Project Name:

EDR

Course:

RTS77381

Student Name:

Tomer Dahan, Snir Tohabash

Table of Contents:

Objectives **3**

Requirements **3**

Starting the EDR **4**

Breaking Down the Program **5**

Imports: 5

Start of the program: 5

Functions: 6

Server Side: 6

main() 6

restrictedsites() 6

handleClient() 7

checkConnections() 8

Client Side: 9

main() 9

MITM() 9

findDNS(pkt) 10

Credits **10**

Objectives

1. Running a listening server on a Linux machine.
2. Clients connecting to the server.
3. Server logs relevant data to log files, data such as duplicated MAC addresses and surfing in blacklist websites.
4. The server monitors if a client has been disconnected and knows how many clients are connected.

Requirements

Server Side:

• Internet Connection.
• Server running Linux.
• Python 3+.
• modules: socket, pathlib, subprocess, threading and time.

Client Side:

• Internet Connection.
• Root privileges user.
• Client running Linux or Windows.
• Python 3+.
• Connect to the same local network of the server.
• modules: socket, os, platform, subprocess, threading, time, requests, colorama and scapy.

Starting the EDR

1. Extract the server file to a folder.
2. Set your server's IP address as HOST, also you may choose a port number.

```
PROJECTPATH = Path(__file__).resolve().parent
HOST = '0.0.0.0'
PORT = 5050
```

3. Open a terminal in the EDR path, run the Server.py and wait for connections from clients.

```
┌──(kali㉿kali)-[~/…/red/projects/edr/Endpoint-Detection-and-Response]
└─$ python Server.py
[INFO] Listening on port 5050 - Waiting for connections ...
```

4. Copy the Client.py to all clients you want to monitor. (Make sure they are on the same network) and set your server's IP address as HOST and the port you chose.

```
HOST = '0.0.0.0'  # Server IP.
PORT = 5050  # Server's listening port.
```

5. Run Client.py to connect and start the monitoring process.

```
┌──(kali㉿kali)-[~/…/red/projects/edr/Endpoint-Detection-and-Response]
└─$ sudo python Client.py
Trying to connect to the server ...
[INFO] You are connected to: 0.0.0.0 in port: 5050.
Successfully connected to EDR Server at 0.0.0.0:5050.
```

6. You have successfully connected a client. Now do it for all other clients in your network.

```
┌──(kali㉿kali)-[~/…/red/projects/edr/Endpoint-Detection-and-Response]
└─$ python Server.py
[INFO] Listening on port 5050 - Waiting for connections ...
[INFO] 127.0.0.1:33228 Connected!
[INFO] Number of Active Connections: 1
```

Breaking Down the Program

Imports:

Server side:

```
import socket
from pathlib import Path
from subprocess import check_output, run
from threading import Thread
from time import sleep
```

Client side:

```
import socket
from os import path, remove
from platform import system
from subprocess import check_output, run
from threading import Thread
from time import sleep
import requests
import colorama
from scapy.all import *
```

Start of the program:

Server side:

```
132  # Start of the Script:
133  if __name__ == '__main__':
134      main()
```

Workflow described is: start the main function

Client side:

```
106  # Start of the Script:
107  if __name__ == '__main__':
108      main()
109      Thread(target=MITM).start()
110      Thread(target=sniff(prn=findDNS)).start()
```

Workflow described is: start the main function, then start 3 threads responsible of completing certain objects. (More info in Functions – Client Side:)

Functions:

Server side:

main()

- Binds socket to ((HOST, PORT)), listening to connections, accepting new connections, sets a format for connName.
- Sends welcome message to new clients, appends new client's socket objects and connName to the lists.
- Starts 2 threads: One for handling clients and the other for checking connections with clients.

```
35  def main():
36      try:
37          serverSocket.bind((HOST, PORT))  # Bind the socket.
38      except socket.error as error:
39          exit(f'\033[1;31;40m[ERROR]\033[0m Error in Binding the Server:\n{error}')
40      print(f'\033[1;32;40m[INFO]\033[0m Listening on port {PORT} - Waiting for connections...')
41      serverSocket.listen()
42      for clientSocket in openClientSocketsList:
43          # Closes all preavious connections if Server.py restarted:
44          clientSocket.close()
45          # Deletes all previous open client sockets and active addresses from the lists:
46          del openClientSocketsList[:], activeAddressesList[:]
47
48      while True:
49          try:
50              # Accepts connections:
51              conn, (address, port) = serverSocket.accept()
52              # Appends the client's socket to the list:
53              openClientSocketsList.append(conn)
54              # Set a format for the connName using client's address and port:
55              connName = '{}:{}'.format(address, port)
56              print(f'\033[1;32;40m[INFO]\033[0m {connName} Connected!')
57              welcomeMessage = f'Successfully connected to EDR Server at {HOST}:{PORT}.'
58              # Sends welcome message to the client:
59              conn.send(welcomeMessage.encode())
60              restrictedsites(conn)
61              global connectionsCount
62              connectionsCount += 1  # Adding +1 to the connections count.
63              # Appends the new address to the activeAddressesList:
64              activeAddressesList.append(connName)
65              # Prints current connections count:
66              print(f'\033[1;32;40m[INFO]\033[0m Number of Active Connections: {connectionsCount}')
67              # Starts a new thread to handle each client (args are the connection and formatted connection name):
68              Thread(target=handleClient, args=(conn, connName)).start()
69              # Starts a checkConnections thread:
70              Thread(target=checkConnections).start()
71          except socket.error as acceptError:
72              print(f'\033[1;31;40m[ERROR]\033[0m Accepting Connection from: {conn.getpeername()}:\n{acceptError}')
73              continue
```

