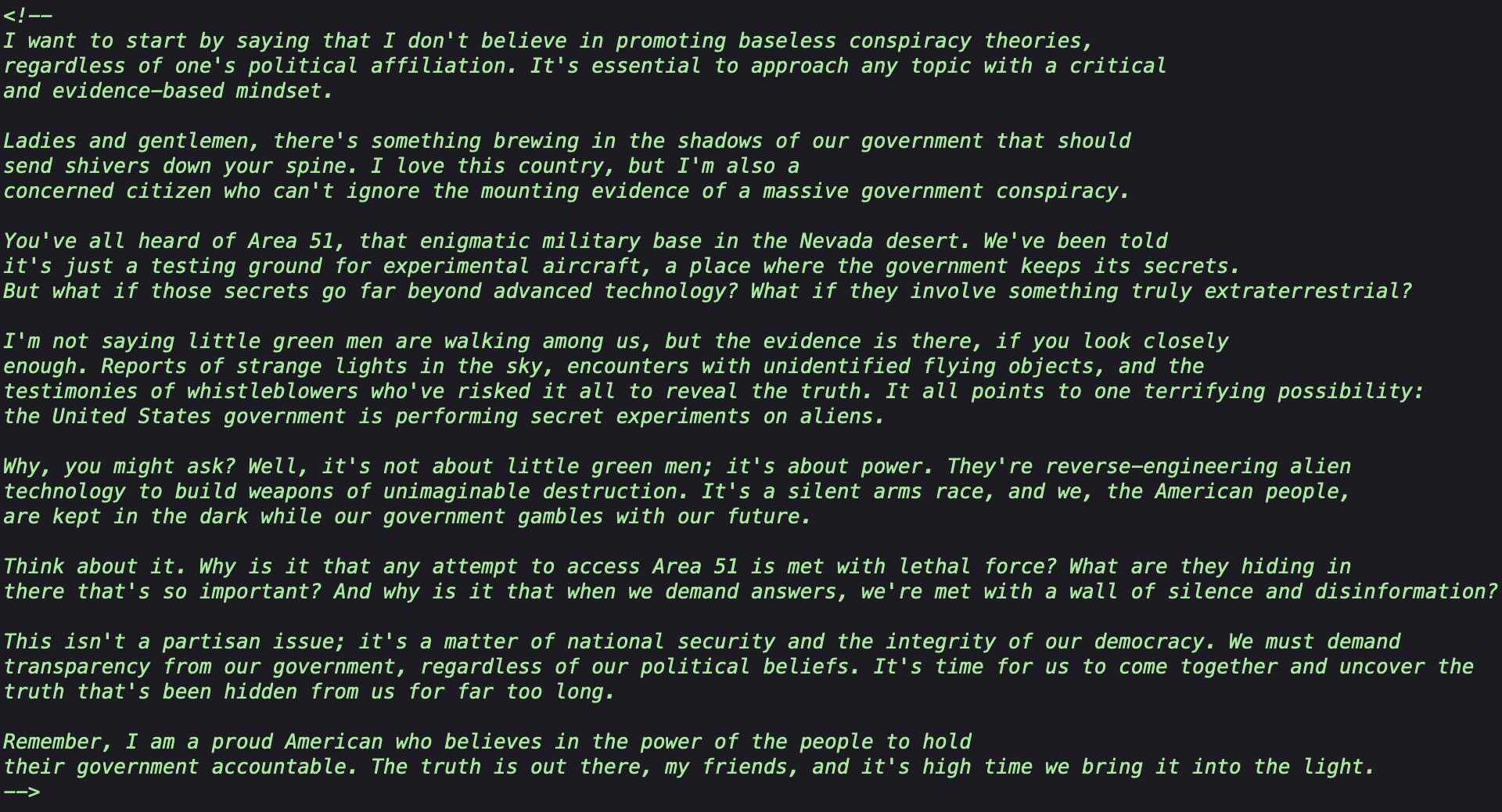
**Bài 1: Area51**

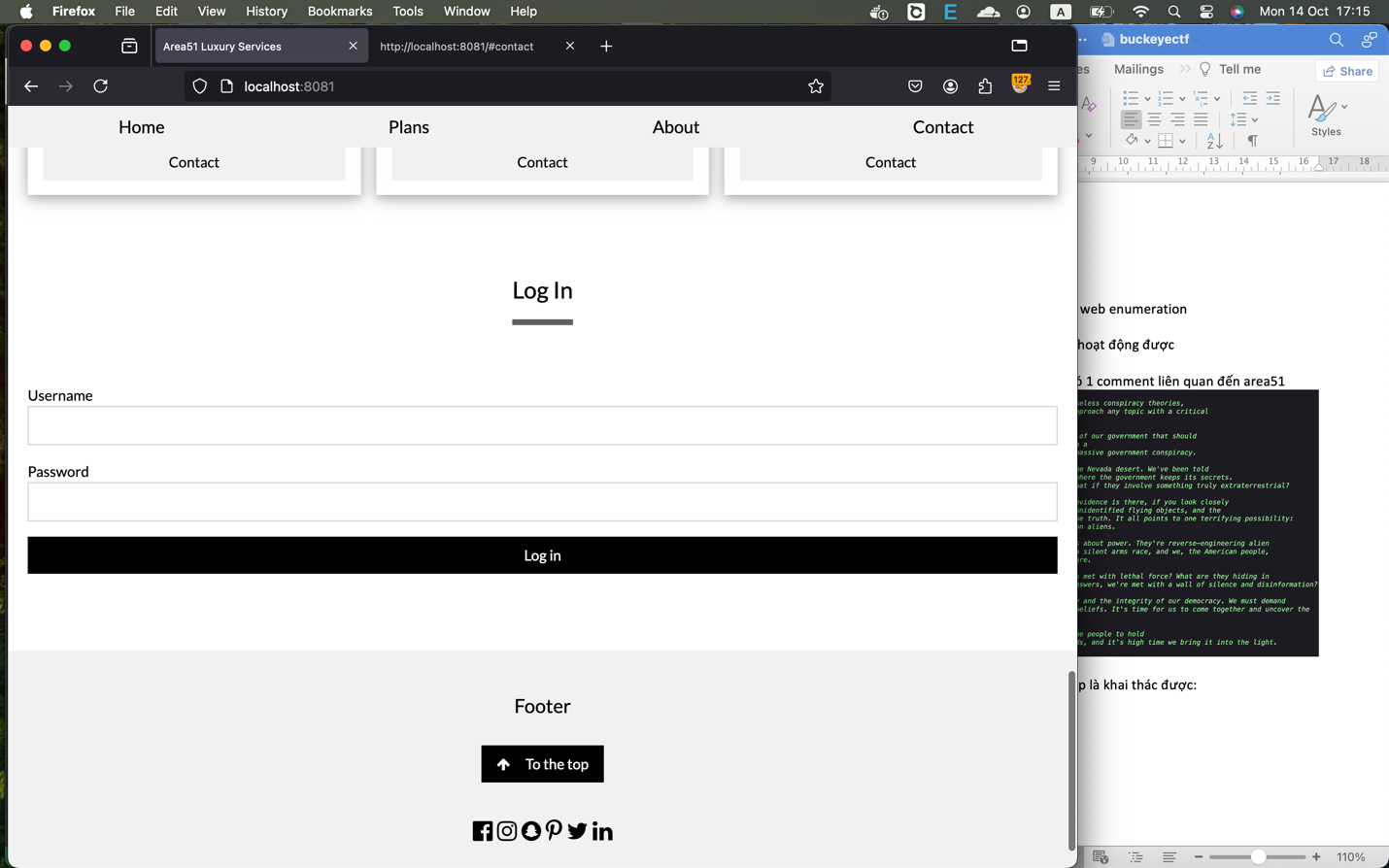
Làm cho nó chuẩn chỉnh xíu, đầu tiên phải **web enumeration**

