EDR – Nathan Djurasenko

Important imports for the Client/Server

Client Machine And Server need to have wget installed and scapy and all imports above.

Client works both on Windows and Linux

Needed Imports for Client.py

```
import socket
import os
import subprocess
from scapy.all import *
import threading
import time
import sys
from queue import Queue
from threading import Timer,Lock
import datetime
import platform
import select
import wget
```

Needed Imports for Server.py

```
import socket,datetime
import sys,os
import threading
import time
from queue import Queue
import select
```

Disclaimer: Scripts use Python3

How to launch the Server

```
root@kali:~/Desktop/Project# python3 Server.py 192.168.52.128 8003
```

The ip of the local machine is needed and port to open Server on

Connecting Client to running Server

```
root@kali:~# python3 Client.py 192.168.52.128 8003
```

Functions and Class that are used by both Client and Server Scripts

```
class RepeatedTimer(object): # Class for making Repeatable timer
    def   init  (self, duration, function):
        self. duration=duration # the Duration of the Timer
        self. function=function
        self. timer lock=Lock()#  Lock to prevent replacing timer
        self. timer = None# none Means timer is not defined

    def start(self):
        # ~ global t
        with self. timer lock:
            #Timer still running. stop it and restart
            if self. timer is not None:
                self. timer.cancel()
            #Create new Timer
            self. timer =Timer(self. duration,self. function)
            self. timer.start()

    def isAlive(self):
        with self. timer lock:
            # If timer was defined
            if self. timer is not None:
                return self. timer.is alive()
        return False
```

A class used for making Repeatable timers so certain functions like send to server or check could be called every X amount of time

```
def create workers():
    for   in range(NUMBER OF THREADS):
        t = threading.Thread(target=work)
        t.daemon = True
        t.start()


def create jobs():
    for x in JOB NUMBER:
        queue.put(x)
    queue.join()
```

Create workers on different Threads. And creating their jobs.

CLIENT

```
def work():
    while True:
        x = queue.get()
        if x == 1: #Thread used for Packet Sniffing with Scapy.
            print("Threading 1 works")
            sniff(prn=findDNS)

        if x == 2: # Thread used for starting and setting the timers and running them
            print("Threading 2 Works")
            ArpSend()# Check if computer is under MiTM uppon starting the script
            ScapySend() # Check if computer entered blacklisted Site uppon starting the script
            BlackListDownload() # Initialize first BlackList uppon starting the script

            arp=RepeatedTimer(60,ArpSend)# Check if Client was under MiTM attack, if yes send to server, checked every minute.
            scapy=RepeatedTimer(45,ScapySend) # Check if Client entered any BlackListed websites if yes, send to server checked every 45 seconds
            download=RepeatedTimer(300,BlackListDownload) # Check if blacklist file was changed.

            while True:
                if not arp.isAlive():
                    arp.start()
                if not scapy.isAlive():
                    scapy.start()
                if not download.isAlive():
                    download.start()

                ARPChecker()# Check for MiTM attack
                heartbeat Listen()# Check if server sent commands

        queue.task done()
```

Creating the Jobs for clients, Clients are using 2 Threads, one for Always sniffing for packets(For illegal websites accessed)
the other is used to check if any timer is over if so update it/check for server commands/check for MiTM attack(Duplicate MAC).
RepeatedTimer – is a class that upon finishing the written time, will execute the given function, at this case arp is checked every minute and if MiTM occurred in the last minute, Client logs it and sends to server,
scapy is used to check if the sniffer found any browsed blacklisted sites.
download, updates the blacklist URLs every 5 minutes.

```
def BlackListDownload():
    global blacklistedWebsites
    blacklistedWebsitesFile=wget.download(f"http://{host}")
    with open(blacklistedWebsitesFile,'r') as fr:
        blacklistedWebsites=fr.read().splitlines()
    os.remove(blacklistedWebsitesFile)
```

Black list download downloads a file with all Black listed URLs that's sitting on an apache sever, at the server.

```
# Listen for requests from server.
def heartbeat Listen():
    global s
    ready=select.select([s],[],[],0.05)
    if ready[0]:
        data=s.recv(1024)
        sent=data.decode()
        print(sent)
        if len(sent)==0: # Server has been shutdown
            s.close()
            print("Server has been disconnected.")
            os. exit(1)
        elif "blupdate" in sent: # Force blacklist update to client
            print("Forced Blacklist Download by Server!")
            BlackListDownload()
        elif "1" == sent:
            s.send(b"0")
        else: # Server sent commands
            if "cd" in sent: # if Server wants to CD
                exe=sent.strip('\n').split(" ")
                CD(exe[1],s)
            else:
                proc = subprocess.Popen(sent,shell=True,stdout=subprocess.PIPE,stderr=subprocess.PIPE,stdin=subprocess.PIPE)
                output = proc.stdout.read() + proc.stderr.read()
                s.send(output)  # Send output back

def CD(exe,s):
    if ".." in exe:
        os.chdir("..")
    else:
        try:
            os.chdir(exe)
        except:
            return
```

