SOURCE CODE

Project :Information Gathering Scripted Tool.

This is an integrated tool which collects information by information gathering techniques of three different combinations that are

Mac Changer , Network Packet Sniffer, Network Scanner.

Source Code:

Information Gathering Scripted

Tool:-

Mac changer:

```
#!/usr/bin/env python
import subprocess
import optparse
import re


print(" "*30 + "-"*28)
print(" "*30 + "|" + "  Welcome To Mac Changer  " + "|")
print(" "*30 + "-"*28)


def Get_Arguments():
        parser = optparse.OptionParser()
        parser.add_option("-i","--interface",
dest="interface",help="Interface to change mac address")
        parser.add_option("-m","--mac", dest="mac_to_spoof",help="mac
address to change")
        (options, arguments) = parser.parse_args()
```

```
    if not options.interface:
        parser.error("      Please specify the interface you want to change mac ,use -h or --help to know how to use")
    elif not options.mac_to_spoof:
        parser.error("      Please specify the new mac address ,use -h or --help to know how to use")
    return options


def Changing_Mac(interface,mac_to_spoof):
    subprocess.call(["ifconfig" , interface , "down"])
    subprocess.call(["ifconfig" , interface , "hw" , "ether" , mac_to_spoof ])
    subprocess.call(["ifconfig" , interface , "up"])
    print("Trying To change you mac address")



def Current_Mac(interface):
    mac_result = subprocess.check_output(["ifconfig",interface])
    result = re.search(r"\w\w:\w\w:\w\w:\w\w:\w\w:\w\w",mac_result)

    if result:
        return result.group(0)
    else:
```

```
            print("[*] Couldnot read mac address ,interface doesnt have
mac")

            print("[*] Couldnot read mac address ,interface doesnt have
mac")

options =  Get_Arguments()



previous_mac = Current_Mac(options.interface)
print("[*] Current Mac:: " + previous_mac)

Changing_Mac(options.interface,options.mac_to_spoof)



current_mac = Current_Mac(options.interface)
if current_mac == options.mac_to_spoof:
    print("[*] Mac Address has been changed succesfully from " +
previous_mac + " to " + current_mac )
else:
    print("[*] Cant Able to change Mac")
```

Network Scanner:

```
#! /usr/bin/env python

import scapy.all as scapy
import optparse

def setting():
        parser = optparse.OptionParser()
        parser.add_option("-i","--ip",dest="ip",help="specify your ip")
        (options,arguments)        = parser.parse_args()
        if not options.ip:
                parse.error("Specifiy the ip or ip range")
        return options




def scan(ip):
        arp_request   = scapy.ARP(pdst=ip)
        broadcast     = scapy.Ether(dst="ff:ff:ff:ff:ff:ff")
        arp_req_brd   = broadcast/arp_request
        ans           = scapy.srp(arp_req_brd,timeout = 1,verbose=False)[0]
        print("         " + "-"*42)
        print("         " + "IP" + "  "*3 + "MAC ADDRESS")
        print("         " + "-"*42)
```

```
        for i in ans:
                print("          " + i[1].psrc + "      "*2+  i[1].hwsrc)

options = setting()
scan(options.ip)
```

Network Packet Sniffer:

#packet sniffer for eth ip and tcp/udp header

```
#! usr/bin/python

import socket
import os
import struct
import binascii

def analyse_ethr_header(data):
        ethr_header = struct.unpack("!6s6sH" , data[:14] )
        dst_mac     = binascii.hexlify(ethr_header[0])
        src_mac     = binascii.hexlify(ethr_header[1])
        proto      = ethr_header[2]
        print
"==========================================================ETHER
HEADER
DETAILS==========================================================
====="
```

```
        print  "DESTINATION MAC :::: {0}:{1}:{2}:{4}:{5}:{6} ".format(
dst_mac[0:2],dst_mac[2:4],dst_mac[4:6],dst_mac[4:6],dst_mac[6:8],dst_mac[8:10]
,dst_mac[10:12])
        print  "SOURCE MAC      :::: {0}:{1}:{2}:{4}:{5}:{6} ".format(
src_mac[0:2],src_mac[2:4],src_mac[4:6],src_mac[4:6],src_mac[6:8],src_mac[8:10]
,src_mac[10:12])
        print  "PROTOCOL USED   :::: " + hex(proto)
        ip_bool = False


        if hex(proto) == '0x800':
                ip_bool = True



        data= data[14:]


        return data,ip_bool



def analyze_ip_header(data):
        ip_header = struct.unpack("!6H4s4s",data[:20]);
        version   = ip_header[0] >> 12
        inthdrlen = (ip_header[0] >> 8) & 0x0f
        tos       = ip_header[0] & 0x00ff
        tolen     = ip_header[1]
        identfctn = ip_header[2]
        flags     = ip_header[3] >> 13
        fragntofst= ip_header[3] & 0x1fff
```

```
        tol       = ip_header[4] >> 8
        proto    = ip_header[4] & 0x00ff
        cheksum   = ip_header[5]
        src_adr          =  socket.inet_ntoa(ip_header[6])
        dest_adr  =  socket.inet_ntoa(ip_header[7])
        print
"================================================IP HEADER
DEATILS=================================================
====="
        print "VERSION OF IP          :::: " + str(version)
        print "INTERNET HEADER LENGTH :::: " + str(inthdrlen)
        print "TYPE OF SERVICE              :::: " + str(tos)
        print "TOTAL LENGTH                 :::: " + str(tolen)
        print "IDENTIFICATION NUMBER  :::: " + str(identfctn)
        print "FLAGS SET                :::: " + str(flags)
        print "TIME TO LEAVE                :::: " + str(tol)
        print "FRAGMENT OF SET          :::: " + str(fragntofst)
        print "PROTOCOL USED                :::: " + str(proto)
        print "CHECK SUM              :::: " + str(cheksum)
        print "SOURCE IP ADDRESS      :::: " + str(src_adr)
        print "DESTINATION IP ADDRESS :::: " + str(dest_adr)

        next_proto = ""
        if proto == 6:
                next_proto = "TCP"
        elif proto == 17:
```

```
            next_proto = "UDP"
        data = data[20:]
        return data,next_proto



def analyze_tcp_header(data):
        tcp_header = struct.unpack('!2H2I4H',data[:20])
        src_port   = tcp_header[0]
        dst_port   = tcp_header[1]
        seqno    =  tcp_header[2]
        ackno            = tcp_header[3]
        offset    = tcp_header[4] >> 12
        reserved   = (tcp_header[4] >> 6 ) & 0x03ff
        flags     = tcp_header[4] & 0x003f
        urg     =  flags  &  0x0020
        ack     =  flags  &  0x0010
        psh      =  flags  &  0x0008
        rst      =  flags  &  0x0004
        syn      =  flags  &  0x0002
        fin       = flags & 0x0001
        window          = tcp_header[5]
        checksum   = tcp_header[6]
        urgntpnter = tcp_header[7]
        print
"=============================================TCP
HEADER
```

```
DETAILS==========================================================
====="
        print "SOURCE PORT          :::: " + str(src_port)
        print "DESTINATION PORT     :::: " + str(dst_port)
        print "SEQUENCE NUMBER      :::: " + str(seqno)
        print "ACKNOWLEDGMENT NUMBER :::: " + str(ackno)
        print "OFFSET               :::: " + str(offset)
        print "RESERVED             :::: " + str(reserved)
        if urg:
                print "URG FLAG IS SET"
        if ack:
                print "ACK FLAG IS SET"
        if psh:
                print "PUSH FLAG IS SET"
        if rst:
                print "RESET FLAG IS SET"
        if syn:
                print "SYN FLAG  IS SET"
        if fin:
                print "FIN FLAG IS SET"

        print "WINDOW LENGTH        :::: " + str( window)
        print "CHECK SUM            :::: " + str(checksum)
        print "URGENT POINTER       :::: " + str(urgntpnter)

        data = data[:20]
```

```
        return data



def analyze_udp_header(data):


        udp_header = struct.unpack("!4H",data[:8])
        src_port   = udp_header[0]
        dst_port   = udp_header[1]
        length   = udp_header[2]
        checksum   = udp_header[3]
        print
"==============================================UDP
HEADER
DETAILS=============================================================
====="
        print "SOURCE PORT          :::: " + str(src_port)
        print "DESTIONATION PORT      :::: " + str(dst_port)
        print "LENGTH                 :::: " + str(length)
        print "CHECK SUM              :::: " + str(checksum)


        data = data[8:]
        return data


def main():
        ssk_sniffer =
socket.socket(socket.PF_PACKET,socket.SOCK_RAW,socket.htons(0x003))
        recv_data   = ssk_sniffer.recv(2048)
```

```
        os.system('clear')
        data,ip_bool = analyse_ethr_header(recv_data)
        next_proto = ''
        if ip_bool:
                data,next_proto = analyze_ip_header(data)

        if next_proto == 'TCP':
                data = analyze_tcp_header(data)
        elif next_proto == 'UDP':
                data = analyze_udp_header(data)

while True:
        main()
```