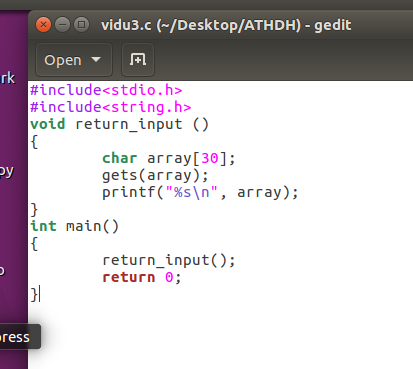
oke, nhận sự uỷ thác của anh Phú thì mình viết bài này hỗ trợ mọi người làm bài thực hành môn ATHDH buổi 1.

Mục tiêu là làm sao để gọi lại hàm return\_input để in ra chuỗi mình nhập 2 lần:

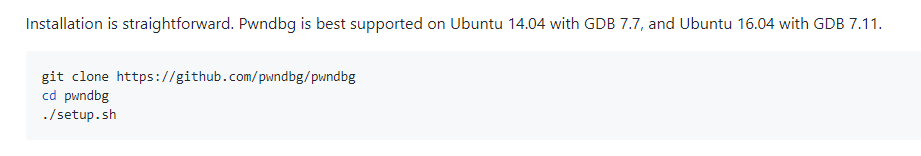


Thì đầu tiên là để làm được bài này thì setting máy 1 tí tẹo.   
1 . mình recommend mọi người download bản ubuntu 16.04 về xài.

2. download cài đặt multilc-gcc theo hướng dẫn ở đây: <https://howtoinstall.co/en/ubuntu/xenial/gcc-multilib>

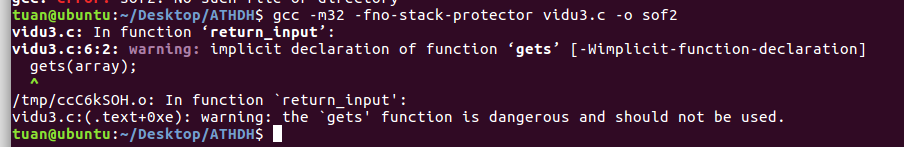
3.mọi người cài đặt cho mình pwndbg:

Theo hướng dẫn dưới đây: <https://github.com/pwndbg/pwndbg>



Oke, sau khi cài đặt xong. Các bạn gõ lại code, lưu với tên gì cũng được, mình lưu là vidu3 , và compile thành file sof2 với dòng lệnh như sau:

(gcc -m32 -fno-stack-protector -no-pie vidu3.c -o sof2) ( các bạn để ý dấu – và dấu cách sao cho đúng nha, sai là nó sai đó) , nếu compile thành công sẽ có 1 cái thông báo như này:



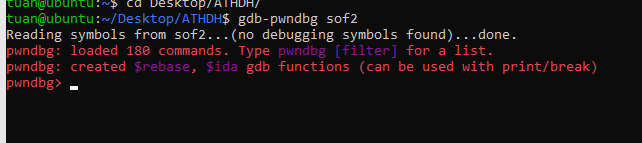
Miễn là nó không ra dòng “error” màu đỏ thì là oke.

Tiếp theo, mình sẽ giải thích cái lỗi ở đây là gì và tại sao lại gọi về được.

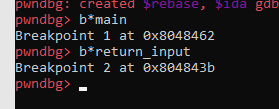
Như các bạn đã biết, 1 hàm nó sẽ có địa chỉ của hàm đó,trong bài này, ta có hàm return\_input().

Để xem được địa chỉ của hàm đó, chúng ta phải debug thì mới thấy được , mình sẽ chụp từng bước debug và cả cách xài cái tool debug pwndbg mới bảo các bạn xài ở trên:

Các bạn bắt đầu debug bằng lệnh sau: gdb sof2 , nhớ rằng sof2 là file sau khi compile thành, chứ không phải file .c nhé.

Câu lệnh của mình hơi khác xíu, các bạn đừng bắt chước. cứ làm như ở trên:  


Tiếp theo các bạn gõ lệnh theo từng bước nha.