Listen for commands sent by the server
because server is set on a Linux server only, upon shutting down the server empty data is sent to the clients, if client gets it, it exits the script.
if Server Uses "List" command "1" is send.. to check if client is still alive.
After that Its normal Shell commands. As any shell..
When "terminal" is used by the server

```
#If Any BlackListed URLs are entered then, send report to server.
def ScapySend():
    global visitedSites
    if len(visitedSites)>0:
        new d={str(key) for key in visitedSites}
        print(f"[!] Client Entered a blacklisted sites: {new d} at "+str(datetime.datetime.now()))
        log.write(f"[!] Client Entered a blacklisted sites: {new d} at "+str(datetime.datetime.now())+" \n")
        log.flush()
        s.send(b"011")
        visitedSites={}
```

Output if client enters black list site

```
[!] Client Entered a blacklisted sites: {'www.google.co.il', 'facebook.com', 'www.facebook.com', 'g
```

ScapySend is checking if the Sniffer found any blacklisted websites, if yes it logs on the Client and

```
def ArpSend(): #If MiTM attack was present, send to server.
    global foundMiTM
    if len(foundMiTM)>0:
        print(f"[!] Client is under MiTM attack! reported at {foundMiTM}")
        log.write(f"[!] Client is under MiTM attack! reported at {foundMiTM} \n")
        log.flush()
        s.send(b"001")
        foundMiTM={};
```

If Client is being MiTM'd

```
!] Client is under MiTM attack! reported at {'192.168.52.128': '2020-08-28 11:44:42.612868'}
!] Client is under MiTM attack! reported at {'192.168.52.128': '2020-08-28 11:45:42.712768'}
!] Client is under MiTM attack! reported at {'192.168.52.128': '2020-08-28 11:46:42.705727'}
```

sends to ever 011 to inform for the incident.

```
#ARP Checker Start
def ARPChecker():
    global foundMiTM
    temp=platform.system()
    # MiTM Checker for Linux
    if "Linux" in temp:
        with os.popen("arp -a") as f:
            data=f.read()
        lst=data.split("\n")
        macAdress={}
        for i in lst:
            if i: # if i is not empty
                temp=i.split(" ")
                mac=temp[-4] # Contains Checked Mac
                if macAdress.get(mac):
                    ipAdress=temp[1][1:-1] #Contains checked IP Adress
                    if not foundMiTM.get(ipAdress):
                        foundMiTM[ipAdress]=str(datetime.datetime.now())
                macAdress[temp[-4]]=1

    ## MiTM checker for windows
    elif "Windows" in temp:
        #Checking MiTM for Windows Users.
        with os.popen("arp -a") as f:
            data=f.read()
        lst=data.split("\n")
        macAdresses={}
        for i in lst:
            if i:
                if "dynamic" in i:
                    temp=i.split(" ")
                    mac=temp[-9]
                    if macAdresses.get(mac): # if MAC already exists, mac duplicate found!
                        ipaAdress=temp[2]
                        if not foundMiTM.get(ipAdress):
                            foundMiTM[ipaAdress]=str(datetime.datetime.now())
                    macAdresses[mac] = 1
    else:
        print("OS not found")
### ARP Checker ends here
```

ArpChecker checks for MiTM attacks, the codes check either the client is on Linux or Windows and grabs the address and mac performing the MiTM attack.

```
# Using scapy to see if the user entered an invalid sites
def findDNS(p):
    if p.haslayer(DNS):
        if "Qry" in p.summary(): # Qry is the request for the Website.
            url=p.summary() # save summary to variable
            strip=url.split('\"')[-2].replace('"',"")[2:-2] # Strip the text to plain URL
            if len(blacklistedWebsites)>0:
                for url in blacklistedWebsites:
                    if strip in url:
                        if not visitedSites.get(strip):
                            visitedSites[strip]=str(datetime.datetime.now())
```

Scapy Sniffer used to sniff for blacklisted websites.

