Vul-Chat 2.0 -100

Writeup by JinZhao(Jin) Su

Problem

Djinn has got some new intel for us. And I think he's giving us a second chance. But he will only speak with you. Let's see what he's got to say.

nc vulnchat2.tuctf.com 4242

hint:

vuln-chat2.0 - md5: d0fe412783aaaeeddc2a6504d6dff631

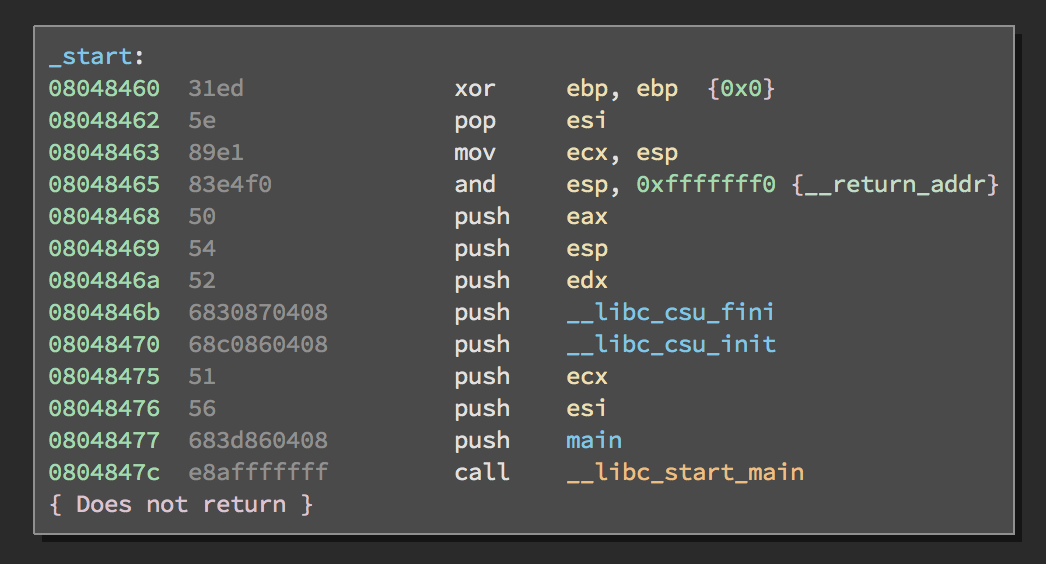
libc.so.6 - md5: f71e966c590bfb1f53ea3410f8b489d4

Answer:

**Overview:**

To solve this problem, you will have to understand where the input values are stored and at what address the return will return.

