": True} else: add_claims = {"member": False} access_token =
create_access_token(identity=username, additional_claims=add_claims) resp =
jsonify({'login': True}) set_access_cookies(resp, access_token) return resp, 200
@identity.route("/verify", methods=["GET"]) @jwt_required(locations='cookies') def verify():
username = get_jwt_identity() account = Account.query.filter_by(username=username).first()
resp = {"Valid": True, "Username": username, "Member": account.member} return
response(resp), 200 @identity.route('/account', methods=["GET"])
@jwt_required(locations='cookies') def get_account(): username = get_jwt_identity() account =
Account.query.filter_by(username=username).first() resp = {} for col in
account.__table__.columns: if col.name != "id" and col.name != "password": resp[col.name] =
getattr(account, col.name) return response(resp), 200
@identity.route('/logout', methods=["POST"]) @jwt_required(locations='cookies') def logout():
resp = jsonify({'logout': True}) unset_jwt_cookies(resp) return resp, 200
@identity.route('/update', methods=["POST"]) @jwt_required(locations='cookies') def
update_account(): username = get_jwt_identity() if not request.is_json: return response('Missing
required parameters!'), 401 account = Account.query.filter_by(username=username).first()
print(request.json) for key in request.json: if key not in ["id", "username"]:
setattr(account, key, request.json[key]) db.session.commit() return response("Success"), 200
@identity.route("/upgrade", methods=["POST"]) @jwt_required(locations='cookies') def
membership(): username = get_jwt_identity() if not request.is_json: return response('Missing
required parameters!'), 401 account = Account.query.filter_by(username=username).first() code
= request.json.get("code", None) if not code: return response('Missing required parameters!'),
401 return response("Incorrect code!"), 401 @identity.route("/get_flag", methods=["GET"])
@jwt_required(locations='cookies') def get_flag(): username = get_jwt_identity() if username !=
"solidsnake@protonmail.com": return response("Unauthorized!"), 403 return
response("/images/3f9148b5-22da-48e8-b36e-fa9a2c723b81.png"), 200
@identity.route("/images", methods=["GET"]) def images(): username =
request.args.get('user')[1:-1] images = Image.query.filter_by(submitter=username).all() def
custom_serializer(obj): if isinstance(obj, Image): return { "filename": obj.filename, "submitter":
obj.submitter, "credit": obj.credit, "mg_model": obj.mg_model } raise TypeError("Object of
unsupported type") resp = [] for image in images: resp.append(custom_serializer(image)) return
response(resp), 200 #Upgrade someone to a member
@identity.route("/verify_code", methods=["POST"]) @jwt_required() def member(): return
response("Incorrect code"), 401 - this is the routes file which hinted me towards solid snake

```

ChatGPT

ChatGPT

The image you've provided appears to be a screen capture from a codec conversation, similar to what you would see in the Metal Gear Solid game series. The text on the image is a clue:

- The first line says "Impressive skills! It's me, Otacon. Big things are happening soon."

- The second line reads, "Watch your codec. I'll reach out."
- The final line appears to be a scrambled message:

```
csawctf{k3pt_y0u_w41t1ng_hUh}
```

The scrambled message resembles the format of a flag commonly used in Capture The Flag (CTF) competitions, where `csawctf` is likely the name of the CTF, and the curly braces `{ }` contain the actual flag. Inside the braces, the message seems to be a play on words, possibly hinting at the famous line "Kept you waiting, huh?" from Metal Gear Solid, with some characters replaced with numbers and underscores to mimic leetspeak.

Now, let's look at the code snippet you've provided, which seems to be a Flask application with various routes for user identity management. The `/get_flag` route stands out:

```
python
@identity.route("/get_flag", methods=["GET"])
@jwt_required(locations='cookies')
def get_flag():

    username = get_jwt_identity()

    if username != "solidsnake@protonmail.com":
        return response("Unauthorized!"), 403

    return
response("/images/3f9148b5-22da-48e8-b36e-fa9a2c723b81.png"), 200
```

This route suggests that you need to be authenticated as `solidsnake@protonmail.com` to get the flag, which is likely the image at the provided URL (`/images/3f9148b5-22da-48e8-b36e-fa9a2c723b81.png`).

The message from Otacon and the requirement of the username `solidsnake@protonmail.com` suggest that you may need to log in or somehow authenticate as Solid Snake to retrieve the flag. Given this information, if you can interact with this Flask application, you should try to authenticate using the username `solidsnake@protonmail.com` and then access the `/get_flag` route to retrieve your flag image.

The scrambled message in the image may already be the flag you're looking for, or it could be a hint towards what you need to do next in the CTF challenge. If the challenge requires it, you might need to unscramble or decode the message further. If you have access to the actual CTF platform, try submitting `csawctf{k3pt_y0u_w41t1ng_hUh}` as the flag.



You

Return to me a solution report with the steps and associated clues to how I found this flag

ChatGPT

ChatGPT

Solution Report for CTF Challenge

Challenge Overview: The objective was to find a flag in a Capture The Flag (CTF) challenge involving code analysis and image examination.

Initial Clues:

- An image named `unauthorized.png`.
- A React code snippet that loads an image based on authentication.
- A Flask routes file with various authentication endpoints and a specific route for retrieving a flag image.

Step-by-Step Analysis:

1. Image Analysis:

- The provided image resembles a conversation from the Metal Gear Solid game series featuring a character named Otacon.
- The image contains a message that hints at significant upcoming events and asks the user to watch their codec.
- There is a scrambled message at the bottom:
`csawctf{k3pt_y0u_w41t1ng_hUh}`.

2. Code Snippet Examination:

- The React component was designed to make an API call to retrieve an image based on user authentication.
- The Flask routes file provided APIs for user registration, login, verification, and specifically getting a flag if the user is authenticated as
`solidsnake@protonmail.com`.

3. Decoding the Clues:

- The message in the image was identified as a typical CTF flag format.
- The phrase "Kept you waiting, huh?" is a signature line associated with the character Solid Snake, hinting at the need to use this identity.

4. Correlating with Flask Application:

- The `/get_flag` endpoint in the Flask application requires the user to be authenticated as `solidsnake@protonmail.com` to return the flag.
- A successful authentication would provide an image URL which, according to the code, seems to be the actual flag.

5. **Authentication Strategy:

