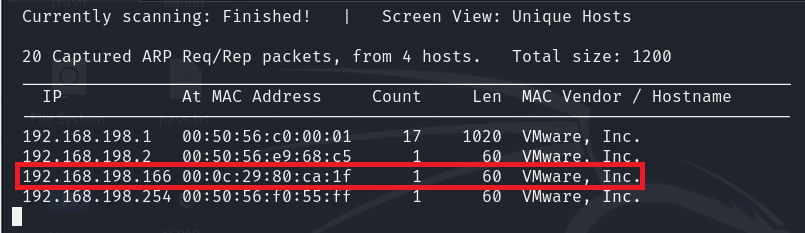
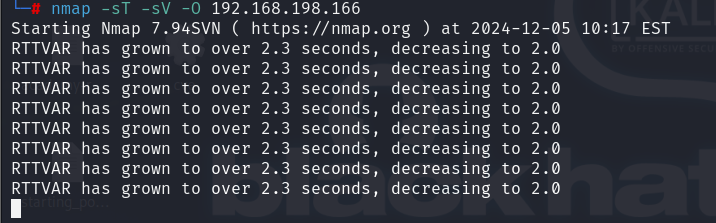
1. **Chuẩn bị**

Trước tiên, hãy tìm IP của VM.

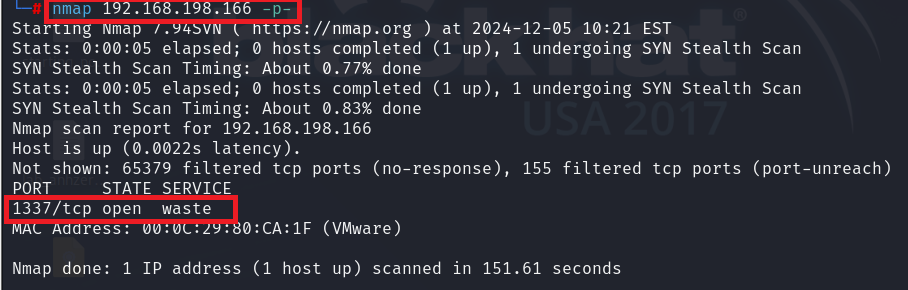
**netdiscover -i eth0 -r 192.168.198.0/24**



Với IP, hãy chạy quét Nmap nhanh, nhưng mọi thứ không dễ dàng và tốt đẹp như bình thường…

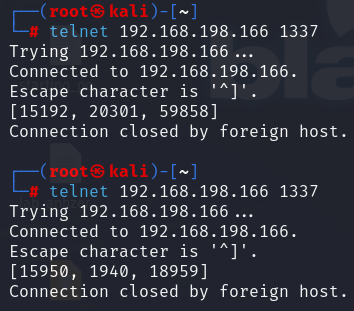


Sau một hồi mày mò với các cờ và kiểu quét, nên quyết định quét toàn bộ cổng và chờ đợi.

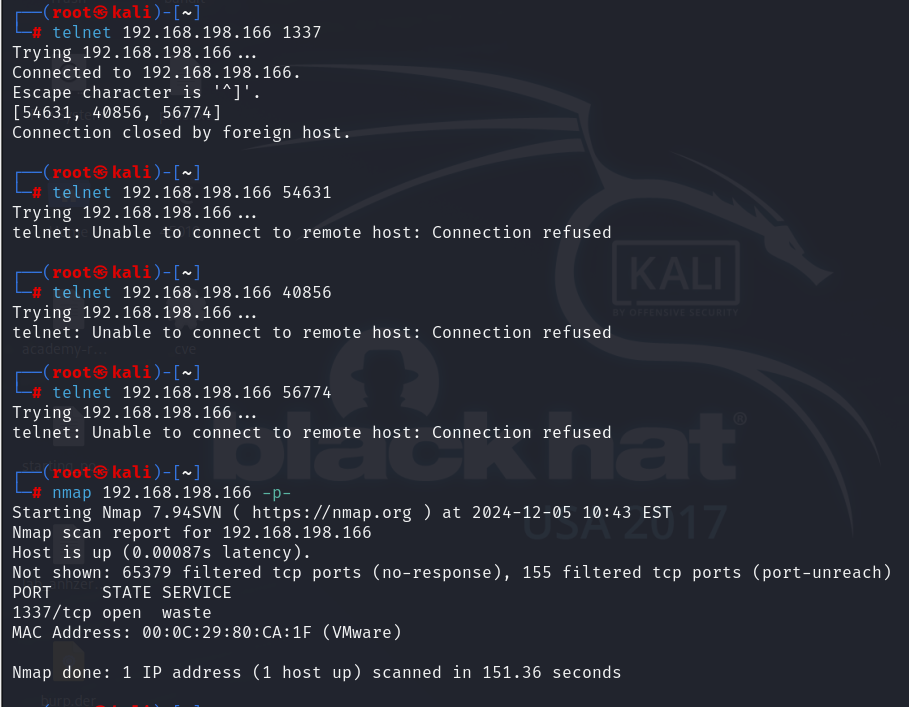


1. **Khai thác**

Vì ít nhất đã có một cổng để bắt đầu, nên ta quyết định telnet vào cổng đó để xem có thể tìm thấy gì.



