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. 4361 _____ Артемьев Д. Н.

Преподаватель _____ Халиуллин Р. А.

Санкт-Петербург

2025

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Артемьев Д. Н.

Группа 4361

Тема работы: Приложение «Сигнатурный сканер»

Исходные данные:

Разработать на языке программирования C или C++ (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, результаты тестирования программы, заключение, список использованных источников, приложение 1 – руководство пользователя, приложение 2 – блок-схема алгоритма, приложение 3 – исходный код программы.

Предполагаемый объем пояснительной записки:

Не менее 40 страниц.

Дата выдачи задания: 09.03.2025

Дата сдачи реферата: 02.06.2025

Дата защиты реферата: 02.06.2025

Студент гр. 4361 _____ Артемьев Д. Н.

Преподаватель _____ Халиуллин Р. А.

АННОТАЦИЯ

Цель работы – реализация программы с консольным интерфейсом, выполняющей сигнатурное сканирование исполняемых файлов. Сигнатурный сканер определяет наличие вредоносного содержания в файле, по сигнатуре (заданной последовательности байтов) по определенному смещению.

Язык реализации программы – C. Программа определяет, является ли файл исполняемым по первым двум байтам (format magic), а также проверяет есть ли смысл искать сигнатуру в зависимости от размера файла. Данные о сигнатуре загружаются из специального файла, путь к которому может ввести пользователь, как и к проверяемому файлу.

Написаны проверки для реализованных и сторонних функций. В интерфейсе используется английский язык.

Результатом работы является приложение, соответствующее заданию.

SUMMARY

The purpose of this work is to implement a program with a console interface that performs signature scanning of executable files. The signature scanner determines the presence of malicious content in a file by a signature (a specified sequence of bytes) at a certain offset.

The program language is C. The program determines whether a file is executable by the first two bytes (format magic) and also checks whether it makes sense to search for a signature depending on the file size. Signature data is loaded from a special file, the path to which can be entered by the user, as well as to the file to be checked.

Checks for implemented and third-party functions are written. English language is used in the in-interface.

The result of the work is an application corresponding to the task.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы – реализация программы «Сигнатурный сканер» с консольным интерфейсом, позволяющая выполнить сигнатурное сканирование исполняемый файлов, определить наличие вредоносного содержания в файле по сигнатуре (по последовательности байтов) по определенному смещению.

Программа написана на языке программирования C.

Программа работает следующим образом: пользователь вводит пути к файлу, который необходимо проверить, и к файлу, который содержит информацию о сигнатуре, которая состоит из названия вредоносного образца, искомой сигнатуры и ее смещения. Далее программа проверяет, является ли файл исполняемым и достаточно ли он большой, чтобы искать в нем сигнатуру. Если в ходе работы программы не будет никаких ошибок, то пользователю выдастся результат сканирования: безопасно ли открывать файл или нет. В противном случае будет показано уведомление об ошибке.

Код содержит две реализованные функции.

Первая – isExe. В качестве аргумента принимает указатель на открытый файл и указатель на переменную, в которую необходимо передать результат работы программы, и выясняет, является ли данный файл исполняемым, проверяя, равны ли первые два байта файла сигнатуре формата EXE.

Вторая – isSizeEnough. В качестве аргументов принимает указатель на открытый файл, значение смещения сигнатуры из файла, а также размер сигнатуры и указатель на переменную, в которую необходимо передать результат работы программы, и проверяет, поместится ли сигнатура в данный файл.

Внутри реализованных функций используются многочисленные сторонние функции для работы с файлами, для каждой из них реализована проверка.

Кроме того, в функции main были написаны проверки ввода и вывода всего, кроме сообщений об ошибках.

Так что, если описывать кратко, то программа принимает пути к проверяемому файлу и файлу с информацией, проверяет, является ли файл исполняемым, вместит ли он сигнатуру вредоносного образца и содержит ли ее.

1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Наиболее популярным методом детектирования вредоносных программ является проверка сигнатур. Под сигнатурой понимается: фрагмент (или набор фрагментов), который всегда встречается в конкретном вирусе и никогда – в иных программах (в том числе и в других вирусах). Сигнатуры используются для детектирования вредоносных программ, сохраняющих свой код постоянным от копии к копии. Кроме того, сигнатурный поиск можно применить и для поиска некоторых разновидностей полиморфных вирусов. Единственная разновидность вирусов, к которой совсем не применимо сигнатурное детектирование, – это «метаморфы», то есть вирусы, использующие идеи замены и пермутации (перемешивания) команд для всего своего тела.

В идеале сигнатура должна включать в себя всю постоянную часть вируса, но на практике – для минимизации требуемой памяти и ускорения поиска в файле – используются сравнительно короткие цепочки байтов, состоящие из нескольких отрезков.

Лет 15–20 назад, когда вирусов было мало, вирусолог мог позволить себе тщательно изучить код очередного представителя «электронной фауны» и выбрать оптимальный вариант сигнатуры. Пример подобного подхода можно найти в главе, посвященной загрузочным вирусам: там использовали в качестве сигнатуры вируса Stoned. AntiEXE «узловой» фрагмент кода, что позволило антивирусу различать «здоровые», «больные» и «вылеченные» секторы и участки памяти. В современных условиях такой подход труднопременим, так как, например, дежурной смене «дятлов» (как в Лаборатории Касперского величают вирусных аналитиков) приходится ежедневно иметь дело с сотнями и тысячами новых вирусов и троянских программ! Существует потребность в алгоритмах, позволяющих автоматизировать процесс выбора «хороших» сигнатур.

