CareerTiQ
CHOOSE THE BEST

Title: Advanced Firewall Rule Optimization using Genetic Algorithm

Project Summary:

The "Advanced Firewall Rule Optimization Using Genetic Algorithm" project focuses on optimizing firewall rules to enhance both efficiency and security. Firewalls play a crucial role in protecting networks by filtering traffic based on predefined security rules. However, as network traffic grows in complexity and volume, the management of firewall rules becomes increasingly challenging. Redundant and conflicting rules can accumulate, leading to performance degradation and potential security vulnerabilities. This project leverages the power of genetic algorithms to tackle these issues by optimizing the given set of firewall rules.

Genetic algorithms, inspired by the process of natural selection, are well-suited for optimization problems. They work by generating an initial population of potential solutions, in this case, various configurations of firewall rules. Each solution is evaluated using a fitness function that considers factors such as rule redundancy, processing efficiency, and overall security effectiveness. Through iterative processes involving selection, crossover, and mutation, the algorithm refines these solutions over successive generations, gradually improving the rule set's quality.

The primary objectives of this project include reducing rule redundancy, enhancing the firewall's performance by optimizing rule order and structure, and ensuring that security is maintained or even improved. The optimization process is automated, allowing for efficient handling of complex rule sets without manual intervention. The implementation involves several stages: data collection and preprocessing, development of the genetic algorithm, execution of the optimization process, and thorough evaluation of the results.

Table Of Contents:

Problem Statement :

In modern network environments, managing and optimizing firewall rules is a significant challenge due to the increasing complexity and volume of network traffic. Firewalls are the first line of defense in securing networks, as they filter incoming and outgoing traffic based on a set of predefined rules. However, over time, these rule sets can become bloated with redundant, conflicting, and outdated rules, leading to several critical issues:

Performance Degradation: As the number of firewall rules increases, the time required to process each packet also increases. This can result in slower network performance and reduced throughput, adversely affecting the overall efficiency of network operations.

Rule Redundancy and Conflicts: Large rule sets often contain redundant or conflicting rules, which can complicate rule management and reduce the effectiveness of the firewall. These issues can cause unnecessary processing overhead and make it difficult to ensure that the firewall behaves as intended.

Security Vulnerabilities: Mismanaged and overly complex rule sets can create security gaps. Redundant and conflicting rules can inadvertently allow unauthorized traffic, leading to potential security breaches and network intrusions.

Maintenance Complexity: Maintaining a large and complex rule set is a labor-intensive task that requires continuous monitoring and manual updates. This complexity increases the likelihood of human error, further compromising network security and performance.

To address these issues, our project proposes the use of advanced firewall rule optimization using genetic algorithms. Genetic algorithms are well-suited for optimization problems due to their ability to search large solution spaces efficiently and evolve solutions over successive generations. By applying genetic algorithms to firewall rule optimization, we can:

Reduce Redundancy: Identify and eliminate redundant rules, streamlining the rule set for more efficient processing.
Resolve Conflicts: Detect and resolve conflicting rules, ensuring that the firewall operates according to intended security policies.
Enhance Performance: Reorder and optimize rules to reduce processing time.

Introduction :

In today's digital landscape, network security is of paramount importance. As organizations rely more on interconnected systems and online operations, the need to safeguard these networks from unauthorized access and potential threats has never been greater. Firewalls play a crucial role in this defense mechanism by monitoring and controlling incoming and outgoing network traffic based on predefined security rules. However, as network environments grow in complexity and traffic volume, managing and optimizing these firewall rules becomes increasingly challenging.

The efficiency and effectiveness of a firewall largely depend on the configuration and optimization of its rule set. Over time, firewall rule sets can become extensive and convoluted, containing redundant, conflicting, or obsolete rules. This not only hampers the performance of the firewall, leading to slower processing times and decreased network throughput, but also introduces security vulnerabilities that can be exploited by malicious actors. Furthermore, maintaining such complex rule sets demands significant manual effort and is prone to human error. To address these pressing issues, our project explores the application of genetic algorithms to optimize firewall rules.

Genetic algorithms, inspired by the principles of natural selection and evolution, are powerful tools for solving complex optimization problems. They work by generating a population of potential solutions and iteratively improving them through processes such as selection, crossover, and mutation. This approach allows for the efficient exploration of a vast solution space, ultimately converging on an optimal or near-optimal solution. By leveraging genetic algorithms, we seek to:

- Minimize Rule Redundancy: Eliminating unnecessary and duplicate rules to create a leaner rule set.

- Improve Processing Performance: Reordering and optimizing rules to enhance the speed and efficiency of firewall operations.

- Maintain or Enhance Security: Ensuring that the optimized rule set continues to provide strong protection against unauthorized access and network threats.

- Automate the Optimization Process: Developing an automated system that can efficiently optimize firewall rules with minimal human intervention.

- The "Advanced Firewall Rule Optimization Using Genetic Algorithm" project aims to enhance the performance and security of firewall systems by automating the optimization process. This not only streamlines the firewall rule set but also ensures that the firewall operates with maximum effectiveness, providing robust protection against network threats. Through this project, we demonstrate the potential of genetic algorithms to revolutionize firewall management, offering a scalable and effective solution to the challenges posed by complex and dynamic network environments. By automating the optimization of firewall rules, we contribute to more secure, efficient, and manageable network security systems.

Project Scope :

The scope of the "Advanced Firewall Rule Optimization Using Genetic Algorithm" project encompasses the following key areas and activities:

Analysis of Existing Firewall Rule Sets:

- Evaluate current firewall configurations to identify issues such as redundancy, conflicts, and inefficiencies.
- Understand the specific requirements and constraints of the network environments where the firewall rules will be optimized.

Development of the Genetic Algorithm:

- Design and implement a genetic algorithm tailored for firewall rule optimization.
- Develop components for initial population generation, fitness function evaluation, selection, crossover, and mutation processes.
- Ensure the algorithm can handle large and complex rule sets effectively.

Implementation of the Optimization Process:

- Integrate the genetic algorithm with the firewall rule management system.
- Automate the optimization process to reduce manual intervention and improve efficiency.
- Ensure the system can continuously adapt and optimize as new rules are added or existing rules are modified.

Performance and Security Testing:

- Conduct thorough testing to validate the effectiveness of the optimized rule sets.
- Measure improvements in processing speed, rule set conciseness, and overall network performance.
- Ensure that security is maintained or enhanced, with no new vulnerabilities introduced by the optimization process.

User Interface and Reporting:

- Develop a user-friendly interface for managing the optimization process and viewing results
- Provide detailed reports on the optimization outcomes, including before-and-after comparisons of rule sets, performance metrics, and security assessments.

Documentation and Training:

- Create comprehensive documentation for the genetic algorithm, optimization process, and user interface.
- Provide training materials and sessions for network administrators to ensure they can effectively use the system.

Future Enhancements:

- Plan for future improvements and extensions, such as incorporating machine learning techniques for adaptive optimization.
- Explore the potential for applying the genetic algorithm approach to other areas of network security management.
- By defining these areas, the project scope ensures a comprehensive approach to optimizing firewall rules using genetic algorithms, addressing both performance and security challenges. This scope outlines the necessary steps to develop, implement, and validate the optimization system, providing a clear roadmap for achieving the project’s objectives and delivering a robust solution for modern network environments.

Project Objectives :

The "Advanced Firewall Rule Optimization Using Genetic Algorithm" project aims to achieve several critical objectives:

Minimize Rule Redundancy: Eliminate unnecessary and duplicate rules within the firewall rule set, resulting in a more concise and streamlined rule set that reduces processing overhead and simplifies rule management.

Improve Processing Performance: Optimize the order and structure of firewall rules to enhance processing speed, leading to faster packet processing times and improved network throughput and overall performance.

Resolve Rule Conflicts: Detect and resolve conflicting rules that may cause security gaps or processing inefficiencies, resulting in a more coherent and effective rule set that operates according to intended security policies.

Enhance Security: Ensure that the optimized rule set maintains or improves the security posture of the network, providing robust protection against unauthorized access and network threats with no new vulnerabilities introduced by the optimization process.

Automate the Optimization Process: Develop an automated system for optimizing firewall rules using genetic algorithms, reducing manual effort in managing and optimizing rules and allowing for continuous and efficient optimization.

Develop a Robust Genetic Algorithm: Design and implement a genetic algorithm tailored specifically for firewall rule optimization, creating an effective algorithm that can handle large and complex rule sets and evolve solutions over successive generations for optimal results.

Provide Detailed Reporting and Analysis: Create a system that offers comprehensive reports and analyses of the optimization process and outcomes, providing clear visibility into the improvements made with before-and-after comparisons of rule sets, performance metrics, and security assessments.

Ensure Scalability and Adaptability: Design the optimization system to be scalable and adaptable to various network environments and evolving rule sets, resulting in a versatile solution that can be applied to different firewall configurations and adapt to changes over time.

Deliver User-Friendly Interfaces: Develop an intuitive user interface for managing the optimization process and viewing results, enhancing usability for network administrators and making it easier to monitor and control the optimization process.

Plan for Future Enhancements: Lay the groundwork for future improvements, such as integrating machine learning for adaptive optimization, creating a forward-looking system that can evolve with technological advancements and continue to offer cutting-edge optimization solutions.

These objectives guide the project towards its ultimate goal of revolutionizing firewall rule management through the use of genetic algorithms, providing a scalable, efficient, and secure solution for modern network environments.

Requirements:

Hardware Requirements :

To ensure smooth development and execution of the "Advanced Firewall Rule Optimization Using Genetic Algorithm" project, the following hardware components are necessary:

Development Machines:

Processor: Multi-core processor (Intel i5 or equivalent)

RAM: At least 16 GB for efficient processing

Storage: Minimum 500 GB SSD for fast read/write operations

Graphics Card: Optional, but a dedicated GPU (NVIDIA or AMD) can be beneficial for parallel processing tasks

Network Interface Card (NIC): High-speed NIC for network simulation and testing

Test Network Environment:

- Switches and Routers: Managed switches and routers to create a simulated network environment
- Firewalls: Hardware or virtual firewalls to apply and test optimized rules
- Network Cables: High-quality Ethernet cables for connecting devices in the test network

Backup and Storage Solutions:

- External Hard Drives or NAS: For backup and storage of project data and versions

Additional Peripherals:

- Monitors: Multiple monitors for efficient multitasking
- UPS: Uninterruptible Power Supply to prevent data loss during power outages

Software Requirements :

The following software components are essential for the development, testing, and deployment of the project:

Operating System:

- Development Machines: Windows 10/11, macOS, or Linux (Ubuntu, CentOS)
- Test Network Devices: Compatible firmware or operating systems for routers, switches, and firewalls

Development Tools and IDEs:

- Integrated Development Environment (IDE): IntelliJ IDEA, PyCharm, or Visual Studio Code for code development

Programming Languages:

- Python for algorithm development, Java or C++ for integration and performance-critical components

Genetic Algorithm Libraries:

- Python Libraries: DEAP (Distributed Evolutionary Algorithms in Python), Pygad, or similar libraries for implementing genetic algorithms

Network Simulation and Testing Tools:

- Simulators: Flask application

Database Management Systems:

- DBMS: CSV

Version Control Systems:

- VCS: Visual Studio Code

Documentation and Reporting Tools:

- Documentation: Microsoft Word
- Reporting: Tableau, Power BI, or custom-built tools for analyzing and visualizing optimization results

Security Tools:

- Vulnerability Scanners: Nessus for assessing the security of the optimized rule sets
- Penetration Testing Tools: Metasploit, Burp Suite for testing the security effectiveness of the firewall

Project Implementation :

Advanced Firewall Rule Optimization in a "Corporate IT Network"

1.Project Planning and Requirement Analysis:

Objective Definition:

- Improve security and performance by optimizing firewall rules in a large corporate network.

Requirement Gathering:

- Identify all firewalls in the network.
- Determine the format and structure of existing firewall rules.
- Establish criteria for optimization (e.g., security, performance, compliance).

2.System Design:

Architecture Design:

Components:

- Rule Collector: Gathers firewall rules from various sources.
- Rule Normalizer: Standardizes the format of collected rules.
- Genetic Algorithm Engine: Optimizes the rules.
- Rule Validator: Ensures optimized rules are functional and compliant.
- User Interface: Allows administrators to input, manage, and review rules.

Data Flow:

- Collection → Normalization → Optimization → Validation → Output

Data Flow Diagrams:

- Create diagrams to visualize the flow of data from collection to output.

3.Firewall Rule Extraction:

- Develop scripts to extract firewall rules from different sources

```python
import os
import json

def extract_rules(directory):
    rules = []
    for filename in os.listdir(directory):
        if filename.endswith(".json"):
            with open(os.path.join(directory, filename), 'r') as file:
                rules.append(json.load(file))
    return rules

firewall_rules = extract_rules('/path/to/firewall/rules')
```

4.Data Normalization:

Normalization:

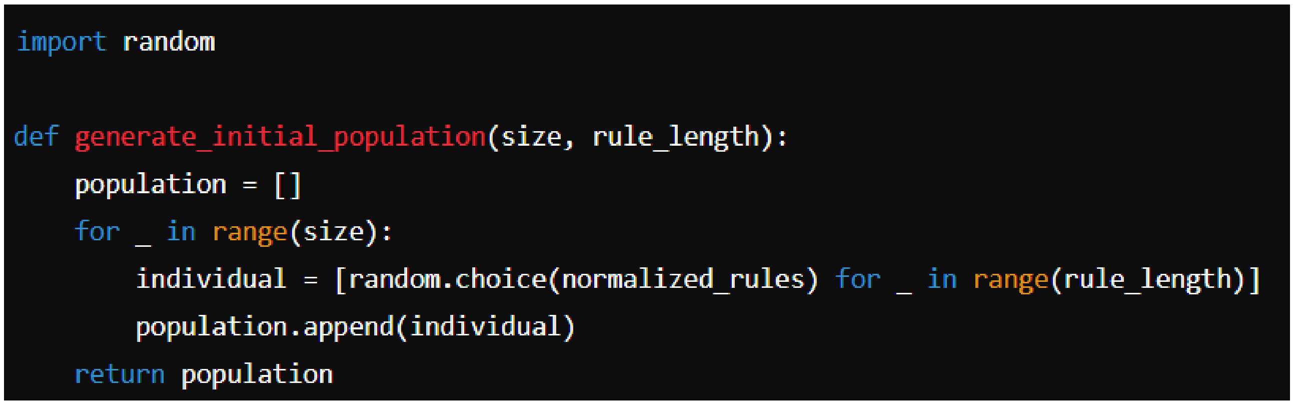
- Convert extracted rules into a standard format for easier processing.

```python
def normalize_rules(rules):
    normalized_rules = []
    for rule_set in rules:
        for rule in rule_set:
            normalized_rule = {
                "source": rule["src_ip"],
                "destination": rule["dest_ip"],
                "port": rule["port"],
                "action": rule["action"]
            }
            normalized_rules.append(normalized_rule)
    return normalized_rules

normalized_rules = normalize_rules(firewall_rules)
```

5.Genetic Algorithm Design and Implementation:

Initial Population Generation:

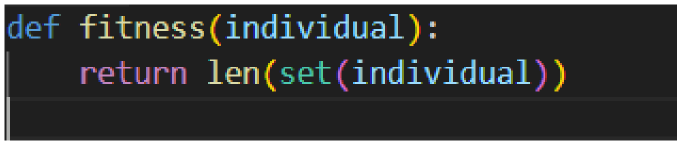
- Generate an initial population of firewall rule sets.



```
import random

def generate_initial_population(size, rule_length):
    population = []
    for _ in range(size):
        individual = [random.choice(normalized_rules) for _ in range(rule_length)]
        population.append(individual)
    return population
```

Fitness Function:

