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.253504 Лавренова А.С.

Проверил: ассистент кафедры информатики Герчик А.В.

Минск 2025

СОДЕРЖАНИЕ

Введение3

[1  [Ц](#_agswffz6g6ei)ель работы](#_Toc178067889) 4

[2  Теоретические сведения 5](#_Toc178067890)

[3  Описание и пример выполнения программы 7](#_Toc178067891)

Заключение8

[Список использованных источников 8](#_Toc178067900)

[Приложение А (обязательное) 10](#_Toc178067901)

# ВВЕДЕНИЕ

Переполнение буфера — это одна из наиболее распространенных уязвимостей в программировании, которая может привести к различным последствиям, включая нарушение работы программы, утечку данных или даже выполнение произвольного кода. Эта уязвимость появляется, когда программа пытается записать больше данных в буфер, чем он может вместить. В результате могут быть повреждены соседние участки памяти, что открывает возможности для злоумышленников использовать эту ошибку в своих целях.

Защита от переполнения буфера является важной задачей в процессе разработки безопасного программного обеспечения. Это особенно актуально в контексте разработки низкоуровневых программ, таких как операционные системы, драйвера, сетевые сервисы и другие приложения, где манипуляции с памятью и производительность имеют первостепенное значение. В современных языках программирования, таких как C и C++, использование небезопасных функций для обработки строк и данных (например, strcpy и strncpy) является одной из главных причин возникновения переполнений буфера.

Данная лабораторная работа направлена на изучение механизма защиты от переполнения буфера, а также на демонстрацию способов предотвращения этой уязвимости в процессе разработки программного обеспечения. В рамках работы будут рассмотрены основные принципы безопасной работы с буферами и строками в языке C++, а также будут использованы безопасные аналоги стандартных функций, таких как strcpy\_s и strncpy\_s. Также в ходе лабораторной работы будет подробно рассмотрен процесс контроля длины строк и предотвращения переполнения буфера при копировании данных.

Переполнение буфера является не только теоретической угрозой, но и реальной проблемой, с которой сталкиваются многие разработчики в своей повседневной практике. Разработка методов защиты от переполнения буфера — это ключевой аспект обеспечения безопасности программных систем. В ходе лабораторной работы будет предложена реализация защиты от переполнения буфера с подробным выводом информации в консоль, что позволит наглядно продемонстрировать, как можно эффективно и безопасно управлять буферами и избегать возможных ошибок, связанных с их переполнением.

Кроме того, внимание будет уделено принципам безопасного копирования данных и проверки их размера перед записью в ограниченные буферы. Все это будет сопровождаться примерами, которые покажут, как можно предотвратить переполнение буфера в реальных приложениях. Учитывая важность этой темы для безопасной разработки, лабораторная работа позволит не только познакомиться с механизмами защиты, но и даст понимание, как эффективно применять эти техники на практике для разработки защищенных программ.

1 ЦЕЛЬ РАБОТЫ

Цель данной лабораторной работы заключается в том, чтобы изучить и продемонстрировать методы защиты программного обеспечения от уязвимости переполнения буфера, а также освоить эффективные практики безопасной работы с буферами и строками в языке программирования C++. Переполнение буфера — это одна из самых распространенных и опасных уязвимостей, которая может привести к серьезным последствиям, таким как сбои в работе программы или даже исполнение произвольного кода. В рамках лабораторной работы будет проведен подробный анализ механизмов переполнения буфера и рассмотрены способы защиты от этой угрозы.

В ходе выполнения лабораторной работы будет изучен процесс безопасного копирования данных в буфер с ограниченным размером, использование безопасных функций, таких как strncpy\_s и strcpy\_s, вместо устаревших и небезопасных функций, таких как strncpy и strcpy. Особое внимание будет уделено правильному контролю длины строк и данных перед их копированием в буфер. Важно отметить, что каждая из безопасных функций предоставляет дополнительные параметры, которые позволяют избежать переполнения буфера, исключив тем самым возникновение ошибок, связанных с манипуляцией памятью.