И такие алгоритмы есть. Например, можно вести большую базу данных со всевозможными сигнатурами всевозможных вирусов, а также наиболее

типичных прикладных и системных программ. Тогда в качестве «хорошей» сигнатуры вируса можно выбирать любую цепочку его байтов, не встретившуюся в этой базе. Но, с одной стороны, такая база будет очень велика. С другой – нет никакой гарантии, что эта сигнатура не встретится в какой-нибудь прикладной программе, отсутствующей в базе. Как, например, произошло в 2011 г. с антивирусом Avira, который случайно внес в свои антивирусные базы сигнатуру, характерную для программного когда... антивируса Avira.

Основной недостаток сигнатурного детектирования – неспособность обнаружения «новых», еще не изученных вирусов. Кроме того, неприятной особенностью сигнатурных антивирусов является большой объем справочных данных. Согласно материалам «Лаборатории Касперского», количество записей в антивирусных базах растет по экспоненте.

Существенно уменьшить объемы требуемой памяти позволяет отказ от сигнатур в пользу контрольных сумм. Контрольная сумма – результат применения к произвольному набору данных некой хешфункции, рассчитывающей короткий «дайджест» постоянной длины (например, всего 4 байта).

В базе данных антивируса можно хранить не сами сигнатуры, а их короткие контрольные суммы. Такие же суммы рассчитываются «на лету» по содержимому тестируемых файлов. Несовпадение хранимого образца с результатом расчета означает отсутствие соответствующего вируса. А совпадение – лишь очень высокую (но не единичную!) вероятность заражения. Причина кроется в сути контрольных сумм: они, как и любые другие хешфункции, представляют собой отображение большого множества «объектов»-прообразов на малое множество «дайджестов»-образов. Соответственно, всегда возможна коллизия: несколько различных «объектов» могут иметь одинаковые «дайджесты», в частности «плохие» контрольные суммы могут оказаться не только у зараженных, но и у вполне «здоровых» файлов.

Ранее неоднократно отмечалось, что одним из важнейших критериев качества современного антивируса является эффективность использования

вычислительных ресурсов. Речь идет о быстродействии, требованиям к дисковой и оперативной памяти и т. п. На момент написания этих строк известны несколько миллионов разновидностей вредоносных программ, следовательно, антивирус в общем случае вынужден выполнять такое же количество детектирующих операций (например, сравнений сигнатур) по отношению к каждому файлу. Неудивительно, что антивирусный монитор способен вызывать многосекундные задержки при контроле доступа к файлам, а антивирусный сканер – затрачивать десятки часов на проверку диска. Дошло до того, что типичный пользователь при покупке выбирает не тот антивирус, который «знает больше» или «лечит лучше», а тот, который «работает быстрее».

Надо сразу оговориться, что в современных условиях удовлетворительного решения проблемы, по-видимому, не существует. Современная антивирусная реализация – это клубок компромиссов: одни производители жертвуют надежностью детектирования, другие – быстродействием, третьи – количеством распознаваемых вредоносных программ, четвертые – возможностью лечения и т. п.

Итак, задача ставится следующим образом. Имеются две таблицы (базы данных) с записями, содержащими два атрибута: «позиция в файле» и «сигнатура постоянной длины». Первая таблица – модель тестируемого файла, вторая – набора вирусных сигнатур. Цель поиска: либо найти единственную запись, встречающуюся сразу в обеих таблицах, либо удостовериться в отсутствии таковой. Необходимо удовлетворить двум взаимнопротиворечивым требованиям: 1) минимизировать время поиска в наихудшем случае (то есть когда пересечения нет); 2) минимизировать размеры баз.

Итак, первая таблица – это тестируемый файл. Атрибут «позиция» в нем физически отсутствует, а цепочки байтов, соответствующих различным файловым смещениям, играют роль атрибута «сигнатура».

Вторая таблица – это справочная база вирусов. В ней уникальны только пары «позиция+сигнатура», но по отдельности и «позиции», и «сигнатуры» могут встречаться неоднократно. На физическую структуру этой базы пока

не накладывается никаких ограничений, но именно ее мы будем оптимизировать.

Существует довольно много направлений оптимизации, которые могут применяться как по отдельности, так и в комплексе.

2. РЕАЛИЗАЦИЯ ПРОГРАММЫ

2.1. Использованное ПО

Используемый язык программирования: C.

Операционная система: Windows 10 (x64).

Среда разработки: Code::Blocks 20.03 (x64).

Компилятор: GNU GCC Compiler.

Hex-редактор: HxD Hex Editor 2.5.

2.2. Реализованные функции

Всего в программе есть две реализованные функции, а именно isExe и isSizeEnough, а также функция main.

1. Функция isExe.

Функция isExe определяет, является ли файл исполняемым по первым двум байтам (format magic).

Исходный код функции isExe находится в файле main.c.

Объявление функции:

```
int isExe(FILE* f, int* not_exe);
```

Тип функции: int.

Аргументы функции:

- `f` – указатель на файл, тип которого необходимо проверить, тип аргумента: `FILE*`;
- `not_exe` – указатель на переменную, в которую необходимо передать результат работы функции, тип аргумента: `int*`.