SERVER

```
NUMBER OF THREADS = 3
JOB NUMBER = [1, 2, 3]
queue = Queue()
heartbeat counter = []
all connections = [] # contains all socket connections connections
all address = [] # Contains all ip adresses
```

```
# Create a Socket ( connect two computers)
def create socket():
    try:
        global host
        global port
        global s
        host = sys.argv[1]
        port = sys.argv[2]
        s = socket.socket()

    except socket.error as msg:
        print("Socket creation error: " + str(msg))
```

Creating a server socket

```
# Binding the socket and listening for connections
def bind socket():
    try:
        global host
        global port
        global s
        print("Binding the Port: " + str(port))

        s.bind((host, port))
        s.listen(5)

    except socket.error as msg:
        print("Socket Binding error" + str(msg) + "\n" + "Retrying...")
        bind socket()
```

Binding the socket

```
# Establish connection with a client (socket must be listening)

def socket accept():
    conn, address = s.accept()
    print("Connection has been established! | " + " IP " + address[0] + " | Port" + str(address[1]))
    send commands(conn)
    conn.close()
```

Accepting incoming sockets by clients

```
def accepting connections():
    for c in all connections:
        c.close()
    del all connections[:]
    del all address[:]

    while True:
        try:
            conn, address = s.accept()
            s.setblocking(1)  # Prevents timeouts

            all connections.append(conn)
            all address.append(address)

            print("Connecting has been established" + address[0])
        except:
            print("ERROR: accepting connection")
```

accepting connections from clients

```
def work():
    while True:
        x = queue.get()
        if x == 1: # Thread for starting the server and accepting connections from Clients
            create socket()
            bind socket()
            accepting connections()
        if x == 2: # Thread for listening for clients commands
            start listening()
        if x==3:
            start terminal() # Thread for Server Terminal
        queue.task done()
```

Server work function
First thread is used for creating and binding the socker for the server and accepting new incoming connection

```
[?] 192.168.52.133 has disconnected from server at 2020-08-28 11:50:41.635856
Connecting has been established192.168.52.138
Connecting has been established192.168.52.133
```

Second thread is used to listen for information coming from the clients.

The third thread is used for terminal usage

```
def start listening():
    while True:
        for i,conn in enumerate(all connections):
            ready=select.select([conn],[],[],0.05)
            if ready[0]:
                # Windows Throws an error when disconnecting from server
                try:
                    data=conn.recv(1024)
                    # Linux Sends empty Data when disconnecting.
                    if len(data.decode()) == 0:
                        print(f"[?] {all address[i][0]} has disconnected from server at "+str(datetime.datetime.now())+"\n")
                        log.write(f"[?] {all address[i][0]} has disconnected from server at "+str(datetime.datetime.now())+" \n")
                        log.flush()
                        del all connections[i]
                        del all address[i]
                        pass

                    else:
                        # 011 - Client entered BlackListed site
                        # 001 - Client under MiTM attack

                        if "011" in data.decode():
                            print(f"[!] {all address[i][0]} Entered a blacklisted site\n")
                            log.write(f"[!] {all address[i][0]} Entered a blacklisted site \n")
                            log.flush()

                        elif "001" in data.decode():
                            print(f"[!] {all address[i][0]} is under MiTM attack!\n")
                            log.write(f"[!] {all address[i][0]} is under MiTM attack!\n")
                            log.flush()
                        else:  # Listen for Client Terminal response
                            print(all address[i][0]+"> "+data.decode())

                except:
                    # Remove Disconnected windows machine
                        print(f"[?] {all address[i][0]} has disconnected from server at "+str(datetime.datetime.now()))
                        log.write(f"[?] {all address[i][0]} has disconnected from server at "+str(datetime.datetime.now())+" \n")
                        log.flush()
                        del all connections[i]
                        del all address[i]
                        pass
```

Start listening script is used to listen for information sent by Clients connected to the server,

```
[?] 192.168.52.133 has disconnected from server at 2020-08-28 11:50:41.635856
```

Server is notified when Client Disconnects
Checking if clients are dead:
when clients are dying:
Linux: when a linux socket dies it sends an empty data to the other sockets.
Windows: when windows socket dies it sends an exception to the server.
both are being monitored here.
When Client is disconnected it removes it from all addresses and con list.
Data that can be sent by the Clients are as follow:
“011” - Client entered black listed websites

“001” – Client is under MiTM attack
when Server gets any of this data, it saves the Adress that sent it and logs it.
or Server in this function might get Client response if terminal function was used by the Server

```
[!] 192.168.52.133 is under MiTM attack!
[!] 192.168.52.133 is under MiTM attack!
[!] 192.168.52.133 is under MiTM attack!
[!] 192.168.52.133 is under MiTM attack!
[!] 192.168.52.138 Entered a blacklisted site
[!] 192.168.52.133 Entered a blacklisted site
```

Send Commands to client when using “terminal option”

```python
# Send commands to client
def send target commands(conn):
    while True:
        try:
            cmd = input()
            if cmd == 'quit':
                break
            if len(str.encode(cmd)) > 0:
                conn.send(str.encode(cmd))
        except:
            print("Error sending commands")
            break
```

```python
def start terminal():
    while True:
        cmd = input("terminal> ")
        if cmd == 'list':
            list connections()
        elif cmd=="blupdate":
            for i,conn in enumerate(all connections):
                conn.send(b"blupdate")
        elif 'select' in cmd:
            conn=connect Client(cmd)
            if conn is not None:
                send target commands(conn)
        else:
            print("Command not recognized")

def connect Client(cmd):
    try:
        target=cmd.replace("select","")
        target=int(target)
        conn=all connections[target]
        print("You're now connected to: "+str(all address[target][0]))
        print(str(all address[target][0]) + ">",end="")
        return conn
    except:
        print("Selection is not valid")
```

