

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра информационной безопасности

КУРСОВАЯ РАБОТА
по дисциплине «Программирование»
Тема: Приложение «Сигнатурный сканер»

Студент гр. 3364

Юсфи А.

Преподаватель

Халиуллин Р.А.

Санкт-Петербург

2024

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент Юсфи А.

Группа 3364

Тема работы: Приложение «Сигнатурный сканер»

Исходные данные:

Разработать на языке программирования С или С++ (по выбору студента) сигнатурный сканер. Сигнатурный сканер должен детектировать заданный образец по наличию в сканируемом файле определенной последовательности байтов (сигнатуры) по определенному смещению. Размер сигнатуры должен быть не менее, чем 6 (шесть) байт и не более, чем 8 (восемь) байт, по выбору студента. Поиск сигнатуры необходимо выполнять только в исполняемых файлах, определение формата файла необходимо выполнять по первым байтам файла. Если сигнатура обнаружена в файле, то необходимо вывести имя файла с указанием имени образца, которому соответствует найденная сигнатура. Сигнатура, смещение сигнатуры и название образца должны храниться в отдельном файле и считываться при запуске сигнатурного сканера, путь к этому файлу может вводиться пользователем. Путь к файлу для сигнатурного сканирования должен вводиться пользователем. Приложение должно иметь консольный или графический интерфейс (по выбору студента). В интерфейсе приложения допускается использовать буквы латинского алфавита для транслитерации букв алфавита русского языка. Интерфейс приложения должен быть интуитивно понятным и содержать подсказки для пользователя. В исходном коде приложения должны быть реализованы функции. В исходном коде приложения должны быть реализованы проверки аргументов реализованных функций и проверки возвращаемых функциями значений (для всех функций — как сторонних, так и реализованных). Приложение должно

корректно обрабатывать ошибки, в том числе ошибки ввода/вывода, выделения/освобождения памяти и т. д.

Содержание пояснительной записки:

Введение, теоретическая часть, реализация программы, результат тестирования программы, заключение, список используемых источников, приложение 1 – руководство пользователь, приложение 2 – блок-схема алгоритма, приложение 3 – исходный код программы.

Предполагаемый объем пояснительной записки:

Не менее 30 страниц.

Дата выдачи задания: 31.03.2024

Дата сдачи реферата: 03.06.2024

Дата защиты реферата: 08.06.2024

Студент гр. 3364

Юсфи А.

Преподаватель

Халиуллин Р.А.

АННОТАЦИЯ

Курсовая работа заключается в проверке целостности исполняемого файла путем сравнения сигнатур. Пользователи вводят пути к исполняемому и файлу с сигнатурой. Программа считывает сигнатуру и смещение из файла, убеждаясь в наличии заголовка MZ в исполняемом файле. Она вычисляет размеры для проверки и проверяет целостность файла. После чтения сигнатуры из исполняемого файла она сравнивает ее с предоставленной, чтобы убедиться в совпадении. Наконец, она уведомляет пользователей, безопасен ли файл или нет.

SUMMARY

The course work involves validating the integrity of an executable file through signature comparison. Users input paths for the executable and signature files. The program reads the signature and offset from the file, ensuring an MZ header in the executable. It calculates sizes for verification and checks file integrity. After reading the signature from the executable, it compares it with the provided one to ensure a match. Finally, it notifies users whether the file is safe or not.

СОДЕРЖАНИЕ

Введение	6
1. Теоретическая часть	7
1.1. История антивирусов	7
1.2. Эволюция антивирусов	8
1.3. Прогноз и возможности перспектив развития	8
1.4. Роль антивирусов в цифровой эпохе	8
2. Реализация программы	10
2.1. Сведения о выбранном языке программирования	10
2.2. Сведения о программном обеспечении	10
2.3. Сведения о реализованных функциях	10
3. Результаты тестирования программы	17
3.1. Сведения о результатах тестирования программы	17
Заключение	21
Список использованных источников	22
Приложение 1. Руководство пользователя	23
Приложение 2. Блок-схема алгоритма	25
Приложение 3. Исходный код программы	26

ВВЕДЕНИЕ

Цель курсовой работы – заключается в разработке приложения для сканирования подписей с использованием языка программирования С и создании отдельных функций для каждой задачи. В начале процесса создаётся блок-схема для визуального представления функционала программы. Интуитивный интерфейс приложения играет важную роль в улучшении взаимодействия с пользователем, поэтому разрабатывается "руководство пользователя". Значимость данной программы заключается в скорости её выполнения и эффективности языка С, что делает его подходящим выбором для разработки сканера подписей, так как программы на языке С могут легко портироваться на различные платформы и использоваться.

