Firewall Simulation with Packet Filtering

**Objective:** Simulate a firewall that filters network traffic based on rules.

**Description:** Implement a firewall simulator where you can set rules to allow or block traffic based on IP addresses, ports, and protocols. The tool will monitor packets and decide which to drop or pass through based on predefined rules.

Piyush Kumar

225891202

Cyber-A



# Abstract

The regulation of network traffic appears to be crucial for defense in the present interconnected digital world, essential for network security. This project simulates a simple firewall that can filter packets and allows for the formulation of rules to accept or deny certain packets based on specific attributes such as IP addresses, ports, and protocols. A lab-tested environment revealing or testing packet filtering mechanisms would be created by a simulation based on the core elements that firewalls operate on.

This simulator also makes it possible for the user to set customizable rules governing how network traffic is controlled by stating conditions in which data packets are allowed or denied. It is very much based on criteria like source and destination IP addresses, port numbers, and protocols like TCP, UDP that is supposed to mimic the core functionality of an actual firewall. It monitors both incoming and outgoing packets. According to the contents and rule criteria, it performs the right action: either forward the packet or drop, by carrying out defined rules.

This project evolves in packets that also handle rules besides the decision engine related to the packet verification against pre-defined rules. It further contributes to cyber education, simulating a hands-on experience related to firewall operation, and could be applied as a basis for more complex systems. In this simulation, users learn the principles of network security, about actions determining packet filtering, and how particular conditions for regulation are realized in accordance with predefined rules.

## Introduction

**Problem Statement: The Significance of Firewalls in Cybersecurity**

In the new digital world, organizations and individuals are making themselves more vulnerable day after day as attackers continually discover new ways to exploit the vulnerabilities of networks. The most fundamental defensive controls against the threats are firewalls, these operate like walls of protection that guard data, control, and moderate data flow while preventing unauthorized access into secure networks located at the edge or boundary of the network. These scan and filter packets along with rules, allowing only trusted data to pass. This ability is critical in preventing hostile traffic from penetrating sensitive systems that could shut down operations.  
  
Firewalls are very important in cybersecurity as they can provide the management of control flow and implement network policies thereby minimizing risks about unauthorized access theft of data, and intrusion by malware. Packet filtering analyzes header inspections comprising IP addresses, protocols, and ports and filters according to the rules that can identify the data stream against which such threats are being sent. This scanning enables the firewall to block malicious packets at the periphery and proves to be an effective first-line defense mechanism. In today's heavy financial and reputational loss because of security breaches, firewalls remain one of the most critical parts of the puzzle for enterprise and personal network defenses.

**Problem Objectives:**

The goal of this project is the design of a firewall simulator, which allows for traffic filtration based on a customizable set of rules, so it will provide hands-on experience with the way firewalls are generally utilized. This simulator lets us define certain filtering conditions, like the source and destination IP addresses, port numbers, and also the protocols being used, such as TCP, UDP. The rules enforced by this simulator act like real firewalls. It verifies all packets to see if they match some criteria set down and then allows or blocks the packets. This simulation will better guide students on packet filtering techniques and principles of network security and, therefore, is a good tool for learning the basics of firewalls.

**Scope: Limitations and Expected Performance**

This is a conceptual model of a firewall simulator that uses packet filtering functionality but does not interface to live traffic. It only particularly inspects packets based on defined IP addresses, ports, and protocols. More advanced features such as stateful inspection or intrusion detection are not included in this model. Those are considered sophisticated features commonly found in full-functioning firewalls. It is thus constructed to function effectively, processing the rules, as well as making right decisions in relation to the traffic that should be filtered by the set of rules. The simulator is, however meant for learning purpose and simulates the core Firewall behavior in a controlled environment and not real-world protection and thus omits sophisticated, resource-intensive security measures such as deep packet inspection.

Literature Review

Firewalls have become imperative ingredients in cybersecurity. There are diverse types of firewalls whose aim is to bar wrong access and activities towards a system. Two forms of firewalls consist of technologies, and these include: packet-filtering firewalls and stateful inspection firewalls. The simplest forms of packet-filtering firewalls filter packets at the network layer based upon a predefined set of rules that regard various attributes such as source and destination IP addresses, ports, and protocol. The best thing about packet-filtering firewalls is that they are extremely resource-efficient and require less resource allocation, making them best for simple filtering but not so effective for detecting sophisticated attacks.

