Project Report

On

Data Leakage Detection

Submitted in partial fulfilment of the requirements for the award of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

(Artificial Intelligence & Machine Learning)

by

Ms. P. SNEHA – 23WH5A6603

Ms. P. KEERTHANA – 23WH5A6605

Ms. P. SUSHMA – 23WH5A6606

Ms. T. RAJESHWARI – 23WH5A6607

Under the esteemed guidance of

Ms. P Anusha

Assistant Professor, CSE(AI&ML)

VISHNU
UNIVERSAL LEARNING
BVRIT H
Estd. 2012

BVRIT HYDERABAD College of Engineering for Women

(UGC Autonomous Institution | Approved by AICTE | Affiliated to JNTUH)

(NAAC Accredited - A Grade | NBA Accredited B.Tech. (EEE, ECE, CSE and IT)

Bachupally, Hyderabad – 500090

2024-25

ABSTRACT

This project focuses on creating a data management and leakage detection system for monitoring the allocation of sensitive data to various agents. The system is designed to allocate specific data entries and alterations to agents, enabling the tracking of sensitive information throughout its lifecycle. The project is structured using a series of data structures representing agents, their allocated data, and alterations.

The core functionality of the system includes:

1. Agent Creation: Each agent is associated with a unique ID and name, and they are allocated a set of data entries and their corresponding alterations.
2. Data Allocation: Agents can be assigned multiple data entries, each with an alteration, which simulates a scenario where sensitive data is distributed across various entities.
3. Leakage Detection: The system detects any potential data leakage by comparing a leaked data entry and its alteration with the data allocated to agents. If a match is found, the system identifies the agent responsible for the leak.

The project utilizes dynamic memory allocation to manage agents and their data entries and provides a basic mechanism to store and compare data. It aims to provide a simple yet effective method for tracking and detecting potential security breaches related to sensitive data handling. This system could be extended to support more complex data management tasks in real-world applications, especially in areas requiring high data security and confidentiality.

Problem Statement:

Data leakage poses a significant threat to organizations, leading to potential breaches of sensitive information and loss of trust. In multi-agent systems, where data is shared among multiple entities, identifying the source of a leak becomes challenging. Current solutions often lack efficiency and scalability in pinpointing responsible agents when unauthorized data dissemination occurs.

This project aims to develop a Data Leakage Detection System that tracks data allocations to agents, monitors alterations, and identifies leaks by analyzing leaked data and alterations. The system provides a user-friendly web interface for managing agents, allocating data, and detecting leaks, ensuring secure and reliable data governance.

Functional Requirements

1. **Agent Management**

- Allow users to add, edit, and manage agent details (e.g., agent ID, name).
- Allocate data and corresponding alterations to agents.

2. **Data Storage and Retrieval**

- Store data entries allocated to each agent in an organized and secure manner.
- Retrieve and compare stored data during leakage detection.

3. **Leakage Detection**

- Input suspected leaked data and its alteration.
- Compare leaked data with stored entries to identify the source of the leak.
- Provide details of the responsible agent if a leak is detected.

4. **Terminal-Based Interaction**

- Enable users to interact with the system through a command-line interface.
- Accept inputs for agent details, data allocation, and leakage verification.

5. **Dynamic Data Allocation**

- Support multiple agents with variable numbers of data entries per agent.

Non-Functional Requirements

1. Performance

- Ensure efficient processing for data comparison and retrieval, even for a large number of agents and data entries.

2. Scalability

- Design the system to handle additional agents and data without significant performance degradation.

3. Reliability

- Ensure accurate identification of data leaks with minimal false positives or negatives.

4. Memory Optimization

- Use dynamic memory management to optimize resource usage and prevent memory leaks.

5. Usability

- Provide a clear and user-friendly interface for smooth interaction in the terminal.

6. Maintainability

- Structure the code to allow easy updates, enhancements, or bug fixes in the future.

7. Portability

- Ensure the program runs seamlessly on different operating systems with a C compiler.

