**Web Enumeration**

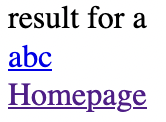
**Giao diện trang web đơn giản:**



**Route /add**



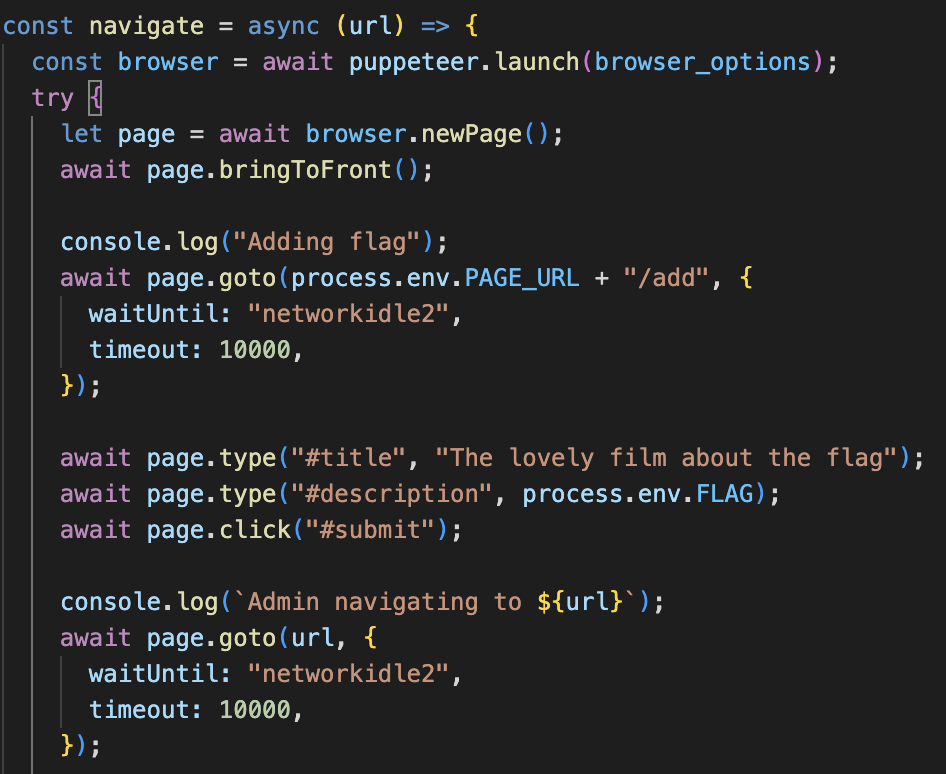
**Route /search?filter=a**



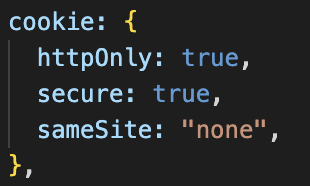
**Route /film?id=0**



Thử thách này có 1 con app và 1 con bot. Mục đích của ứng dụng là lưu trữ dữ liệu film trong đó có tittle và description trong session cookie và cho phép người dùng tìm kiếm nó. Con bot là 1 người dùng khác có thể tương tác với ứng dụng và flag nằm trong description của 1 trong những bộ phim của anh ấy:



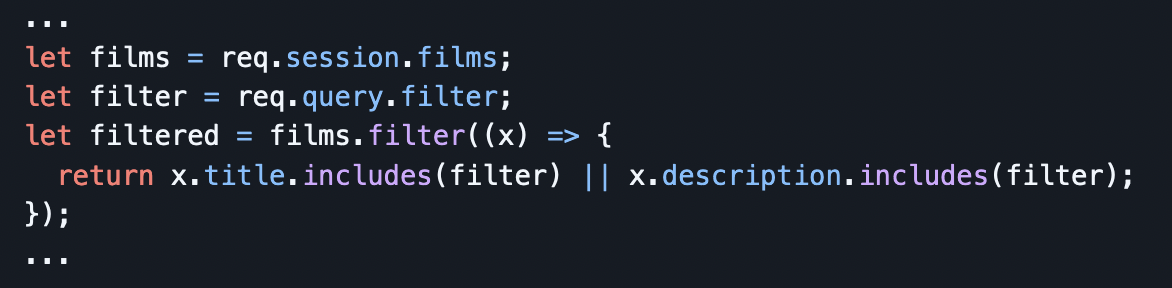
Trang web set cookie như sau:



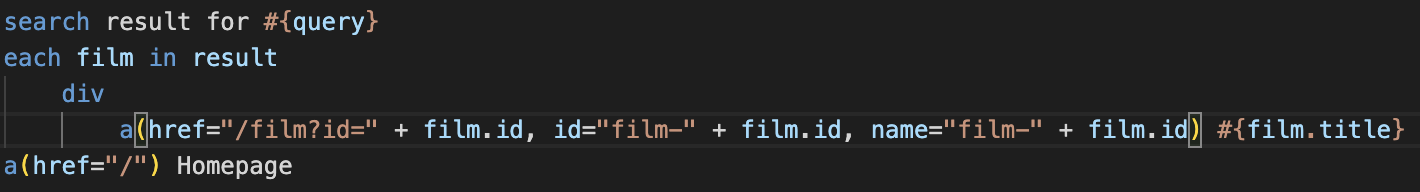
Thuộc tính SameSite được đặt là None 🡪 Cookie không thể truy cập được trực tiếp từ phản hồi của 1 yêu cầu ví dụ như fetch, vì Same Origin Policy được áp dụng vì không có headers CORS nào được đặt ở đây. Thậm chí nếu chúng ta tìm được lỗ hổng XSS thì nó cũng không dễ khai thác vì cookie được đặt httpOnly.