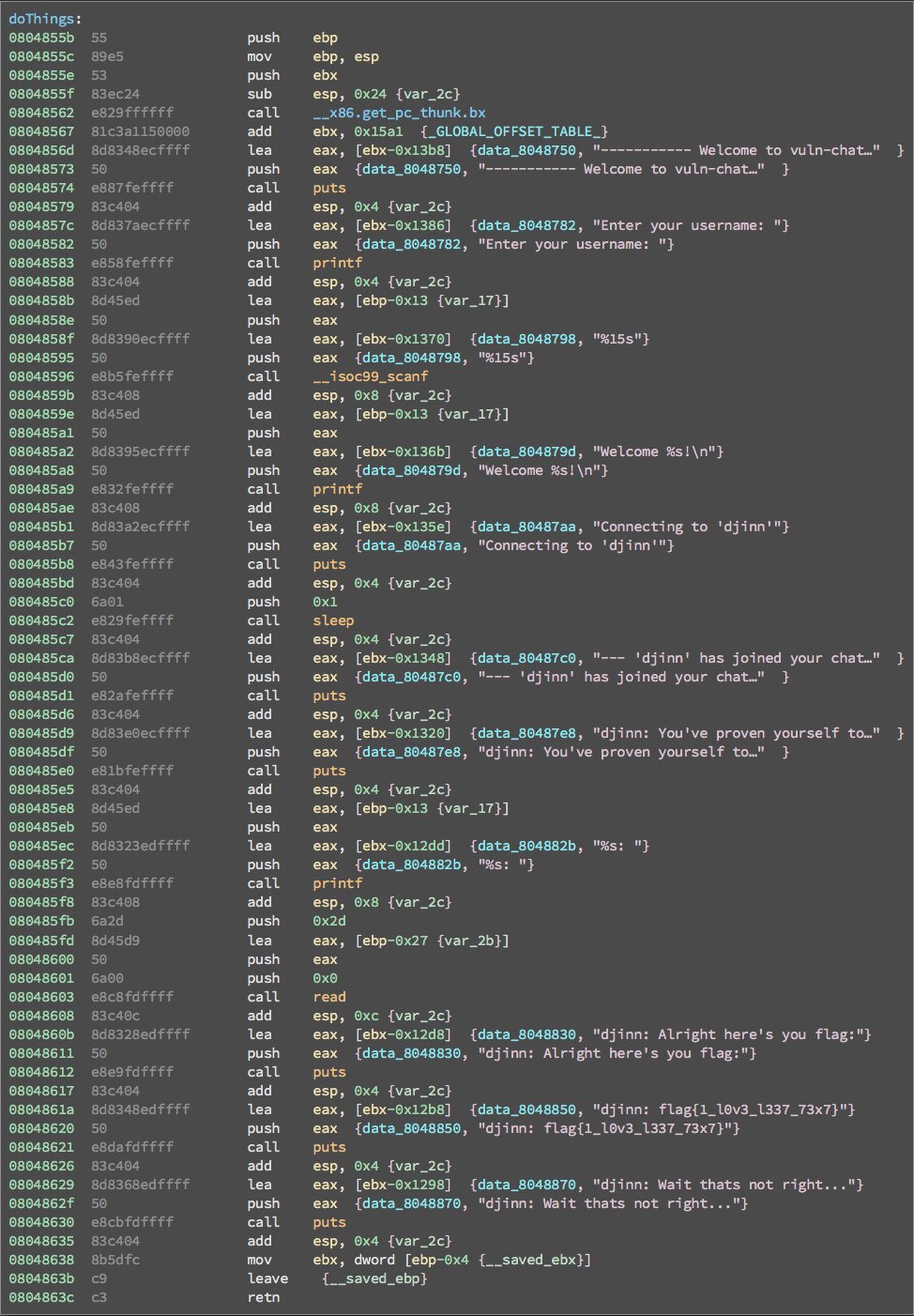
**Details:**

The program is a simple two input function. The first input is for entering the username and the second is for “What information do you need?” The first thing that we needed to do was open the program in Binary Ninja. 

Nothing unusual can be seen here. Let us enter main!

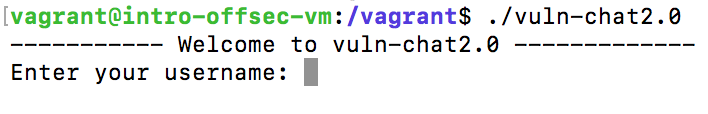


Interesting. We see setvbuf, so potentially it could be a buffer overflow. However, we should not rush to a conclusion. Let us enter doThings!



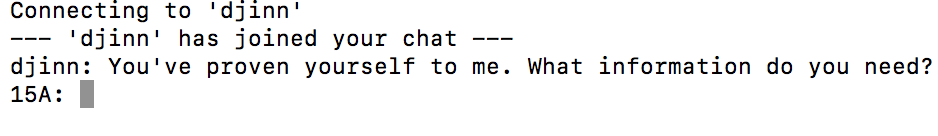
The whole function just allows the user to input their desired username and their answer to “What information do you need?” However, on address 0804861a there is a flag. If you try putting that flag into the flag submitter box, you’ll get rick rolled so badly that you’ll ask the admin what happen. Only to find out that you got rick rolled, which I swear I didn’t do.

Since the solution was not part of the doThings, we should try running the program in GDB.



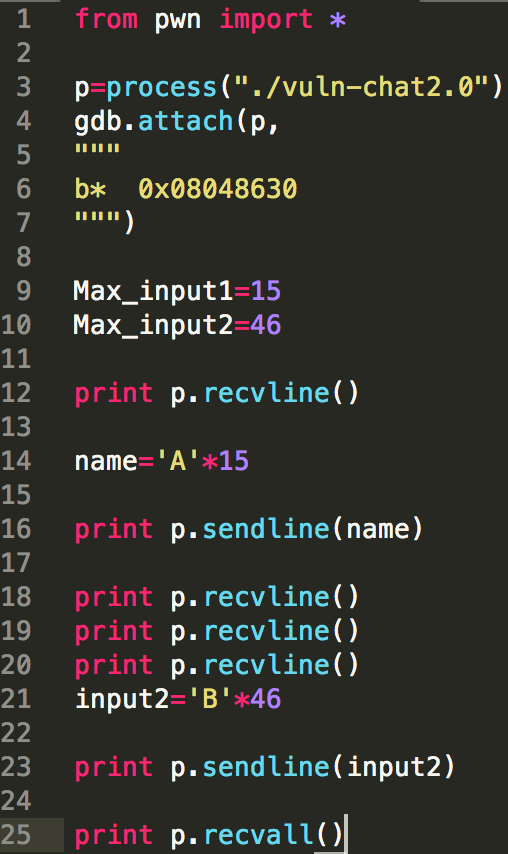
Since there are potential inputs, there is always the possibility that this is a ROP problem.

