Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Информационные сети. Основы безопасности

ОТЧЁТ
к лабораторной работе №5
на тему

ЗАЩИТА ОТ АТАКИ НА ПЕРЕПОЛНЕНИЕ БУФЕРА

Выполнил: студент гр.253501
Станишевский А.Д.

Проверил: ассистент кафедры информатики
Герчик А.В.

Минск 2025

СОДЕРЖАНИЕ

1 ЦЕЛЬ РАБОТЫ

Целью работы является исследование методов защиты от атак на переполнение буфера и разработка демонстрационного программного решения, иллюстрирующего эксплуатацию уязвимостей, связанных с некорректной обработкой данных в стеке, а также методы их предотвращения.

В рамках работы реализованы две версии программы (уязвимая и защищённая), позволяющие провести сравнительный анализ эффективности защитных механизмов. Программа моделирует сценарий атаки, при котором переполнение буфера приводит к изменению критической переменной, и демонстрирует методы защиты.

Особое внимание уделено реализации функционала, наглядно демонстрирующего этапы эксплуатации уязвимости и работу защитных механизмов. Программа предоставляет возможность анализа состояния памяти в процессе выполнения, что позволяет отследить перезапись переменных в стеке и визуализировать срабатывание мер безопасности.

Таким образом, в ходе работы получены навыки анализа и предотвращения атак на переполнение буфера, а также создана программа, иллюстрирующая базовые принципы защиты памяти.

2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Буфер – это блок памяти, назначенный программной программе операционной системой. Программа обязана запросить у операционной системы объем памяти, необходимый для правильной работы. В некоторых языках программирования, таких как Java, C #, Python, Go и Rust, управление памятью выполняется автоматически. В других языках, таких как C и C ++, программисты должны вручную управлять выделением и освобождением памяти и следить за тем, чтобы границы памяти не пересекались, проверяя длину буфера.

Во время таких атак переполняется буфер с фиксированным объемом памяти, зарезервированный для процесса ввода. Большой процент атак составляют атаки с переполнением буфера, направленные на перезапись информации в смежных ячейках памяти во фрейме стека.

Существует два типа атак с переполнением буфера:

1. Атаки на стек: объекты стековой памяти используются для хранения введенных пользователем данных. Это наиболее распространенный тип атак.

2. Атаки на динамическую память: заполняется память, зарезервированная для программы. Это довольно редкий вид атак.

Ниже приводятся этапы атаки на стек с переполнением буфера.

При написании программы для данных резервируется определенный объем памяти. При записи данных, объем которых превышает зарезервированное в стеке пространство, происходит переполнение стека. Это вызывает проблему, только если введенные данные являются вредоносными.

Программа ожидает ввода данных пользователем. Если злоумышленник вводит исполняемую команду, размер которой превышает размер стека, эта команда сохраняется за пределами зарезервированного пространства.

Полезная нагрузка средства использования уязвимости, также именуемая шелл-кодом, выполняет вредоносные действия в системе. Эти действия могут включать в себя добавление новых пользователей, изменение прав пользователей, создание или изменение файлов в системе или загрузку и запуск вредоносных программ.

Сначала в результате переполнения буфера происходит сбой программы. Если злоумышленник предоставил обратный адрес блока памяти, указывающий на вредоносную полезную нагрузку, то программа пытается выполнить восстановление, используя обратный адрес. Если обратный адрес является допустимым, вредоносная полезная нагрузка выполняется.

Полезная нагрузка теперь запускается с теми же разрешениями, что и приложение, в отношении которого совершена атака. Поскольку программы обычно выполняются в режиме ядра или с разрешениями, унаследованными от учетной записи службы, теперь злоумышленник может получить полный контроль над операционной системой.

