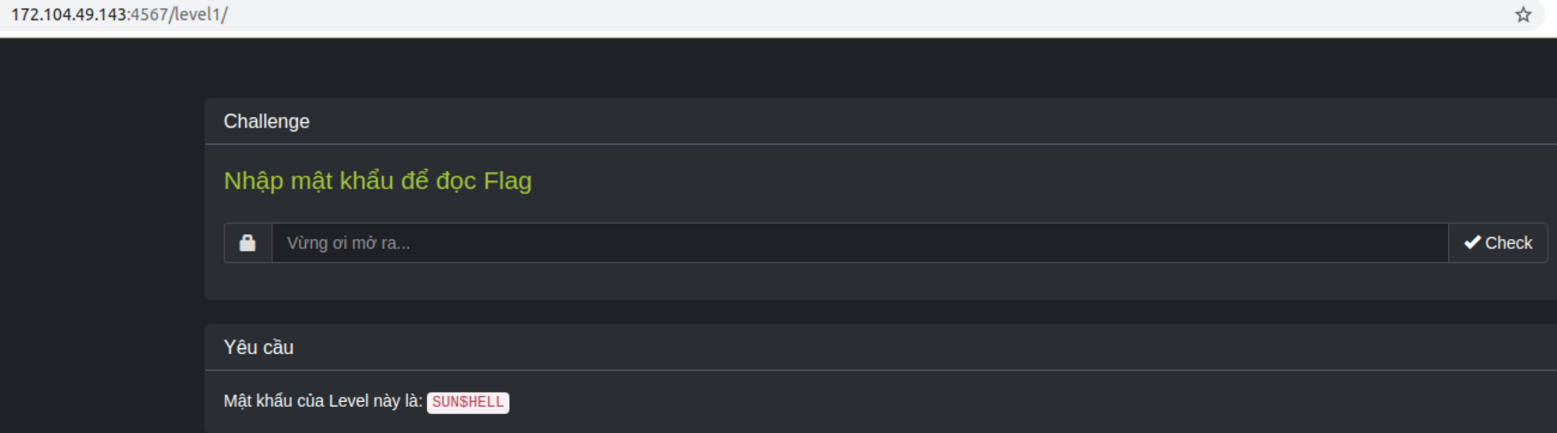
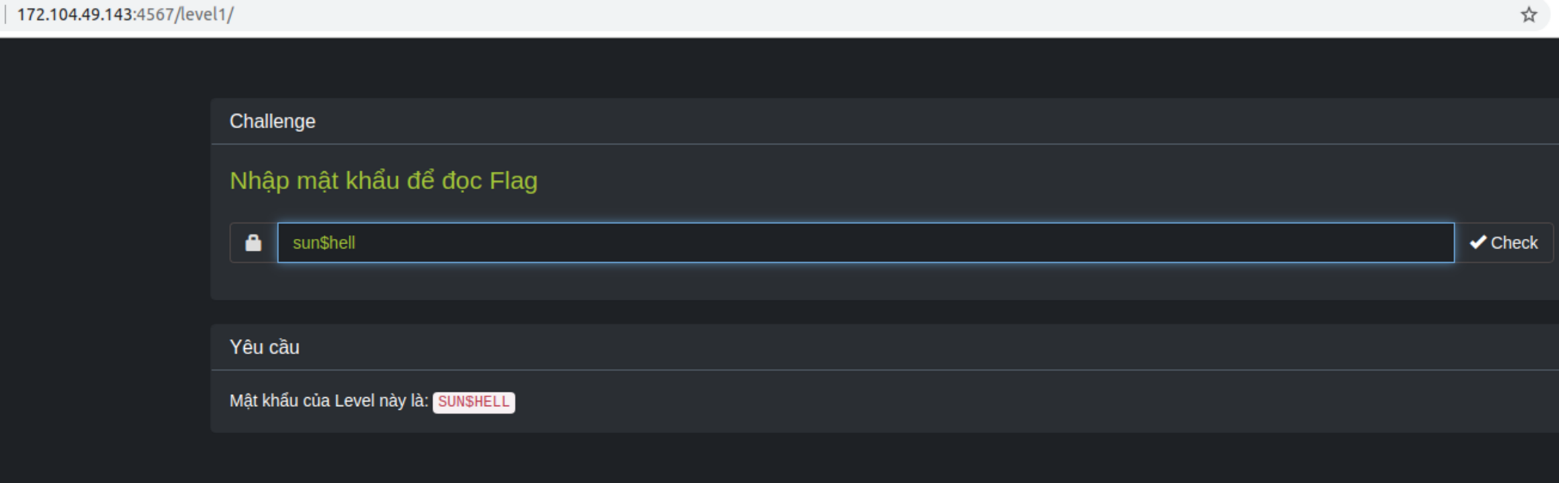
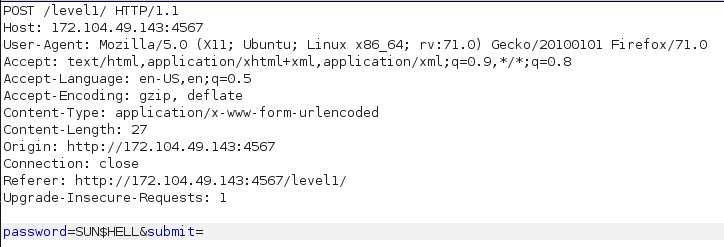
Mình được 1 đứa bạn giới thiệu cho Viblo CTF tại địa chỉ: ctf.viblo.asia cũng khá lâu. Giới thiệu qua thì viblo.asia là sản phẩm của công ty Sun\*, bao gồm rất nhiều sub domain và ctf là 1 trong số đó (mình cũng từng apply vào team cybersecurity của Sun\* 2 lần, 1 lần thì mình reject phỏng vấn do bận, lần sau thì tạch :v ). Thôi không dài dòng nữa, bắt đầu với phần chính luôn. Mảng của mình muốn theo là pentest nên trong bài sẽ toàn là writeup mảng web, tuy nhiên cũng có 3 bài mình chưa làm được :v. Các challenge trên Viblo CTF sẽ được chia thành các rank từ E-S theo mức độ khó tăng dần.

**1. Rank E**

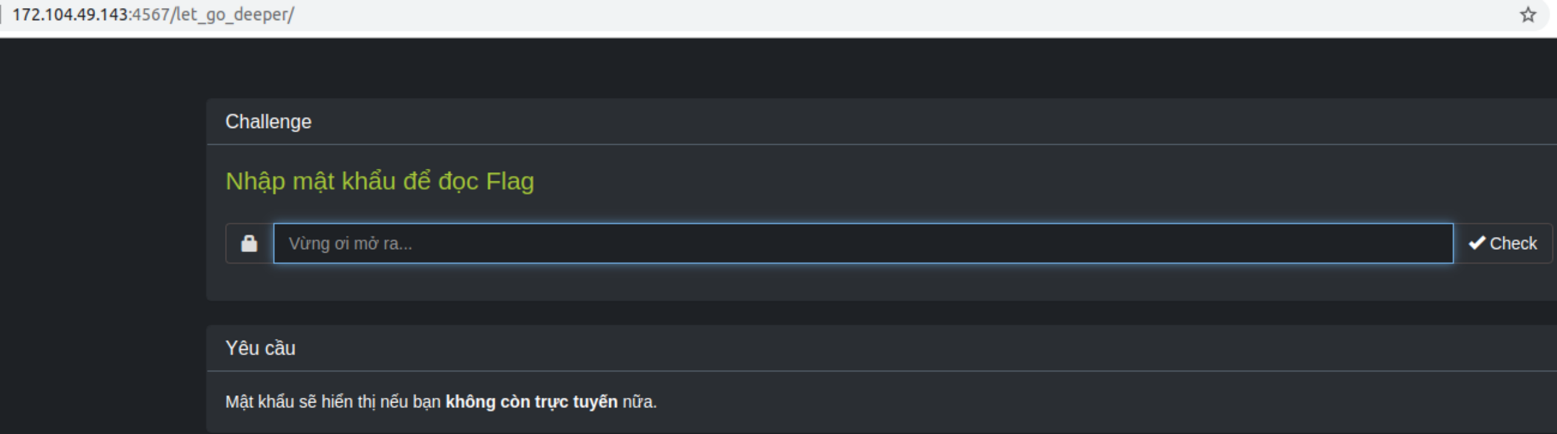
**1.1. Tricky Sneaky Weby (100)**



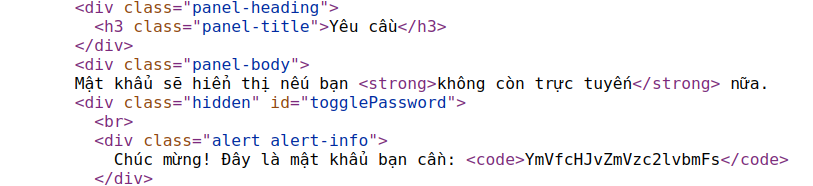
Nhìn URL thì mình cũng đoán là bài này có khá nhiều level :v. Với level 1 thì yêu cầu rất đơn giản, nhập password để xem flag và đã cho sẵn password. Tuy nhiên, khi nhập password thì toàn bộ chữ cái sẽ bị đổi thành chữ thường.

Để vượt thì mình dùng burpsuite bắt lại request rồi sửa thôi

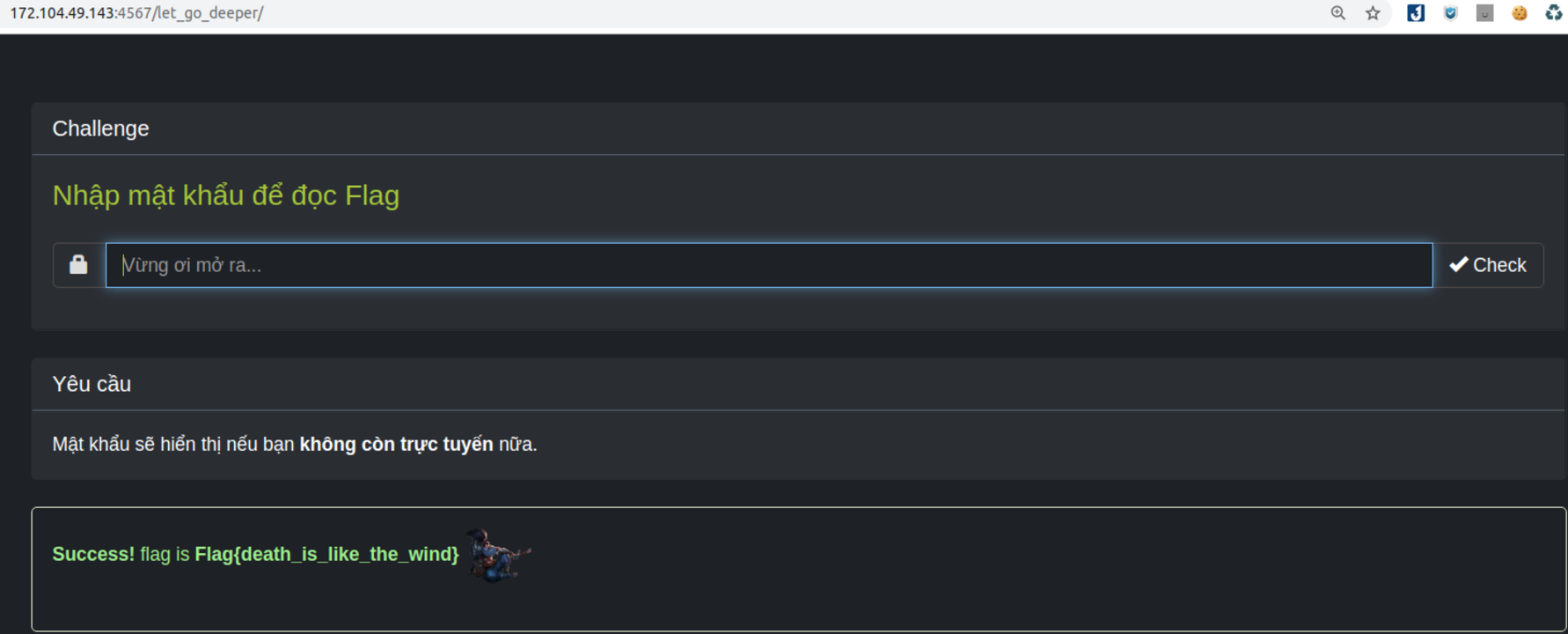
Foward request cho server thì sẽ đến được level tiếp theo



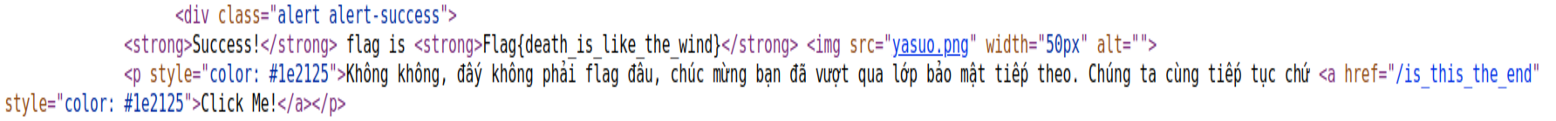
Với level này thì yêu cầu là không còn trực tuyến nữa, tuy nhiên theo thói quen thì mình vẫn xem qua source của web và cũng không ngạc nhiên lắm, password nằm trong đó rồi :v



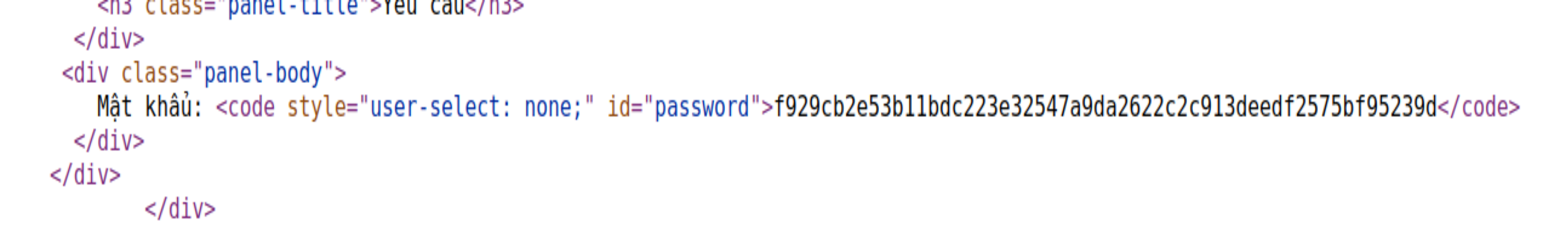
Nhận thấy đây là 1 đoạn mã base64, decode ra ta có password: be\_professional.



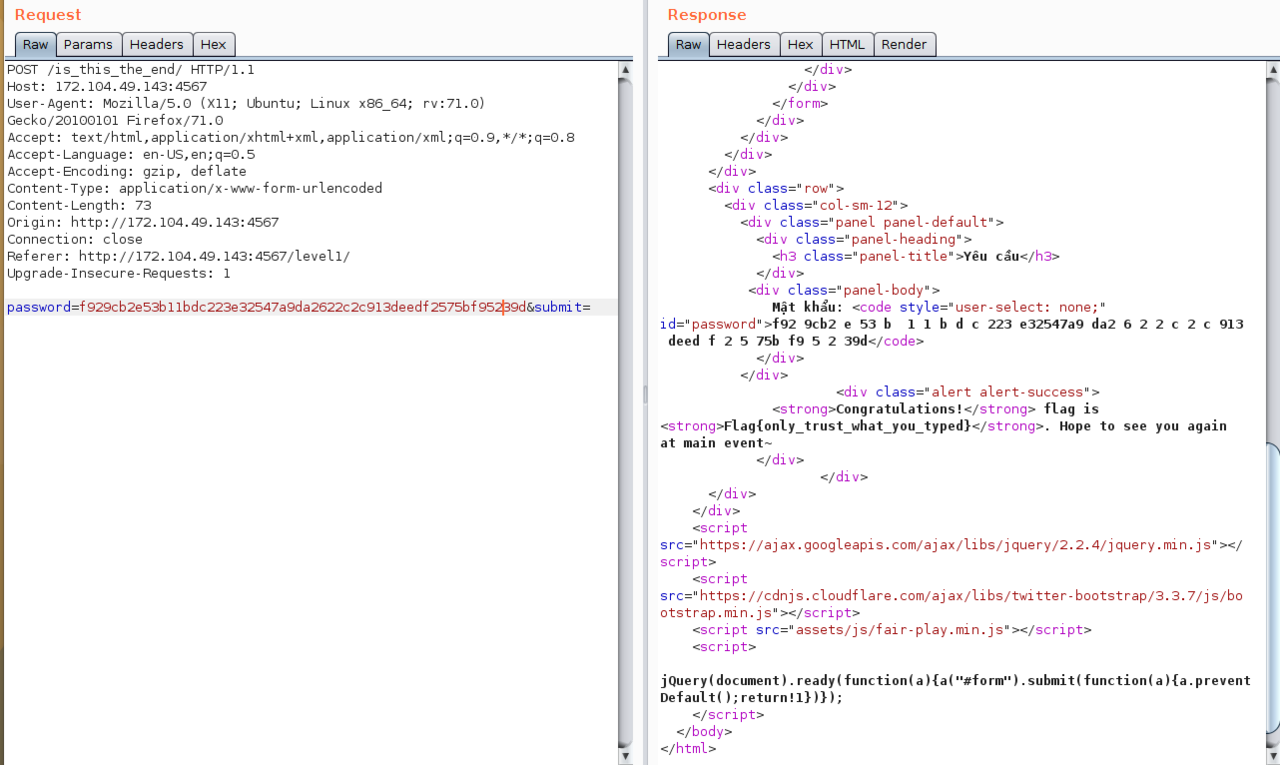
Đến đấy khá nhiều bạn tưởng rằng đã xong nhưng không, tiếp tục check source để biết rằng đây là 1 cú lừa và đến level tiếp theo.