Stateful inspection firewalls are significantly more secure in that they keep track of active connections and only allow packets that correspond to known active sessions. This again has a larger resource demand but allows malicious patterns of traffic to be discovered over time. Proxies are application layer firewalls and work on the application layer for filtering of traffic either by content or protocol, which makes them efficient against threats that are implanted inside specific applications or protocols. Next Generation Firewalls comprises of several functionalities including intrusion prevention, deep packet inspection, and application awareness that offer complete protection for modern networks.

This packet-filtering firewall simulator, developed in this project, emulates very early generation packet inspection and filtering rules. It teaches the user on how firewalls work with simple rule-based traffic control, emulating the very early generations of packet-filtering techniques. More advanced features characteristic of the more modern firewalls like the NGFWs, such as stateful inspection, deep packet analysis are not being employed. Rather, it provides a variant that depends solely on rules with packet checking to help account for some of the very most elementary concepts-it obviously limits it to something that could not be matched by firewalls either in terms of protection and complexity, but is good at explaining principles like packet filtering and network security.

Methodology and Design

This firewall simulator project aims to design a simple proof-of-concept demonstration of the packet-filtering process in the firewall with rule-based decision-making. Main components include the packet filtering module, the rule manager, and the decision engine, all three of them being critical modules used to process incoming packets and decide on an appropriate response according to predefined rules. The simulation methodology can be implemented to break down simple basics of attributes for any type of packets, such as IP addresses, ports, and protocols, that can form a basic demonstration of the ability of a firewall to monitor and regulate network traffic.

1. **Architecture**

Architecture of the Firewall simulator consists of several major components followed:  
• **Rule Manager:** This manages how the rules may be created, stored, and retrieved. It can be uploaded manually or else through CSV uploading too, so it is highly flexible and user-friendly. Organized by the Rule Manager, these are the rules that filter traffic based on protocol, source/destination IP addresses, and port numbers.  
• **Packet Filter Module:** This is the core processing module that relies on the attribute attributes of incoming packets. In this case, the module works together with the rule manager to match the incoming packet with a predefined set of rules as to whether to allow or filter the packet.  
• **Decision Engine:** This is the logic rule engine that employs the meaning of rules for applying them on packet attributes. It will ensure that each packet is gauged in relation to all rules sequentially as if implementing appropriate action from the matched criteria. If any packet matches more than one rule, then the engine exercises the action of the first matched rule.

1. **Packet Structure**

The simulator processes packets based on the critical attributes necessary for filtering:  
Packets are processed based on critical attributes:  
Protocol: This establishes the type of protocol to be used, such as TCP, UDP, ICMP, etc. Such an aspect dictates how a given packet should be treated.  
• **Source IP Address:** The source IP address refers to the IP of the device that originated that packet. The firewall could then filter packets based on unwanted sources through looking at this field.  
• **Source Port:** The port number of the source. A port is used to identify application-layer data and provides finer control over traffic.  
• **Destination IP Address:** The IP address of the receiving device, so rules can be targeted for specific internal or external devices.  
• **Destination Port:** This is the port number of the receiving device, and it helps in defining the rules regarding the kind of service that is being accessed.

1. **Rule Definition**

Rules form the heart of the filtering criteria. They state what actions and conditions packets are allowed or blocked. Each rule is comprised of the following:  
• **Action:** Specifies, for the packets which the rule matches, whether they should be "ALLOWED" or "BLOCKED".  
• **Protocol:** If the rule applies to only a specific protocol (such as TCP, UDP, or ICMP), then this is defined. An asterisk (\*) can be used for this attribute to match all protocols.  
• **Source and Destination IP Address:** Some rules will affect a certain set of IP addresses or network. To do network-level filtering, CIDR notation is applied, while for wildcard substitution that accepts any IP address, \* can be used in lieu.  
• **Source and Destination Port:** Specifies special ports (such as 80 for HTTP), whereas \* denotes any port.  
Rules are stored as the Rule named tuples, so accessing and handling are not a problem. Rules can be written to or read from a CSV file, that is easy to handle rule sets between sessions.