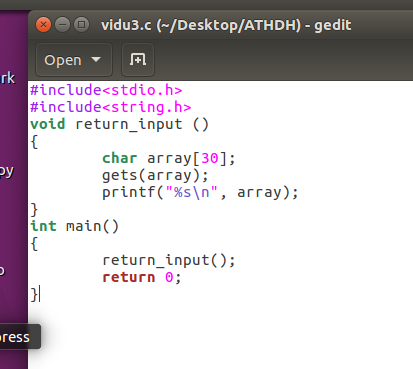
Đây, như các bạn thấy, 2 câu lệnh trên, mình vừa đặt breakpoint ở 2 hàm, hàm main và hàm return\_input, breakpoint là gì, để làm gì thì khi học môn C++ , các bạn debug chắc cũng biết về breakpoint rôi, mình ko giải thích dài.

ở trên , ta thấy khi đặt breakpoint ở main thì nó hiện ra là “Breakpoint 1 at 0x8048462” , thì tức là địa chỉ của main là 0x8048462.

Tương tự, địa chỉ của hàm return\_input ở 0x804843b.

Rồi, bây giờ ta đã có địa chỉ, tức là nếu muốn gọi hàm đấy, ta chỉ cần gọi cái địa chỉ đó, là hàm đó sẽ được gọi. Các bạn ghi 2 thông tin này ra giấy, kẻo quên.

LỖI: chương trình không cung cấp 1 quá trình nào giúp các bạn gọi hàm return\_input đấy 2 lần, nhưng đề yêu cầu, vì thế chúng ta phải khai thác 1 trong những lỗi của đề bài. Đọc lại code, ta sẽ xem đề bài có lỗi gì?



Dễ dàng nhìn thấy, trong hàm return\_input, chương trình dùng hàm gets( ) để nhập vào biến array, nhưng các bạn có tự hỏi rằng: vì sao không thấy nó kiểm soát số lượng kí tự được phép nhập vào không? Không có đúng không, vậy nếu ta nhập 100, 1000, 1 triệu kí tự thì sao? Trong khi phần tử của array là hữu hạn, chỉ có 30 kí tự được phép nhập vào thôi.

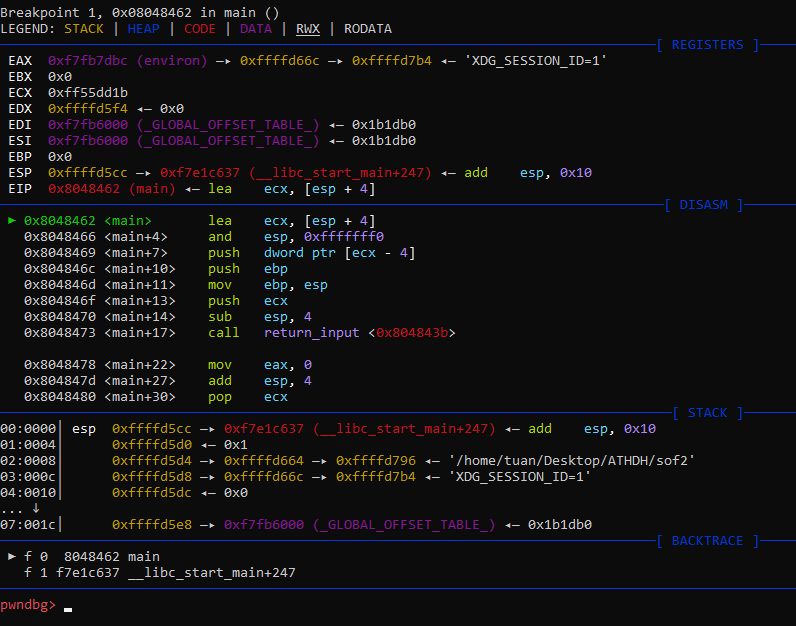
Tức là, ở hàm input\_array đã xảy ra lỗi Stack Overflow ( lỗi tràn stack).

Dựa vào đấy, ta sẽ khai thác lỗi này để thực hiện yêu cầu của đề.

Tiếp theo ta sẽ debug để tiếp tục quan sát quá trình gây lỗi của hàm này.

 các bạn bấm r + enter để nó bắt đầu chạy debugger.