1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1. История антивирусов

История программированных антивирусных программ уходит корнями в ранние годы компьютерной эры, когда вместе с развитием цифровых технологий начали появляться и вредоносные программы. Одними из первых известных антивирусных решений 1980-х годов стали программы типа FluShot Plus и ThunderBYTE Antivirus. Они в основном сосредотачивались на выявлении и удалении известных вирусов с помощью методов обнаружения по сигнатуре.

С развитием компьютерной техники уровень угроз также возрос, и разработчики антивирусов вынуждены были идти в ногу с временем. В 1990-е годы стало популярным использование эвристического анализа, который позволял антивирусным программам обнаруживать новые, ранее неизвестные угрозы на основе поведенческих паттернов. В это время также были основаны крупные антивирусные компании, такие как Symantec, McAfee и Kaspersky, которые сыграли ключевую роль в формировании антивирусного ландшафта.

Начало 2000-х годов принесло значительные изменения с распространением интернета и ростом угроз, связанных с электронной почтой, такими как черви и фишинговые атаки. Антивирусное программное обеспечение адаптировалось, включив в себя функции, такие как сканирование электронной почты и защита в реальном времени. Более того, появление машинного обучения и искусственного интеллекта позволило антивирусным программам улучшить возможности обнаружения, что позволяет им распознавать и противодействовать всё более сложным угрозам.

Сегодня антивирусное программное обеспечение стало неотъемлемым инструментом для обеспечения безопасности цифровых систем на различных платформах, включая настольные компьютеры, серверы и мобильные устройства. С постоянными технологическими усовершенствованиями и постоянно меняющимся ландшафтом угроз разработчики антивирусов остаются на передовой, чтобы обеспечить надёжную защиту цифровой инфраструктуры.

В 21 веке антивирусные программы на смартфонах и компьютерах стали доступнее, предлагая защиту в реальном времени и обнаружение вредоносных программ. Используя машинное обучение и передовые алгоритмы, они эффективно борются с постоянно меняющимися тактиками киберпреступников. Эти технологии продолжают развиваться, обеспечивая надежную защиту цифровых систем.

1.2. Эволюция антивирусов

Антивирусное программное обеспечение эволюционировало от обнаружения по сигнатуре к эвристическому и поведенческому анализу. Последние достижения включают интеграцию искусственного интеллекта и машинного обучения для проактивного обнаружения угроз. Платформы защиты конечных точек и облачные решения обеспечивают всестороннюю защиту от изменяющихся киберугроз.

1.3. Прогноз и возможности перспектив развития

Прогноз и возможности перспектив развития в контексте антивирусного программного обеспечения означают предвидение будущих угроз и развитие соответствующих методов защиты. Анализируя текущие тенденции в кибербезопасности и эволюцию вирусных атак, разработчики антивирусов могут предугадать новые угрозы и адаптировать свои продукты для более эффективной защиты. Это включает в себя постоянное обновление баз данных угроз, разработку интеллектуальных алгоритмов обнаружения и реагирования на новые вирусы и вредоносные программы.

1.4. Роль антивирусов в цифровой эпохе

Роль в цифровой эпохе антивирусные программы обеспечивают безопасность компьютерных систем и данных. Они обнаруживают и блокируют вирусы, предотвращают атаки злоумышленников, мониторят активность системы в реальном времени и анализируют поведение программ. Постоянные

обновления баз данных и алгоритмов защиты помогают бороться с новыми угрозами. Антивирусные программы также защищают личную информацию пользователей от кражи и злоупотреблений. Все это делает их неотъемлемой частью цифровой безопасности.

2. РЕАЛИЗАЦИЯ ПРОГРАММЫ

2.1. Сведения о выбранном языке программирования

Язык программирования C – объединяет в себе характеристики языков высокого уровня и производительность ассемблера, обеспечивая универсальную компилируемую среду со статической типизацией. Это позволяет эффективно взаимодействовать с аппаратной частью компьютера и создавать сложные конструкции на высоком уровне, а также обеспечивает переносимость исходного кода программы на широкий спектр платформ.

