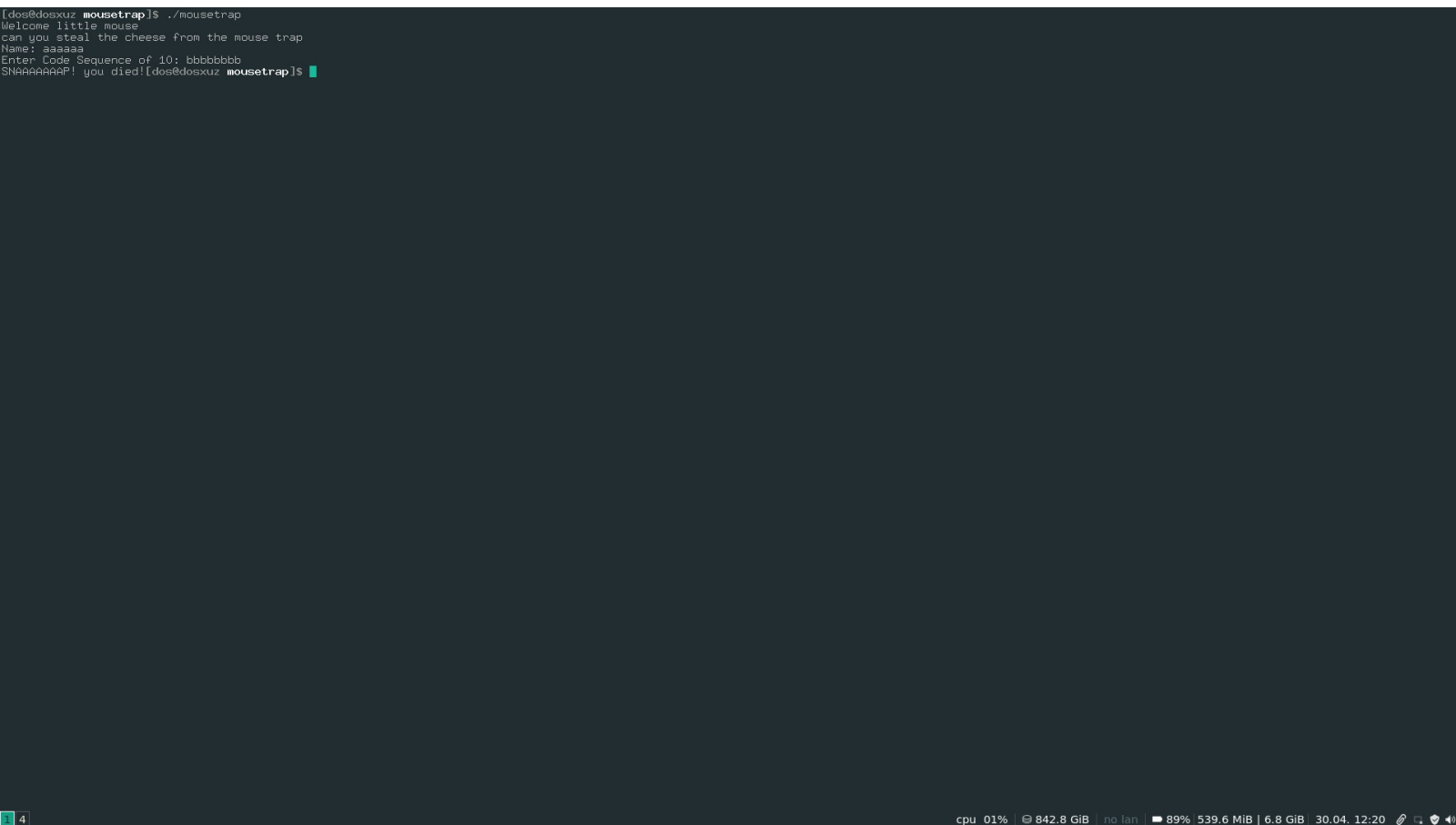


MOUSETRAP

ANALYSIS OF THE BINARY:

When we run the program, we get something as follows:

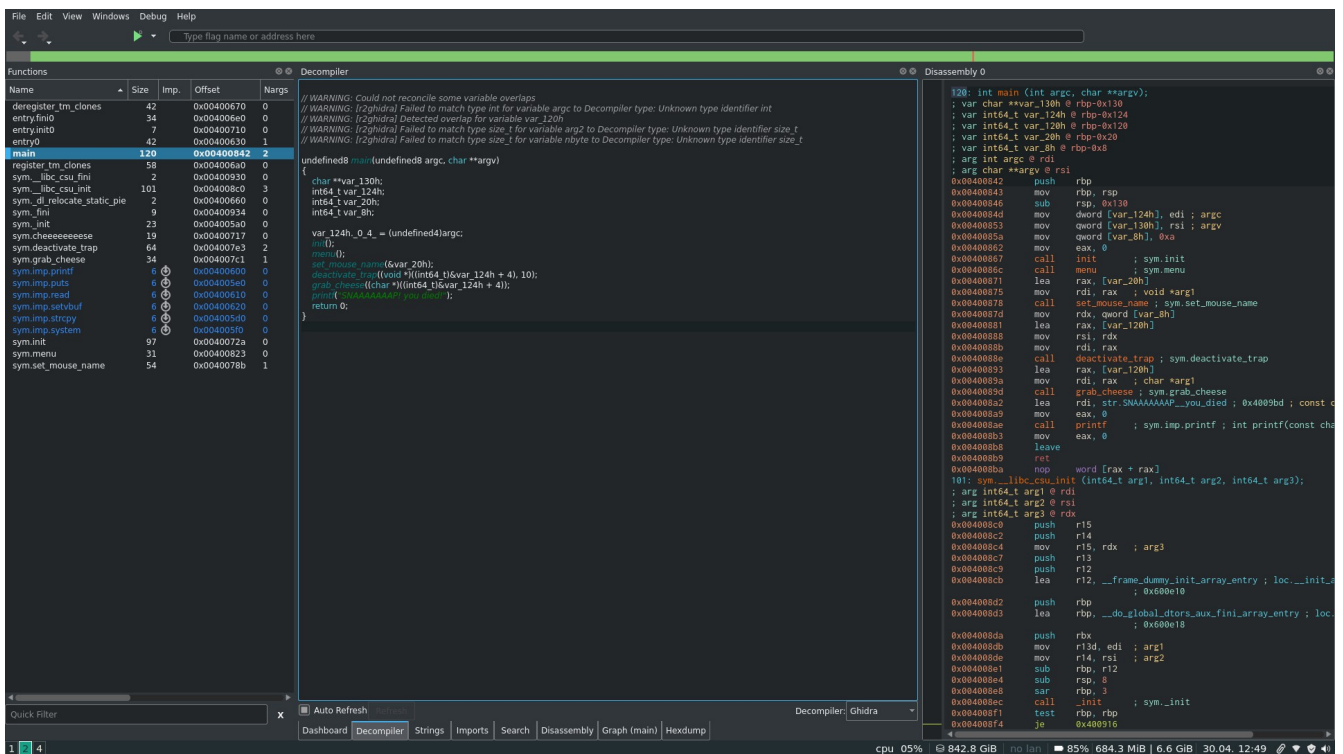
```
[dos@doshuz mousetrap]$ ./mousetrap
Welcome little mouse
can you steal the cheese from the mouse trap
Name: aaaaaa
Enter Code Sequence of 10: bbbbbbb
SNAAAAAAAP! you died[dos@doshuz mousetrap]$
```



```
Welcome little mouse
can you steal the cheese from the mouse trap
Name: aaaaaa
Enter Code Sequence of 10: bbbbbb
SNAAAAAAAP! you died
```

FINDING THE BUG:

So we decompile the binary in cutter in order to get a proper look at the binary:



After fidling with the binary and analysing the decompilation of the binary for sometime, we see that there is a possibility of a buffer overflow. If we input more than 24 characters, we find something as follows:

```
Welcome little mouse
can you steal the cheese from the mouse trap
Name: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Enter Code Sequence of 7016996765293437281: SNAAAAAAAP! you died![dos@dosxuz mousetrap]
$ aaaaaaaaaaaaaaaaaa
bash: aaaaaaaaaaaaaaaaa: command not found
```

If we convert the number to hex, we will find that the variable which was initially holding the value 10 has been overwritten with a's. So we enter exactly 25 a's in byte format, as given in the exploit.

Then in order to trigger a buffer overflow we enter more than 24 b's. This can be found out using a cyclic pattern or by trial and error method.

EXPLOITING THE BUFFER OVERFLOW :

After taking a closer look at all the functions we find that the function `deactive_trap` has a `strcpy()` function which is actually triggering the buffer overflow. Also, there is another function, `cheeeese()` which basically has a `sys_exec` for executing a shell. So our target is to perform a buffer overflow and redirect the code execution to the function `cheeeese()`.

For this we add the address of the function after adding 24 b's to the payload.

THE MOVAPS ERROR:

However, when I tried to use the above mentioned exploit, I was getting a segmentation fault. Upon tracing the error I found out that the error was due to a `movaps` instruction. Upon further looking into the problem I found out that it is due to stack misalignment and the stack must be aligned to 16-bytes when calling a function. For some reason the compiler assumed that the stack was aligned to 16-bytes and hence executed the function from the misaligned stack which basically caused the fault.

To know more about this error and getting deeper into it, refer to the following page:

<https://research.csiro.au/tsblog/debugging-stories-stack-alignment-matters/>

FINAL EXPLOIT:

In order to overcome the error, I used the instruction just before the `syscall`, that is the point where it is loading the string. It works fine like that.

I obtained the source code from the server. So you can also refer to that.