restrictedsites()

- Sends restricted websites list to the client

```
26  def restrictedsites(conn):
27      recvthanks = conn.recv(1024)
28      restrictedWebsites = 'facebook youtube ynet netflix blackweb'
29      conn.send(restrictedWebsites.encode())
```

handleClient()

- Main function to recieve data from all clients.
- Handles client connections using args from main.
- If data has "MAC" in it, logs the data to 'MitMLogger.log'
- If data has "restricted" in it, logs the data to 'RestrictedSitesLogger.log'

```python
def handleClient(conn, connName):
    while True:
        try:
            data = conn.recv(4096).decode()
            if "MAC" in data:
                # Timestamp for the log file:
                timestamp = check_output("date -u +'%d/%m/%Y %H:%M'", shell=True).decode().rstrip()
                print('\033[1;33;40m[WARNING]\033[0m Possible Man in the Middle attack. Check MitMLogger.log')
                with open(f"{PROJECTPATH}/MitMLogger.log", "a+") as MitMLog:
                    MitMLog.write(f"[{timestamp}]\t[{connName}]:\n{data}")  # Logs the MitM attack from the client to 'MitMLogger.log'

            if "restricted" in data:
                # Timestamp for the log file:
                timestamp = check_output("date -u +'%d/%m/%Y %H:%M'", shell=True).decode().rstrip()
                print(f'\033[1;35;40m[ALERT]\033[0m Someone entered to a restricted site. Check RestrictedSitesLogger.log')
                with open(f'{PROJECTPATH}/RestrictedSitesLogger.log', 'a+') as restrictedLog:
                    restrictedLog.write(f"[{timestamp}]\t[{connName}]:\n{data}")  # Logs the restricted site from the client to 'RestrictedSitesLogger.log'
        except:
            pass
```

- Output:

```
[WARNING] Possible Man in the Middle attack. Check MitM Logger.log
[WARNING] Possible Man in the Middle attack. Check MitM Logger.log
[WARNING] Possible Man in the Middle attack. Check MitM Logger.log
[ALERT] Someone entered to a restricted site. Check Restricted Sites Logger.log
[ALERT] Someone entered to a restricted site. Check Restricted Sites Logger.log
[ALERT] Someone entered to a restricted site. Check Restricted Sites Logger.log
```

- The server knows where the data came from and logs its details in the log with the current timestamp.

```
┌──(kali㉿kali)-[~/…/red/projects/edr/Endpoint-Detection-and-Response]
└─$ cat RestrictedSitesLogger.log
[29/04/2022 07:41]      [192.168.198.129:58668]:
[ALERT] Entered a restricted website:
netflix
```

checkConnections()

- Checks what clients are alive by iterating through every client socket object and trying to send a whitespace string.
- If an exception occurs, it means that the client is dead.
- Deletes the client socket object and address from the lists and decreasing 1 from connections count.
- This check happens every 30 seconds.

```
107 def checkConnections():
108     while True:
109         global connectionsCount
110         if len(openClientSocketsList) != 0:
111             for x, currentSocket in enumerate(openClientSocketsList):
112                 try:
113                     # Send a whitespace to every socket in the list:
114                     pingToClientMessage = ' '
115                     currentSocket.send(pingToClientMessage.encode())
116                 except:
117                     print(f'\033[1;32;40m[INFO]\033[0m Client {x} Disconnected!')
118                     # Deletes the client socket and address from the lists:
119                     del openClientSocketsList[x], activeAddressesList[x]
120                     connectionsCount -= 1
121                     if connectionsCount == 0:  # If no connections left:
122                         print(f'\033[1;32;40m[INFO]\033[0m No active connections left.')
123                     else:  # If there are still connections left:
124                         print(f'\033[1;32;40m[INFO]\033[0m Number of Active Connections: {connectionsCount}')
125                         print(f'\033[1;32;40m[INFO]\033[0m Active addresses connected:')
126                         # Prints a list of the current open connections:
127                         for index, value in enumerate(activeAddressesList):
128                             print(f'{index}. {value}')
129                     continue
130         sleep(30)
```