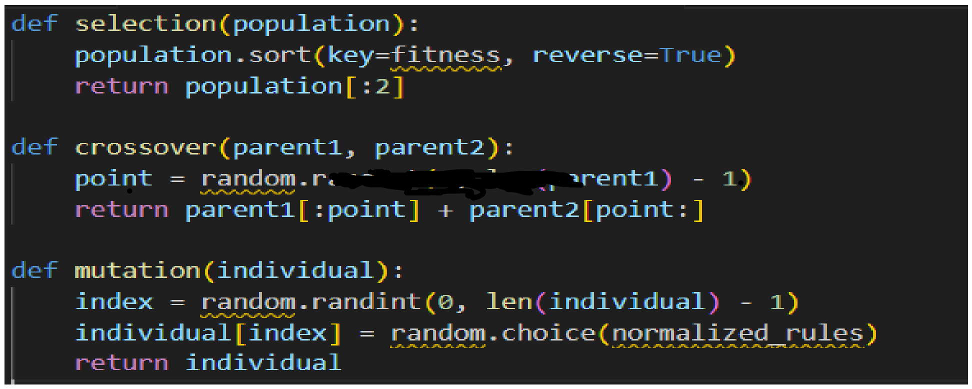
- Define a fitness function to evaluate rule sets based on security and performance



```
def fitness(individual):
    return len(set(individual))
```

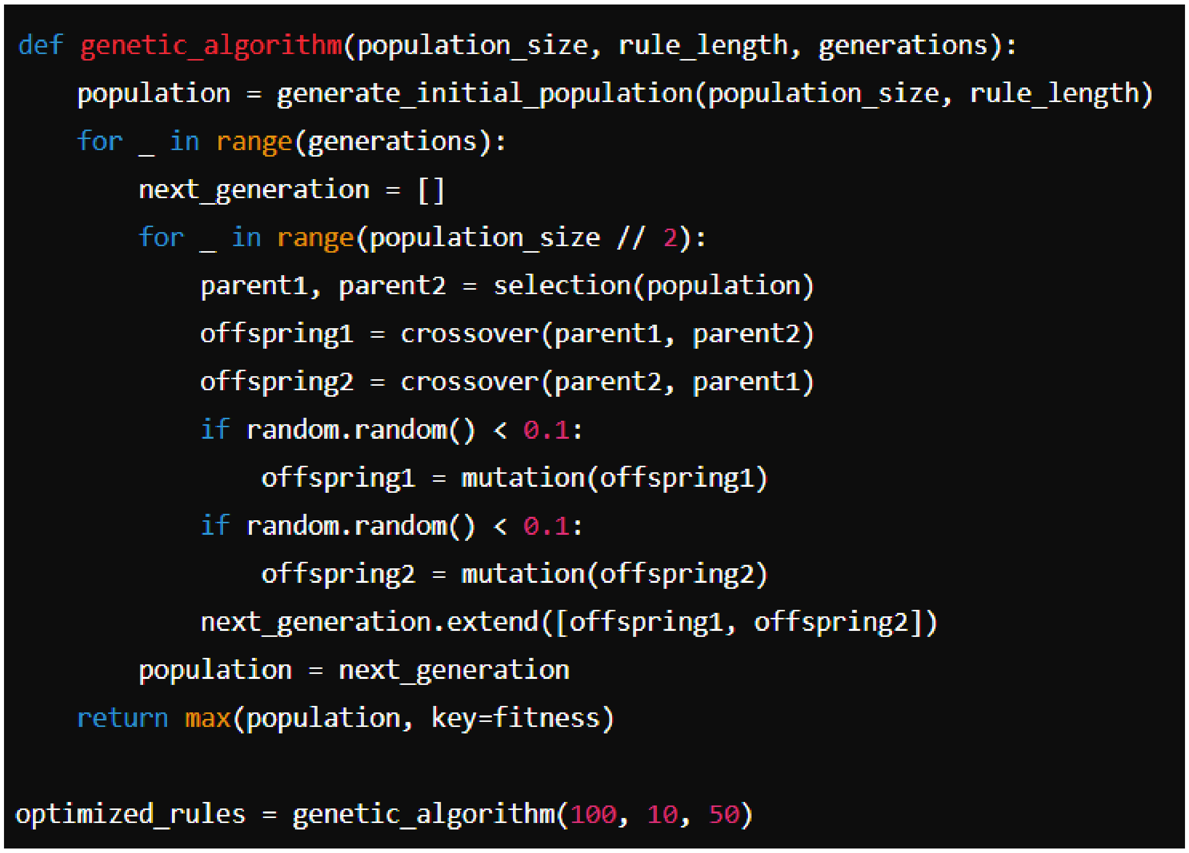
Genetic Operators:

- Implement selection, crossover, and mutation operators.



```
def selection(population):
    population.sort(key=fitness, reverse=True)
    return population[:2]

def crossover(parent1, parent2):
    point = random.r[redacted]rent1) - 1)
    return parent1[:point] + parent2[point:]

def mutation(individual):
    index = random.randint(0, len(individual) - 1)
    individual[index] = random.choice(normalized_rules)
    return individual
```

Genetic Algorithm Execution :

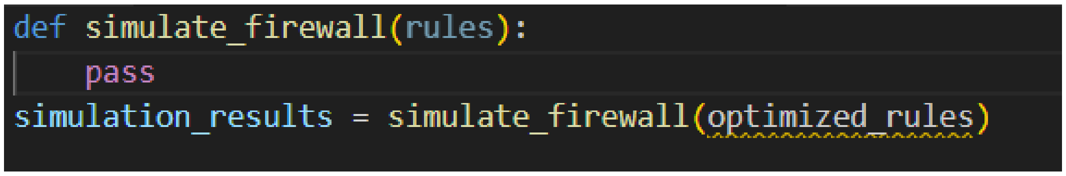


```python
def genetic_algorithm(population_size, rule_length, generations):
    population = generate_initial_population(population_size, rule_length)
    for _ in range(generations):
        next_generation = []
        for _ in range(population_size // 2):
            parent1, parent2 = selection(population)
            offspring1 = crossover(parent1, parent2)
            offspring2 = crossover(parent2, parent1)
            if random.random() < 0.1:
                offspring1 = mutation(offspring1)
            if random.random() < 0.1:
                offspring2 = mutation(offspring2)
            next_generation.extend([offspring1, offspring2])
        population = next_generation
    return max(population, key=fitness)

optimized_rules = genetic_algorithm(100, 10, 50)
```

6.Validation and Testing:

Simulation:

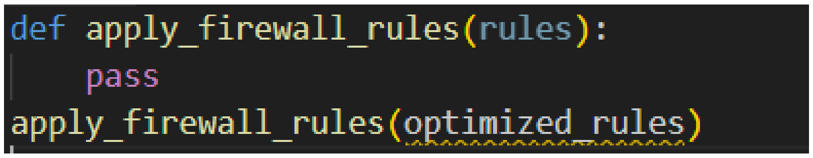
- Test the optimized firewall rules in a simulated environment.



```python
def simulate_firewall(rules):
    pass
simulation_results = simulate_firewall(optimized_rules)
```

Real-World Testing:

- Implement the optimized rules in a controlled real-world environment.

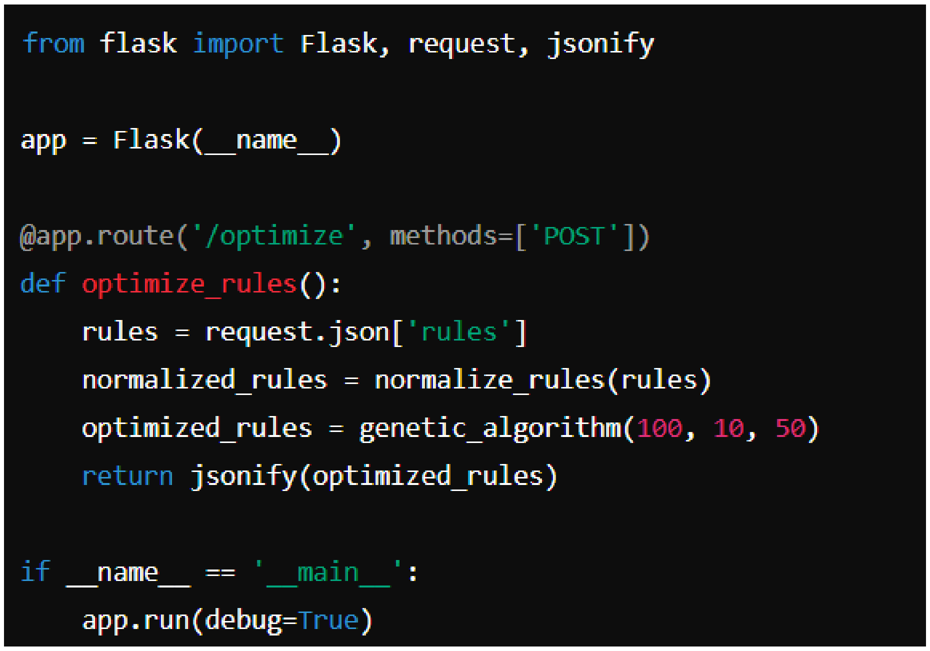


```python
def apply_firewall_rules(rules):
    pass
apply_firewall_rules(optimized_rules)
```

User Interface Development:

Dashboard Design:

- Develop a user-friendly dashboard using Flask for inputting and managing firewall rules.

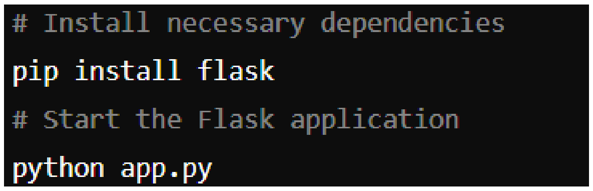


```python
from flask import Flask, request, jsonify

app = Flask(__name__)

@app.route('/optimize', methods=['POST'])
def optimize_rules():
    rules = request.json['rules']
    normalized_rules = normalize_rules(rules)
    optimized_rules = genetic_algorithm(100, 10, 50)
    return jsonify(optimized_rules)

if __name__ == '__main__':
    app.run(debug=True)
```

8.Deployment and Maintenance:

Deployment:

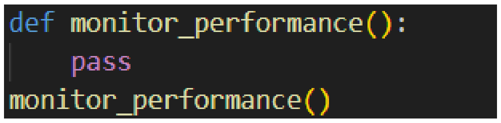
- Deploy the optimization tool within the corporate network.



```
# Install necessary dependencies
pip install flask
# Start the Flask application
python app.py
```

Monitoring and Maintenance:

- Continuously monitor the performance and effectiveness of the optimized firewall rules.



```
def monitor_performance():
    pass
monitor_performance()
```

9.Documentation and Training:

Documentation:

- Create comprehensive documentation for the system, including user guides and technical specifications.

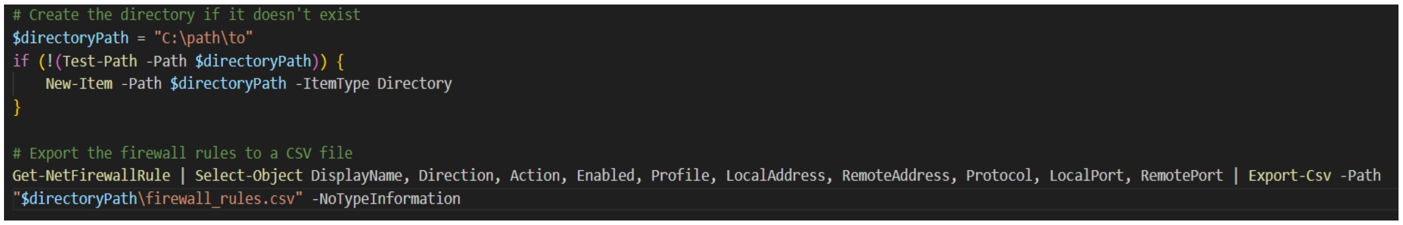
Training:

- Provide training sessions for network administrators on using the optimization tool and understanding the optimized ru

Methods & Methodologies :

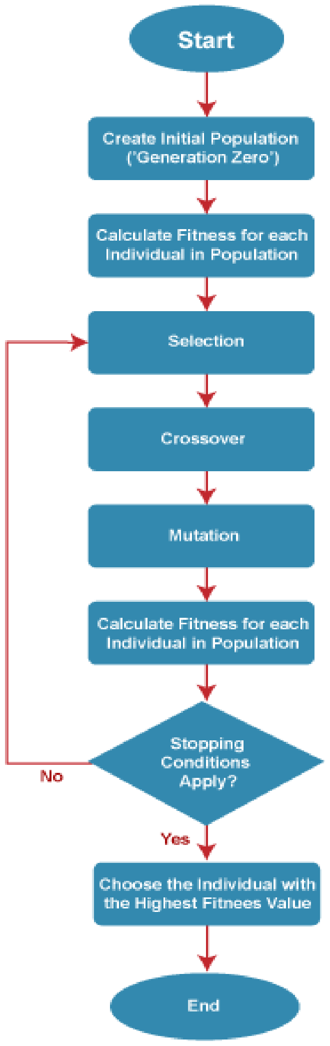
Extraction of firewall rules :

- Extract the firewall rules from the computers.
- Save and merge those rules in one single csv file (.csv)
- This process can be achieved by giving the following command in the windows command prompt

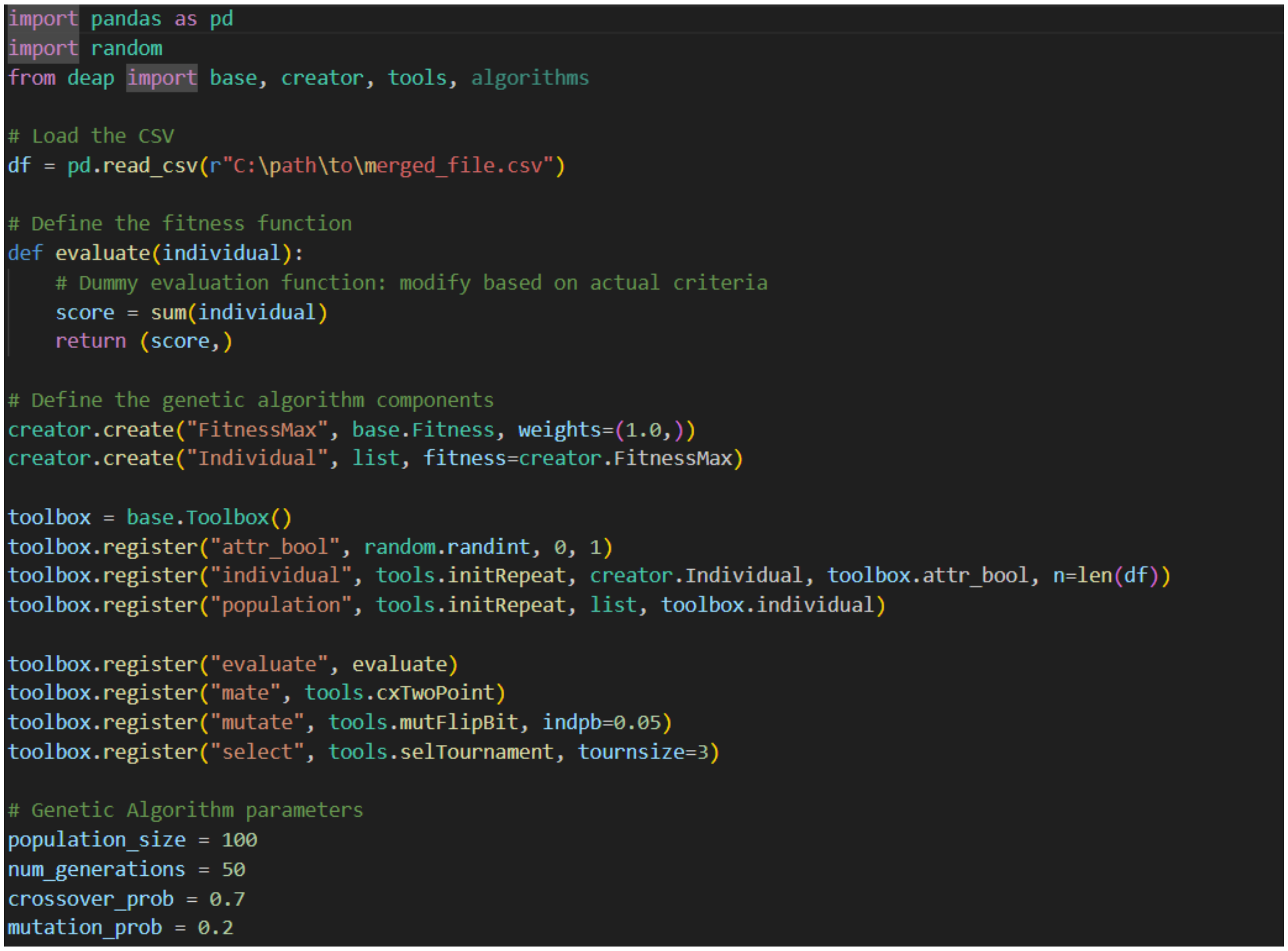


```
# Create the directory if it doesn't exist
$directoryPath = "C:\path\to"
if (!(Test-Path -Path $directoryPath)) {
    New-Item -Path $directoryPath -ItemType Directory
}

# Export the firewall rules to a CSV file
Get-NetFirewallRule | Select-Object DisplayName, Direction, Action, Enabled, Profile, LocalAddress, RemoteAddress, Protocol, LocalPort, RemotePort | Export-Csv -Path
"$directoryPath\firewall_rules.csv" -NoTypeInformation
```

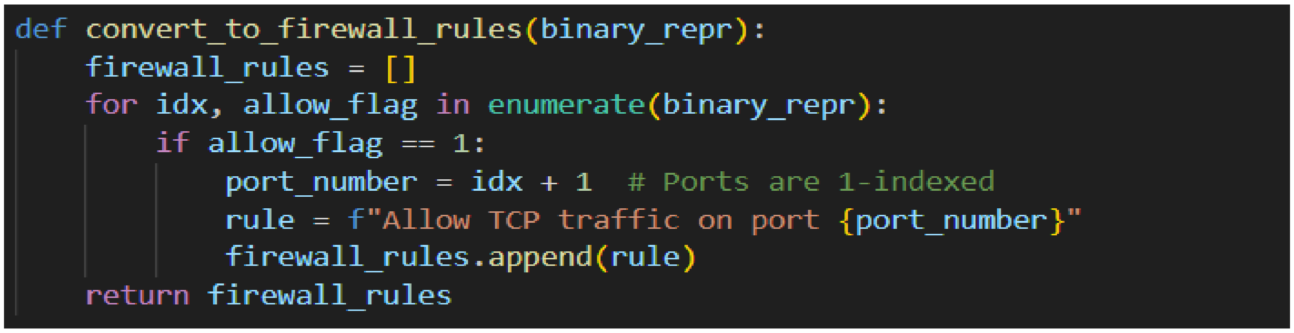
Genetic algorithm :


Start
Create Initial Population ("Generation Zero")
Calculate Fitness for each Individual in Population
Selection
Crossover
Mutation
Calculate Fitness for each Individual in Population
Stopping Conditions Apply?
No
Yes
Choose the Individual with the Highest Fitnees Value
End

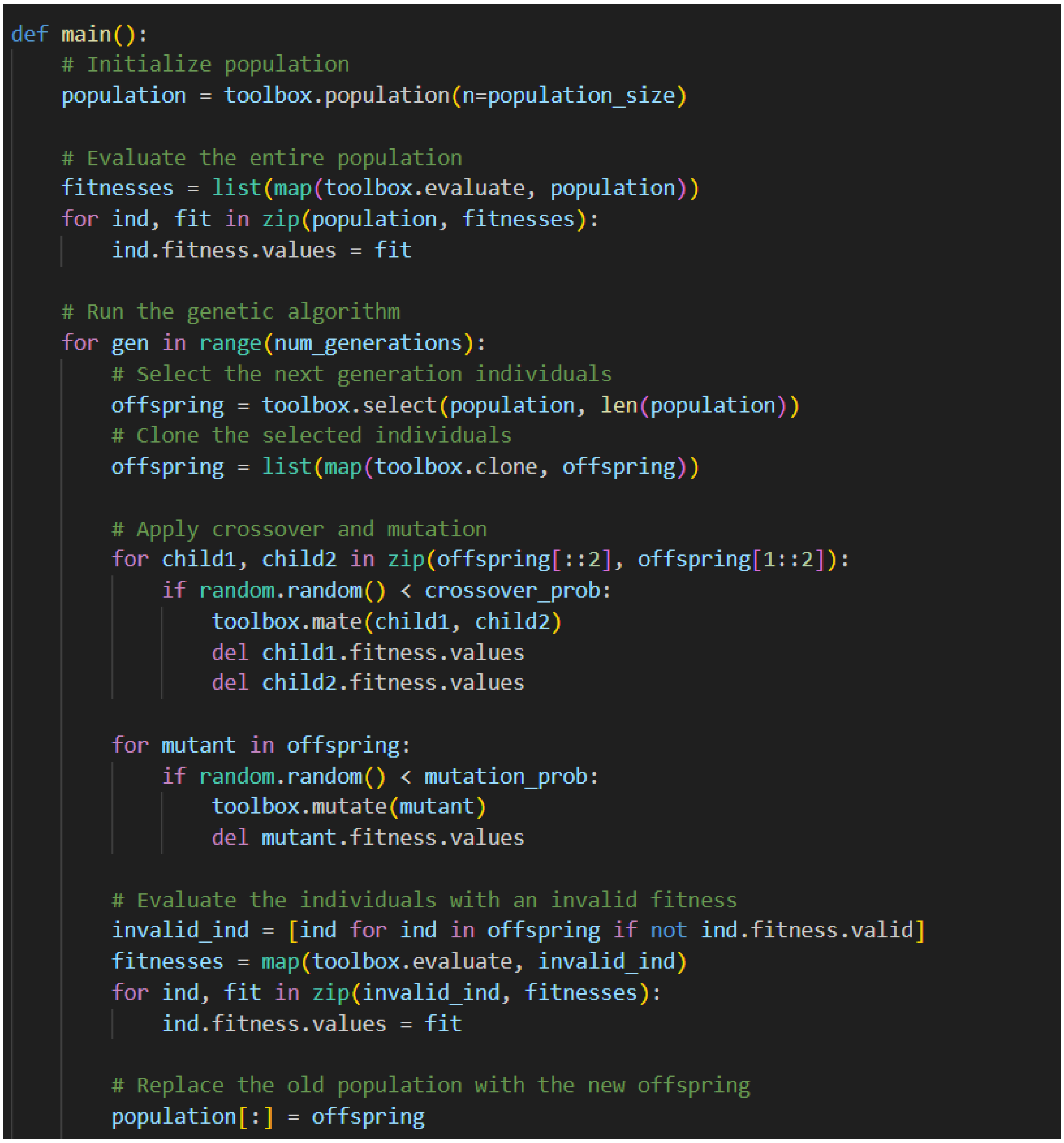
Python code using genetic algorithm (Conversion Of New Firewall Rules) :

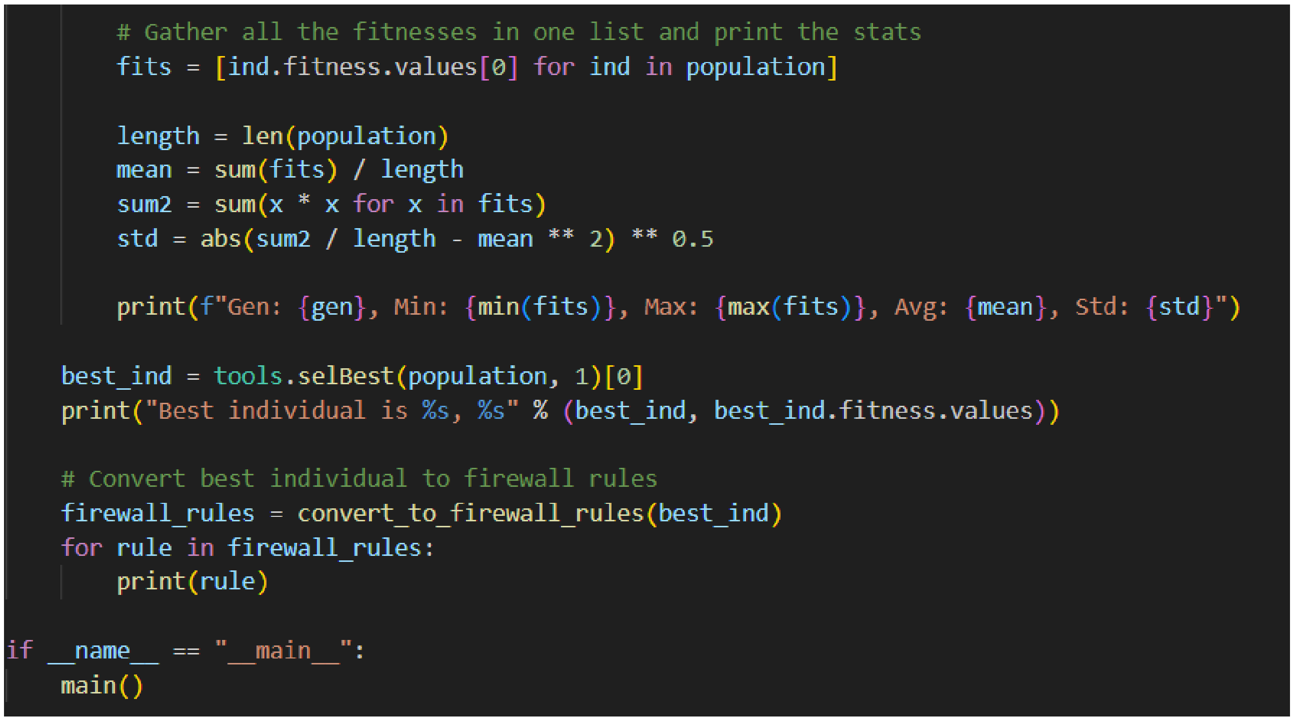


```python
import pandas as pd
import random
from deap import base, creator, tools, algorithms

# Load the CSV
df = pd.read_csv(r"C:\path\to\merged_file.csv")

# Define the fitness function
def evaluate(individual):
    # Dummy evaluation function: modify based on actual criteria
    score = sum(individual)
    return (score,)

# Define the genetic algorithm components
creator.create("FitnessMax", base.Fitness, weights=(1.0,))
creator.create("Individual", list, fitness=creator.FitnessMax)

toolbox = base.Toolbox()
toolbox.register("attr_bool", random.randint, 0, 1)
toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.attr_bool, n=len(df))
toolbox.register("population", tools.initRepeat, list, toolbox.individual)

toolbox.register("evaluate", evaluate)
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutFlipBit, indpb=0.05)
toolbox.register("select", tools.selTournament, tournsize=3)

# Genetic Algorithm parameters
population_size = 100
num_generations = 50
crossover_prob = 0.7
mutation_prob = 0.2
```



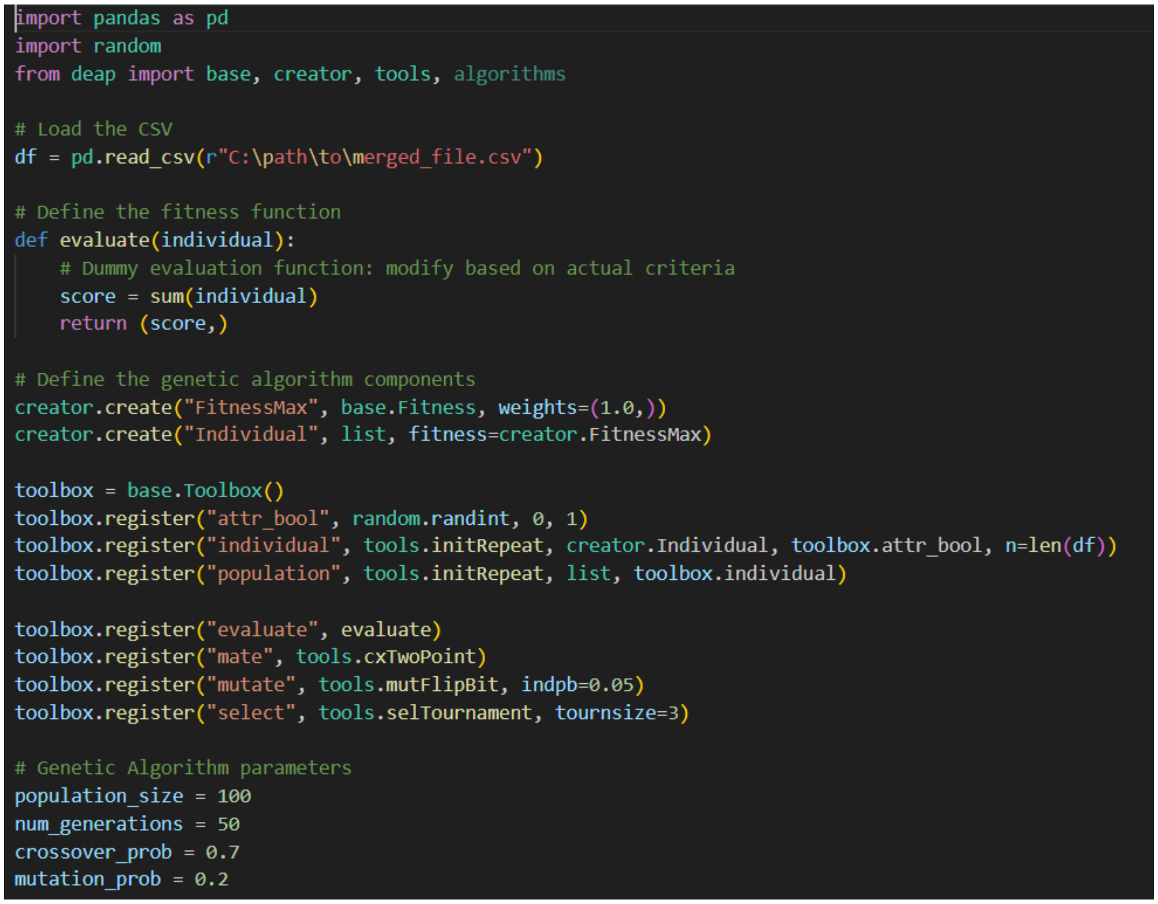
```python
def convert_to_firewall_rules(binary_repr):
    firewall_rules = []
    for idx, allow_flag in enumerate(binary_repr):
        if allow_flag == 1:
            port_number = idx + 1  # Ports are 1-indexed
            rule = f"Allow TCP traffic on port {port_number}"
            firewall_rules.append(rule)
    return firewall_rules
```


```python
def main():
    # Initialize population
    population = toolbox.population(n=population_size)

    # Evaluate the entire population
    fitnesses = list(map(toolbox.evaluate, population))
    for ind, fit in zip(population, fitnesses):
        ind.fitness.values = fit

    # Run the genetic algorithm
    for gen in range(num_generations):
        # Select the next generation individuals
        offspring = toolbox.select(population, len(population))
        # Clone the selected individuals
        offspring = list(map(toolbox.clone, offspring))

        # Apply crossover and mutation
        for child1, child2 in zip(offspring[::2], offspring[1::2]):
            if random.random() < crossover_prob:
                toolbox.mate(child1, child2)
                del child1.fitness.values
                del child2.fitness.values

        for mutant in offspring:
            if random.random() < mutation_prob:
                toolbox.mutate(mutant)
                del mutant.fitness.values

        # Evaluate the individuals with an invalid fitness
        invalid_ind = [ind for ind in offspring if not ind.fitness.valid]
        fitnesses = map(toolbox.evaluate, invalid_ind)
        for ind, fit in zip(invalid_ind, fitnesses):
            ind.fitness.values = fit

        # Replace the old population with the new offspring
        population[:] = offspring
```