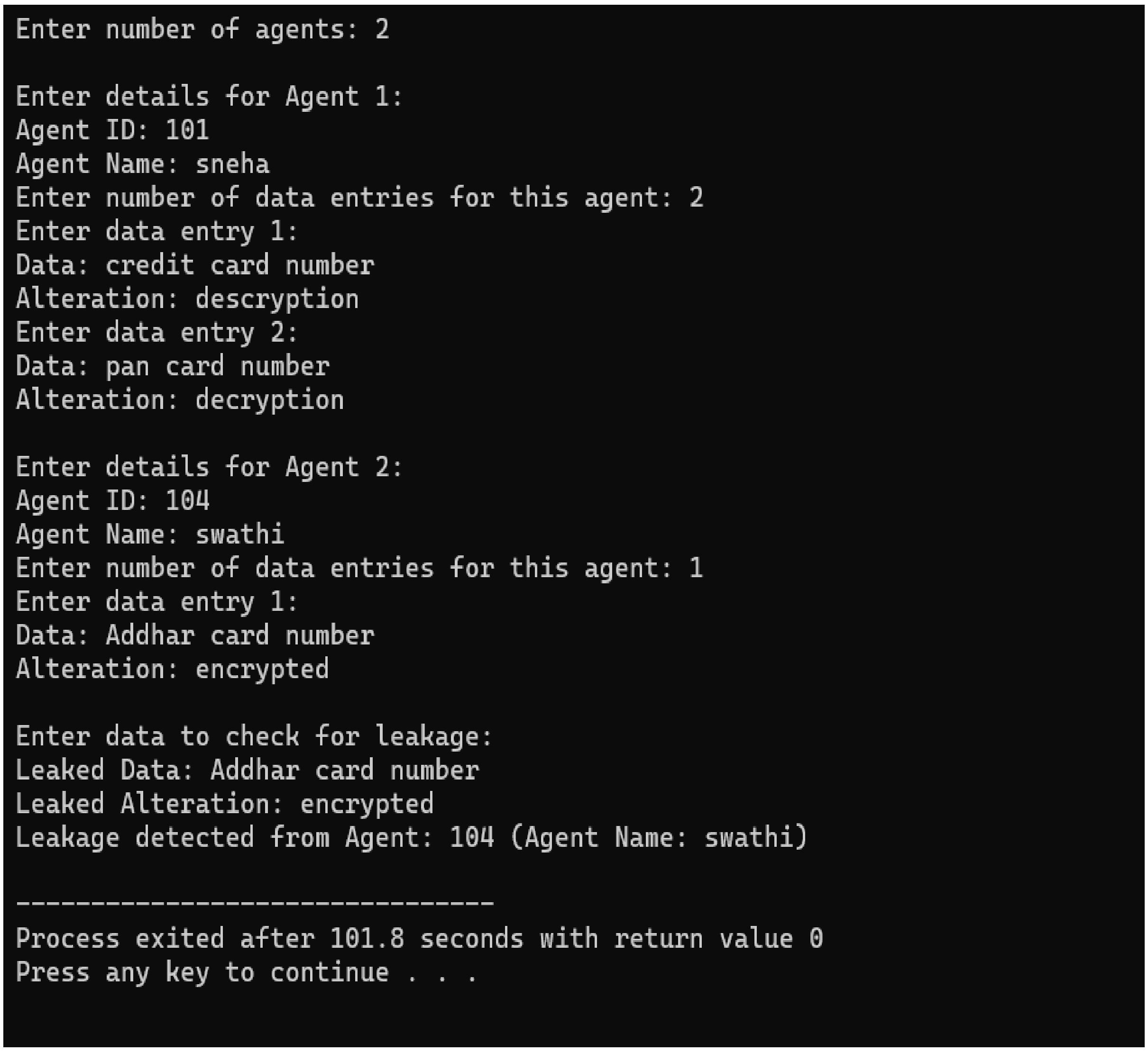
SOURCE CODE

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_DATA 100
typedef struct {
    char *data;
    char *alteration;
} DataEntry;
typedef struct {
    char *agentId;
    char *agentName;
    DataEntry allocatedData[MAX_DATA];
    int dataCount;
} Agent;
typedef struct AgentNode {
    Agent *agent;
    struct AgentNode *next;
} AgentNode;
typedef struct {
    AgentNode *storage;
} DataStorage;
typedef struct {
    DataStorage *dataStorage;
} LeakageDetector;
// Function to create a new agent
Agent* createAgent(const char *agentId, const char *agentName) {
    Agent *agent = (Agent *)malloc(sizeof(Agent));
    agent->agentId = strdup(agentId);
    agent->agentName = strdup(agentName);
    agent->dataCount = 0;
    return agent;
}
// Function to allocate data and its alteration to the agent
void allocateData(Agent *agent, const char *data, const char *alteration) {
    if (agent->dataCount < MAX_DATA) {
        agent->allocatedData[agent->dataCount].data = strdup(data);
        agent->allocatedData[agent->dataCount].alteration = strdup(alteration);
        agent->dataCount++;
    }
}
// Function to create a data storage
DataStorage* createDataStorage() {
    DataStorage *ds = (DataStorage *)malloc(sizeof(DataStorage));
    ds->storage = NULL;
    return ds;
}
// Function to store agent data in the data storage
void storeData(DataStorage *ds, Agent *agent) {
    AgentNode *newNode = (AgentNode *)malloc(sizeof(AgentNode));
    newNode->agent = agent;
    newNode->next = ds->storage;
    ds->storage = newNode;
}
// Function to create a leakage detector
LeakageDetector* createLeakageDetector(DataStorage* dataStorage) {
    LeakageDetector* detector = (LeakageDetector*)malloc(sizeof(LeakageDetector));
    detector->dataStorage = dataStorage;
    return detector;
}
// Function to detect leakage by comparing leaked data
```

