Thông tin sinh viên:

Võ Anh Kiệt - 20520605

Nguyễn Bùi Kim Ngân - 20520648

Nguyễn Bình Thục Trâm - 20520815

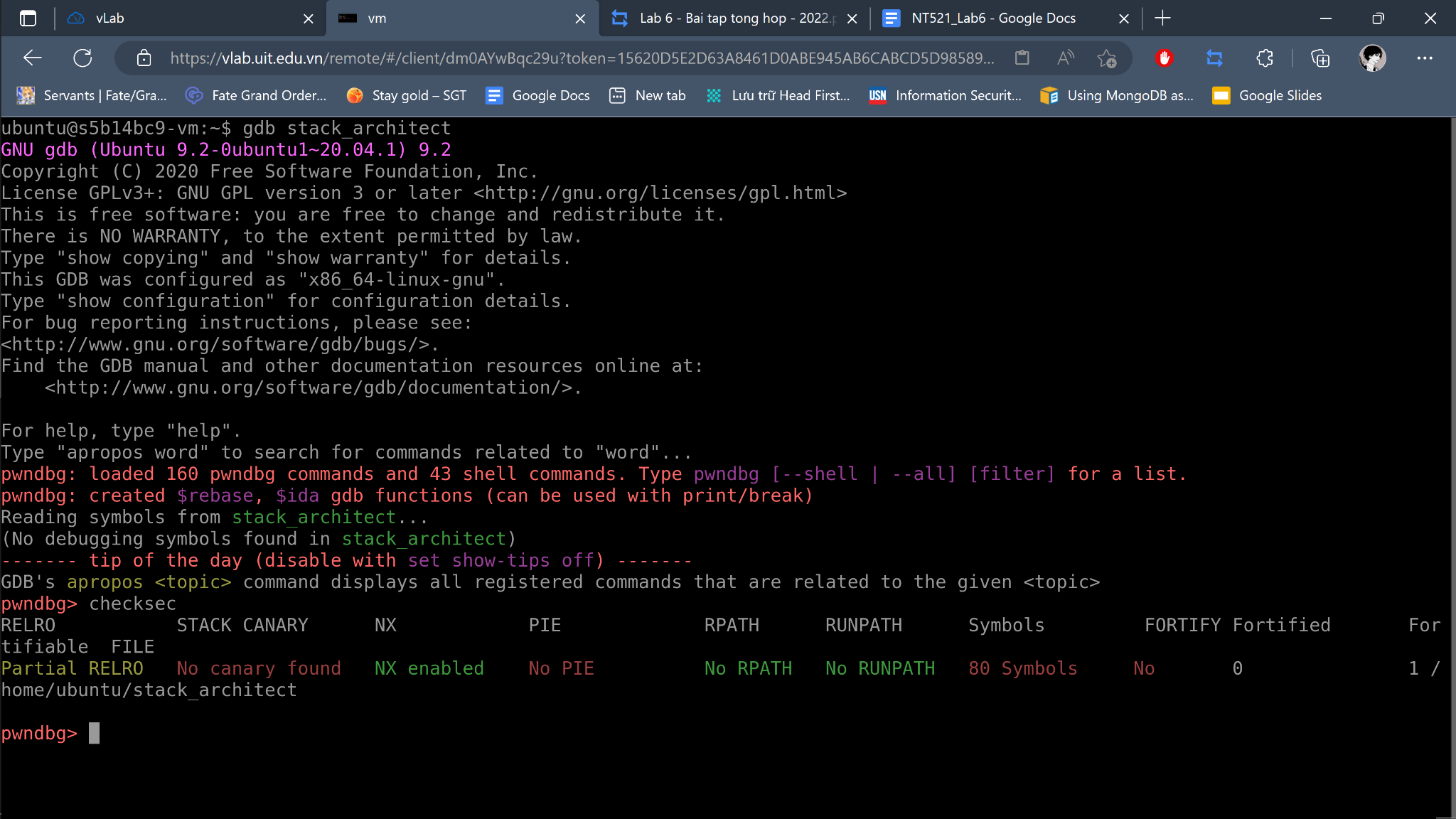
Bài làm

**Stack Architect:**

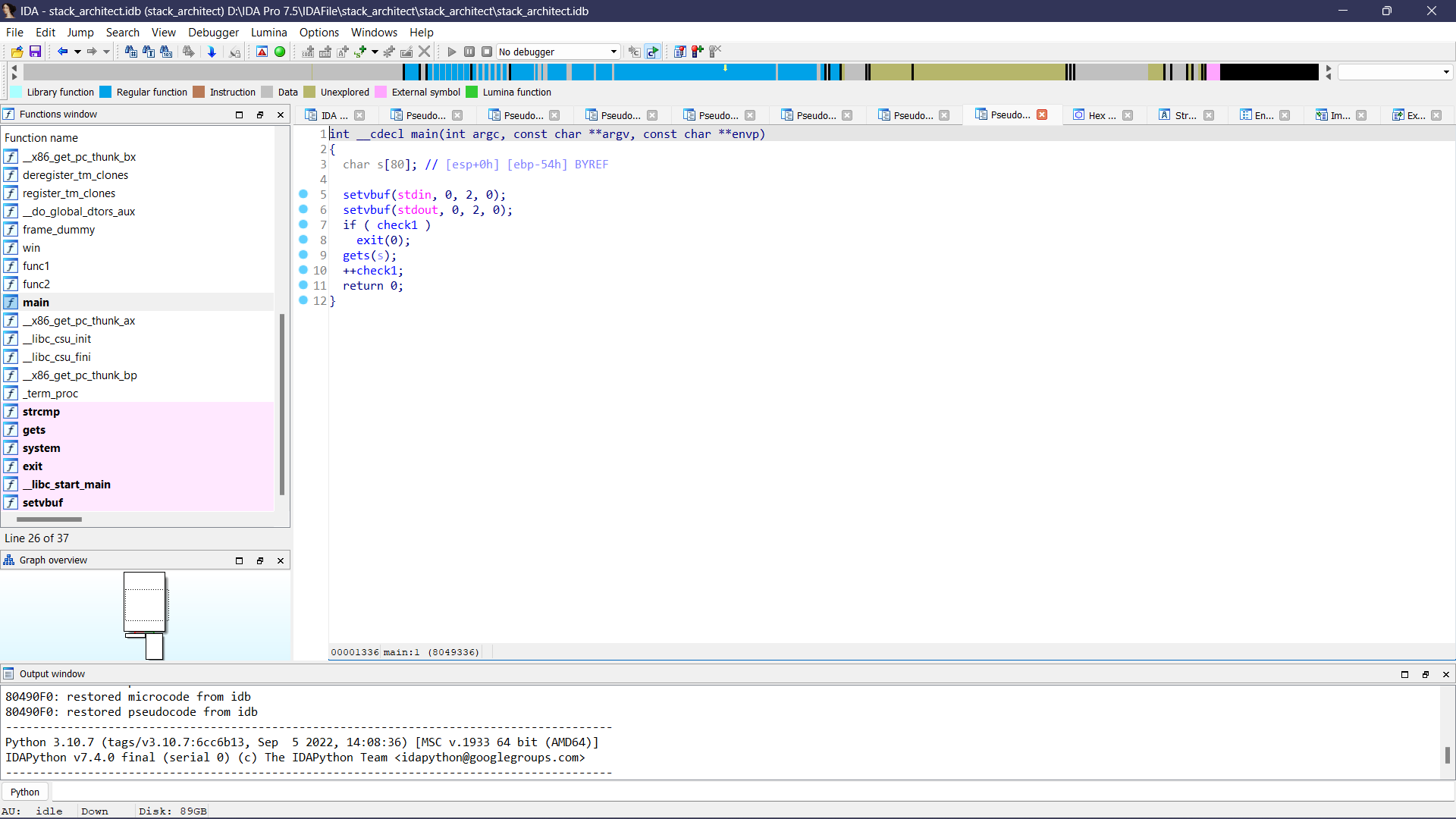
flag.txt: W1{neu\_ban\_chinh\_phuc\_duoc\_chinh\_minh\_ban\_co\_the\_chinh\_phuc\_duoc\_the\_gioi}