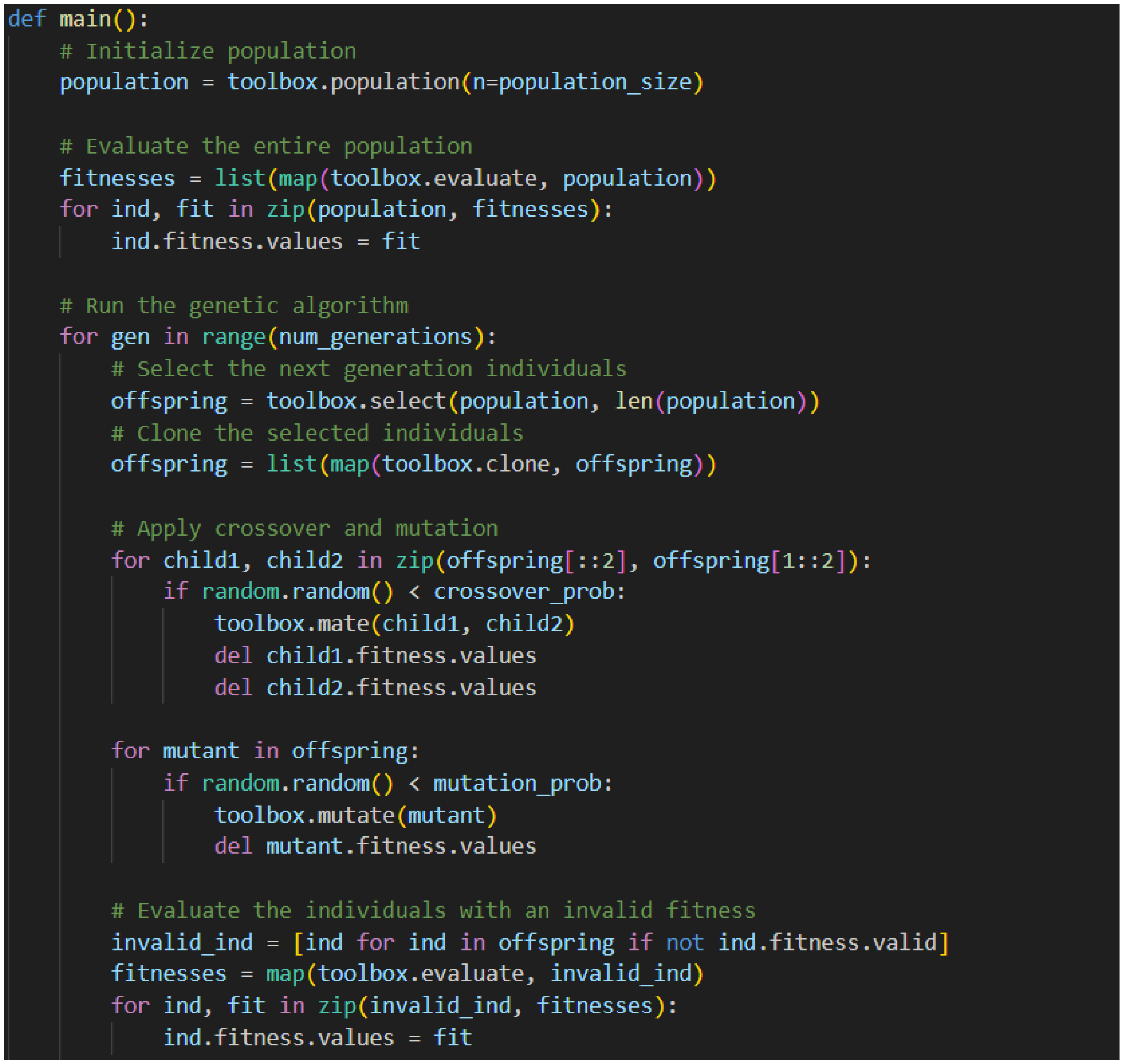


```
        # Gather all the fitnesses in one list and print the stats
        fits = [ind.fitness.values[0] for ind in population]

        length = len(population)
        mean = sum(fits) / length
        sum2 = sum(x * x for x in fits)
        std = abs(sum2 / length - mean ** 2) ** 0.5

        print(f"Gen: {gen}, Min: {min(fits)}, Max: {max(fits)}, Avg: {mean}, Std: {std}")

    best_ind = tools.selBest(population, 1)[0]
    print("Best individual is %s, %s" % (best_ind, best_ind.fitness.values))

    # Convert best individual to firewall rules
    firewall_rules = convert_to_firewall_rules(best_ind)
    for rule in firewall_rules:
        print(rule)

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

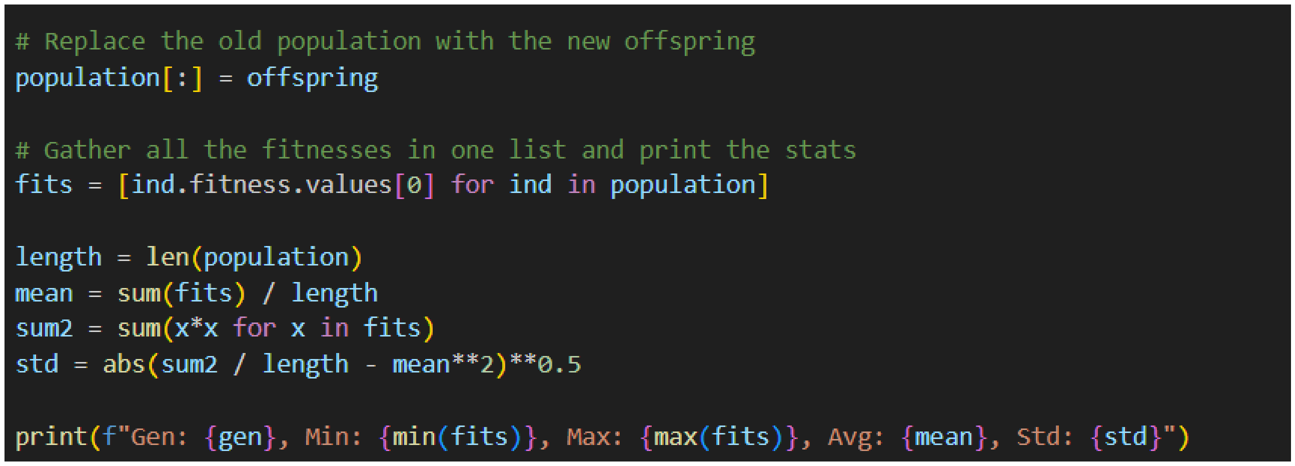
Python code using genetic algorithm (Parameter Extraction) :



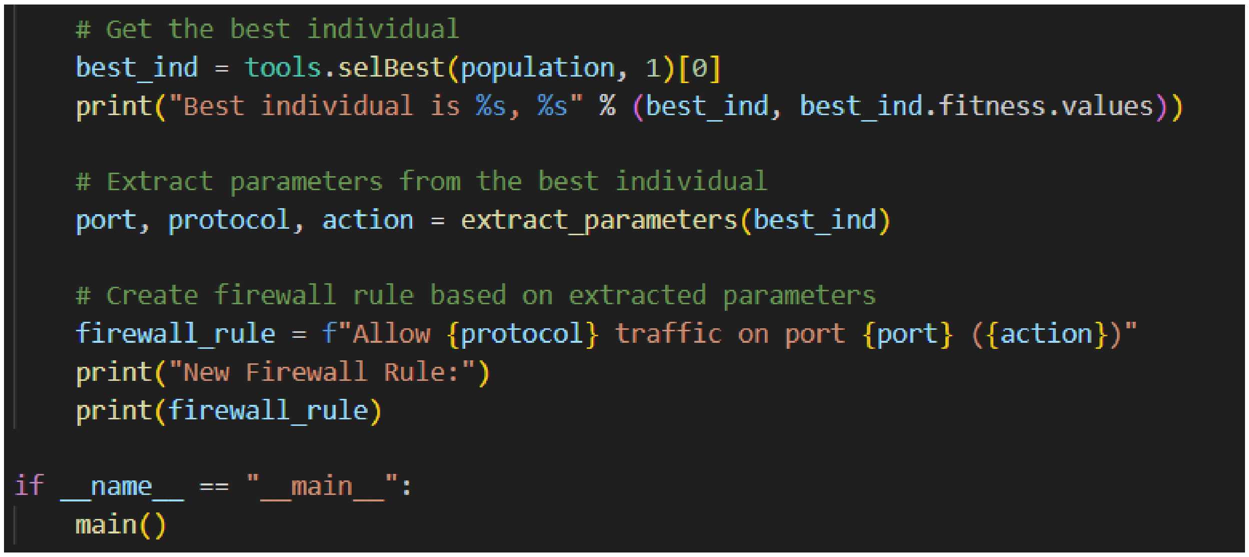
```
import pandas as pd
import random
from deap import base, creator, tools, algorithms

# Load the CSV
df = pd.read_csv(r"C:\path\to\merged_file.csv")

# Define the fitness function
def evaluate(individual):
    # Dummy evaluation function: modify based on actual criteria
    score = sum(individual)
    return (score,)

# Define the genetic algorithm components
creator.create("FitnessMax", base.Fitness, weights=(1.0,))
creator.create("Individual", list, fitness=creator.FitnessMax)

toolbox = base.Toolbox()
toolbox.register("attr_bool", random.randint, 0, 1)
toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.attr_bool, n=len(df))
toolbox.register("population", tools.initRepeat, list, toolbox.individual)

toolbox.register("evaluate", evaluate)
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutFlipBit, indpb=0.05)
toolbox.register("select", tools.selTournament, tournsize=3)

# Genetic Algorithm parameters
population_size = 100
num_generations = 50
crossover_prob = 0.7
mutation_prob = 0.2
```



```python
def main():
    # Initialize population
    population = toolbox.population(n=population_size)

    # Evaluate the entire population
    fitnesses = list(map(toolbox.evaluate, population))
    for ind, fit in zip(population, fitnesses):
        ind.fitness.values = fit

    # Run the genetic algorithm
    for gen in range(num_generations):
        # Select the next generation individuals
        offspring = toolbox.select(population, len(population))
        # Clone the selected individuals
        offspring = list(map(toolbox.clone, offspring))

        # Apply crossover and mutation
        for child1, child2 in zip(offspring[::2], offspring[1::2]):
            if random.random() < crossover_prob:
                toolbox.mate(child1, child2)
                del child1.fitness.values
                del child2.fitness.values

        for mutant in offspring:
            if random.random() < mutation_prob:
                toolbox.mutate(mutant)
                del mutant.fitness.values

        # Evaluate the individuals with an invalid fitness
        invalid_ind = [ind for ind in offspring if not ind.fitness.valid]
        fitnesses = map(toolbox.evaluate, invalid_ind)
        for ind, fit in zip(invalid_ind, fitnesses):
            ind.fitness.values = fit
```

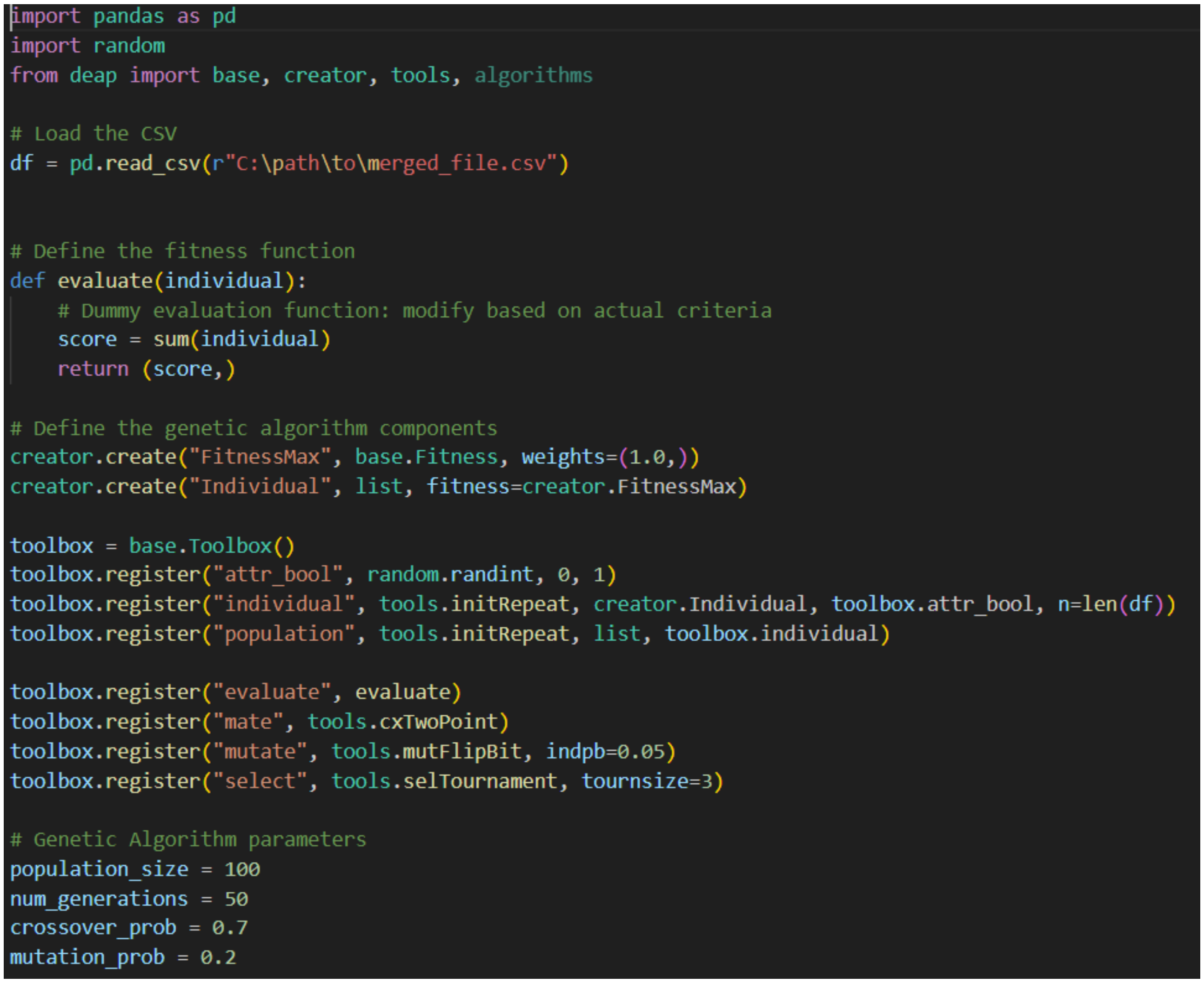


```python
# Replace the old population with the new offspring
population[:] = offspring