Возвращаемые функцией значение:

- 0 – функция завершилась без ошибок;
- 1 – значение первого аргумента является некорректным;
- 2 – значение второго аргумента является некорректным;
- 3 – ошибка работы функции `fread`;

- 4 – ошибка вывода.

2. Функция isSizeEnough.

Функция isSizeEnough проверяет, достаточного ли размера файл, чтобы в нем могла находиться сигнатура.

Исходный код функции isSizeEnough находится в файле main.c.

Объявление функции:

```
int isSizeEnough(FILE* f, long offset, size_t signature_size, int* not_enough);
```

Тип функции: int.

Аргументы функции:

- `f` – указатель на файл, размер которого надо проверить, тип аргумента: `FILE* f;`
- `offset` – смещение, по которому будет находиться сигнатура, тип аргумента: `long`;
- `signature_size` – длина сигнатуры из файла с информацией, тип аргумента: `size_t`;
- `not_enough` – указатель на переменную, в которую необходимо передать результат работы функции, тип аргумента: `int*`.

Возвращаемые функцией значение:

- 0 – функция завершилась без ошибок;
- 1 – значение первого аргумента некорректно;
- 2 – значение второго аргумента некорректно;
- 3 – значение третьего аргумента некорректно;
- 4 – значение четвертого аргумента некорректно;
- 5 – ошибка работы функции `fseek`;
- 6 – ошибка работы функции `ftell`;
- 7 – ошибка вывода.

3. Функция main.

Функция main служит отправной точкой для выполнения программы.

Исходный код функции main находится в файле main.c.

Объявление функции:

```
int main();
```

Тип функции: int.

У данной функции нет аргументов.

Возвращаемые функцией значение:

- 0 – функция завершилась без ошибок, файл содержит вредоносный образец;
- 1 – ошибка вывода;
- 2 – ошибка ввода;
- 3 – ошибка работы функции `fopen`;
- 4 – значение первого аргумента функции `isExe` некорректно;
- 5 – значение второго аргумента функции `isExe` некорректно;
- 6 – ошибка работы функции `fread`;
- 7 – ошибка работы функции `fclose`;
- 8 – функция завершилась без ошибок, файл безопасно открывать, то есть, либо он не исполняемый, либо слишком небольшой, либо сигнатура не найдена;
- 9 – ошибка работы функции `fscanf`;
- 10 – значение первого аргумента функции `isSizeEnough` некорректно;
- 11 – значение второго аргумента в функции `isSizeEnough` некорректно;
- 12 – значение третьего аргумента в функции `isSizeEnough` некорректно;
- 13 – значение четвертого аргумента в функции `isSizeEnough` некорректно;
- 14 – ошибка работы функции `fseek`;
- 15 – ошибка работы функции `ftell`.

3. РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ ПРОГРАММЫ

На рисунке 1 представлено то, что видит пользователь при запуске программы.

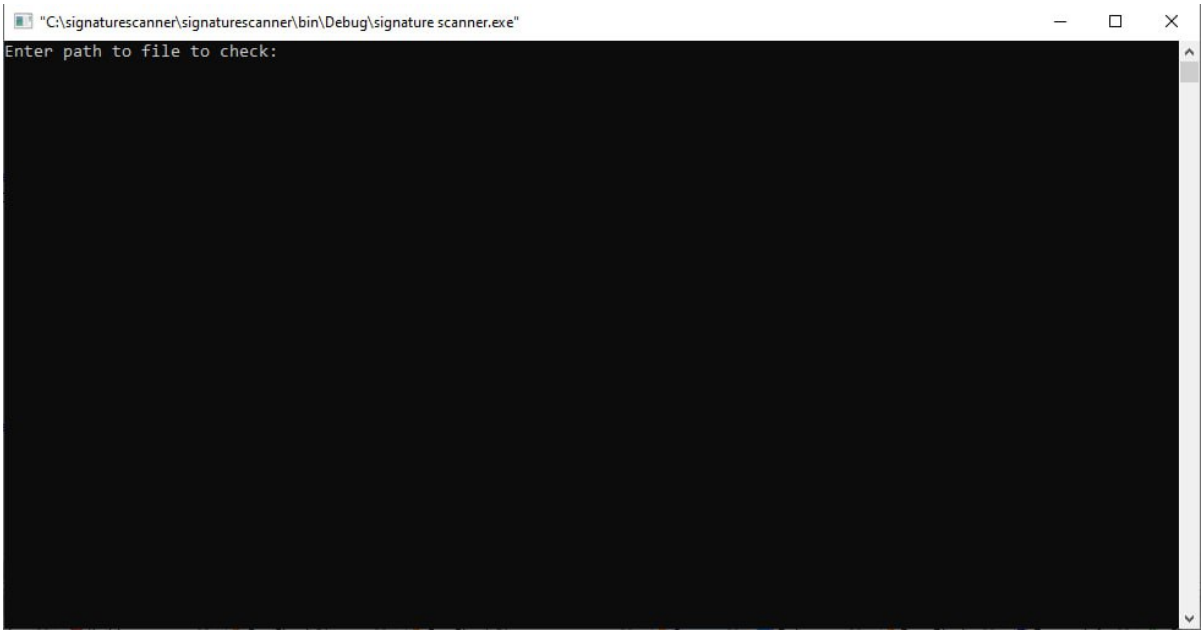


Рисунок 1 – Стартовый экран программы

Пользователю предложено ввести путь к файлу, который нужно проверить. На рисунке 2 показано, что вернет программа: она попросит ввести путь к файлу с информацией.