1. **Decision Engine**

The Decision Engine filters as well as decides to pass or block the packet according to the predefined rules. Operations of the Decision Engine  
• **Packet Matching:** The Decision Engine compares each packet sequentially about its list of rules. If the packet's attributes match the conditions in the rule, it applies the action in the rule to the packet. The Decision Engine uses the \_match\_rule() function to perform matching: The function checks every packet attribute against the specifications set for that attribute in the rule.  
• **IP Matching:** The ip\_in\_network() helper function is used to check if the source or destination IP address of a packet is within the rules of a given network range for a rule. This enables proper filtering by network conditions.  
• **Protocol and Port Checks:** This tests for a match regarding the protocol number and port number associated with the packet against rule criteria. If the rule specifies that a protocol or port is present, then the packet must match; otherwise, any protocol or port is a wildcard match.  
• **Default Action:** The packet should be "BLOCK"ed if no matching rule is found. It is a conservative approach and thereby ensures that by default, only the access that explicitly happens from the allowed packets.

**Simulation of Packet Filtering:**

The function simulate\_packet() triggers packet filtering. It forwards packet attributes into the check\_packet() method to process packets and take appropriate actions according to the rule. For each packet, it prints a message showing its details and action taken to offer a very clear view of the decision making of a firewall:

This simulator employs a simple firewall model. This has a filtering functionality that is a good source of understanding packet filtering and rule-based security logic. Much more rudimentary than the advanced firewalls, it's as simple as most basic firewall concepts can be framed to exhibit a notion of basic firewall behaviors, applied to educational purposes to explore network security.

Implementation:

This Firewall Simulator Project is implemented using Python so it could easily handle packet filtering and rule enforcement through the power of standard libraries and built-in tools. The following sections then describe the technologies, source code, and sample configurations that make up the core functionality of the firewall simulator.

1. **Technologies Used:**

Using standard libraries, the firewall simulator will be lightweight and portable and easy to extend. Major libraries used in the process include:  
  
• **IP address**: Used when handling IP addresses and networks, that one can compare CIDR notation for IP ranges.  
• **csv:** Provides tools for reading and writing rules to CSV files, allowing for efficient rule management.  
• **collections.namedtuple:** It defines the Rule data structure. In its use, you store your firewall rules in an easily readable and accessible manner.  
  
These libraries enable one to work with IP addresses, handle files for persistence over rules, and hold data in structured forms, which gives them some critical functionalities with minimal external library dependencies.

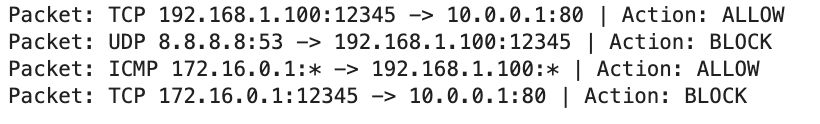
1. **Source Code**

The following annotated code snippets cover the core functionality of the firewall simulator, including packet parsing, rule setting, and traffic filtering.





**OUTPUT:**

****

* **Packet Parsing and Inspection**

**A screenshot of a computer program

Description automatically generated**

**simulate\_packet():** Accepts packet parameters and invokes check\_packet() to determine if the packet matches any rules. It prints the packet’s attributes and action, giving real-time feedback on packet handling.

**check\_packet():** Iterates through the rules to check if the packet matches any, returning “ALLOW” or “BLOCK” based on the first matching rule or defaulting to “BLOCK” if no match is found.

* **Rule Setting and Updating**

A screen shot of a computer

Description automatically generated

**add\_rule():** Creates a new rule as a Rule namedtuple and appends it to the rules list.

**load\_rules\_from\_csv():** Reads rules from a CSV file, parsing each row and calling add\_rule() to add them to the simulator’s rule list.

**save\_rules\_to\_csv():** Exports current rules to a CSV file, preserving the rule set between sessions.

* **Traffic Monitoring and Filtering**

**A screenshot of a computer

Description automatically generated**

**\_match\_rule():** Verifies if a packet’s attributes match a rule’s conditions. If any attribute fails to match, it returns False; otherwise, it returns True when all conditions are met.

**\_ip\_in\_network():** Checks if an IP address is within a specified CIDR network, enabling IP range matching.

1. **Sample Rules**

The following sample configurations demonstrate common rule settings for packet filtering:

1. **Block All UDP Traffic to Port 53 (DNS)**:



This rule blocks all UDP traffic to port 53, which is typically associated with DNS requests.

1. **Allow All TCP Traffic from Local Network to Port 80 (HTTP)**:



This rule allows all TCP traffic from the local network (192.168.1.0/24) to access HTTP servers on port 80.

1. **Block Traffic from a Specific IP**:



This rule blocks all traffic originating from the IP address 203.0.113.5, irrespective of the protocol or destination.