Кроме того, лабораторная работа направлена на демонстрацию того, как правильно обрабатывать ситуации, когда данные превышают размер буфера, и как корректно их обрезать. Также будет наглядно показано, как можно выводить подробную информацию о процессе защиты и предотвращения переполнения в консоль, чтобы разработчики могли не только применить защиту на практике, но и увидеть, как работает механизм безопасности в реальном времени. Важным аспектом будет являться способность адаптировать этот механизм для различных типов приложений, где безопасность и управление памятью имеют решающее значение.

Таким образом, цель лабораторной работы — не только овладение инструментами защиты от переполнения буфера, но и получение глубоких знаний о безопасной работе с памятью и строками в низкоуровневых языках программирования, таких как C++. Это знание критически важно для создания надежных, защищенных и устойчивых к атакам программных систем, а также для формирования практических навыков безопасной разработки, которые будут полезны в реальных условиях работы над проектами.

2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Проблема переполнения буфера давно известна в области защиты информации. Переполнение буфера возникает, когда компьютерная программа записывает данные поверх других данных в буфере.

Червь Морриса - первый самораспространяющийся интернет-червь 1988 года - использовал именно переполнение буфера для распространения между компьютерами.

Даже спустя почти тридцать лет проблема переполнения буфера остается актуальной: при анализе атак и обнаруженных уязвимостей выяснилось, что данная проблема является самой распространенной.

Переполнение буфера - один из наиболее распространенных способов, с помощью которого взламывают компьютерные системы. Причина данного явления в том, что большинство высокоуровневых языков программирования (C++, C#) используют стековую технологию (ее суть заключается в том, что данные размещаются в стеке процесса), происходит смешение данных программы и управляющих данных между собой (в том числе адреса начала стекового кадра и адреса возврата из исполняемой функции). [1]

Переполнение буфера может привести к аварийному завершению или зависанию программы, ведущим к отказу обслуживания. С помощью некоторых видов переполнений (например, переполнение в стеке) злоумышленник получает возможность загружать и выполнять произвольный код от имени программы и с правами учетной записи, от которой она выполняется.

Способы защиты от переполнения буфера:

1. Корректировка кода программы для устранения уязвимостей. Чаще всего переполнение буфера происходит по причине того, что в программе не предусмотрены проверки выхода за границу буфера. Отдельное место в этом вопросе занимает язык С++. Он не содержит средств контроля соответствия типов, в связи с чем в переменные одного типа можно занести значения других типов. Также в С++ используются функции (sprint, strcpy), которые не проверяют длину переменных, что также приводит к переполнению. Существуют следующие способы, которые позволят минимизировать количество уязвимостей: замена уязвимых функций sprint, strcpy на аналогичные функции, которые проверяют длину строки, - snprint, strncpy и использование средств, имитирующих переполнение буфера при отладке программы. Это поможет выявить слабые места и устранить их.
2. Использование неисполнимого буфера. Данный метод заключается в том, что в стеке и сегментах данных запрещается исполнение кода, происходит только запись и чтение. С помощью настоящего метода можно защититься только от атак с внедрением кода. При остальных видах атак он бесполезен.
3. Проверка выхода за границы. При данном методе при каждом обращении к переменной происходит проверка ее длины (превышения допустимых границ). Настоящий метод реализован в ряде компиляторов (Compaq C, Alpha Linux). Известно дополнение для gcc, в котором реализована проверка выхода за границы. Данный метод нейтрализует все атаки, которые связаны с переполнением буфера, так как при его использовании переполнение невозможно. Но у него есть существенный минус - снижение производительности программы в несколько раз.
4. Использование различных защитных утилит, таких как PointGuard, StackGuard, StackShield и др.