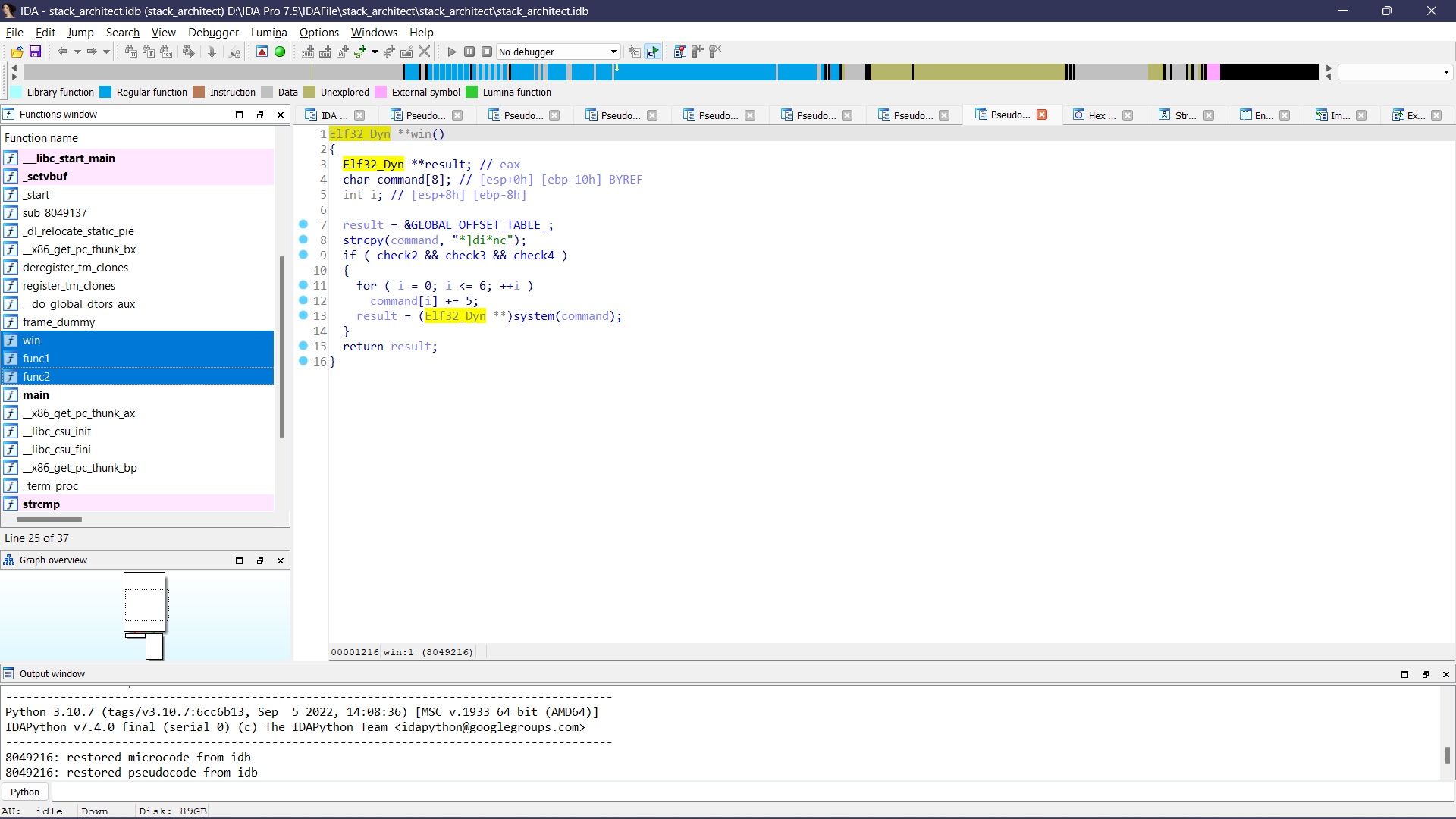
Đầu tiên, ta chạy gdb đối với file stack\_architect.

