PROJECT REPORT

Title: Security Vulnerability Detection Framework

Operating System-(CSE316)

By

By ***Sr. No.***	***Registration No.***	***Name of Student***	***Roll No.***
1	12406393	Rijo	24
2	12406893	Prem yadav	25
3	12417509	Shreesti	26

Submitted to: *Prof. Nitish Chandok*

Lovely Professional University, Jalandhar, Punjab, India

Project Report

Project Overview

With the increasing use of computers, networks, and online services, system security has become a major concern. Operating systems and applications are often targeted by attackers to exploit vulnerabilities such as buffer overflows, unauthorized access, malware injection, cache poisoning, and trapdoors. If these vulnerabilities are not detected in time, they can lead to data loss, system crashes, and security breaches.

The **Security Vulnerability Detection Framework** is a simulation-based project that focuses on identifying and alerting potential security threats in a system. The framework monitors different system activities such as memory usage, user access behavior, and suspicious operations. When an abnormal or malicious activity is detected, the system generates real-time alerts to notify the user or administrator.

This project demonstrates how vulnerability detection works at a basic level using operating system concepts such as process monitoring, memory protection, and access control. The framework is designed to help students understand security threats and their detection mechanisms in a simple and practical manner.

The main objective of this project is to simulate common security vulnerabilities and provide an alert-based detection mechanism to improve system awareness and protection.

2. Module-Wise Breakdown

The project is divided into three major modules:

1. Vulnerability Monitoring Module

This module continuously monitors system activities such as memory access, input handling, and process behavior. It checks for abnormal patterns that may indicate vulnerabilities like buffer overflow attempts or unauthorized memory access.

2. Attack Detection Module

This module analyzes the monitored data to detect known attack patterns such as brute-force login attempts, cache poisoning, and trapdoor access. If suspicious behavior is identified, the module classifies the type of attack.

3. Alert and Mitigation Module

Once a vulnerability or attack is detected, this module generates real-time alerts with details such as attack type and time of detection. It also suggests basic preventive measures such as restricting access, terminating suspicious processes, or strengthening input validation.

Functionalities

- Monitors system behavior for suspicious activities
- Detects common vulnerabilities such as buffer overflow and unauthorized access
- Generates real-time security alerts
- Displays attack type and detection time
- Simulates basic mitigation and prevention strategies
- Provides a simple console-based interface

Technology Used

- **Programming Language:** C++
- **Concepts Used:**
  - Operating System Security
  - Process Monitoring
  - Memory Protection
  - Access Control
- **Libraries Used:**
  - `<iostream>`
  - `<cstring>`
  - `<ctime>`
- **Compiler:** GCC / G++
- **Platform:** Windows / Linux

Flow Diagram Description

The program starts by initializing the monitoring system. It continuously checks system activities for suspicious behavior. If normal activity is detected, the system continues monitoring. If a potential vulnerability is identified, the detection module analyzes the threat. Once confirmed, the alert module displays a security warning along with attack details and time. The system then returns to monitoring mode.

Revision Tracking on GitHub

The project is maintained using GitHub for version control. Different commits track the development of modules such as vulnerability detection logic, alert generation, and testing. This ensures proper tracking of changes and improves code management.

Repository Name: Security-Vulnerability-Detection-Framework
GitHub Link:https://github.com/premyadav07/Operating-project

Conclusion and Future Scope

The **Security Vulnerability Detection Framework** successfully demonstrates how security threats can be detected in an operating system environment. The project provides a basic understanding of how vulnerabilities are monitored, identified, and reported. It helps students gain practical knowledge of system security and threat detection mechanisms.

In the future, the framework can be enhanced by adding real-time logging, encryption techniques, intrusion detection systems (IDS), and machine learning-based attack detection. A graphical user interface and network-based attack simulation can also be included for better visualization.

References

- Silberschatz, Galvin, Gagne – *Operating System Concepts*
- Andrew S. Tanenbaum – *Modern Operating Systems*
- NPTEL – Operating System & Cyber Security Courses
- GeeksforGeeks – System Security and Vulnerability Detection

Appendix A: AI-Generated Project Elaboration / Breakdown

This project was developed with the assistance of AI guidance to understand security vulnerability concepts and system-level threat detection. AI helped in structuring the project,

defining modules, and designing the detection and alert mechanisms. The guidance simplified complex security concepts and helped in organizing the project in a clear and systematic manner.

