**BÁO CÁO BÀI THỰC HÀNH SỐ 9**

**KIỂM THỬ LỖ HỔNG XSS và CSRF**

***Họ và tên: Lê Ngọc Lộc***

**Phần 1: Kiểm thử XSS**

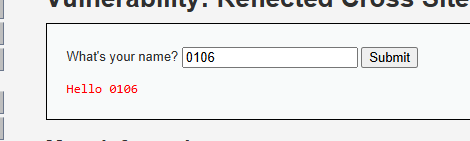
Thực hiện kiểm thử lỗ hổng XSS trên website: chọn 1 trong các trang sau để thử nghiệm

1. [http://demo.testfire.net](http://demo.testfire.net/)`
2. [http://php.testsparker.com](http://php.testsparker.com/)
3. [http://testasp.vulnweb.com](http://testasp.vulnweb.com/)
4. [http://testasp.vulnweb.com](http://testasp.vulnweb.com/)
5. [http://testphp.vulnweb.com](http://testphp.vulnweb.com/)
6. [http://www.webscantest.com](http://www.webscantest.com/)
7. [http://testhtml5.vulnweb.com](http://testhtml5.vulnweb.com/)
8. [http://aspnet.testsparker.com](http://aspnet.testsparker.com/)
9. [http://zero.webappsecurity.com](http://zero.webappsecurity.com/)

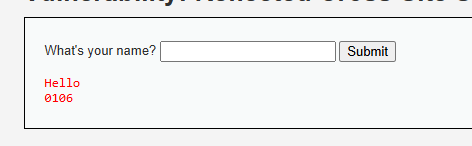
**Câu 1**

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

**- Bước 1: Nhập vào ô nhập liệu giá trị An chúng ta được kết quả như sau:**

****

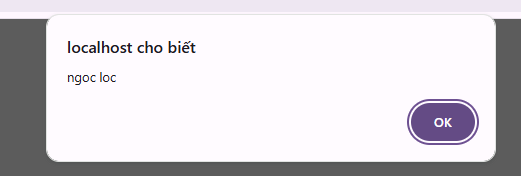
**- Bước 2: Nhập vào ô nhập liệu giá trị <br>0106</br> chúng ta được kết quả:**

****

Tên người đã được in đậm cho thấy thẻ &lt;b&gt; đã được xử lý để hiện thị. Đây là cơ sở để nghi ngờ chức năng này có lỗ hổng XSS để khai thác.

**Bước 3: Nhập chuỗi <script>alert(“0106”)</script> chúng ta sẽ thấy một hộp thoại cảnh báo được bật**

**lên**

****

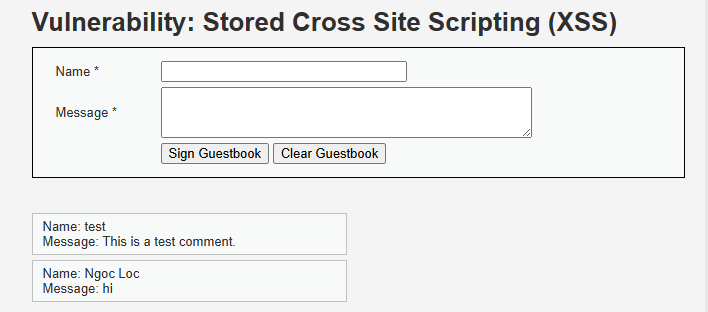
**Từ ví dụ này có thể kết luận 2 điều.**

* Thứ nhất biến name có thể nhận giá trị đầu vào bất kỳ và truyền lên server xử lý.
* Thứ 2, server đã không kiểm soát giá trị đầu vào này trước khi trả về cho trình duyệt. Dẫn đến việc đoạn mã javascript đã bị chèn vào trong source code.

