REPORT

IDS/IPS Program in C++

GROUP#10

GROUP MEMBERS

- *MUHAMMAD-ASIM 242385*
- *NOOR-UL-HASSAN 242379*
- *M.HASSAN MEHDI 242347*
- *SYED M.Zain Ul Arifeen 242317*

Contents

Project Report: IDS/IPS Program

1. Group Members:

1. Noor Ul Hassan

2. Muhammad Asim

3. Muhammad Hassan Mehndi

4. Syed Muhammad Zain Ul Arifeen

2. Task Allocation Among Team Members:

Group Members	Tasks
Noor Ul Hassan	1. Created a GitHub Repository for a combined work. 2. Implement `processTraffic()` function for traffic processing based on rules. 3. Log appropriate actions in the output file based on rule matching.
Muhammad Asim	1. Define and implement `TrafficPacket` structure. 2. Implement `parseTraffic()` function to extract data from traffic lines.
M. Hassan Mehndi	1. Define and implement `Rule` structure. 2. Implement `parseRule()` function to extract rule component
Syed M. Zain Ul Arifeen	1. Integrate all components into the `main.cpp` file. 2. Handle file input/output operations and coordinate the program flow. 3. Implement error handling for file operations and invalid formats.

3. HighLevel Design of the Solution (Program Flow):

The program flow consists of the following functions:

1. main():

Initializes file streams for input traffic and output log. Reads each line from the traffic file and calls **parseTraffic()** to convert the line into a TrafficPacket. Calls **processTraffic()** to match the traffic packet with rules and logs the appropriate action.

2. parseTraffic():

Takes in a line from traffic, processes it to pull packet info out in TrafficPacket format, namely, source IP, source port, destination IP, destination port, protocol, and data.

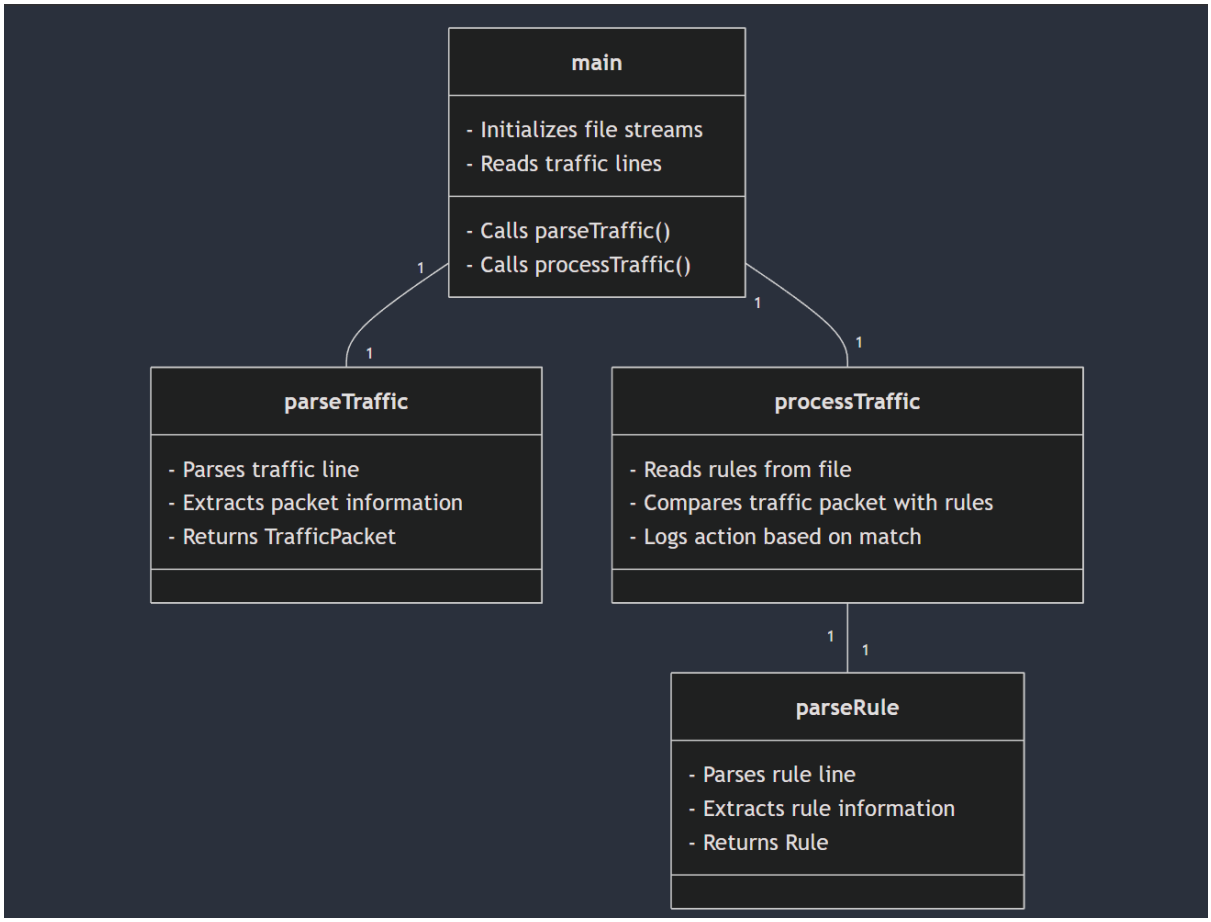
3. parseRule():