2.2. Сведения о программном обеспечении

Операционная система Windows 11 Pro, версия 22H2, 64-разрядная, среды разработки – Clion и CodeBlocks, компилятор – GNU GCC Compiler.

2.3. Сведения о реализованных структуре и функциях

Реализованные структуре: `Virus{}`.

Структура `Virus`.

Структура `Virus` содержит следующие поля: имя вируса, смещение для подписи вируса в файле, ожидаемая подпись вируса в шестнадцатеричном формате, фактическая подпись вируса, считанная из исполняемого файла, и первые два байта исполняемого файла для проверки подписи "MZ". Эта структура используется для хранения и сравнения данных вируса, чтобы определить, заражен ли файл.

Объявление структуры:

```
Virus{} vir;
```

Поля структуры:

- `NameVirus[]` — имя вируса.
- `OffsetPosition` — смещение для подписи вируса в файле.
- `sign[]` — ожидаемая подпись вируса в шестнадцатеричном формате.

- `VirusSignature[]` — фактическая подпись вируса, считанная из исполняемого файла.
- `MZ[]` — первые два байта исполняемого файла.
- `MZexpected[]` — ожидаемая сигнатура "MZ" для исполняемого файла.

Реализованные функции: `DataRead ()`, `SeekBegan ()`, `SeekEnd ()`, `GoToSignatureOffset ()`, `ScanForMZSignature ()`, `CloseFile ()`, `main()`.

Функция `main`.

Функция `main` запрашивает пути к файлу с подписями и исполняемому файлу, считывает данные, проверяет наличие вирусной подписи и выводит сообщение о безопасности файла

Исходный код функции находится в файле `main.c`.

Объявление функции:

```
int main();
```

Тип функции: `int`.

Аргументы функции:

- Аргументы отсутствуют.

Возвращаемое функцией значение:

- 0 — выполнение функции прошло без ошибок;
- 1 — ошибка функции `printf()`;
- 1 — ошибка функции `SeekBegan ()`;
- 1 — значение первого аргумента из какой-либо внешней реализованной функции равно 0;
- 2 — ошибка функции `printf()`;
- 2 — ошибка функции `ScanForMZSignature()`;
- 2 — ошибка при открытии файла;
- 3 — ошибка закрытия файла;
- 3 — ошибка функции `fgetc()`;
- 3 — ошибка функции `DataRead()`;
- 3 — ошибка функции `printf()`;

- 4 — ошибка функции printf();
- 4 — ошибка функции SeekEnd();
- 5 — ошибка функции print();
- 5 — ошибка функции ftell();
- 6 — ошибка функции print();
- 7 — ошибка функции GoToSignatureOffset();
- 8 — ошибка функции fread();
- 9 — ошибка функции print();
- 10 — ошибка функции print().

Функция DataRead.

Функция DataRead считывания вирусных данных из текстового файла.

Исходный код функции Somme находится в файле main.c.

Объявление функции:

```
int DataRead(char *textfilepath);
```

Тип функции: int.

Аргументы функции:

- textfilepath — переменная, принимает указатель на путь к файлу с данными о вирусе., тип аргумента: char;

Возвращаемое функцией значение:

- 0 — выполнение функции прошло без ошибок;
- 1 — значение первого аргумента функции некорректно.
- 2 — ошибка открытия файла с данными о вирусе.
- 3 — ошибка чтения данных о вирусе из файла.
- 4 — ошибка закрытия файла.

Функция SeekBegan.

Функция SeekBegan определяет перемещение указателя на начало файла.

Исходный код функции SeekBegan находится в файле main.c.

Объявление функции:

```
int SeekBegan(FILE *f)
```

Тип функции: int.

Аргументы функции:

- f — переменная, принимает указатель на файл., тип аргумента: file;

Возвращаемое функцией значение:

- 0 — выполнение функции прошло без ошибок;
- 1 — значение первого аргумента функции некорректно.
- 2 — ошибка функции fseek();

Функция SeekEnd.

Функция SeekEnd определяет перемещение указателя в конец файла.

Исходный код функции SeekEnd находится в файле main.c.

Объявление функции:

```
int SeekEnd(FILE *f);
```

Тип функции: int.

Аргументы функции:

- f — переменная, принимает указатель на файл., тип аргумента: file;

Возвращаемое функцией значение:

- 0 — выполнение функции прошло без ошибок;
- 1 — значение первого аргумента функции некорректно.
- 2 — ошибка функции fseek();

Функция GoToSignatureOffset.

