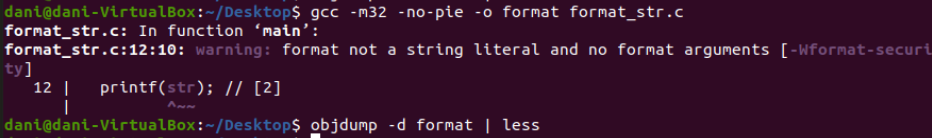
**Problem 1**

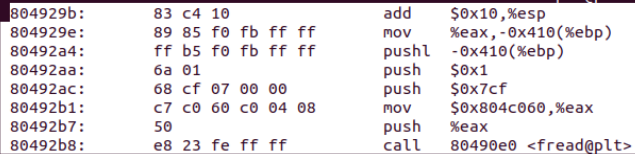
**1**

Here is how I compiled the program:

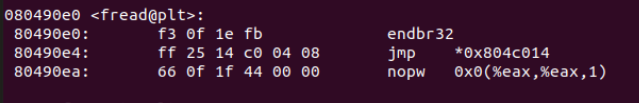


**2**

Here is the part that is responsible for passing in arguments:



Here is the disassembled code for fread:



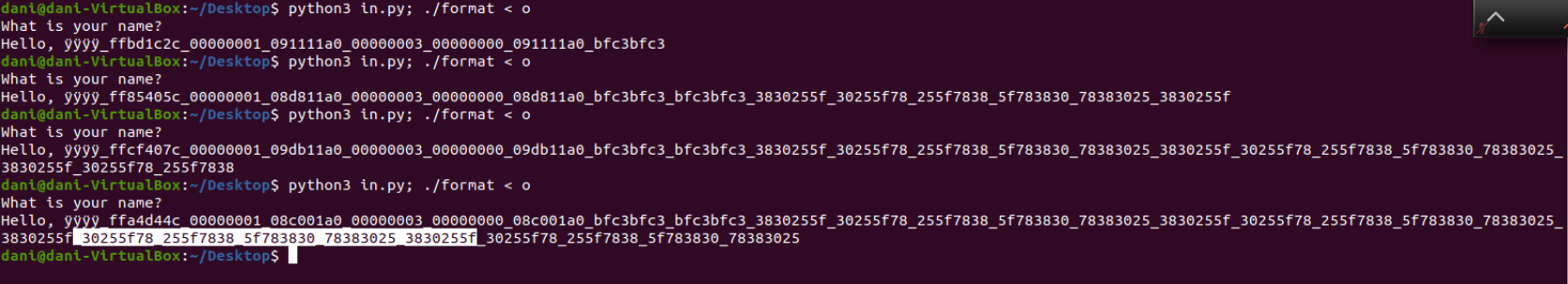
So as seen in the first screenshot, we see that the arguments get passed into the function from right to left. So the first pushl is used to pass in the memory address of where the file that pointer f is referring too. The next push 0x1 passes in the argument 1 to the function fread and similarly push 0x7cf pushes 1999 to fread. We see that the value being moved into eax and then subsequently being pushed to the stack will have to be the memory address of passwd hence, we have the location in memory where passwd is located.

**3**

0x804c060 since that is the last argument passed into the function fread.

**4**

I added “\_%08x” until I began to see a pattern:



So the repeated string is:

\_30255f08\_255f7838\_5f783830\_78383025\_3830255f

FINISH THE LAST SENTENCE

**5**

We essentially just change the line printf(str); to printf(“%s”, str); this will elimate the vulnerability that an attacker is able to use the format string attack and hence just print out what is inputted rather than looking at any memory. The attack no longer works since we provide the formatting of the string so the only thing passed in to printf is a string rather a formatting of the string and what to print out eliminating the extra step that is taken when using the first printf.

**Problem 2**

**1**

Firstly, the attacker will now have the private keys of every user on the system and this in itself is already compromising the identity of any user on the system. Harm can be done here by the attacker getting into a user than may have admin privileges and exposing all parts of the server that should not be exposed. This also can cause the attacker to seem like another user when in reality they are just using a fake identity. Secondly, the attacker now has his own key in the authorized keys of the server meaning he will be able to access the server remotely and be able to view information that he would not otherwise be able to if he did not have access to the server to begin with. This is very harmful to the system in general since the attacker now has a way to enter the system and view very sensitive information that may be hidden to a typical user or the public.

**2**

As we seen by the first command the file id\_rsa is rw- for a user, meaning that a user than read and write into a file, which is dangerous since this is where all the private ssh keys are contained. A way to protect against this would be to set the permission of the first three bits to the root so that only a root user would be able to read or write the id\_rsa file. And any other user has no permission according to the permission bits to either read or write meaning this file is much more secure.

For the authorized\_keys file we see that a user has both read and write access based off the first 6 bits, however the last three bits show that the other group would only have read access to this file. In order to protect the system better I would change the groups to both root and root to allow only the root to read and write and therefore a regular user would only be able to read public keys and not be able to write or in the case above add a public key of the attacker. This would have to be done using the root user which would be much harder to obtain.

**3**