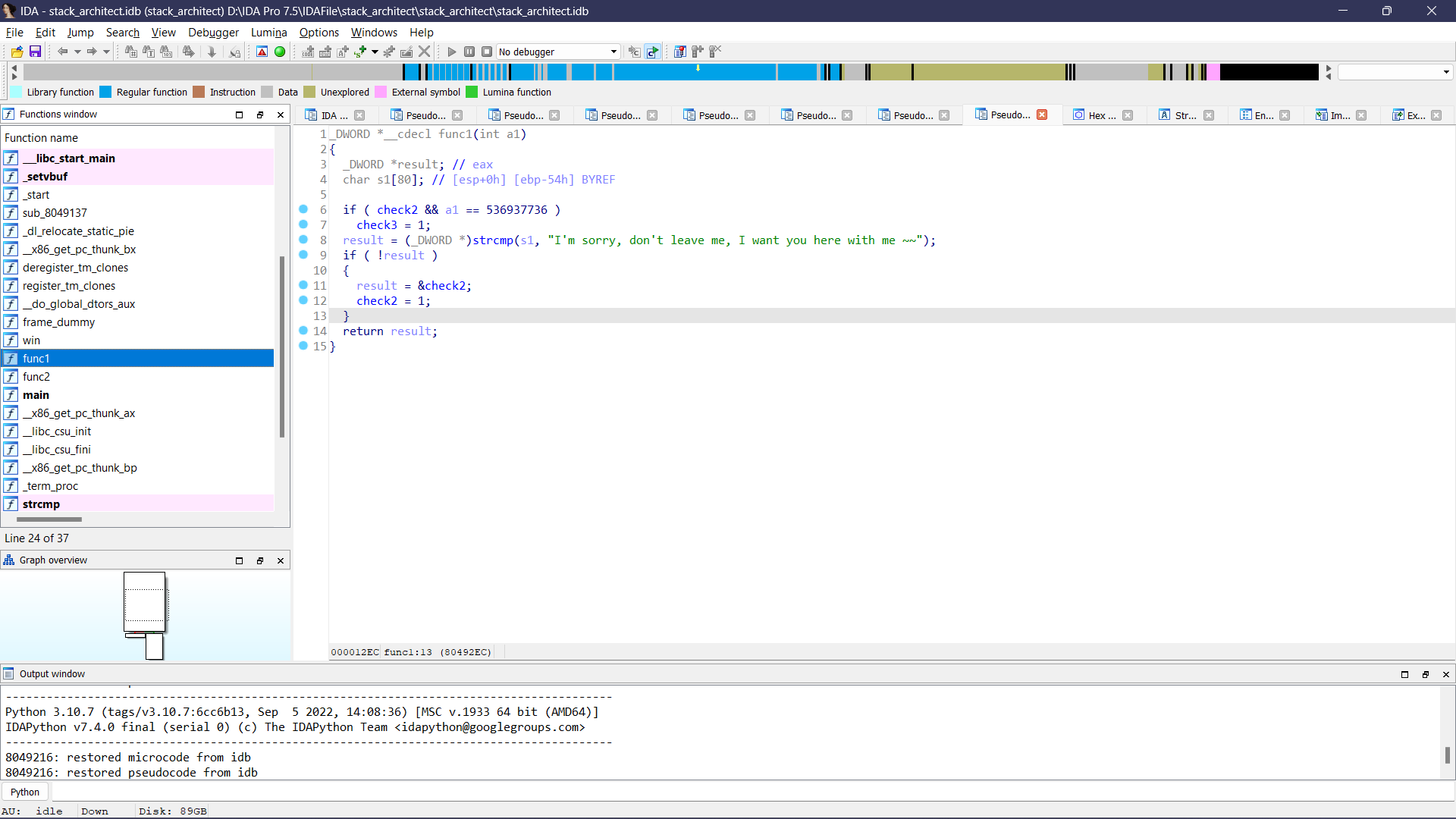
Khi này, ta có thể xem được checksec của file. Trong checksec có bật NX, vì vậy sẽ không thể truyền shellcode vào chạy được.