Функция GoToSignatureOffset определяет перемещение указателя положение сигнатуры вируса.

Исходный код функции GoToSignatureOffset находится в файле main.c.

Объявление функции:

```
int GoToSignatureOffset(FILE *f);
```

Тип функции: int.

Аргументы функции:

- `f` — переменная, принимает указатель на файл., тип аргумента: `file`;

Возвращаемое функцией значение:

- 0 — выполнение функции прошло без ошибок;
- 1 — значение первого аргумента функции некорректно.
- 2 — ошибка функции `fclose()`;

Функция `ScanForMZSignature`.

Функция `ScanForMZSignature` определяет чтение первых двух байт из файла.

Исходный код функции `ScanForMZSignature` находится в файле `main.c`.

Объявление функции:

```
int ScanForMZSignature(FILE *f);
```

Тип функции: `int`.

Аргументы функции:

- `f` — переменная, принимает указатель на файл., тип аргумента: `file`;

Возвращаемое функцией значение:

- 0 — выполнение функции прошло без ошибок;
- 1 — значение первого аргумента функции некорректно.
- 2 — ошибка функции `fread ()`;

Функция `CloseFile`.

Функция `CloseFile` определяет закрытие файла.

Исходный код функции `CloseFile` находится в файле `main.c`.

Объявление функции:

```
int CloseFile (FILE *f);
```

Тип функции: `int`.

Аргументы функции:

- `f` — переменная, принимает указатель на файл., тип аргумента: `file`;

Возвращаемое функцией значение:

- 0 — выполнение функции прошло без ошибок;
- 1 — значение первого аргумента функции некорректно.
- 2 — ошибка функции `fclose()`;

3. РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ ПРОГРАММЫ

3.1. Сведения о результатах тестирования программы

При запуске программы пользователь видит приветственное сообщение и инструкции по использованию сканера подписей. После этого программа попросит вас ввести первый путь к текстовому файлу (рисунок 1).

```
    ** Hello,Welcome to the signature scanner **
        *** Guide of use ***
    * Prepare the text file as this examples *
    The first line : Name of Program.
    Second line:offset in Decimal
    Third line:Signature in HexDecimal and in 6 bytes with Spaces.

*****Examples of text file *****
Notepad
242835
8A 67 25 CA 82 5F

Enter the path of the text file:
```

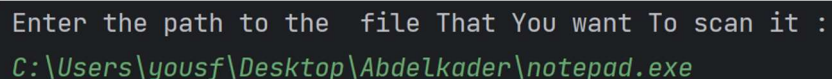
Рисунок 1 — Главный экран запуска программы

Приложение запрашивает у пользователя ввод первого пути к текстовому файлу, содержащему информацию, необходимую для сканирования (рисунок 2).

```
Enter the path of the text file:
C:\Users\yousf\Desktop\Abdelkader\new.txt
```

Рисунок 2 — Ввод пути к текстовому файлу

Приложение просит пользователя ввести второй путь к исполняемому файлу, который необходимо отсканировать значение (рисунок 3).



```
Enter the path to the file That You want To scan it :  
C:\Users\yousf\Desktop\Abdelkader\notepad.exe
```

Рисунок 3 — Ввод пути к исполняемому файлу

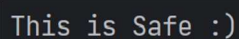
Когда вы вводите второй путь к исполняемому файлу и нажимаете Enter, программа начинает работать и печатать, программа считается вирусной, если подпись совпадает (рисунок 4).



```
This is a Virus !  
Virus name: Notepad  
Location file: C:\Users\yousf\Desktop\Abdelkader\notepad.exe
```

Рисунок 4 — Вывод результата сканирования

Когда вы вводите второй путь к исполняемому файлу и нажимаете Enter, программа начинает работать и печатать, программа безопасна, если подписи совпадают (рисунок 5).



```
This is Safe :)
```

Рисунок 5 — Вывод результата сканирования

После ввода второго пути к сканируемому файлу, если файл не является исполняемым, что означает, что это не тот файл, который необходимо сканировать, то выводится, что этот файл безопасен. (рисунок 7).