# Gather all the fitnesses in one list and print the stats
fits = [ind.fitness.values[0] for ind in population]

length = len(population)
mean = sum(fits) / length
sum2 = sum(x*x for x in fits)
std = abs(sum2 / length - mean**2)**0.5

print(f"Gen: {gen}, Min: {min(fits)}, Max: {max(fits)}, Avg: {mean}, Std: {std}")
```

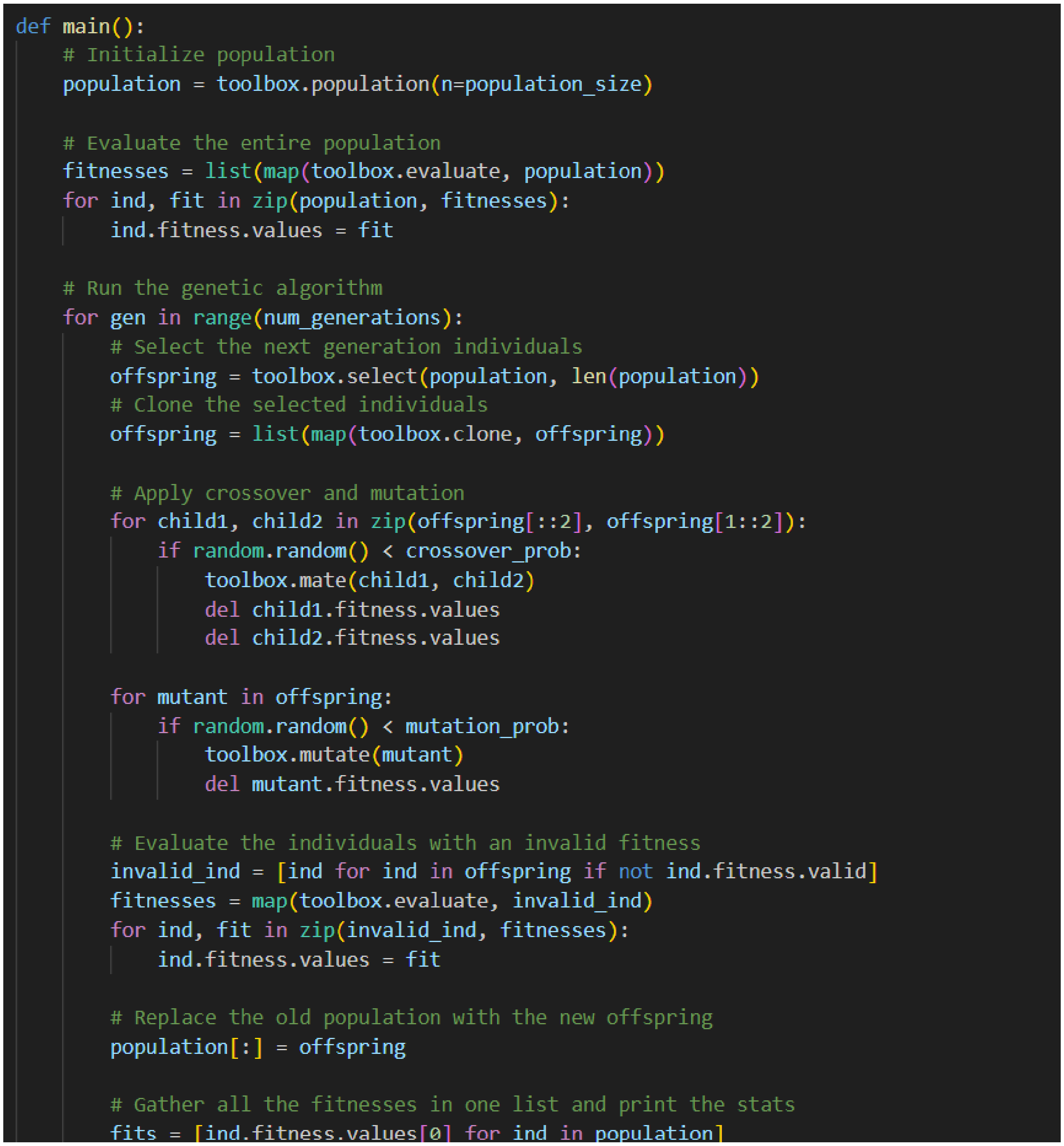


```python
    # Get the best individual
    best_ind = tools.selBest(population, 1)[0]
    print("Best individual is %s, %s" % (best_ind, best_ind.fitness.values))

    # Extract parameters from the best individual
    port, protocol, action = extract_parameters(best_ind)

    # Create firewall rule based on extracted parameters
    firewall_rule = f"Allow {protocol} traffic on port {port} ({action})"
    print("New Firewall Rule:")
    print(firewall_rule)

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

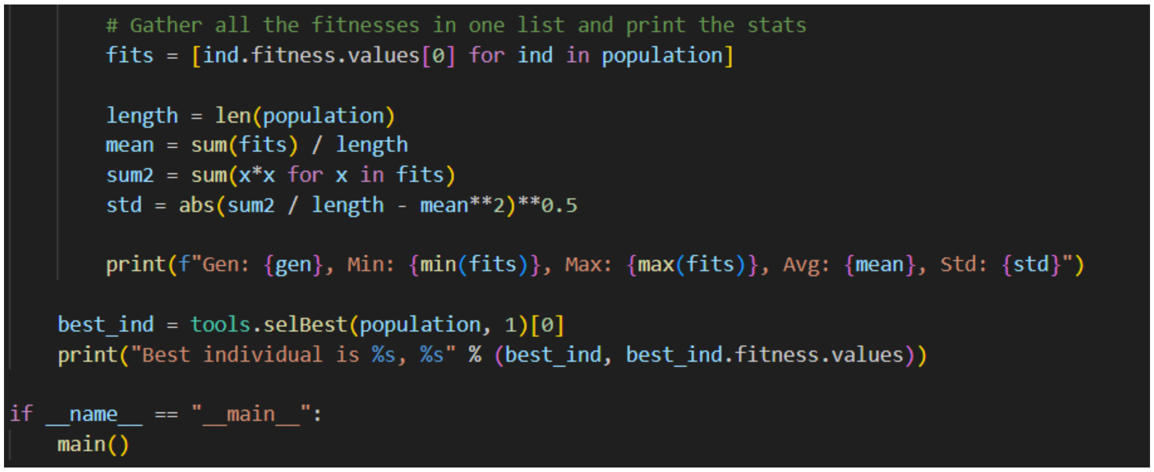
Final Python code using genetic algorithm :



```python
import pandas as pd
import random
from deap import base, creator, tools, algorithms

# Load the CSV
df = pd.read_csv(r"C:\path\to\merged_file.csv")


# Define the fitness function
def evaluate(individual):
    # Dummy evaluation function: modify based on actual criteria
    score = sum(individual)
    return (score,)

# Define the genetic algorithm components
creator.create("FitnessMax", base.Fitness, weights=(1.0,))
creator.create("Individual", list, fitness=creator.FitnessMax)

toolbox = base.Toolbox()
toolbox.register("attr_bool", random.randint, 0, 1)
toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.attr_bool, n=len(df))
toolbox.register("population", tools.initRepeat, list, toolbox.individual)

toolbox.register("evaluate", evaluate)
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutFlipBit, indpb=0.05)
toolbox.register("select", tools.selTournament, tournsize=3)

# Genetic Algorithm parameters
population_size = 100
num_generations = 50
crossover_prob = 0.7
mutation_prob = 0.2
```

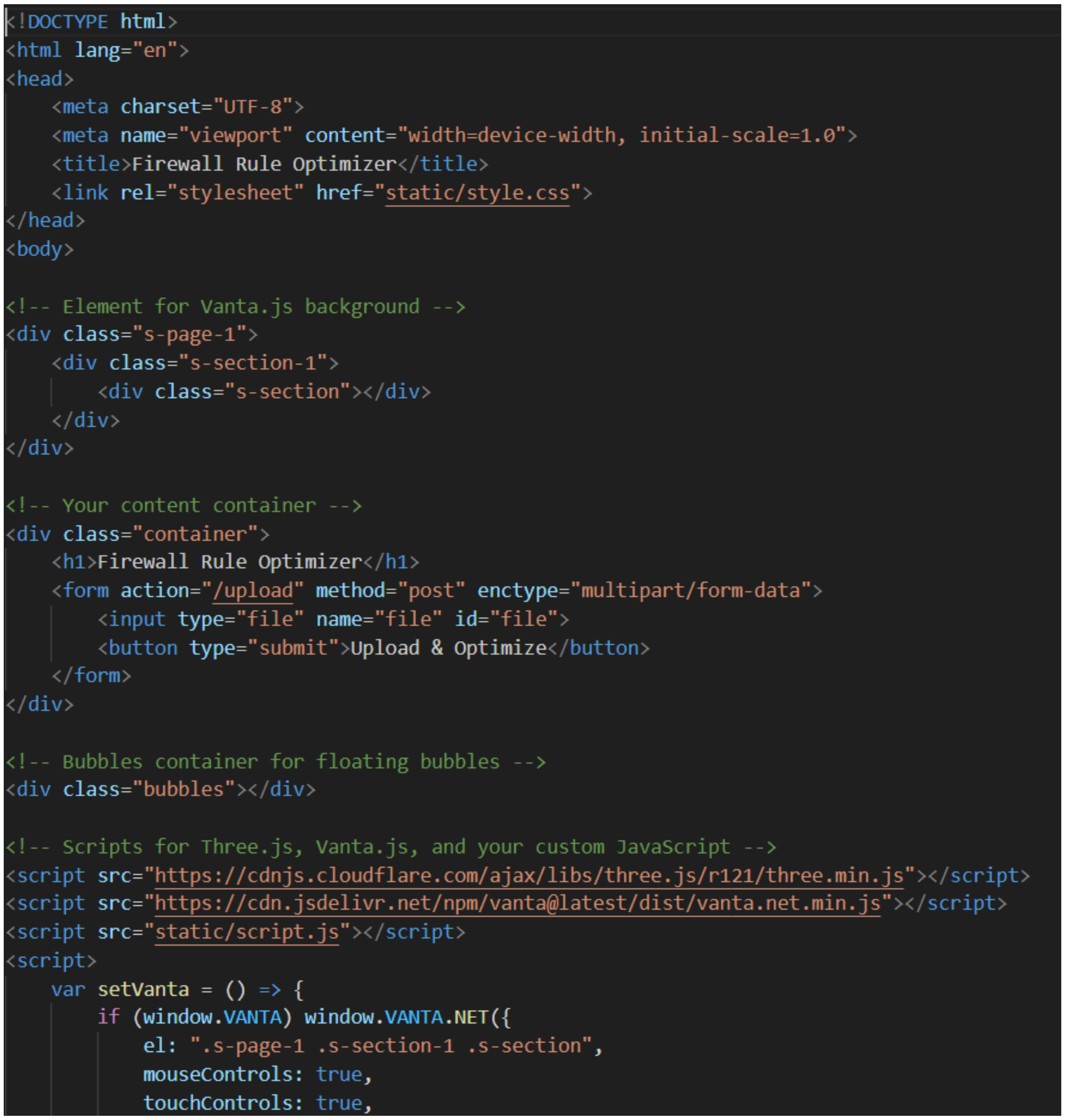


```python
def main():
    # Initialize population
    population = toolbox.population(n=population_size)

    # Evaluate the entire population
    fitnesses = list(map(toolbox.evaluate, population))
    for ind, fit in zip(population, fitnesses):
        ind.fitness.values = fit

    # Run the genetic algorithm
    for gen in range(num_generations):
        # Select the next generation individuals
        offspring = toolbox.select(population, len(population))
        # Clone the selected individuals
        offspring = list(map(toolbox.clone, offspring))

        # Apply crossover and mutation
        for child1, child2 in zip(offspring[::2], offspring[1::2]):
            if random.random() < crossover_prob:
                toolbox.mate(child1, child2)
                del child1.fitness.values
                del child2.fitness.values

        for mutant in offspring:
            if random.random() < mutation_prob:
                toolbox.mutate(mutant)
                del mutant.fitness.values

        # Evaluate the individuals with an invalid fitness
        invalid_ind = [ind for ind in offspring if not ind.fitness.valid]
        fitnesses = map(toolbox.evaluate, invalid_ind)
        for ind, fit in zip(invalid_ind, fitnesses):
            ind.fitness.values = fit

        # Replace the old population with the new offspring
        population[:] = offspring

        # Gather all the fitnesses in one list and print the stats
        fits = [ind.fitness.values[0] for ind in population]
```

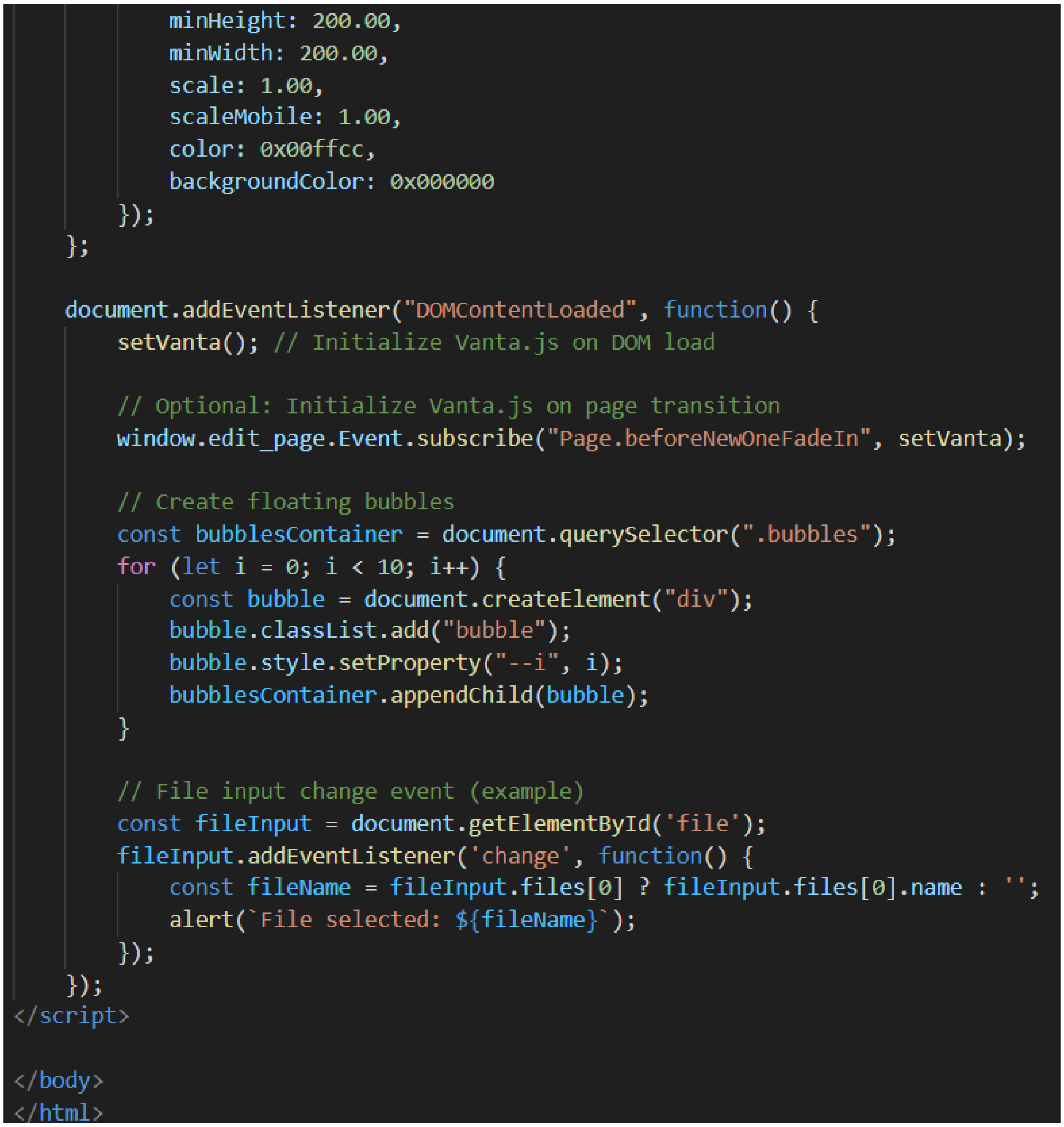


```python
        # Gather all the fitnesses in one list and print the stats
        fits = [ind.fitness.values[0] for ind in population]

        length = len(population)
        mean = sum(fits) / length
        sum2 = sum(x*x for x in fits)
        std = abs(sum2 / length - mean**2)**0.5

        print(f"Gen: {gen}, Min: {min(fits)}, Max: {max(fits)}, Avg: {mean}, Std: {std}")

    best_ind = tools.selBest(population, 1)[0]
    print("Best individual is %s, %s" % (best_ind, best_ind.fitness.values))

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

HTML Code for web interface :



