

POST 31: 6-2-20/Article: Tool: Creation of “MAC Address Changer” Tool in Python

Dilpreet Singh Bajwa

Tool: Creation of “MAC Address Changer” Tool in Python

Software Requirement: VMWare for creating Virtual Machine, Linux(Any Distribution) but we prefer Kali Linux, Pycharm.

Hardware Requirement: Laptop or PC (i3 or i5, 2 GB RAM).

Prerequisite Knowledge: Basic Programming logics, OOPs concept and Python Basic Level.

Steps for Creation of Tool in Python:

Step1: Create a virtual machine of Kali Linux through Vmware. It's not necessary that machine must be prepared virtually. You may also install Kali Linux normally on your Laptop or PC. Kali Linux is not also necessary, you may choose any Linux distribution like Redhat, Ubuntu or Debian etc. but we prefer Kali Linux always. Python comes preinstalled on almost all of the linux distributions.

Step2: Install Pycharm on your Kali Linux Machine. You may download it from: <https://www.jetbrains.com/pycharm/> for Linux or windows as well.

Step3: Use pycharm to write, edit and execute your tool.

Here we will focus on only on step-3 i.e. we code and execute the tool and share the code and its explanation with you.

What is MAC Address?

In order to establish some communication between two computers over the network, we need some address as in our postal system i.e. if we want to transfer some mail or message to someone; we need an address of the person. Similarly to communicate over the network we need some address. There are various addresses work In Computer Network.; each works at different layer. Media Access Control Address is a physical address which works at Data Link Layer for node to node delivery whereas IP address works at Network Layer for source to destination delivery and Port addresses uses at Transport Layer for process to process delivery.

MAC addresses are used at data link layer and is 48-bit unique address (12-digit hexadecimal or 6 byte binary ($8 \times 6 = 48$ bits)), which is embedded in to Network Interface Card (NIC) of a computer. It is also known as Physical Address. It must be unique to identify a particular machine, since millions of network devices are in existence and there must be some mechanism to identify them. It is embedded by the manufacturer in to the network card.

MAC address normally represented by Colon-Hexadecimal notation where first 6-digits (say 00:40:96) identifies manufacturer, called as OUI (**Organizational Unique Identifier**). These OUI are provided by IEEE Registration Authority Committee to its registered vendors. Other than colon notation, MAC addresses can also be represented by “Hyphen Hexdecimal notation” and “Period-Separated Hexadecimal Notation”.

Example:

00-0c-84-d1—c0-7e (**Hyphen Hexadecimal notation**)

00:0c:84:d1:c0:7e (**Colon Hexadecimal notation, Most commonly used**)

00.0c.84.d1.c0.7e (**Period Separated Hexadecimal notation**)

You may check the MAC address of your computer network card by typing “ifconfig” command in Linux and “ipconfig” command in windows at command prompt.

Why anyone wants to change the MAC address?

There are various reasons for a person to change his/her or anyone else MAC address. Some of the reasons are given below:

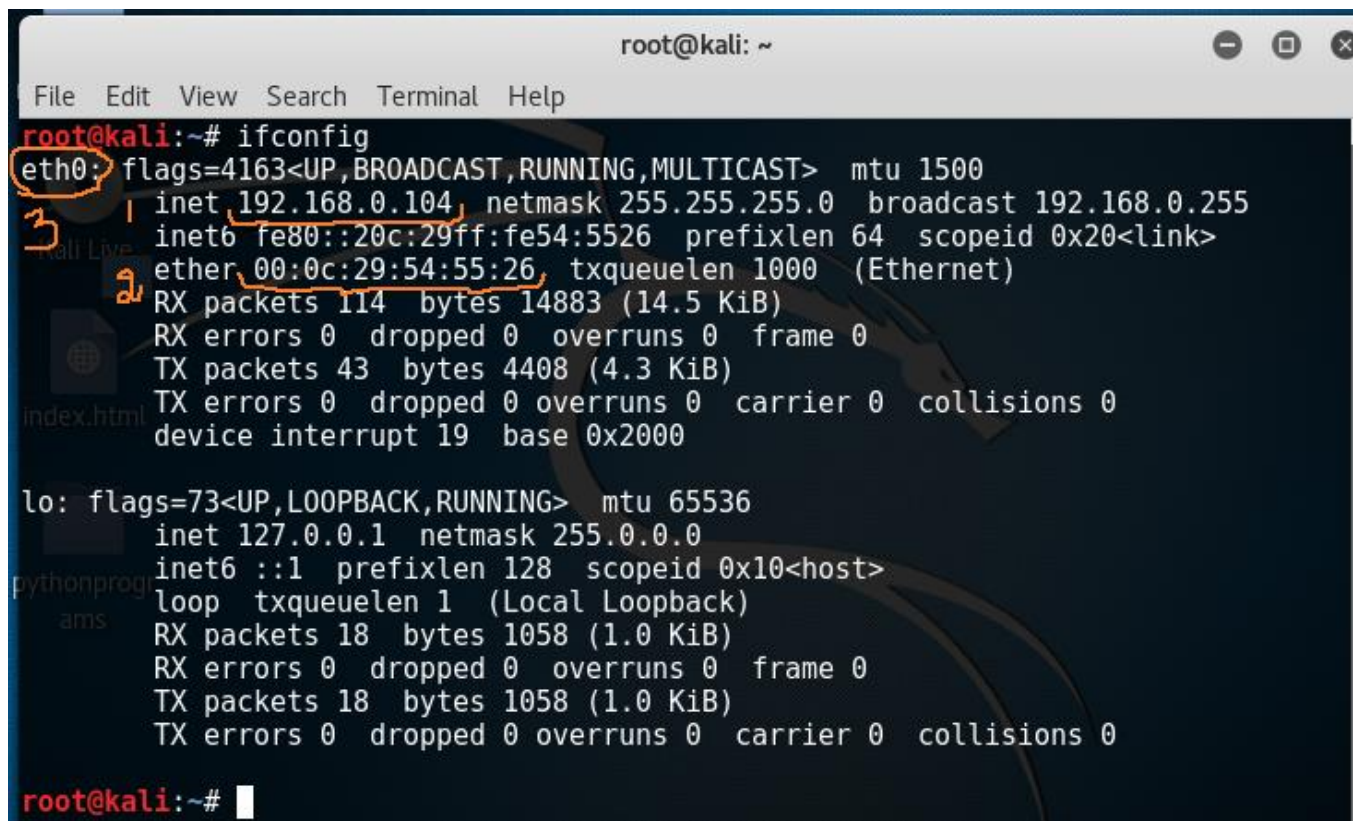
- To be anonymous up to some extent.
- Our Internet service provider (ISP) used our MAC address to identify and authenticate our internet connection. For some reasons, if we change our NIC, the new card have different MAC address and without registering our new MAC address with the ISP, our internet won't work and that process may take longer time. So in order to save time we may change MAC address of our new network card to old.
- Some firewalls and IDSs provide or limit the access to a network based on MAC addresses. To access such network which forbids your machine based on the MAC address, you may change machine's MAC address to the one for which it is allowed. Here point to be note that at a time not more than one machine is allowed on the network with same MAC address i.e. no two computers have same MAC address over the same network.
- Sometimes privacy is also a concern for changing MAC address because various tools are available which provides information about MAC addresses available on some network and a hacker can track your machine with this information and it is really a big threat for anyone.
- A hacker can impersonate you over a network by spoofing your original MAC address and can get access to a restricted network where you are allowed through authentication mechanism based on MAC address. This threat increases if you are using any wireless network.

So now we know what is a MAC address? Why it is important? And why anyone wants to change it?

Now we proceed to our main topic that:

- **How we check MAC address on our Kali Linux machine.**
- **Which commands we can use to change our current MAC address and**
- **How to automate this process by writing the code in python?**

1. Checking MAC address of our Linux machine by using command “ifconfig”



```

root@kali: ~
File Edit View Search Terminal Help
root@kali:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.0.104 netmask 255.255.255.0 broadcast 192.168.0.255
    inet6 fe80::20c:29ff:fe54:5526 prefixlen 64 scopeid 0x20<link>
    ether 00:0c:29:54:55:26 txqueuelen 1000 (Ethernet)
    RX packets 114 bytes 14883 (14.5 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 43 bytes 4408 (4.3 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
    device interrupt 19 base 0x2000

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1 (Local Loopback)
    RX packets 18 bytes 1058 (1.0 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 18 bytes 1058 (1.0 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@kali:~#

```

Here the underlined address at point “1” in figure is the ip address of machine, the address at point “2” is the MAC address i.e. 00:0c:29:54:55:26 and “eth0” is the interface or card. If you are using any wireless card then interface may be “wlan0”.