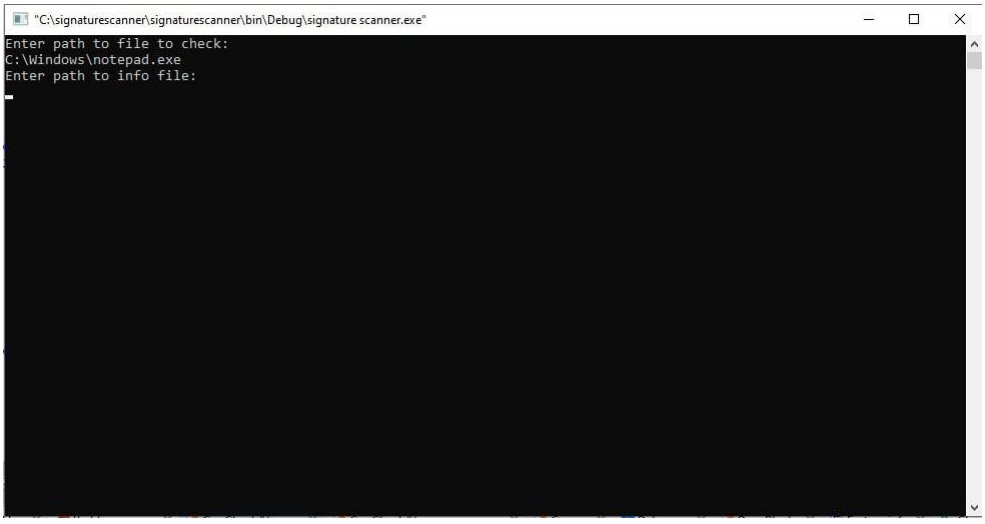


Рисунок 2 – Ввод путей к файлам

На рисунке 3 показано, что программа выводит в случае обнаружени вредоносной сигнатуры в файле.

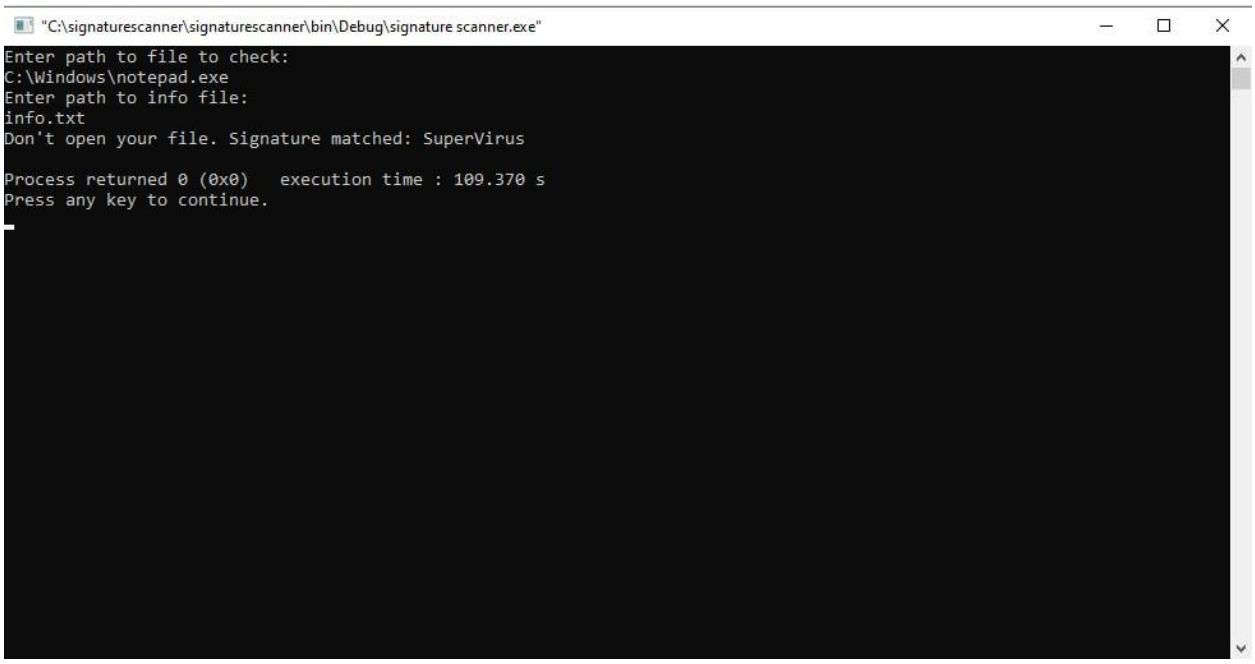


Рисунок 3 – Случай обнаружения вредоносного образца в файле

На рисунке 4 показано, что программа выводит в случае попытки проверки не исполняемого файла.