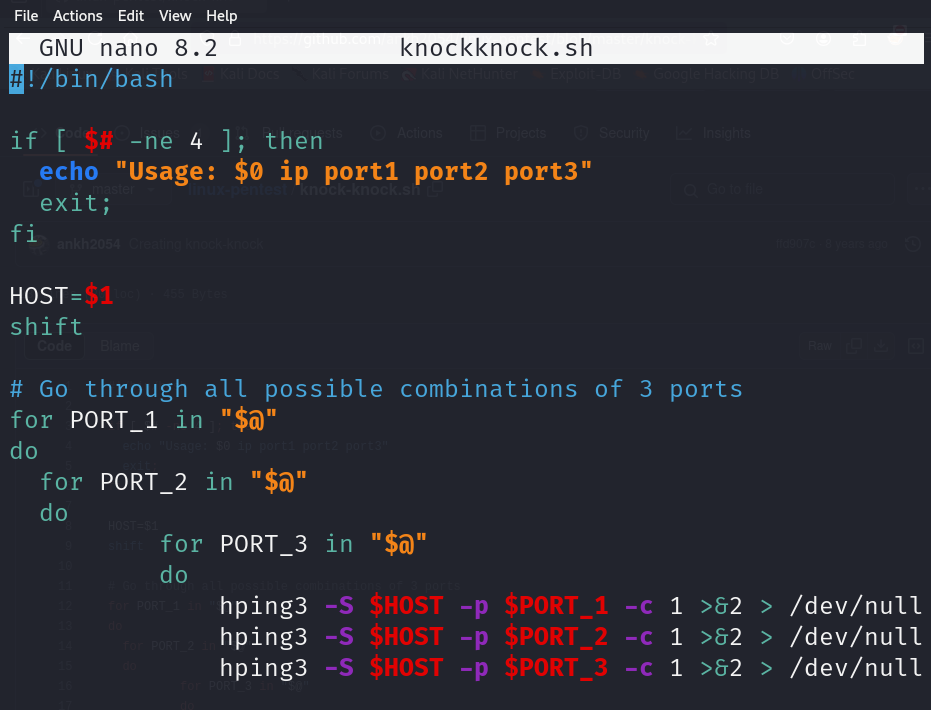
Có vẻ như có 3 cổng được cấp sau khi telnet vào 1337 và dựa trên tên của VM, ta có thể cho rằng đó chỉ là một lệnh port-knock đơn giản (tất nhiên là không phải vậy).



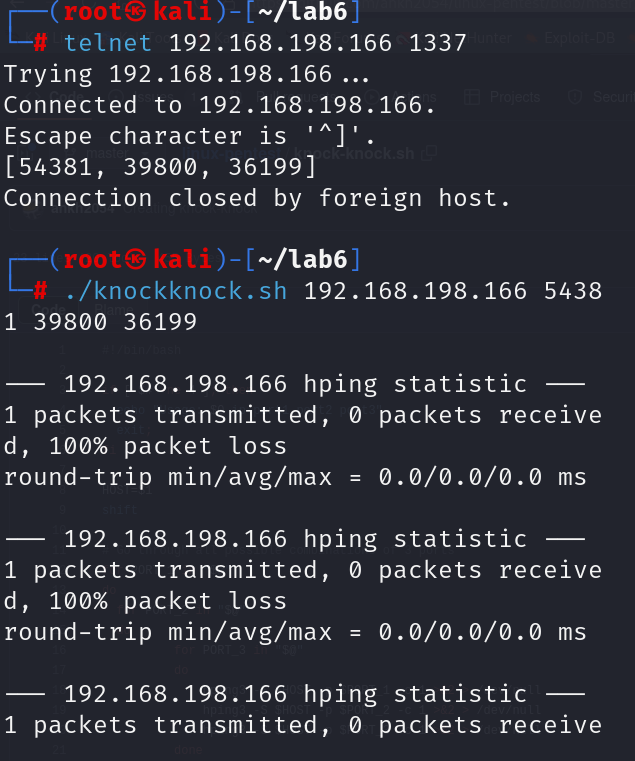
Vì phương pháp \*bang fist against door and hope for access\* không hiệu quả nên ta quyết định thử một phương pháp tinh tế hơn (hoặc ít nhất là theo more scripted).

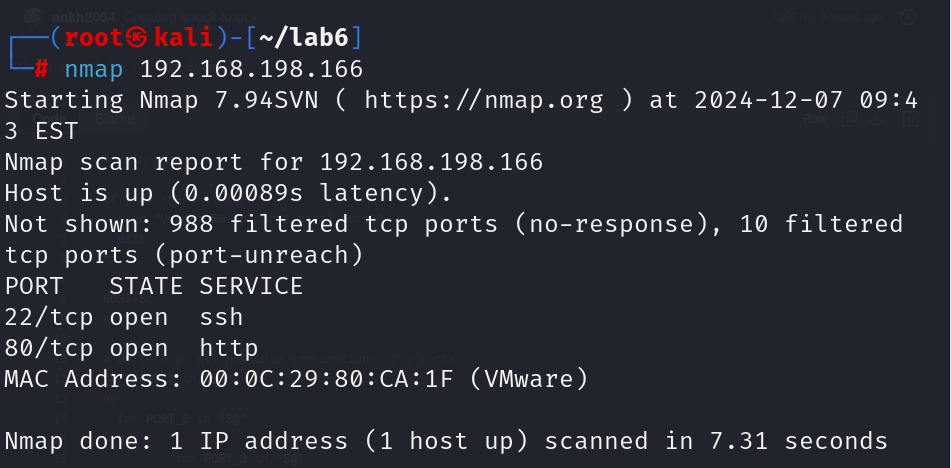
Đây là kỹ thuật port knocking, cho phép giấu đi các port và như phương thức giải đố, phải yêu cầu thì mới có thể mở các port thực sự hoạt động.