```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Firewall Rule Optimizer</title>
    <link rel="stylesheet" href="static/style.css">
</head>
<body>

<!-- Element for Vanta.js background -->
<div class="s-page-1">
    <div class="s-section-1">
        <div class="s-section"></div>
    </div>
</div>

<!-- Your content container -->
<div class="container">
    <h1>Firewall Rule Optimizer</h1>
    <form action="/upload" method="post" enctype="multipart/form-data">
        <input type="file" name="file" id="file">
        <button type="submit">Upload & Optimize</button>
    </form>
</div>

<!-- Bubbles container for floating bubbles -->
<div class="bubbles"></div>

<!-- Scripts for Three.js, Vanta.js, and your custom JavaScript -->
<script src="https://cdnjs.cloudflare.com/ajax/libs/three.js/r121/three.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/vanta@latest/dist/vanta.net.min.js"></script>
<script src="static/script.js"></script>
<script>
    var setVanta = () => {
        if (window.VANTA) window.VANTA.NET({
            el: ".s-page-1 .s-section-1 .s-section",
            mouseControls: true,
            touchControls: true,
```

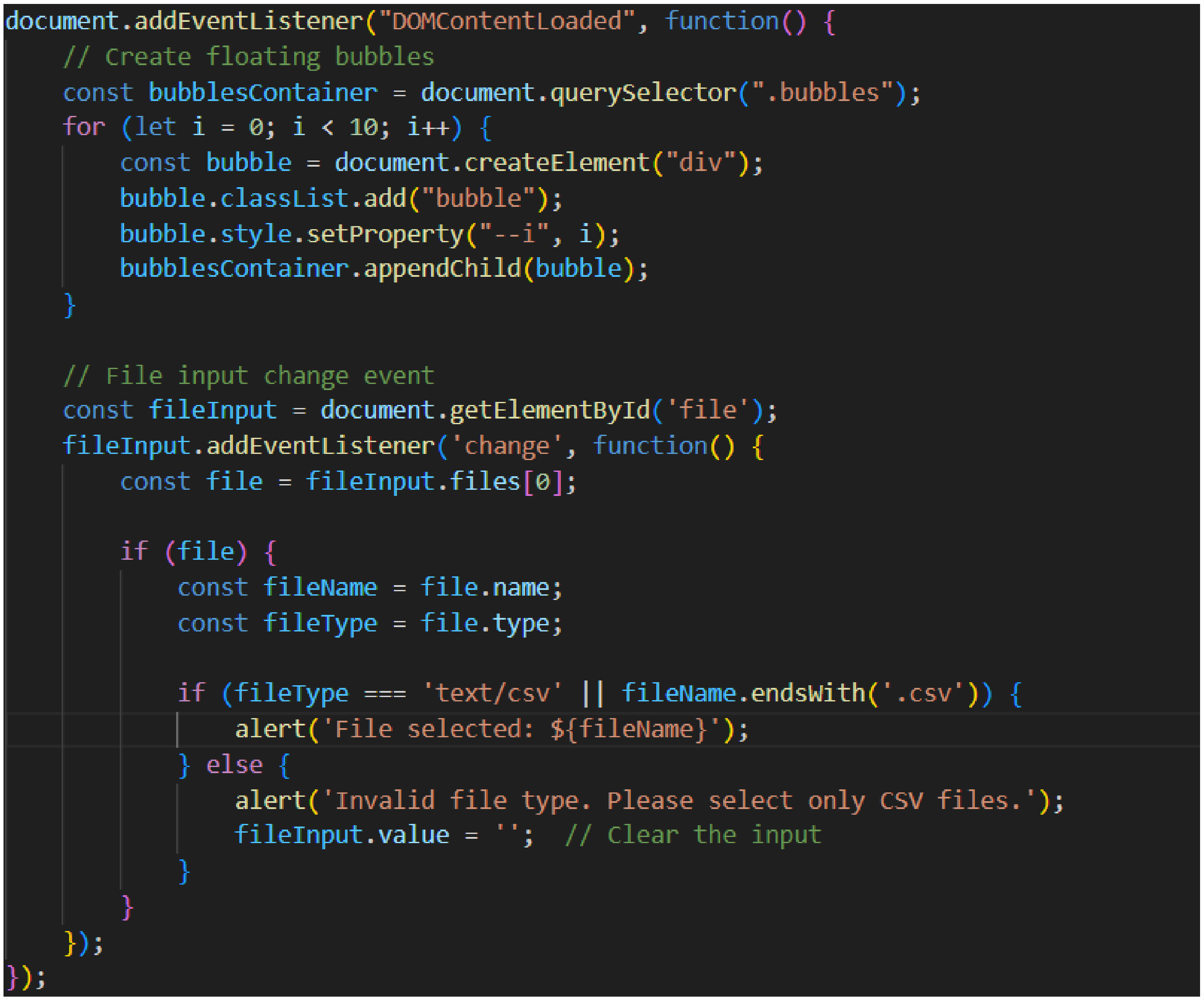

```
            minHeight: 200.00,
            minWidth: 200.00,
            scale: 1.00,
            scaleMobile: 1.00,
            color: 0x00ffcc,
            backgroundColor: 0x000000
        });
    };

    document.addEventListener("DOMContentLoaded", function() {
        setVanta(); // Initialize Vanta.js on DOM load

        // Optional: Initialize Vanta.js on page transition
        window.edit_page.Event.subscribe("Page.beforeNewOneFadeIn", setVanta);

        // Create floating bubbles
        const bubblesContainer = document.querySelector(".bubbles");
        for (let i = 0; i < 10; i++) {
            const bubble = document.createElement("div");
            bubble.classList.add("bubble");
            bubble.style.setProperty("--i", i);
            bubblesContainer.appendChild(bubble);
        }

        // File input change event (example)
        const fileInput = document.getElementById('file');
        fileInput.addEventListener('change', function() {
            const fileName = fileInput.files[0] ? fileInput.files[0].name : '';
            alert(`File selected: ${fileName}`);
        });
    });
</script>

</body>
</html>
```

CSS Code for styling & animations :

```
@import url('https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&display=swap');

body {
    font-family: 'Roboto', sans-serif;
    color: #ffffff;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    margin: 0;
    overflow: hidden;
    position: relative;
}

.container {
    background-color: rgba(0, 0, 0, 0.8);
    padding: 30px;
    border-radius: 10px;
    box-shadow: 0 0 20px rgba(0, 0, 0, 0.5);
    text-align: center;
    max-width: 400px;
    width: 100%;
    animation: fadeIn 1s ease-in-out;
    position: relative;
    z-index: 2;
}

h1 {
    margin-bottom: 20px;
    font-size: 28px;
    letter-spacing: 2px;
    color: #00ffcc;
    text-shadow: 0 0 10px rgba(0, 255, 204, 0.8);
    animation: slideIn 1.5s ease-in-out;
}

input[type="file"] {
    margin: 20px 0;
    padding: 10px;
    border: 2px solid #00ffcc;
    border-radius: 5px;
    background-color: #1c3a3e;
    color: #ffffff;
    cursor: pointer;
    transition: border-color 0.3s ease, background-color 0.3s ease;
    animation: fadeInUp 1.5s ease-in-out;
}

input[type="file"]::file-selector-button {
    padding: 5px 15px;
    border: 2px solid #00ffcc;
    border-radius: 5px;
    background-color: #00ffcc;
    color: #1c3a3e;
    cursor: pointer;
    transition: background-color 0.3s ease, color 0.3s ease;
}

input[type="file"]::file-selector-button:hover {
    background-color: #1c3a3e;
    color: #00ffcc;
}

button {
    padding: 10px 20px;
    background-color: #00ffcc;
    color: #1c3a3e;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    font-weight: bold;
    text-transform: uppercase;
    transition: background-color 0.3s ease, transform 0.3s ease;
    animation: fadeInUp 1.7s ease-in-out;
}

button:hover {
    background-color: #00e6b8;
    transform: translateY(-2px);
}

button:active {
    transform: translateY(0);
    background-color: #00bfa5;
}

@keyframes fadeIn {
    from {
        opacity: 0;
    }
    to {
        opacity: 1;
    }
}
```

```
@keyframes slideIn {
    from {
        transform: translateY(-100%);
        opacity: 0;
    }
    to {
        transform: translateY(0);
        opacity: 1;
    }
}

@keyframes fadeInUp {
    from {
        transform: translateY(20px);
        opacity: 0;
    }
    to {
        transform: translateY(0);
        opacity: 1;
    }
}

@keyframes floating {
    0%, 100% {
        transform: translateY(0);
    }
    50% {
        transform: translateY(-10px);
    }
}

.bubbles {
    position: absolute;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    overflow: hidden;
    z-index: 1;
}

.bubble {
    position: absolute;
    bottom: -100px;
    width: 40px;
    height: 40px;
    background-color: rgba(255, 255, 255, 0.15);
    border-radius: 50%;
    animation: floating 5s ease-in-out infinite;
    animation-delay: calc(0.5s * var(--i));
}

.bubble:nth-child(1) {
    left: 10%;
    animation-duration: 6s;
}

.bubble:nth-child(2) {
    left: 20%;
    animation-duration: 7s;
}

.bubble:nth-child(3) {
    left: 25%;
    animation-duration: 4s;
}

.bubble:nth-child(4) {
    left: 40%;
    animation-duration: 6s;
}

.bubble:nth-child(5) {
    left: 55%;
    animation-duration: 8s;
}

.bubble:nth-child(6) {
    left: 70%;
    animation-duration: 6s;
}

.bubble:nth-child(7) {
    left: 80%;
    animation-duration: 7s;
}
.bubble:nth-child(8) {
    left: 90%;
    animation-duration: 5s;
}
.s-page-1 .s-section-1 .s-section {
    position: fixed;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    z-index: -1;
}
```

JSON Code for web response :

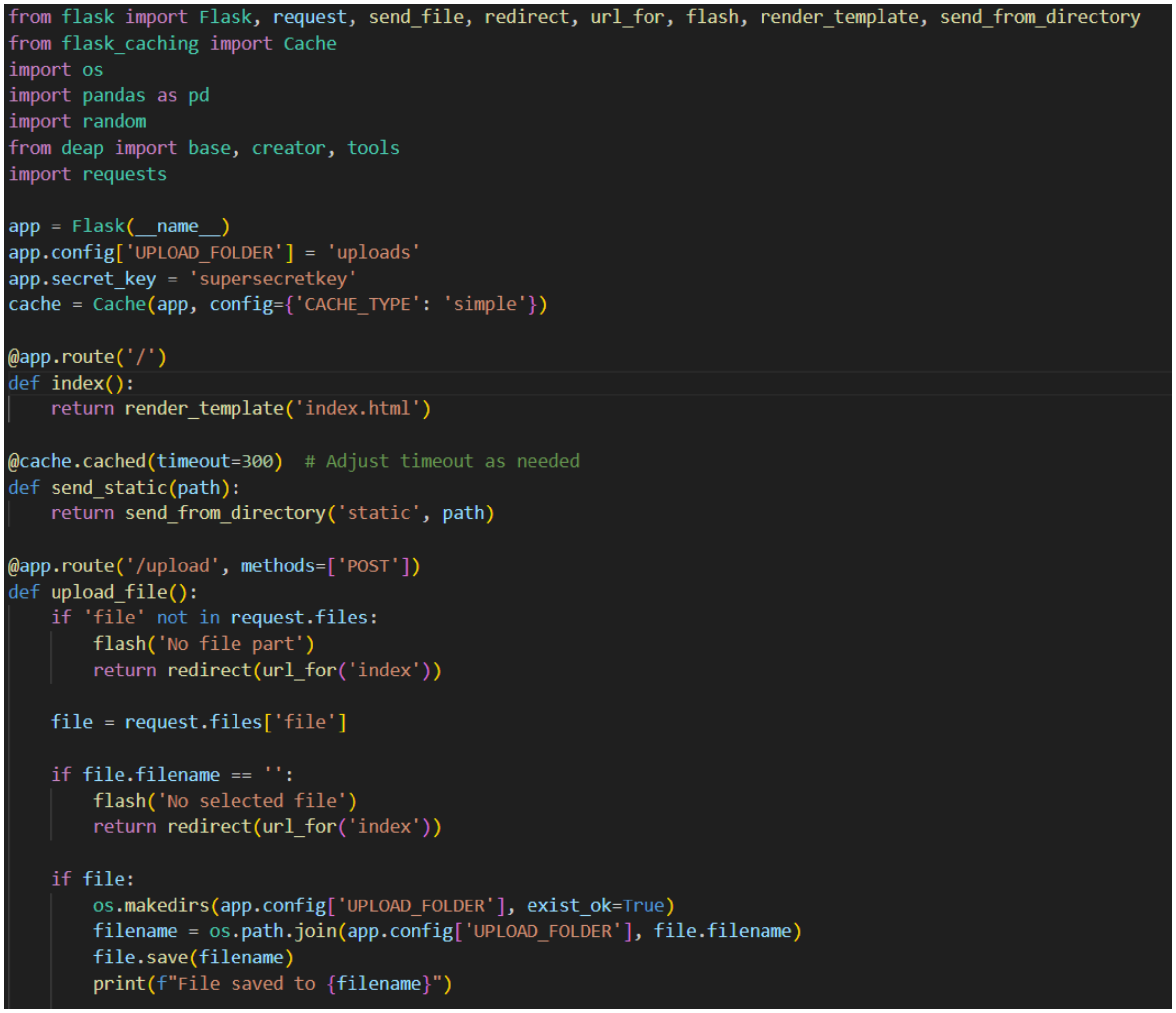


```
document.addEventListener("DOMContentLoaded", function() {
    // Create floating bubbles
    const bubblesContainer = document.querySelector(".bubbles");
    for (let i = 0; i < 10; i++) {
        const bubble = document.createElement("div");
        bubble.classList.add("bubble");
        bubble.style.setProperty("--i", i);
        bubblesContainer.appendChild(bubble);
    }