1. **Allow ICMP Traffic for Network Diagnostics**:



This rule allows ICMP traffic (e.g., pings), which is useful for network diagnostics and connectivity testing.

* These sample rules showcase the flexibility of the firewall simulator, enabling a range of configurations to block or permit traffic based on IP ranges, protocols, and ports.

Testing and Evaluation

Testing is crucial for the firewall simulator project because it ensures that the implemented rules behave correctly, and packets are filtered accurately. The section below details the implemented test cases for the different rule types and summarizes the observed performance and accuracy of the tests.

1. **Test Scenarios:**
   1. **Allow Rule for TCP Traffic**

**Scenario:** Test TCP packets from a specific source network to a permitted destination port.

**Packet:**

**• Protocol:** TCP

**• Source IP:** 192.168.1.100

**• Source Port:** 12345

**• Destination IP:** 10.0.0.1

**• Destination Port:** 80

**Expected Outcome:** The packet should be allowed based on the rule permitting TCP traffic from the local network (192.168.1.0/24) to destination port 80.

* 1. **Block Rule for UDP Traffic**

**Scenario:** Test UDP packets destined for a blocked port (DNS).

**Packet:**

**• Protocol:** UDP

**• Source IP:** 8.8.8.8

**• Source Port:** 53

**• Destination IP:** 192.168.1.100

**• Destination Port:** 12345

**Expected Outcome**: The packet should be blocked due to the rule preventing UDP traffic to port 53.

* 1. **Allow Rule for ICMP Traffic**

**Scenario:** Test ICMP echo request (ping) packets.

**Packet:**

**• Protocol:** ICMP

**• Source IP:** 172.16.0.1

**• Source Port:** \*

**• Destination IP:** 192.168.1.100

**• Destination Port:** \*

**Expected Outcome:** The packet should be allowed as per the rule permitting all ICMP traffic.

* 1. **Block Specific IP Address**

**Scenario:** Test traffic from a blocked IP.

**Packet:**

**• Protocol:** TCP

**• Source IP:** 203.0.113.5

**• Source Port:** 54321

**• Destination IP:** 192.168.1.100

**• Destination Port:** 80

**Expected Outcome**: The packet should be blocked as it originates from the IP address 203.0.113.5, which is explicitly blocked.

1. **Results**

From the testing of the Firewall Simulator, the following observations in terms of performance and accuracy could be drawn:

**• Accuracy:** The simulator matched and applied all the tested cases correctly. It showed high accuracy in packet filtering. The simulator analyzed the packet against defined rules and executed appropriate actions based on the first matching rule that met their expected outcome.

**• Performance:** The simulation operated with almost no delay in packet processing. Even in a batch of packets, rule matching and decision-making processes were performed promptly, thereby establishing that the implementation can support several packets evaluation calls made simultaneously without experiencing undue delay.

**• Robustness:** The firewall correctly passed wildcard entries in the rules, which thus allowed for broader matching criteria without any loss of security in doing so. Such flexibility was particularly handy when specific ports or IP addresses were not predefined to respond to messages in the environment.

**• Boundary Testing**: These were then subjected to boundary conditions-tests on cases where IP addresses are tested along the boundary of a CIDR. The input was tested against instances wherein the \_ip\_in\_network() function correctly evaluates whether the address falls within the network.

Conclusion

The development and testing of the firewall simulator gave insights into the complexities of filtering network traffic and how firewalls play a very crucial role in any network's security system. Major findings from this project are:

• **Effective Rule Implementation**: Such a set of variants of rules (ALLOW, BLOCK) for different protocols (TCP, UDP, ICMP) was placed in the simulator. Testing scenarios proved the correctness of the packet to match the rules defined, thus confirming the basic functionality of a packet filtering firewall.

• **Accurate Packet Inspection**: The \_match\_rule() function correctly matched packets against the rules and reliably determined the outcome based on specific criteria, including IP addresses, ports, and protocols. There was also the option to use wildcard entries, which made it possible to have increased flexibility when defining rules thereby greatly enhancing the utility of the simulator.

• **Performance and Efficiency**: The tester performed with good results and processed the packets with minimum latency. Boundary testing further proved the correctness of the \_ip\_in\_network() function since it can be applied to test, among others, CIDR edge range cases.

• **Challenges Faced**: Developmental Issues There were a few developmental issues that were faced. These include manipulating and ensuring that evaluations made by IP range are correct. In addition to such considerations, the balancing simplicity of user interface against the power of rule configuration was challenging.