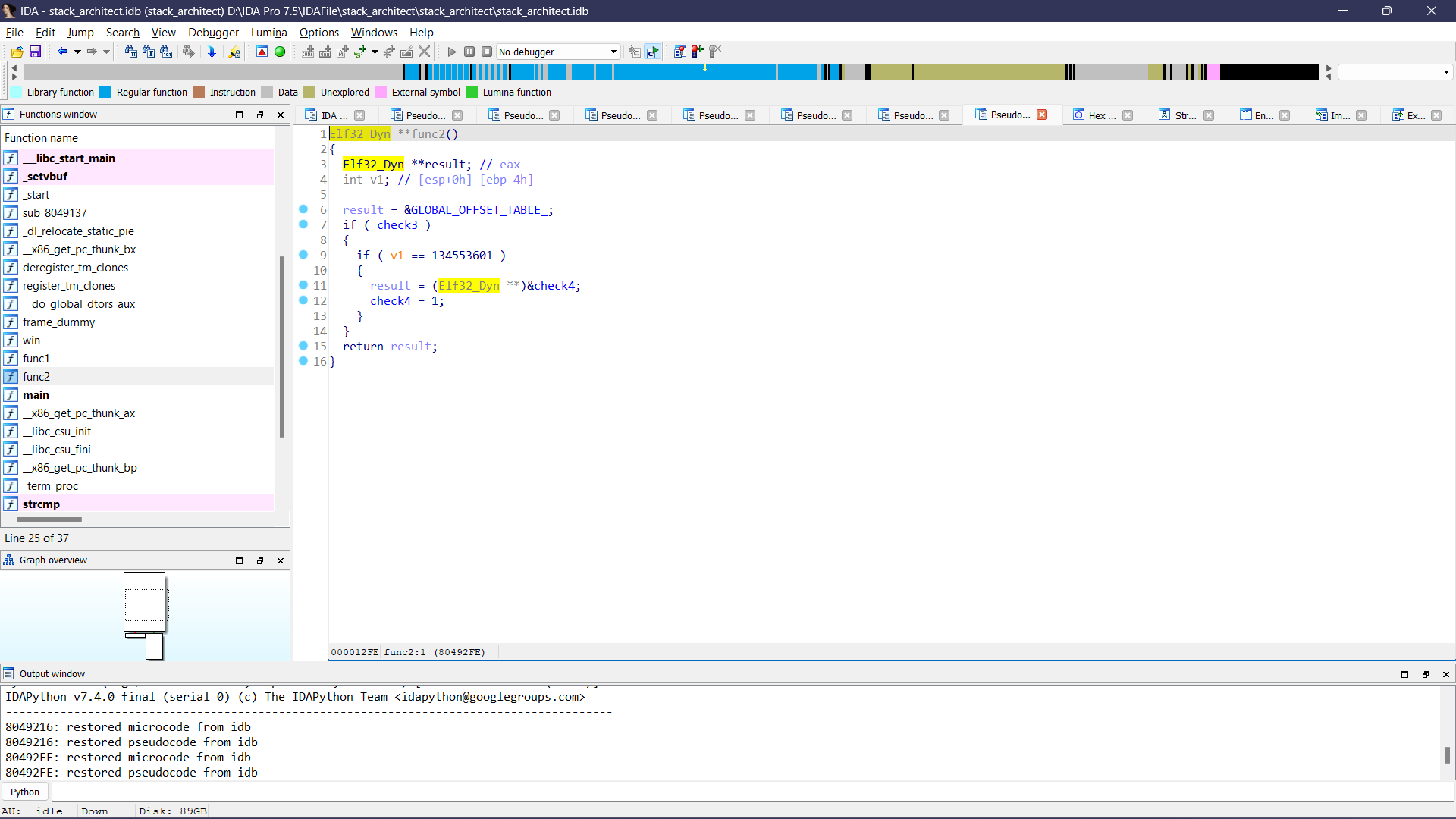


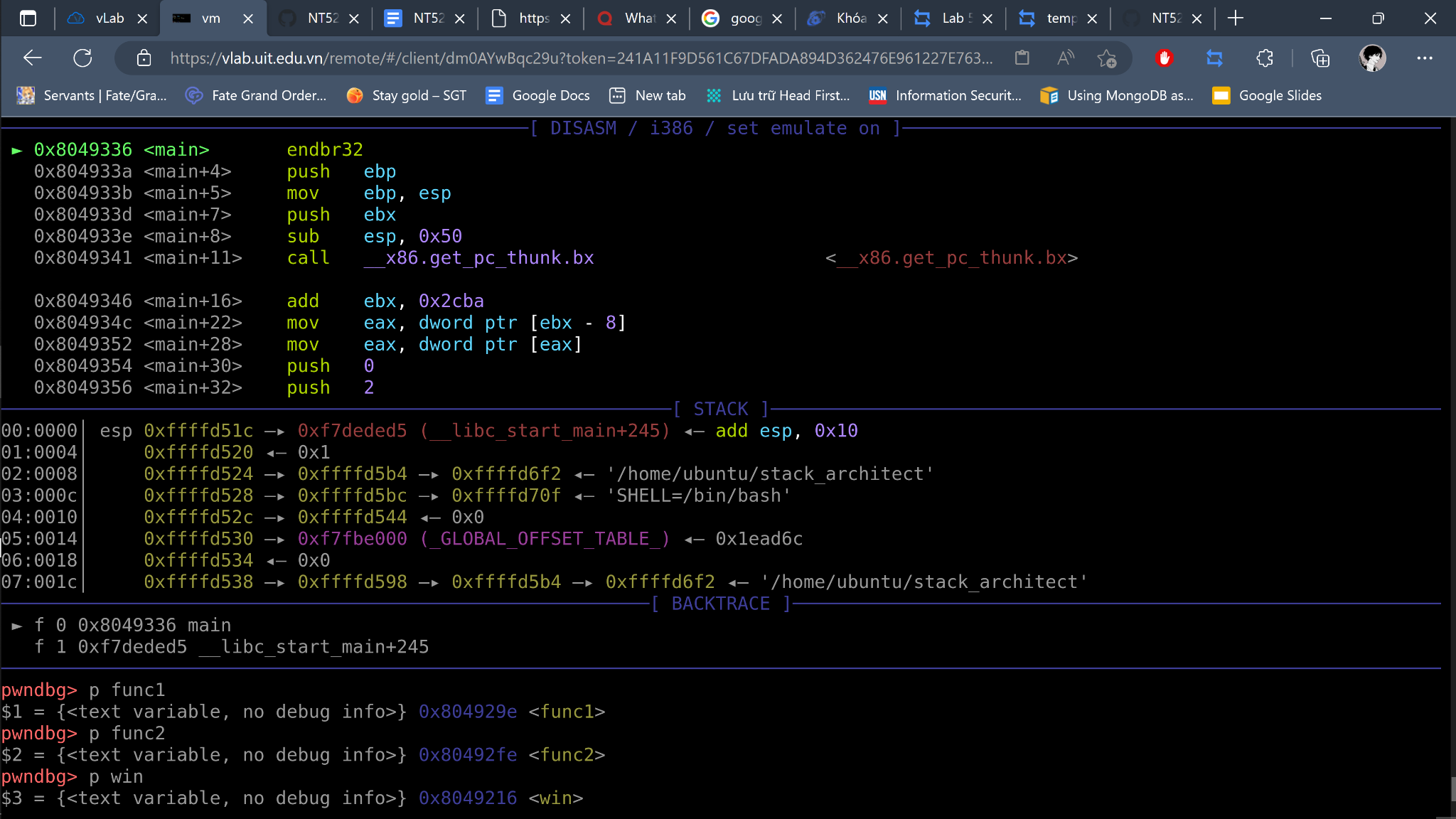
Theo ý tưởng trước đó, ta sẽ cần tìm địa chỉ của các hàm: func1, func2 và win trong file compile để thực hiện việc exploit.