Переполнение буфера - весьма актуальная проблема, которая на протяжении почти уже 30 лет создает угрозы для безопасного хранения данных. Зачастую уязвимость к переполнению заложена в самом языке программирования.

К основным видам атак, связанным с переполнением буфера, относятся атаки на стек, атаки на функции форматирования строк, атаки на heap.

Существуют способы, которые помогают предотвратить переполнение буфера: корректировка кода программы для устранения уязвимостей, использование неисполнимого буфера, проверка выхода за границы, использование различных защитных утилит.

К основным ошибкам пользователя C++, способствующим переполнению буфера, относятся использование функций, не проверяющих длину переменных, ошибки при преобразовании данных, «ошибка на единицу». [2]

Соблюдение данных способов и предупреждение основных ошибок пользователей будет способствовать предотвращению проблемы переполнения буфера.

**3 ОПИСАНИЕ И ПРИМЕР ВЫПОЛНЕНИЯ ПРОГРАММЫ**

Программа выполняет несколько тестов с безопасным и опасным вводом, проверяя, как она справляется с возможным переполнением буфера и как предотвращает такие ситуации.

Пример работы программы будет представлен на рисунке 3.1. В процессе выполнения на экране будет выводиться информация, которая позволяет отслеживать действия программы.

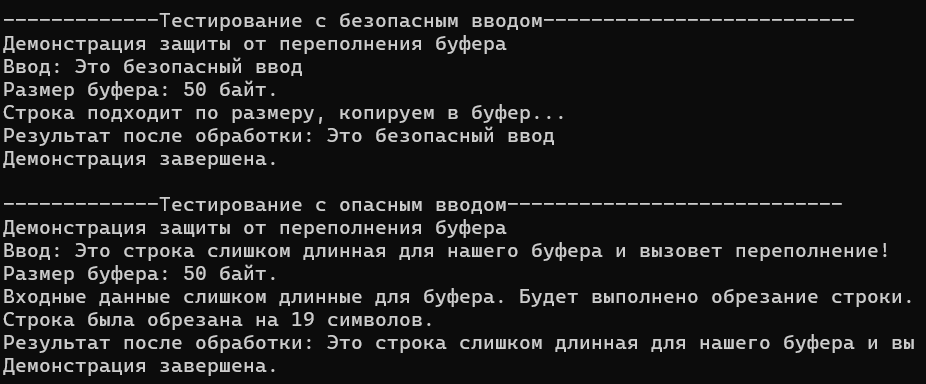


Рисунок 3.1 – Работа программы

При запуске программы сначала выполняется тест с безопасным вводом, когда длина строки не превышает размер буфера (50 символов). В этом случае программа без проблем копирует строку в буфер.

Вторым шагом выполняется тест с опасным вводом, когда длина строки превышает размер буфера. В этом случае программа обрезает строку до возможной длины и выводит информацию о том, сколько символов было обрезано. Это предотвращает переполнение буфера и возможные ошибки, связанные с этим.

В процессе работы программы все важные действия, такие как проверка длины строки, копирование данных в буфер, обрезка данных при переполнении и вывод результата, сопровождаются подробными сообщениями в консоль.

В ходе выполнения лабораторной работы была продемонстрирована важность защиты от переполнения буфера. Использование безопасных функций, таких как strncpy\_s и strcpy\_s, позволяет предотвратить возможные уязвимости, связанные с переполнением памяти. Программа успешно справляется с переполнением буфера, обрезая данные, если их длина превышает размер буфера. Также были приведены примеры работы программы с безопасными и опасными вводами, которые демонстрируют, как правильно обрабатывать данные и избегать ошибок, связанных с переполнением буфера.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения данной лабораторной работы были изучены ключевые принципы защиты программного обеспечения от переполнения буфера, а также продемонстрированы эффективные методы предотвращения данной уязвимости в языке программирования C++. Переполнение буфера является одной из наиболее опасных уязвимостей, которая может привести к серьезным сбоям в работе программы, утечке конфиденциальных данных или даже выполнению произвольного кода. Устранение этой угрозы путем правильного управления памятью и применением безопасных функций стало основным фокусом работы.

