**Intrusion Detection System**

Adam Sin: 322453689

Eliyahu Fridman: 211691159

**Contents:**

1) Introduction

2-7) Structure

2) Docker, Attacker.py

3) Sniffer.py, Analyzer.py

4) TCP tests

5) UDP,DNS,ICMP tests

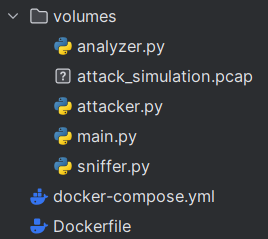
6) HTTP tests

7) TLS tests, Main.py

8) Summery

**Introduction:**

Our goal was to build an IDS focusing on packets suspicious of data exfiltration, looking only into the structure of the packets. The system will sniff traffic on an interface, test each packet, and add to the list of sniffed packets the general information of the packet, a flag of whether the packet failed a test, and the reason why it failed.

**Structure:  
**

**Docker-compose.yml & Dockerfile:**

We would like to run the IDS inside a docker container for a main reason:

**Isolation** - When we start to sniff the traffic, random packets that go through the traffic will get sniffed and tested, which is the point of the IDS, but when testing the system using TCP replay we don’t want any other packets to interfere with the process.

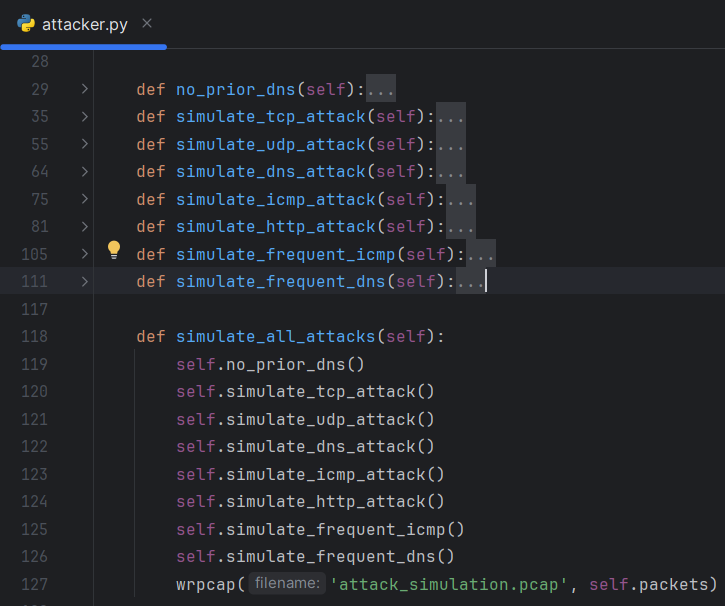
Docker-compose.yml will set up the container and Dockerfile contains the interfaces we want to install to run the IDS like tkinter, pyshark and more.

**Attacker.py:**

This script’s only purpose is to create “attack\_simulation.pcap” which is used for the TCP replay testing each and every test the analyzer conducts on the packets.

The script creates a list with 132 packets, most of them meant to be suspicious of data exfiltration while a small amount are valid and their purpose will be explained later. The list is them used to create the pcap file using scapy’s wrpcap function.

All of the packets are outgoing, meaning they are sent from our company’s internal IPs to unknown external IPs. Because internal traffic shouldn’t cause data exfiltration unless we have a physical mole, external traffic has nothing to do with our company, and the assignment says to not test ingoing packets (from outside in).



**Sniffer.py:**

The class that will be later used in the main IDS script, is responsible for sniffing and storing every packet coming through the traffic using pyshark’s LiveCapture function.

The sniffer has a Queue which is the buffer, this buffer is later used by the analyzer to test the packets.

It also has a flag that determines whether to keep sniffing the traffic or stop.



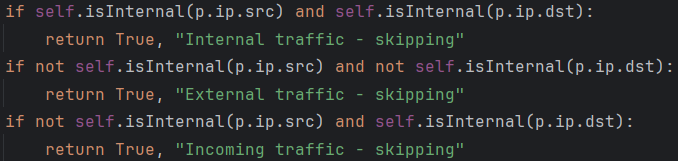
**Analyzer.py:**

Here we conduct all the tests on the packets, to look and flag the suspicious packets.