Link script: <https://github.com/ankh2054/linux-pentest/blob/master/knock-knock.sh>

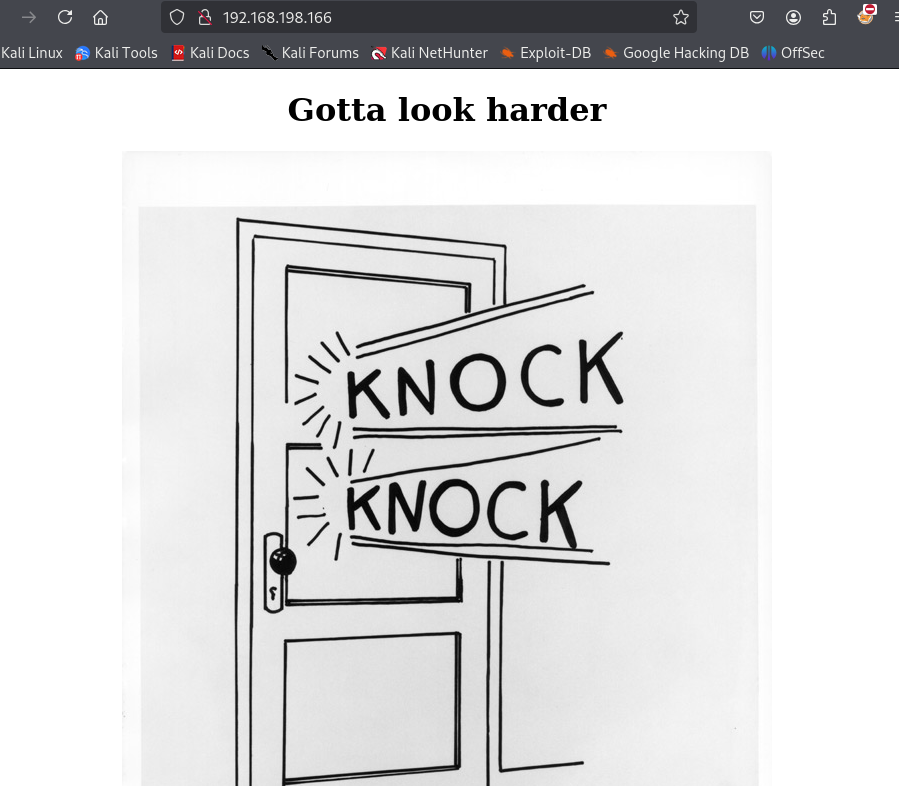


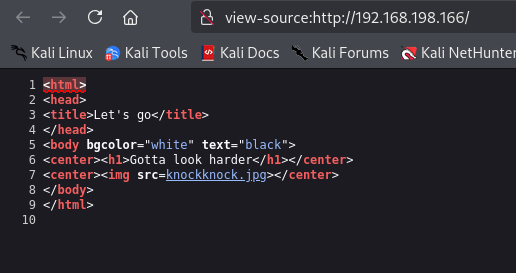
Với tập lệnh shell đã chuẩn bị, chạy nó và lần này có thêm một số cổng mở



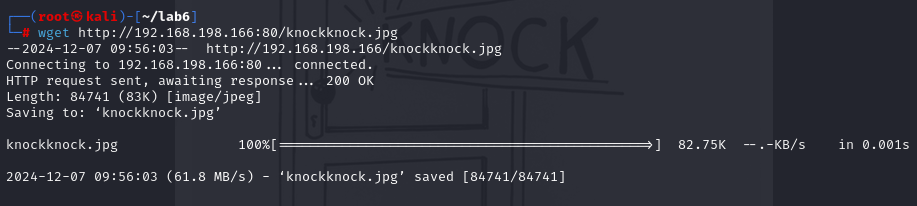


Chúng ta hãy mở trình duyệt và truy cập cổng 80 đang chạy dịch vụ http



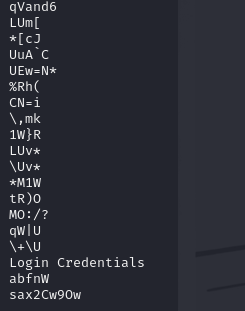


Thật không may, chúng ta không thể tìm thêm thông tin về dirbuster, nguồn hoặc bất kỳ thông tin nào khác hiện có, vì vậy chúng ta quyết định xem xét kỹ hơn tệp hình ảnh.

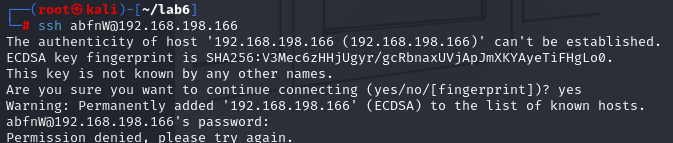


Sau nhiều lần thử tìm thông tin trong tệp hình ảnh, strings đã cung cấp cho ta một phần có vẻ là thông tin đăng nhập, đây là một khởi đầu tuyệt vời.

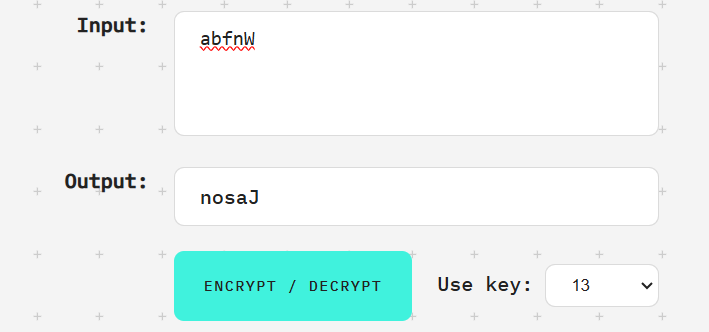


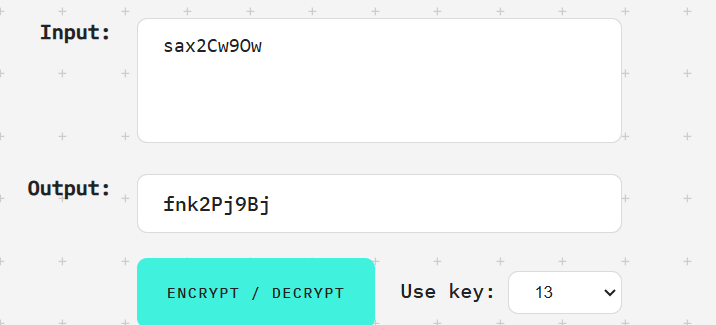


Thật không may, thoạt nhìn, những "credentials" trông lạ này thực ra không có tác dụng gì với chiếc hộp.