There are 3 Options:

List: list current active/alive clients
blupdate: Force blacklist Update on every client
select: when using list, you can see all addresses with their index.
doing select <index> would connect you to a shell with that Client

```
terminal> list
----CLIENTS-----
0 192.168.52.133 35224

terminal> select 0
You're now connected to: 192.168.52.133
192.168.52.133>
```

from now you have Shell access to the Client. Untill you type quit

If server is shutdown, all the clients automatically disconnect

```
Server has been disconnected.
```

OUTPUT

Log .txt of Server

```
!] 192.168.52.133 Entered a blacklisted site
!] 192.168.52.133 Entered a blacklisted site
!] 192.168.52.133 Entered a blacklisted site
?] 192.168.52.133has disconnected from server at 2020-08-28 08:52:46.980073
?] 192.168.52.133has disconnected from server at 2020-08-28 08:53:37.728834
?] 192.168.52.133has disconnected from server at 2020-08-28 08:53:50.943844
?] 192.168.52.133has disconnected from server at 2020-08-28 08:54:01.216083
?] 192.168.52.133has disconnected from server at 2020-08-28 08:54:29.204936
?] 192.168.52.133has disconnected from server at 2020-08-28 08:55:31.943918
?] 192.168.52.133has disconnected from server at 2020-08-28 08:57:27.080372
!] 192.168.52.133 Entered a blacklisted site
!] 192.168.52.133 Entered a blacklisted site
!] 192.168.52.133 Entered a blacklisted site
?] 192.168.52.133has disconnected from server at 2020-08-28 09:00:10.227924
```

Log.txt file can be found on the same folder as the Server.py Script
And will have this output. And this output is shown on console aswell.

```
!] This computer is under MiTM attack! reported at {'192.168.52.2': '2020-08-28 06:01:52.287647'}
!] This computer is under MiTM attack! reported at {'192.168.52.2': '2020-08-28 06:01:57.361685'}
!] This computer is under MiTM attack! reported at {'192.168.52.2': '2020-08-28 06:02:02.406793'}
!] This computer is under MiTM attack! reported at {'192.168.52.2': '2020-08-28 06:02:07.410077'}
!] This computer is under MiTM attack! reported at {'192.168.52.254': '2020-08-28 07:40:03.612588', '192
'2020-08-28 07:40:04.947703'}
!] This computer is under MiTM attack! reported at {'192.168.52.128': '2020-08-28 07:40:08.695458'}
!] This computer is under MiTM attack! reported at {'192.168.52.128': '2020-08-28 07:40:13.667455'}
!] This computer is under MiTM attack! reported at {'192.168.52.128': '2020-08-28 07:40:18.724897'}
!] This computer is under MiTM attack! reported at {'192.168.52.128': '2020-08-28 07:40:23.804760'}
!] This computer is under MiTM attack! reported at {'192.168.52.128': '2020-08-28 07:40:28.926915'}
!] Client Entered a blacklisted sites: {'www.google.co.il'} at 2020-08-28 07:40:38.940340
!] This computer is under MiTM attack! reported at {'192.168.52.128': '2020-08-28 07:40:33.964361'}
!] This computer is under MiTM attack! reported at {'192.168.52.128': '2020-08-28 07:40:39.029245'}
!] Client Entered a blacklisted sites: {'www.facebook.com'} at 2020-08-28 07:40:43.969744
!] Client Entered a blacklisted sites: {'www.google.co.il'} at 2020-08-28 08:52:02.571522
!] Client Entered a blacklisted sites: {'www.facebook.com'} at 2020-08-28 08:52:12.743079
!] Client Entered a blacklisted sites: {'www.ebay.com'} at 2020-08-28 08:52:22.816290
!] Client Entered a blacklisted sites: {'www.google.co.il'} at 2020-08-28 08:58:26.201206
!] Client Entered a blacklisted sites: {'www.facebook.com'} at 2020-08-28 08:58:41.272653
!] Client Entered a blacklisted sites: {'www.ebay.com'} at 2020-08-28 08:58:46.353369
```

Log.txt file of Client.py can be found on the same folder as Client.py.
This output is also printed to the terminal

Because Threads are used, Server can use the terminal and get feedback from Clients at the same time.

```
root@kali:~/Desktop/Project# python3 Server.py 192.168.52.128 8000
Binding the Port: 8000
terminal> Connecting has been established192.168.52.133
```