So, to be concise I found out that the maximum input for username is 15 characters.



Next is the second input that is supposed to answer, “What information do you need?” Again, I was being concise and found out that the maximum input for this input was 46 characters.

To begin the test, I made a simple script that gives the first input 15 A’s, the second input 15 B’s, and the breakpoint to be 0x08048630. It’s best to be a few lines before the return so you can see what’s happening.



Prints out the data

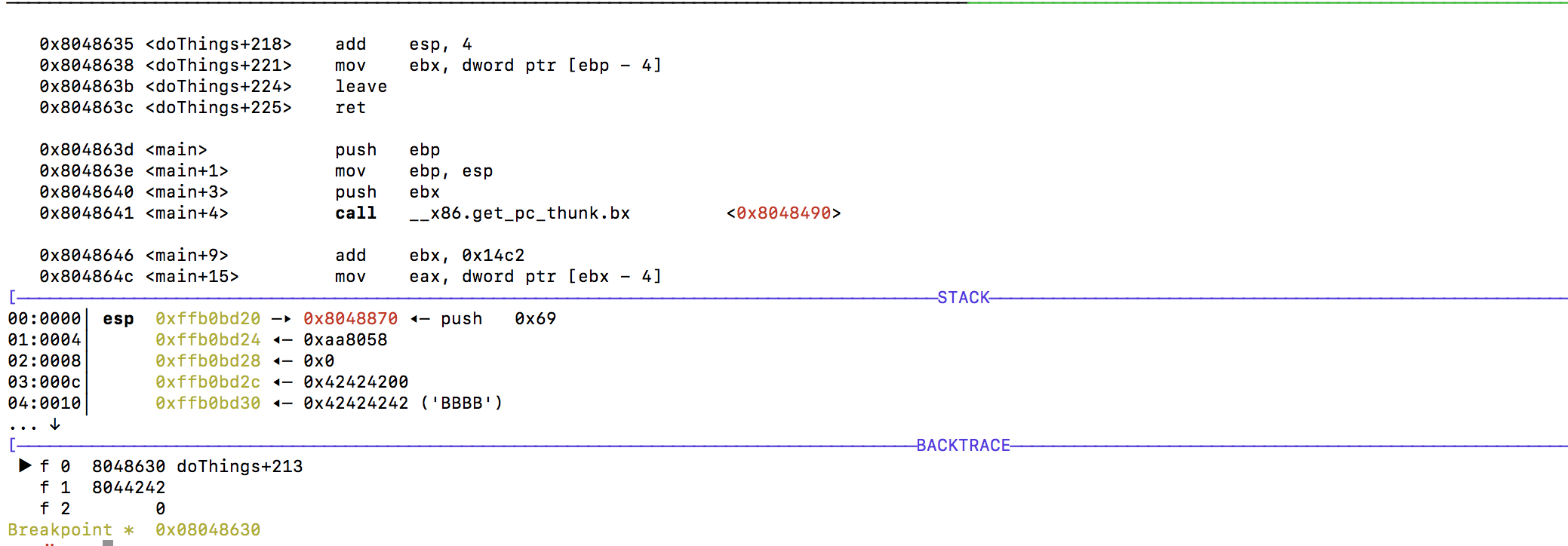
Attach to GDB with a breakpoint

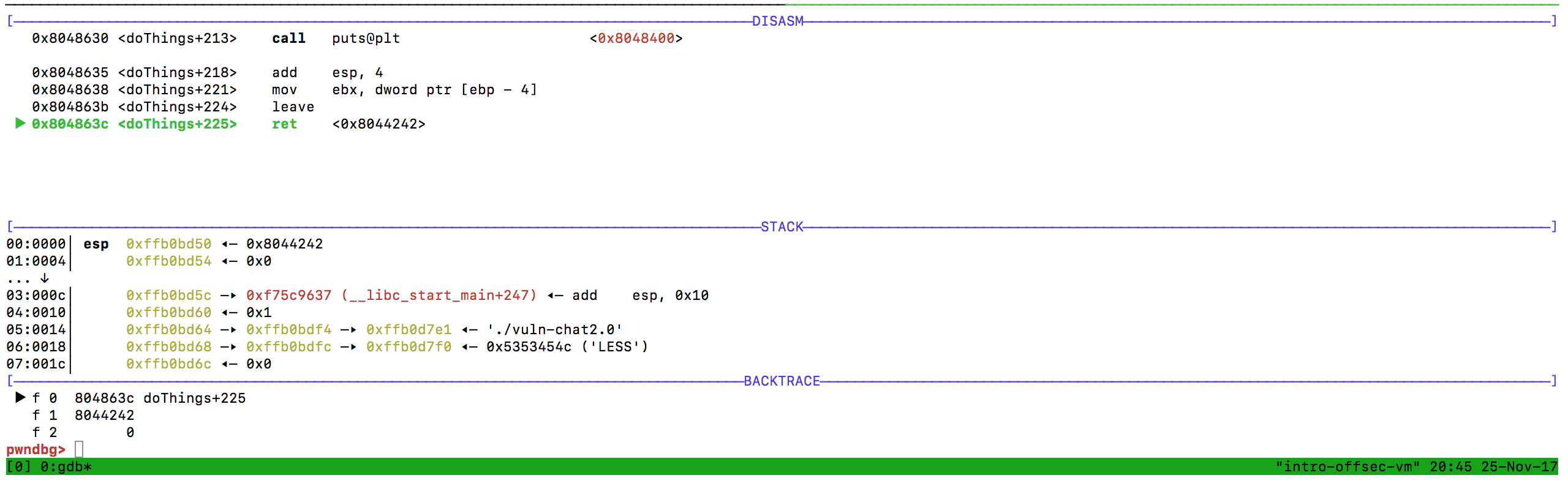
Sends input two

Sends input one

Begins the process

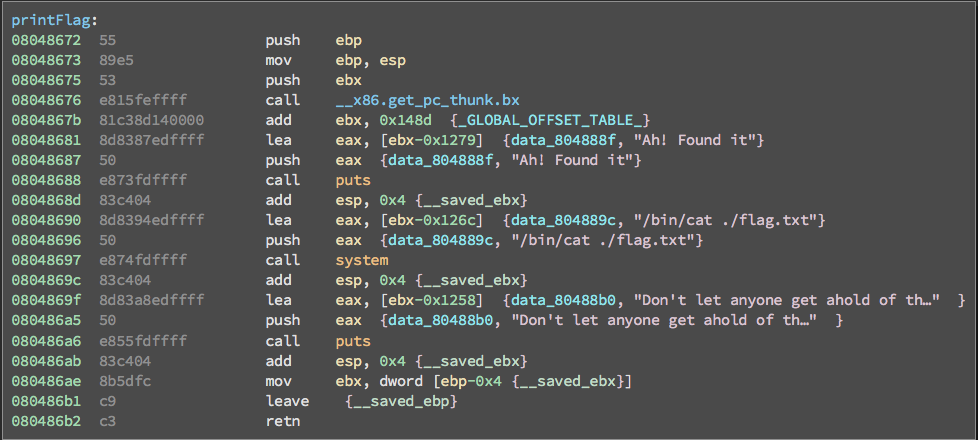
Once it ran GDB had a break point at 0x08048630. Below the import break points of GDB.





Interestingly the return address is at 0x08044242. From the picture above this, the data store starting at 0xffb0d2c starts to store the 46 B’s and has over ridden the last 2 bytes of the return address. I thought that if we increased the input of B’s then it would be possible to override the return address. However, I was mistaken. I tried to input 100 B’s and the result was the same.

Now back tracking to the Binary Ninja, I recognized something very interesting.



What’s this? A print flags? I wonder what it does? Wait a minute. The first few bytes that can’t be overridden is 0x0804…., which matches with the return. Mhmm. I wonder what it could be~

The script was then updated so that the second input would send in 0x00000008048672 \* 23 times since a block of memory is 4 characters long. However, when the script ran, it was mismatched with 0x08047200. So now the issue is with the alignment. Knowing that I decided to rearrange the address sent so that the return would be 0x00000008048672.



I ran the script and the return address was 0x08048672! Pop came the flag: TUCTF{0n3\_by73\_15\_4ll\_y0u\_n33d}