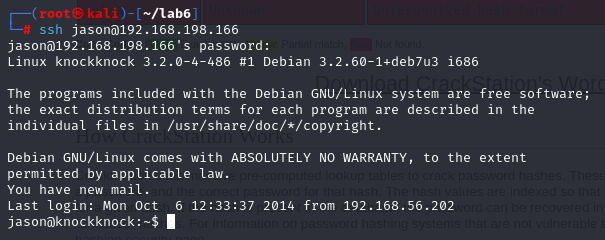


Sau nhiều lần mày mò với [công cụ giải mã Caesar](https://raw.org/tool/caesar-cipher/), cuối cùng ta cũng nhận ra rằng việc dịch chuyển 13 (ROT-13, đáng lẽ phải thử sớm hơn) sẽ cho ra tên người dùng là nosaJ, thực chất là Jason đảo ngược lại.

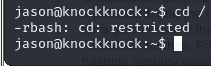




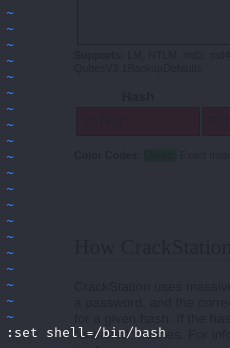
Với thông tin đăng nhập chưa được đảo ngược (Jason/jB9jP2knf) trong tay, ta đã có thể đăng nhập thành công vào hộp.

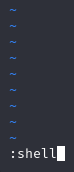


Tuy nhiên, khi ta thực sự vào hệ thống, ta bị mắc kẹt bên trong một lớp vỏ bị hạn chế.



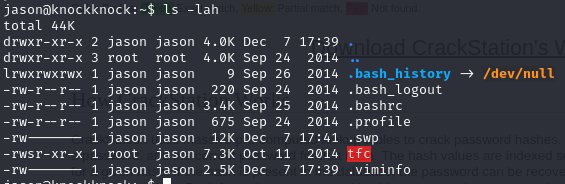
Để thoát khỏi shell bị hạn chế, mở VI và thiết lập biến shell.



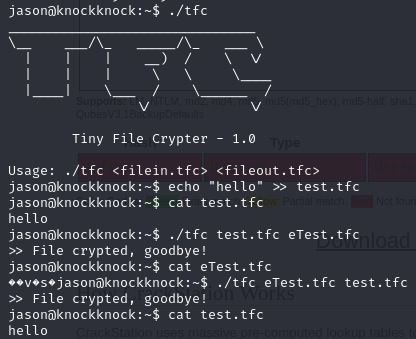




Bên trong thư mục home của jason có một file root set-uid có tên là tfc, vì vậy có lẽ đó là điều tiếp theo ta cần khai thác.

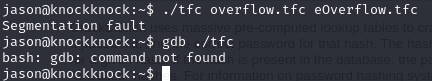


Có vẻ như tfc là một chương trình mã hóa hai chiều xử lý các tệp văn bản thuần túy có phần mở rộng .tfc.

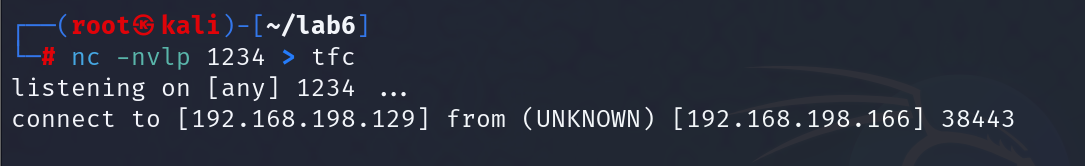


Đầu tiên, kiểm tra lỗi tràn bộ đệm và đã tìm thấy lỗi này. Thật không may, GDB không nằm trên hệ thống đích.



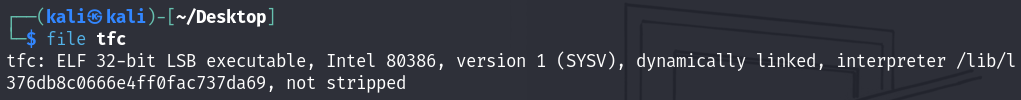


Chuyển ứng dụng tfc trở lại hộp tấn công của mình và thử chạy nó theo một mô hình tuần hoàn để cố gắng tìm ra offset của EIP.