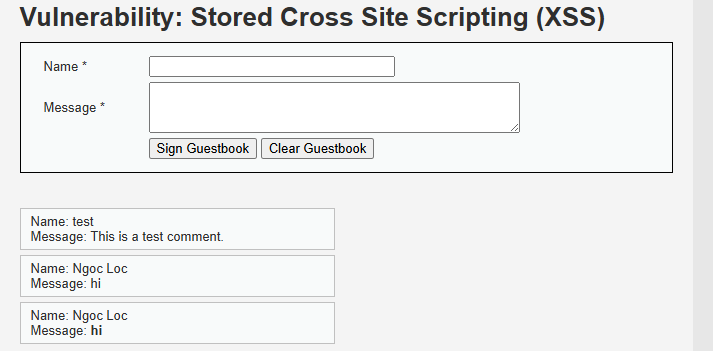
**Câu 2**

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:

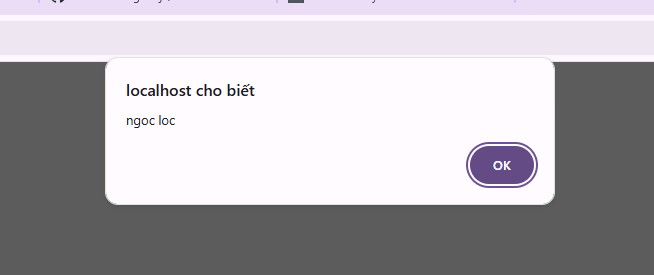
**- B1: Nhập vào ô nhập liệu tên người là An và thông điệp Hi. Chúng ta được kết quả như sau:**

****

**- B2: Nhập vào ô nhập liệu tên người là An và thông điệp &lt;b&gt;Hi&lt;/b&gt;. Chúng ta được kết quả như sau:**

****

- B3: Nhập vào ô nhập liệu tên người là An và thông điệp <script>alert(“Ngoc Loc”)</script>. Chúng ta sẽ thấy hộp thoại cảnh báo được bật lên

****

Suy luận tương tự như ví dụ về Reflected XSS, chúng ta thấy chức năng này có thể có lỗ hổng Stored XSS

Hơn nữa, khi truy cập lại vào địa chỉ http://localhost/dvwa/vulnerabilities/xss\_s/ ở thời điểm bất kỳ, ta đều nhận được hộp thoại này. Như vậy có thể thấy, mã Javascripts đã được đưa vào CSDL của website thành công.

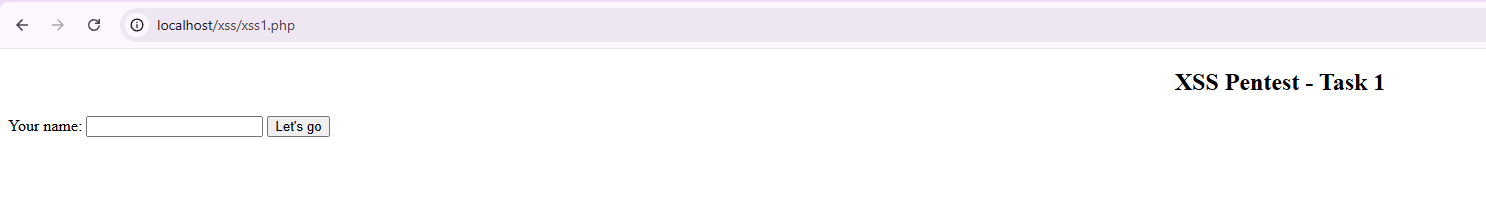
Đến đây có thể kết luận trang web đã có lỗ hổng Stored XSS.