Validate will go through the buffer for as long as the sniffer runs and there are packets to validate.

isValid returns for each packet whether the packet is valid or not using the functions we’ll expand on



Like said earlier, packets that are not outgoing don’t interest us:  


**tcp\_handle:**

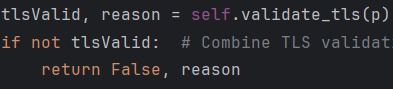
Checks for usual tcp packet ports for secure traffic. Unusual ports may indicate tries to leak data unnoticed. More secure ports can and should be added.



Tests for HTTPS packets on TCP protocol



A https packet must pass security tests, which is its purpose (tls tests are explained later)



https packets sized more than 1460 bytes is unusual, we won’t allow sending so much data.

תמונה שמכילה טקסט, גופן, צילום מסך, גרפיקה

התיאור נוצר באופן אוטומטי

Unusually large TCP size might be a sign to leaked data in the payload



Both flags that represent both opening and closing a connection are very contrary and might be due to a try to use the server in an unusual way.

תמונה שמכילה טקסט, גופן, צילום מסך, גרפיקה

התיאור נוצר באופן אוטומטי

PSH flag means packets that should be processed by the receiver before any other given packets in the buffer. A packet too big of that importance might indicate a try to process and get the data fast.

תמונה שמכילה טקסט, גופן, צילום מסך

התיאור נוצר באופן אוטומטי

URG flag means packets with some important info that should be processed by the receiver before any other given packets in the buffer. A packet too big of that importance might indicate a try to process and get the data fast.

תמונה שמכילה טקסט, גופן, צילום מסך

התיאור נוצר באופן אוטומטי

**udp\_handle:**

Checks for usual udp packet ports for secure traffic. Unusual ports may indicate tries to leak data unnoticed. More secure ports can and should be added.



NTP packets are for syncing servers and clients’ computers. This packet is usually sized 48 bytes so any other number of bytes is suspicious and should be checked for having different data.



**dns\_handle:**

If the packet is used as DNS



Any other port than 53, the usual port for DNS might be used for malicious causes. An example is trying to get a DNS response without being detected as a DNS request. 

DNS packets with a big payload indicates added data within getting leaked.

תמונה שמכילה טקסט, גופן, צילום מסך, טיפוגרפיה

התיאור נוצר באופן אוטומטי

track\_attempts() checks how many DNS responses were asked from the same IP and if too many packets in a small window of time might be a try to leak data undetected using many small packets



if the packet is not a DNS packets with port 53 it may indicate a try to send data using a trusted port to not be detected.



For UDP packets that aren’t DNS we check that the payload doesn’t contain a DNS response. Secretly transmitted DNS responses should alert us.

תמונה שמכילה טקסט, צילום מסך, גופן

התיאור נוצר באופן אוטומטי

To get here means the packet has nothing to do with DNS. If so, DNS\_set() check that there was a valid DNS response before sending this packet. A case where a packet knew to be sent to our server without a DNS means someone knew in advance our information or were somehow under our radar. Both cases are suspicious and might be intended for data leaks.

תמונה שמכילה טקסט, גופן, צילום מסך

התיאור נוצר באופן אוטומטי

**icmp\_handle:**

ICMP packets with a big payload indicates added data within getting leaked.



track\_attempts() checks how many ICMP responses were asked from the same IP and if too many packets in a small window of time might be a try to leak data undetected using many small packets



**http\_handle:**

HTTP packets with a big payload indicates added data within getting leaked.



Host is a mandatory for HTTP 1.1 or higher in order to determine the specific site asking for response. Missing host in the header is unusual and may be to stay undetected.

תמונה שמכילה טקסט, גופן, צילום מסך, קו

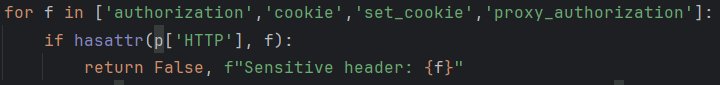
התיאור נוצר באופן אוטומטי

HTTP packets with a big header indicates added settings or data within getting leaked.