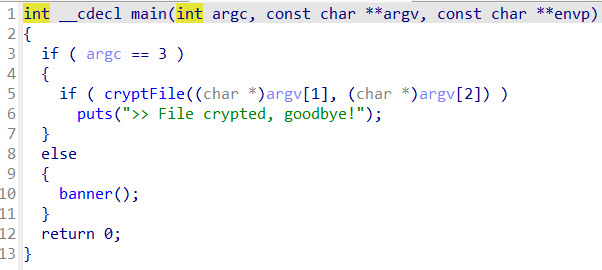




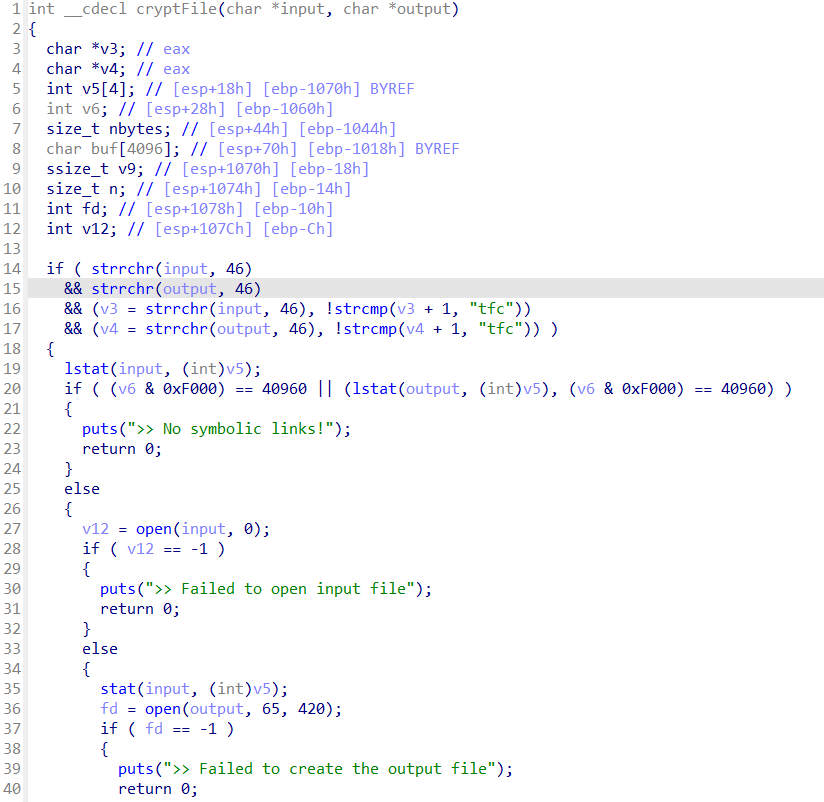
* Phân tích file trên máy Kali, thông tin cho thấy đây là file thực thi 32 bit của linux

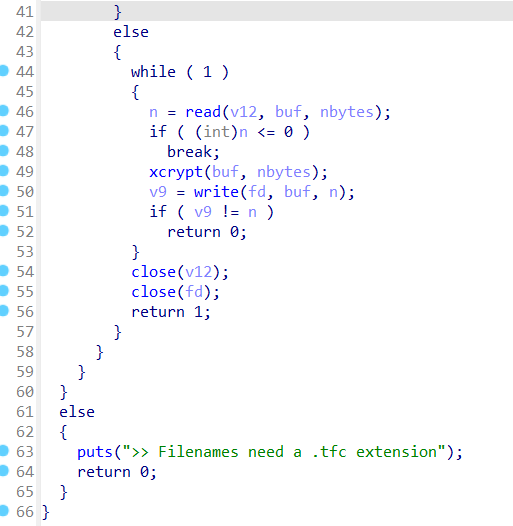


* Dùng IDA để phân tích kỹ hơn
* Thông tin trong hàm main cho thấy người dùng phải truyền 2 tham số tương ứng với với filein.tfc và fileout.tfc vào hàm cryptFile

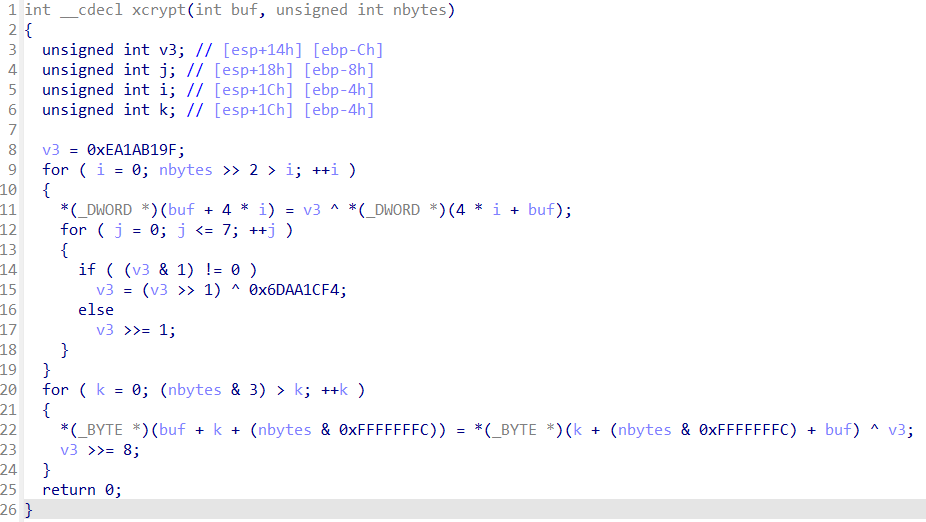


* Trong hàm cryptFile, đoạn If đầu sẽ kiểm tra xem file in và file out người dùng truyền vào có đuôi .tfc không, sau đó đọc thông tin từ file in bằng hàm read lưu vào biến buf được khai báo là mảng char 4096 byte với giá trị cho phép người dùng nhập tối đa lưu trong biến nbytes.
* Sau đó truyền giá trị giá trị buf và nbytes vào hàm xcrypt để thưc hiện mã hoá. Kết quả cho ra sẽ được lưu trong hàm file out, tạo mới nếu file out chưa tồn tại.





* Hàm xcrypt thực hiện mã hoá với 2 tham số đầu vào là buf và nbytes và trả giá trị buf đã được mã hoá



* Trong hàm cryptFile trên nếu biến nbytes có giá trị lớn hơn 4096 tức cho phép file được lưu trữ lớn hơn 4096 ký tự sẽ bị tràn bộ đệm khi lưu vào biến biến buf
* Thực hiện kiểm tra giá trị của biến nbytes qua phân tích động bằng gdb – gef



* Tạo file in

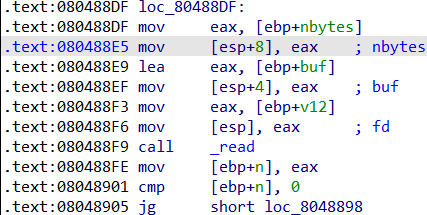


* Truyền 2 đối số filein và fileout trước khi gọi hàm cryptFile tránh thoát khỏi chương trình trước khi đến được biến nbytes