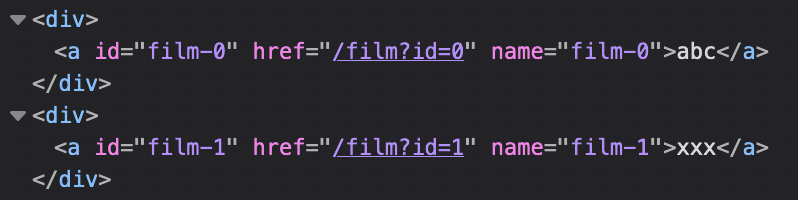
Ứng dụng dùng pugjs để kết xuất các trang nhưng không có lỗ hổng tiêm template nào ở đây vì các templates luôn được biên dịch từ các tập tin.

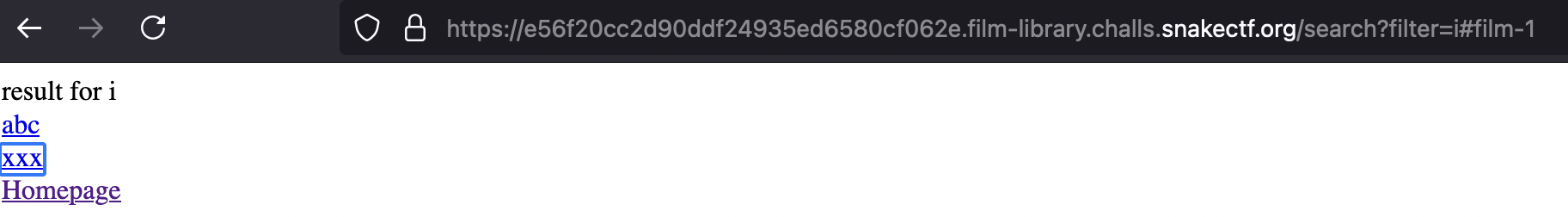
Đầu tiên, bạn để ý thấy điểm vào search cho phép tìm kiếm cả title và description của films:



Vì con bot có flag trong description của 1 film, mình sẽ tấn công dồn dập kết quả của truy vấn để phát hiện sự có mặt của flag. Thuộc tính id có thể được dùng vì nó xuất hiện trong trang kết quả:



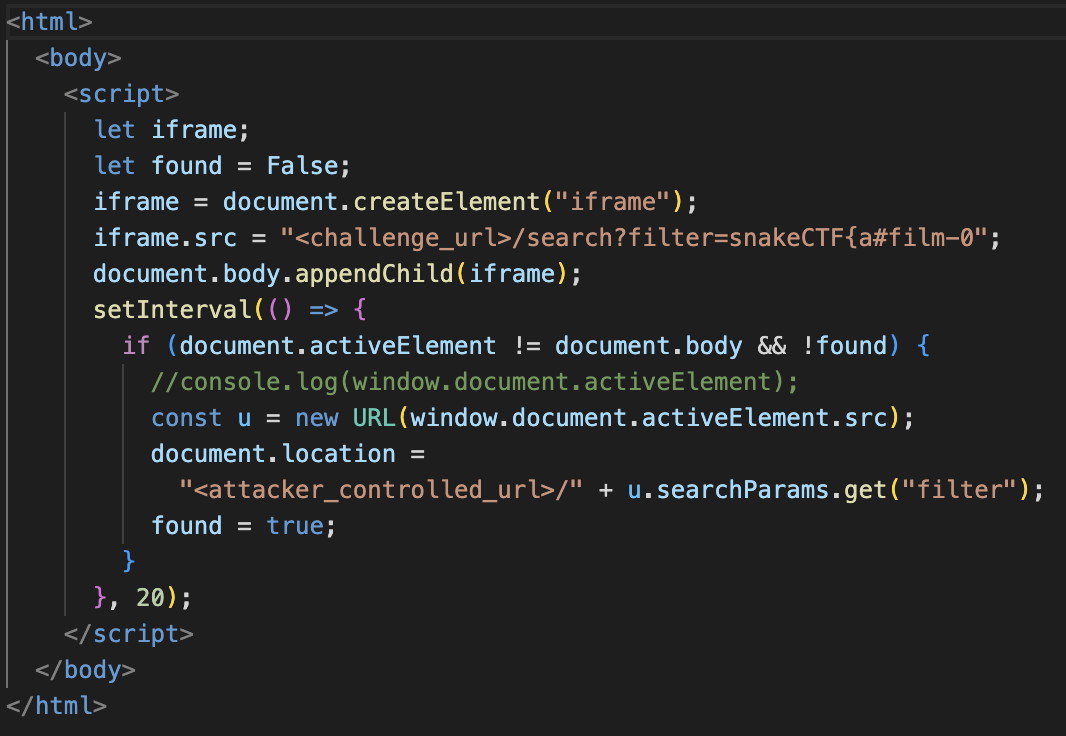




Loại tấn công này gọi là XS-Leak, cụ thể hơn là nhắm đến thuộc tính id và có thể thực hiện bằng thẻ iframes:

* Thẻ iframe sẽ tải trang search với filter là snakectf{a
* Url sẽ có phân đoạn là thuộc tính id ví dụ như #film-0
* Nếu xuất hiện film có id trên, thẻ iframe sẽ focus
* Javascript có thể phát hiện khi nào thẻ iframe được focus bằng cách dùng window.document.activeElement

Payload mẫu:



Mỗi 20ms Javascript kiểm tra xem thẻ iframe có được focus không, chuyển hướng đến trang của attacker