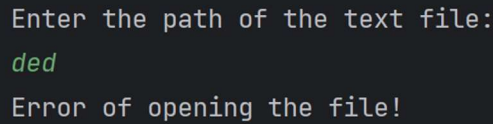


```
This is Safe program
```

Рисунок 7 — Файл не является исполняемым

Приложение запрашивает у пользователя ввод корректного пути к файлу. Если пользователь вводит некорректный путь, программа выводит сообщение об

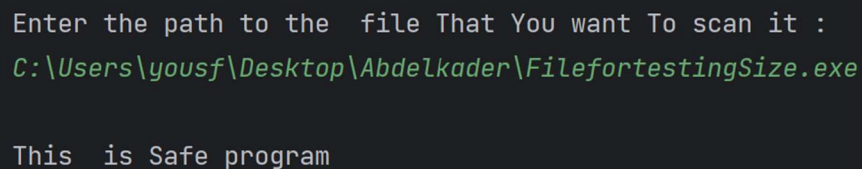
ошибке. Однако программа не автоматически повторяет процесс для исправления ошибки; вместо этого пользователю нужно повторно запустить программу для продолжения работы. (рисунок 8).



```
Enter the path of the text file:
ded
Error of opening the file!
```

Рисунок 8 — Ввод недопустимого пути

При вводе исполняемого файла, размер которого меньше размера подписи и смещения, то выводится, что этот файл безопасен. (рисунок 9).

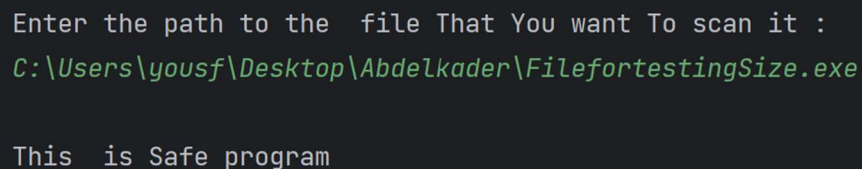


```
Enter the path to the file That You want To scan it :
C:\Users\yousf\Desktop\Abdelkader\FilefortestingSize.exe

This is Safe program
```

Рисунок 9 — Файл с размером, меньшим подписи и смещения

После ввода путей и размера подписи, равного или меньшего нулю, программа завершает работу и что этот файл безопасен. (рисунок 10).



```
Enter the path to the file That You want To scan it :
C:\Users\yousf\Desktop\Abdelkader\FilefortestingSize.exe

This is Safe program
```

Рисунок 10 — Размер подписи не превышает нуля

Вид законченной программы (рисунок 11).

```

    ** Hello,Welcome to the signature scanner **
        *** Guide of use ***
* Prepare the text file as this examples *
The first line : Name of Program.
Second line:offset in Decimal
Third line:Signature in HexDecimal and in 6 bytes with Spaces.

*****Examples of text file *****
Notepad
242835
8A 67 25 CA 82 5F

Enter the path of the text file:
C:\Users\yousf\Desktop\Abdelkader\new.txt
Enter the path to the file That You want To scan it :
C:\Users\yousf\Desktop\Abdelkader\notepad.exe

This is a Virus !
Virus name: Notepad
Location file: C:\Users\yousf\Desktop\Abdelkader\notepad.exe
```

Рисунок 11 — Окно успешно выполненной программы

ЗАКЛЮЧЕНИЕ

В результате выполнения работы была достигнута основная цель — предоставить эффективный инструмент, способный обеспечить проверку целостности исполняемых файлов путем сравнения подписей. Создание отдельных функций для каждой задачи и разработка интуитивного интерфейса, сопровождаемого "руководством пользователя", демонстрируют высокий уровень организации проекта. Благодаря использованию языка С, приложение обладает быстрой скоростью выполнения и высокой эффективностью. Результаты данной работы подтверждают перспективность и практичность разработки сигнатурного сканера на основе С, что является важным вкладом в область информационной безопасности.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Компьютерные вирусы // Санникова Т. С. – 2015. – URL: <https://cyberleninka.ru/article/n/kompyuternye-virusy> (дата обращения: 10.04.2024).
2. Koret, J., & Bachaalany, E. (2015). The Antivirus Hacker's Handbook (1st ed.). ISBN: 978-1-119-02875-8 (Print), ISBN: 978-1-119-02876-5 (eBook).
3. Sikorski, M., & Honig, A. (2012). Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software (1st ed.). ISBN-10: 1593272901, ISBN-13: 978-1593272906.

ПРИЛОЖЕНИЕ 1. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Программа «Сигнатурный сканер».