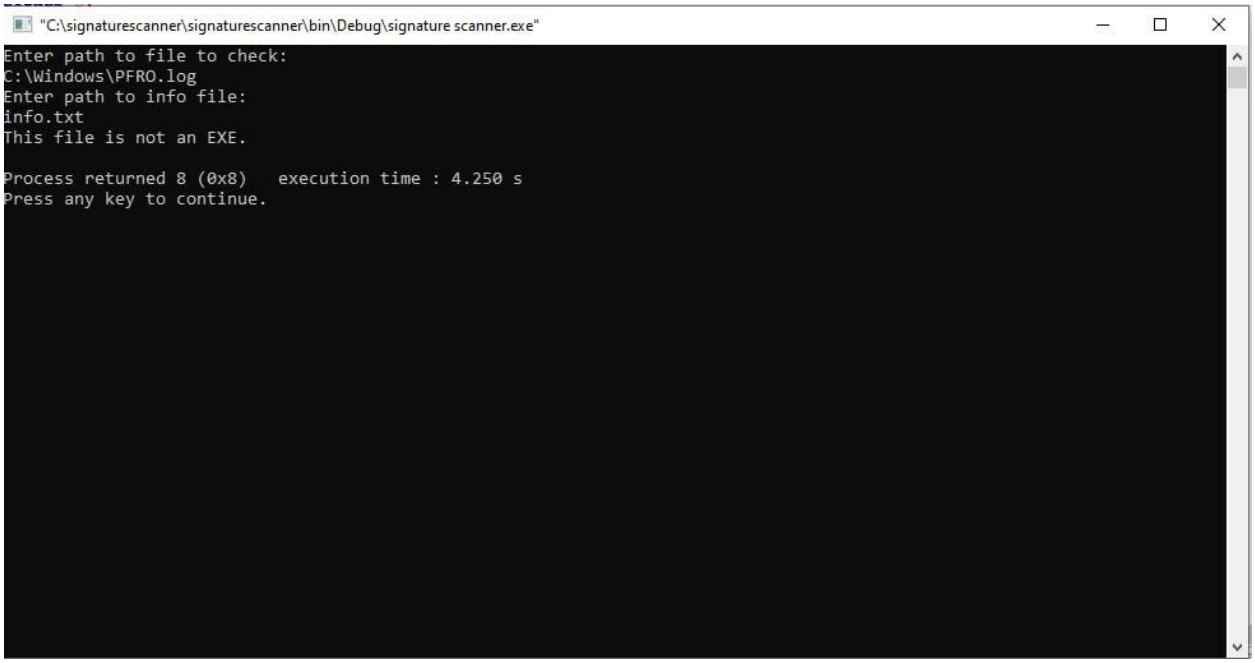


Рисунок 4 – Случай проверки не исполняемого файла

На рисунке 5 показано, что программа выводи в случае предоставления недостаточно большого файла для нахождения там данной сигнатуры.

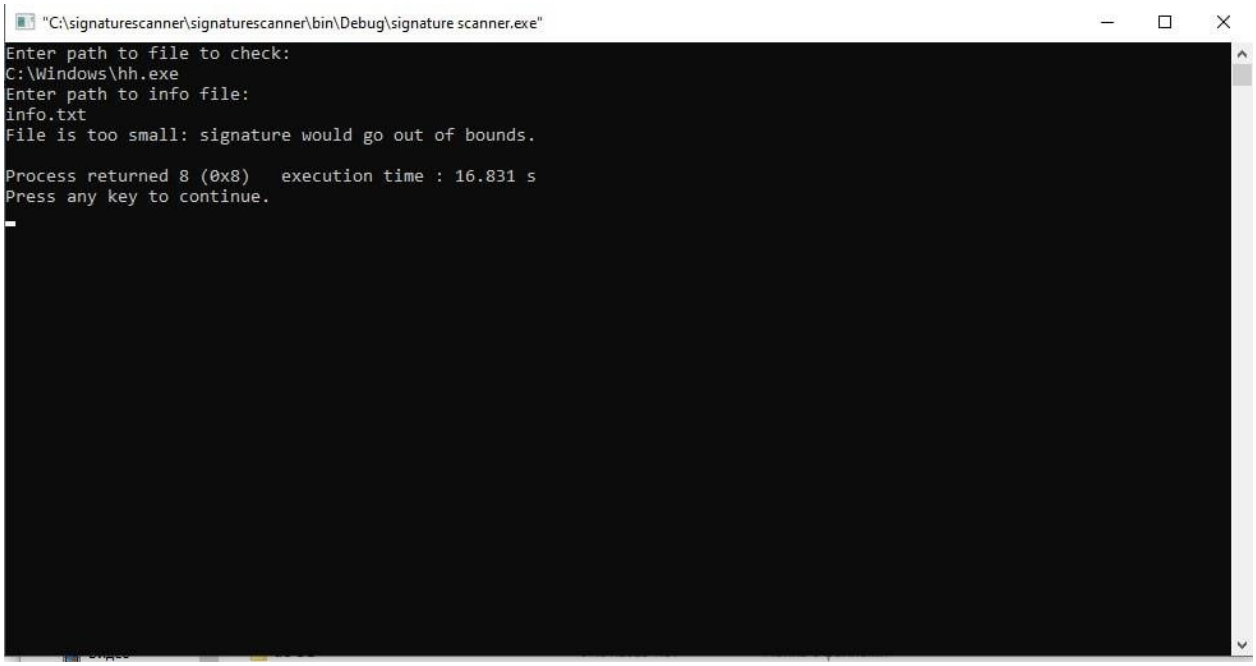


Рисунок 5 – Случай предоставления недостаточно большого файла

На рисунках 6 и 7 показана обработка ошибок открытия файла из-за некорректности путей.

