Level 1.3 - Malware signature in HTTP

We will implement DPI (Deep-Packet-Inspection) and detect malware downloads over HTTP! We can assume HTTP servers are on port 80.

Password

`59zfz404y0`

Brainstorm

1. Let's define the problem:
    - You know Wireshark's Right click -> Follow -> TCP Stream -> Save? We want to do that!
    - We want to track HTTP downloads (data from server to client)
    - When the download is finished, we will hash the data (only the body of course - the HTTP headers are irrelevant)
    - If the hash matches a hash from the list - we will display an alert
2. Streams
    - A TCP session between A and B consists of two streams - from A to B, and from B to A
    - Each session can be identified as (SrcIP, SrcPort, DstIP, DstPort)
3. We get a single packet every time, but a stream can be made of many packets
4. We need to devise an algorithm for track and reassemble each TCP stream

Part A: Stream tracking algorithm overview

1. First, we will need to track each TCP packet as it arrives
    - Previously, we only needed to track a single packet
    - Now, we need to track a stream, which involves multiple packets that make up the stream
2. We will need an identifier to track each stream = When a **TCP packet** arrives, extract its `SrcIP, SrcPort, DstIP, DstPort` - we will call this the **stream identifier**
3. If this is the first packet of the stream: create a new list to store the stream's data
    - The first packet will usually be a SYN packet
4. Add the subsequent packets from that stream to the list, until the stream has ended
    - Use the stream identifier to identify which packets are part of the stream
5. If a packet is received that signifies the end of the stream (e.g. TCP FIN or RST), that means we have the entire stream
    - Time to inspect its contents and try to match with a known hash
6. Once the stream is inspected, we don't need to continue tracking it
    - Delete it to free up memory
    - Also, delete the stream in case a new stream is started with the same source and destination ports

Part A Instructions

1. Initialise a `Stream` class
    - This class will be used for track each TCP stream
2. This class should have 2 attributes - `stream_id` and `data`
    - `data` will track the data in the stream

3. Add a method, `process_packet` to the class
    - This method should handle the adding of packets to `data`.
    - It should also stop adding packets once it receives an end of stream packet
    - Consider also printing a message whenever you start/stop tracking a stream
4. Now add a new function to main the does the following:
    - Whenever a TCP SYN packet arrives, create a new stream object and add it to a tracking list/dict
    - Track the stream's contents until the stream receives an end of stream packets
5. Verify your code works, and can track TCP streams properly

Part B Instructions

1. Now, once a tracked stream ends, its body should be checked for malware
2. Extract the HTTP body by splitting the body and the headers
    - Hint: Which characters indicate the border between the headers and the body?
3. Next, hash the body using SHA-1, and compare it with the provided list of known malware hashes
4. If the stream's body contains malware, print an alert indicating the stream id, and the malware contained
    - e.g., `*ALERT* Malware detected in HTTP download! Client: 10.0.99.7; Server: 10.0.99.10; Hash: 63427ee7f405ea2c818ddf55745cd9fb9e336133 (TeslaCrypt)`
5. Once a stream has been checked, you can delete it
    - This is so we can reuse the stream id if a new stream is started

Notes

KNOWN*MALWARE*HASHES

- **Lokibot**
  - `b3f067e63fff8e171ee26bcde6a6010737c8b22c`
  - `73480f2548244fb6a3e9db83a4e74082dd2fa500`
- **TeslaCrypt**
  - `efbd4555c4b881d77d28f659289373a813e79650`
  - `13427e27f405ea2c818d4f55745cd9fb9e336134`

> **Note:** Exercises may involve one or both malware variants—ensure coverage for both. These are the most updated hashes that you should follow.

GUIDING QUESTIONS

Use the guiding questions below to identify the malicious trigger and structure your code's detection:

- What are the characteristics of the target packets to be parsed and tracked?
  - What layers are present in these packets?
  - Are we expecting it to come from a specific port?
- How can we keep track of a UNIQUE connection (src*ip:src*port -> dst*ip:dst*port) and the packets involved in that connection?
  - If you are struggling with implementing classes, what other data types have a UNIQUE identifier or key that links to specific data?
- How do we identify the end of a TCP stream?
  - Are there specific characteristics of the final packets e.g. flags, keywords, etc.?

- How do we extract the data from the packets in the TCP stream?
    - Is there a specific packet layer with the data we want?
    - Are there any extra information that we do not need e.g. protocol headers, metadata, etc.?
- How do we hash the data for comparison?
    - What hashing algorithm do we use?
    - What library can we use for hashing?

To submit

Submit a ZIP `nids.zip` containing all `.py` files.