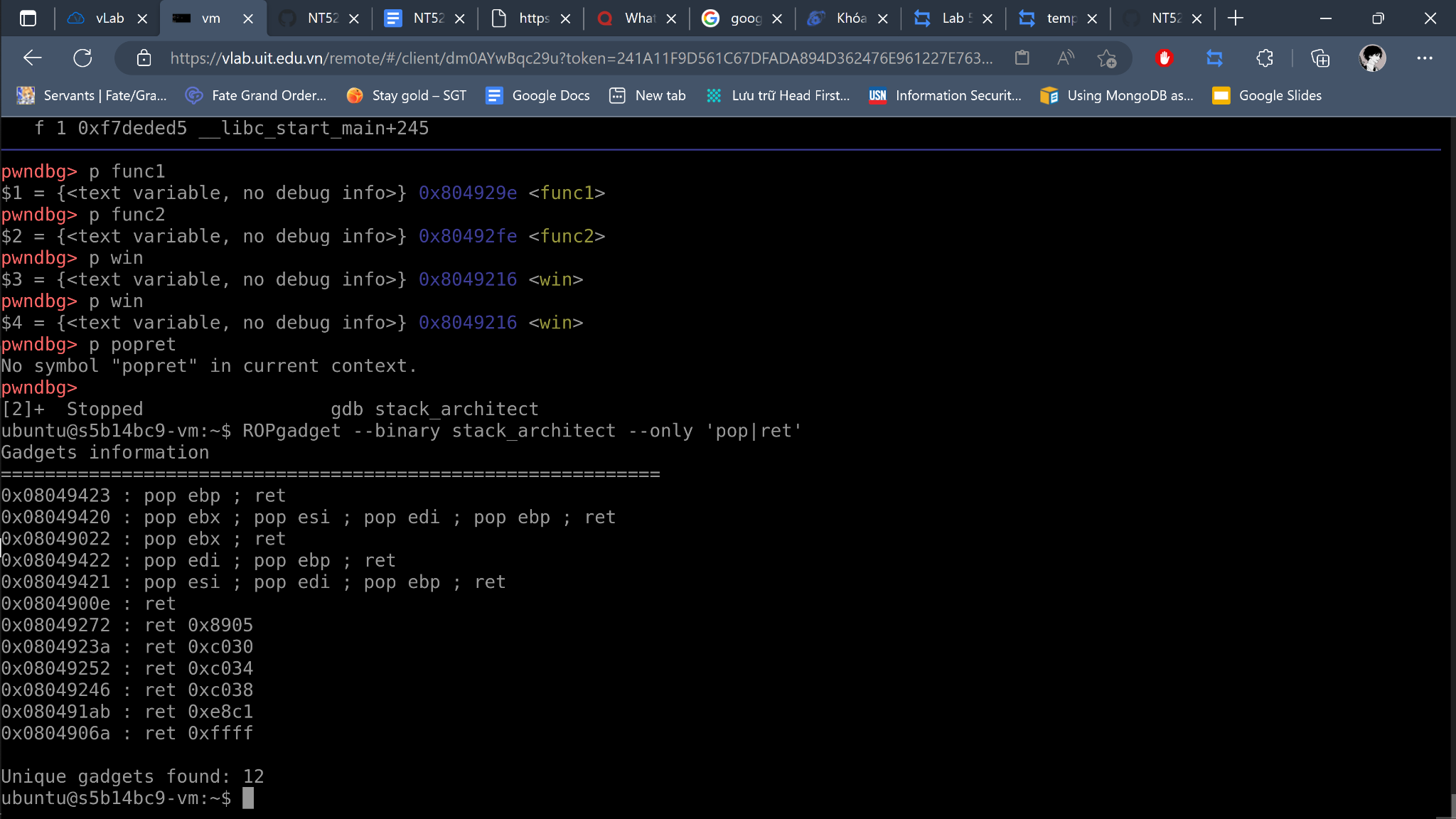


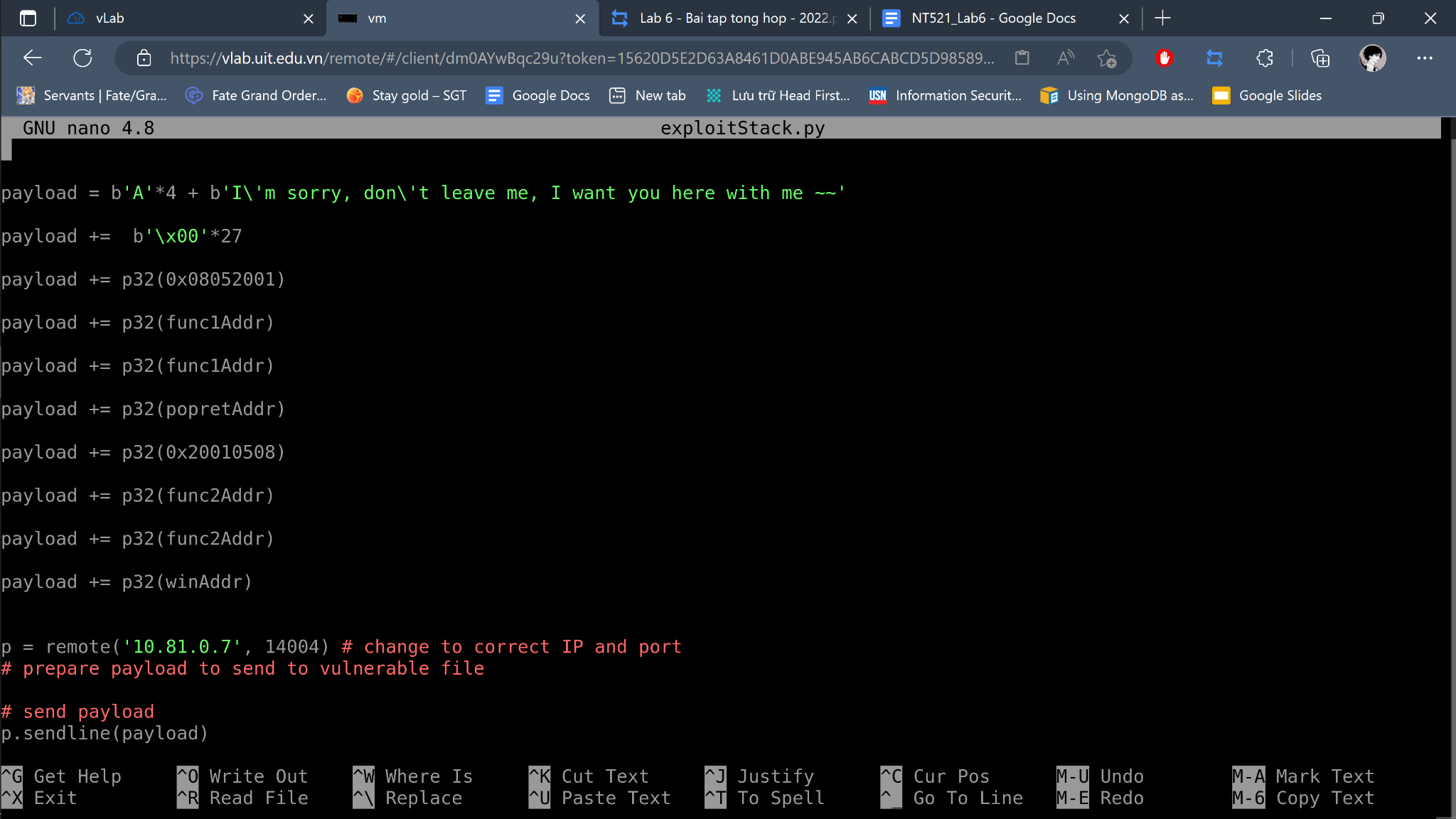






Tiếp theo, ta cần tìm tiếp địa chỉ của pop;ret. Vì vậy ta sẽ cần dùng đến ROPgadget với command bên dưới.

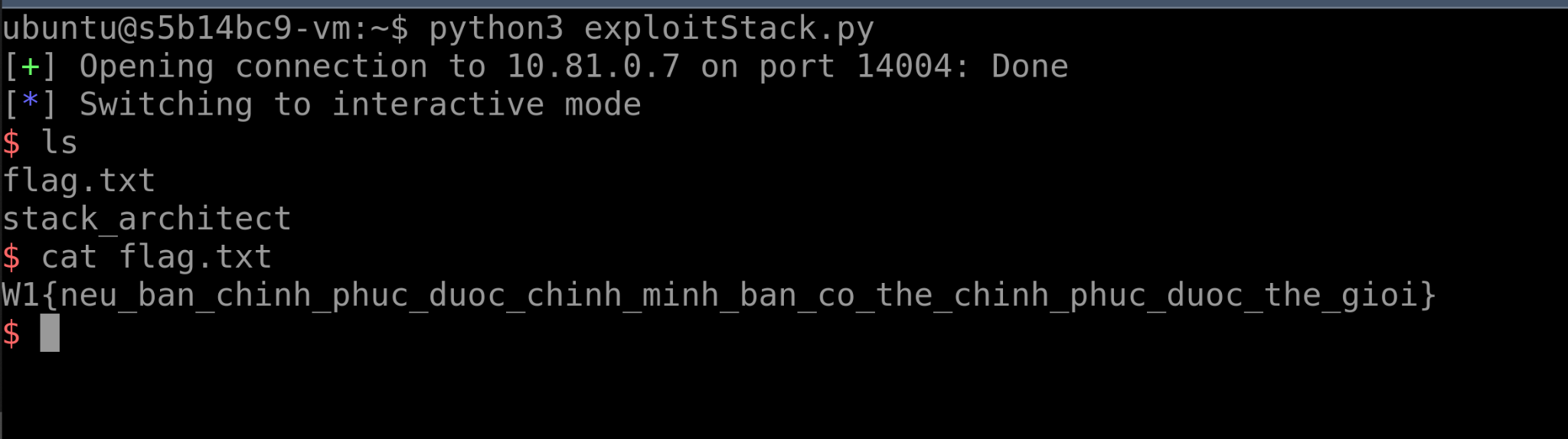
Sau khi có đủ các địa chỉ cần thiết, ta tạo file exploit để chuẩn bị và truyền payload đến máy vul.