    // File input change event
    const fileInput = document.getElementById('file');
    fileInput.addEventListener('change', function() {
        const file = fileInput.files[0];

        if (file) {
            const fileName = file.name;
            const fileType = file.type;

            if (fileType === 'text/csv' || fileName.endsWith('.csv')) {
                alert('File selected: ${fileName}');
            } else {
                alert('Invalid file type. Please select only CSV files.');
                fileInput.value = '';  // Clear the input
            }
        }
    });
});
```

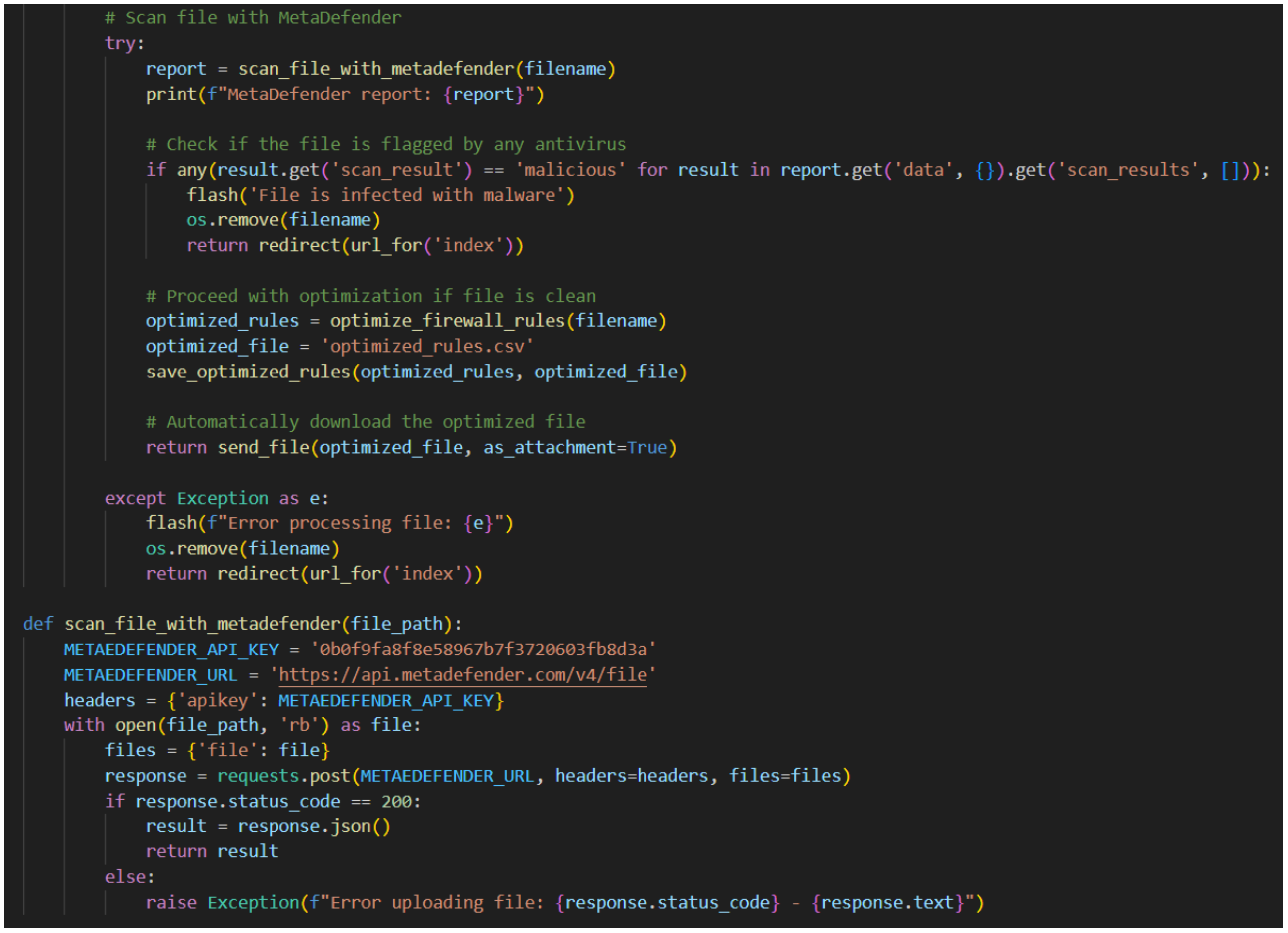
The above HTML, CSS & JSON file are used for the website interface which is being hosted using the FLASK application. The flask application code which is integrated with the genetic algorithm for optimizing the firewall rule is being given below.

Main Python file :

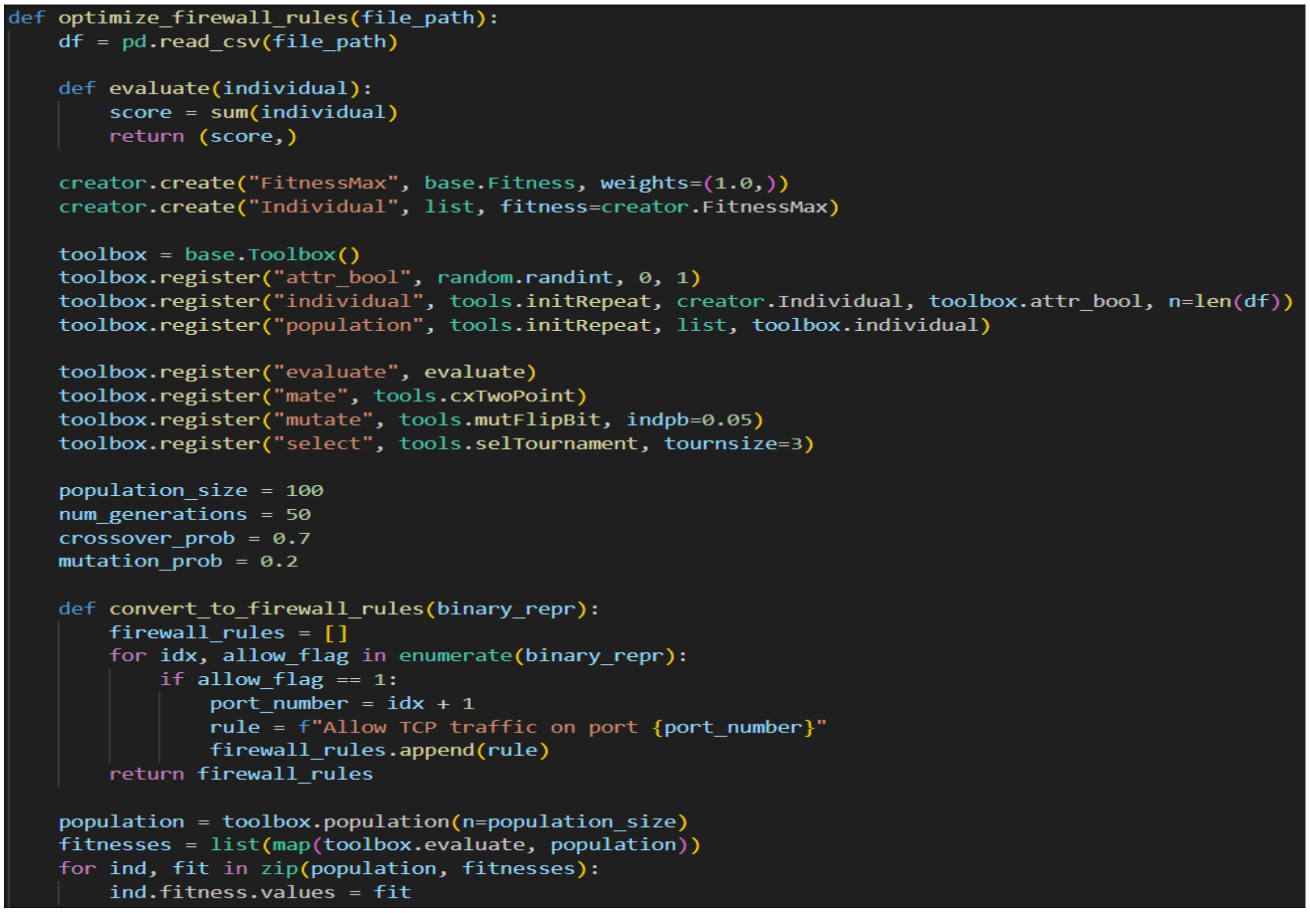
- The main python file is the file which must run to execute the optimized firewall rules
- This file consists of the Flask application which is integrated with the genetic algorithm for optimization of the rules
- When the host web server is running a web interface opens up and asks for the firewall rules
- The user gives in the merged firewall rules and it scans the csv file using OPSWAT. META-DEFENDER
- The user can give in only csv file and not anyother file in different file formats
- Once the file is being scanned and when no malicious activity is being found in that particular file then a optimized_firewall_rules.csv file is being created which contains the optimized firewall rules and it is automatically downloaded on the system
- With the help of this the optimized firewall rules can be integrated in the system and it reduces network traffic on it as it has less and optimized rules



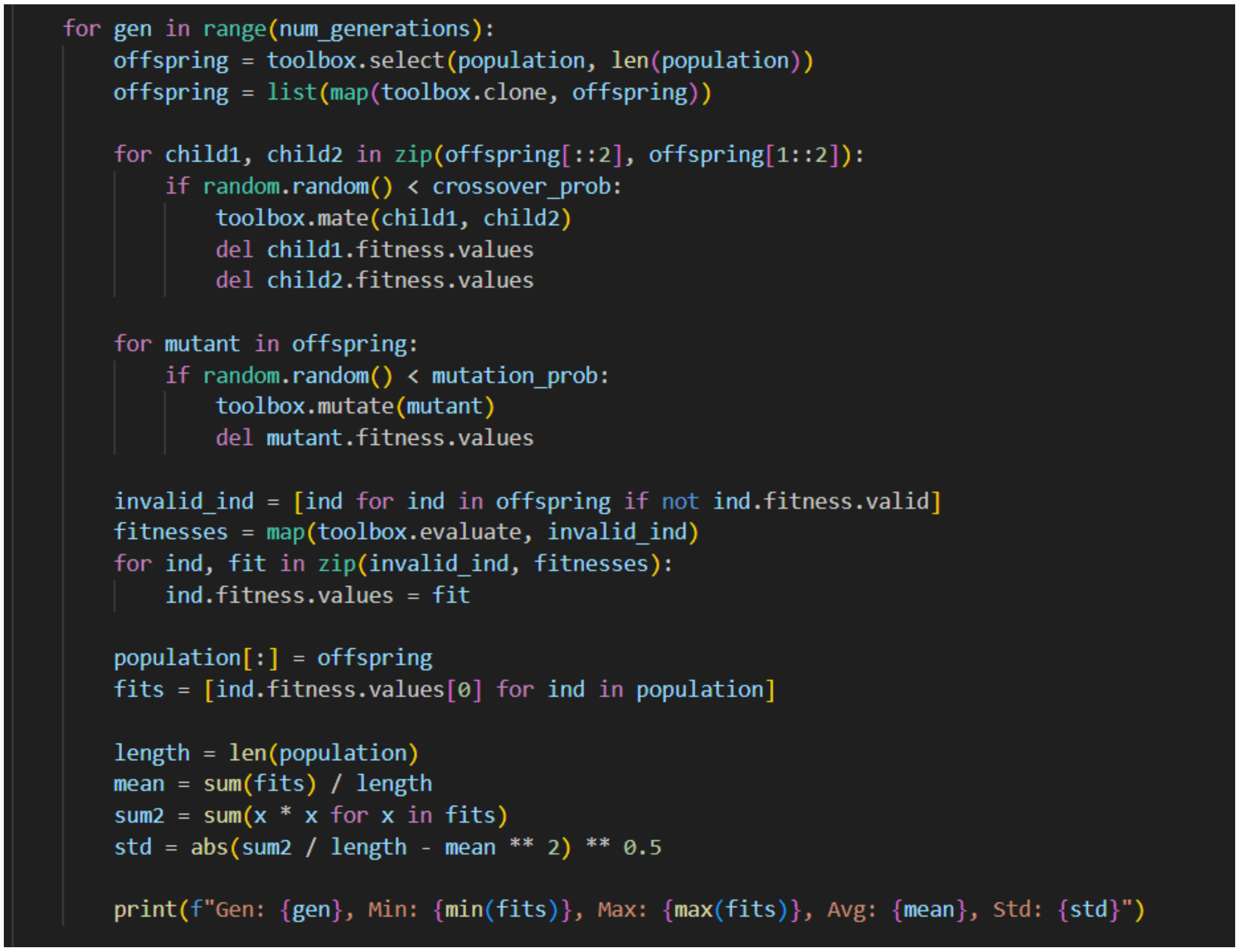
```python
from flask import Flask, request, send_file, redirect, url_for, flash, render_template, send_from_directory
from flask_caching import Cache
import os
import pandas as pd
import random
from deap import base, creator, tools
import requests

app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = 'uploads'
app.secret_key = 'supersecretkey'
cache = Cache(app, config={'CACHE_TYPE': 'simple'})

@app.route('/')
def index():
    return render_template('index.html')

@cache.cached(timeout=300)  # Adjust timeout as needed
def send_static(path):
    return send_from_directory('static', path)

@app.route('/upload', methods=['POST'])
def upload_file():
    if 'file' not in request.files:
        flash('No file part')
        return redirect(url_for('index'))

    file = request.files['file']

    if file.filename == '':
        flash('No selected file')
        return redirect(url_for('index'))

    if file:
        os.makedirs(app.config['UPLOAD_FOLDER'], exist_ok=True)
        filename = os.path.join(app.config['UPLOAD_FOLDER'], file.filename)
        file.save(filename)
        print(f"File saved to {filename}")
```



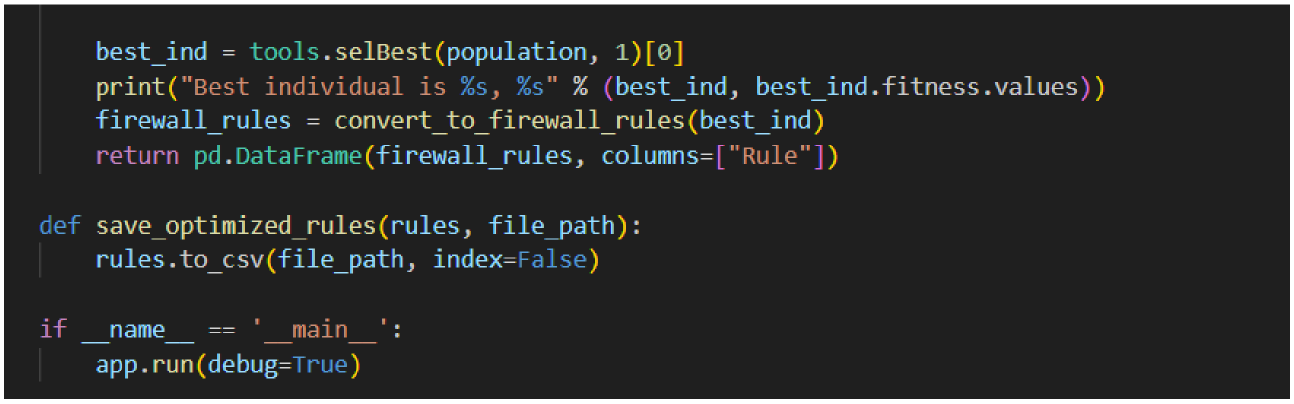
```python
        # Scan file with MetaDefender
        try:
            report = scan_file_with_metadefender(filename)
            print(f"MetaDefender report: {report}")

            # Check if the file is flagged by any antivirus
            if any(result.get('scan_result') == 'malicious' for result in report.get('data', {}).get('scan_results', [])):
                flash('File is infected with malware')
                os.remove(filename)
                return redirect(url_for('index'))

            # Proceed with optimization if file is clean
            optimized_rules = optimize_firewall_rules(filename)
            optimized_file = 'optimized_rules.csv'
            save_optimized_rules(optimized_rules, optimized_file)

            # Automatically download the optimized file
            return send_file(optimized_file, as_attachment=True)

        except Exception as e:
            flash(f"Error processing file: {e}")
            os.remove(filename)
            return redirect(url_for('index'))

def scan_file_with_metadefender(file_path):
    METAEDEFENDER_API_KEY = '0b0f9fa8f8e58967b7f3720603fb8d3a'
    METAEDEFENDER_URL = 'https://api.metadefender.com/v4/file'
    headers = {'apikey': METAEDEFENDER_API_KEY}
    with open(file_path, 'rb') as file:
        files = {'file': file}
        response = requests.post(METAEDEFENDER_URL, headers=headers, files=files)
        if response.status_code == 200:
            result = response.json()
            return result
        else:
            raise Exception(f"Error uploading file: {response.status_code} - {response.text}")
```