Level tiếp theo cũng giống với level 1 khi đã có sẵn password chỉ cần nhập thôi, tuy nhiên có vẻ thao tác click chuột của chúng ta đã bị chặn. Nhiều bạn tìm cách bỏ chặn tuy nhiên mình lại dùng phím tắt Ctrl-I để xem source :v

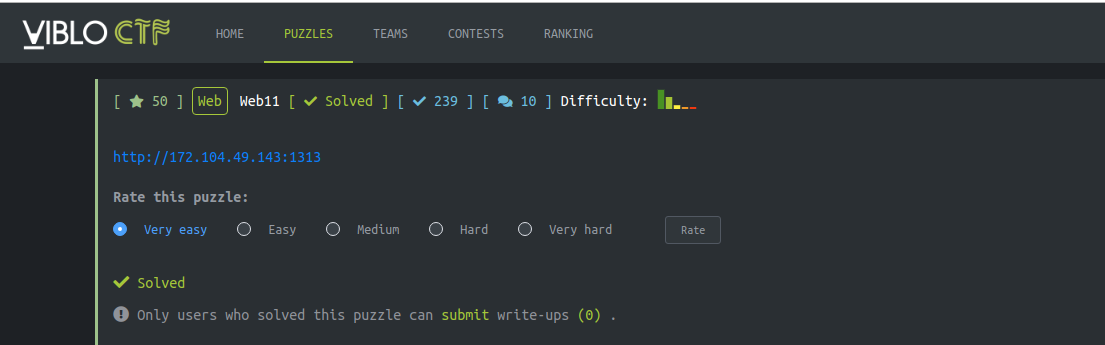


Lúc này chúng ta có thể dễ dàng copy lại password, tuy nhiên vì thao tác click chuột đã bị chặn nên mình tiếp tục dùng burp suite để gửi request lên server:



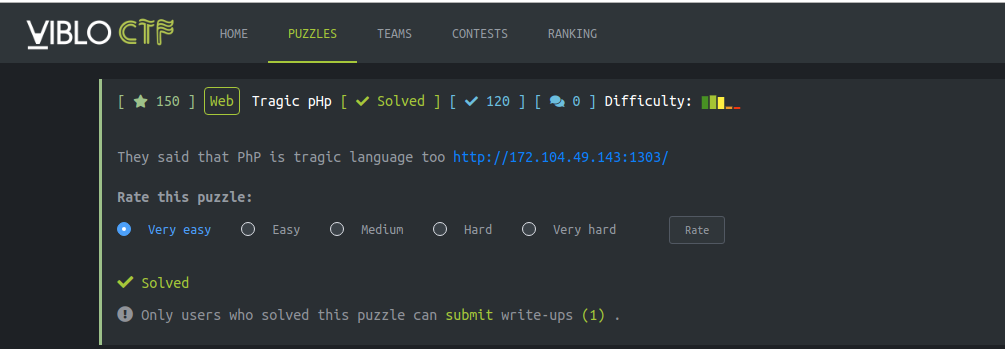
Ở bước này lưu ý nếu các bạn copy paste mật khẩu vào burp suite thì sẽ thấy các ký tự lạ xen giữa trong password, đó chính là các bit unicode. Bình thường thì chrome sẽ tự chuyển các ký tự unicode này sang ascii và hiển thị, trong trường hợp này unicode của các bit null đã được chèn vào giữa nên khi nhìn bình thường sẽ chẳng khác gì 1 đoạn text cả. Xóa hết các ký tự lạ trong password và gửi request lên chúng ta sẽ có được flag.

    1.2. Web 11 (50)

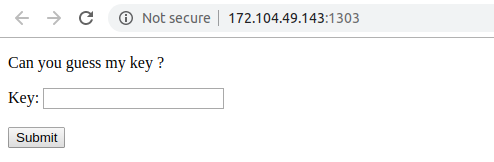


Bài này cho web khá đơn giản. Flag sẽ nằm trong ảnh đầu tiên nháy lên khi tải trang. Vì flag cũng khá ngắn nên mình quyết định f5 vài lần để lấy flag cho nhanh :v

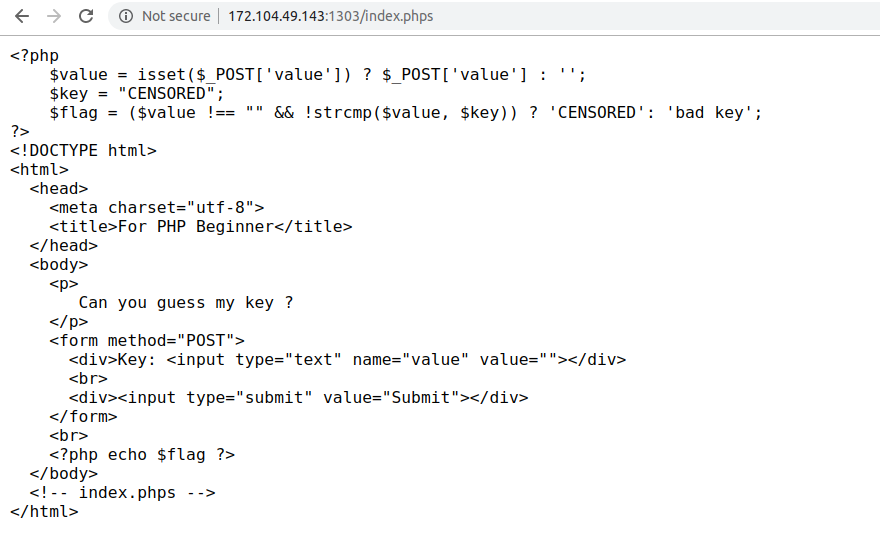
**1.3 Tragic pHp (150)**



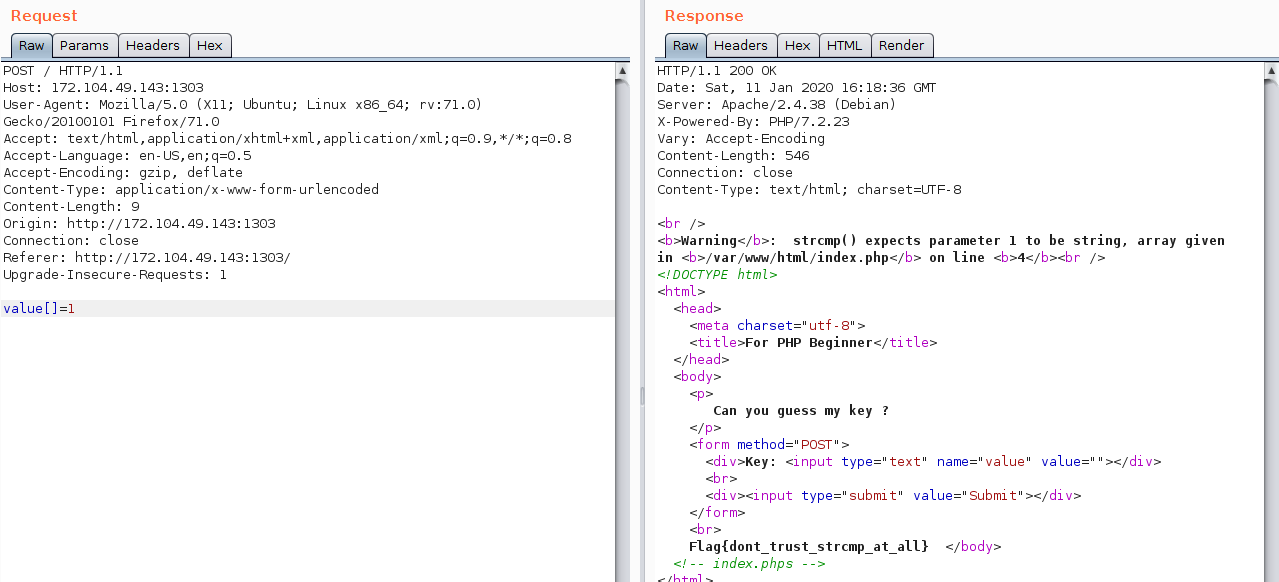
Bài này yêu cầu chúng ta đoán key để xem flag. Mình thử 1 vài cái thông dụng như admin, password, user,... nhưng có vẻ đều không có tác dụng.



Tuy nhiên khi xem source, chúng ta sẽ có chỉ dẫn để đi đến index.phps để xem source php của trang.

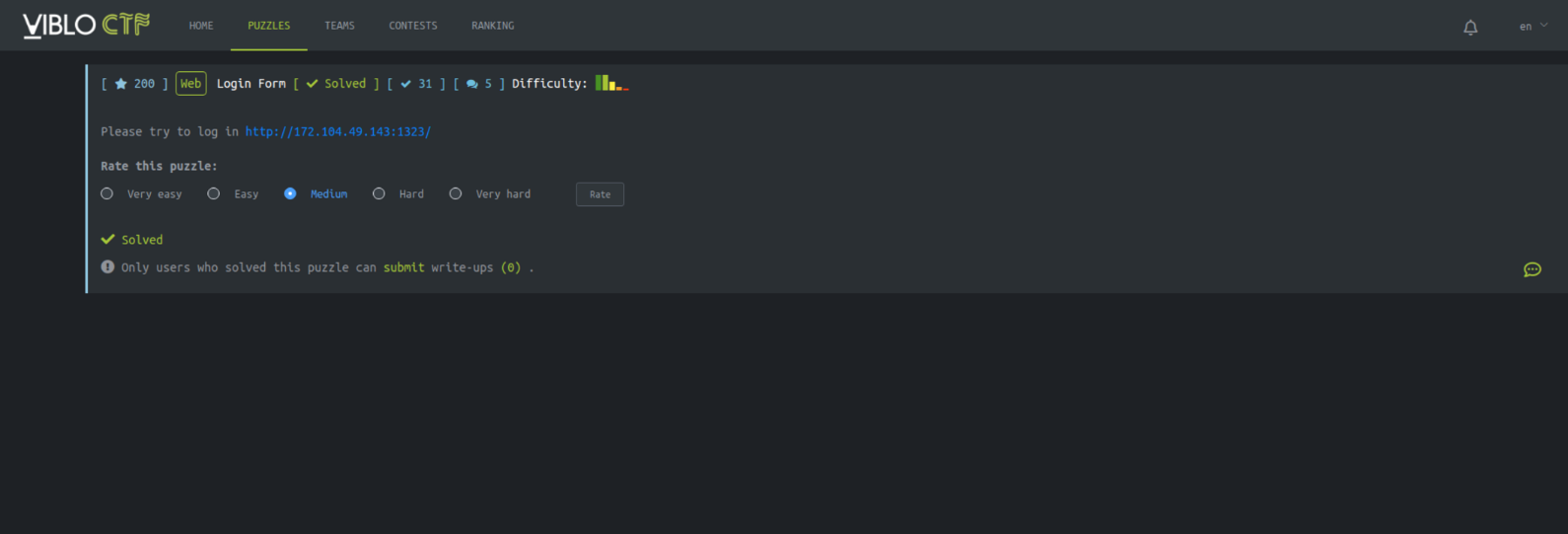


Đoạn code này sẽ lấy dữ liệu từ trường value trong request post gửi lên và so sánh nó với biến $key, nếu giống thì sẽ hiện ra gán giá trị cho biến $flag. Tuy nhiên vì đều không biết giá trị của 2 biến này nên chúng ta sẽ phải tìm cách bypass hàm strcmp (hàm dùng để so sánh chuỗi trong php). Google 'strcmp php bypass' sẽ thấy rất nhiều hướng dẫn. Đơn giản chỉ cần truyền lên 1 mảng khi đó hàm strcmp sẽ phải so sánh 2 biến có kiểu dữ liệu khác nhau => trả về Null. Đoạn code bên trên sử dụng !strcmp($value, $key) tương đương với strcmp($value, $key) == 0. Toán tử == trong php sẽ trả về true nếu thực hiện phép so sánh Null == 0 => bypass được hàm strcmp. Mình sẽ sử dụng burp suite thực hiện bắt, sửa request sau đó gửi lên server.

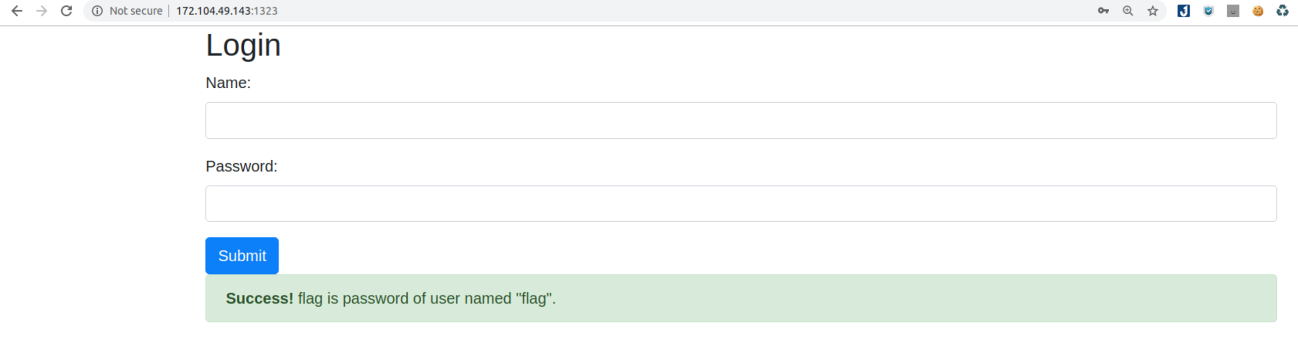


**2. Rank D**

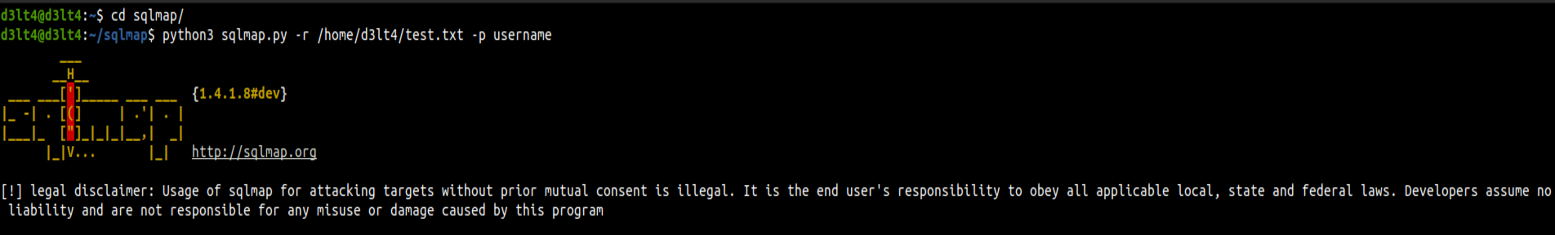
**2.1. Login Form (200)**



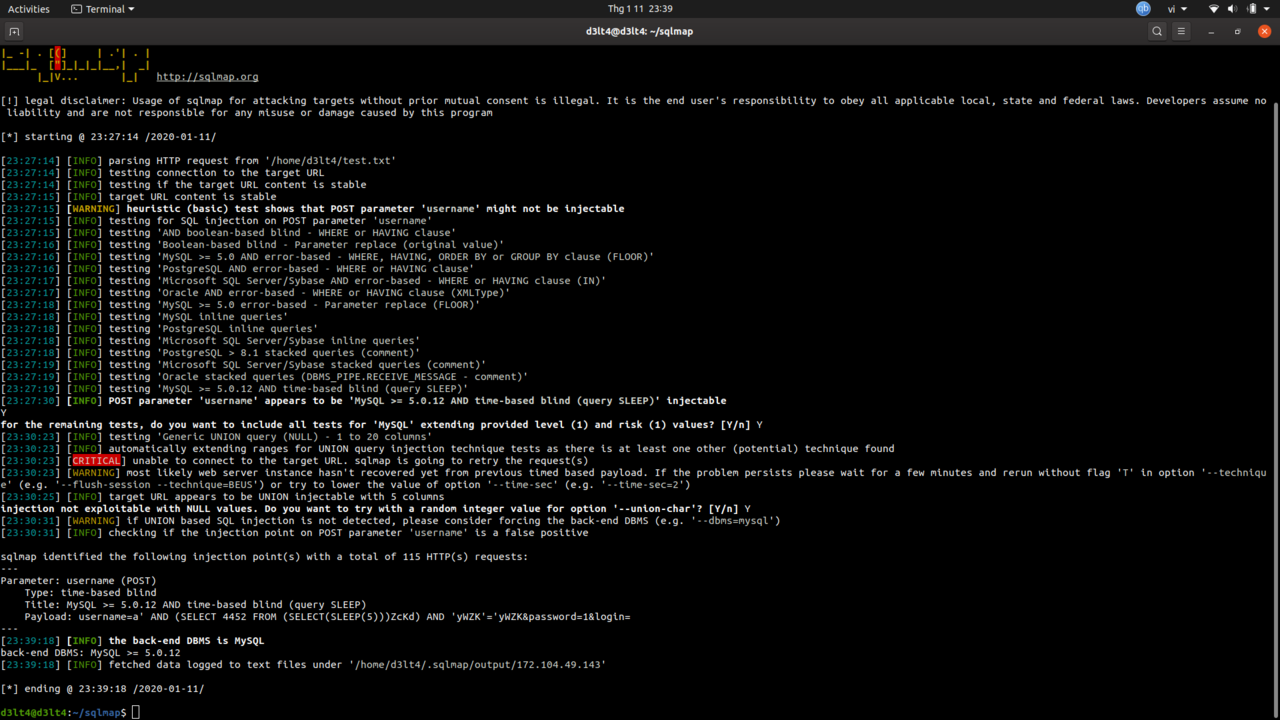
Đề bài cho 1 web login, mình đã thử 1 vài cặp username-password thông dụng và xem source các thứ nhưng không có kết quả. Đến đây mình bắt đầu thử 1 lỗi mà các form login thường rất hay dính. Đó là SQL injection. Thử với payload login bypass kinh điển là a' or 1=1 # thì mình confrim là web này dính SQLi. Việc cần là dump database ra để lấy flag là password của user flag thôi.



