FIREWALL

Tutor

Graphical User Interface

19thMay,2024

By:

Iqra Noor(232113)

CYS(eve)II-A

Submitted to: Dr. Muhammad Imran



DEPARTMENT OF

**CYBER SECURITY**

Air University, Islamabad

Eman Naseer(232143)

Fizza Afzal(232959)

z

---

**Table of Contents:**

1. Introduction
2. Implementation
3. Designing
   1. Class Description and Hierarchies
   2. Program Flow
   3. User Manual
4. Testing
5. Challenges Faced
6. Lessons Learned
7. Suggestions for Improvement
8. Conclusion
9. Task Divison

**GUI-based Firewall Tutor**

**1. Introduction**

**1.1 Packet Filtering Firewall**

A packet filtering firewall controls the flow of data to and from a network based on predetermined security rules.

**1.2 How It Works**

A packet filtering firewall works by inspecting the headers of data packets and making decisions to allow or block them based on criteria such as source and destination IP addresses, port numbers, and the protocol used.

* **Inspection of Packets:** When a packet arrives at the firewall, it inspects the packet's header information. The header includes details such as the source and destination IP addresses, source and destination port numbers, and the protocol type (e.g., TCP, UDP, ICMP).
* **Rules Evaluation:** The firewall has a set of predefined rules, often created by network administrators. These rules specify which packets are allowed to pass through and which are to be blocked. The rules can be based on:
* IP Addresses: Allowing or blocking traffic based on the source or destination IP address.
* Port Numbers: Allowing or blocking traffic based on the source or destination port number.
* Protocols: Filtering based on the type of protocol used (e.g., allowing HTTP traffic but blocking FTP).
* **Decision Making:** The firewall compares the packet header information with its rules. If the packet matches a rule that allows it, the packet is permitted to pass through. If it matches a rule that blocks it, the packet is discarded.
* **Logging and Alerts:** Many packet filtering firewalls also have logging capabilities, recording information about packets that are allowed or blocked. This logging helps in monitoring and troubleshooting network security.

**2. Language**

We used C# for this project. C# is a versatile, powerful, and modern programming language. Its strong type system, comprehensive standard library, and support for modern programming paradigms make it a popular choice for a wide range of applications, from desktop and web development to mobile and game development.

**3. Designing**

**3.1 Class Description and Hierarchies**

**Rule Class**

* Purpose: Represents a firewall rule.
* Attributes:
  + Source IP: The source IP address condition.
  + Destination IP: The destination IP address condition.
  + Source Port: The source port condition.
  + Destination Port: The destination port condition.
  + Protocol: The protocol condition (e.g., TCP, UDP).
  + Decision: Allowed/Denied
* Methods:
* ToString(): Converts user input into a string.

**Packet Class**

* + Purpose: Represents a network packet.
  + Attributes:
  + Source IP: The source IP address of the packet.
  + Destination IP: The destination IP address of the packet.
  + Source Port: The source port of the packet.
  + Destination Port: The destination port of the packet.
  + Protocol: The protocol of the packet (e.g., TCP, UDP).
  + Data: The data content of the packet.
  + Timestamp: The timestamp when the packet is created.
* Methods:
  + ToString(): Converts user input into a string.

**Firewall Class**

* Purpose: Contains the main methods of the firewall.
* Methods:
  + AddRule(Rule rule)
  + List<Rule> LoadRulesFromFile(string filePath): Loads rules from a rule file to temporary storage in a container.
  + List<Packet> LoadPacketsFromFile(string filePath): Loads packets from a packet file to temporary storage in a container.
  + ApplyRules(Packet packet, List<Rule> rules): Applies rules to packets.
  + CompareIP(string ruleIP, string packetIP): Compares IP addresses for any subnet, wildcard, or exact match.
  + uint IPToUInt32(string ipAddress): Converts IP addresses into binary.

**FormRules Class**

* + Purpose: The user interface of the application where users enter rules.
  + Methods:
  + SaveRuleBtn\_Click(object sender, EventArgs e): Adds a new rule to the rules file.
  + ViewRuleBtn\_Click(object sender, EventArgs e): Shows rules in a list box from the rules file.
  + ResetBtn\_Click(object sender, EventArgs e): Clears the rules file and list box.
  + ClearFields(): Clears text boxes after entering a rule.