**-** Các bước kiểm thử và phán đoán: **0-1 điểm** theo các bước kiểm thử và logic trong phán đoán

- Giá trị kiểm thử cho thấy lỗ hổng và giải thích: **0.5 điểm**

**Câu 3**

**- Bước 1: Truy cập vào địa chỉ** [**http://localhost/xss/xss1.php**](http://localhost/xss/xss1.php)

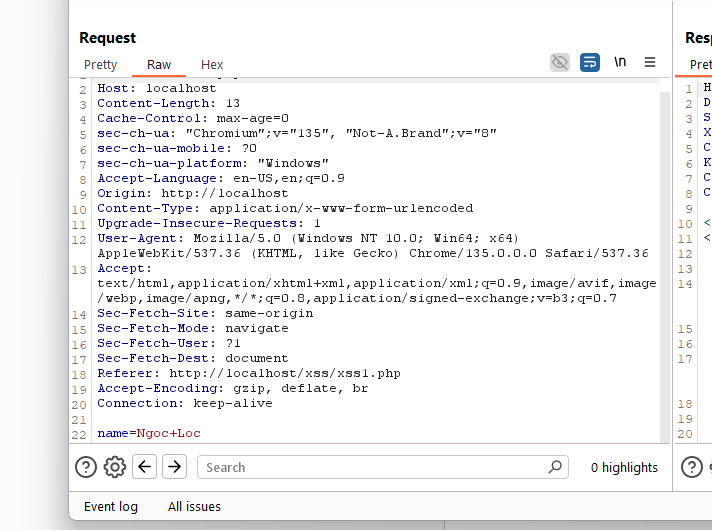
****

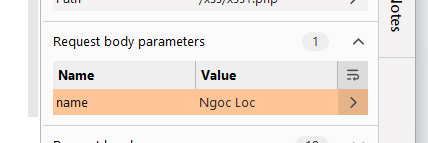
**- Bước 2: Nhấn tổ hợp phím Ctrl+U để xem mã nguồn HTML của trang kết quả. Có thể thấy giá trị đầu vào của người dùng nằm trong mã nguồn HTML. Bên cạnh đó, chúng ta có thể thấy một form HTML sử dụng phương thức POST để gửi tham số có tên là name tới website. Trên mã nguồn, ta không phát hiện ra trường ẩn trong form.**

****

**- Bước 3: Để xác định hết các tham số mà có thể không quan sát được từ mã nguồn HTML, sử dụng phần mềm Burp Suite như hướng dẫn bài thực hành số 5 để quan sát các thông điệp HTTP trao đổi giữa trình duyệt và máy chủ.**

**- Bước 4: Trên Burp Suite, chọn thẻ Proxy HTTP Proxy. Chọn thông điệp HTTP Request truy cập tới địa chỉ ở trên. Ở thẻ Raw chúng ta có thể thấy đây là thông điệp sử dụng phương thức POST để truy cập. Ở phần thân của thông điệp này có tham số và giá trị mà ta đã truyền.**

****

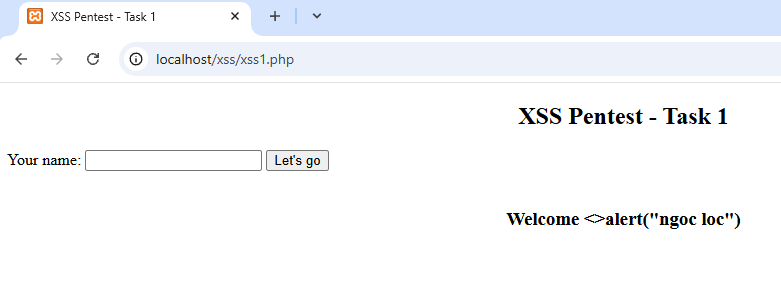
Trên thẻ Params chúng ta có thể khẳng định lại có tham số name được gửi bằng phương thức POST.

**Phân tích tham số đầu vào và xác định sự ảnh hưởng**

Ở phần trên, ta đã phân tích để thấy rằng giá trị đầu vào được ghép thành chuỗi “Wellcome Giá trị đầu

vào”. Ta sẽ thử với một số giá trị đầu vào điển hình để kiểm thử lỗ hổng XSS.