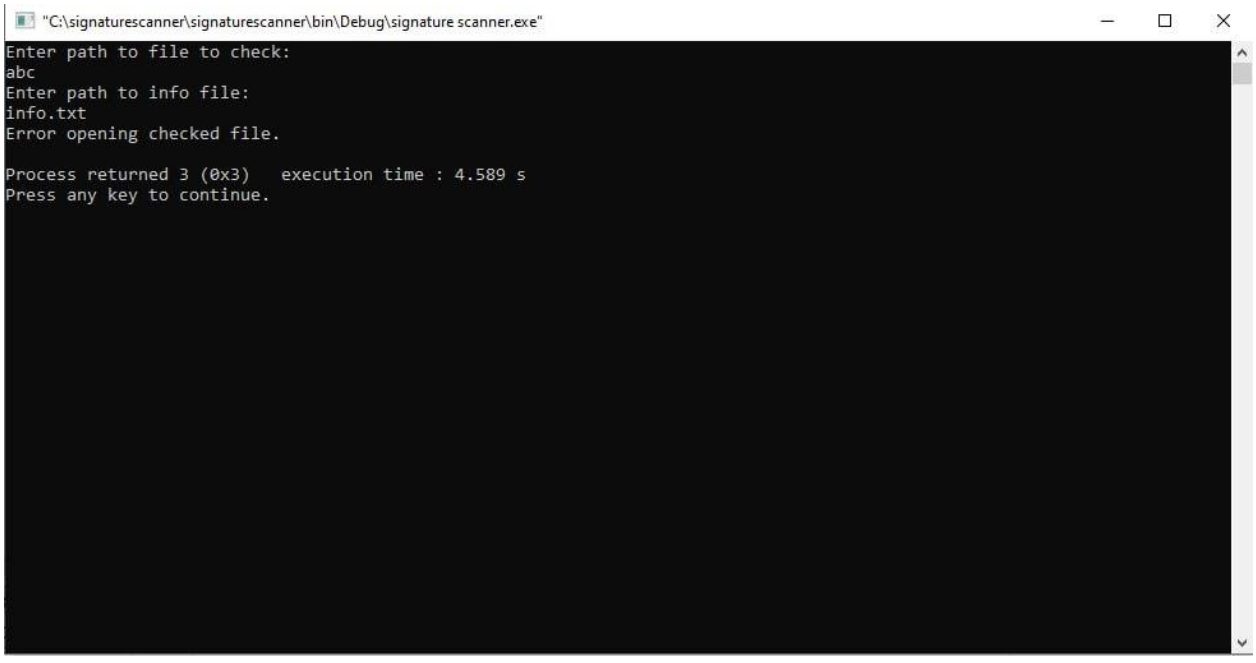


Рисунок 6 – Обработка ошибки открытия проверяемого файла

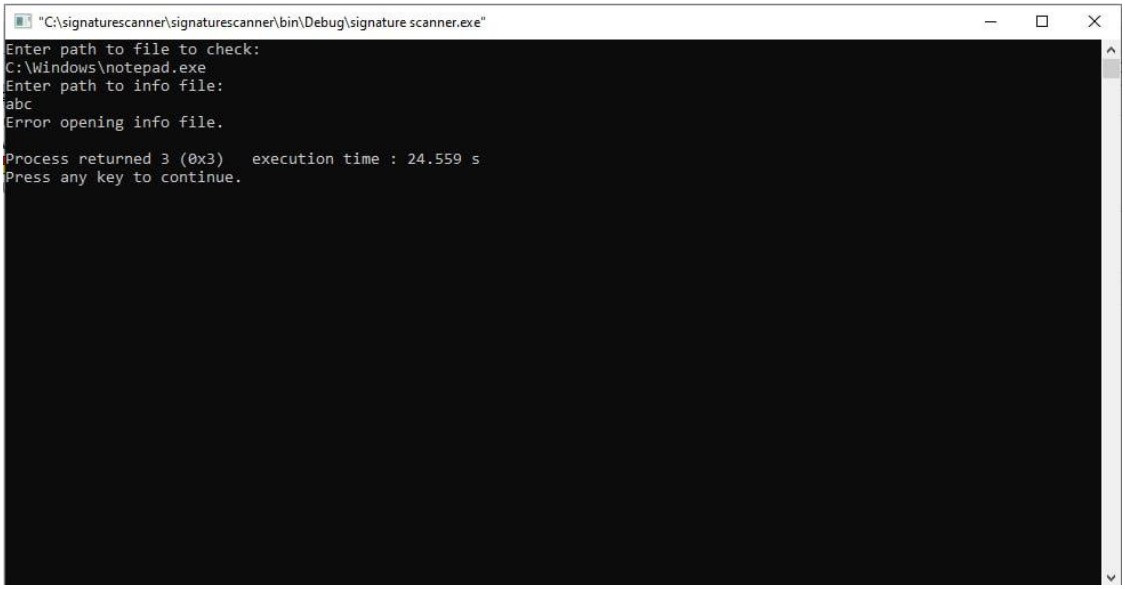


Рисунок 7 – Обработка ошибки открытия информационного файла

На рисунке 8 показан случай, когда в проверяемом файле не найдена сигнатура, то есть его безопасно открывать.