Эта программа запрашивает у пользователя два пути к файлам. Перед вводом путей пользователю предоставляются инструкции. Первый путь предназначен для текстового файла, содержащего три информационных элемента: сигнатуру, смещение и имя программы. Второй путь относится к исполняемому (exe) файлу, который мы собираемся просканировать и проверить, содержит ли он вирус.

Минимальные требования:

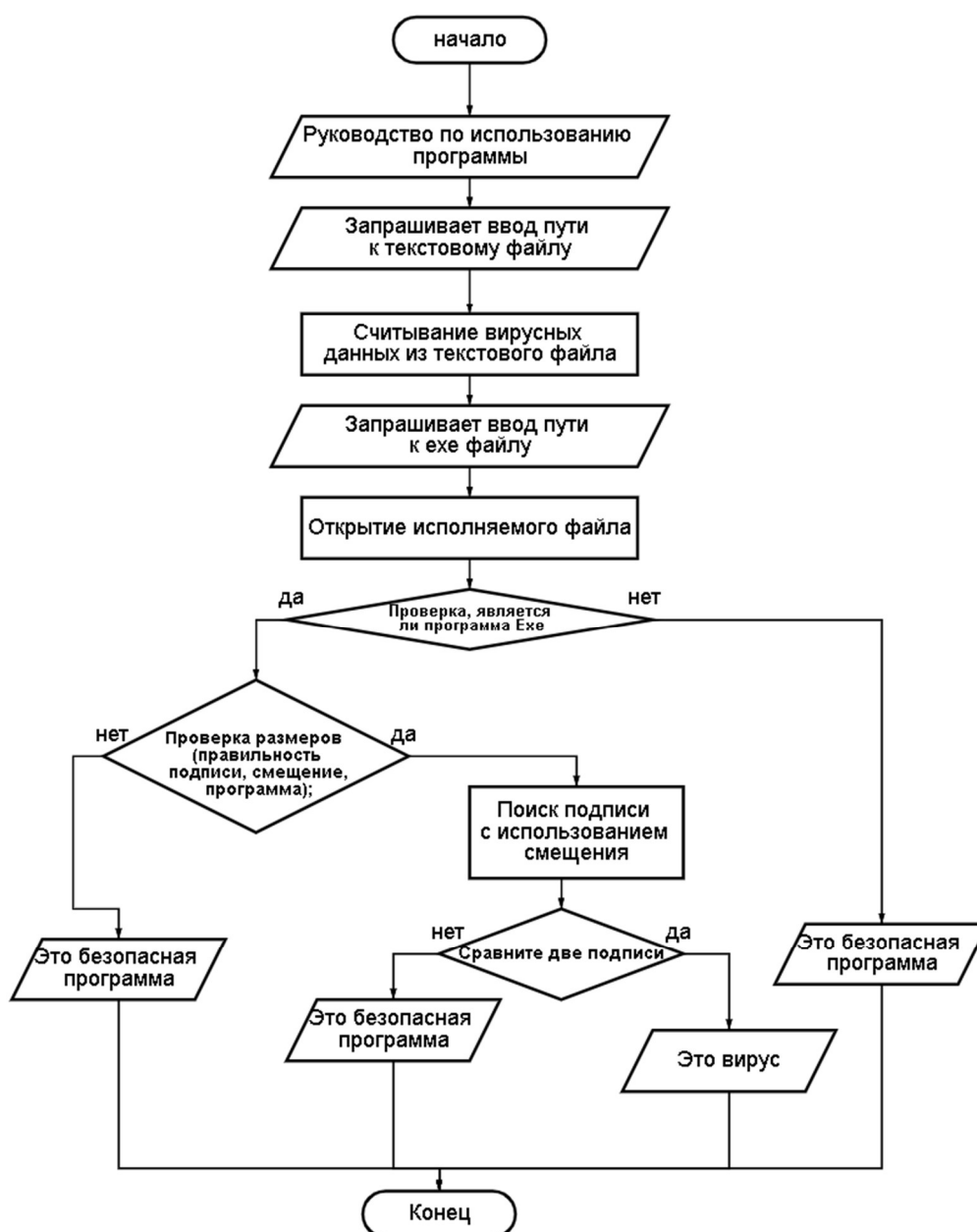
- Операционная система: Windows 7 и вся версия после;
 - Процессор: 1 (ГГц) или быстрее, с 2 или большим количеством ядер на совместимом 64-разрядном процессоре (32). Процессор на компьютере будет основным фактором, определяющим запуск Windows 7 и вся версия после;
 - ОЗУ: 2 гигабайта (ГБ);
 - 32 ГБ или большее хранилище;
 - Требуемое свободное место на жёстком диске: не менее 1 Мб;
 - Отображения: дисплей высокой четкости (720p), который больше 9” по диагонали, 8 бит на цветной канал;
 - Среда разработки, например, Clion последней версии, Microsoft Visual Studio и другие;
1. Запуск программы.