- Bước 1: Thử giá trị <script>alert(“Ngoc Loc”)</script>. Kết quả cho phép phán đoán từ script đã bị lọc.

****

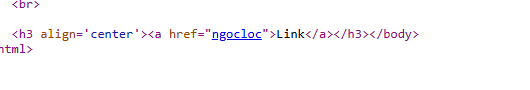
Bước 2: Để kiểm tra xem bộ lọc có thực hiện đệ quy không, chúng ta sử dụng giá trị <sscriptscript>alert(1)</sscriptscript>. Kết quả không thay đổi với bước trên. Như vậy, có thể thấy rằng mọi từ script sẽ bị lọc khỏi giá trị đầu vào cho tới khi hết.

- Bước 3: Để kiểm tra bộ lọc có kiểm tra dạng in hoa hay không, chúng ta sử dụng giá trị <sCript>aler(1)</sCript>;. Kết quả cho thấy bộ lọc không thực hiện kiểm tra dạng in hoa nên đoạn mã đã được thực thi.

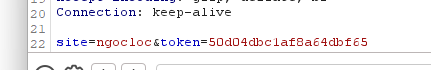
**Ví dụ 2**

- Bước 1: Truy cập vào địa chỉ http://localhost/xss/xss2.php

- Bước 2: Điền địa chỉ một website nào đó và nhấn nút Create link. Kết quả cho thấy một đường link được tạo ra. Xem mã nguồn của trang trả về cho thấy địa chỉ trên được gán vào thuộc tính href của thẻ a;



- Thực hiện phân tích tương tự trên ta thấy có hai tham số được truyền bằng phương thức POST là site và token. Trong đó token là một tham số ẩn được nhúng vào form của trang Web.

****

**Phân tích tham số đầu vào và xác định sự ảnh hưởng**

Ở trên, ta đã thấy có tham số ẩn là token nhưng không thấy sự hiện diện của nó trong kết quả trả về là đường dẫn được tạo ra. Do không đủ cơ sở đến phán đoán nên ta không xét đến tham số này trong ví dụ.

Tiếp theo, ta thấy rằng giá trị tham số được gán cho thuộc tính href của thẻ a như sau:

<a href=”gia tri dau vao”></a>

**Cách thức kiểm thử 1**

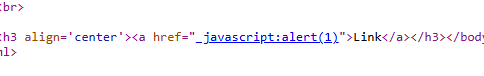
Ta thử truyền trực tiếp mã thực thi dưới dạng javascript:Mã thực thi

- Bước 1: Kiểm thử với giá trị đầu vào javascript:alert(1). Kết quả cho thấy từ script đã bị lọc.

- Bước 2: Thử nghiệm tương tự với các cách thức biến đổi như ở ví dụ trước ta thấy từ script luôn bị

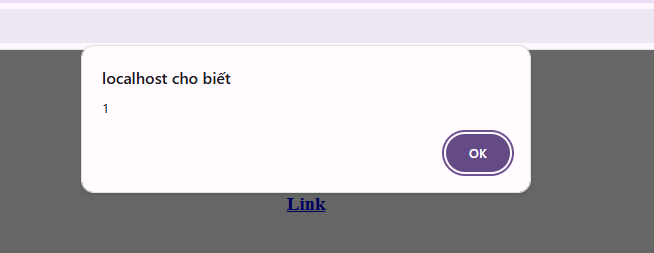
lọc. Do đó, ta thử các dạng biểu diễn bằng mã Unicode cho từ khóa javascript là &amp;#x6A&amp;#x61&amp;#x76&amp;#x61&amp;#x73&amp;#x63&amp;#x72&amp;#x69&amp;#x70&amp;#x74:alert(1). Kết quả cho ta thấy giá trị

đầu vào này vẫn giữ nguyên mà không bị lọc.



- Bước 3: Kiểm tra lại kết quả bằng các nhấn vào đường dẫn đã tạo ra trên trang kết quả, ta thấy đoạn

mã đã được thực thi.



