Advanced Firewall Application Documentation

Overview

The Advanced Firewall Application is designed to filter network packets based on IP addresses, ports, and protocols. It uses the Singleton, Strategy, and Factory design patterns to ensure flexibility and maintainability.

1. Components

1.1 Singleton Pattern: `Settings` Class

The `Settings` class ensures that only one instance of the class exists. This instance holds the default policy for the firewall.

```
class Settings:
    _instance = None

    def __new__(cls):
        if cls._instance is None:
            cls._instance = super(Settings, cls).__new__(cls)
            cls._instance.default_policy = "allow"
        return cls._instance

    def get_default_policy(self):
        return self.default_policy
```

1.2 Strategy Pattern: `FirewallStrategy` Abstract Base Class

The `FirewallStrategy` abstract base class defines the interface for all filter strategies.

```
from abc import ABC, abstractmethod

class FirewallStrategy(ABC):
    @abstractmethod
    def filter(self, packet: dict) -> bool:
        pass
```

1.3 Concrete Strategies

1.3.1 `IPFilter` Class

Filters packets based on IP addresses using whitelists and blacklists.

```
python
Copy code
class IPFilter(FirewallStrategy):
    def __init__(self):
```

```
        self.whitelist = set()
        self.blacklist = set()

    def add_to_whitelist(self, ip):
        self.whitelist.add(ip)

    def add_to_blacklist(self, ip):
        self.blacklist.add(ip)

    def filter(self, packet: dict) -> bool:
        ip = packet.get("ip")
        if ip in self.blacklist:
            return False
        if self.whitelist and ip not in self.whitelist:
            return False
        return True
```

1.3.2 `PortFilter` Class

Filters packets based on allowed ports.

```
class PortFilter(FirewallStrategy):
    def __init__(self):
        self.allowed_ports = set()

    def allow_port(self, port):
        self.allowed_ports.add(port)

    def filter(self, packet: dict) -> bool:
        port = packet.get("port")
        if self.allowed_ports and port not in self.allowed_ports:
            return False
        return True
```

1.3.3 `ProtocolFilter` Class

Filters packets based on allowed protocols.

```
class ProtocolFilter(FirewallStrategy):
    def __init__(self):
        self.allowed_protocols = set()

    def allow_protocol(self, protocol):
        self.allowed_protocols.add(protocol)

    def filter(self, packet: dict) -> bool:
        protocol = packet.get("protocol")
        if self.allowed_protocols and protocol not in
self.allowed_protocols:
            return False
        return True
```

1.4 Factory Pattern: `StrategyFactory` Class

The `StrategyFactory` class provides a method to instantiate different filter strategies.

```
class StrategyFactory:
```

```
    @staticmethod
    def get_strategy(strategy_type: str):
        if strategy_type == "ip_filter":
            return IPFilter()
        elif strategy_type == "port_filter":
            return PortFilter()
        elif strategy_type == "protocol_filter":
            return ProtocolFilter()
        else:
            raise ValueError(f"Unknown strategy type: {strategy_type}")
```