Nó sẽ bắt đầu dừng ở breakpoint 1( là hàm main) cho chúng ta xem, bởi vì lúc nãy mình đã đặt breakpoint rồi:



3

2

1

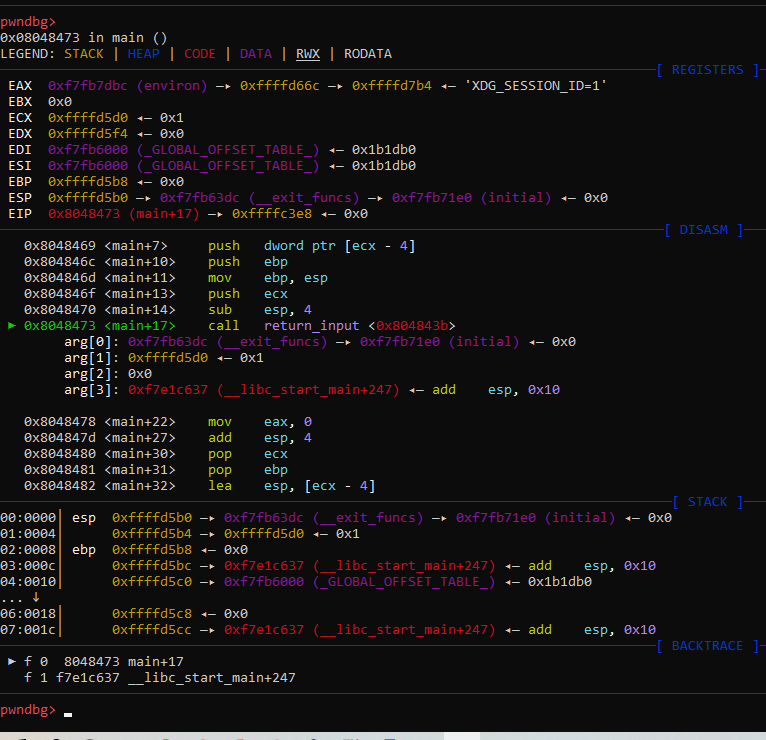
Các bạn thấy, ở khung đầu tiên, sẽ là các thanh ghi, chức năng thanh ghi, chưa dữ liệu gì thì các bạn đã học qua, mở vở xem lại nghen.

Ở khung thứ 2, chính là lệnh assembly và địa chỉ của lệnh đó.

ở khung thứ 3, chính là stack của chúng ta.

Ta bắt đầu đọc khung thứ 2, vì quan trọng nhất vẫn là xem cái lỗi nó làm gì, còn lại từ từ.

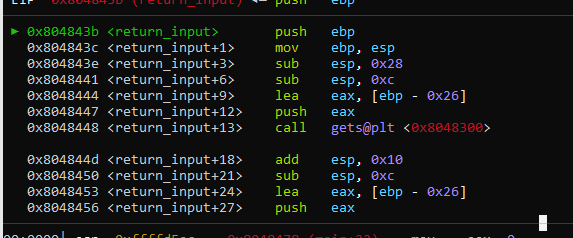
Để chạy từng dòng lệnh, các bạn bấm “ni” rồi enter, nó sẽ chạy từng lệnh, từng lệnh.



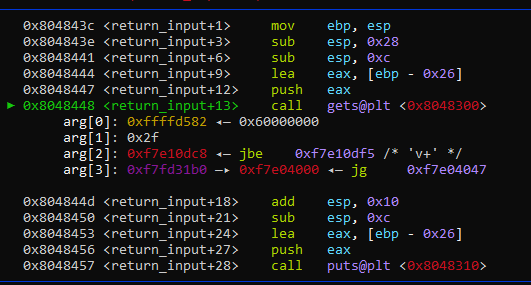
ở đây, mình cho nó chạy đến lệnh call hàm return\_input, các bạn có thấy bên cạnh cái tên hàm là địa chỉ hàm luôn không? Chính là cái địa chỉ mình bảo các bạn lưu lại đấy.

phía dưới hàm gets() là địa chỉ của lệnh (main+22) , cái địa chỉ này là gì, như các bạn đã học, khi gặp ngắt, nó thực thi ngắt, hết ngắt nó sẽ quay lại vị trí cũ và tiếp tục, thì main+22 này cũng vậy, sau khi nó gọi hàm return\_input thì nó sẽ quay lại thực thi tiếp lệnh main+22 này, các bạn lưu cái lệnh này lại, nhớ nó, lát nữa nó có ý nghĩa đặc biệt.

tiếp theo, chúng ta sẽ gõ lệnh “si” +enter để vào hàm return\_input xem nó làm gì.