2. Now we are changing the MAC address by using Linux commands given below:

Ifconfig eth0 down

Here “ifconfig” is the command, “eth0” and “down” are the arguments for the command where “eth0” is the name of interface and “down” turn off the interface eth0.

Ifconfig eth0 hw ether 00:11:22:33:44:55

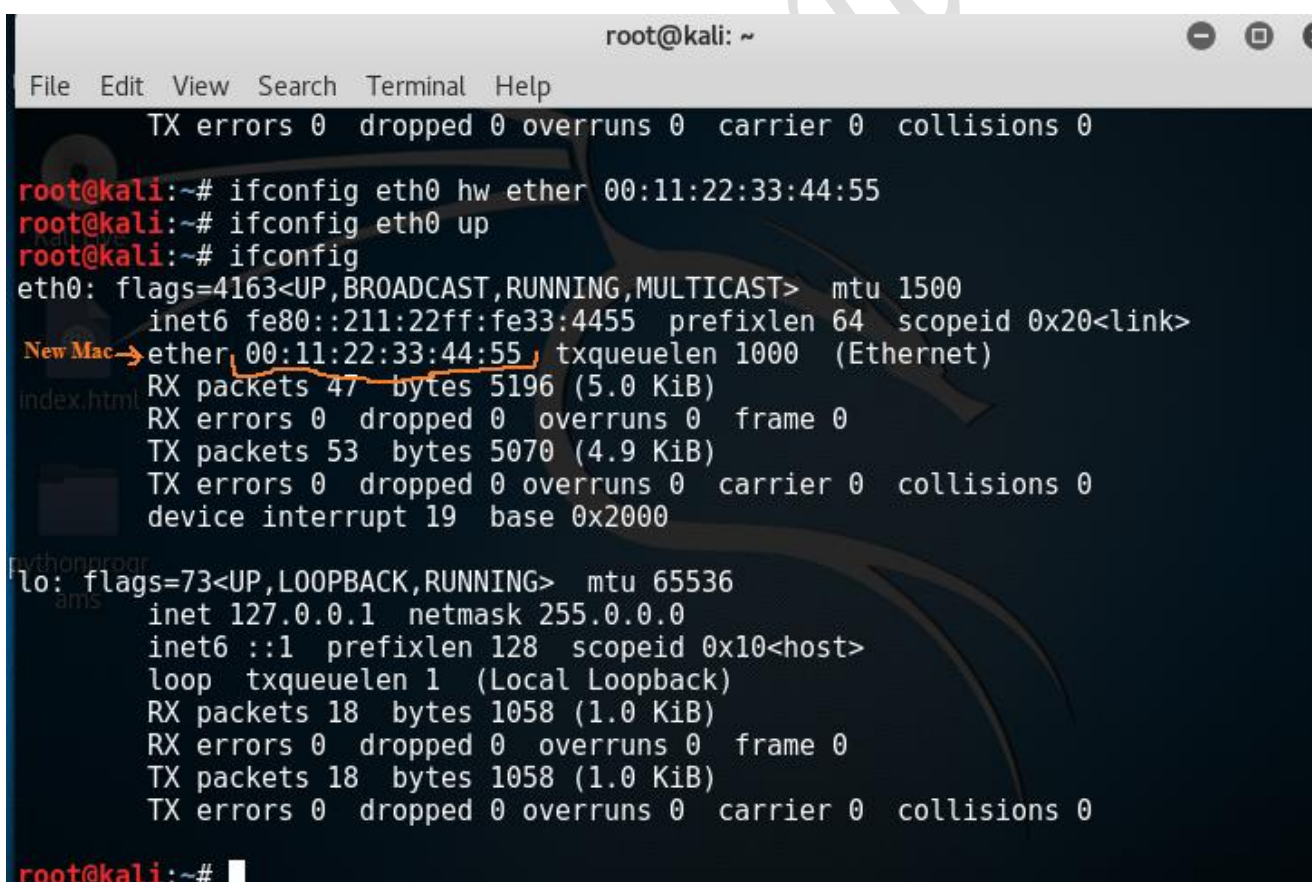
To change the old MAC address with new one, we are using above command. Here 00:11:22:33:44:55 is the new MAC address. You may use any valid MAC address.” hw” means set the hardware address of this interface. It must be followed by the hardware

class (ether) and the new mac address (in our case it is 00:11:22:33:44:55). Hardware classes supported include ether (Ethernet), ax25 (AMPR AX.25), ARCnet and netrom (AMPR NET/ROM).

