NETWORK ATTACK DETECTION SYSTEM
CGT PROJECT REPORT

ROLLNO.:106124131

106124037

106124065

COURSE CODE: CSPE01

SECTION: CSE-A

Project Report: Network Attack Detection System (Graph-Based)

Aim of the project:

The Network Attack Detection System is designed to examine computer networks for potential security issues. It analyzes the connections between devices and identifies unusual patterns, such as routing loops or disconnected nodes, which may indicate network attacks or faults.

The system performs three main tasks:

- Detects suspicious routing anomalies (e.g., attack loops in communication paths).
- Verifies complete network connectivity, ensuring no devices are left isolated.
- Builds a secure and cost-efficient network layout by minimizing risk using graph optimization.

By using graph-based algorithms, the system can assess the security and efficiency of a network, making it useful in cybersecurity simulations, IoT systems, and large-scale communication networks.

Implementation

The project is implemented in **C++**, using a graph representation for the network where:

- Each **node** represents a network device.
- Each **edge** represents a connection between devices, with a **weight** indicating cost or risk.

Steps of Implementation

1. **User Input:**
   The user enters the number of devices and the connections between them along with their weights.
2. **Graph Representation:**
   - A struct Edge is used to represent each connection (source, destination, and weight).
   - A vector<Edge> stores all the connections.
   - An adjacency list (vector<vector<int>> adj) is maintained for connectivity checks.
3. **Algorithms Used:**

   - **Bellman-Ford Algorithm:** Detects *negative cycles* in the network, which could represent potential attack loops or routing anomalies.
   - **Depth-First Search (DFS):** Verifies that all devices are reachable, ensuring no part of the network is isolated.
   - **Kruskal's Algorithm (Minimum Spanning Tree):** Suggests a minimal-cost, secure layout for the network, reducing both risk and redundancy.

Key Code Snippets

- ***Graph Representation:***

```
struct Edge {
   int src, dest, weight;
};
vector<Edge> edges(E);
vector<vector<int>> adj(V);
```

- ***Bellman-Ford for Anomaly Detection*:**

```
void bellmanFord(vector<Edge> &edges, int V, int src) {
   vector<int> dist(V, INT_MAX);
   dist[src] = 0;

   for (int i = 0; i < V - 1; i++)
      for (auto &e : edges)
         if (dist[e.src] != INT_MAX && dist[e.src] + e.weight <
dist[e.dest])
            dist[e.dest] = dist[e.src] + e.weight;

   bool attack = false;
   for (auto &e : edges)
      if (dist[e.src] != INT_MAX && dist[e.src] + e.weight <
dist[e.dest])
         attack = true;

   if (attack)
      cout << "Alert: Suspicious negative cycle detected!\n";
   else
      cout << "No anomalies detected in routing paths.\n";}
```

- ***DFS for Connectivity Check:***

```
void dfsUtil(int v, vector<vector<int>> &adj, vector<bool>
&visited) {
   visited[v] = true;
   for (int u : adj[v])
      if (!visited[u])
         dfsUtil(u, adj, visited);
}
```

- ***Kruskal's Algorithm (Secure Minimal Network):***

```
void kruskal(vector<Edge> &edges, int V) {
   sort(edges.begin(), edges.end(), [](Edge a, Edge b) { return
a.weight < b.weight; });
   DSU dsu(V);
   int cost = 0;

   cout << "\nSecure Minimal Network Design (MST):\n";
   for (auto &e : edges)
      if (dsu.find(e.src) != dsu.find(e.dest)) {
         dsu.unite(e.src, e.dest);
         cout << "  " << e.src << " -- " << e.dest << "  (Weight: " <<
e.weight << ")\n";
         cost += e.weight;
      }
 cout << "Total Secure Network Cost: " << cost << endl;}
```

- ***Sample Input***

```
Enter number of devices (nodes): 5
Enter number of connections (edges): 7
Enter connections (src dest weight):
0 1 6
0 3 7
1 2 5
1 3 8
1 4 -4
2 4 3
3 4 9
```

Output

```
Alert: Suspicious negative cycle detected (Possible Attack Loop)!
All devices are connected (No isolation detected).

Secure Minimal Network Design (MST):
  1 -- 4  (Weight: -4)
  2 -- 4  (Weight: 3)
  0 -- 1  (Weight: 6)
  0 -- 3  (Weight: 7)
Total Secure Network Cost: 12
```

Network Representation

A simple text-based representation of the sample network:

```
       (0)
      /   \
    (1)  (3)
    / \
 (4) (2)
```

- Edges are labeled with weights, for example:
  0 -- 1 (6), 1 -- 4 (-4)
- Negative edges highlight risk or attack-prone links.

Github repo:

https://github.com/Ameesha2007/network_attack_detection_system

Real-Life Applications

- **Cybersecurity Monitoring:**
  Detects potential routing loops that could be exploited by attackers to disrupt traffic.
- **Network Optimization:**
  Builds efficient communication paths with minimal cost and risk.
- **IoT Security:**
  Ensures that all IoT devices remain connected without any isolated or compromised nodes.

- **Critical Infrastructure Protection:**
  Used in power grids, transportation systems, and communication networks to identify vulnerabilities.

Future Enhancements:

To make this system more practical for real-world use, the following features can be added:

1. **Real-Time Packet Monitoring:** Integrate live packet data to detect ongoing attacks dynamically.
2. **Machine Learning-Based Anomaly Detection:** Use trained models to identify suspicious patterns.
3. **Network Visualization Dashboard:** Implement an interactive GUI for real-time graph visualization.
4. **Integration with Firewalls:** Automatically block or isolate detected malicious connections.

Conclusion:

The Network Attack Detection System helps in finding and reducing security problems in a computer network. It checks for unusual paths, disconnected devices, and builds a safe, low-cost network layout. By using simple graph-based algorithms, it makes it easier to understand how data moves in a network and where issues might occur.