Nó sẽ ra như vày, tức là ta đang debug hàm return\_input() , các bạn có thấy lệnh call gets() không, tức là nó gọi hàm gets() đấy, trong code C ban đầu chúng ta có gọi hàm gets() trong hàm return\_input() đúng không, các bạn liên hệ giữa assembly và code C để dễ dàng hiểu được nó đang làm gì.

Tạm thời thì không thể nào minh hướng dẫn các bạn từng dòng lệnh nó làm gì, nên chúng ta sẽ đi thẳng tới hàm khi nó call hàm gets() luôn. Các bạn tiếp tục chạy từng dòng bằng lệnh ni + enter để tới hàm gets(). Đây, nó sẽ ra như này:  


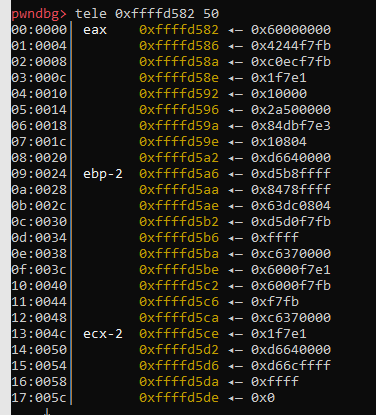
Mình giải thích ngắn gọn mấy cái ở dưới hàm gets() là gì. Nó để arg[0], arg ở đây có nghĩa là argument: tham số. tức là đó là tham số của hàm gets() . Mà trên code C, hàm gets() chỉ nhận 1 tham số, đó là array[30] , thì arg[0] đấy , chính là địa chỉ của array[30] .

Các bạn lưu lại địa chỉ array[30] cho mình nha: 0xffffd582

Warning: địa chỉ của các bạn có thể khác mình , sure là khác luôn, nên đừng hoảng hốt, cứ dựa vào nó mà xem tiếp.

Ta đã nói ở trên , cái array này chỉ chứa 30 kí tự, mà hàm này thì cho nhập vào vô hạn luôn. Ta sẽ dùng lệnh tele 0xffffd582 50 để quan sát 50 địa chỉ stack bắt đầu từ array này.

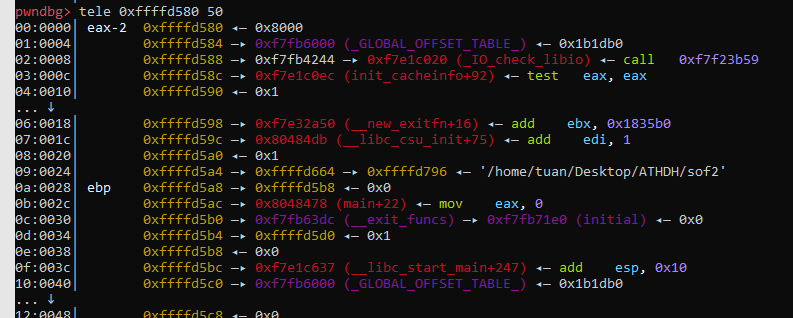
Nếu các bạn bấm lệnh trên mà nó ra như vày:



Thì các bạn lấy địa chỉ đó, trừ đi 2, rồi làm lại , ví dụ là mình thì sẽ là:

tele 0xffffd582 50

Nếu nó ra như vày là ổn:



Bây giờ, các bạn có thấy cái ebp kia không, thanh ghi ebp giữ cái gì , bạn nào quên thì mình giải thích luôn, đấy là nó lưu địa chỉ (ebp frame) cũ và sau khi nó chạy xong hàm return\_input nó sẽ quay về ebp này để quay lại thực thi tiếp lệnh tiếp theo ở ebp+4.

Mà địa chỉ ebp+4 , như các bạn thấy trong hình, đó là địa chỉ của main+22. Mình dã bảo các bạn lưu địa chỉ đó ở trên, ý nghĩa đặc biệt của nó chính là địa chỉ trở về sau khi gọi hàm return\_input( ) . và nó đang được đặc trong stack.

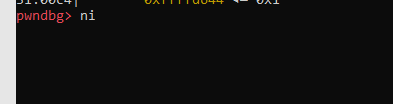
Cái địa chỉ màu vàng là địa chỉ của stack, thì giá trị return của return\_input nằm ở địa chỉ ebp+4 = 0xffffd5ac, các bạn cũng lưu cái này lại, để đấy tí nữa mới có trò vui.

Giờ các bạn để ý nha, array cũng nằm trên stack, địa chỉ là 0xffffd582.

