**BÁO CÁO BÀI THỰC HÀNH SỐ 9**

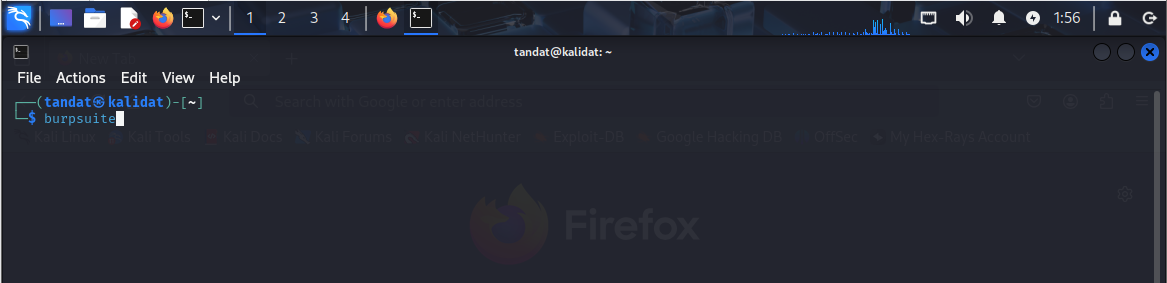
**KIỂM THỬ LỖ HỔNG XSS và CSRF**

Họ và tên: Nguyễn Tấn Đạt

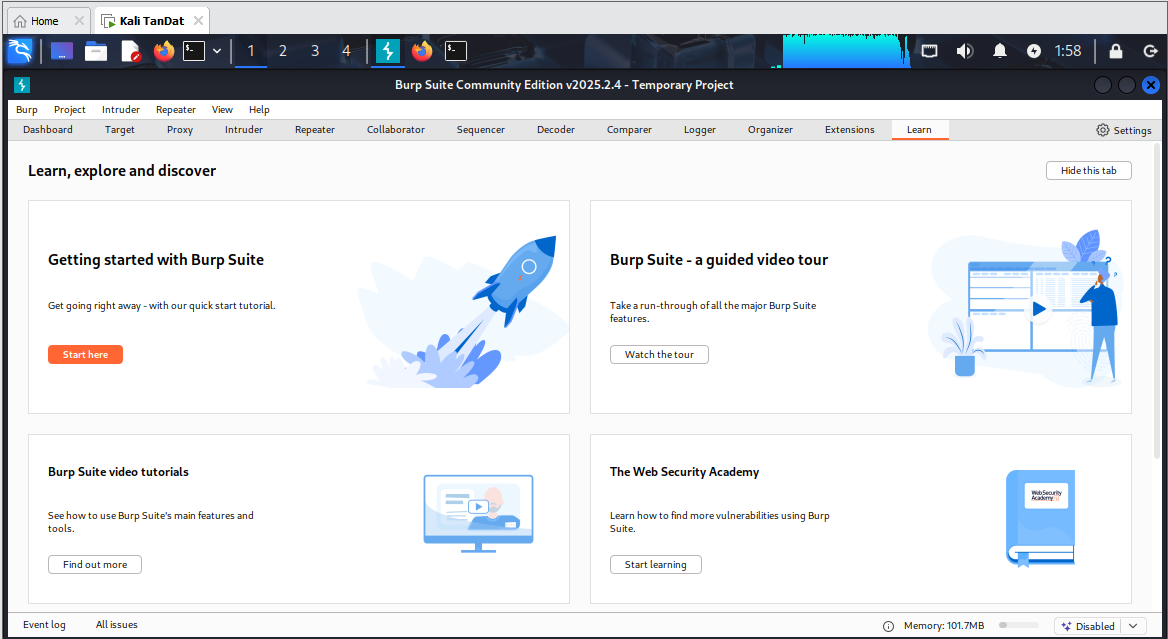
Yêu cầu kết quả thực hành:

Câu 1:

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

Đầu tiên truy cập 

Hiển thị giao diện

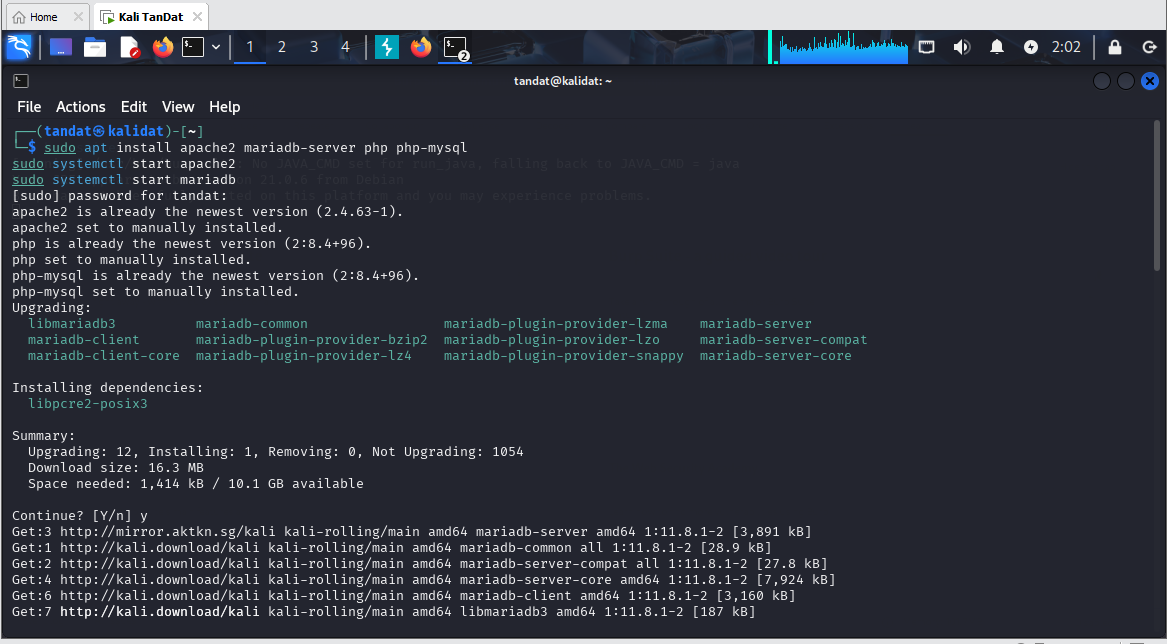


Cài đặt chạy lệnh :

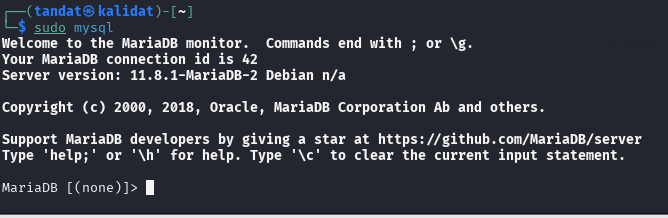
sudo apt install apache2 mariadb-server php php-mysql