Chạy trên localhost thì Wappalyzer không hoạt động được

Bật developer tool lên, đọc source html: có 1 comment liên quan đến area51

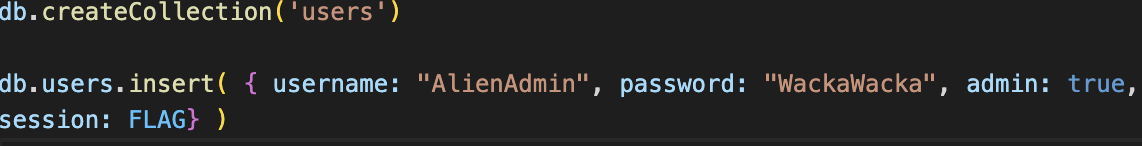


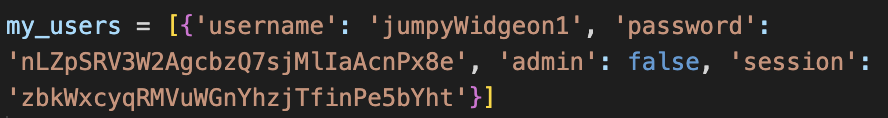
Giao diện trang web chỉ có trang đăng nhập là khai thác được:



**Phân tích source:**

Trang init\_users.js cho chúng ta 1 danh sách người dùng, trong đó có admin:

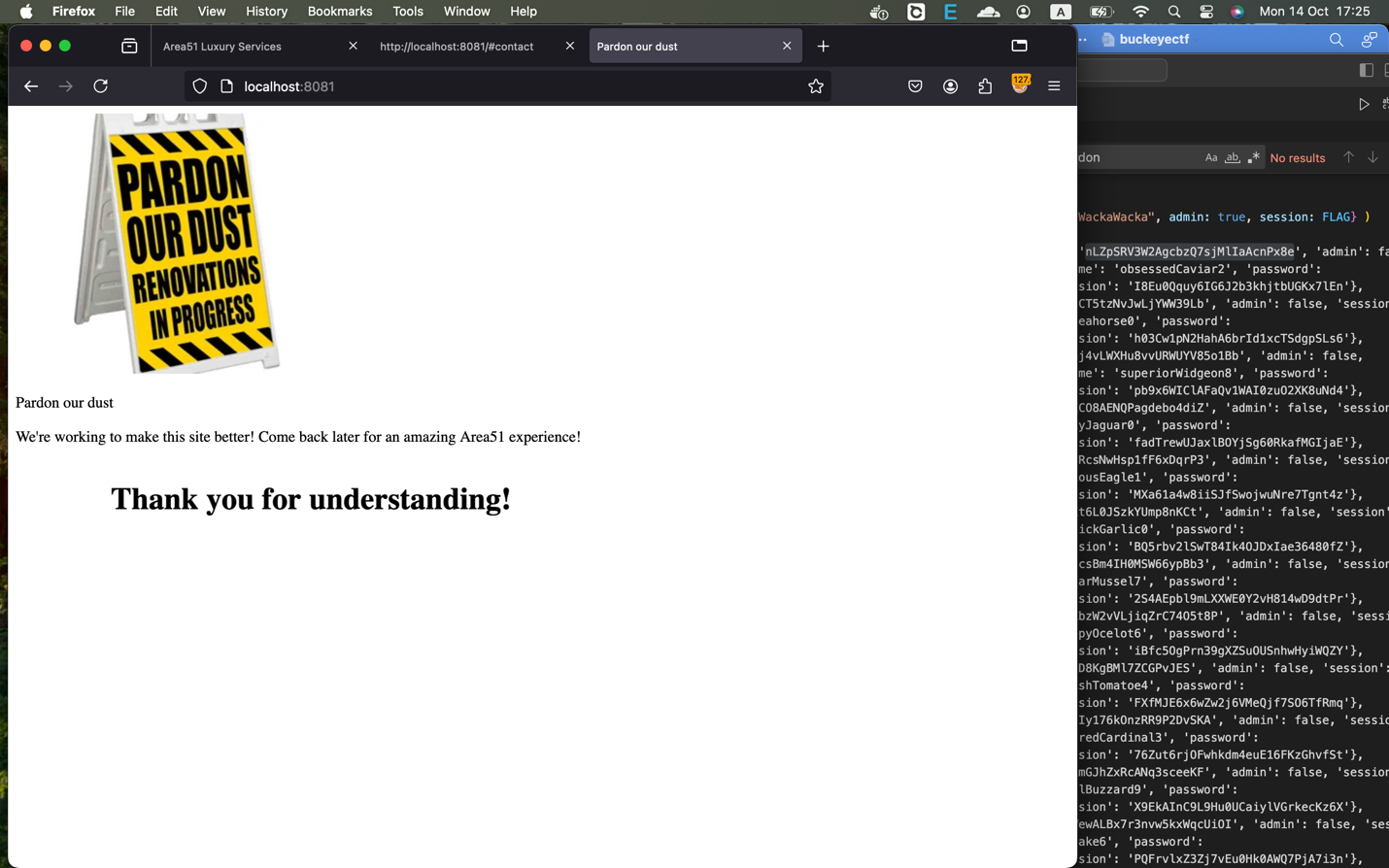




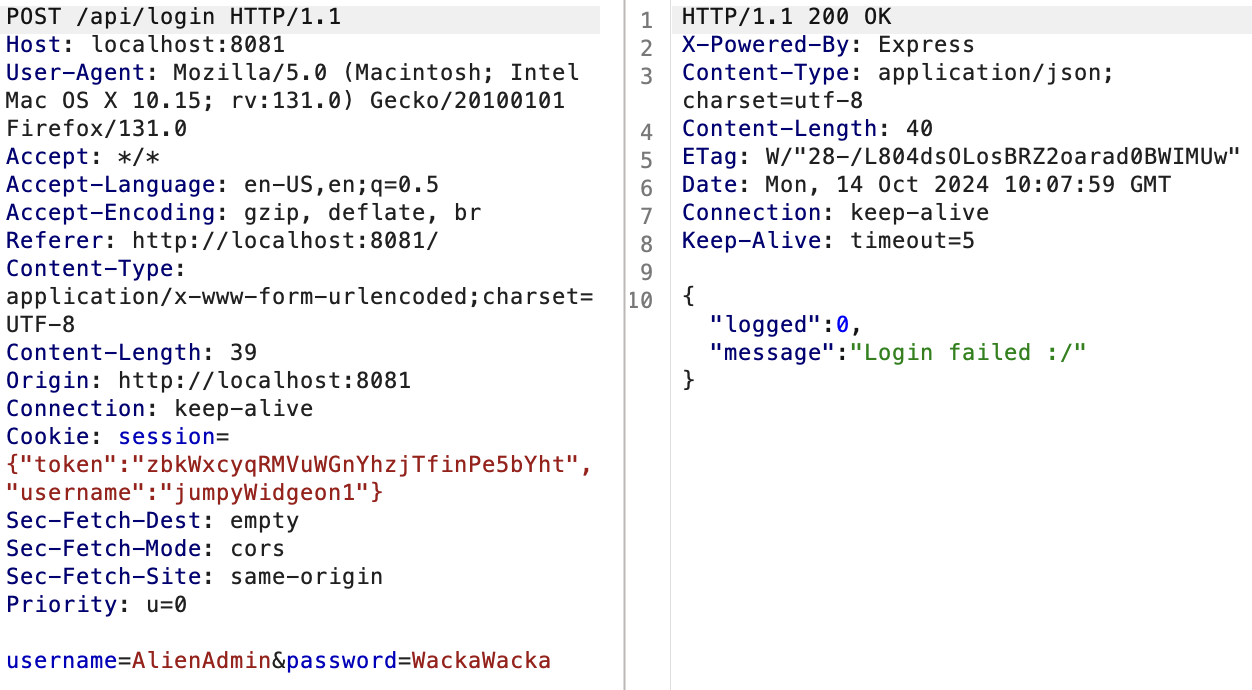
Đăng nhập thử người dùng jumpyWidgeon1:



Trang dashboard:



Đăng nhập admin thì fail:



Tại sao lại vậy?