Запустите файл main.exe, чтобы начать программу. Для этого не требуется установка. Просто скопируйте исполняемый файл main.exe в любую папку на вашем компьютере и запустите его. Также подготовьте текстовый файл, содержащий необходимую информацию, такую как имя, смещение и сигнатура файла для сканирования. Кроме того, подготовьте исполняемый (exe) файл для сканирования.

2. Работа с программой «Сигнатурный сканер».

После открытия программа отображает приветственное окно с инструкциями. Пользователь вводит первый путь к текстовому файлу, затем второй путь и нажимает Enter. После этого программа отображает результат сканирования. Если сигнатура, взятая из текстового файла, совпадает, мы выводим в консоль "Программа безопасна". Если сигнатуры не совпадают, это означает, что программа является вирусом, и выводится сообщение "Эта программа не безопасна". Программа также сканирует файл на предмет того, является ли он исполняемым (EXE). Пользователь несет ответственность за результат работы программы в случае несоблюдения правил, изложенных в «Руководства пользователя».

ПРИЛОЖЕНИЕ 2. БЛОК-СХЕМА АЛГОРИТМА



ПРИЛОЖЕНИЕ 3. ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct Virus {
    char NameVirus[100];
    int OffseTPosition;
    char sign[6];
    char VirusSignature[6];
    char MZ[2];
    char MZexpected[2];
} vir;

int DataRead(char *textfilepath) {
    FILE *f;
    int ResultforChecking;

    if(textfilepath == NULL){
        return 1;
    }

    f = fopen(textfilepath, "r");
    if (f == NULL)
        return 2;

    if (fgets(vir.NameVirus, sizeof(vir.NameVirus), f) ==
        NULL)
        return 3;
```

```

vir.NameVirus[strcspn(vir.NameVirus, "\n")] = 0;

if (fscanf(f, "%d", &vir.OffsetPosition) != 1)
    return 3;

for (int i = 0; i < 6; i++) {
    if (fscanf(f, "%hhx", &vir.sign[i]) != 1)
        return 3;
}

ResultforChecking = fclose(f);
if (ResultforChecking != 0) return 4;

return 0;
}

```

```

int SeekBegan(FILE *f) {
    if(f==NULL){
        return 1;
    }
    int ResultCheckingSeekBegan;
    ResultCheckingSeekBegan = fseek(f, 0, SEEK_SET);
    if (ResultCheckingSeekBegan != 0) return 2;
    return 0;
}

```

```

int SeekEnd(FILE *f) {
    if(f==NULL){
        return 1;
    }
}

```

```

    }
    int ResultCheckingSeekEnd;
    ResultCheckingSeekEnd = fseek(f, 0, SEEK_END);
    if (ResultCheckingSeekEnd != 0) return 2;
    return 0;
}

int GoToSignatureOffset(FILE *f) {
    if(f==NULL){
        return 1;
    }
    int ResultCheckingGotoSignature;
    ResultCheckingGotoSignature = fseek(f,
    vir.OffsetTPosition, SEEK_SET);
    if (ResultCheckingGotoSignature != 0) return 2;
    return 0;
}

int ScanForMZSignature(FILE *f) {
    if(f==NULL){
        return 1;
    }
    int result;
    vir.MZexpected[0] = 'M';
    vir.MZexpected[1] = 'Z';
    for (int i = 0; i < 2; i++) {
        result = fread(&vir.MZ[i], sizeof(char), 1, f);
        if (result != 1) return 2;
    }
    return 0;
}

```

```

}

int CloseFile(FILE *f) {
    if(f==NULL){
        return 1;
    }
    int ResultcheckingforCLosingfile;
    ResultcheckingforCLosingfile = fclose(f);
    if (ResultcheckingforCLosingfile != 0) return 2;
    return 0;
}

int main() {
    FILE *f;
    char FileName[100];
    char SignatureTxt[100];
    int Result, Length;
    int CChecking;
    char *ScanCheck;