Processes a rule from a line and retrieves action, protocol, source IP, source port, destination IP, destination port, message, and SID.

4. processTraffic():

It reads rules from the file "rules.txt".Compares the parsed traffic packet with each rule. Logs the action taken in case a rule matches as ALLOW, DENY or PASS.

Program Flowchart (Function Calls):



4. Problems Faced in Implementing the Project and How We Resolved Them:

1. File Parsing Issues:

Problem: Parsing the traffic and rule files with various formats gave errors, more so when the line format did not make sense.

Solution: We used **strstr()** to determine the positions of the key fields in the line, and thus ensured that the right fields were being extracted using **sscanf().** This also had the effect of skipping over lines with missing or incorrect fields.

2. Complexity of Rule Matching:

Problem: The complexity of rule matching with multiple fields such as IPs, ports, and protocols was a bit of an issue.

Solution: We standardized the rule fields (e.g., we used ""any" for the wildcards) and coded conditional checks for each of the fields in the processTraffic() function. Then, we were able to compare each packet of the traffic against the rules quite accurately.

3. Logging Format:

Issue: The log file to be correctly formatted with meaningful output for both matched and unmatched traffic.

Solution: We came up with a uniform logging structure so that both matched and unmatched traffic results with PASS action were clearly written to the log file and hence easy to interpret.

4. File Handling Errors:

Problem: Handling the cases where files could not be opened or lines had invalid formats.

Solution: We added appropriate error handling in the main() function, ensuring files were opened successfully before processing further. If the traffic format was bad, the program logged an appropriate error message.

5. Project Limitations:

1. Fewer Rule Types:

The project currently supports only basic traffic rules based on IPs, ports, and protocols. It does not support more advanced rule configurations, such as stateful inspection or advanced matching criteria (for example, ranges of IPs or ports).

2. Error Handling for Traffic Data:

Although the invalid traffic formats are logged, it does not give any more specific information about why a line of traffic is invalid (e.g., missing fields, wrong data formats). More detailed error messages may be added to improve this.

3. Scalability:

This solution processes traffic line by line, which might be inefficient for a big log of traffic. This kind of scenario would definitely improve if an optimized data structure or parallel processing was done in hightraffic situations.

4. No RealTime Traffic Processing:

This project is actually supposed to process static traffic from a file. The handling of realtime capture and processing of traffic cannot be done, which may help in live network monitoring.

6. Conclusion:

This project shows the ability to process traffic data according to predefined rules and log the resulting actions. The solution is modular and wellstructured, with a clear division of tasks among team members. However, while the project works fine within its scope, there are areas such as scalability and realtime processing that could be improved for more advanced implementations.