```
void detectLeakage(LeakageDetector* detector, const char* leakedData, const char*
alteration) {
    int leakDetected = 0;
    AgentNode *current;
    // Iterate through all stored agents and their data entries
    for (current = detector->dataStorage->storage; current != NULL; current = current->next) {
        Agent *agent = current->agent;
        int i;
        for (i = 0; i < agent->dataCount; i++) {
            if (strcmp(leakedData, agent->allocatedData[i].data) == 0 &&
                strcmp(alteration, agent->allocatedData[i].alteration) == 0) {
                printf("Leakage detected from Agent: %s (Agent Name: %s)\n",
                       agent->agentId, agent->agentName);
                leakDetected = 1;
                break;
            }
        }
        if (leakDetected) break;
    }
    if (!leakDetected) {
        printf("No leakage detected for data: %s with alteration: %s\n",
               leakedData, alteration);
    }
}
// Function to free the memory allocated for an agent
void freeAgent(Agent *agent) {
    int i;
    free(agent->agentId);
    free(agent->agentName);
    for (i = 0; i < agent->dataCount; i++) {
        free(agent->allocatedData[i].data);
        free(agent->allocatedData[i].alteration);
    }
    free(agent);
}
// Function to free the memory allocated for the data storage
void freeDataStorage(DataStorage *ds) {
    AgentNode *current = ds->storage;
    while (current != NULL) {
        AgentNode *temp = current;
        current = current->next;
        freeAgent(temp->agent);
        free(temp);
    }
    free(ds);
}
int main() {
    DataStorage *dataStorage = createDataStorage();
    LeakageDetector *leakageDetector = createLeakageDetector(dataStorage);
    int numAgents;
    int i, j;
    printf("Enter number of agents: ");
    scanf("%d", &numAgents);
    getchar(); // To consume the newline character left by scanf
    // Input agent details and allocate data
    for (i = 0; i < numAgents; i++) {
        char agentId[50], agentName[50];
        int numDataEntries;
        printf("\nEnter details for Agent %d:\n", i + 1);
        printf("Agent ID: ");
        fgets(agentId, sizeof(agentId), stdin);
        agentId[strcspn(agentId, "\n")] = 0;
        printf("Agent Name: ");
        fgets(agentName, sizeof(agentName), stdin);
        agentName[strcspn(agentName, "\n")] = 0;

        Agent *agent = createAgent(agentId, agentName);
        printf("Enter number of data entries for this agent: ");
        scanf("%d", &numDataEntries);
        getchar(); // To consume the newline character left by scanf
        for (j = 0; j < numDataEntries; j++) {
            char data[100], alteration[100];
            printf("Enter data entry %d:\n", j + 1);
            printf("Data: ");
            fgets(data, sizeof(data), stdin);
            data[strcspn(data, "\n")] = 0;
            printf("Alteration: ");
            fgets(alteration, sizeof(alteration), stdin);
            alteration[strcspn(alteration, "\n")] = 0;
            allocateData(agent, data, alteration);
        }
        // Store agent in data storage
        storeData(dataStorage, agent);
    }
    // Simulate leakage detection
    char leakedData[100], leakedAlteration[100];
    printf("\nEnter data to check for leakage:\n");
    printf("Leaked Data: ");
    fgets(leakedData, sizeof(leakedData), stdin);
    leakedData[strcspn(leakedData, "\n")] = 0;
    printf("Leaked Alteration: ");
    fgets(leakedAlteration, sizeof(leakedAlteration), stdin);
    leakedAlteration[strcspn(leakedAlteration, "\n")] = 0;
    // Detect leakage
    detectLeakage(leakageDetector, leakedData, leakedAlteration);
    // Free allocated memory
    freeDataStorage(dataStorage);
    free(leakageDetector);
  return 0;
}
```

Output



```
Enter number of agents: 2

Enter details for Agent 1:
Agent ID: 101
Agent Name: sneha
Enter number of data entries for this agent: 2
Enter data entry 1:
Data: credit card number
Alteration: descryption
Enter data entry 2:
Data: pan card number
Alteration: decryption

Enter details for Agent 2:
Agent ID: 104
Agent Name: swathi
Enter number of data entries for this agent: 1
Enter data entry 1:
Data: Addhar card number
Alteration: encrypted

Enter data to check for leakage:
Leaked Data: Addhar card number
Leaked Alteration: encrypted
Leakage detected from Agent: 104 (Agent Name: swathi)

--------------------------------
Process exited after 101.8 seconds with return value 0
Press any key to continue . . .
```