    Result = printf("\n\t** Hello,Welcome to the
signature scanner ** \n \t\t*** Guide of use *** \n "
        "\t\t** Prepare the text file as this
examples * \n "
        "\t\tThe first line : Name of Program. \n
"
        "\t\tSecond line:offset in Decimal \n "
        "\t\tThird line:Signature in HexDecimal
and in 6 bytes with Spaces. \n"

```

```

        "\n*****Examples of text file
*****\nNotepad\n"
        "242835\n"
        "8A 67 25 CA 82 5F\n\n");
if (Result < 0) {
    printf("Error printf!");
    return 1;
}

Result = printf("Enter the path of the text file:
\n");
if (Result < 0) {
    printf("Error printf!");
    return 2;
}

ScanCheck = fgets(SignatureTxt, sizeof(SignatureTxt),
stdin);
if (ScanCheck == NULL) {
    printf("Error: Unable to read user input.\n");
    return 3;
}
SignatureTxt[strcspn(SignatureTxt, "\n")] = 0;

Result = DataRead(SignatureTxt);
switch (Result) {
    case 1:
        printf("The argument is Null");
        return 1;
    case 2:

```

```

        printf("Error of opening the file!\n");
        return 2;
    case 3:
        printf("Error of reading the database\n");
        return 3;
    case 4:
        printf("Error of closing the file!\n");
        return 2;
}

```

```

Result = printf("Enter the path to the file That You
want To scan it : \n");
if (Result < 0) {
    printf("Error printf!");
    return 4;
}

```

```

Result = scanf("%99s",FileName);
if (Result !=1 ) {
    printf("Error: Unable to read user input.\n");
    return 5;
}

```

```

f = fopen(FileName, "rb");
if (f == NULL) {
    printf("Error open file!");
    return 2;
}

```

```

Result = SeekBegan(f);

```

```

if (Result == 1) {
    printf("Error moving to the beginning of the
file!");
    return 1;
}

Result = ScanForMZSignature(f);
if (Result == 1) {
    printf("Error of fread!");
    return 2;
}

if (vir.MZ[0] != vir.MZexpected[0] || vir.MZ[1] !=
vir.MZexpected[1]) {
    Result = printf("\nThis is Safe program\n");
    if (Result < 0) {
        printf("Error printf!");
        return 3;
    }
    Result = CloseFile(f);
    if (Result == 1) {
        printf("Error close file!");
        return 3;
    }
    return 0;
}

Result = SeekEnd(f);
if (Result == 1) {

```

```

        printf("Error when moving to the end of the
file!");
        return 4;
    }
    Length = ftell(f);
    if (Length == -1) {
        printf("Error of ftell!");
        return 5;
    }
    if (Length < vir.OffsetPosition) {
        Result = printf("\nThis is Safe program\n");
        if (Result < 0) {
            printf("Error printf!");
            return 6;
        }
        Result = CloseFile(f);
        if (Result == 1) {
            printf("Error close file!");
            return 3;
        }
        return 0;
    }

    Result = GoToSignatureOffset(f);
    if (Result == 1) {
        printf("Error when moving to the beginning of the
signature!");
        return 7;
    }

```



```

CHecking = 0;
for (int i = 0; i < 6; i++) {
    Result = fread(&vir.VirusSignature[i],
sizeof(vir.VirusSignature[0]), 1, f);
    if (Result != 1) {
        printf("Error fread :(");
        return 8;
    }
    if (vir.VirusSignature[i] == vir.sign[i]) {
        CHecking += 1;
    }
}

if (CHecking == 6) {
    Result = printf("\nThis is a Virus ! \nVirus
name: %s \nLocation file: %s\n", vir.NameVirus,
FileName);
    if (Result < 0) {
        printf("Error printf!");
        return 9;
    }
    Result = CloseFile(f);
    if (Result == 1) {
        printf("Error close file!");
        return 3;
    }
    return 0;
} else {
    Result = printf("\nThis is Safe :)\n");
    if (Result < 0) {

```

```
        printf("Error printf!");  
        return 10;  
    }  
    Result = CloseFile(f);  
    if (Result == 1) {  
        printf("Error close file!");  
        return 3;  
    }  
    return 0;  
}  
}
```