3 ХОД РАБОТЫ

Уязвимая версия программы, написанная на C++, содержит буфер фиксированного размера 16 байт и переменную isAdmin, инициализированную нулём. Использование функции cin.getline(buffer, 255) при размере буфера 16 байт создаёт условие для переполнения: ввод более 16 символов приводит к перезаписи соседних областей стека, включая переменную isAdmin, что позволяет злоумышленнику получить несанкционированный доступ. На рисунке 3.1 изображена попытка атаки на незащищенную версию программы, на рисунке 3.2 – на защищенную.

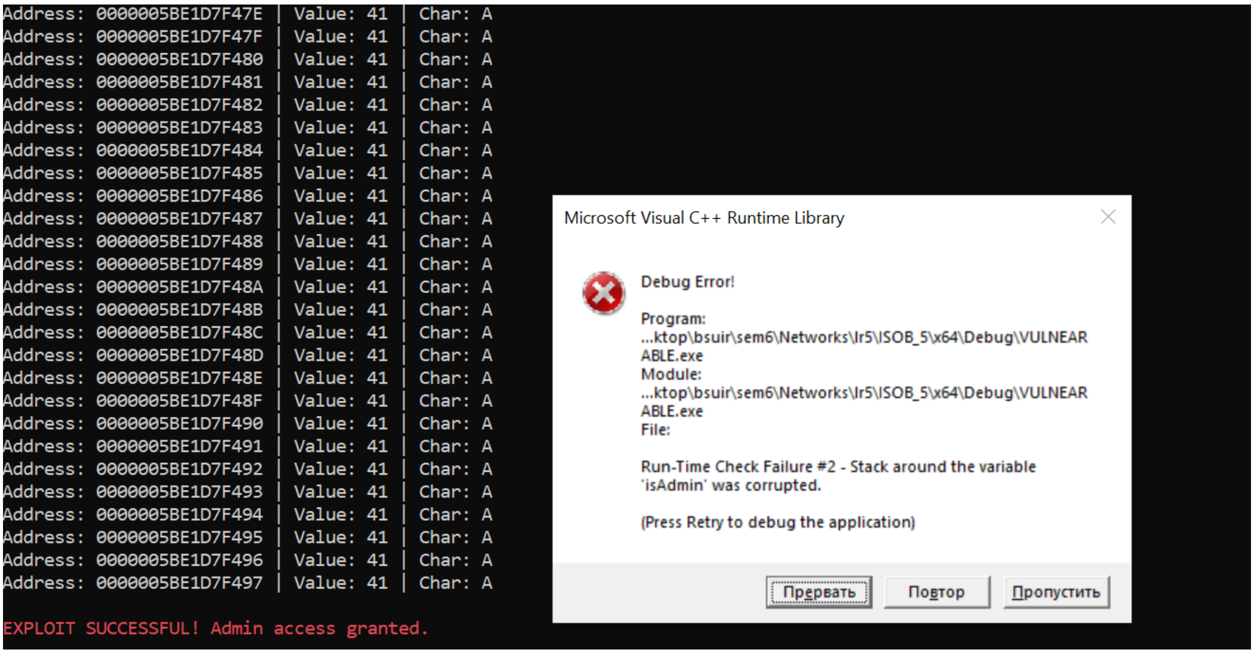

Рисунок 3.1 – Попытка атаки на незащищенную версию

В защищенной версии программы функция cin.getline(buffer, sizeof(buffer)) жёстко ограничивает длину ввода размером буфера, предотвращая переполнение на этапе обработки данных.