Địa chỉ return cũng nằm trên stack, địa chỉ là 0xffffd5ac.

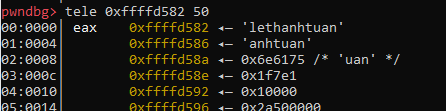
Tức là, nó nằm trên cùng 1 vùng nhớ stack, nhớ điểm này, cực kì quan trọng.

Rồi, tiếp theo là mình thử nhập xem nó làm gì. Bấm ni +enter phát nữa xem nó làm gì?

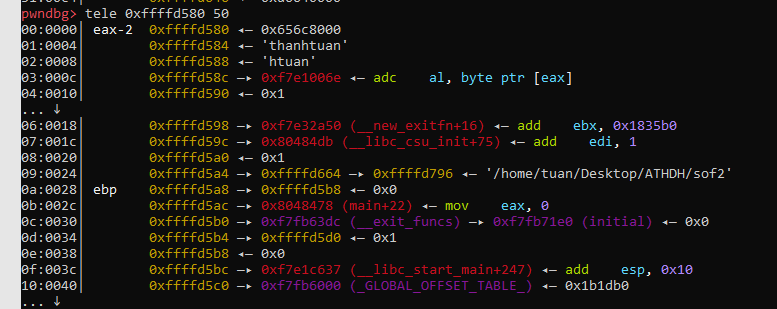
 ở đây là nó đang dừng cho mình nhập á, nhập vào nha các bạn, thử nhập “lethanhtuan”

 rồi enter cho nó chạy tiếp.

Ta đã nhập vào array, giờ ta sẽ xem thử array có chứa đúng giá trị mà ta vừa nhập hay không?

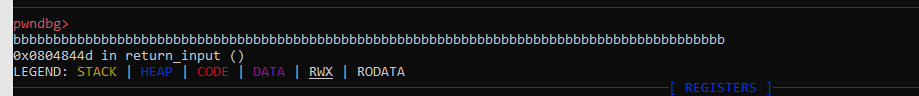
 oke, sure là chắc chắn đúng rồi, khỏi bàn.

Bây giờ, các bạn nhìn nhé, khi ta nhập giá trị vào thì giá trị sẽ được ghi dần vào các địa chỉ tiếp theo. Mà array ở địa chỉ 0xffffd582, còn return nằm ở 0xffffd5ac. Tức là địa chỉ return nằm dưới địa chỉ array đúng chưa? Vì 0xffffd5ac > 0xffffd582. Đây , các bạn nhìn hình dưới sẽ thấy rõ:

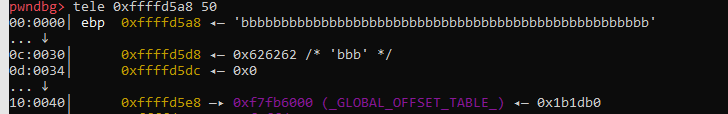


Vậy nếu, ta nhập thiệt nhiều nhiều thì sẽ ghi dần ghi dần và ghi vào địa chỉ chứa return luôn, đúng không?

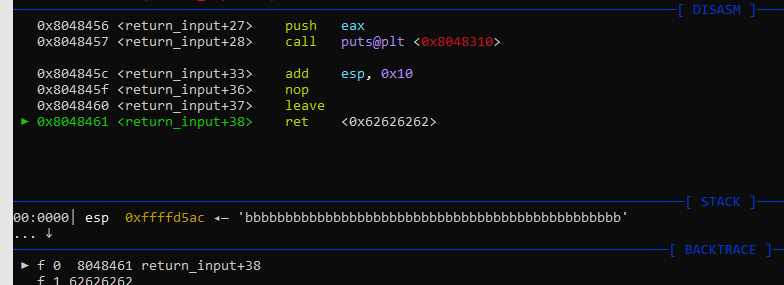
Ví dụ mình run lại 1 lần nữa rồi chạy tới hàm gets() , rồi nhập thiệt nhiều chứ ‘b’ như vày:



Giờ cùng xem stack nó ra sao nhé, coi thử địa chỉ return ở ebp+4 có còn tồn tại hay không?