Như vậy stack sau khi bị ghi đè sẽ có nội dung như sau

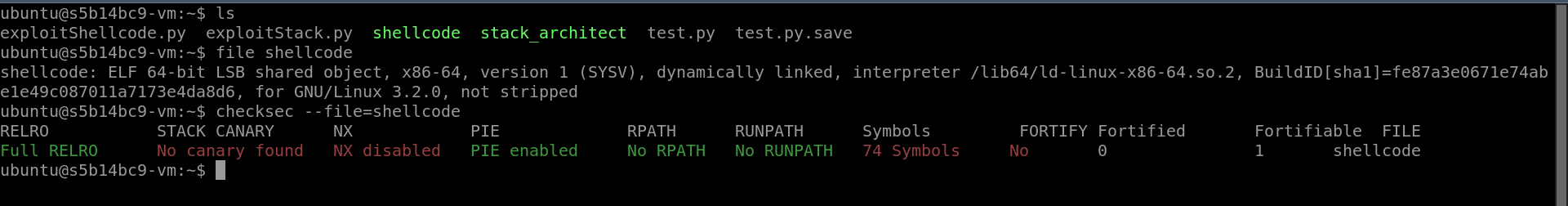
|  |
| --- |
| Địa chỉ win |
| Địa chỉ func2 |
| Địa chỉ func2 |
| 0x20010508 |
| Địa chỉ popret |
| Địa chỉ func1 |
| Địa chỉ func1 |
| Ebp (0x08052001) |
| \x00  …  \x00 |
| ‘I'm sorry, don't leave me, I want you here with me ~~’ |
| AAAA |

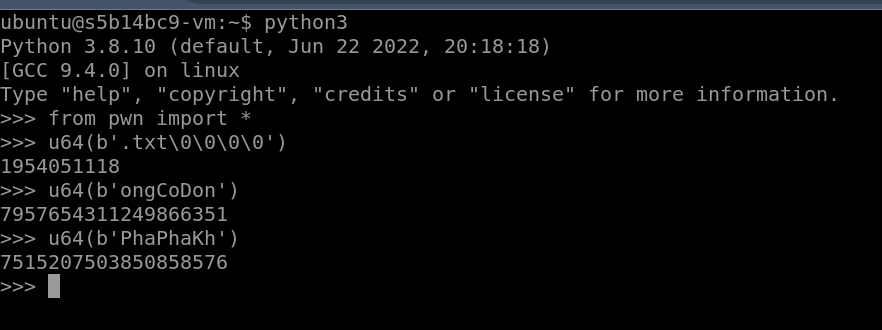
Sau khi chạy file exploit, ta sẽ lấy được flag.

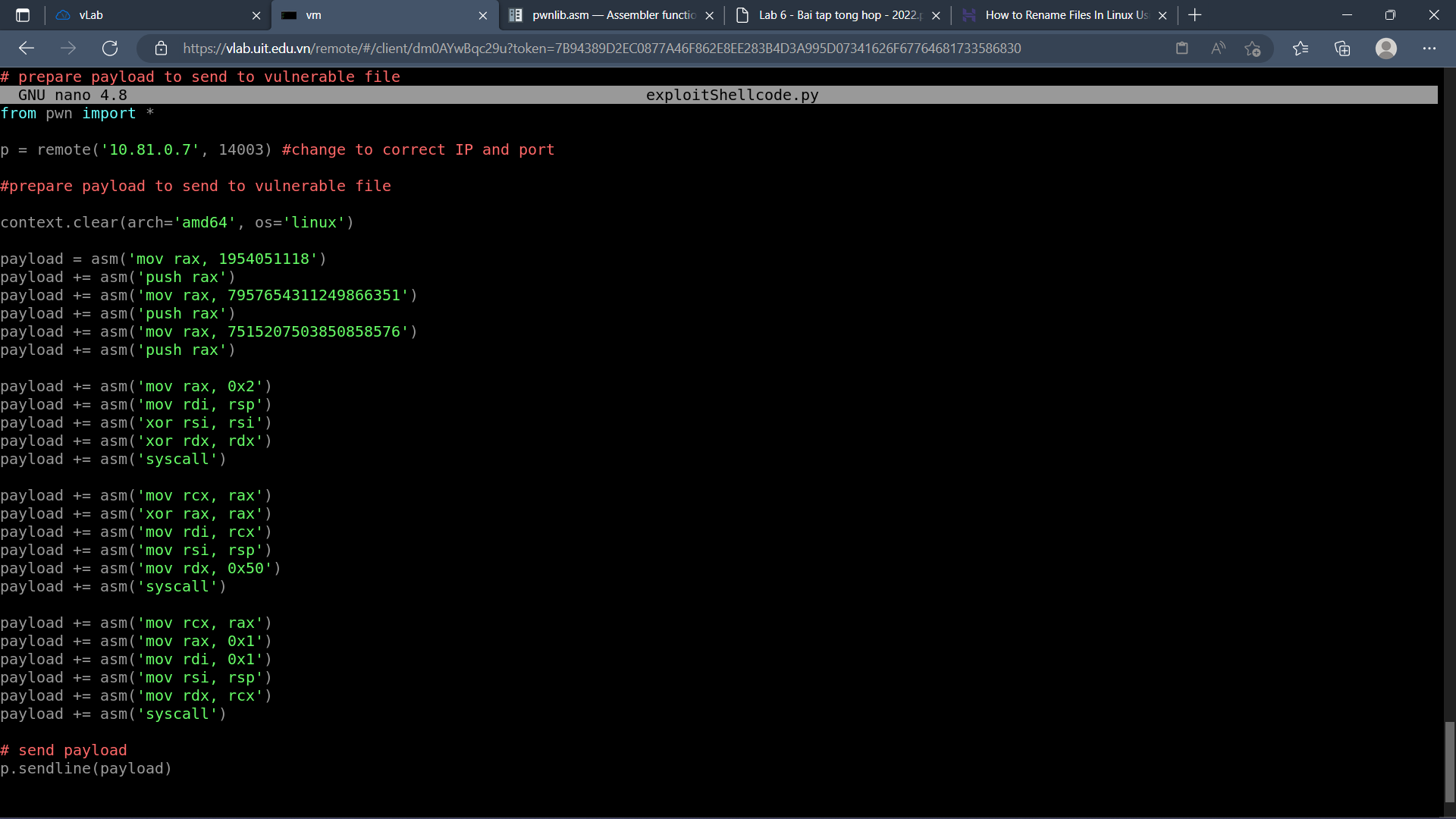


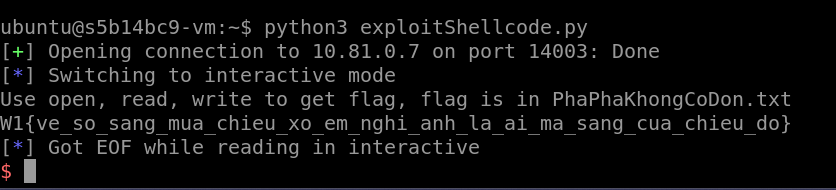
**Shellcode**

Đầu tiên ta sẽ thực hiện kiểm tra bằng gdc ta thấy một số trường không được kích hoạt, điều đó ta có thể thấy rằng là ta có thể khai thác bằng shellcode với các phương thức read, write và open