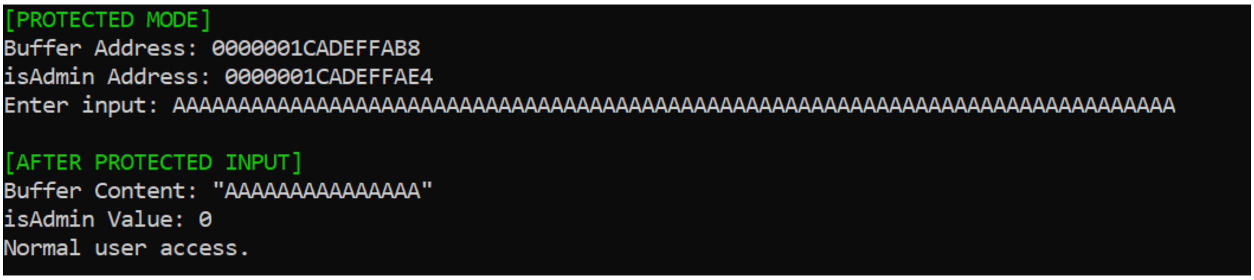

Рисунок 3.2 – Попытка атаки на защищенную версию

Таким образом, защищенная программа успешно справляется с поставленными задачами.

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы была достигнута поставленная цель – исследование методов защиты от атак на переполнение буфера и реализация демонстрационного решения, иллюстрирующего как эксплуатацию уязвимостей, так и механизмы их нейтрализации. Разработанные версии программы (уязвимая и защищённая) наглядно демонстрируют ключевые принципы обеспечения безопасности при обработке данных: контроль границ буфера, валидацию входных значений и защиту целостности структур памяти.

Успешная эксплуатация уязвимости в незащищённой версии, приводящая к изменению критических переменных (например, isAdmin), подтверждает необходимость строгого соблюдения правил безопасного программирования.

Реализация защищённого режима показала, что ограничение длины ввода, сочетаемое с многоуровневой проверкой целостности данных, эффективно блокирует попытки переполнения буфера, предотвращая несанкционированный доступ.

Таким образом, в результате проделанной работы были получены практические навыки реализации и анализа механизмов защиты от атаки на переполнение буфера, а также создана программа, которая может быть использована для демонстрации атаки на переполнение буфера.

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг программного кода

```
#include <iostream>
#include <cstring>
#include <iomanip>
using namespace std;
void printMemory(const char* buffer, int size, const char* label) {
    cout << "\n=== Memory Dump (" << label << ") ===\n";
    for (int i = 0; i < size; i++) {
        cout << "Address: " << (void*)&buffer[i]
            << " | Value: " << hex << (int)buffer[i]
            << dec << " | Char: " << buffer[i] << endl;
    }
}
void demo() {
    char buffer[16];
    int isAdmin = 0;
    memset(buffer, 0, sizeof(buffer));
    cout << "\033[1;31m" << "[VULNERABLE MODE]" << "\033[0m\n";
    cout << "Buffer Address: " << (void*)buffer << endl;
    cout << "isAdmin Address: " << (void*)&isAdmin << endl;
    cout << "Initial isAdmin: " << isAdmin << "\n\n";
    cout << "Enter input: ";
    cin.getline(buffer, 255);
    cout << "\n\033[1;31m" << "[AFTER OVERFLOW]" << "\033[0m\n";
    cout << "Buffer Content: \"" << buffer << "\"\n";
    cout << "isAdmin Value: " << isAdmin << endl;
    printMemory(buffer, 32, "Buffer + Overflow");
    if (isAdmin) {
        cout << "\n\033[1;31m" << "EXPLOIT SUCCESSFUL! Admin access
granted.\033[0m\n";
    }
    else {
        cout << "\nNormal user access.\n";
    }
}
int main() {
    demo();
    return 0;
}
void demo() {
    char buffer[16];
    int isAdmin = 0;
    cout << "\033[1;32m" << "[PROTECTED MODE]" << "\033[0m\n";
    cout << "Buffer Address: " << (void*)buffer << endl;
    cout << "isAdmin Address: " << (void*)&isAdmin << endl;
    cout << "Enter input: ";
    cin.getline(buffer, sizeof(buffer));
    cout << "\n\033[1;32m" << "[AFTER PROTECTED INPUT]" << "\033[0m\n";
    cout << "Buffer Content: \"" << buffer << "\"\n";
    cout << "isAdmin Value: " << isAdmin << endl;
    if (isAdmin) {
        cout << "Admin access granted.\n";
    }
    else {
        cout << "Normal user access.\n";
    }
}
```