Đến đây mình sẽ dùng tool sqlmap để scan, vì bài này thuộc dạng Blind SQli và mình không đủ kiên nhẫn để test tay từng chữ :v. Save request post ra 1 file text sau đó chạy sqlmap thôi.

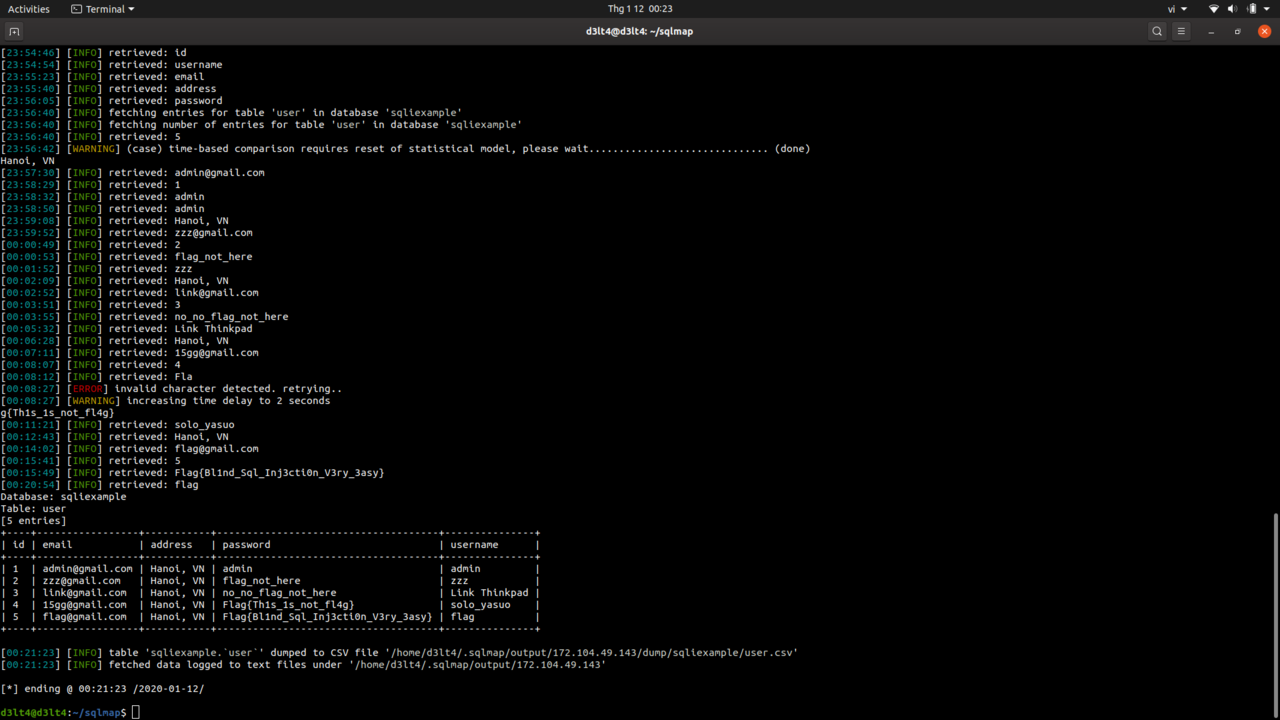


Kết quả trả về:



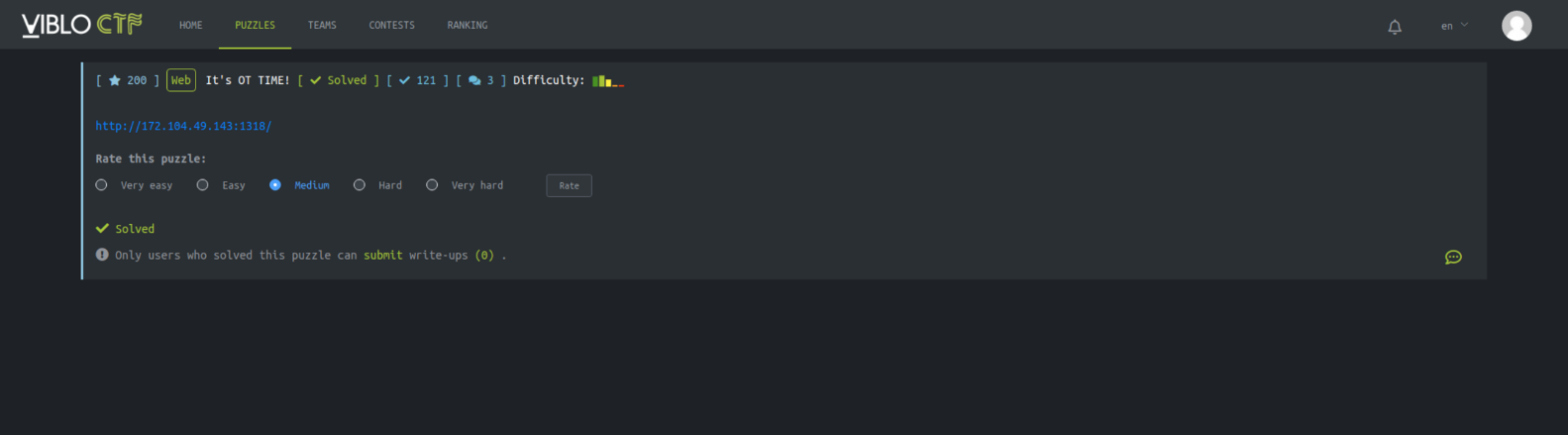
Sau khi xác định lỗi, mình sẽ sử dụng option --dbs để tìm các database có trên server:

3 database information\_schema, mysql và performance\_schema đều là database hệ thống của mysql nên mình sẽ dump database còn lại - sqliexample ra:



(Blind SQLi khai thác bằng tool cũng rất lâu :v. Bài này đơn giản mà scan cũng mất tầm 40 phút :v. Bên cạnh đó, nếu tận dụng đc hết các option của SQLmap thì đây là 1 tool khai thác SQLi rất mạnh)

**2.2. It's OT TIME! (200)**



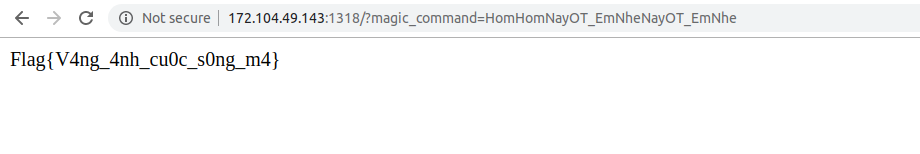
Web đề bài cho đã cho chúng ta xem sẵn source, việc còn lại là phân tích xem source làm gì thôi



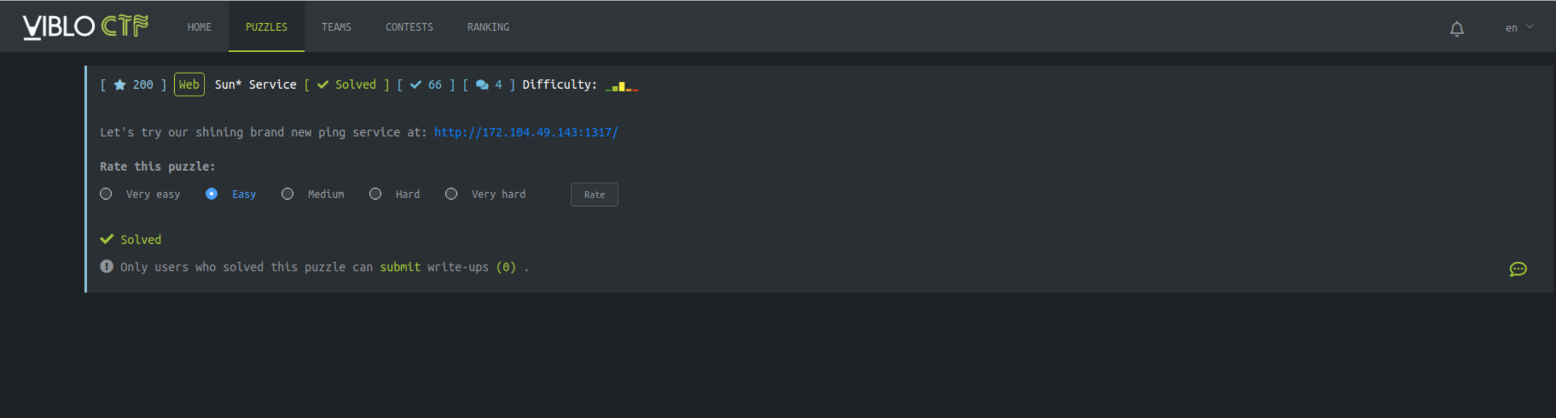


Đoạn code trên sẽ lấy dữ liệu từ giá trị magic\_command mà chúng ta gửi lên thông qua GET method bằng URL và lưu vào biến $what\_he\_said. Sau đó sẽ xóa toàn bộ string $what\_you\_dont\_want\_to\_hear = 'HomNayOT\_EmNhe' trong string $what\_he\_said bằng hàm preg\_replace. Và so sánh string sau khi xóa với string cần xóa, nếu giống nhau thì sẽ cho chúng ta flag. Nghe rất vô lý đúng không :v

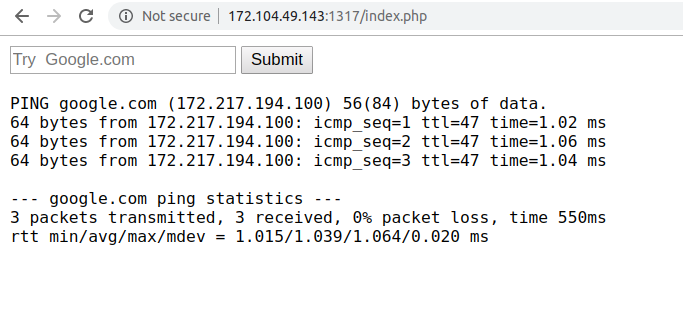
Loay hoay google 1 hồi mình mới hiểu cách hoạt động của hàm preg\_replace. Hàm này có 1 đặc điểm là sau khi thay thế nó sẽ không duyệt lại từ đầu string mà sẽ tiếp tục duyệt cho đến hết. Nếu như string sau khi thay thế lần đầu trở thành string cần thay thế thì hàm preg\_replace sẽ không kiểm tra được. Do đó payload của mình sẽ là ?magic\_command=HomHomNayOT\_EmNheNayOT\_EmNhe. Và đây là kết quả:



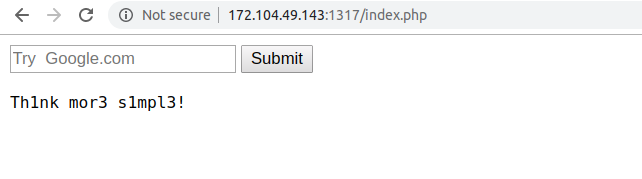
**2.3  Sun\* Service (200)**

****

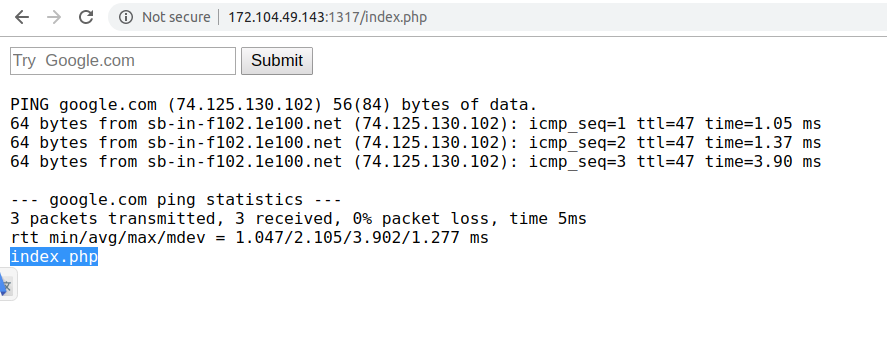
Web đề bài cho đúng như cái tên, nhập 1 URL vào và web sẽ cho ra kết quả ping đến URL đó:



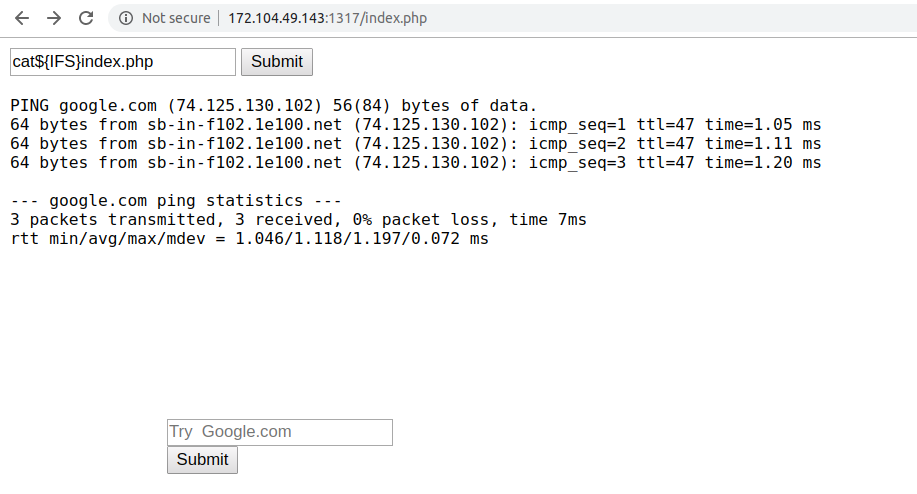
Để ý kết quả trả về khá giống với khi thực hiện lệnh ping trên linux nên mình đoán web sẽ lấy dữ liệu mà chúng ta gửi lên sau đó lưu vào 1 biến nào đó, ở đây mình ví dụ là biến $a. Sau đó sẽ có 1 string dạng $b="ping '$a'", string này sẽ ghép giá trị của biến $a vào để có 1 lệnh command line trong linux. Vì thế mình nghĩ rất có thể web này dính lỗi Command injection. Tuy nhiên, khi thử 1 vài payload cơ bản thì lại không cho kết quả:



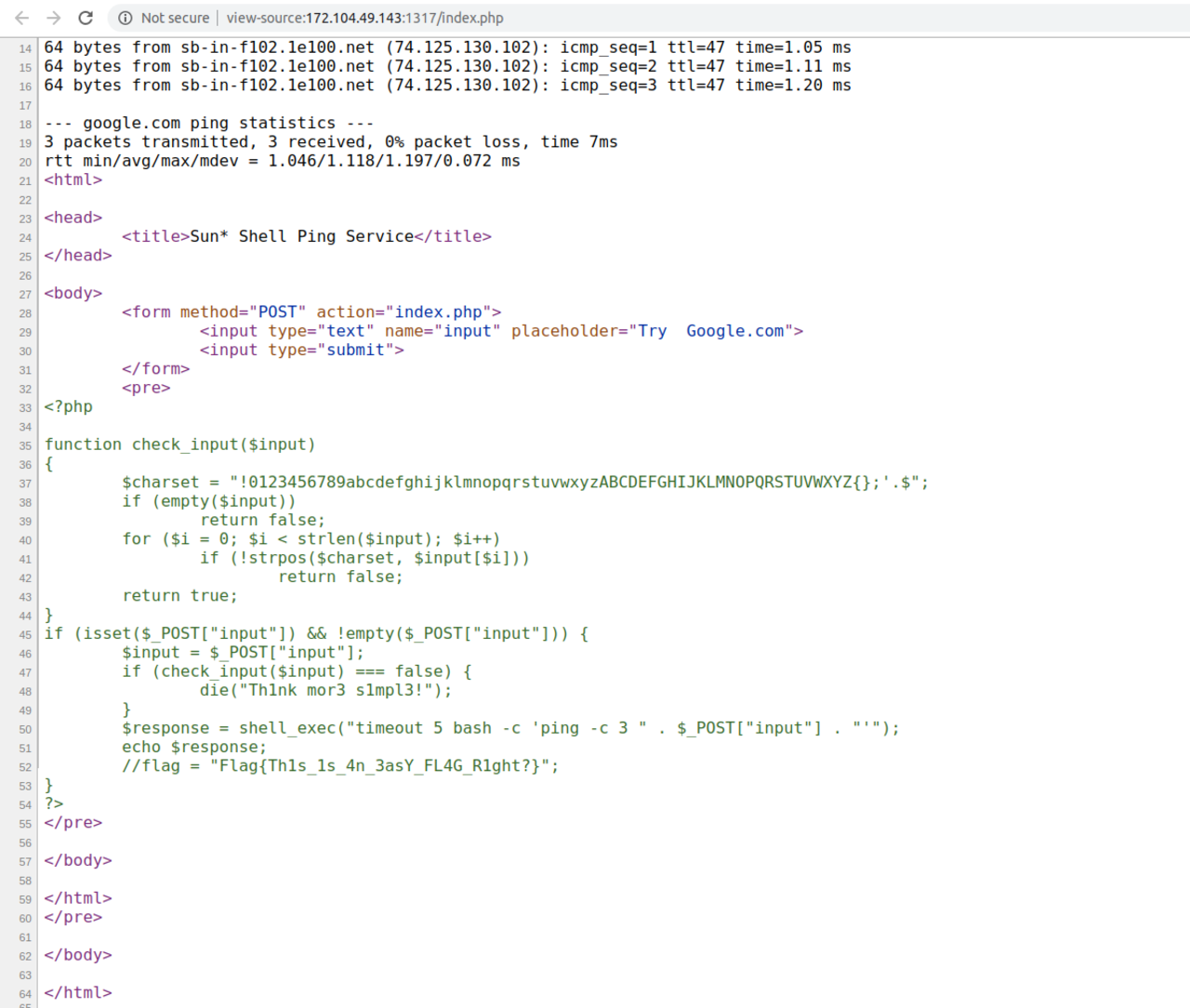
Mình tiếp tục đoán có thể người code đã có 1 vài filter để chặn lỗi này. Tiếp tục mò 1 vài filter thường dùng thì mình phát hiện ra bài này đã filter mất dấu cách. Tuy nhiên hoàn toàn có thể thay thế dấu cách bằng ký tự ${IFS}. Đồng thời cũng dùng ký tự ; để có thể chạy đồng thời nhiều lệnh command line cùng 1 lúc. Đầu tiên, mình sẽ dùng ls để xem cấu trúc thư mục:



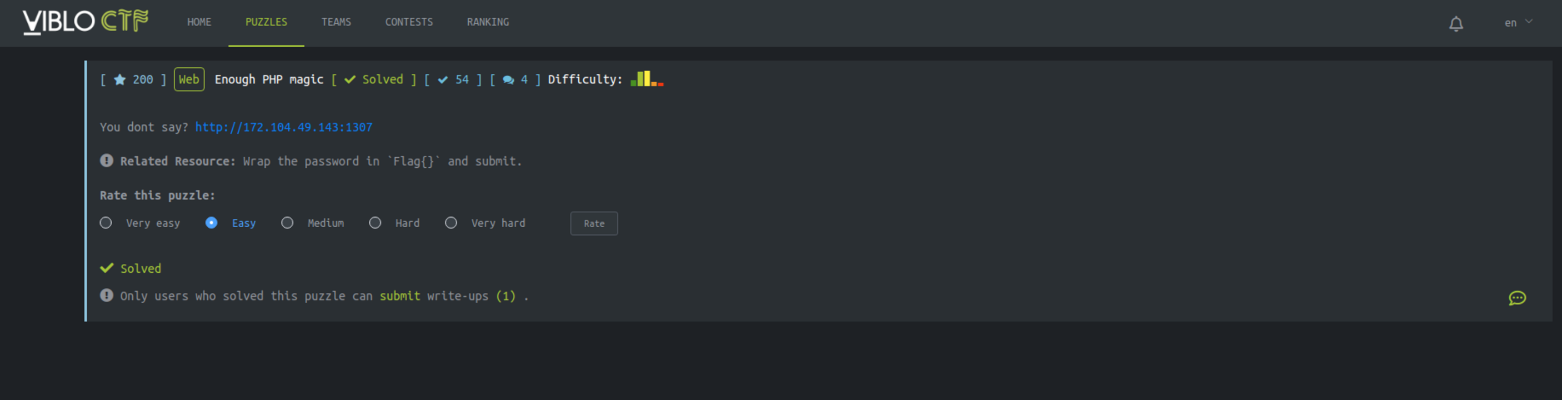
Đến đây mình thử dùng lệnh cat và ký tự ${IFS} để đọc file [**index.php**](https://spiderum.com/index.php) xem có gì có thể khai thác được không:



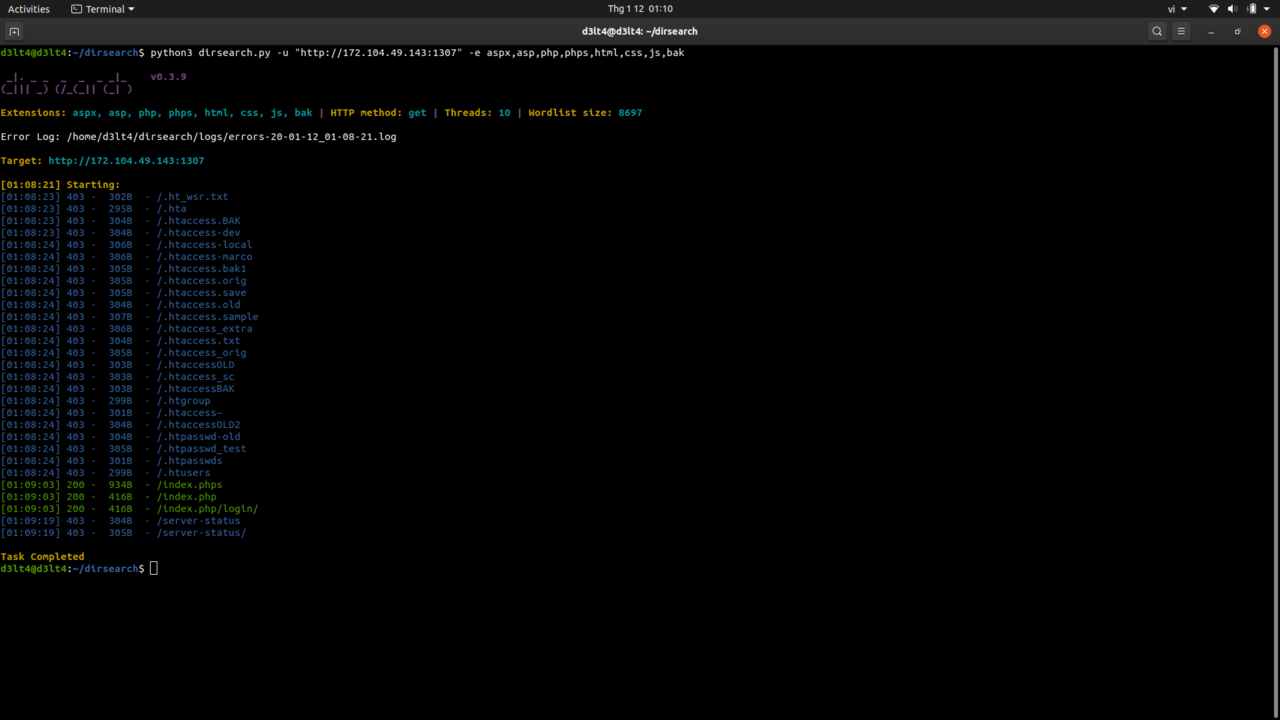
File [**index.php**](https://spiderum.com/index.php) sẽ luôn là file được server thực hiện đầu tiên để hiển thị web. Có vẻ khi cat ra thì server đã thực hiện luôn file đó :v. Mình tiếp tục thử xem source xem có gì đặc biệt không. Và kết quả là có luôn được flag :v



**2.4.Enough PHP magic (200)**



Web đề bài cho yêu cầu chúng ta đoán mật khẩu để xem flag. Mình đã thử 1 vài mật khẩu thông dụng đồng thời xem source tuy nhiên không thấy gì có thể khai thác được. Đến đây mình sẽ sử dụng dirsearch để xem có những file nào trong folder root của web:



Có 1 file index.phps, mình sẽ thử đi đến file này xem sao. Và đây là kết quả:



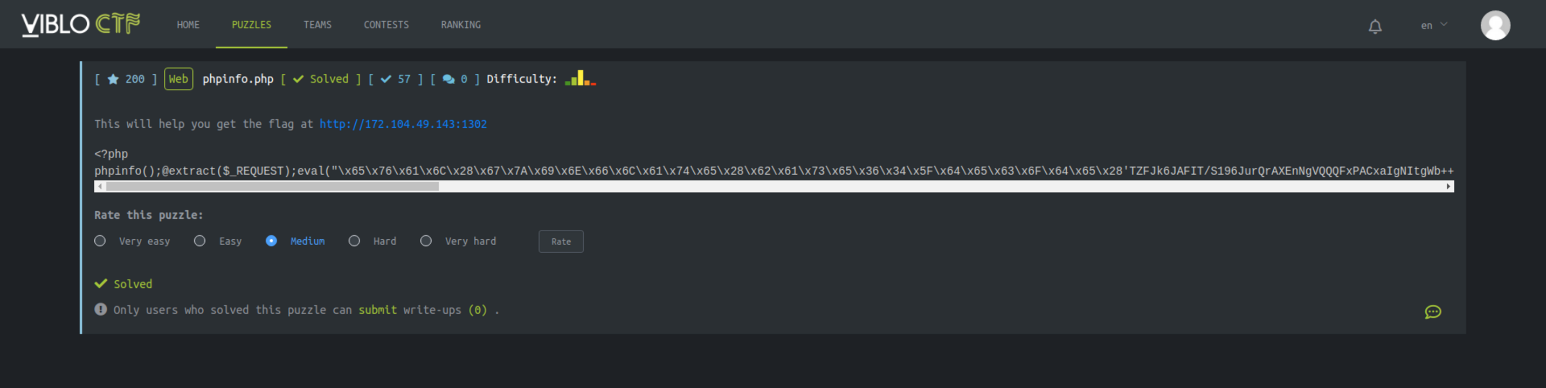
Tiếp tục kiểm tra source thì chúng ta có thể thấy source php của file [**index.php:**](https://spiderum.com/index.php%3A)



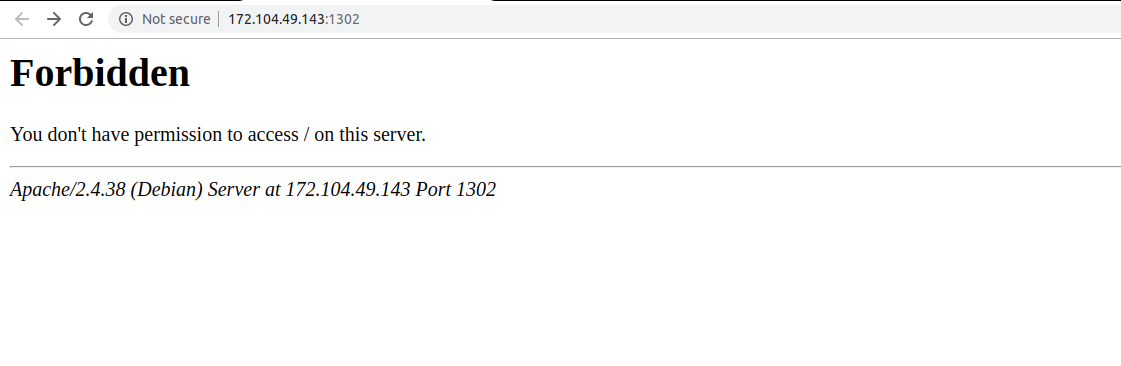
Đoạn code trên sẽ extract các biến có trong request GET vào các biến có trong  đoạn code. Vì hàm extract xuất hiện sau khi khai báo các biến nên chúng ta hoàn toàn có thể lợi dụng hàm này để ghi đè giá trị của các biến đã khai báo trước đó. Ví dụ như khi gửi 1 request GET dạng ?attempt=a&filename=b, khi hàm extract được gọi thì giá trị của biến $filename sẽ bị thay đổi thành b. Phần code dưới sẽ lấy nội dung của file có tên là giá trị cúa biến $filename và so sánh với biến $attempt lấy từ request GET. Nếu đúng thì sẽ in ra flag nằm trong file 'xxxxxxxx.txt'. Đến đây có rất nhiều cách để làm nhưng mình sẽ chọn cách đơn giản nhất, gửi 1 string rỗng và ghi đè biến filename thành 1 file không tồn tại => biến $combination và biến $attempt đều sẽ rỗng => profit. Payload của mình sẽ là ?attempt=&filename=DOESNOTEXIT.



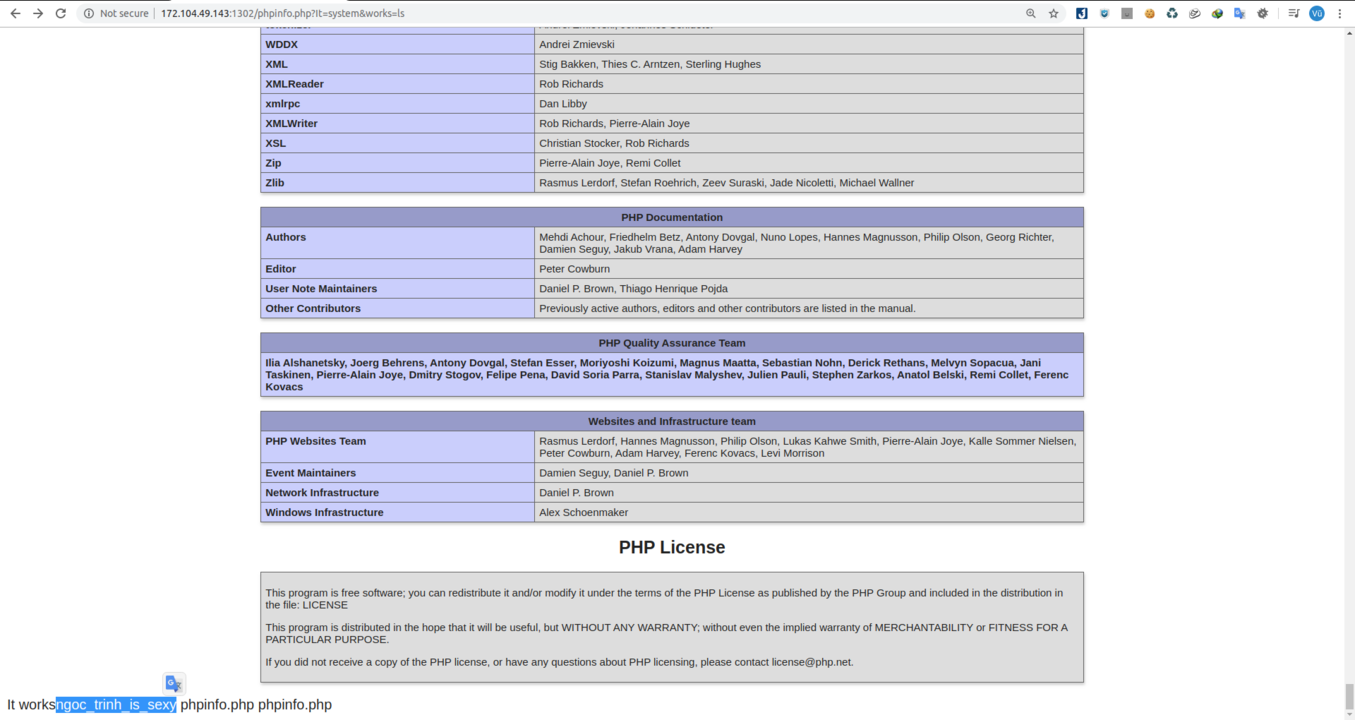
[**2.5.phpinfo.php**](https://spiderum.com/2.5.phpinfo.php)**(200)**