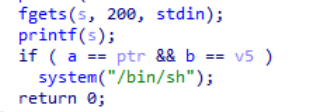




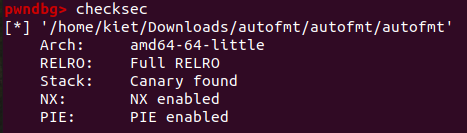
**AutoFmt**

W1{do\_cac\_ban\_tren\_the\_gian\_nay\_khoang\_cach\_nao\_la\_xa\_nhat}

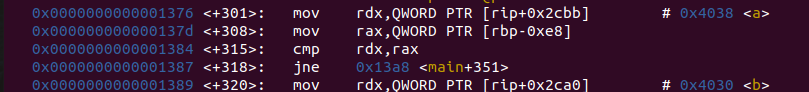
Đầu tiên sử dụng ida để xem, và check hàm main ta thấy được system đang gọi đến bin/sh, như vậy ta cần thực hiện overwrite a và b để có thể gọi được shell



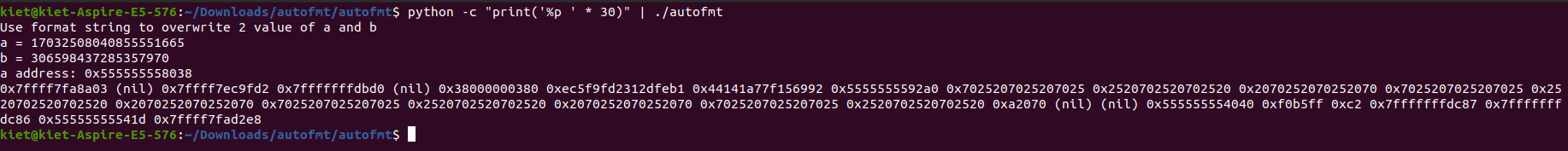
Check security thì thấy được là tất cả các yếu tố đều được bật lên



Ở đây ta thấy đươc rằng là việc đia chỉ của a đang cách địa chỉ của b 0x8



Ta thấy được là ở vị trí thứ 10 có sự thay đổi với dòng code python ta vừa thực hiện nhập



Tại vị trí 10 đó chính là phần in ra của %p \* 30 ta vừa nhập

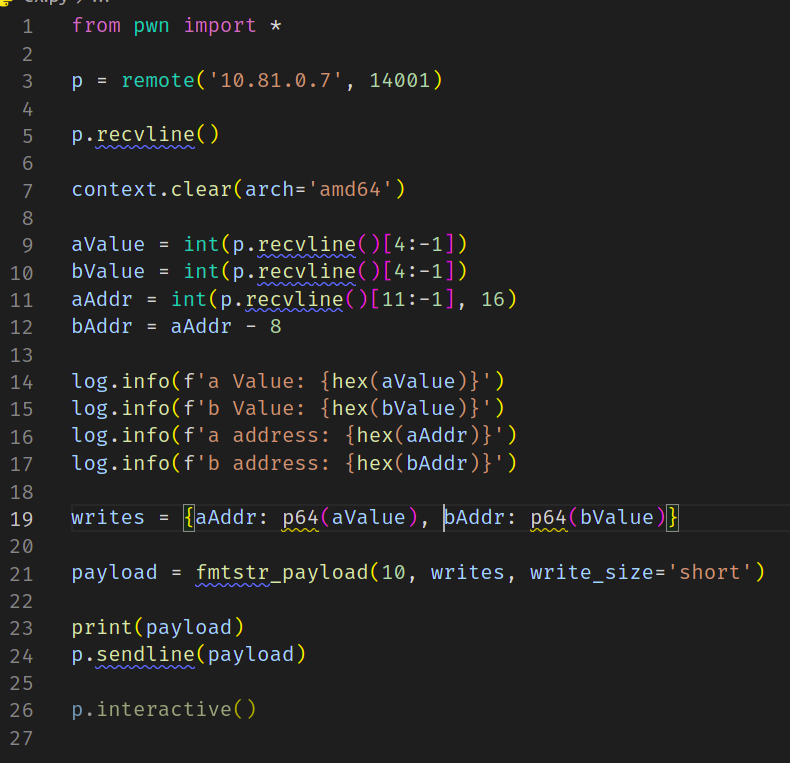


Sau đó, ta sẽ thực hiện viết code khai thác như sau

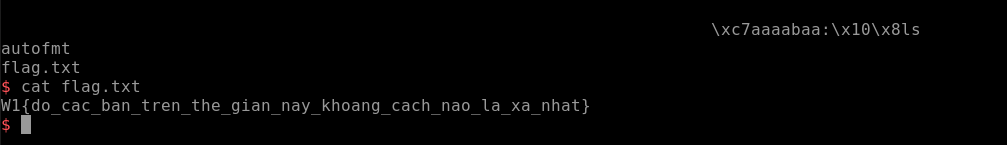
Việc thực hiện code này ta sẽ thực hiện lấy địa chỉ và giá trị của a và b khi chạy code thực thi, sau đó ta sẽ ghi vào biến tạm ở dạng payload 64 theo dạng: “Địa chỉ: giá trị tương ứng”.