Ifconfig eth0 up

Here “ifconfig” is the command, “eth0” and “up” are the arguments for the command where “eth0” is the name of interface and “up” turn on or enable the interface eth0.

3. Again execute the “ifconfig” command to check the MAC address:



```

root@kali: ~
File Edit View Search Terminal Help
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
root@kali:~# ifconfig eth0 hw ether 00:11:22:33:44:55
root@kali:~# ifconfig eth0 up
root@kali:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet6 fe80::211:22ff:fe33:4455 prefixlen 64 scopeid 0x20<link>
    New Mac -> ether 00:11:22:33:44:55 txqueuelen 1000 (Ethernet)
    RX packets 47 bytes 5196 (5.0 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 53 bytes 5070 (4.9 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
    device interrupt 19 base 0x2000
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1 (Local Loopback)
    RX packets 18 bytes 1058 (1.0 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 18 bytes 1058 (1.0 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
root@kali:~#
  
```

Here as shown in above snapshot that the MAC address is now changed of interface “eth0” and the new MAC address is “00:11:22:33:44:55”

4. Now we write the code in python to automate this process, so that whenever we run our tool, the MAC address automatically changes:

Code without comments:

```
mac_changer.py x
1 import subprocess
2
3 print("[++] Tool to automate the process of changing MAC Address")
4 interface1 = input("Enter Interface Name: ")
5 new_mac1 = input("Enter New Mac Address: ")
6
7
8 def mac_changer(interface,new_mac):
9     subprocess.call("ifconfig " + interface, shell=True)
10    subprocess.call("ifconfig " + interface + " down",shell=True)
11    subprocess.call("ifconfig " + interface + " hw ether " + new_mac, shell=True)
12    subprocess.call("ifconfig " + interface + " up", shell=True)
13
14
15 mac_changer(interface1,new_mac1)
16 print("[++] " + interface1 + " MAC Address changes to New MAC Address: " + new_mac1)
17 subprocess.call("ifconfig " + interface1, shell=True)
```

Code with comments:

```
mac_changer.py x
1 import subprocess
2
3 print("[++] Tool to automate the process of changing MAC Address")
4
5 interface1 = input("Enter Interface Name: ")# Taking input from user of Interface Name
6 # (our case it is "eth0") and assigning it to interface1 variable
7 new_mac1 = input("Enter New Mac Address: ")# Asking user to provide New Mac Address and assigning
8 # it to "new_mac1" variable
9
10
11 # Below Function "mac_changer" is used to actually changing the old MAC address with new one
12 def mac_changer(interface,new_mac):
13     subprocess.call("ifconfig " + interface, shell=True)# "ifconfig" command execution
14     subprocess.call("ifconfig " + interface + " down",shell=True)# "ifconfig eth0 down" command execu
15     subprocess.call("ifconfig " + interface + " hw ether " + new_mac, shell=True)
16     # above statement executes "ifconfig eth0 hw ether 00:11:22:33:44:55" command
17     subprocess.call("ifconfig " + interface + " up", shell=True)# "ifconfig eth0 up command execution
18
19
20 mac_changer(interface1,new_mac1)# Calling of function Mac_changer and passing arguments "interface1"
21 # which refers to interface name eth0 and second argument we are passing "new_mac1" which
22 # refers to new mac address 00:11:22:33:44:55
23 print("[++] " + interface1 + " MAC Address changes to New MAC Address: " + new_mac1)
24 # Below statement again executing "ifconfig" command and the output will show the changed MAC address
25 subprocess.call("ifconfig " + interface1, shell=True)
```

Explanation:

1st line: `1 import subprocess`

It is the most important part of program as we are here importing an inbuilt module “subprocess” in our program. The subprocess module allows you to spawn new processes, connect to their input/output/error pipes, and obtain their return codes. The subprocess module enables you to start new applications from your Python program. It is used to execute system commands. Here in our program we use it to run Linux commands by using its “call()” function. The subprocess module contain many functions like call().

The method call() has two prototypes.

- (a) subprocess.call(“Command”, shell=True)
- (b) subprocess.call([“Comand”, “Argument_1”, “Argument_n”])

In our above program we use prototype call(“command”, shell=True”) and later at the end of this article, we also provide code for other (b) prototype used for the same program.