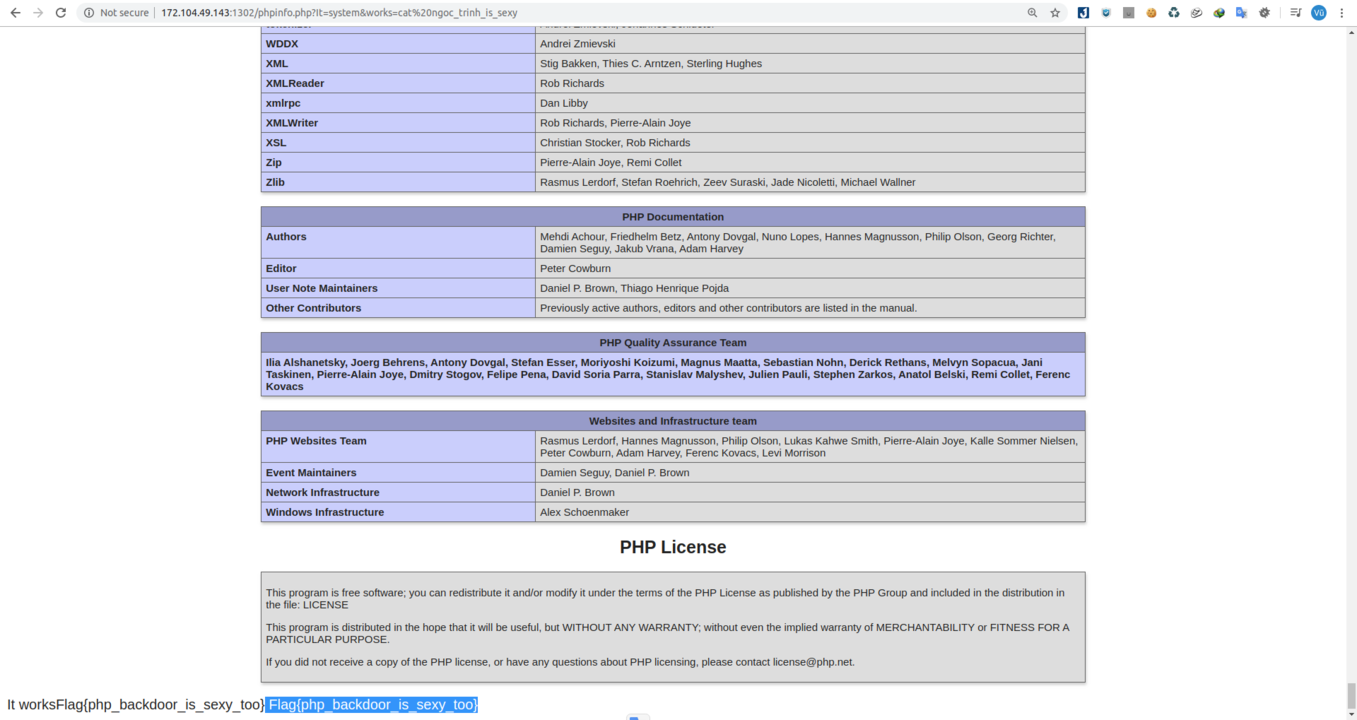
Đề bài cho 1 web và 1 đoạn code php trông khá lạ. Thử đi đến web trước thì thấy bị 403 Forbidden:



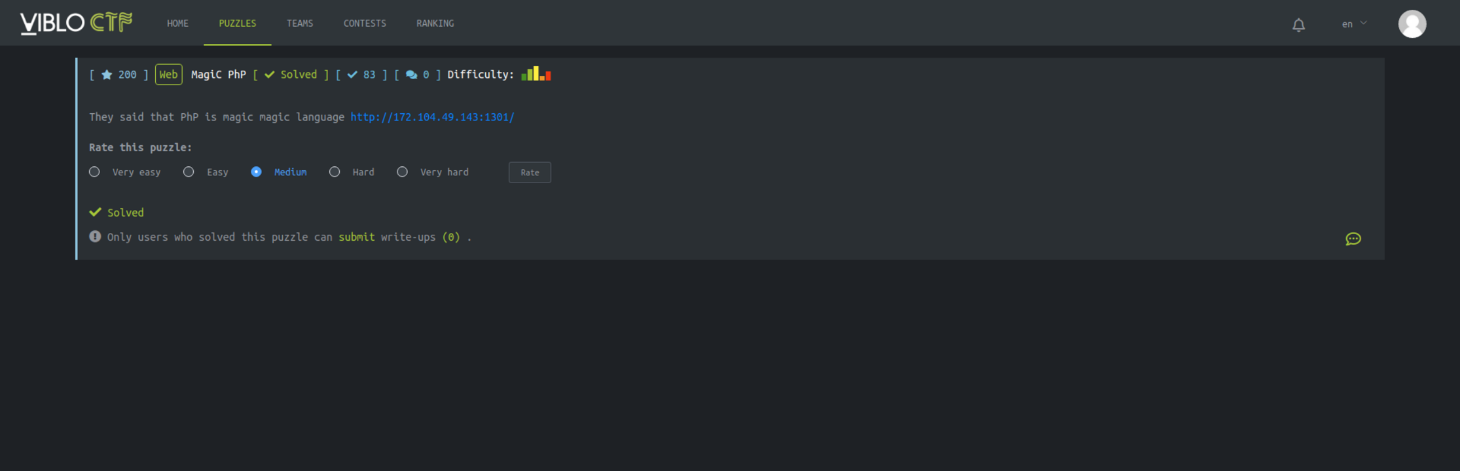
Mình đã thử search các file có trên server, xem source,... nhưng không có kết quả cho lắm. Quay lại với đoạn code php đề bài cho, loay hoay google 1 hồi mình nhận ra đây là đoạn 1 đoạn php backdoor có tên là PHP Extract Backdoor. Backdoor này cho phép thực thi các lệnh command line thông qua 2 biến $It và $works nằm trong request GET hoặc POST.  Về chi tiết thì lỗi ở hàm extract giống như mình đã nói ở trên, hàm extract sẽ extract toàn bộ nội dung nằm trong request và gán những giá trị đó vào các biến. Sau đó, khi echo($It($works)) được thực hiện thì những gì hacker gửi đến thông qua 2 biến $It và $works sẽ được thực hiện. Đến đây mình sẽ thử payload ?It=system&works=ls để xem cấu trúc thư mục trước:



Đến đây mình sẽ thử cat file ngoc\_trinh\_is\_sexy ra xem bằng payload ?It=system&works=cat ngoc\_trinh\_is\_sexy:



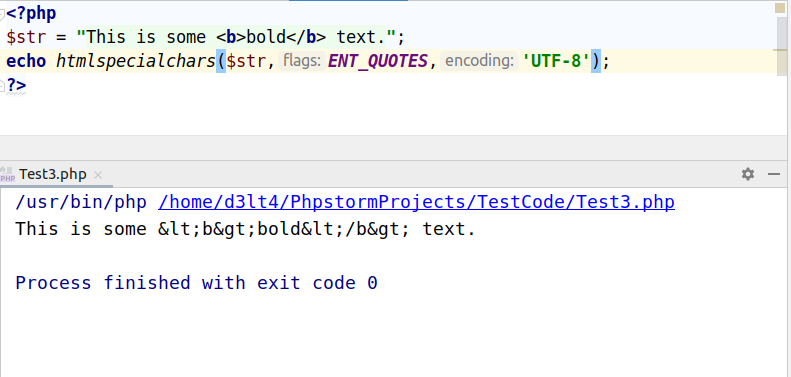
**2.6.MagiC PhP (200)**



Bài này cũng gần giống với bài Tragic pHp ở trên, cũng xem source và có chỉ dẫn đến file index.phps để xem source php của web:

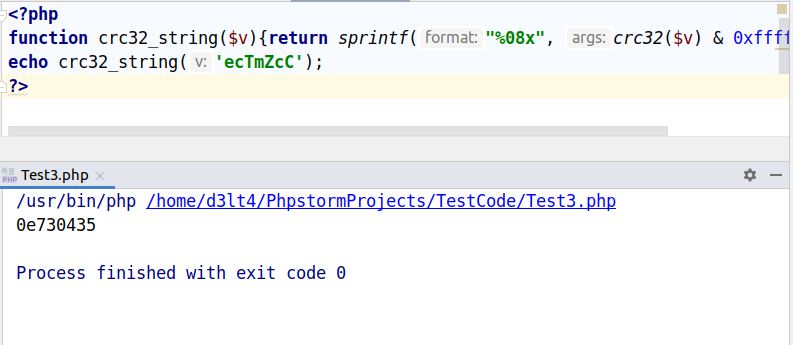


Đoạn code này sẽ lấy dữ liệu từ giá trị value trong request POST và gán vào biến $value. Hàm h($s) sẽ chuyển các ký tự đặc biệt trong xâu thành các HTML entities. Ví dụ cho hàm này:



Hàm crc32\_string($v) sẽ trả về 1 chuỗi băm crc độ dài 32 bit cho string $v đươc cung cấp. Sau đó, code sẽ kiểm tra giá trị của chuỗi $value phải khác rỗng và khác "ecTmZcC", đồng thời giá trị chuỗi băm của $value và giá trị chuỗi băm của chuỗi "ecTmZcC" phải giống nhau. Lại nghe rất vô lý :v

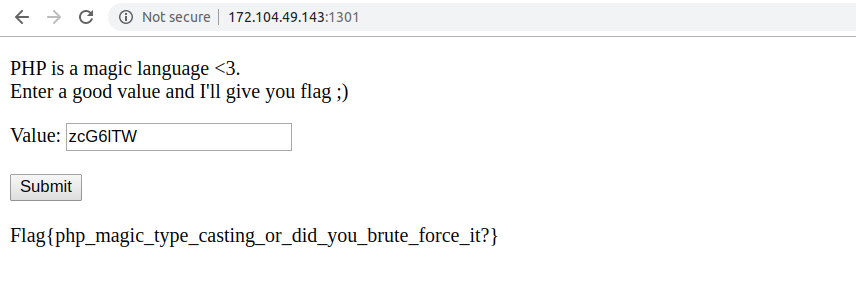
Tuy nhiên, để ý thì khi so sánh 2 chuỗi băm, người code sử dụng toán tử ==. Trong php, toán tử == có thể gây ra lỗi Type Juggling (Về chi tiết thì các bạn có thể đọc ở [**đây**](https://www.owasp.org/images/6/6b/PHPMagicTricks-TypeJuggling.pdf)). Mình sẽ thử kiểm tra chuỗi băm của "ecTmZcC" xem sao:



Có thể thấy chuỗi băm của "ecTmZcC" có dạng '0e{các số từ 0-9}' => có thể sử dụng Type Juggling để bypass bằng cách gửi 1 giá trị value sao cho khi băm ra cũng có dạng '0e{các số từ 0-9}'. Mình sẽ sử dụng 1 đoạn code php để brute tìm ra string phù hợp:

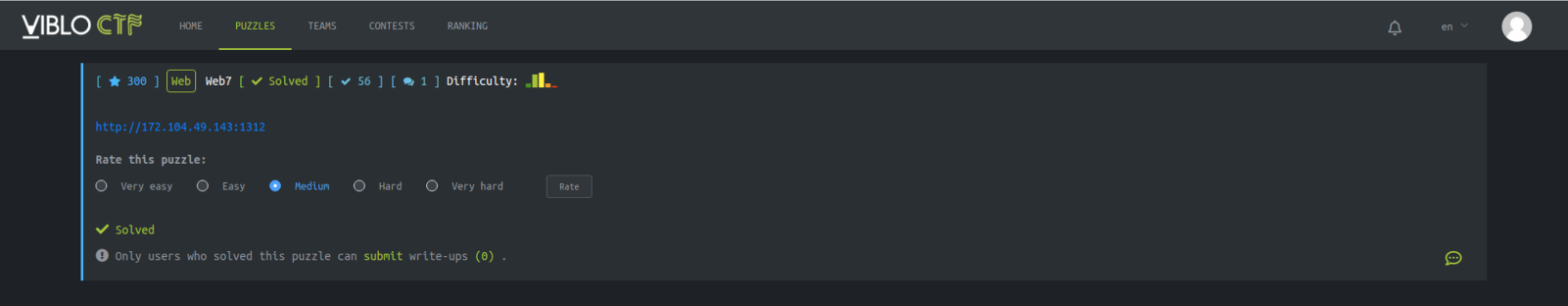


Xác định được giá trị cần gửi rồi thì việc còn lại là lầy flag thôi:

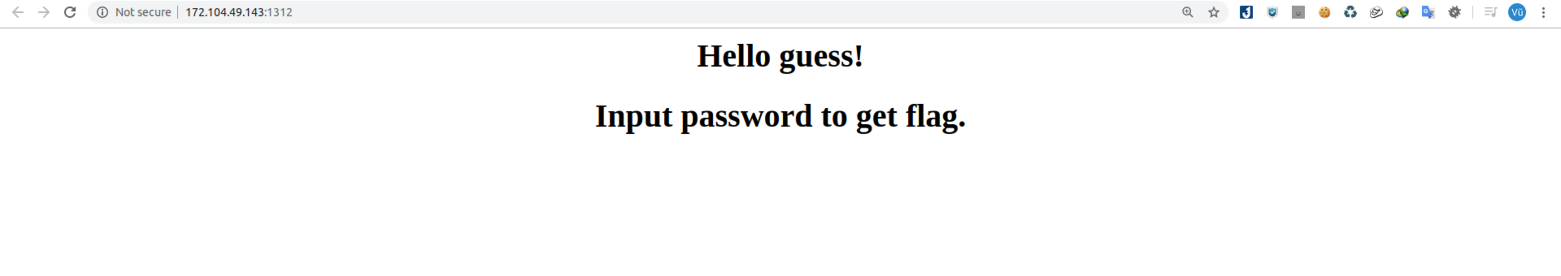


**3. Rank C**

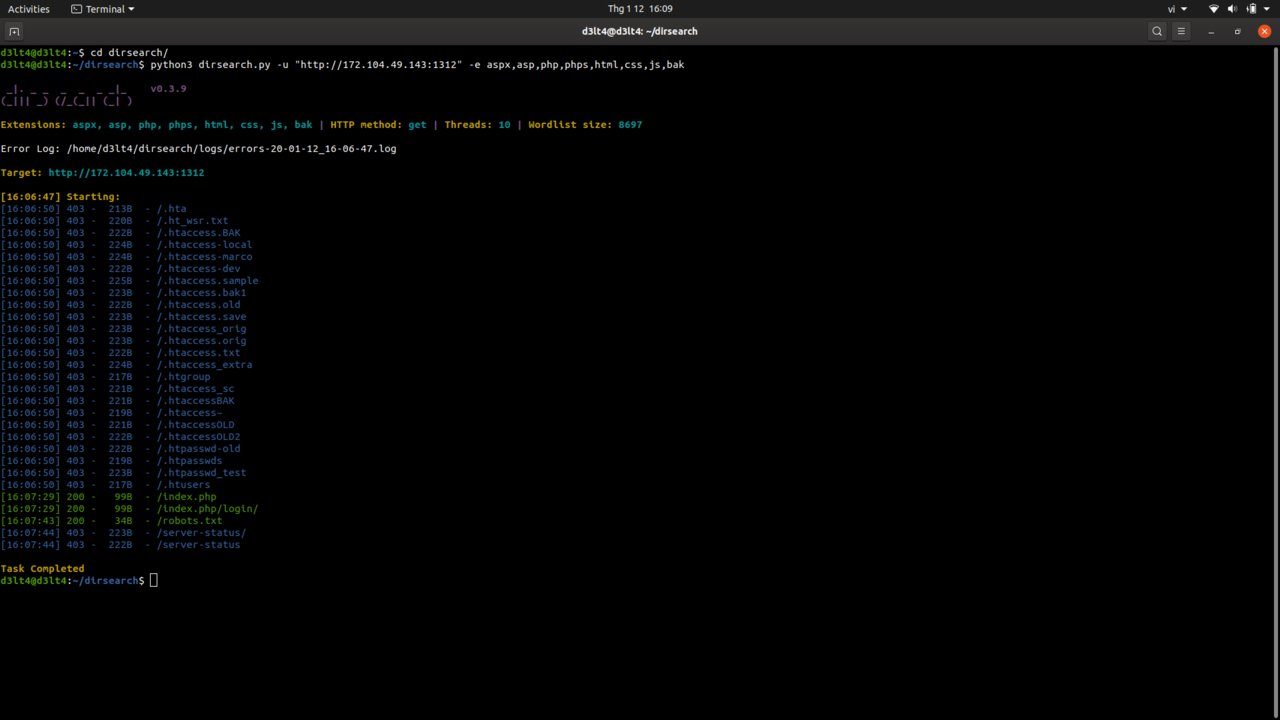
**3.1. Web7 (300)**



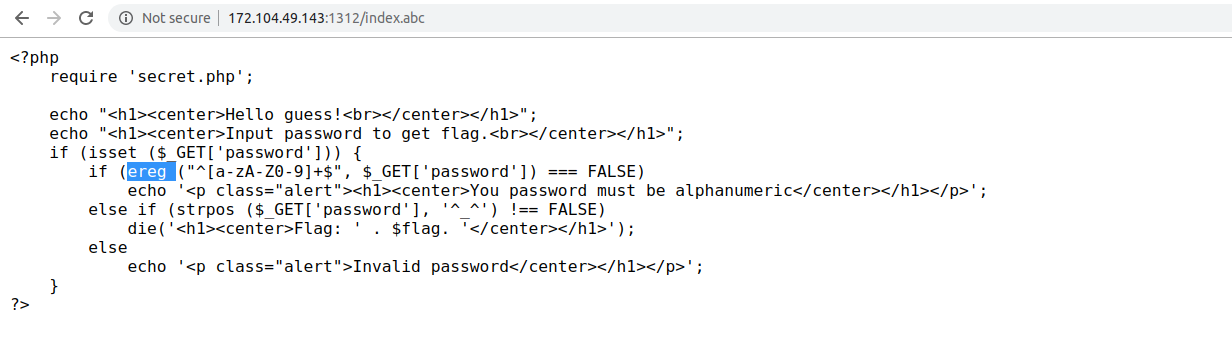
Đi đến web đề bài cho:



Web yêu cầu nhập password để lấy flag nhưng lại không cho form input :v. Mình sẽ thử quét xem có những file nào trong folder web bằng dirsearch:

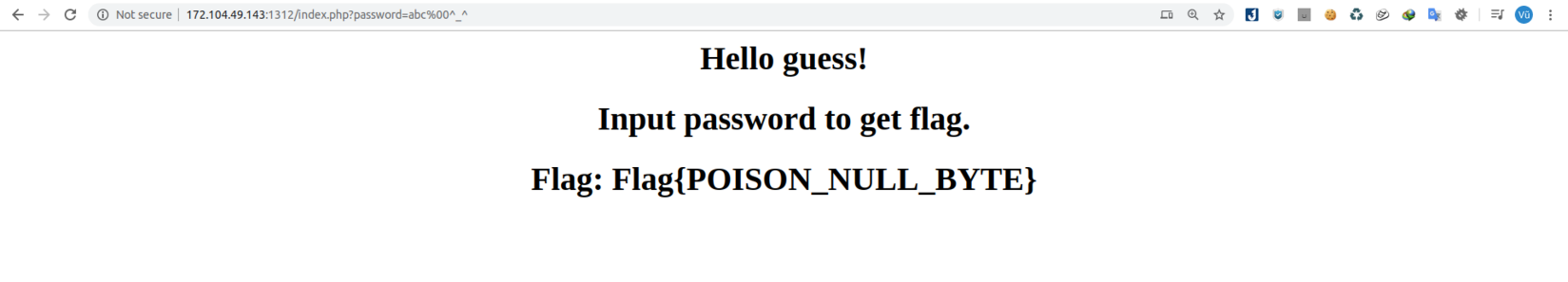


Mình thử đi đến file [**robots.txt**](https://spiderum.com/robots.txt) thì thấy file [**index.abc**](https://spiderum.com/index.abc) đã bị ẩn đi. Đi đến file [**index.abc**](https://spiderum.com/index.abc) sẽ thấy được source php của file:

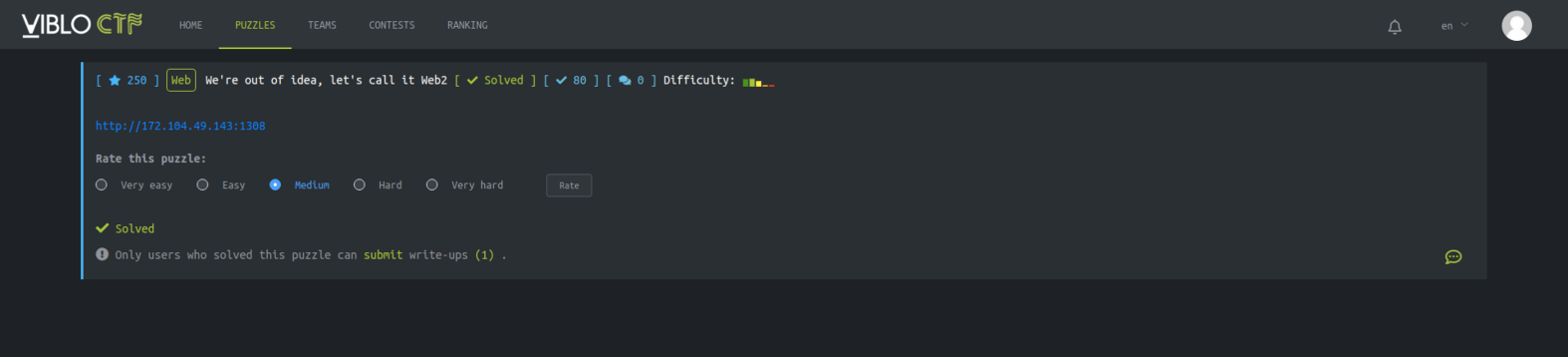


Đoạn code này sẽ lấy dữ liệu tư giá trị password nằm trong request GET, sau đó sẽ tìm xem trong string gửi lên có ký tự non alphanumeric bằng hàm ereg. Nếu không sẽ tìm trong string đó có đoạn '^\_^' nào không, nếu có thì sẽ echo ra flag. Tuy nhiên nhận thấy '^\_^' bao gồm toàn các ký tự non alphanumeric mà các ký tự non alphanumeric đã bị filter bởi hàm ereg :v

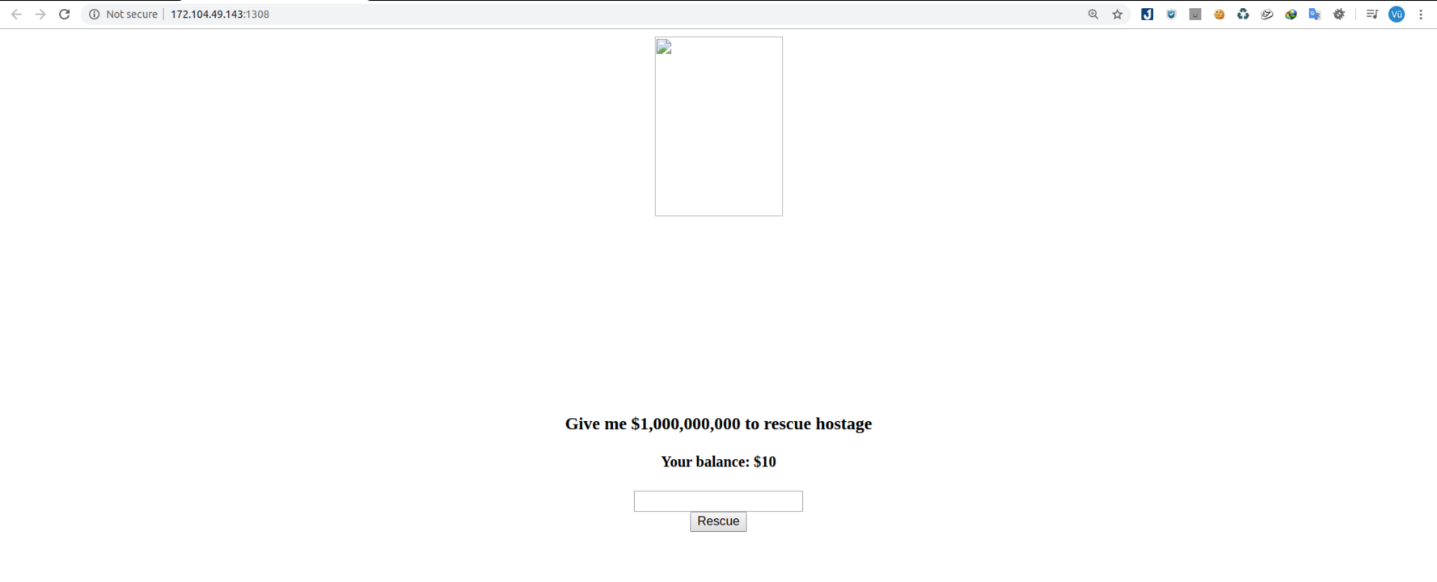
Nếu để ý thì chức năng của hàm ereg và strpos là gần giống nhau, câu hỏi đặt ra là tại sao người code lại dùng 2 hàm này. Loay hoay google 1 hồi mình tìm được 1 cách bypass hàm ereg. Bằng cách chèn null byte vào string gửi lên, mình hoàn toàn có thể bypass được hàm ereg => payload của mình sẽ là: /?password=abc%00^\_^:



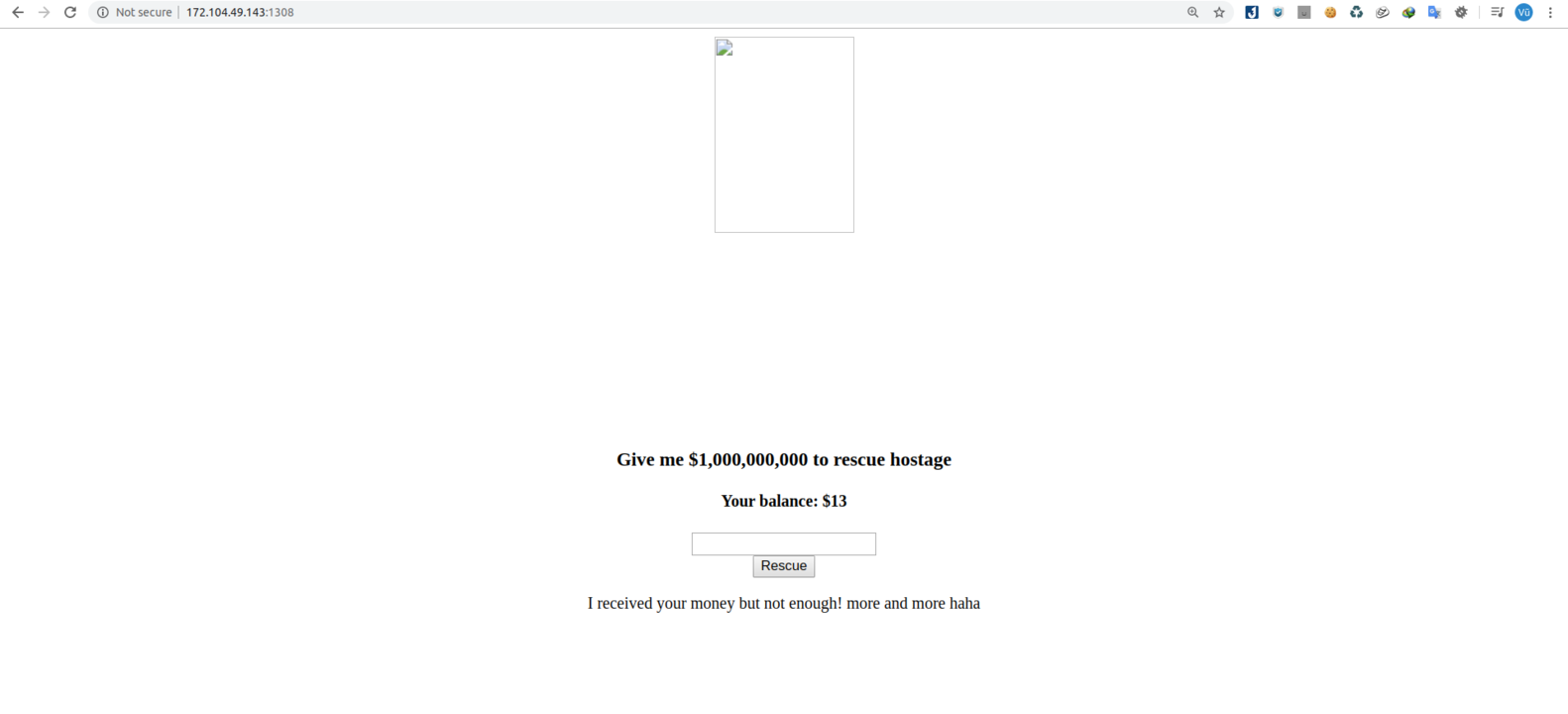
**3.2.We're out of idea, let's call it Web2 (250)**



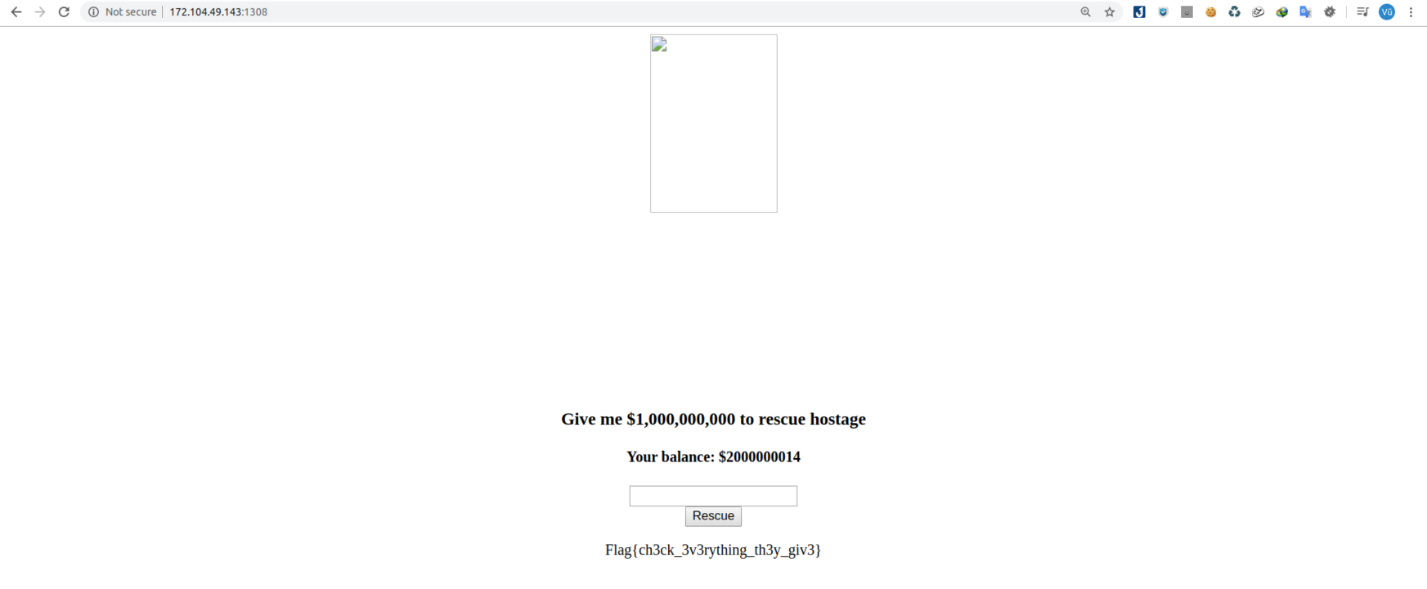
Đi đến web đề bài cho thì thấy yêu cầu đưa $1,000,000,000 để xem flag nhưng chỉ cho chúng ta $10 :v



Mỗi khi nhập 1 số nhỏ hơn số dư thì nhận thấy tiền trong số dư bị trừ đi. Mình nghĩ thử nhập số âm xem sao và kết quả là tiền trong số dư tăng lên :v