Cách thức kiểm thử 2

Ở cách thức trên, ta thấy chuỗi script bị lọc. Do đó ta có thể tìm cách thực thi mã nguồn thông qua các

thuộc tính sự kiện. Ở dạng này thì mã thực thi không cần đặt trong thẻ <script> và cũng không cần đến

tiếp đầu ngữ javascript. Ta tìm cách đưa giá trị đầu vào để đường dẫn tạo ra có dạng như sau:

<a href="Địa chỉ nào đó" onclick="alert(1)">Link</a>

Lưu ý rằng, giá trị tham số được truyền cho thuộc tính href như sau:

<a href="Giá trị đầu vào">Link</a>

Vì vậy có thể sử dụng giá trị đầu vào là:

anything" onclick="alert(1)

**2.5. Sử dụng công cụ hỗ trợ kiểm thử Burp Suite**

Trong phần này, chúng ta sử dụng công cụ Burp Suite để truyền giá trị cho các tham số khi thực hiện

kiểm thử. Kỹ năng này hữu dụng khi chúng ta cần truyền giá trị cho các trường ẩn, thay đổi các giá trị

mặc định trong Web form hay thậm chí thay đổi cả giá trị của các tiêu đề HTTP.

- Bước 1: Cấu hình và khởi động Burpsuite như hướng dẫn ở phần trước

- Bước 2: Trên cửa sổ công cụ Burpsuite, mở thẻ Proxy Intercept và chắc chắn tính năng Intercept is

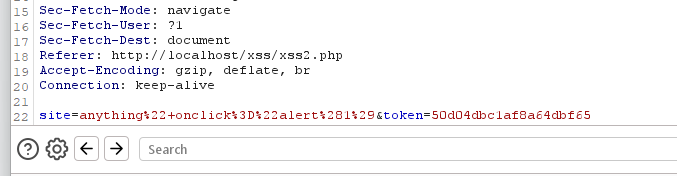
on đã được bật

- Bước 3: Mở địa chỉ trang Web cần kiểm thử. Ví dụ

http://localhost/xss/xss2.php

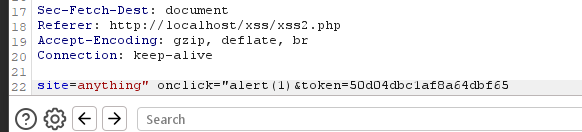
- Bước 4: Điền các giá trị đầu vào bất kỳ nào đó và gửi yêu cầu từ trình duyệt

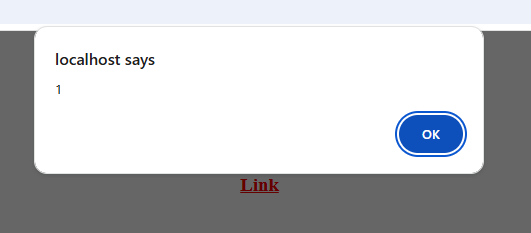
- Bước 5: Trên thẻ Intercept chúng ta có thể thấy thông điệp yêu cầu bị chặn giữ



- Bước 7: Sửa trực tiếp các giá trị tham số cần thiết trên thẻ Raw hoặc trong thẻ Params. Sau đó nhấn

Forward.