sudo systemctl start apache2

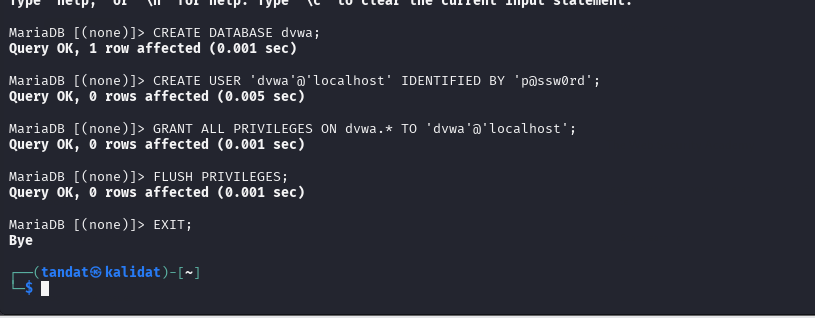
sudo systemctl start mariadb



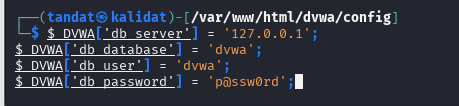
Cài đặt MySQL

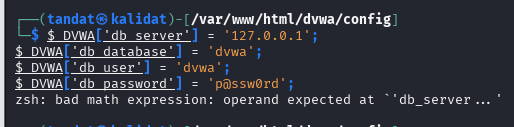


Tạo bảng

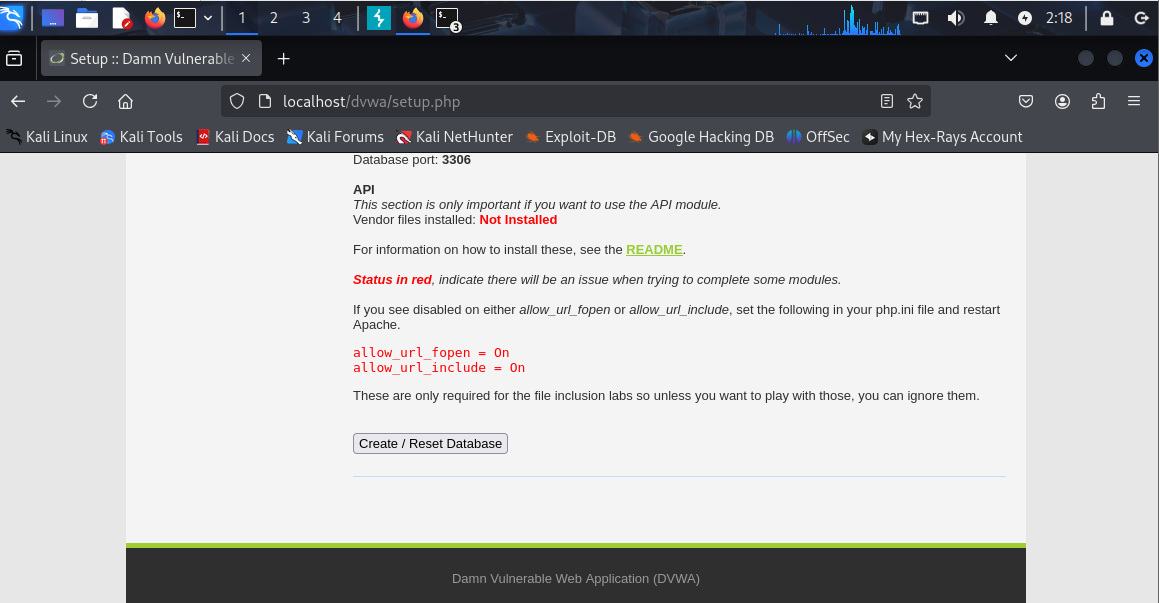


Sửa file config

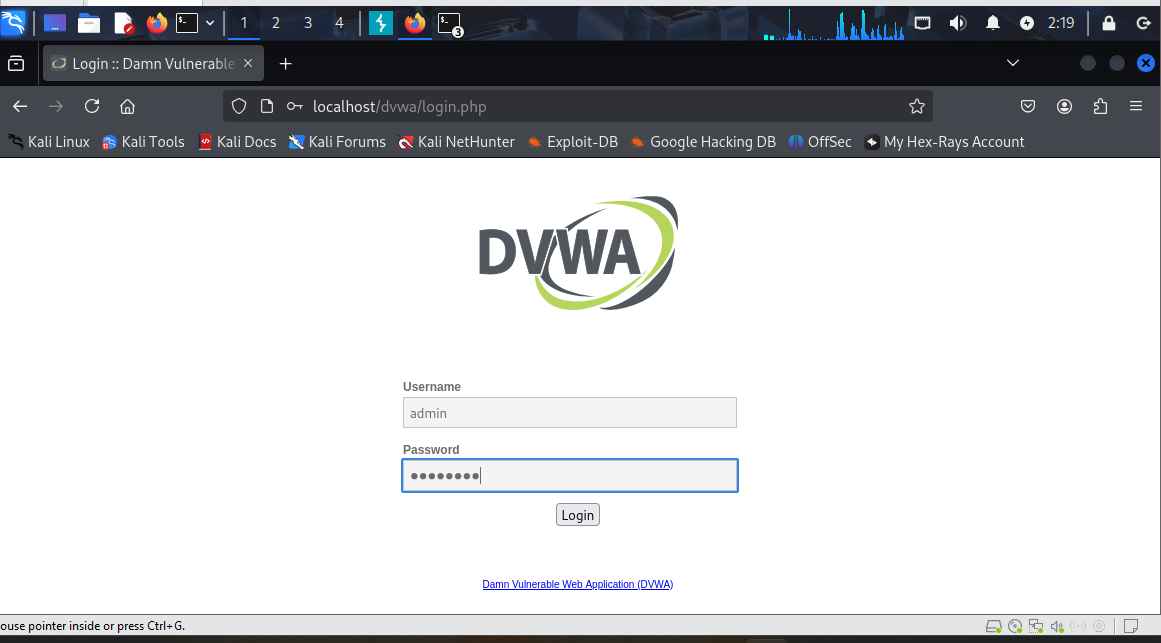