```python
def optimize_firewall_rules(file_path):
    df = pd.read_csv(file_path)

    def evaluate(individual):
        score = sum(individual)
        return (score,)

    creator.create("FitnessMax", base.Fitness, weights=(1.0,))
    creator.create("Individual", list, fitness=creator.FitnessMax)

    toolbox = base.Toolbox()
    toolbox.register("attr_bool", random.randint, 0, 1)
    toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.attr_bool, n=len(df))
    toolbox.register("population", tools.initRepeat, list, toolbox.individual)

    toolbox.register("evaluate", evaluate)
    toolbox.register("mate", tools.cxTwoPoint)
    toolbox.register("mutate", tools.mutFlipBit, indpb=0.05)
    toolbox.register("select", tools.selTournament, tournsize=3)

    population_size = 100
    num_generations = 50
    crossover_prob = 0.7
    mutation_prob = 0.2

    def convert_to_firewall_rules(binary_repr):
        firewall_rules = []
        for idx, allow_flag in enumerate(binary_repr):
            if allow_flag == 1:
                port_number = idx + 1
                rule = f"Allow TCP traffic on port {port_number}"
                firewall_rules.append(rule)
        return firewall_rules

    population = toolbox.population(n=population_size)
    fitnesses = list(map(toolbox.evaluate, population))
    for ind, fit in zip(population, fitnesses):
        ind.fitness.values = fit
```

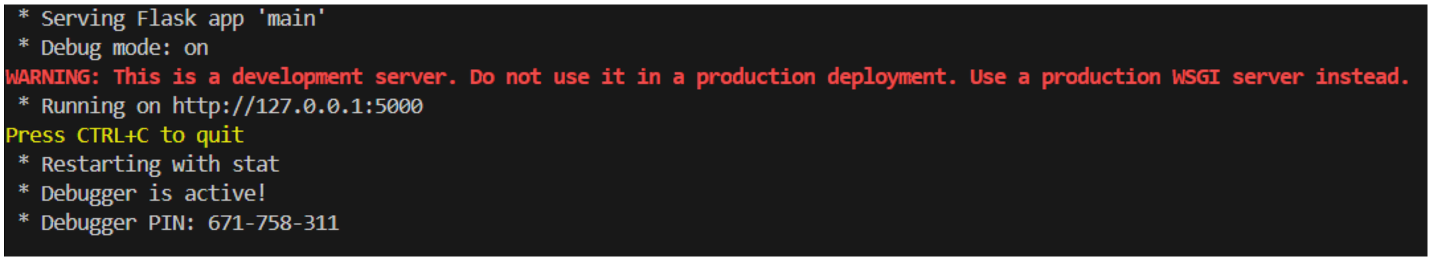


```python
    for gen in range(num_generations):
        offspring = toolbox.select(population, len(population))
        offspring = list(map(toolbox.clone, offspring))

        for child1, child2 in zip(offspring[::2], offspring[1::2]):
            if random.random() < crossover_prob:
                toolbox.mate(child1, child2)
                del child1.fitness.values
                del child2.fitness.values

        for mutant in offspring:
            if random.random() < mutation_prob:
                toolbox.mutate(mutant)
                del mutant.fitness.values

        invalid_ind = [ind for ind in offspring if not ind.fitness.valid]
        fitnesses = map(toolbox.evaluate, invalid_ind)
        for ind, fit in zip(invalid_ind, fitnesses):
            ind.fitness.values = fit

        population[:] = offspring
        fits = [ind.fitness.values[0] for ind in population]

        length = len(population)
        mean = sum(fits) / length
        sum2 = sum(x * x for x in fits)
        std = abs(sum2 / length - mean ** 2) ** 0.5

        print(f"Gen: {gen}, Min: {min(fits)}, Max: {max(fits)}, Avg: {mean}, Std: {std}")
```



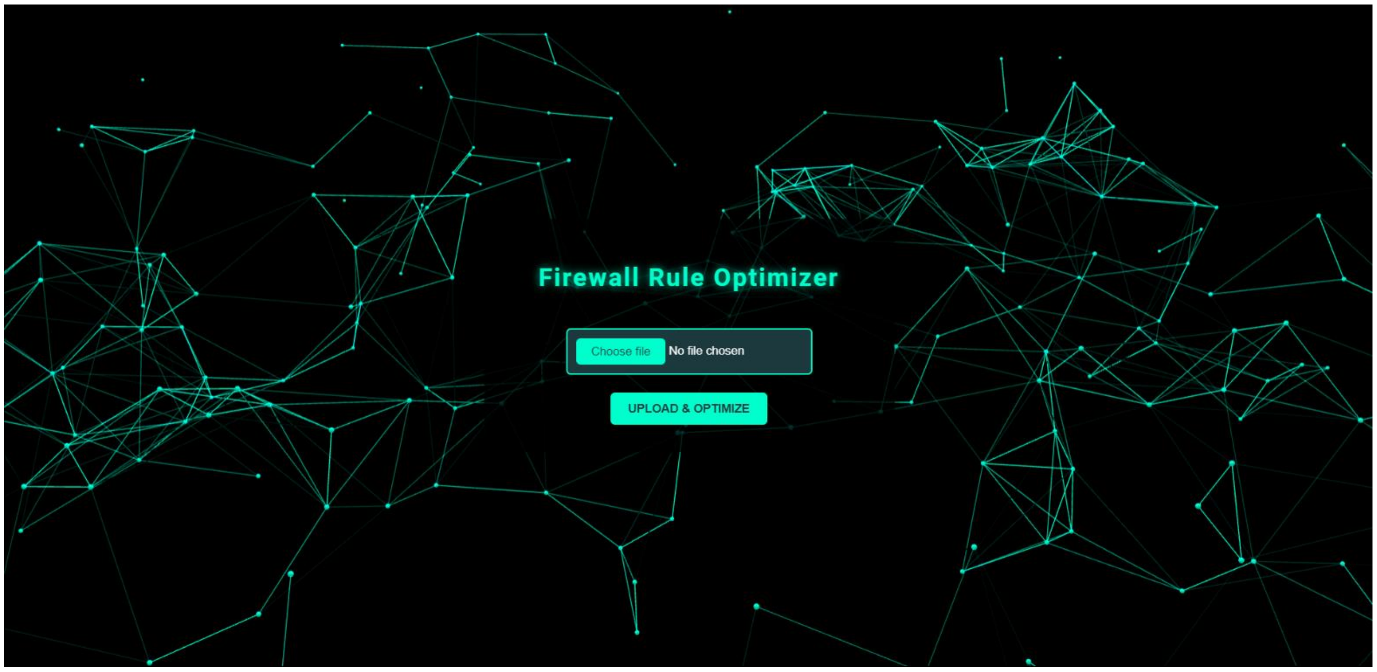
```python
    best_ind = tools.selBest(population, 1)[0]
    print("Best individual is %s, %s" % (best_ind, best_ind.fitness.values))
    firewall_rules = convert_to_firewall_rules(best_ind)
    return pd.DataFrame(firewall_rules, columns=["Rule"])

def save_optimized_rules(rules, file_path):
    rules.to_csv(file_path, index=False)

if __name__ == '__main__':
    app.run(debug=True)
```

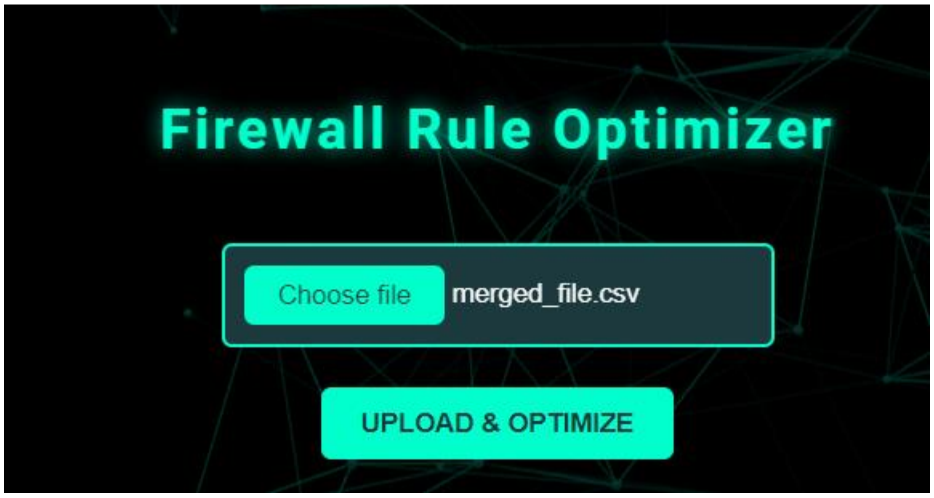
Output :



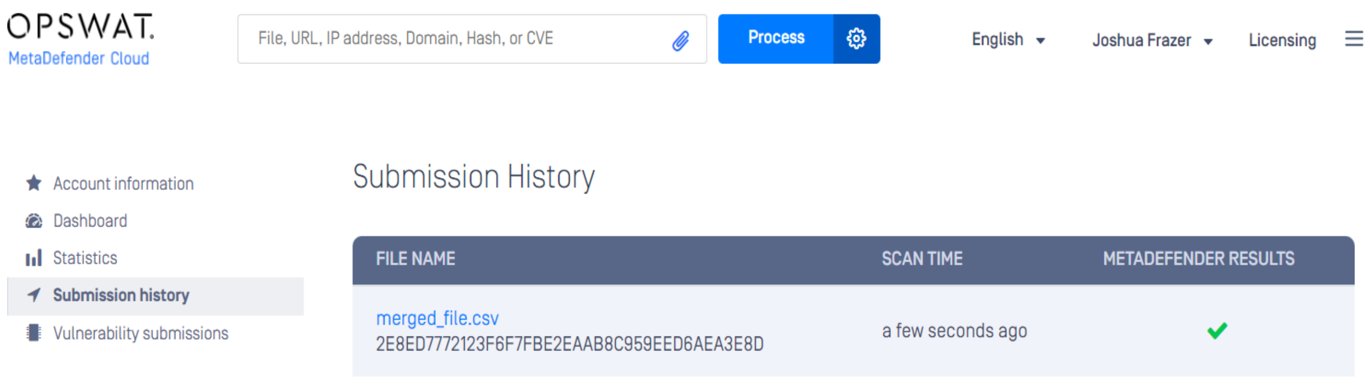
```
 * Serving Flask app 'main'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 671-758-311
```

- From the above outcome snapshot we can see that the local host is running on http://127.0.0.1:5000
- Click or type in that link in the desired web browser and a web page shoud open as shown below





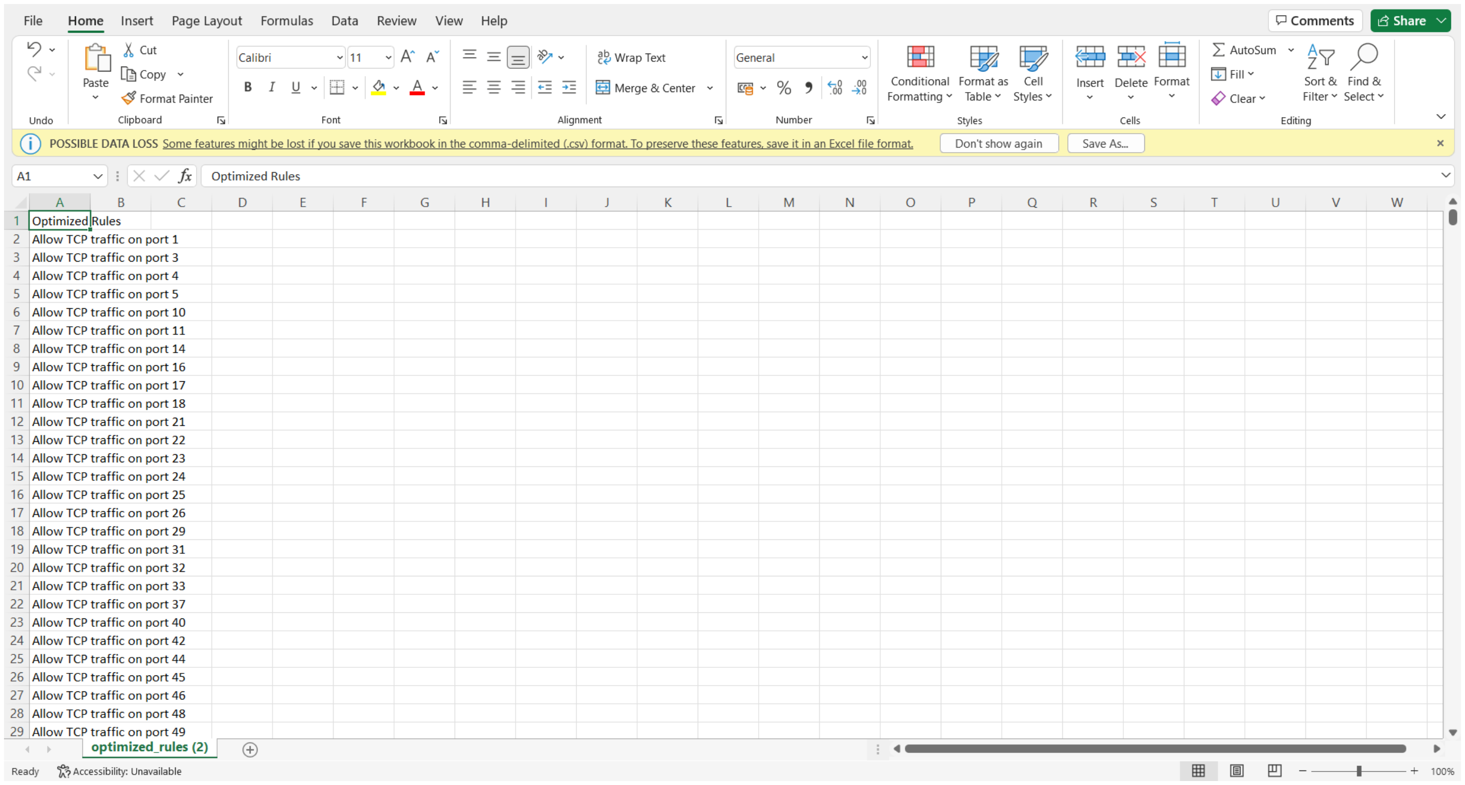
- In the above snapshot the merged_file.csv is selected and the "UPLOAD & OPTIMIZE" button is clicked



Submission History

FILE NAME	SCAN TIME	METADEFENDER RESULTS
merged_file.csv 2E8ED7772123F6F7FBE2EAAB8C959EED6AEA3E8D	a few seconds ago	✓

- Once the file is given then that file is being scanned by the "OPSWAT. MetaDefender Cloud"
- The scan history is being saved and the report and result of the scan can be viewed in the submission history
- If the file is safe then there is a tick mark in the results column and the optimized file is then being downloaded automatically on the system



POSSIBLE DATA LOSS Some features might be lost if you save this workbook in the comma-delimited (.csv) format. To preserve these features, save it in an Excel file format.

A1 | *fx* Optimized Rules

	A
1	Optimized Rules
2	Allow TCP traffic on port 1
3	Allow TCP traffic on port 3
4	Allow TCP traffic on port 4
5	Allow TCP traffic on port 5
6	Allow TCP traffic on port 10
7	Allow TCP traffic on port 11
8	Allow TCP traffic on port 14
9	Allow TCP traffic on port 16
10	Allow TCP traffic on port 17
11	Allow TCP traffic on port 18
12	Allow TCP traffic on port 21
13	Allow TCP traffic on port 22
14	Allow TCP traffic on port 23
15	Allow TCP traffic on port 24
16	Allow TCP traffic on port 25
17	Allow TCP traffic on port 26
18	Allow TCP traffic on port 29
19	Allow TCP traffic on port 31
20	Allow TCP traffic on port 32
21	Allow TCP traffic on port 33
22	Allow TCP traffic on port 37
23	Allow TCP traffic on port 40
24	Allow TCP traffic on port 42
25	Allow TCP traffic on port 44
26	Allow TCP traffic on port 45
27	Allow TCP traffic on port 46
28	Allow TCP traffic on port 48
29	Allow TCP traffic on port 49

- The above snapshot is the final output of the optimized firewall rules
- These rules get saved in a csv file
- With the help of this the optimized firewall rules can be integrated in the system and it reduces network traffic on it as it has less and optimized rules