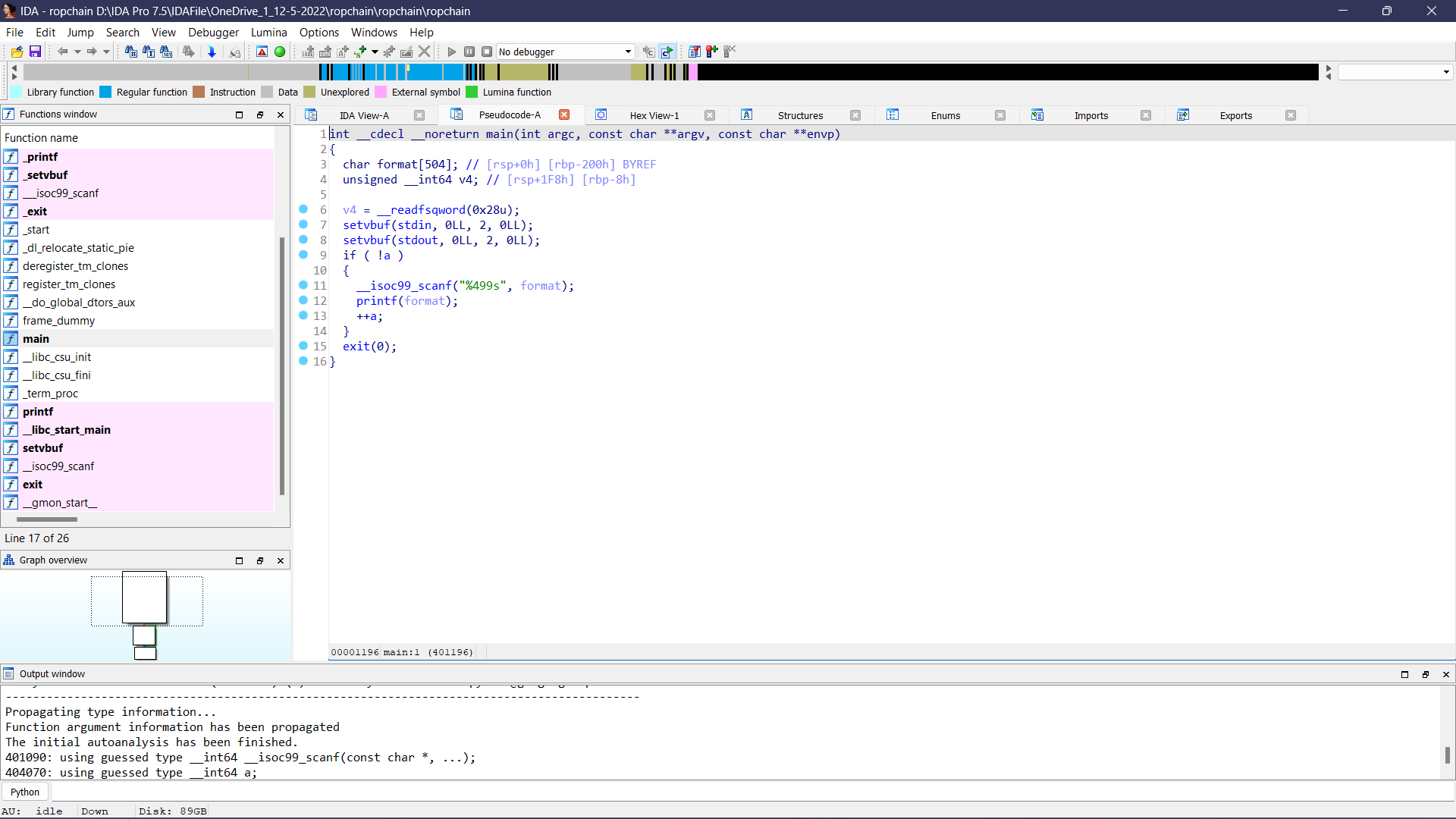
Sau đó ta thực hiện tạo payload ở với hàm fmtstr\_payload, với hàm này ta sẽ chọn được vị trí đè payload, tiếp tục với biến tạm ở dạng payload 64 ta vừa tạo ở trên và với kích thước là short.

Cuối cùng ta thực hiện truyền payload để gọi shell



Cuối cùng ta thực hiện trên shell của máy vlab và ta được flag





(Đạt ANTN.2020)

**Ropchain**

* **Phân tích code**

**Text

Description automatically generated**

* Chương trình kiểm tra giá trị của biến **global a**, nếu a = 0 sẽ đọc input từ người dùng và lưu vào biến **format**
* Chương trình sẽ gọi **printf(format)** 🡪 Lỗi Format string
* Tăng biến a thêm 1 và gọi hàm **exit**
* Ý tưởng bài này như sau
  + Payload1: Dùng format string để leak địa chỉ libc và overwrite **exit@got** thành địa chỉ hàm main 🡪 mỗi khi chương trình gọi exit sẽ quay lại hàm main 🡪 có vòng lặp vô tận, ngoài ra ta cần ghi đè giá trị của **a** thành **-1** để thỏa điều kiện cho phép người dùng nhập input
  + Payload2: Dùng format string để ghi đè **printf@got** thành địa chỉ hàm **system** trong libc (ở lần lặp kế tiếp chương trình sẽ gọi **system(format)** thay vì **printf(format)** ), ghi đè giá trị của **a** thành **-1**
  + Payload3: truyền vào chương trình chuỗi **“/bin/sh\x00”** 🡪 chương trình gọi **system(“/bin/sh\x00”)** và ta sẽ có shell để đọc flag
* **Chuẩn bị khai thác**
* Sử dụng **checksec** để kiểm tra, ta thấy chương trình chỉ có **Partial RELRO** và **NO PIE**, tức là ta dễ dàng có được các địa chỉ cần tìm cũng như ghi đè được **exit@got** / **printf@got**

**Text

Description automatically generated**

* Debug trong gdb, ta thấy input người dùng nhập vào sẽ nằm tại quadword đầu tiên trên stack

**Text

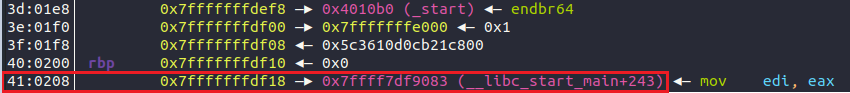
Description automatically generated**

* Do đó, để trỏ tới input, ta sẽ sử dụng **%(6+0)$** = **%6$\_**
* Để tại payload, ta sẽ dùng lại hàm **make\_payload** của **Autofmt**
* **Khai thác**
* **Payload1**
* Tạo payload 1 với tham số là dictionary chứa key là địa chỉ biến **a**, **exit@got** và value là giá trị **0xffffffffffffffff = -1,** địa chỉ hàm main (**0x401196**)
* Do giá trị tại **exit@got** và địa chỉ hàm **main** chỉ khác nhau 2 bytes đầu tiên, ta chỉ cần ghi 2 bytes vào exit@got

**Text

Description automatically generated**

* Để có được địa chỉ hàm system, ta cần phải leak một địa chỉ libc trên stack
* Trên stack, quan sát thấy tại quadword thứ 65 có một địa chỉ thuộc libc **(\_\_libc\_start\_main)**

****

* Vậy ta sẽ dùng **%(0x41+6)$p** = **%71$p** để đọc được giá trị **libc** này

**Text

Description automatically generated**

**Hàm make\_payload có chức năng leak giá trị trên stack**

* Ta sẽ đọc giá trị libc được leak ra và cập nhật địa chỉ hàm system tương ứng

**Text

Description automatically generated**

* **Payload2**
* Tạo payload 2 với tham số là dictionary chứa key là địa chỉ biến a, printf@got và value là giá trị 0xffffffffffffffff (-1), địa chỉ hàm system vừa tìm được

**Calendar

Description automatically generated**

**Tạo payload2**

* **Payload3**
* Sau khi overwrite printf@got thành địa chỉ hàm system, ta chỉ cần gửi **“/bin/sh\x00”** sẽ có được shell

**Text

Description automatically generated**

**Text

Description automatically generated**

**Kết quả khai thác**