3rd line: `3 print("[++] Tool to automate the process of changing MAC Address") :`

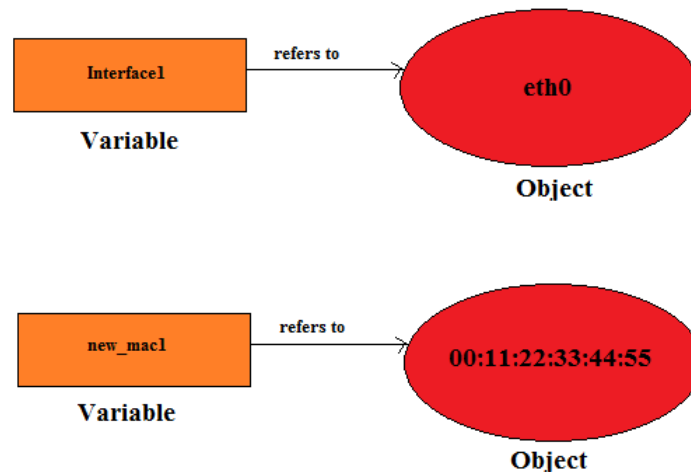
Here we use the print() function of python to print a string message on screen. Everything within double quotes consider as string and printed on standard output device i.e. monitor screen. So this line of code print he message: “**[++] Tool to automate the process of changing MAC Address**”

4th and 5th line: `4 interface1 = input("Enter Interface Name: ")
5 new_mac1 = input("Enter New Mac Address: ")`

In these lines of code we are using two variables “interface1” and “new_mac1” (you may give any name as per rules for python language naming convention of identifiers but name should be meaningful).

“interface1” is actually a reference and refers to a String object which is the name of the interface (eth0) taken from user through the input() function. Here input(“Enter Interface Name:”) function prints the message “Enter Interface Name: “ on screen and waits for the user to

provide the name of interface “eth0” which stores in the memory as String object and the variable “interface1” refers to it. Whenever we are using “interface1” in the program we are actually refers to the value “eth0” of string object to which this “interface1” variable points. Same is true for the variable “new_mac” in line 5. But it refers to the new MAC address (00:11:22:33:44:55) provided by the user. Both scenarios depicted in the figure given below:



Line 8th to 13th:

```

8  def mac_changer(interface,new_mac):
9      subprocess.call("ifconfig " + interface, shell=True)
10     subprocess.call("ifconfig " + interface + " down", shell=True)
11     subprocess.call("ifconfig " + interface + " hw ether " + new_mac, shell=True)
12     subprocess.call("ifconfig " + interface + " up", shell=True)
13

```

This is the main function or logic of our program. Here we have defined the function “mac_changer”. You may also write the code without using the function but to use a function to define any specific logic have their own advantages as it increases reusability of code. You may write the code once in a function and use it many times if required anywhere in the program by calling the function without writing the code again

Further you may import the module where you define the function and use that function anywhere in different projects or different modules.

The function name is mac_changer and we are passing two arguments to it that is first argument is the interface name(eth0) corresponding to which we have to change the MAC address and the

second argument is the new MAC address(00:11:22:33:44:55) which we want to provide to the interface. The function collects both the arguments in parameters “interface” and “new_mac” that is in the function “interface” refers to “eth0” and “new_mac” refers to MAC address “00:11:22:33:44:55”.

In the function definition we are using the call() function of subprocess module to execute Linux commands. For e.g: statement: **subprocess.call(“ifconfig”, shell=True)** will execute “ifconfig” command for us and the output (if any corresponding to command which executes) displayed on screen.

The prototype for the call() method is:

call(“command”, shell=True) where “command” is the linux command and “shell=True” means the commands get executed through a new shell environment. Sometimes using “shell=True” is considered as insecure. So it will be better to avoid it. We will discuss it about sometime later on. The call() function returns the return code of the command that is, If commands executed successfully then the function returns zero otherwise a CalledProcessError exception raised.

In 10th Line the statement used is: **subprocess.call(“ifconfig “ + interface + “ down”, shell=True)**

The whole command interpret as: “ifconfig eth0 down” because the “interface” variable in the function mac_changer refers to value eth0 and the command when executes turn off or disable the interface eth0.

In 11th Line the statement used is: **subprocess.call(“ifconfig ” + interface + “hw ether “ + new_mac, shell=True)**

This line in actual changes the MAC address from old to new one as provided by user. The whole line interpreted as: “ifconfig eth0 hw ether 00:11:22:33:44:55” the value of “interface” parameter and “new_mac” parameter is substituted in the whole command as “eth0” and “00:11:22:33:44:55”.