Tiếp theo truy cập <http://localhost/dvwa/setup.php>

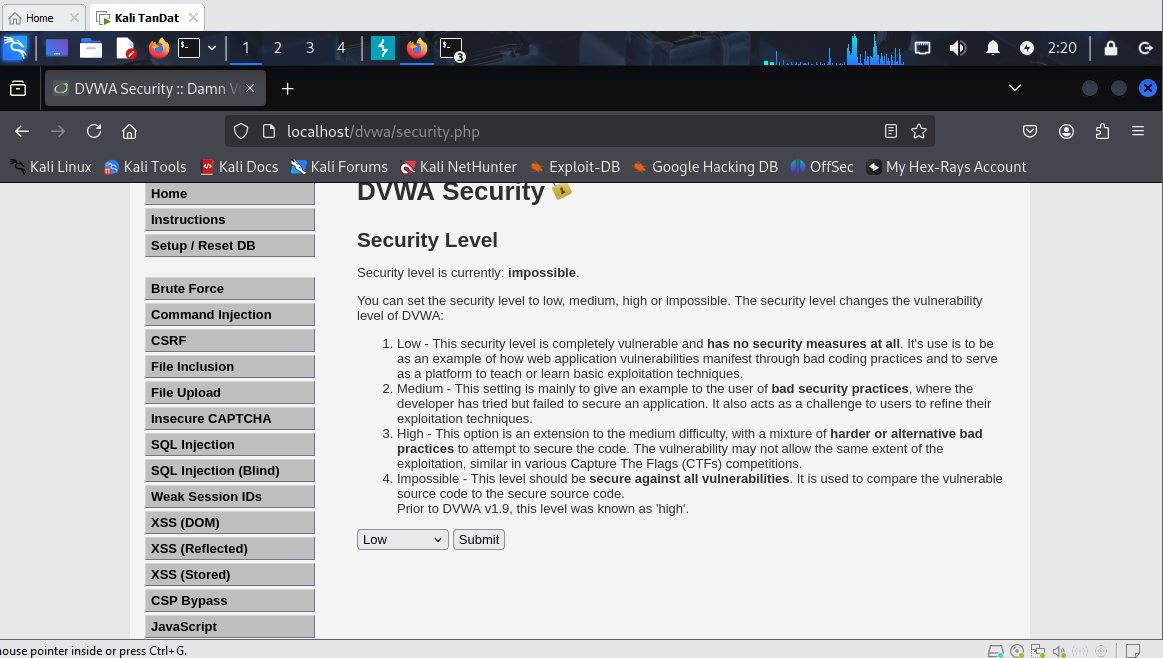


Ấn Create/Reset Database

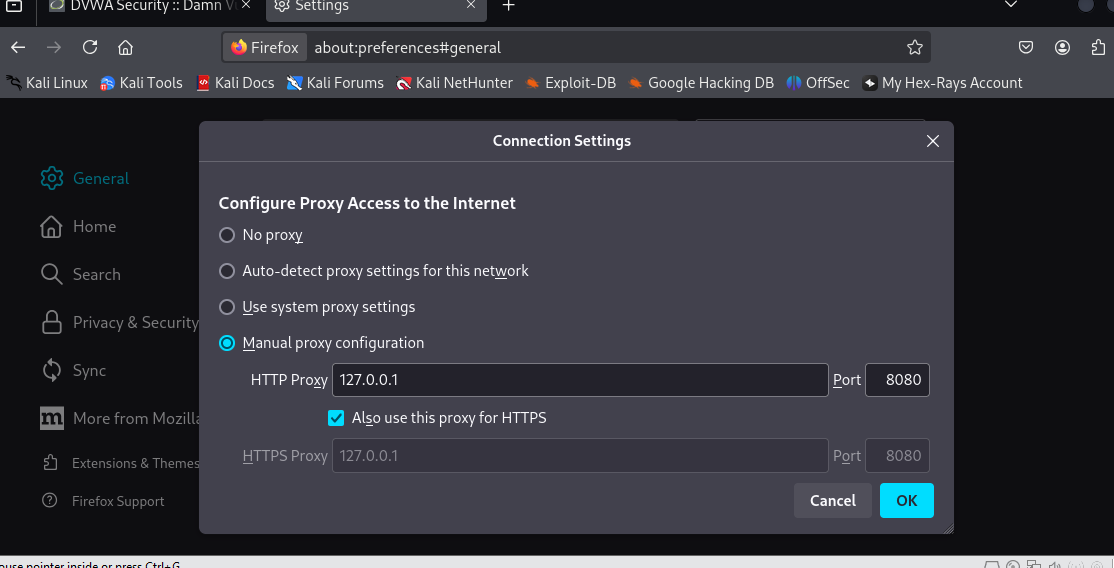


Thực hiện đăng nhập admin/password

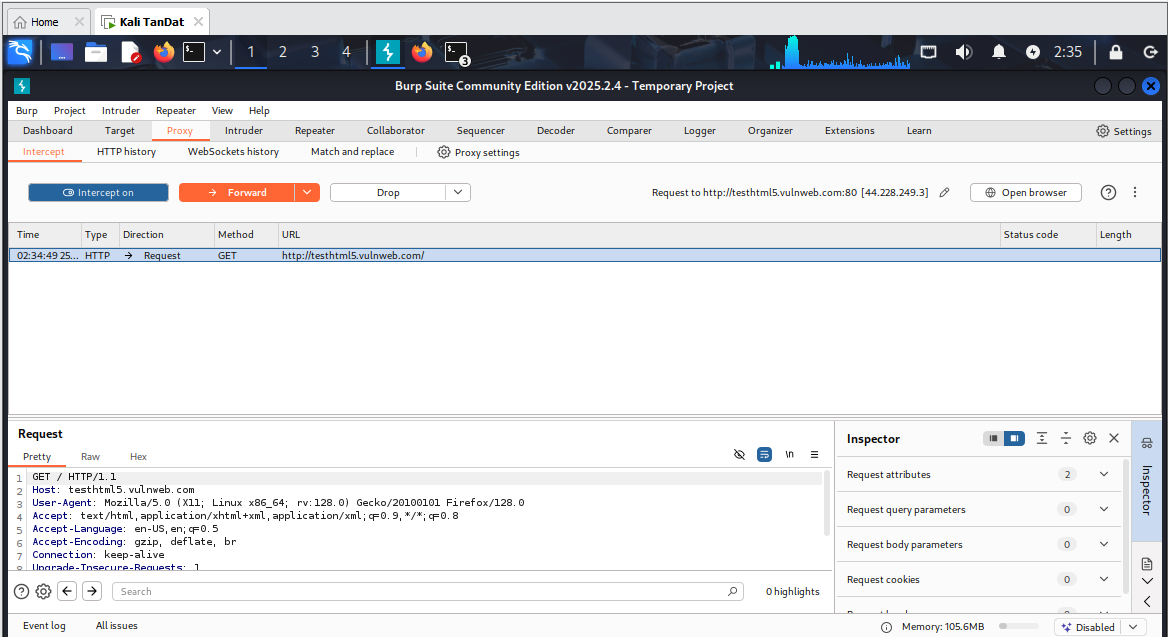
Sau đó chỉnh DVWA Security ở mức Low để dễ kiểm thử XSS và CSRF.