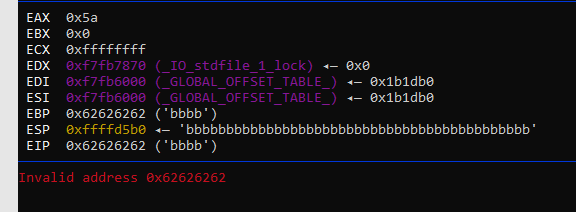


Đây là ebp đây, omg, các bạn xem, toàn là chữ “b” , tức là chúng ta đã làm tràn cái array đến địa chỉ ebp và làm mất đi địa chỉ cũ. Giờ chạy tiếp xem nó có return về được main+22 như đã nói không?



Các bạn nhìn hàm này , đúng là nó return về địa chỉ 0xffffd5ac, như chúng ta đã nói, nhưng ở đó lại không phải lệnh main+22 như trước nữa, mà là 1 mớ bbbbbbbbbbbbbbbbbbbbbbbbbbb.

Rồi giơ “ni” tiếp nó sẽ lỗi luôn cho xem. Đấy, nó bảo địa chỉ 0x62626262 là gì,tao không hiểu.



Vậy , giờ các bạn đã hiểu được làm sao mà ta có thể overflow rồi. vậy ta khai thác ra sao?

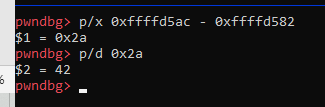
Khai thác như sau, vì chúng ta có thể tự do nhập vào và làm tràn địa chỉ ebp+4, vậy nếu chúng ta ghi vào địa chỉ ebp+4 không phải mấy dòng chữ “bbbb” vô nghĩa mà là 1 địa chỉ nào đấy thì sao???

Đề bài yêu cầu quay về hàm return\_input đúng không, vậy ta chọn địa chỉ của nó luôn, địa chỉ này các bạn đã lưu lại rồi mà.

Vậy, câu hỏi thắc mắc tiếp theo là: Nhập bao nhiêu cho nó tràn tới địa chỉ ebp+4?

Mình sẽ chỉ các bạn tính toán luôn:

Ta lấy (địa chỉ ebp+4) – (địa chỉ array) , ta được:



Ta được 0x2c, tức là ta cần nhập đến (0x2c)h = 42 kí tự thì mới tràn tới được địa chỉ ebp+4.

Vậy payload của ta sẽ như sau, nhập 44 kí tự bất kì rồi tiếp theo là 4 bytes địa chỉ của hàm return\_input:

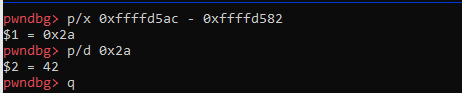
Payload = ‘a’\*42 + địa chỉ return\_input

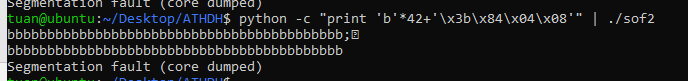
Địa chỉ return\_input ta phải đổi sang dạng byte hexa thì nó mới hiểu được:

0x804843b = ‘\x3b\x84\x04\x08’

Vậy: payload = ‘a’\*42 + ‘\x3b\x84\x04\x08’

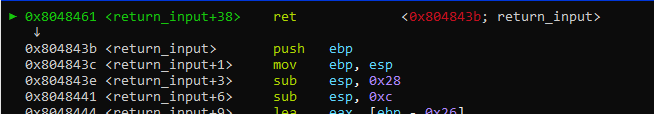
Rồi ta, thoát ra bằng lệnh ‘q’ + enter, rồi thử payload ta vừa tạo xem có đúng hay không?





YEAHHHHHHHHHHHHHHHHHHHH ! xong rồi, ngon chưa các bạn.

Vì như ta đã nói ở trên, sau khi nhập 42 bytes kí tự thì 4 bytes tiếp theo sẽ ghi đè ebp+4 , cũng chính là địa chỉ return của hàm return\_input, giờ ta lại ghi đè vào địa chỉ hàm return\_input, thì đương nhiên nó sẽ return về lại hàm return\_input rồi:



ENDING !!!

Thêm tí: Câu lệnh payload ở trên mình giải thích như sau, gồm 2 phần:

Phần 1: python -c "print 'b'\*42+'\x3b\x84\x04\x08'" : dùng python để in ra chuỗi payload.

Phần 2: | ./sof2 , dùng để truyền chuỗi payload đã tạo kia, thay vì đưa ra in ra màn hình, thì lấy nó làm đầu vào input cho file thực thi sof2.