In 12th Line we again up or enable the interface eth0 by using the statement: **subprocess.call(“ifconfig “ + interface + “ up”, shell=True)**

The whole command interpret as: “ifconfig eth0 up” because the “interface” variable in the function `mac_changer` refers to value `eth0` and the command when executes ON or enable the interface `eth0`.

15th Line: `15 mac_changer(interface1,new_mac1)`

In this line of our program, we are actually calling the function `mac_changer` which changes our machine’s `MAC_address`. Above from line 8th to 12th we only define the function. Function definition itself does not execute the function until we call it. So here in this we call the function, passing required argument that is the interface name and new MAC address and initiate its execution. After calling all statements within the function “`mac_changer`” will execute as specified above.

16th Line:

`16 print("[++] " + interface1 + " MAC Address changes to New MAC Address: " + new_mac1)`

Here again we use the inbuilt function “`print()`” and print the message “eth0 MAC address changes to New MAC address 00:11:22:33:44:55” on our standard output device monitor screen. We are also using string concatenation and use the variable “`interface1`” and “`new_mac1`” to display their values that is “`eth0`” and “`00:11:22:33:44:55`”.

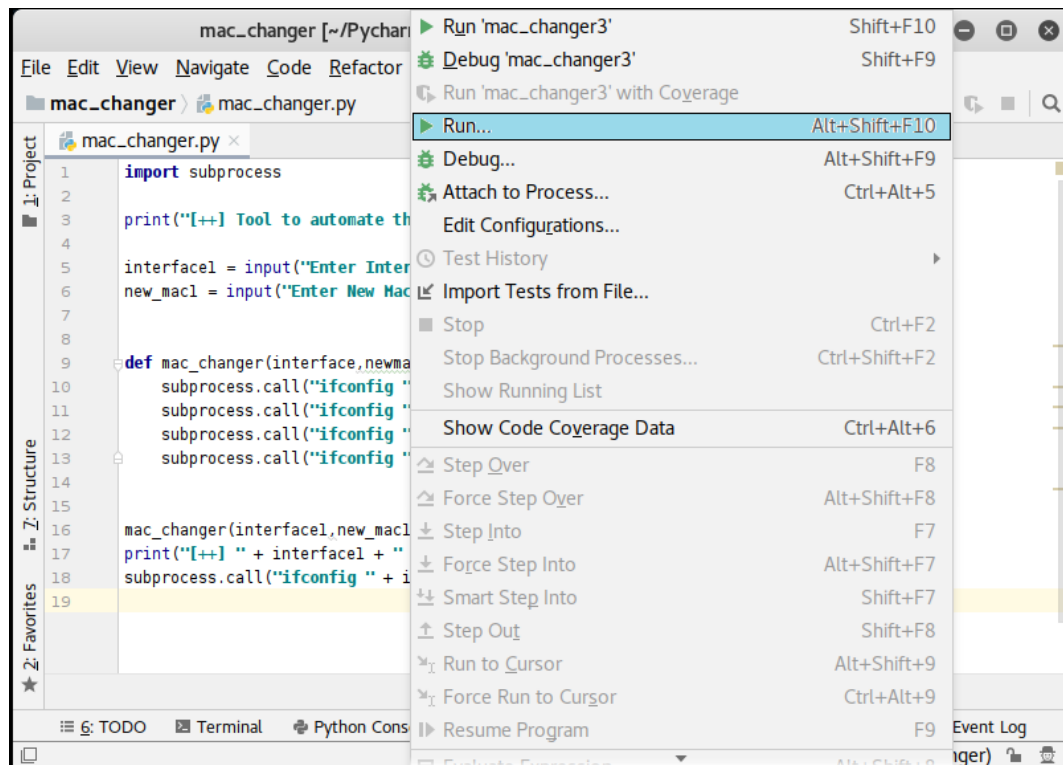
17th Line: `17 subprocess.call("ifconfig " + interface1, shell=True)`

This is the last line of our program. Here we are again executing our “`ifconfig`” command to see whether our MAC address changed or not.