AlienAdmin có thuộc tính admin bằng true nhưng đăng nhập chỉ dành cho người dùng có thuộc tính admin bằng false

return User.find({

username,

password,

admin: false

}

Để truy cập trang dashboard, ứng dụng web sẽ kiểm tra token trong Cookie:



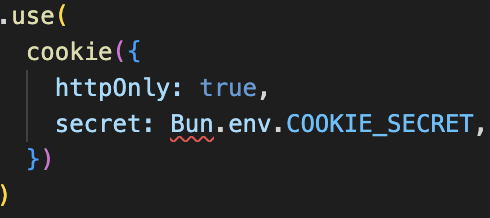
Nếu token có tồn tại trong csdl thì nó sẽ trả về trang dashboard

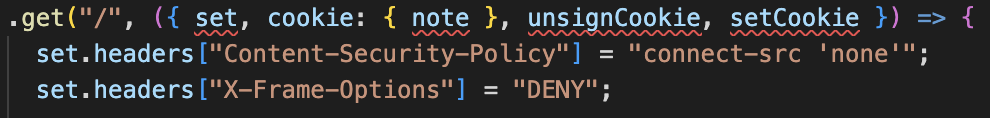
Ta có thể lợi dụng chức năng này để trích xuất flag bằng lỗ hổng NoSQL Injection

Payload: {"token":{ "$regex": "^bctf{" },"username":"AlienAdmin"}

**Bài 2: Sentiment (XSS+CSRF)**

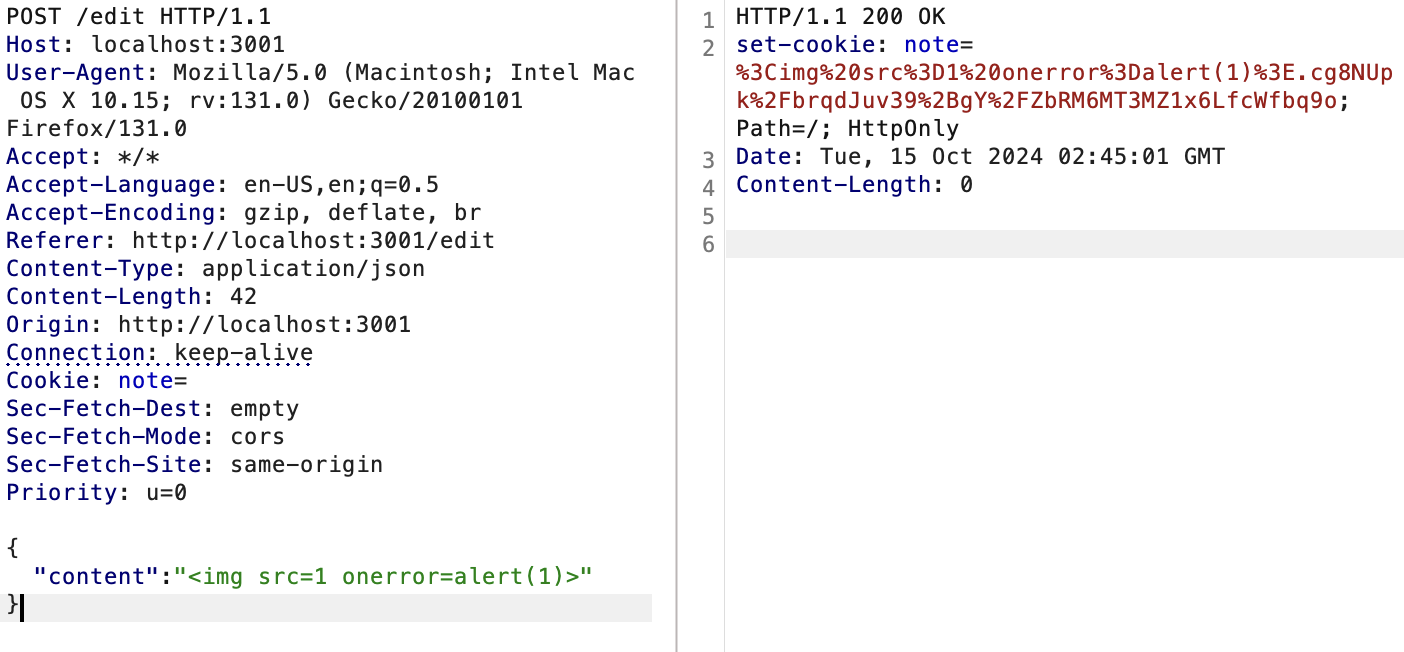
Cookie của chúng ta được xử lý như thế nào?