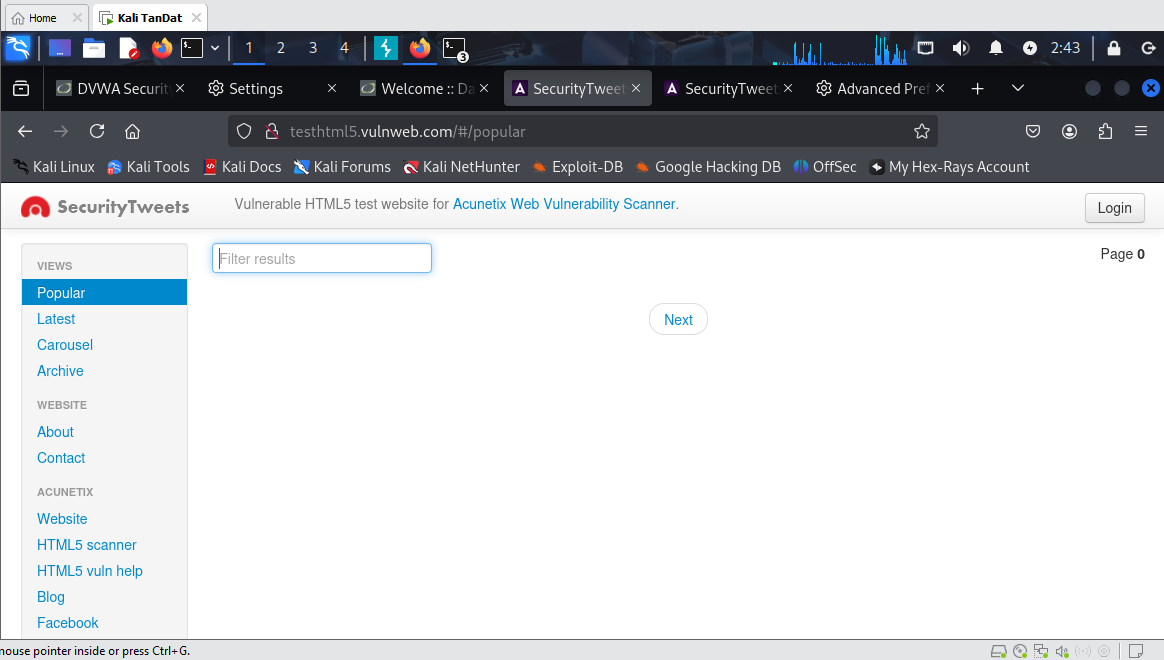
Tiếp theo settings Fire Fox



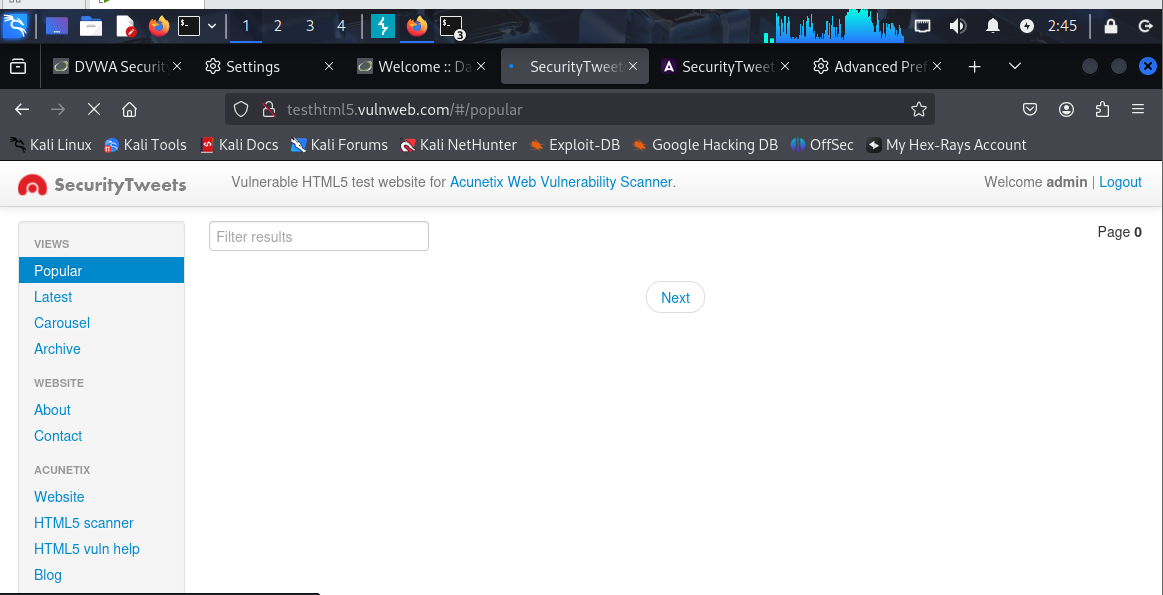
Kiểm thử XSS: http://testhtml5.vulnweb.com/