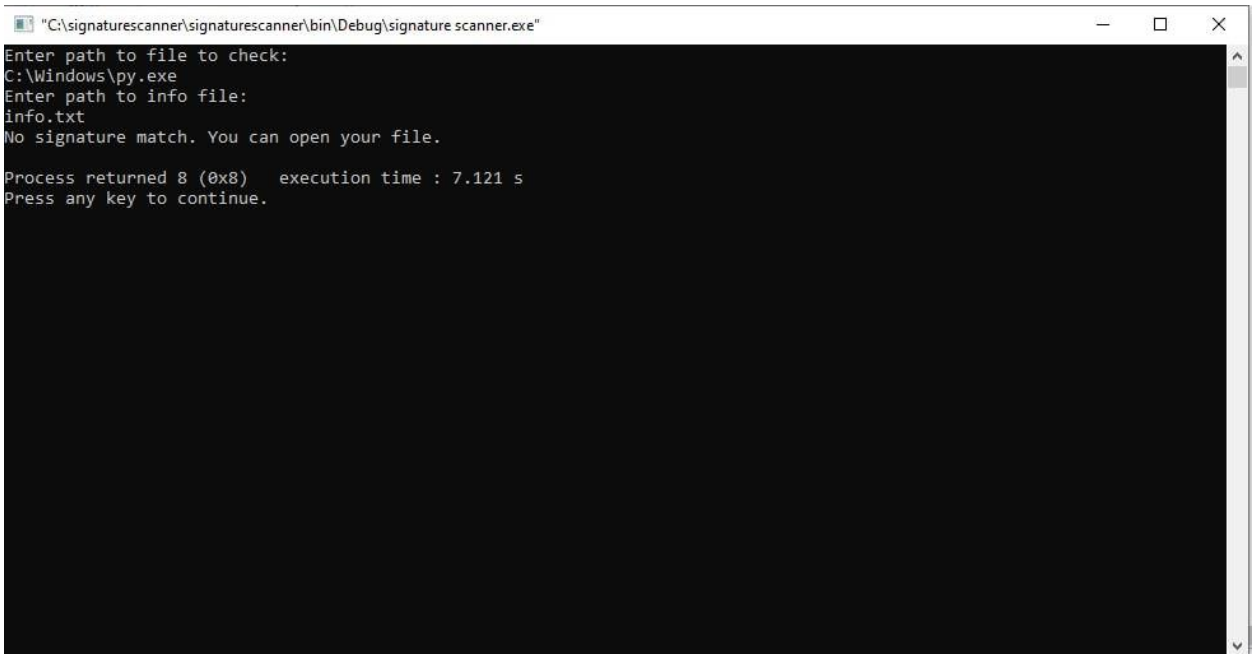


Рисунок 8 – Случай отсутствия в файле вредоносный образцов

На рисунке 9 показан случай обработки ошибки работы функции `fscanf`.

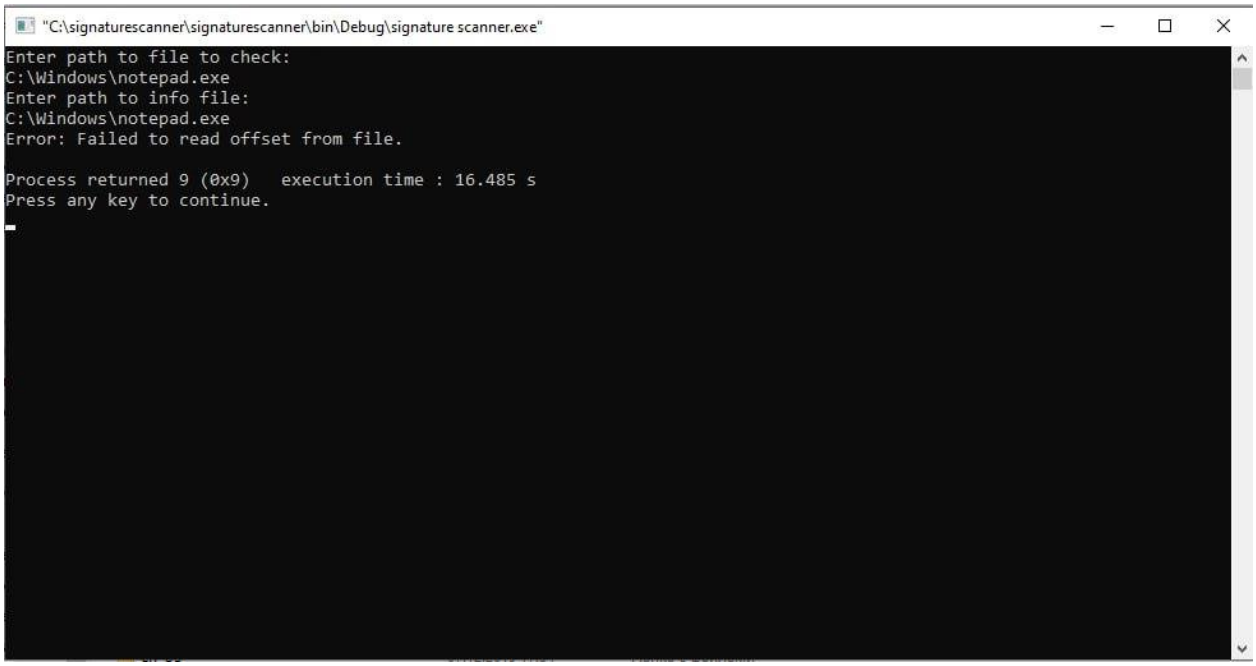


Рисунок 9 – Обработка ошибки функции `fscanf`

Кроме того, в программе реализованны проверки принимаемых функциями аргументов, ввода/вывода и работы сторонних функций для работы с файлами.