Appendix B: Problem Statement

The **Security Vulnerability Detection Framework** is designed to simulate the detection of common security vulnerabilities in an operating system. The system monitors activities, detects suspicious behavior, identifies potential attacks, and generates alerts. The main objective is to provide a practical understanding of security threats and their detection using operating system principles.

Appendix C: Complete Code

Below is the complete **C++ implementation** of the Security Vulnerability Detection Framework used in this project:

```
#include <iostream>
#include <cstring>
#include <fstream>
#include <vector>
#include <map>
using namespace std;

/* ====================== GLOBAL CONSTANTS
====================== */
const int BUFFER_LIMIT = 8;
const int MEMORY_THRESHOLD = 80;
const double CPU_THRESHOLD = 2.0;

/* ====================== UTILITY FUNCTION
====================== */
void printLine() {
    cout <<
"-----------------------------------------------\n";
}

/* ====================== BUFFER OVERFLOW MODULE
====================== */
```

```
void bufferOverflowSimulation() {
    printLine();
    cout << "[MODULE] Buffer Overflow Simulation\n";

    char buffer[BUFFER_LIMIT];
    cout << "Enter input (max 8 characters): ";
    cin >> buffer;   // intentionally unsafe

    cout << "Stored Data: " << buffer << endl;

    if (strlen(buffer) > BUFFER_LIMIT) {
        cout << "[ALERT] Buffer Overflow Detected!\n";
    } else {
        cout << "[INFO] Input within safe limit\n";
    }
}

/* ==================== TRAPDOOR MODULE
==================== */
void trapdoorSimulation() {
    printLine();
    cout << "[MODULE] Trapdoor (Backdoor) Simulation\n";

    char username[20], password[20];
    cout << "Username: ";
    cin >> username;
    cout << "Password: ";
    cin >> password;

    if (strcmp(username, "admin") == 0 &&
        strcmp(password, "backdoor123") == 0) {
        cout << "[ALERT] Trapdoor Access Granted!\n";
    } else {
        cout << "[INFO] Authentication Failed\n";
    }
}

/* ==================== CACHE POISONING MODULE
==================== */
void cachePoisoningSimulation() {
```

```
    printLine();
    cout << "[MODULE] Cache Poisoning Simulation\n";

    string domain = "example.com";
    string originalIP = "93.184.216.34";
    string poisonedIP = "192.168.1.100";

    cout << "Original Cache:\n";
    cout << domain << " -> " << originalIP << endl;

    cout << "[WARNING] Cache Entry Modified!\n";
    cout << domain << " -> " << poisonedIP << endl;
}

/* ====================== SYSTEM MONITORING MODULE
====================== */
double getCPULoad() {
    ifstream file("/proc/loadavg");
    double load;
    file >> load;
    return load;
}

void systemMonitoring() {
    printLine();
    cout << "[MODULE] System Monitoring\n";

    double cpuLoad = getCPULoad();
    cout << "CPU Load: " << cpuLoad << endl;

    if (cpuLoad > CPU_THRESHOLD) {
        cout << "[ALERT] High CPU Usage Detected!\n";
    } else {
        cout << "[INFO] CPU Usage Normal\n";
    }
}

/* ====================== ANOMALY DETECTION MODULE
====================== */
void anomalyDetection() {
```

```
    printLine();
    cout << "[MODULE] Anomaly Detection\n";

    int memoryUsage = 90;   // simulated value
    cout << "Memory Usage: " << memoryUsage << "%\n";

    if (memoryUsage > MEMORY_THRESHOLD) {
        cout << "[ALERT] Anomalous Memory Usage Detected!
\n";
        cout << "Possible Cause: Buffer Overflow or
Malware\n";
    } else {
        cout << "[INFO] Memory Usage Normal\n";
    }
}

/* ==================== MITIGATION MODULE
==================== */
void mitigation(const string& vulnerability) {
    printLine();
    cout << "[MODULE] Mitigation & Prevention\n";

    map<string, vector<string>> fixes;

    fixes["Buffer Overflow"] = {
        "Use bounds checking",
        "Avoid unsafe functions like gets()",
        "Enable ASLR"
    };

    fixes["Trapdoor"] = {
        "Remove hardcoded credentials",
        "Audit authentication logic"
    };

    fixes["Cache Poisoning"] = {
        "Flush cache",
        "Enable DNSSEC",
        "Validate DNS responses"
    };
```

```
    if (fixes.count(vulnerability)) {
        cout << "Mitigation steps for " << vulnerability <<
":\n";
        for (auto &step : fixes[vulnerability]) {
            cout << "- " << step << endl;
        }
    } else {
        cout << "No mitigation available\n";
    }
}

/* ====================== MAIN CONTROLLER
====================== */
int main() {
    int choice;

    while (true) {
        printLine();
        cout << " Security Vulnerability Detection
Framework\n";
        printLine();

        cout << "1. Buffer Overflow Simulation\n";
        cout << "2. Trapdoor Simulation\n";
        cout << "3. Cache Poisoning Simulation\n";
        cout << "4. System Monitoring\n";
        cout << "5. Anomaly Detection\n";
        cout << "6. Exit\n";

        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                bufferOverflowSimulation();
                mitigation("Buffer Overflow");
                break;

            case 2:
```

```
                trapdoorSimulation();
                mitigation("Trapdoor");
                break;

            case 3:
                cachePoisoningSimulation();
                mitigation("Cache Poisoning");
                break;

            case 4:
                systemMonitoring();
                break;

            case 5:
                anomalyDetection();
                mitigation("Buffer Overflow");
                break;

            case 6:
                cout << "Exiting Framework...\n";
                return 0;

            default:
                cout << "Invalid choice! Try again.\n";
        }
    }
}
```

