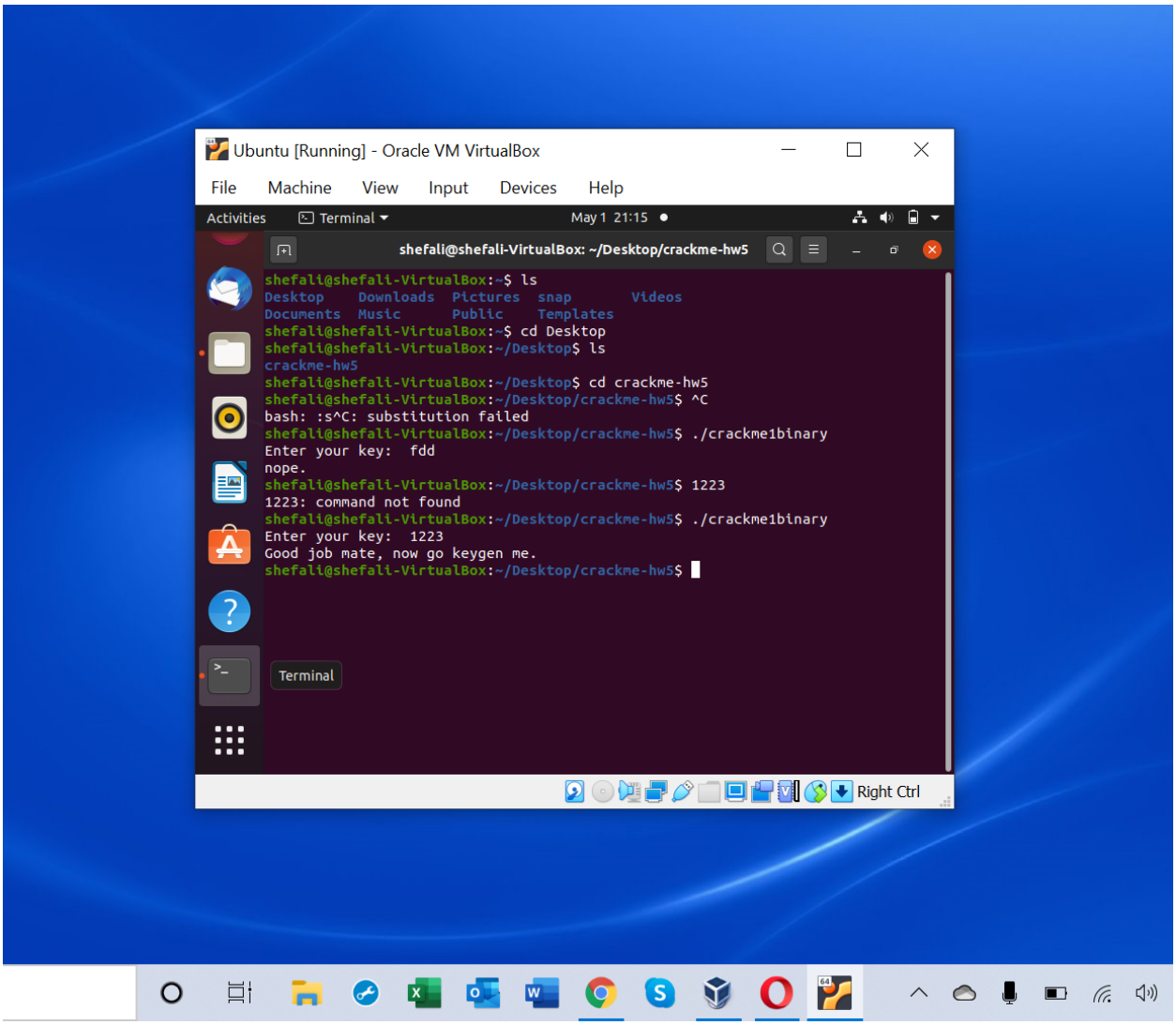
Crackme1binary Solution

- Installed Ghidra, JVM.
- Ghidra analysed the code “crackme1binary”.
- Focusing on the main function, the
    1. “**validate_key**” function was processing **local_14**
- In the validate_key function
    2. **return(ulong)(param_1 % 0x4c7 == 0)** indicated towards the solution.
- Now converting
    3. **hexadecimal “0x4c7” to decimal, answer was 1223**.
- Entered this while running in the VirtualBox Ubuntu Terminal, the key was finally accepted.



Crackme2binary Solution (The crackme2binary file could not be executed on VirtualBox so I used online editor)

- Similarly, after Ghidra analyzed the crackme2binary code,
- Some things were declared very clearly like the value of-
    1. *local_2b[0] == '6'*
    2. *local_2b[1] == '9'*.
    3. *Strcmp* function and "*Evilzone*" comparison shows the solution will be an ASCII value.
- By observation, if we convert "*Evilzone*" into ASCII, the first two digits are also "69".
- Thus after full comparison, entered the entire ASCII Code after conversion into the terminal
- The key was accepted.

```
~$ ./crackme2binary
Please enter the secret number: qwerty
Nope.
~$ ./crackme2binary
Please enter the secret number: 1234567890
Nope.
~$ ./crackme2binary
Please enter the secret number: 690000000000
Nope.
~$ ./crackme2binary
Please enter the secret number: 691181051081221111110101
The Password translates into Evilzone, Good job.
~$
```

Crackme3binary Solution

- Now, looking at the code after analysis in Ghidra,
- There are 3 things which are certain and that are as follows
    1. **_s[4] = '-'**
    2. **Length of the string _s[] is 9**, which means the password length is 9.
    3. Anywhere in the password if there is **"@",** the key will be accepted.
- Thus, I tried the same in the terminal and the key was accepted.
- It can be of the format -> **"**_ _ _ _ - _ _ _ _**"** with an "@" at anyone of the places excluding the hyphen.