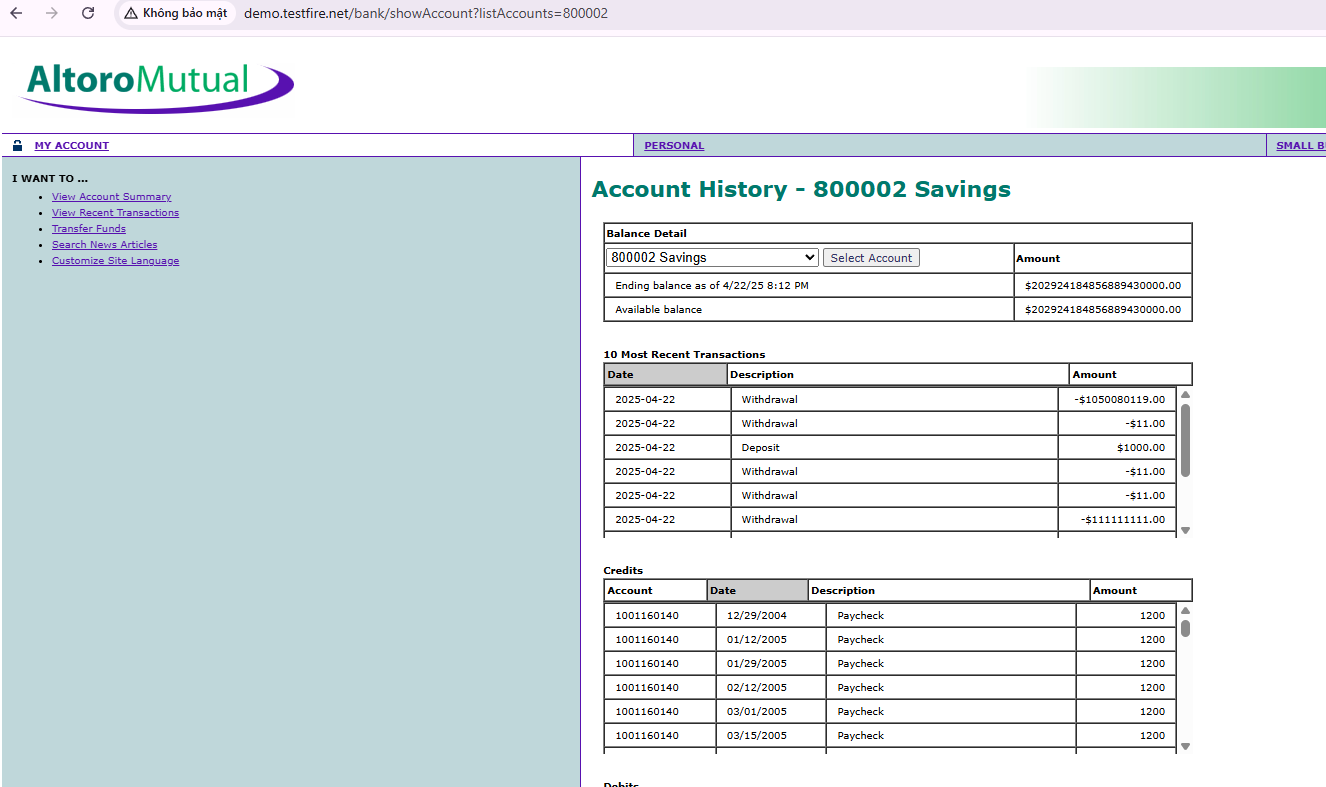




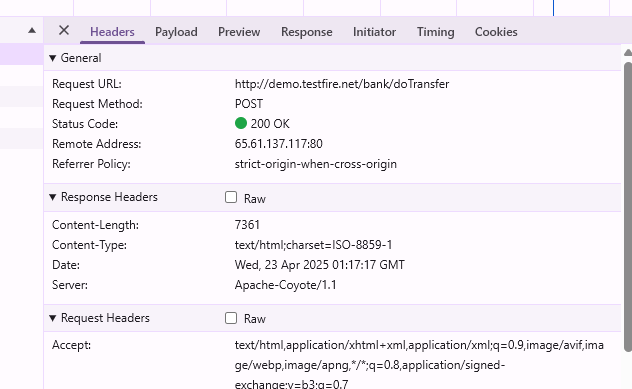
**Phần 2: Kiểm thử CSRF (2 điểm)**

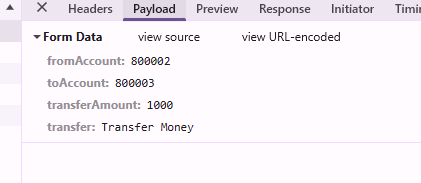
Trình bày các bước thực hiện và kết quả kiểm thử lỗ hổng CSRF trên website chọn 1 trong các trang sau để thử nghiệm