2. Main Script

The main script sets up the firewall filters and processes packet inputs from the user.

```
def main():
    import re

    def is_valid_ip(ip):
        pattern = re.compile(r'^(\d{1,3}\.){3}\d{1,3}$')
        return pattern.match(ip) is not None

    settings = Settings()
    default_policy = settings.get_default_policy()

    ip_filter = StrategyFactory.get_strategy("ip_filter")
    port_filter = StrategyFactory.get_strategy("port_filter")
    protocol_filter = StrategyFactory.get_strategy("protocol_filter")

    ip_filter.add_to_whitelist("192.168.1.1")
    ip_filter.add_to_blacklist("10.0.0.1")

    port_filter.allow_port(80)
    port_filter.allow_port(443)

    protocol_filter.allow_protocol("TCP")
    protocol_filter.allow_protocol("UDP")

    allowed_protocols = {"TCP", "UDP"}

    print("Advanced Firewall is running...")
    print("Enter packet details in the format 'ip, port, protocol' (e.g.,
'192.168.1.1, 80, TCP')")
    while True:
        packet_input = input("Enter packet details (or type 'exit' to
quit): ")
        if packet_input.lower() == 'exit':
            break
        try:
            errors = []
            parts = packet_input.split(', ')
            if len(parts) != 3:
                errors.append("Invalid packet format. Please enter the
details in the format 'ip, port, protocol' (e.g., '192.168.1.1, 80,
TCP').")
            else:
                ip, port, protocol = parts

                if not is_valid_ip(ip):
```

```
                    errors.append("Invalid IP format. Please enter a valid
IP address (e.g., 192.168.1.1).")

                try:
                    port = int(port)
                except ValueError:
                    errors.append("Invalid port number. Please enter a
numeric port value.")

                if protocol not in allowed_protocols:
                    errors.append(f"Invalid protocol. Please enter a valid
protocol (e.g., {', '.join(allowed_protocols)}).")

            if errors:
                for error in errors:
                    print(error)
                continue

            packet = {
                "ip": ip,
                "port": port,
                "protocol": protocol
            }

            if ip_filter.filter(packet) and port_filter.filter(packet) and
protocol_filter.filter(packet):
                print("Packet allowed")
            else:
                print("Packet denied")
        except Exception as e:
            print("An unexpected error occurred:", e)

if __name__ == "__main__":
    main()
```

3. Web Interface

3.1 HTML and CSS

The web interface allows users to input packet details and see if the packet is allowed or denied.

```
<!DOCTYPE html>

<html>

<head>

   <title>Firewall Packet Checker</title>

   <style>

      body {

         font-family: Arial, sans-serif;
```

```
    background-color: #f2f2f2;
    margin: 0;
    padding: 20px;
}
h1 {
    color: #333;
    text-align: center;
}
form {
    background-color: #fff;
    border-radius: 8px;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
    max-width: 400px;
    margin: 0 auto;
    padding: 20px;
}
label {
    display: block;
    margin-bottom: 8px;
    color: #555;
}
input[type="text"] {
    width: 100%;
    padding: 10px;
    margin-bottom: 20px;
```

```
            border: 1px solid #ccc;
            border-radius: 4px;
        }
        button {
            display: block;
            width: 100%;
            padding: 10px;
            background-color: #007BFF;
            color: #fff;
            border: none;
            border-radius: 4px;
            cursor: pointer;
            font-size: 16px;
        }
        button:hover {
            background-color: #0056b3;
        }
        #result {
            margin-top: 20px;
            text-align: center;
            font-size: 16px;
            color: #333;
        }
    </style>
</head>
```

```
<body>
    <h1>Firewall Packet Checker</h1>
    <form id="packetForm">
        <label for="ip">IP Address:</label>
        <input type="text" id="ip" name="ip">
        <label for="port">Port Number:</label>
        <input type="text" id="port" name="port">
        <label for="protocol">Protocol:</label>
        <input type="text" id="protocol" name="protocol">
        <button type="button" onclick="checkPacket()">Check</button>
    </form>
    <div id="result"></div>

    <script>
        async function checkPacket() {
            const ip = document.getElementById('ip').value;
            const port = document.getElementById('port').value;
            const protocol = document.getElementById('protocol').value;

            const response = await fetch('/check', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify({ ip, port, protocol })
```

```
        });


        const result = await response.json();

        document.getElementById('result').innerText = result.message;

    }

  </script>

</body>

</html>
```

4. Usage

1. **Run the Python Script:**

   Execute the Python script to start the firewall. The script will will give you an html link then click it

   Web Interface:

   Open the HTML link in a web browser. Enter the IP address, port, and protocol of the packet to check if it is allowed or denied by the firewall.

Conclusion

The Advanced Firewall Application provides a robust and flexible framework for filtering network packets based on customizable criteria. By using design patterns, the application ensures maintainability and ease of extension for future requirements.