Работа с безопасными функциями, такими как strncpy\_s и strcpy\_s, продемонстрировала, как можно избежать переполнения буфера, а также как важно контролировать размер данных перед их копированием в буфер. Кроме того, была рассмотрена обработка ситуаций, когда данные превышают размер буфера, и подходы к их обрезке, что позволяет предотвратить возможные ошибки и сбои в работе программы. Вывод подробной информации в консоль о ходе работы программы позволил наглядно понять, как работает механизм защиты от переполнения и какие последствия могут возникнуть при неправильной обработке данных.

Задачи лабораторной работы были успешно выполнены, и, как результат, была разработана программа, эффективно защищающая от переполнения буфера и демонстрирующая важность использования безопасных функций для работы с памятью. Эта лабораторная работа подчеркнула значимость предотвращения переполнений буфера как неотъемлемой части безопасной разработки программного обеспечения. Полученные знания и навыки будут полезны для создания надежных и защищенных приложений, что особенно важно в современных условиях, когда безопасность является приоритетом в разработке программных систем.

Таким образом, лабораторная работа продемонстрировала не только теоретическое понимание проблемы переполнения буфера, но и практическое применение методов защиты, которые могут быть использованы для предотвращения подобных уязвимостей в реальных программных проектах.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Защита от переполнения буфера. В. А. Рычков [Электронный ресурс]. – Режим доступа: https://naupri.ru/journal/176?ysclid=m8evoojrmp683502959.

[2] [4 типовые ловушки в работе со строками в С и С++ и как их избежать / Хабр](https://habr.com/ru/companies/ncloudtech/articles/830918/) [Электронный ресурс]. – Режим доступа: https://habr.com/ru/companies/ ncloudtech/articles/830918/.

ПРИЛОЖЕНИЕ А

(обязательное)

Исходный код

#include <iostream>

#include <cstring>

#include <cstdlib>

void check\_buffer\_overflow(char\* input) {

const size\_t buffer\_size = 50;

char buffer[buffer\_size];

std::cout << "Размер буфера: " << buffer\_size << " байт." << std::endl;

// переполнения буфера

size\_t input\_length = strlen(input);

if (input\_length >= buffer\_size) {

std::cout << "Входные данные слишком длинные для буфера. Будет выполнено обрезание строки." << std::endl;

strncpy\_s(buffer, buffer\_size, input, buffer\_size - 1);

buffer[buffer\_size - 1] = '\0'; // завершение строки

size\_t truncated\_length = input\_length - (buffer\_size - 1);

std::cout << "Строка была обрезана на " << truncated\_length << " символов." << std::endl;

}

else {

std::cout << "Строка подходит по размеру, копируем в буфер..." << std::endl;

strcpy\_s(buffer, buffer\_size, input);

}

std::cout << "Результат после обработки: " << buffer << std::endl;

}

void demonstrate\_buffer\_overflow(const char\* input) {

std::cout << "Демонстрация защиты от переполнения буфера" << std::endl;

std::cout << "Ввод: " << input << std::endl;

// защищата от переполнения

check\_buffer\_overflow(const\_cast<char\*>(input));

std::cout << "Демонстрация завершена." << std::endl;

}

int main() {

setlocale(LC\_ALL, "Russian");

const char\* safe\_input = "Это безопасный ввод";

std::cout << "-------------Тестирование с безопасным вводом--------------------------" << std::endl;

demonstrate\_buffer\_overflow(safe\_input);

const char\* unsafe\_input = "Это строка слишком длинная для нашего буфера и вызовет переполнение!";

std::cout << "\n-------------Тестирование с опасным вводом----------------------------" << std::endl;

demonstrate\_buffer\_overflow(unsafe\_input);

return 0;

}