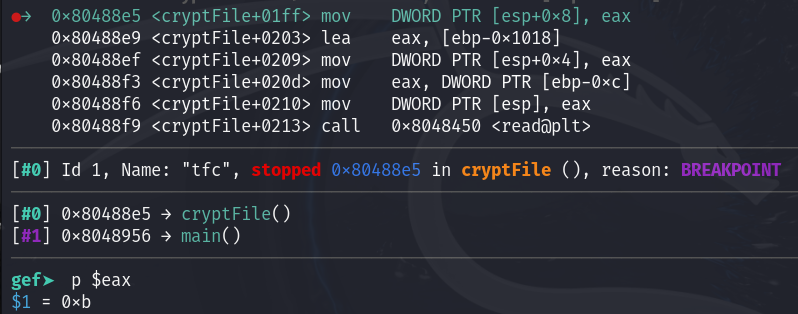
* Sử dụng IDA để tìm địa chỉ của biến nbytes trong bộ nhớ





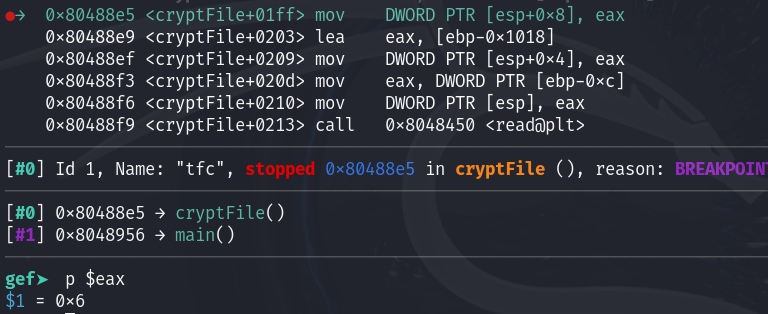
* Đặt breakpoint tại địa chỉ 0x080488E5 khi mà giá trị của biến nbytes đã được lưu trong thành ghi eax, chạy gdb đến breakpoint và in giá trị eax ra màn hình



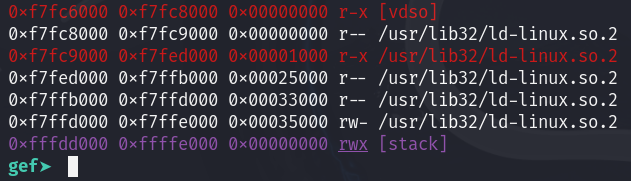


* Khi người dùng lưu 10 ký tự trong filein thì nbytes là 0xb tức 11, thử lại với 5 ký tự lưu trong filein thì nbytes là 0x6 tức 6. Vậy nbytes chính là số bytes được lưu trong filein, từ đây có thể khai thác lỗ hỏng buffer overflow khi mà không giới hạn đầu vào trong khi buf chỉ được khai báo 4096 byte

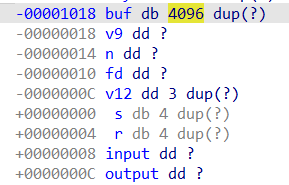




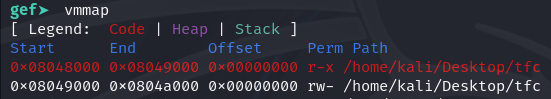
* Sử dụng vmmap trong gdb cho biết rằng bộ nhớ stack có thể thực thi



* Ý tưởng bài này là sẽ ghi shellcode vào địa chỉ stack, khai thác lỗ hỏng buffer overflow ghi đè giá trị eip để trỏ đến shellcode và thực thi
* Thông tin trên IDA cho biết từ biến buf sẽ phải ghi 4124 giá trị để đến được r (thanh ghi eip)



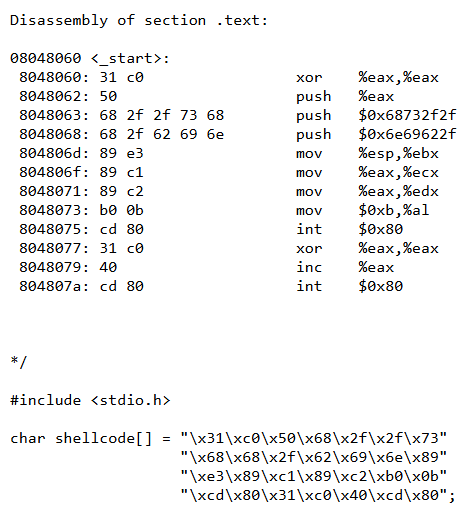
* Tuy nhiên để thực thi được shellcode cần có lệnh jmp hoăc call. Ghi đè địa chỉ jmp esp hoặc call esp vào eip sau đó ghi tiếp shellcode lúc run đến eip next step sẽ chuyển tiếp đến lệnh jmp esp hoặc call esp và shellcode lúc này sẽ nằm trên đỉnh stack nơi mà thanh ghi esp sẽ trỏ tới và khi run tiếp sẽ thực thi shellcode.
* Dùng vmmap trên gdb cho biết có địa chỉ vùng nhớ của file từ 0x08048000 đến 0x08049000 có thể đọc và thực thi



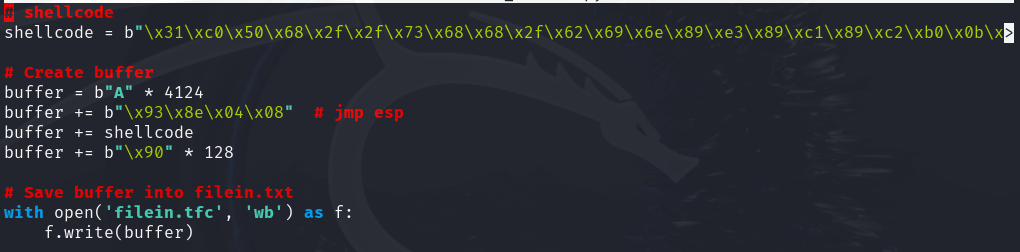
* Tìm kiếm opcode của jmp esp (FF E4) bằng lệnh find trên gdb thì tìm thấy địa chỉ 0x8048e93 có chứa lệnh jmp esp và có thể thực thi được