Truy cap demo.testfire.net và đăng nhập bằng username/password sau jsmith/demo1234

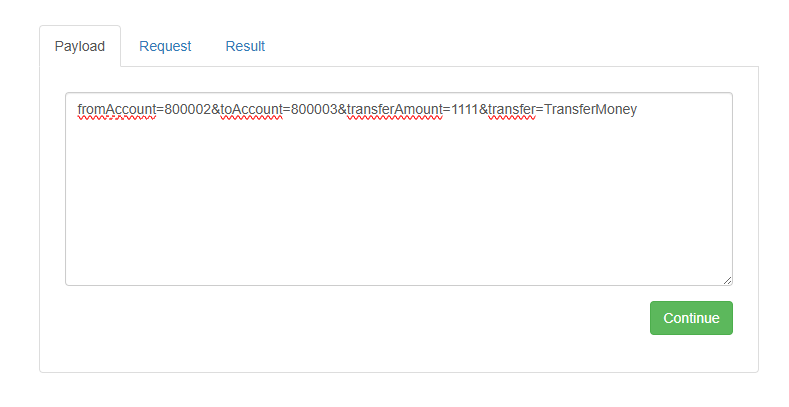


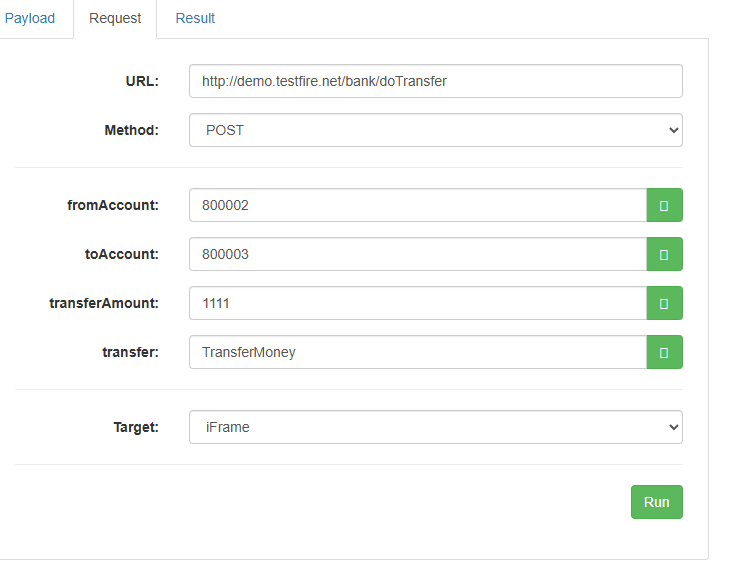
Thử 1 chức năng transfer và ta sẽ xem network



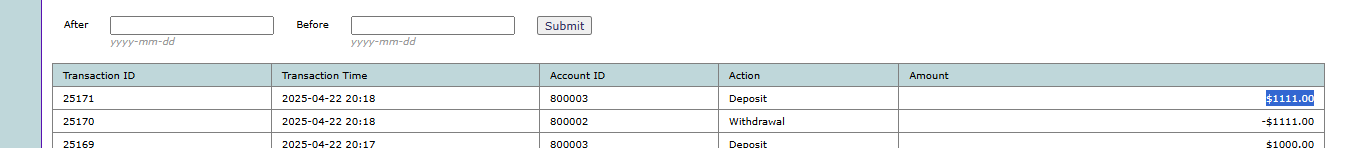


Sau khi xác định được các trường trong form ta tiến thành test





Kiểm tra lịch sử giao dịch trong transfer



Ta thấy web này có lỗ hổng CSRF