- Output:

```
┌──(kali㉿kali)-[~/…/red/projects/edr/Endpoint-Detection-and-Response]
└─$ python Server.py
[INFO] Listening on port 5050 - Waiting for connections...
[INFO] 127.0.0.1:33228 Connected!
[INFO] Number of Active Connections: 1

[INFO] Client 0 Disconnected!
[INFO] No active connections left.
```

Client side:

main()

- Creates a socket object.
- Connects to server and prints the welcome message.

```
28  def main():
29      global clientSocket
30      global restrictedSitesList
31      # Client's Socket Object:
32      clientSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
33
34      print('Trying to connect to the server...')
35      try:
36          clientSocket.connect((HOST, PORT))  # Connects to the server's socket.
37          print(f'\033[1;32;40m[INFO]\033[0m You are connected to: {HOST} in port: {PORT}.')
38          welcomeMessage = clientSocket.recv(1024)  # Receives welcome message.
39          print(welcomeMessage.decode())
40          thankyou = 'thanks'
41          clientSocket.send(thankyou.encode())
42          sites = clientSocket.recv(1024)  # Receives restricted sites list and adds to restrictedSitesList.
43          sites_decoded = sites.decode()
44          websites = sites_decoded.split()
45          for site in websites:
46              restrictedSitesList.append(site)
47
48      except socket.error as error:
49          exit(f'\033[1;31;40m[ERROR]\033[0m Connecting to the server failed:\n\033[31m{error}\033[0m')
```

MITM()

- Checks for duplications in ARP table in both Linux and Windows.
- Iterates through the MAC addresses in the ARP table, adding them to a list.
- If a duplication occurs - the value of the MAC in the dictionary will rise by 1.
- For every MAC key that has a value of more than 1, it will send a warning message to the server.
- The scan happens every 15 seconds, can be changed.

```
57  def MITM():
58      while True:
59          macList = []
60          macDict = {}
61          if runningOS == "Windows":
62              ARPmacs = check_output("arp -a", shell=True).decode()
63
64              for line in ARPmacs.splitlines():
65                  if "dynamic" in line:
66                      macList.append(line[24:41])
67
68              for MAC in macList:
69                  if MAC in macDict:
70                      macDict[MAC] = macDict[MAC] + 1
71                  else:
72                      macDict[MAC] = 1
73
74              for MAC, value in macDict.items():
75                  if value >= 2:
76                      clientSocket.send(f'\033[1;33;40m[WARNING]\033[0m Found MAC address duplication. Possible Man in the Middle Attack!\nCheck this MAC: {MAC}\n\n'.encode())
77
78          elif runningOS == "Linux":
79              ARPmacs = check_output("arp | awk '{print $3}' | grep -v HW | grep -v eth0", shell=True).decode()
80              for line in ARPmacs.splitlines():
81                  macList.append(line)
82
83              for MAC in macList:
84                  if MAC in macDict:
85                      macDict[MAC] = macDict[MAC] + 1
86                  else:
87                      macDict[MAC] = 1
88              for MAC, value in macDict.items():
89                  if value >= 2:
90                      clientSocket.send(f'\033[1;33;40m[WARNING]\033[0m Found MAC address duplication. Possible Man in the Middle Attack!\nCheck this MAC: {MAC}\n\n'.encode())
91          sleep(15)
```

findDNS(pkt)

- Sniffs DNS quearys of the client.
- Gets only the name of the website from the queary. Setting it to url variable.
- If the name of the site from the restrictedSitesList found in the current sniffed url variable - sends an alert to the server.

```
97  def findDNS(pkt):
98      if pkt.haslayer(DNS):
99          if "Qry" in pkt.summary():  # Only quearys.
100             # Gets only the name of the website from the queary:
101             url = pkt.summary().split('\"')[-2].replace("", "")[2:-2]
102             for site in restrictedSitesList:
103                 if site in url:
104                     clientSocket.send(f'\033[1;35;40m[ALERT]\033[0m Entered a restricted website:\n{site}\n\n'.encode())
```

Credits:

RobertJonnyTiger/Endpoint-Detection-and-Response
url: https://github.com/RobertJonnyTiger/Endpoint-Detection-and-Response