* Sử dụng shellcode này để có được shell /bin/bash trong file thực thi 32 bit



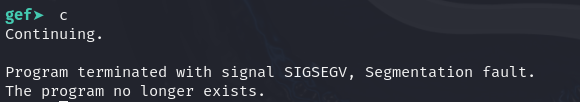
* Code python tạo payload lưu vào filein.tfc



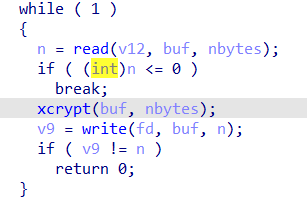
* Chạy file python trên, thự mục hiện tại sẽ tạo file filein.tfc
* Mở gdb tfc, set đối số là filein.tfc vừa tạo, fileout.tfc để lưu kết quả, thực thi file tfc bằng lệnh start và c (continue)



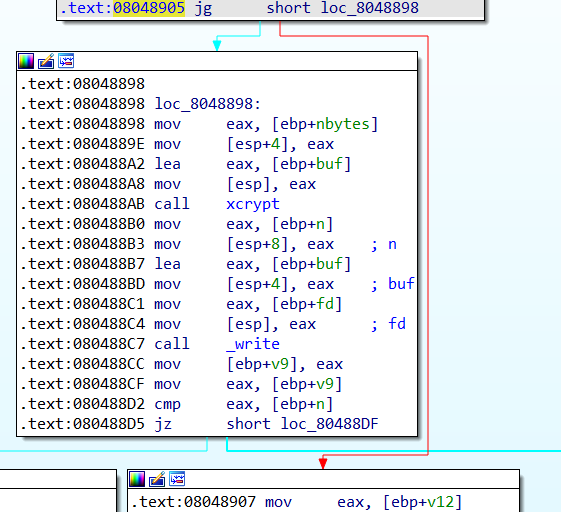
* Kết quả chương trình đã bị Segmentation fault, quay trở lại IDA thấy rằng biến buf có thay đổi giá trị khi được truyền vào hàm xcrypt thực hiện mã hoá dẫn đến địa chỉ lệnh jmp esp và shellcode đã bị thay đổi và file không còn thực thi như mong đợi



* Thử bypass qua đoạn xcrypt để giá trị buf khi đọc vẫn giữ nguyên xem payload có hoạt động bằng chỉnh cờ ZF (zero flag) cho là 1 để nhảy vào if và break ra trước khi gọi đến hàm xcrypt.



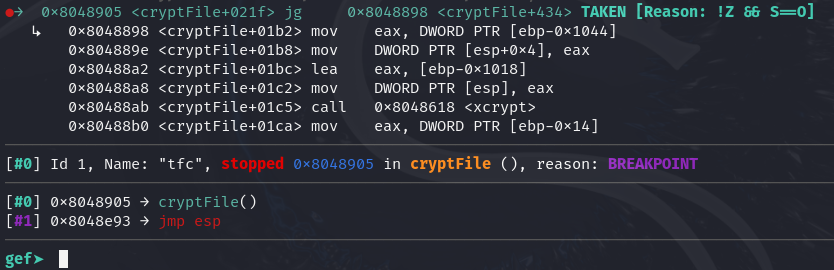
* Từ IDA biết được địa chỉ của lệnh nhảy là 0x08048905



* Dùng gdb đặt breakpoint đến địa chỉ này và cho nhảy đến đó

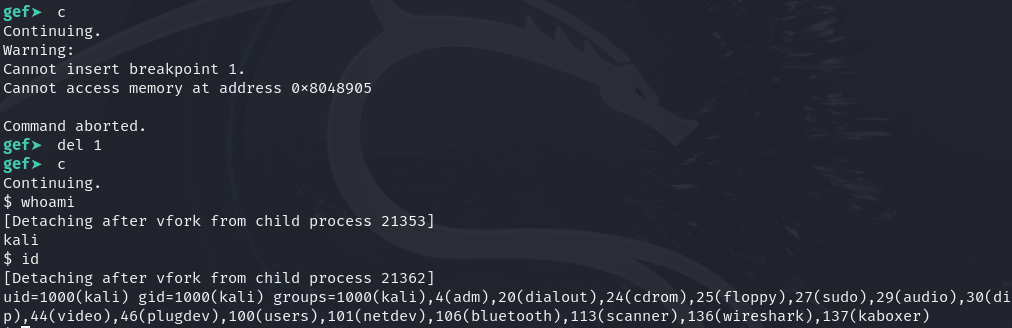


* Khi nhảy đến đến địa chỉ này, phải set flag ZF là 1 và dùng lênh c tiếp tục để break khỏi hàm while mà không vào hàm xcrypt