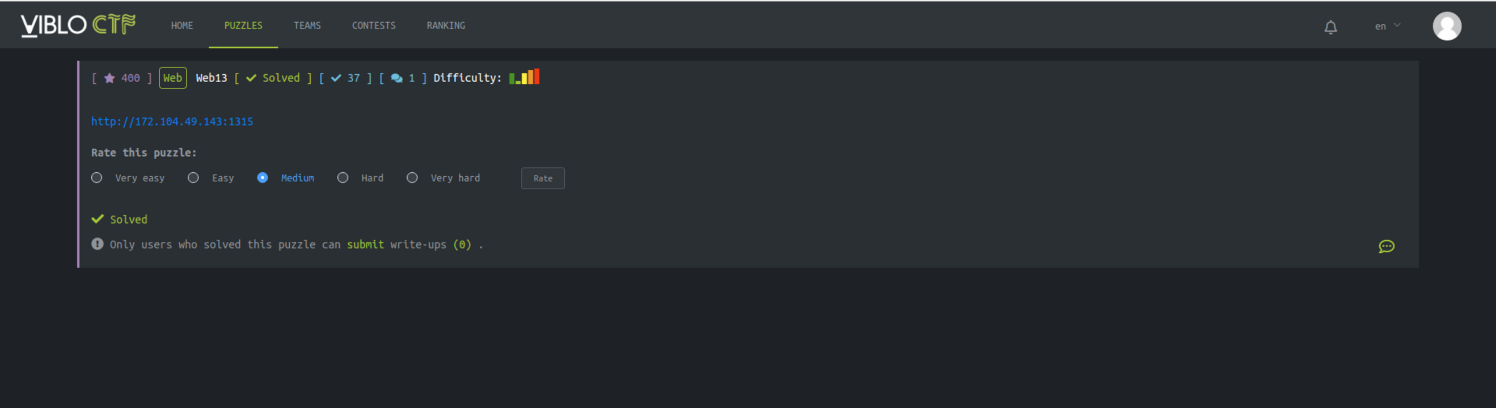


Đến đây thì chỉ cần nhập -1000000000 để đủ tiền xem flag thôi :v

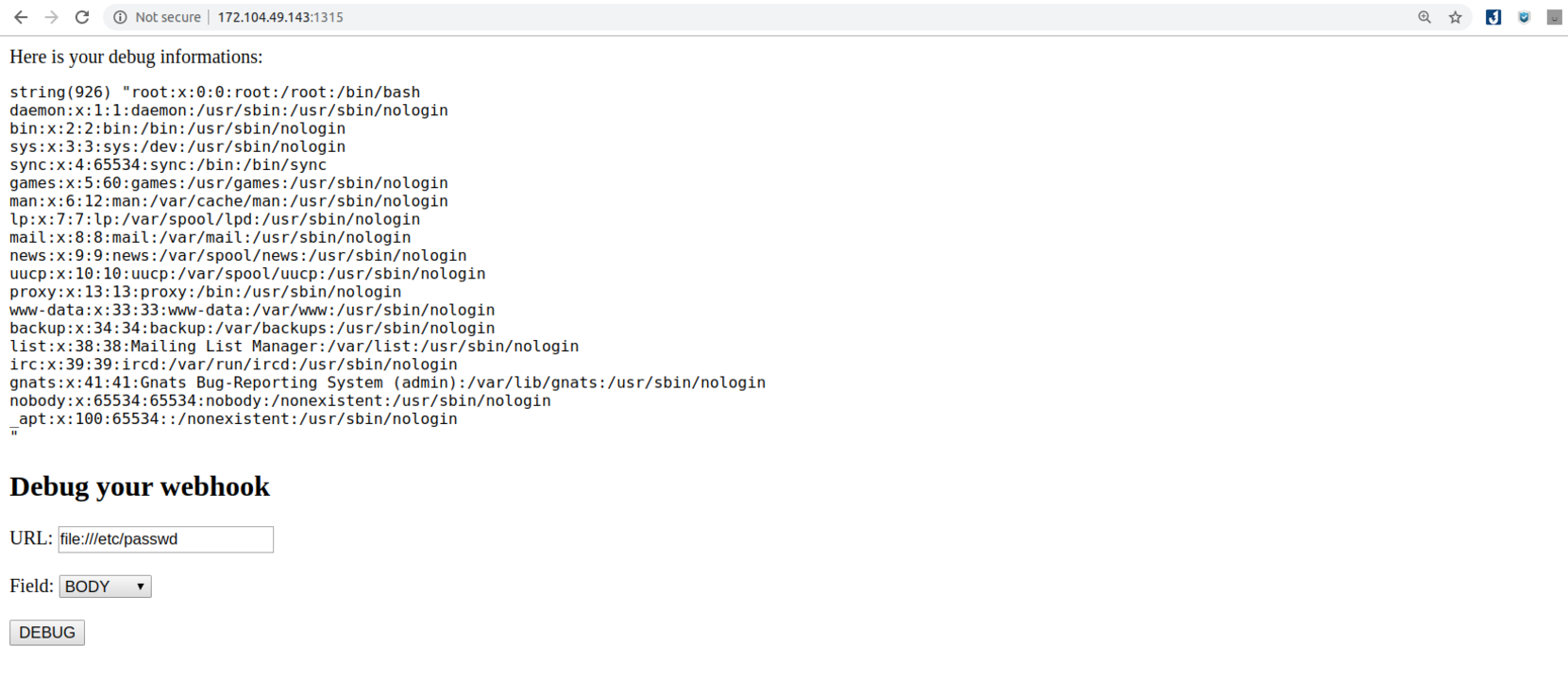


**4. Rank B**

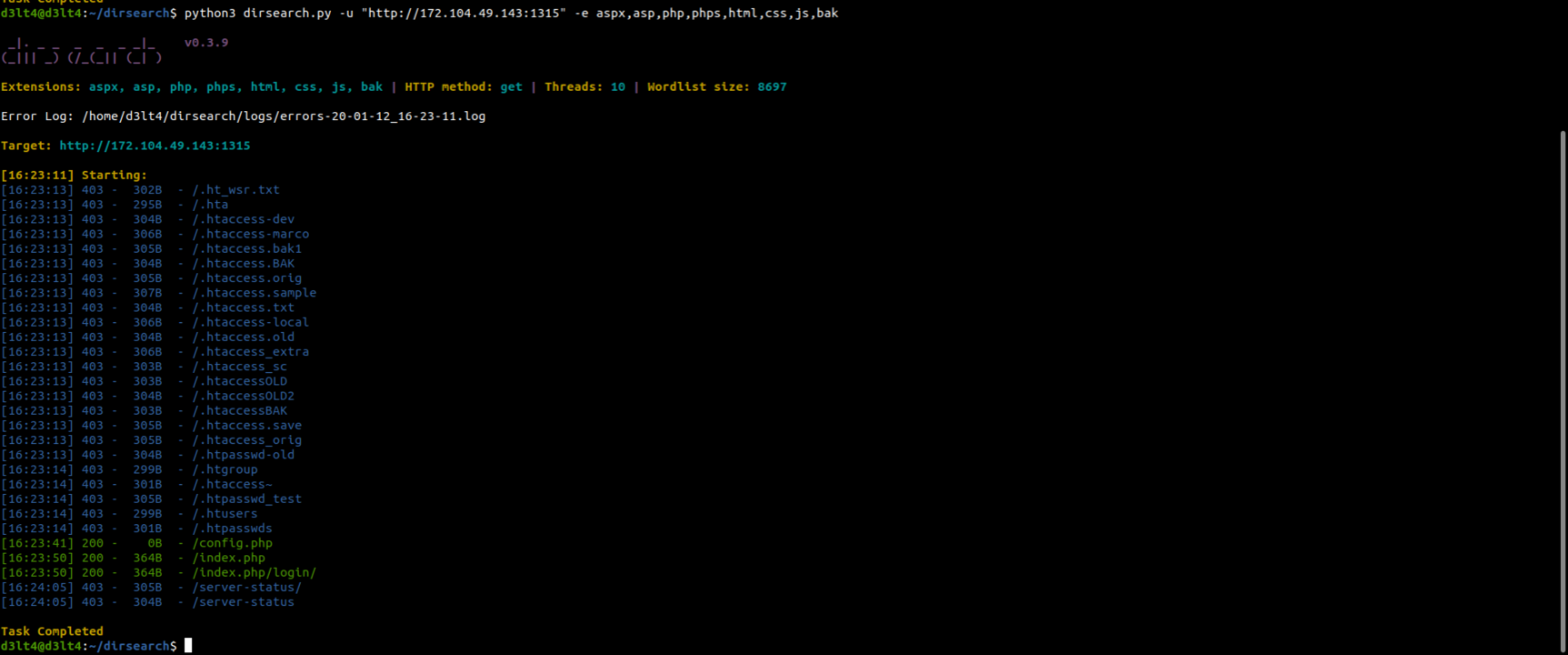
**4.1.Web13 (400)**

****

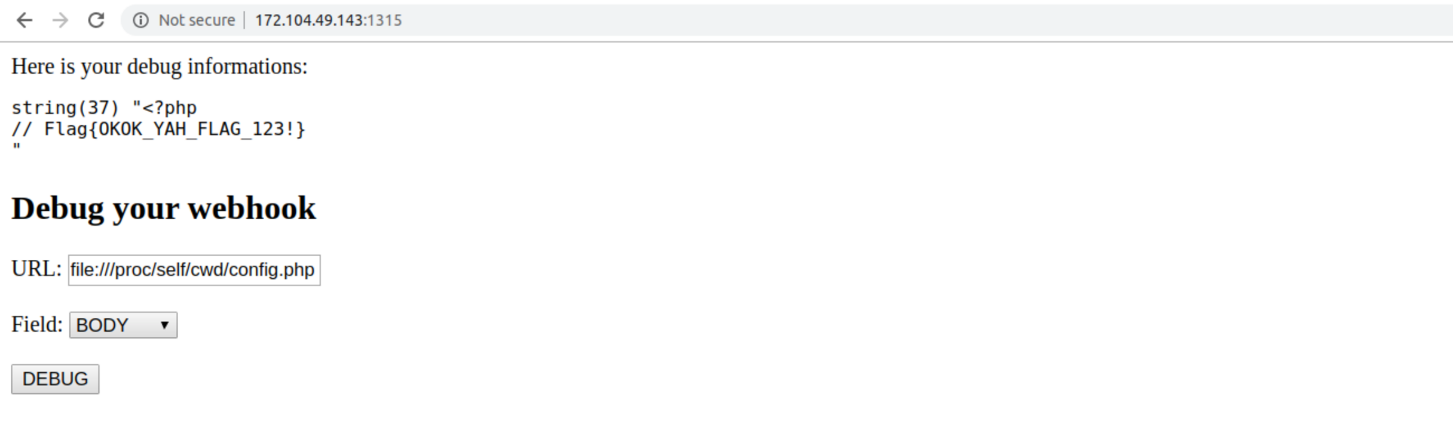
Đề bài cho 1 web Debug Webhook. Để nói về khái niệm Debug Webhook là gì thì sẽ rất dài nên mình để link ở [**đây**](https://viblo.asia/p/tim-hieu-ve-khai-niem-webhook-1Je5EvdmKnL). Test sơ qua chức năng thì khi cung cấp 1 URL dạng http://.... và chọn field là Body hoặc Header thì web sẽ trả về kết quả. Đến đây thì mình tịt luôn ý tưởng nên loay hoay đi hỏi. Một hồi mới biết web này có lỗi SSRF. Loay hoay thêm 1 lúc nữ thì mình phát hiện lỗi nằm ở field Header:



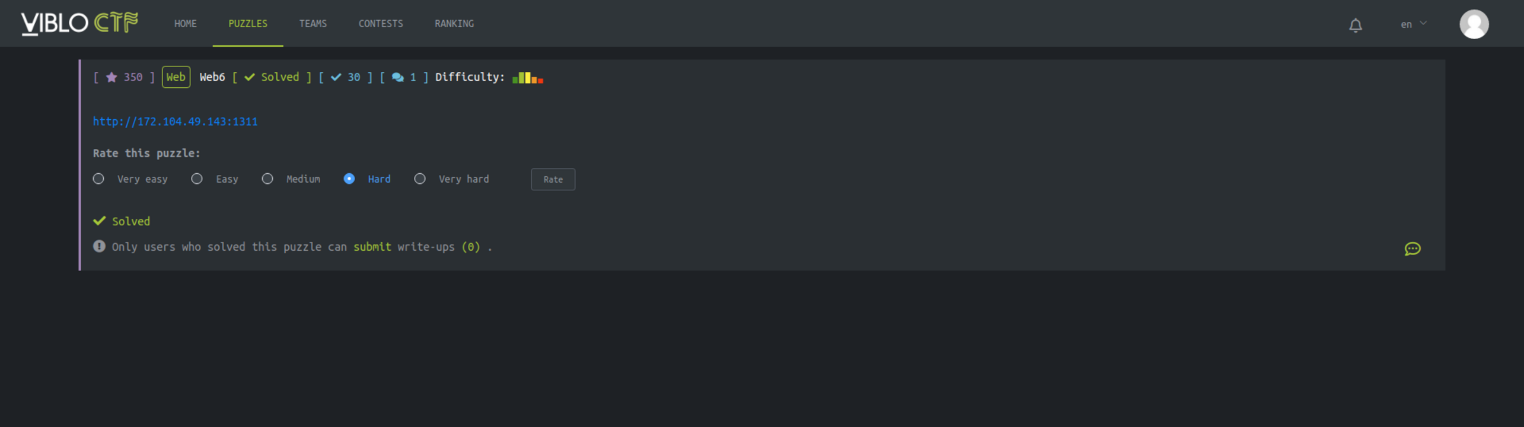
Đến đây, mình tiếp tục dùng dirsearch để search các file có trong folder web:



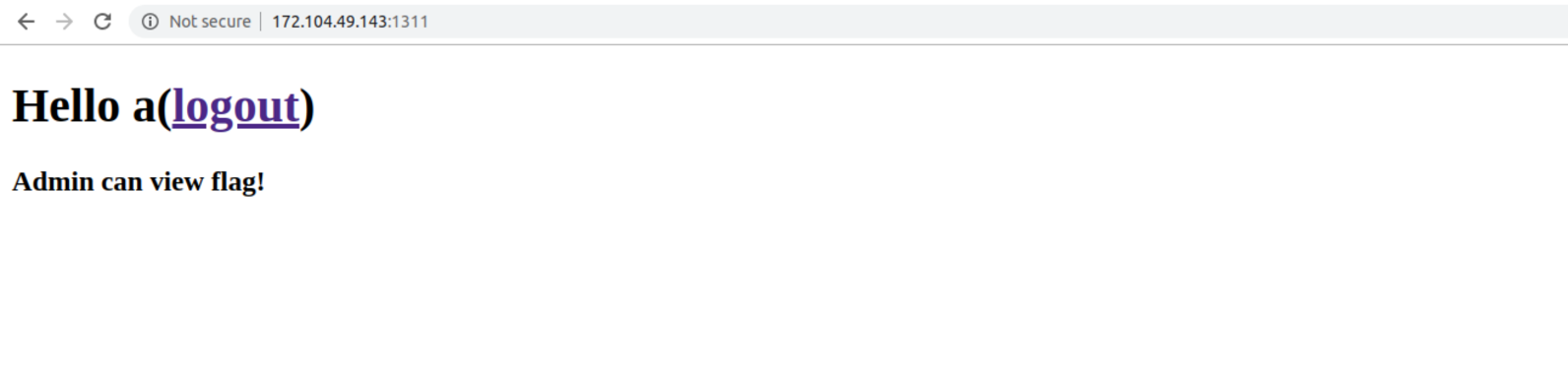
Mình sẽ thử tìm cách đọc file config.php. Vì không đoán được folder root của web nên mình sẽ sử dụng payload file:///proc/self/cwd/config.php. Trong đó /proc/self/cwd là 1 liên kết tượng trưng đến folder hiện tại web đang làm việc hay nói cách khác chính là folder root của web:



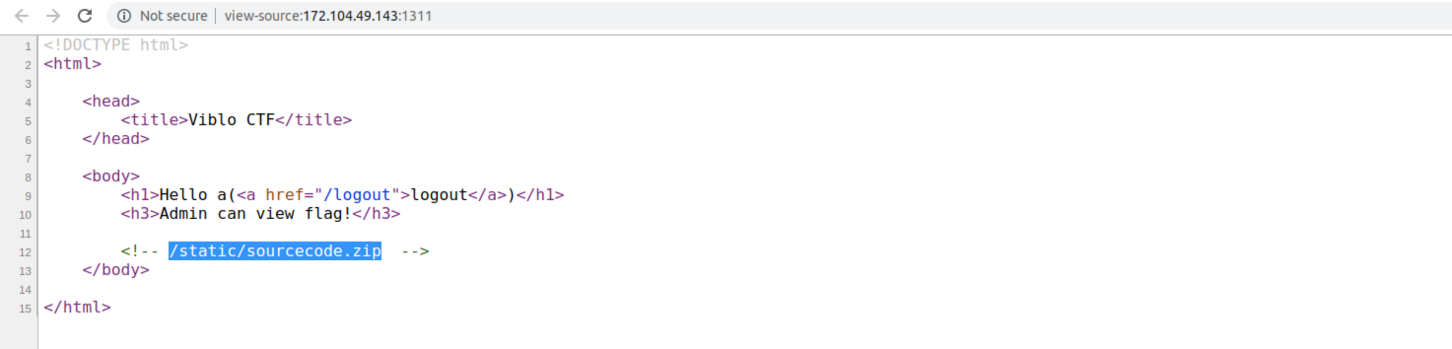
**4.2.Web6 (350)**



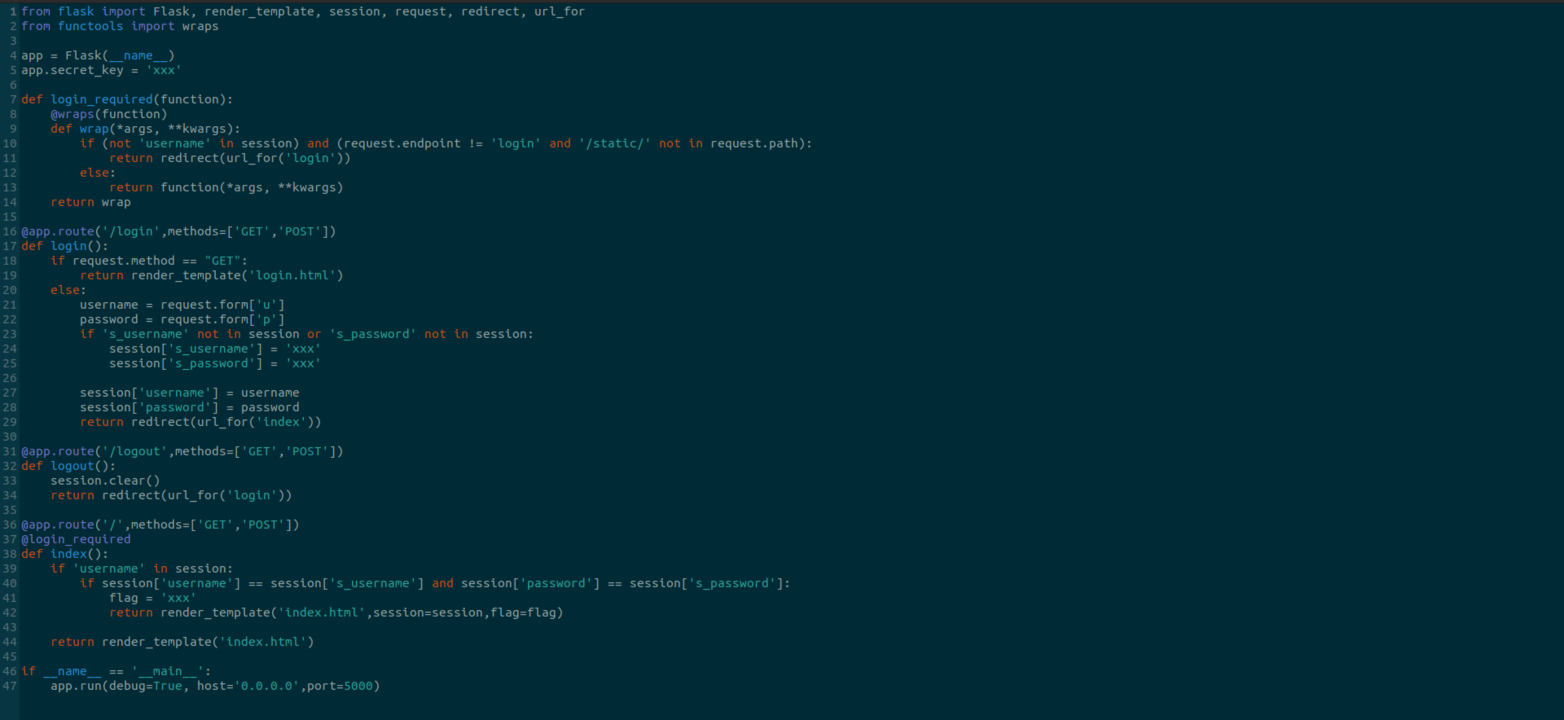
Đề bài cho 1 web login thông thường. Khi test qua thì mình nhận thấy dù nhập username hay password là gì thì đều có thể login và chỉ có admin mới xem được flag:



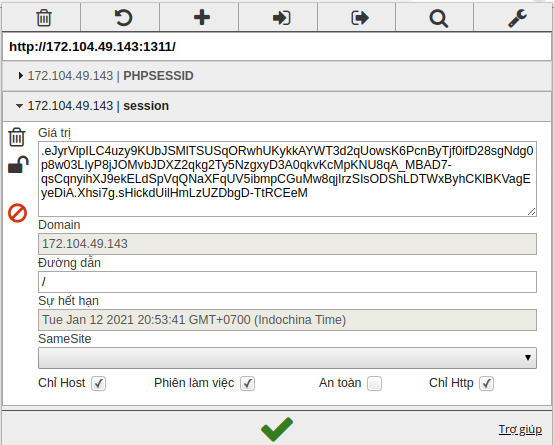
Xem source thì cũng có chỉ dẫn đến 1 file zip chứa source của web nằm tại /static/sourcecode.zip, chỉ cần đi đến và tải file này về thôi:



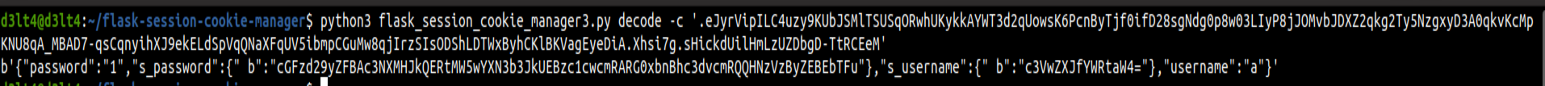
Xem qua các file có trong file zip thì đây là 1 Web được viết bằng Flask trong Python:



Để ý chức năng login của Web thì sẽ lấy username và password từ request đồng thời tạo các trường s\_username, s\_password, username và password nằm trong session. Trong đó trường username và password sẽ được lấy từ request. Khi login thì Web sẽ kiểm tra xem giá trị của username có trùng với giá trị của s\_username và giá trị của password có trùng với giá trị của s\_password hay không. Nếu có thì chúng ta sẽ lấy được flag => chúng ta cần phân tích session để lấy được giá trị của s\_username và s\_password. Session của web nằm trong cookies:



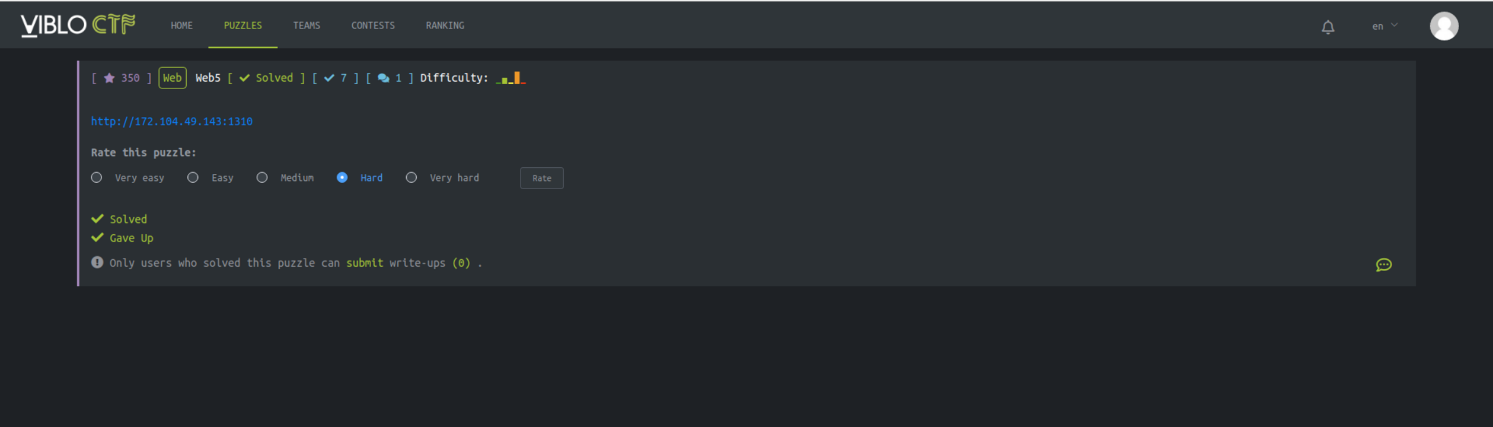
Đến đây thì mình tìm 1 tool decode session flask cookie trên github tại [**đây**](https://github.com/noraj/flask-session-cookie-manager). Clone về và chạy thôi:



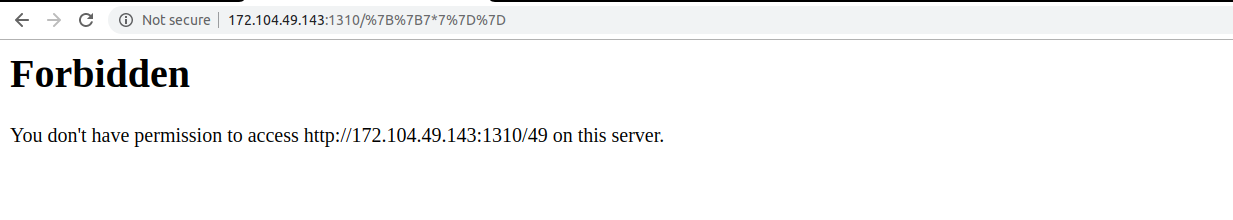
Có được 2 giá trị s\_password và s\_username rồi tuy nhiên nhận ra 2 giá trị này đều đã bị encode base64 nên mình decode ra và có s\_password là paswordP@ssW0rd@Dm1npaswordP@ssW0rd@Dm1npaswordP@ssW0rd@Dm1n và s\_username là super\_admin. Login bằng 2 giá trị này và có flag:



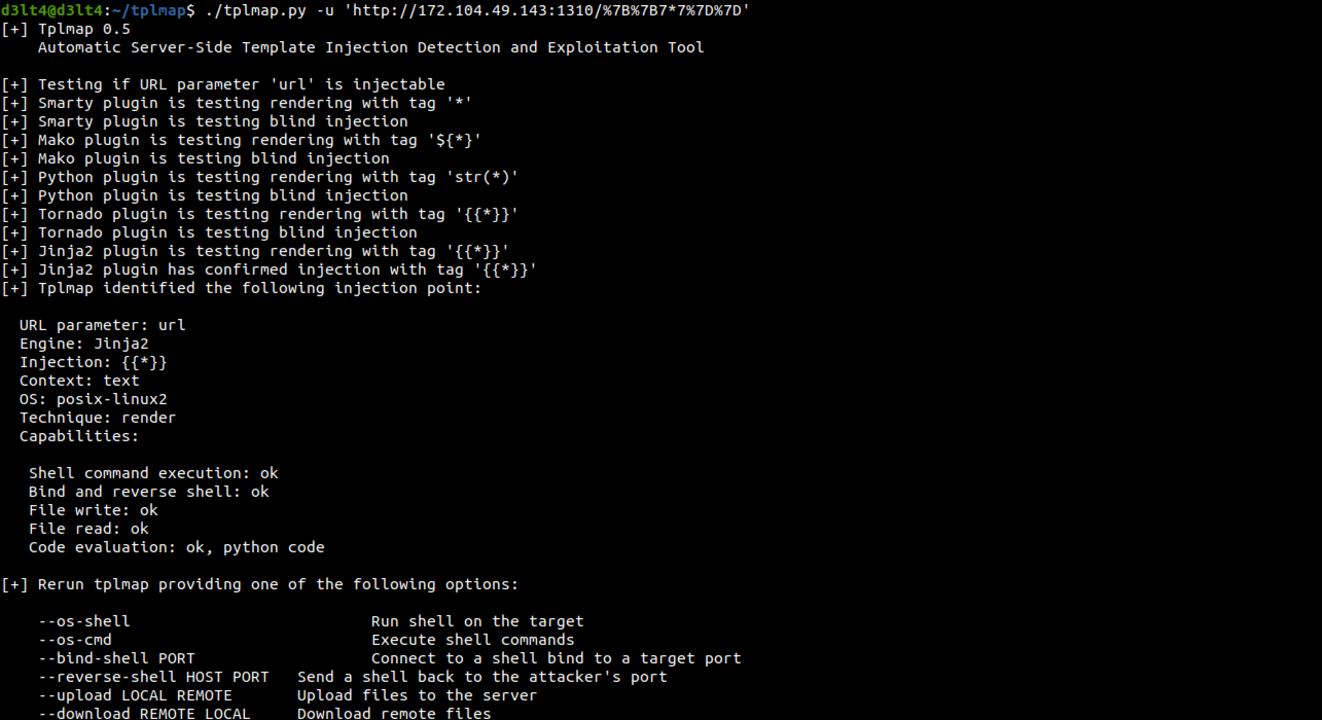
**4.3.Web5 (350)**



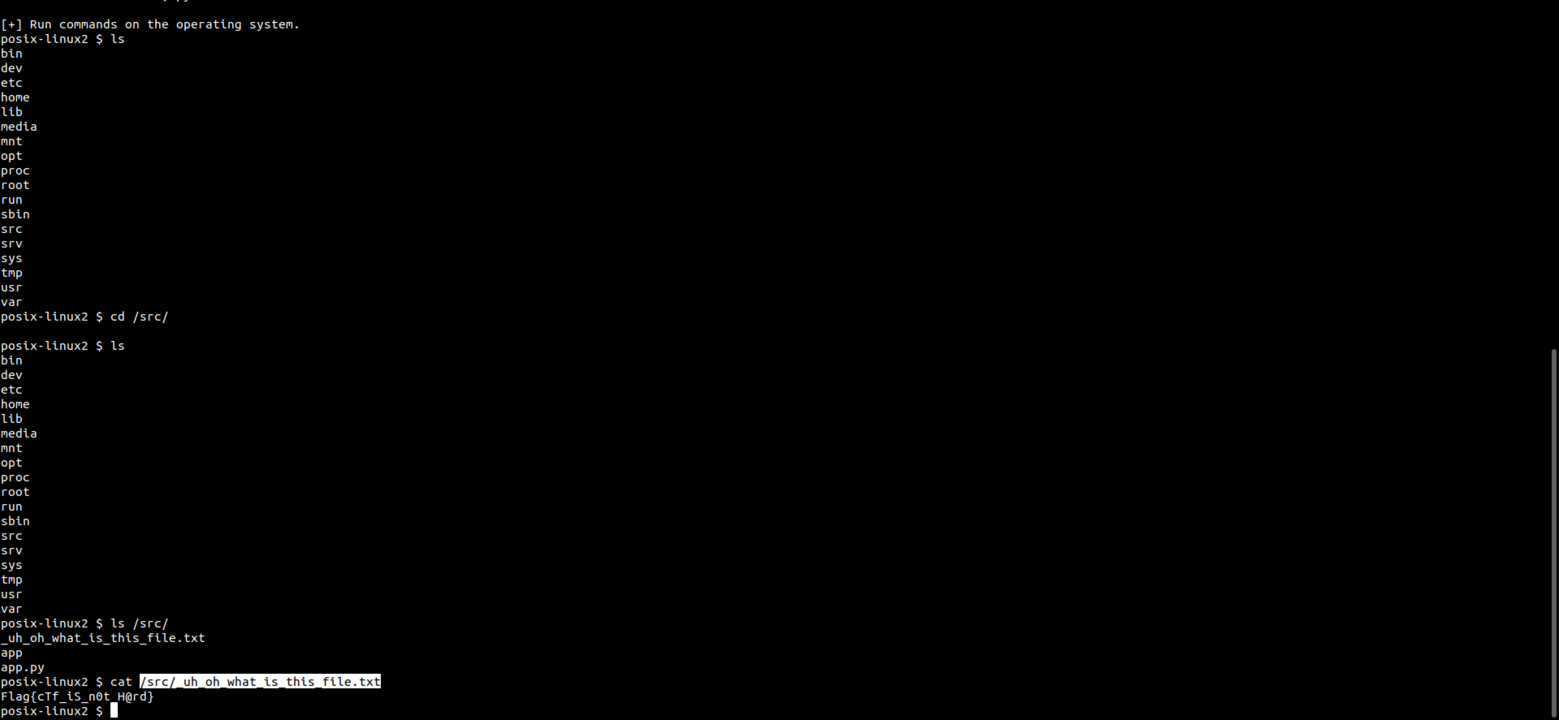
Đề cho 1 cái blog. Mình đã thử xem source và các tính năng thì thấy các trang /about và /contact đều bị 403 và cũng không có cái gì có thể khai thác được cái gì. Loay hoay thêm 1 lúc nữa mình cũng nhận ra là cái blog này được viết bằng python :v. Sau đó thì hoàn toàn tịt :v. Cuối cùng thì mình chấp nhận give up để xem hint :v. Và surprise mother fucker :v, bài này dính SSTI :v:



Cơ mà cái lỗi này mình cũng không nắm rõ nên phương án của mình là dùng tool tplmap, 1 tool cho phép scan lỗi SSTI trên web:



Đến đây mình sẽ dùng option --os-shell để thông qua SSTI tạo 1 shell trên server. Chạy lệnh ls ra và mò 1 hồi thì mình nhận ra flag nằm trong file \_uh\_oh\_what\_is\_this\_file.txt ở folder src. Chỉ cần cat ra là có flag thôi:



Bài này vì mình chưa nắm rõ SSTI lắm nên làm theo cách khá là mất não :v Mình không khuyến khích các bạn làm theo cách này :v. Các bạn nên tìm hiểu kỹ thêm về lỗi này vì nó cũng rất hay :v