תמונה שמכילה טקסט, גופן, צילום מסך, גרפיקה

התיאור נוצר באופן אוטומטי

HTTP packets with these headers mean they contain sensitive information like passwords, permissions, etc. The server should be alerted if it wasn’t meant to be sent.



Requests that were sent using potentially malicious scripts may try extract data from the server.



Allows sending data in chunks instead of all at once. Can be used to hide the amount of leaked data.



if the referrer that sent the request isn’t a trusted domain, we shouldn’t allow him to receive the response.

תמונה שמכילה טקסט, צילום מסך, גופן

התיאור נוצר באופן אוטומטי

track\_attempts() checks how many HTTP responses were asked from the same IP and if too many packets in a small window of time might be a try to leak data undetected using many small packets



**validate\_tls:**

A HTTPS packet as to have a TLS layer by definition.



Handshake types other than “Client Hello”, “Server Hello” in a TLS layer may be of a malicious intent.

תמונה שמכילה טקסט, גופן, צילום מסך, טיפוגרפיה

התיאור נוצר באופן אוטומטי

A TLS without a SNI that supposed to contain information about the receiver may be due to security breaches or an untrusted connection.



if the TLS version is not secure enough might be to try and conduct malicious activity.

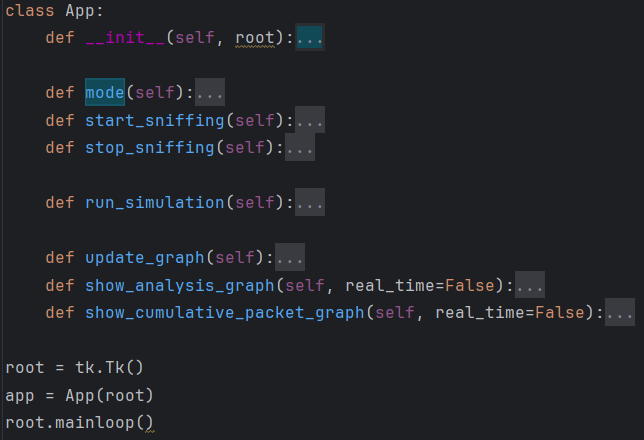


if the cipher suite, which is the encryption algorithm aren’t secure enough, it may be so it will be easier to read leaked data.



**Main.py:**

Combines the scripts to simulate the IDS sniffing, analyzing, statistics and testing stages.