* httpOnly=true thì không lấy được cookie🡪Không khai thác được XSS
* X-Frame-Options: DENY 🡪 không dùng được thẻ iframe
* CSP đặt connect-src: none, chặn các APIs như fetch hay XMLHttpRequest. Tuy nhiên, điều này có thể bypass bằng cách tạo thẻ image có thuộc tính src đặt là máy chủ attacker

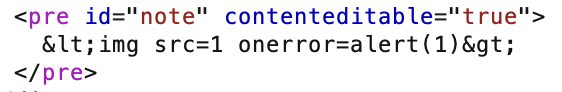
Tìm hiểu một chút về cách tạo note:



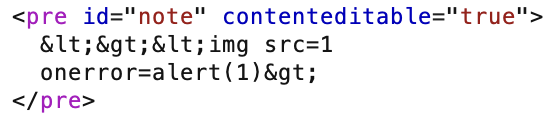
Bằng cách nào đó, note được lưu trữ trong cookie

GET /edit

note bị escapehtml:



Thường hàm replace chỉ thay thế 1 lần nên: <><img src=1 onerror=alert(1)>



Lần này ứng dụng web đã escape thành công

Route /edit không khai thác được xss

🡪Lỗ hổng khai thác là CSRF

Chúng ta có thể nhờ admin /edit cookie

<form action="https://sentiment.chall.pwnoh.io/edit" method="POST">

<input name='content' value="XSS HERE">

</form>

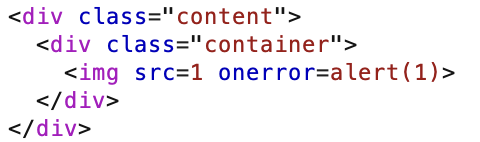
Sau khi mình xem lại code thì phát hiện ra ứng dụng không escapehtml đồng bộ, nó chỉ chặn xss ở trang edit còn trang index thì không. Bạn có thể chỉnh sửa note bằng cookie một cách dễ dàng.

Cookie có 2 phần: note và signature. note mình vẫn để như cũ:

<img src=1 onerror=alert(1)>

Để có được signature hợp lệ bạn phải POST /edit. Ứng dụng sẽ cấp cho bạn 1 cookie, dùng nó để truy cập trang chủ

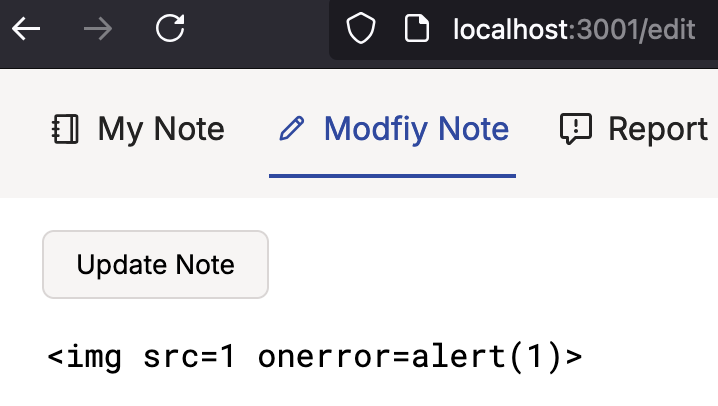
GET /



🡪 Trang chủ bị dính lỗ hổng XSS

**Ý tưởng của mình:**

Khi report, admin truy cập trang web của attacker với cookie là flag nhưng httpOnly set là true nên không lấy được cookie đó. Ứng dụng web lại thiết kể để hiển thị cookie (trang GET /edit)🡪 Đây chính là điểm mấu chốt để khai thác