OUTPUT SCREENSHOT;

```
-----------------------------------------------
-------
 Security Vulnerability Detection Framework
-----------------------------------------------
1. Buffer Overflow Simulation
2. Trapdoor Simulation
3. Cache Poisoning Simulation
```

```
4. System Monitoring
5. Anomaly Detection
6. Exit
Enter your choice: 1
------------------------------------------------
[MODULE] Buffer Overflow Simulation
Enter input (max 8 characters): abcdefghi
Stored Data: abcdefghi
[ALERT] Buffer Overflow Detected!
------------------------------------------------
[MODULE] Mitigation & Prevention
Mitigation steps for Buffer Overflow:
- Use bounds checking
- Avoid unsafe functions like gets()
- Enable ASLR

------------------------------------------------
 Security Vulnerability Detection Framework
------------------------------------------------
1. Buffer Overflow Simulation
2. Trapdoor Simulation
3. Cache Poisoning Simulation
4. System Monitoring
5. Anomaly Detection
6. Exit
Enter your choice: 2
------------------------------------------------
[MODULE] Trapdoor (Backdoor) Simulation
Username: admin
Password: backdoor123
[ALERT] Trapdoor Access Granted!
------------------------------------------------
[MODULE] Mitigation & Prevention
Mitigation steps for Trapdoor:
- Remove hardcoded credentials
- Audit authentication logic

------------------------------------------------
```

```
 Security Vulnerability Detection Framework
------------------------------------------------
1. Buffer Overflow Simulation
2. Trapdoor Simulation
3. Cache Poisoning Simulation
4. System Monitoring
5. Anomaly Detection
6. Exit
Enter your choice: 3
------------------------------------------------
[MODULE] Cache Poisoning Simulation
Original Cache:
example.com -> 93.184.216.34
[WARNING] Cache Entry Modified!
example.com -> 192.168.1.100
------------------------------------------------
[MODULE] Mitigation & Prevention
Mitigation steps for Cache Poisoning:
- Flush cache
- Enable DNSSEC
- Validate DNS responses

------------------------------------------------
 Security Vulnerability Detection Framework
------------------------------------------------
1. Buffer Overflow Simulation
2. Trapdoor Simulation
3. Cache Poisoning Simulation
4. System Monitoring
5. Anomaly Detection
6. Exit
Enter your choice: 4
------------------------------------------------
[MODULE] System Monitoring
CPU Load: 1.25
[INFO] CPU Usage Normal

------------------------------------------------
```

```
 Security Vulnerability Detection Framework
------------------------------------------------
1. Buffer Overflow Simulation
2. Trapdoor Simulation
3. Cache Poisoning Simulation
4. System Monitoring
5. Anomaly Detection
6. Exit
Enter your choice: 5
------------------------------------------------
[MODULE] Anomaly Detection
Memory Usage: 90%
[ALERT] Anomalous Memory Usage Detected!
Possible Cause: Buffer Overflow or Malware
------------------------------------------------
[MODULE] Mitigation & Prevention
Mitigation steps for Buffer Overflow:
- Use bounds checking
- Avoid unsafe functions like gets()
- Enable ASLR

------------------------------------------------
 Security Vulnerability Detection Framework
------------------------------------------------
1. Buffer Overflow Simulation
2. Trapdoor Simulation
3. Cache Poisoning Simulation
4. System Monitoring
5. Anomaly Detection
6. Exit
Enter your choice: 6
Exiting Framework...
```