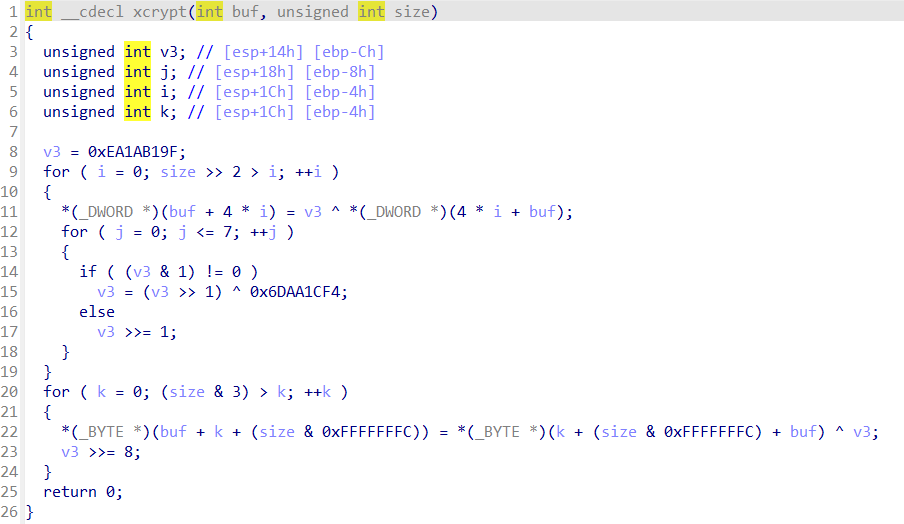




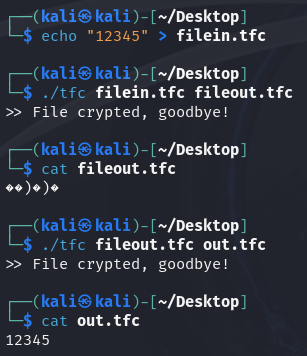
* Cần phải xoá breakpoint đã đặt trước đó vì khi tạo tiến trình mới địa chỉ breakpoint đã đặt trước đó không còn khả dụng, kết quả cho thấy payload đã hoạt động.



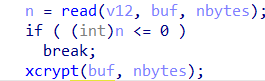
* Vì trên máy nạn nhân không có gdb để có thể thực hiện thay đổi cờ vì vậy cần phải tìm cách khác để bypass qua hàm xcrypt. Quay trở lại với hàm xcrypt dễ thấy buf được xor với một giá trị v3 cho trước thay đổi theo quy luật nhất định tạo thành buf mới, vậy nếu dùng buf mới này xor với v3 này thì sẽ tạo thành buf ban đầu

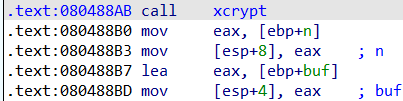


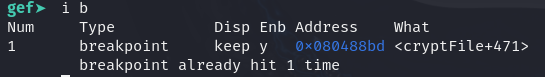
* Kiểm chứng lý thuyết phía trên cho kết quả chính xác



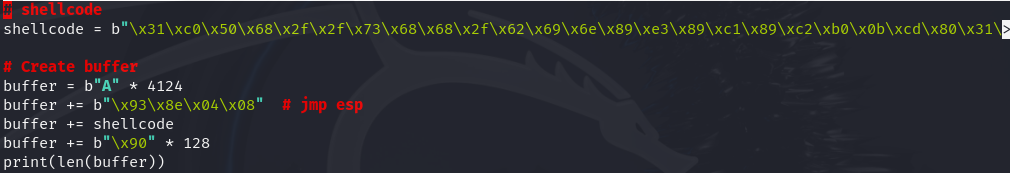
* Ý tưởng là sẽ lấy giá trị biến buf sau khi qua hàm xcrypt cho vào filein.tfc là đầu vào để thực thi file tfc sẽ có được shell, tuy nhiên vì payload truyền vào đã ghi đè lên eip nên kết qua trả ra nếu theo cách trên dùng fileout để làm filein là không chính xác bị lỗi segmentation fault
* Dùng gdb để trích xuất giá trị buf sau khi qua hàm xcrypt. Trên IDA có thể thấy rằng giá trị biến buf sau khi qua hàm xcrypt sẽ được lưu tại thanh ghi eax, đặt breakpoint tại 0x080488bd trên gdb và chạy đến đó







* In độ dài payload

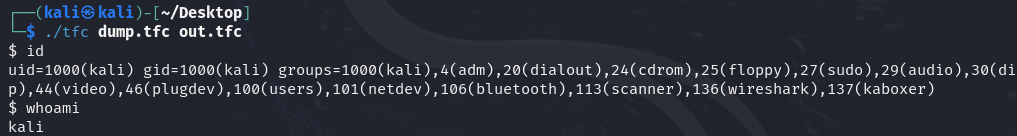




* Sử dung dump memory để lấy 4284 byte tại địa chỉ mà thanh ghi eax đang trỏ tới lưu vào file dump .tfc



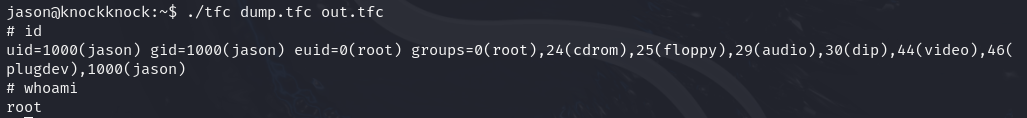
* Chạy thử kết quả thành công



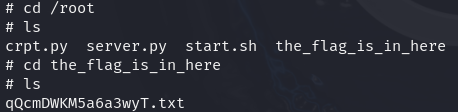
* Từ shell jason có được từ ssh thực hiện download file dump.tfc qua scp

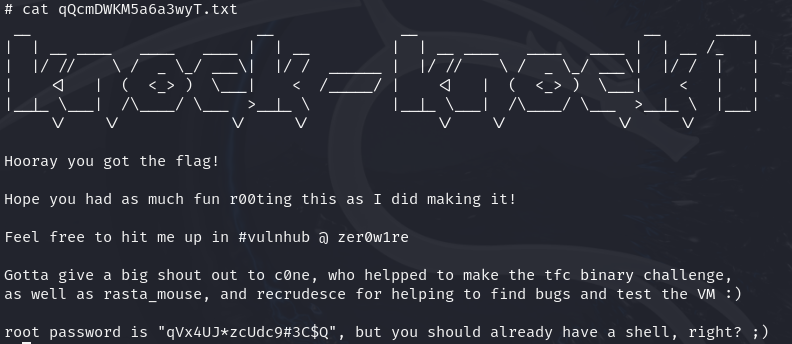


* Thực thi file ./tfc lúc này attacker đã có được shell của root



* Vào thư mục root và lấy flag lưu trong file





# KẾT QUẢ

* Lên được quyền root và lấy được flag trong thư mục /root