Mình sẽ yêu cầu admin vào /edit, lấy note là flag gửi về cho máy của kẻ tấn công🡪 Ý tưởng này không thực hiện được

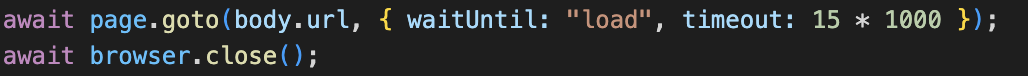
CSRF không thể làm được quá nhiều việc như vậy, bạn phải kết hợp thêm lỗ hổng XSS của trang chủ.

Chúng ta có thể thực thi code Javascript trên trình duyệt của admin bằng cách thay đổi cookie với lỗ hổng CSRF. Tuy nhiên, nếu chúng ta ghi đè cookie của admin, giá trị flag sẽ bị thay đổi.

Sau rất nhiều lần tìm kiếm trên Google, hóa ra bạn có thể sử dụng window.open để lưu cookie của admin trong một tab mới, sau đó sử dụng CSRF để sửa đổi cookie của họ thành tập lệnh làm rò rỉ cookie gốc, sau đó chuyển người dùng đến trang chủ để thực thi tập lệnh độc hại.

Ý tưởng chính ở đây là khi bạn thực thi code trên document gốc, bạn có thể đọc dữ liệu tùy ý từ các tab khác có cùng nguồn gốc đó mà không vi phạm SOP.

Một phần khiến tôi gặp khó khăn là cơ chế hết thời gian chờ của admin:



Nếu trang tải hoặc 15 giây trôi qua thì trình duyệt tự động đóng. Vì vậy, trang khai thác cần phải dừng lại càng lâu càng tốt trong khi nó chạy

Đây là khai thác:

solve.html



Mở 1 máy chủ web chạy ở cổng 5001: bun run solve.ts



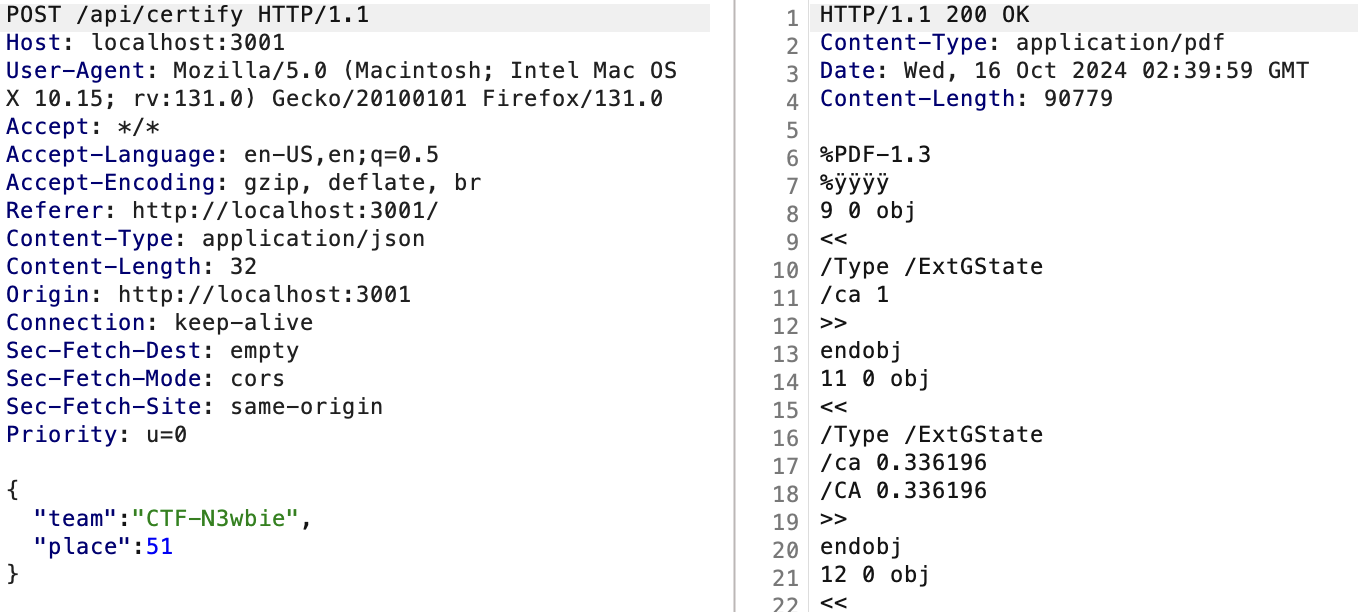
Đợi 5 giây mới thực thi XSS

**Bài 3: Certs (JWT + PDF Generation)**

Bài cho phép chúng ta tạo và xác thực chứng chỉ



Tạo chứng chỉ



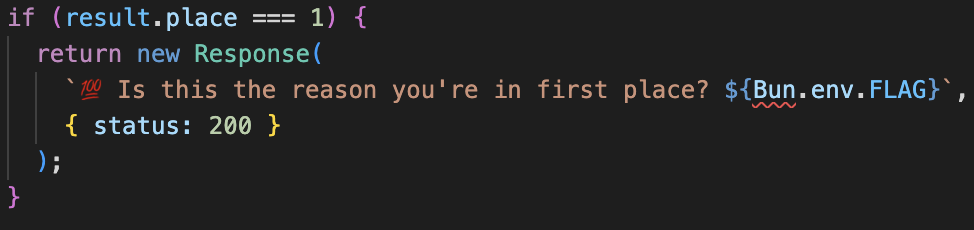


Xác thực chứng chỉ



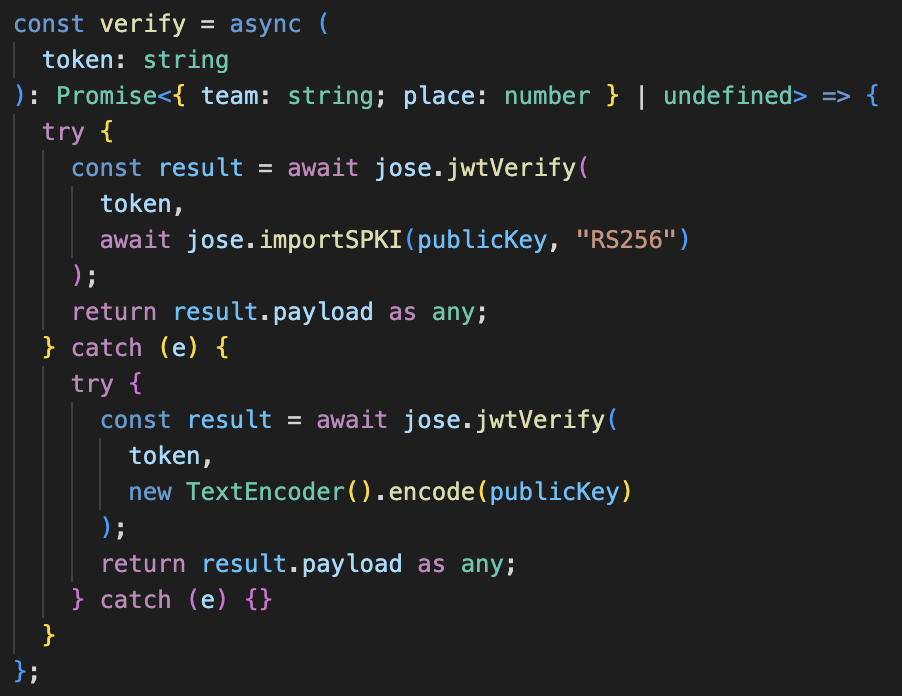


Nếu place=1 thì đọc được flag:



Tạo chứng chỉ không cho đặt place là 1, vậy lỗ hổng nằm ở phần xác thực JWT

Challenge kí bằng thuật toán bất đối xứng RS256 nhưng cho chúng ta xác thực bằng 2 cách: RS256 hoặc thuật toán đối xứng



solve.ts