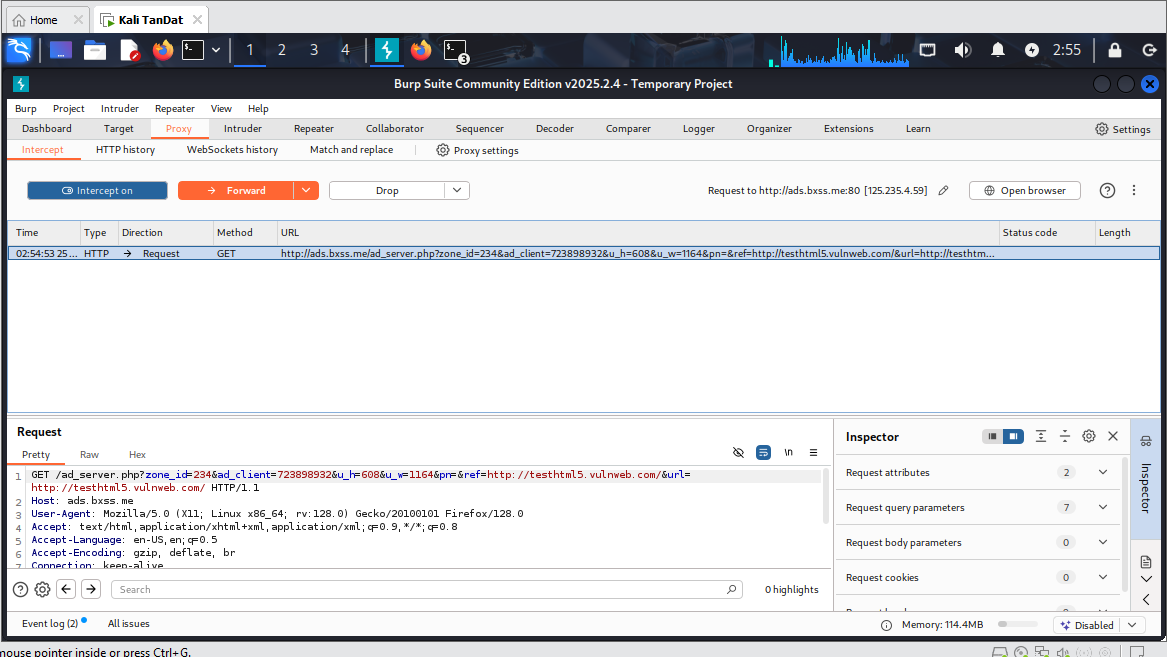


Truy cập vào web

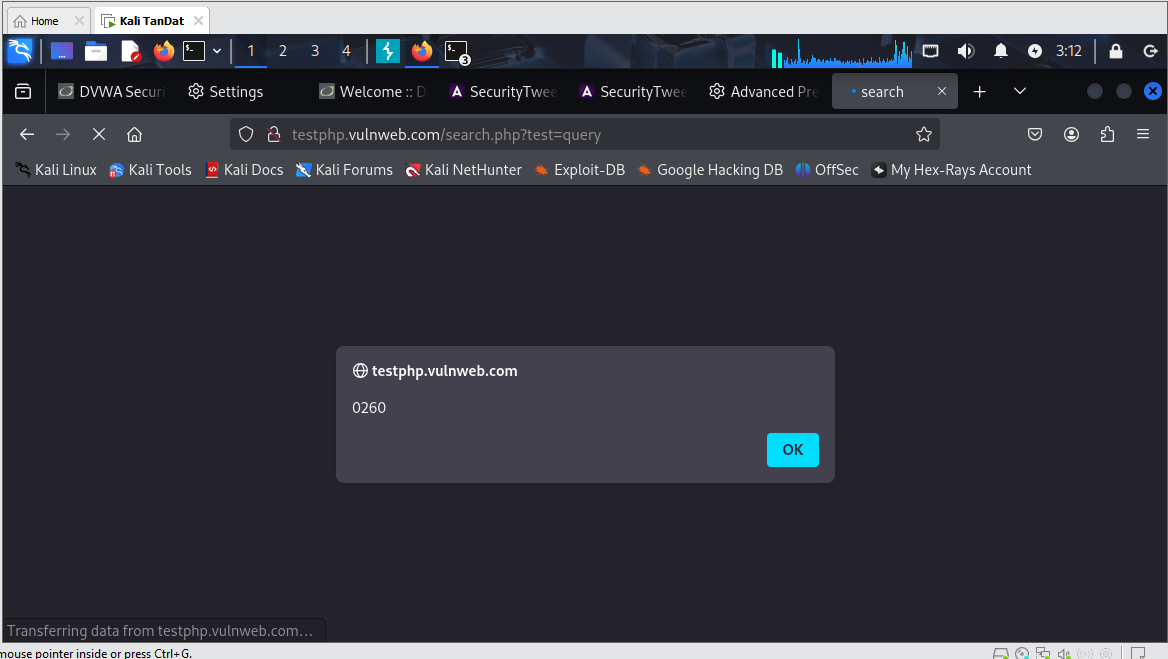


Login thành công





Nhập 4 số cuối mssv (0260) vào ô Search



-Xác định tham số đầu vào:

Tham số đầu vào: test.

Chi tiết:

* Trang http://testphp.vulnweb.com/search.php có một form tìm kiếm (hoặc URL) nhận tham số test qua phương thức GET.
* Khi bạn gửi payload <script>alert('0260');</script> qua URL

- Xác định giá trị đầu vào xuất hiện trong trang kết quả

Giá trị đầu vào xuất hiện:

* Sau khi gửi payload <script>alert('0260');</script> qua tham số test, trang đã thực thi mã JavaScript và hiển thị hộp thoại alert('0260').
* Điều này cho thấy giá trị đầu vào (<script>alert('0260');</script>) không được mã hóa hoặc lọc, mà được trình duyệt thực thi trực tiếp dưới dạng mã JavaScript.

- Các bước kiểm thử và phán đoán

Truy cập trang tìm kiếm:

* Mở trình duyệt và truy cập http://testphp.vulnweb.com/search.php.
* Xác định rằng trang có một ô tìm kiếm (hoặc có thể chỉnh sửa URL để thêm tham số test).

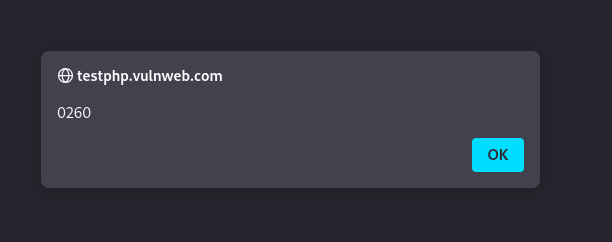
Thử nghiệm đầu vào đơn giản:

* Nhập test vào ô tìm kiếm và gửi.

**Thử payload XSS**:

* Nhập payload <script>alert('0260');</script> vào ô tìm kiếm và gửi.

Kết quả nhận được



- Giá trị kiểm thử cho thấy lỗ hổng và giải thích

Giá trị kiểm thử:

* Payload: <script>alert('0260');</script>.
* Kết quả: Hộp thoại alert('0260') xuất hiện, chứng minh rằng mã JavaScript đã được thực thi.

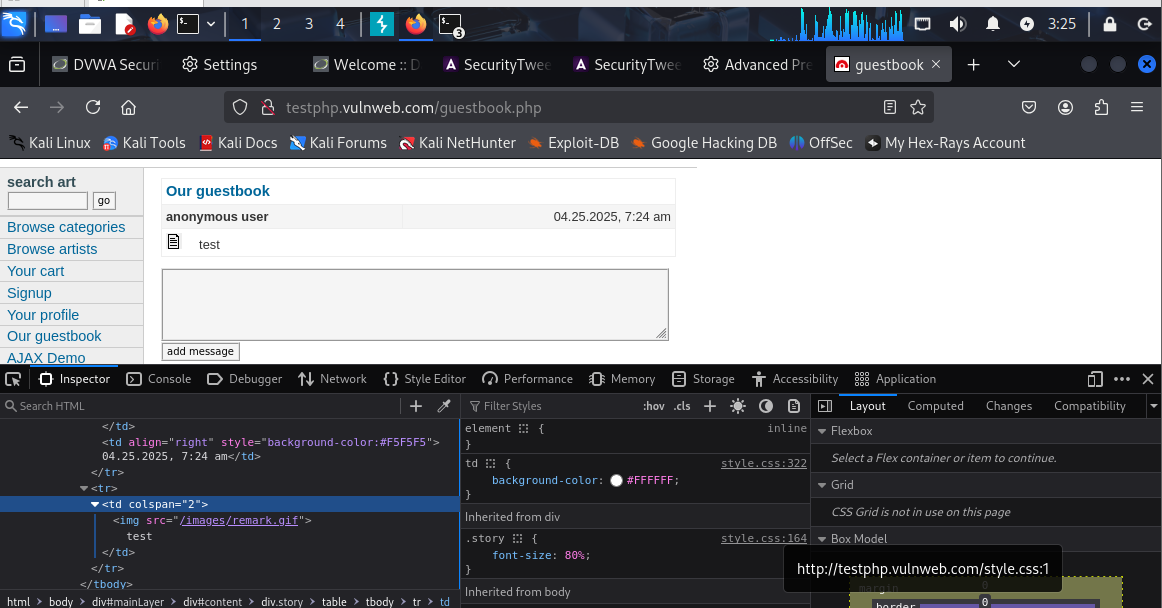
Giải thích:

* Lỗ hổng Reflected XSS tồn tại vì trang http://testphp.vulnweb.com/search.php không mã hóa hoặc lọc tham số test trước khi phản ánh dữ liệu đầu vào trong phản hồi.
* Khi tham số test chứa mã JavaScript, mã này được trình duyệt thực thi ngay lập tức, dẫn đến việc hiển thị hộp thoại alert('0260').
* Điều này cho thấy trang không áp dụng các biện pháp bảo vệ như:
  + Mã hóa ký tự đặc biệt (chuyển < thành <, > thành >).
  + Loại bỏ hoặc vô hiệu hóa các thẻ nguy hiểm như <script>.
* Hệ quả: Kẻ tấn công có thể sử dụng lỗ hổng này để chèn mã JavaScript độc hại

Câu 2: Thực hiện kiểm thử lỗ hổng XSS để thực thi đoạn mã hiển thị thông báo có chứa tên của sinh viên.

Tham số đầu vào: comment (POST)

Nhập dòng test vào ô



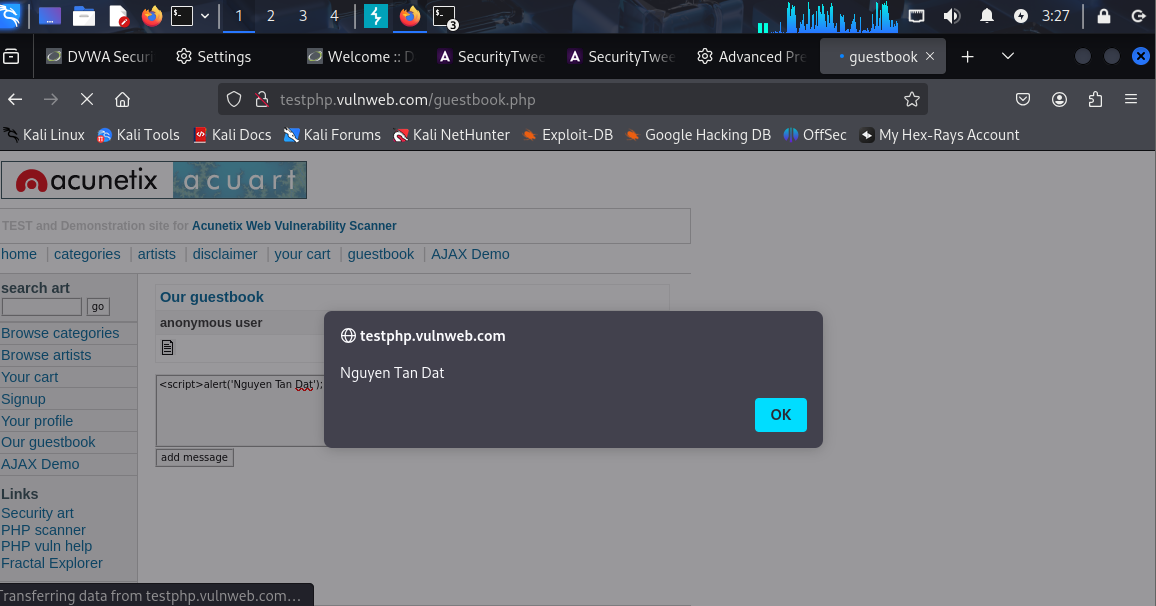
test không được mã hóa (không chuyển thành <test>), trường này dễ bị Stored XSS

**Thử payload Stored XSS**:

Nhập payload để hiển thị tên sinh viên (Nguyen Tan Dat)

Điền form:

* Comment: <script>alert('Nguyen Van A');</script>
* Gửi.



Giá trị kiểm thử cho thấy lỗ hổng và giải thích (0.5 điểm)

* Giá trị kiểm thử:
  + Payload: <script>alert('Nguyen Tan DaT');</script>.
  + Kết quả: Hộp thoại alert('NguyeN Tan Dat') xuất hiện mỗi khi tải trang <http://testphp.vulnweb.com/guestbook.php>.

Giải thích:

* Lỗ hổng Stored XSS tồn tại vì trang http://testphp.vulnweb.com/guestbook.php không mã hóa hoặc lọc tham số comment trước khi lưu vào cơ sở dữ liệu và hiển thị lại.
* Khi tham số comment chứa mã JavaScript, mã này được lưu trên server và thực thi cho tất cả người dùng truy cập trang, dẫn đến việc hiển thị hộp thoại alert('Nguyen Tan Dat').
* Điều này cho thấy trang không áp dụng các biện pháp bảo vệ như:
  + Mã hóa ký tự đặc biệt (chuyển < thành <, > thành >).
  + Loại bỏ hoặc vô hiệu hóa các thẻ nguy hiểm như <script>.
* Hệ quả: Kẻ tấn công có thể chèn mã JavaScript độc hại

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