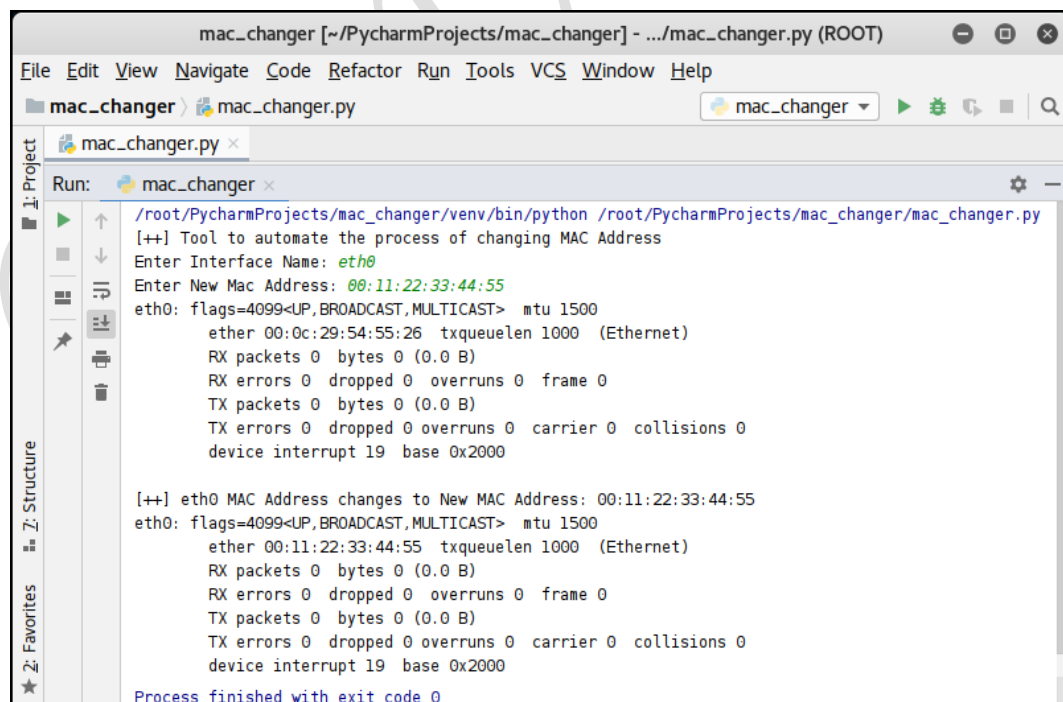
Execution of Program:

We can execute the program either in pycharm to check its output or run it through command prompt. Both the ways are depicted below through snapshot

Go to: Pycharm>Menu>Run>Run.. and execute the program:



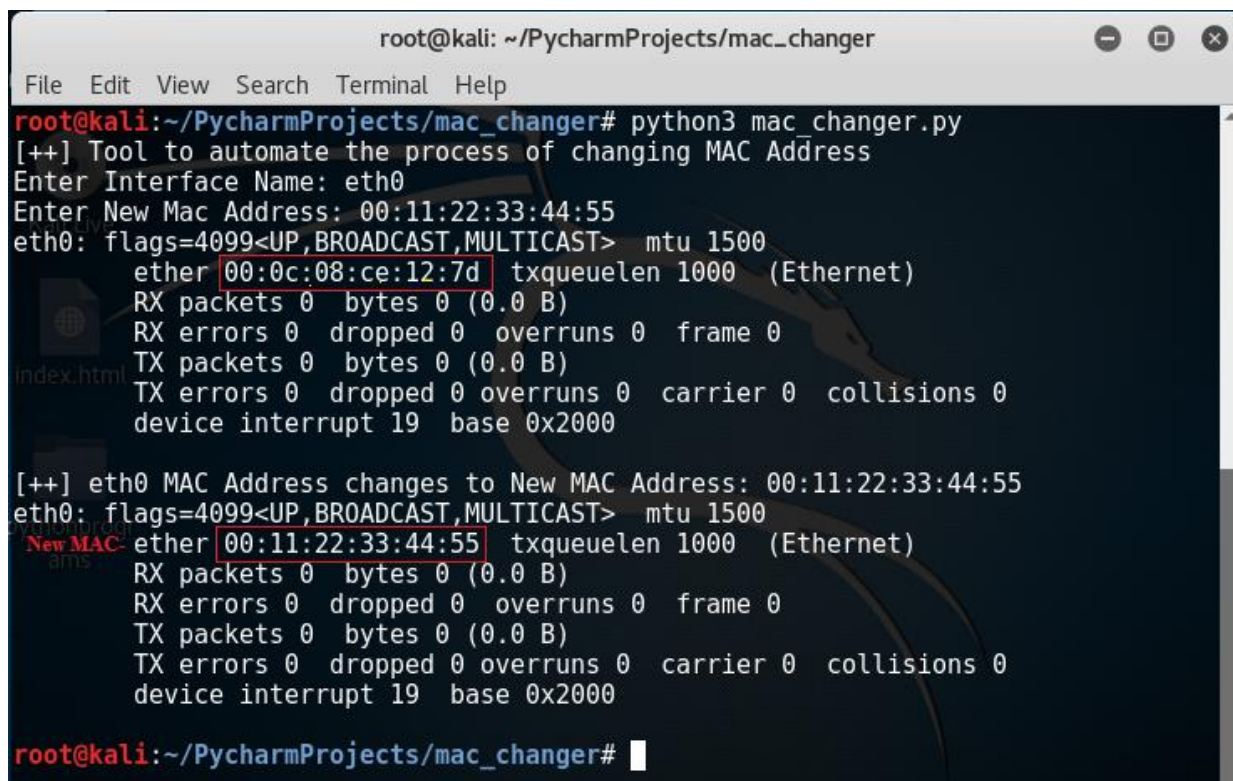
Output:



We may also execute the program through terminal of Linux as follows:

root@kali:~/path to our python file #python3 programname

In our case: **root@kali:~/path to our python file #python3 mac_changer**



```

root@kali: ~/PycharmProjects/mac_changer
File Edit View Search Terminal Help
root@kali:~/PycharmProjects/mac_changer# python3 mac_changer.py
[++] Tool to automate the process of changing MAC Address
Enter Interface Name: eth0
Enter New Mac Address: 00:11:22:33:44:55
eth0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    ether 00:0c:08:ce:12:7d txqueuelen 1000 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
    device interrupt 19 base 0x2000

[++] eth0 MAC Address changes to New MAC Address: 00:11:22:33:44:55
eth0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
New MAC: ether 00:11:22:33:44:55 txqueuelen 1000 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
    device interrupt 19 base 0x2000

root@kali:~/PycharmProjects/mac_changer#
  
```

The whole program will execute without any error because we provide the right values as per expectations of our program but sometimes user may enter wrong or unexpected values and in that case your programs stop abnormally and also shows you corresponding error or there may be chances that due to some run-time error your program crashes and stops abnormally. In both the scenarios we do not want that our program crashes that way. We must have some mechanism or code so that we handle such run-time errors and provide helpful appropriate message for the user.

To handle such errors we must use Exception Handling mechanism of python. In above program we can't use it as the program is a basic program to teach that how we execute system commands and change the MAC address of Linux Machine. In later version of programs we

also add the exception handling as well as other functionality to make it more reliable and efficient.

Note: As told you earlier that there are two prototypes available to use call() method

- (a) `subprocess.call("Command", shell=True)`
- (b) `subprocess.call(["Comand", "Argument_1", "Argument_n"])`

In our above program we use prototype `call("command", shell=True)`. "shell=True" means the commands get executed through a new shell environment. Sometimes using "shell=True" is considered as insecure because an malicious user may able to get execute some other command through this.

So we can use another prototype of call() method:

`subprocess.call(["Command", "Argument_1", "Argument_n"])`, Here we are passing the arguments to functions as a list where 1st element of list is considered as command and all other elements are considered as arguments for the command(1st element of list). So if any malicious user substitute any wrong command while giving input, then still it can't be able to execute because it will not be a valid argument for our command passing as first element of list.

Below we are giving the snapshot of our second version of program which is almost same, only the call() function changes as we are passing list to it. Execution and output remains the same as explained above. Also try this.

mac_changer [~/PycharmProjects/mac_changer] - ~/PycharmProjects/tempmac/progrm1.py (ROOT) — □ ×

File Edit View Navigate Code Refactor Run Tools VCS Window Help

/ > / root > / PycharmProjects > / tempmac > progrm1.py

progrm1 mac_changer.py

```

1  import subprocess
2
3
4  print("[++] Program to automate the process of changing MAC Address")
5  interface1 = input("Enter Interface Name: ")
6  new_mac1 = input("Enter New Mac Address: ")
7
8
9  def mac_changer(interface,new_mac):
10     subprocess.call(["ifconfig", interface])
11     subprocess.call(["ifconfig", interface, "down"])
12     subprocess.call(["ifconfig", interface, "hw", "ether", new_mac])
13     subprocess.call(["ifconfig", interface, "up"])
14
15
16  mac_changer(interface1,new_mac1)
17  print("[++] " + interface1 + " MAC Address changes to New MAC Address: " + new_mac1)
18  subprocess.call(["ifconfig " + interface1,shell=True])

```