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 НА ПЕРЕПОЛНЕНИЕ БУФЕРА

Выполнил: студент гр.253505 Сенько Н. С.

Проверил: ассистент кафедры информатики
Герчик А.В.

Минск 2025

СОДЕРЖАНИЕ

1 ЦЕЛЬ РАБОТЫ

Цель данного исследования заключается в изучении природы и механизмов возникновения уязвимости, связанной с переполнением буфера, а также в разработке программы, наглядно демонстрирующей как уязвимый, так и защищённый подходы к обработке пользовательского ввода.

Переполнение буфера представляет собой одну из наиболее распространённых и опасных уязвимостей, позволяющую злоумышленникам изменять память программы, вызывать сбои в её работе или даже выполнять произвольный код.

В ходе исследования были созданы две версии программы: уязвимая и защищённая. Уязвимая версия не проверяет длину входной строки перед записью данных в буфер, что может привести к выходу за пределы выделенной памяти и созданию условий для эксплуатации уязвимости.

Защищённая версия, напротив, ограничивает длину вводимой строки размером буфера, обрезая лишние символы и выдавая предупреждение пользователю, если длина строки превышает допустимый размер.

Такая реализация наглядно демонстрирует, как простые, но критически важные защитные меры могут предотвратить потенциальные атаки. Проверка длины входных данных и управление памятью являются ключевыми элементами безопасной разработки, минимизирующими риск эксплуатации уязвимостей и обеспечивающими стабильную работу программы.

Исследование подчёркивает важность внимательной обработки пользовательского ввода и демонстрирует, как даже базовые защитные механизмы могут существенно повысить уровень безопасности программного обеспечения.

2 ХОД РАБОТЫ

Ход работы начинается с запуска программы, которая включает две версии функции обработки данных: уязвимую и защищенную. Для наглядной демонстрации программа тестируется на двух вариантах ввода: строке, длина которой меньше или равна размеру буфера, и строке, превышающей его размер.

В первом случае вводится строка, длина которой не превышает размер буфера (например, "*some text*"). Защищенная версия программы корректно записывает строку в буфер и выводит сообщение о том, что данные успешно сохранены. Уязвимая версия также работает без ошибок, поскольку длина строки укладывается в пределы буфера.

Затем вводится строка, длина которой превышает размер буфера (например, "*some text, some text*"). В защищенной версии программа определяет, что длина ввода превышает допустимый лимит, выводит предупреждение и обрезает строку до размера буфера, чтобы избежать выхода за границы памяти. В уязвимой версии программа пытается записать всю строку в буфер, игнорируя его размер, что может привести к непредсказуемому поведению, включая перезапись соседних ячеек памяти или аварийное завершение работы.

Такой эксперимент позволяет на практике увидеть разницу между уязвимой и защищенной реализациями, наглядно продемонстрировать последствия переполнения буфера и подчеркнуть важность проверки длины входных данных для обеспечения безопасности программ.

ЗАКЛЮЧЕНИЕ

В рамках лабораторной работы были исследованы механизмы возникновения уязвимостей, связанных с переполнением буфера. Были созданы две версии программы: уязвимая и защищённая.

В процессе экспериментов было обнаружено, что при вводе строки, длина которой превышает размер буфера, уязвимая программа выходит за границы выделенной памяти. Это может привести к сбоям в работе программы или создать условия для эксплуатации уязвимости.

Защищённая версия программы успешно предотвращает переполнение буфера. Она проверяет длину входных данных и обрезает лишние символы, что гарантирует, что запись в буфер не выйдет за его пределы.

Эксперименты показали, что даже простые меры, такие как проверка длины ввода, могут значительно повысить безопасность программного кода.

В результате работы стало очевидно, что тщательное управление памятью, особенно при работе с пользовательскими данными, является критически важным аспектом разработки программного обеспечения. Применение базовых защитных механизмов позволяет минимизировать риск уязвимостей и повысить надёжность программных решений. Это является неотъемлемой частью безопасной разработки.