**FormPackets Class**

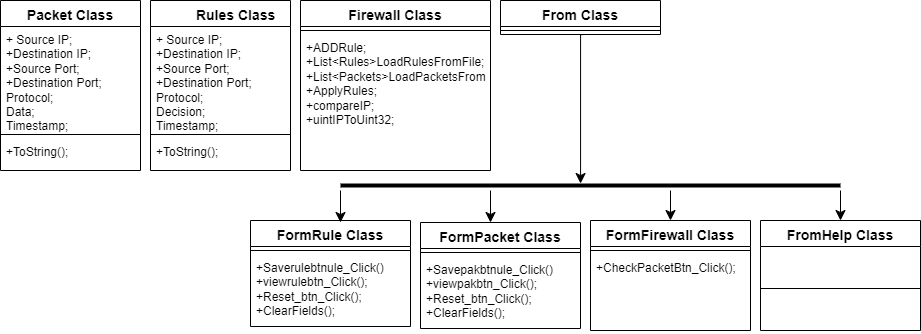
* + Purpose: The user interface of the application where users enter packets.
  + Methods:
  + SavePacketBtn\_Click(object sender, EventArgs e): Adds a new packet to the packet file.
  + ViewPacketBtn\_Click(object sender, EventArgs e): Shows packets in a list box from the packets file.
  + ResetBtn\_Click(object sender, EventArgs e): Clears the packet file and list box.
  + ClearFields(): Clears text boxes after entering a packet.

**FormFirewall Class**

* + Purpose: The main user interface of the application.
  + Methods:
  + CheckPacketBtn\_Click(object sender, EventArgs e): Checks packets against the rules and displays results.
  + \*FormHelp Class\*
  + Purpose: The user interface of the application which explains how the firewall works.

**3.2 Program Flow**

* **User Interface Initialization:** The application starts and the main form is loaded. Menu content is displayed to the user.
* **Adding Firewall Rules:** The user enters firewall rule details (Source IP, Destination IP, etc.) and clicks the "Add Rule" button. The rule is added to the list and displayed in a ListBox through the "View" button with an option to reset previously entered rules.
* **Entering Packet Data:** The user enters packet details (Source IP, Destination IP, etc.) and clicks the "Add Packet" button. The packet is added to the file and displayed in a ListBox through the "View" button with an option to reset previously entered packets.
* **Firewall:** The user clicks the "Check" button to view the results in a list box.
* **Help:** This interface contains a guide on how a packet filtering firewall works.



**3.3 User Manual**

The user interface consists of the following components:

* + TextBoxes for entering rule and packet details (Source IP, Destination IP, Source Port, Destination Port, Protocol, Data).
  + Buttons for actions (Add Rule, Add Packet).
  + ListBox to display the list of rules.
  + RichTextBox or Label for displaying tutorial content and results.

**4. Implementation**

* Created classes for rules and packets.
* Added rules and packets handling logic.

**5. Testing**

**Testing Scenarios**

* **Adding Rules:**
  + Test adding valid rules and verify they appear in the list.
  + Test adding rules with missing or invalid data and ensure proper error handling.
* **Checking Packets:**
  + Test packet checking with different combinations of rules and packet data.
  + Verify that packets are correctly identified as allowed or denied based on the rules.
* **User Interface:**
  + Test the layout and usability of the user interface.
  + Ensure all controls are responsive and display the correct information.

**Testing Results**

* **Functionality:** All functionalities (adding rules, checking packets, displaying results) work as expected.
* **Usability:** The user interface is intuitive and easy to navigate. Users can easily add rules, enter packet data, and view results.
* **Performance:** The application performs well with a reasonable number of rules and packets. There is no noticeable delay in processing.

**6. Challenges Faced**

* **Data Validation**: Ensuring all user inputs are valid and correctly formatted. Handling invalid data gracefully without crashing the application.
* **Matching Logic** Implementing the matching logic for rules and packets, ensuring it is accurate and efficient.
* **User Interface Design:** Designing an intuitive and user-friendly interface that is easy to navigate and use.

**7. Lessons Learned**

* **User Input Handling:** Importance of validating and sanitizing user inputs to prevent errors and ensure data integrity.
* **Object-Oriented Design:** Designing classes and methods to encapsulate data and behavior effectively.
* **Event-Driven Programming:** Handling events and user interactions in a Windows Forms application.
* **Debugging and Testing:** Systematic debugging and testing to ensure the application functions correctly under various scenarios.

**8. Conclusion**

This project successfully created an interactive Windows Forms application in C# to teach users about firewall functionality. By allowing users to define firewall rules, input network packet data, and determine packet permissions based on these rules, the project provides a hands-on learning experience for understanding the concepts of Object-Oriented Programming, including classes, inheritance, and encapsulation.

**9. Task Division**

**Fizza Afzal:** GUI Designing

**IqraNoor:** Coding

**Eman Naseer:** Documentation and coding