**To run the grogram:**

1) Download Xlaunch from <https://sourceforge.net/projects/vcxsrv/>

2) Run WSL

3) Go to the docker-compose.yml & volumes directory

4) docker-compose build

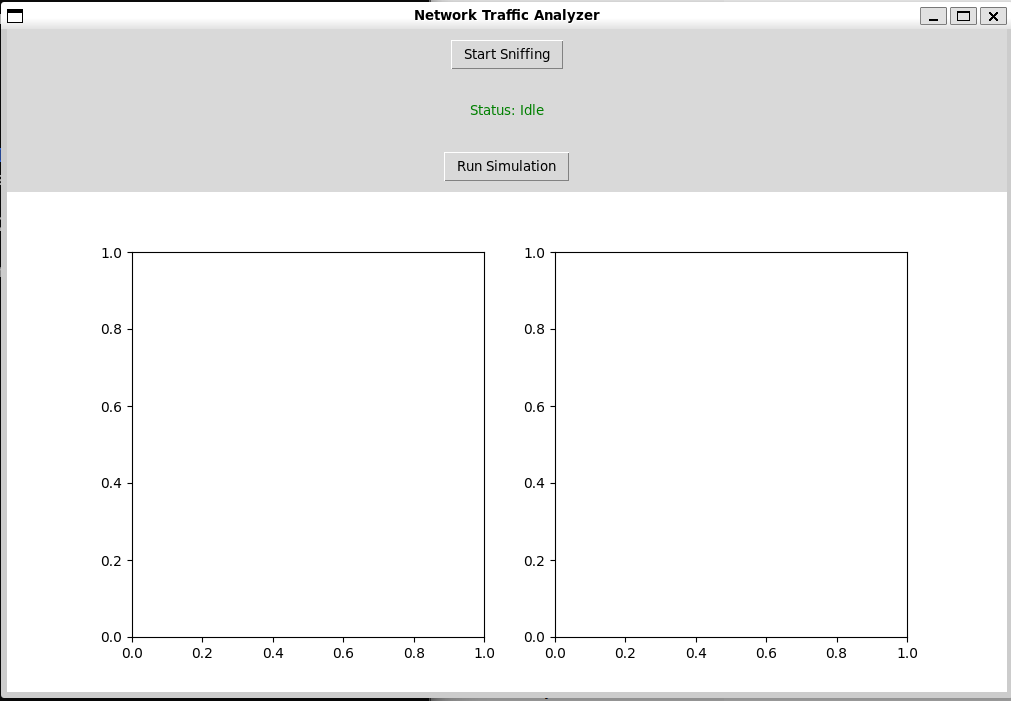
5) docker-compose up

6) docker exec -it protocols\_pyshark\_sniffer\_1 bash

7) cd volumes

8) python3 main.py

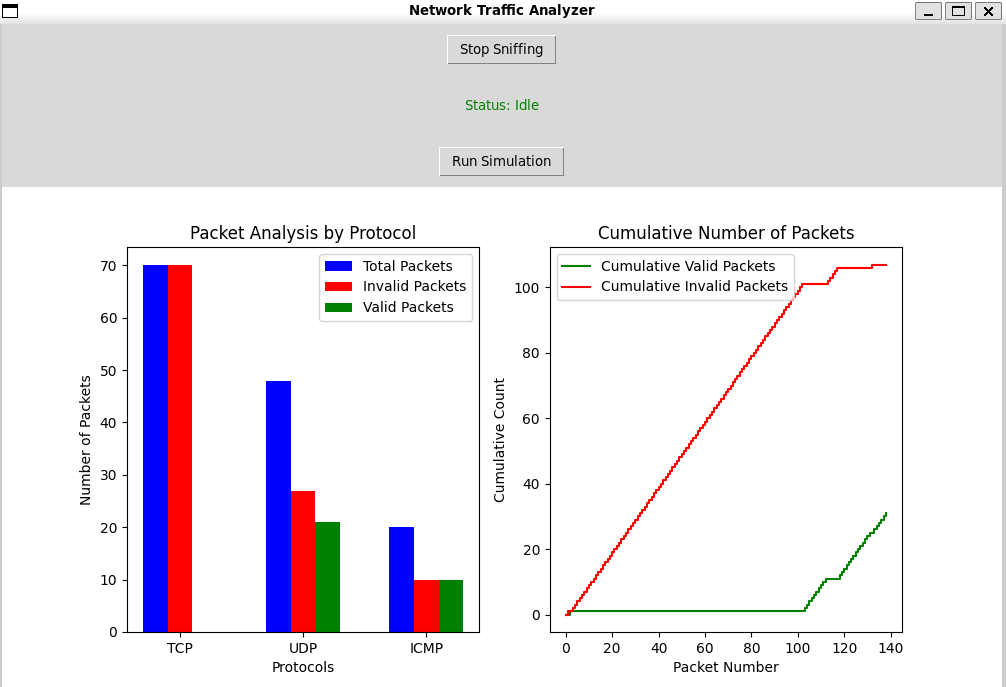
When running the program, we will see a GUI with two options:



“Start Sniffing” will run the sniffer across the interface inside the docker. Another click will stop.

“Run simulation” will use TCP replay to simulate the pcap across the interface so the sniffer could sniff the packets and have the analyzer test them.

At the end of the simulation the next statistics:



**Packets Analysis by protocol:**

for each protocol TCP,UDP,ICMP we count:

Blue - the number of packets sniffed

green - the number of valid packets sniffed

red - the number of packets that are suspicious of data exfiltration that were sniffed

**Cumulative Number of Packets:**

We want to see the cumulative number of valid/invalid packets over time:

Red – the cumulative number of packets that are suspicious of data exfiltration

Green - the cumulative number of valid packets