ЗАКЛЮЧЕНИЕ

Программа, написанная на языке программирования C, позволяет проверять наличие в файле вредоносных образцов по их сигнатурам по определенному смещению. Интерфейс интуитивно понятен. В коде реализованы функции, а также проверка аргументов и возвращаемых функциями значений. Приложение корректно обрабатывает ошибки.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Климентьев К. Е. К49 Компьютерные вирусы и антивирусы: взгляд программиста. – М.: ДМК Пресс, 2013. – 656 с.: ил. ISBN 978-5-94074-885-4.

ПРИЛОЖЕНИЕ 1. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

1. Краткое описание программы.

Перед вами программа «Сигнатурный сканер». Вы можете ввести пути к файлу, который необходимо проверить, и к файлу, который содержит информацию о сигнатуре, которая состоит из названия вредоносного образца, искомой сигнатуры и ее смещения. Далее программа проверяет, является ли файл исполняемым и достаточно ли он большой, чтобы искать в нем сигнатуру. Если в ходе работы программы не будет никаких ошибок, то вам выдастся результат сканирования: безопасно ли открывать файл или нет. В противном случае будет показано уведомление об ошибке.

2. Минимальные системные требования.

- Операционная система: Windows 10 (x64);
- Процессор: не менее 1 ГГц или SoC;
- ОЗУ: 2 ГБ;
- Место на жестком диске: 20 ГБ;
- Видеоадаптер: DirectX 9 или более поздняя версия с драйвером WDDM 1.0;
- Экран: 800 x 600.

3. Установка программы.

Для установки программы необходимо скопировать исполняемый файл программы signature_scanner.exe и файл с информацией о вредоносных образцах info.txt в выбранную пользователем директорию. Других файлов необходимых для работы программы нет.

4. Запуск программы.

Для запуска программы необходимо запустить исполняемый файл программы signature_scanner.exe из директории, в которую была установлена программа.

5. Работа с программой.

После запуска программы откроется ее стартовый экран, он показан на рисунке 10.

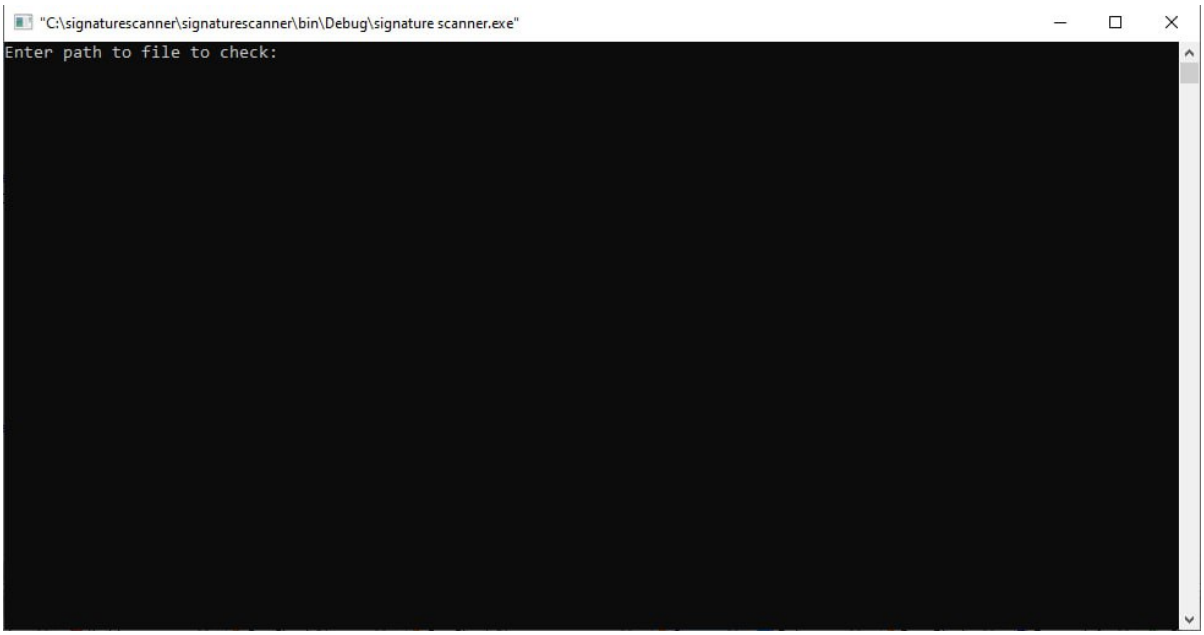


Рисунок 10 – Стартовый экран программы

На нем указана подсказка для пользователя.

То есть необходимо ввести путь к файлу, который необходимо проверить.

После данного ввода программа запросит путь к файлу, в котором хранится информация о вредоносных образцах, что показано на рисунке 11.

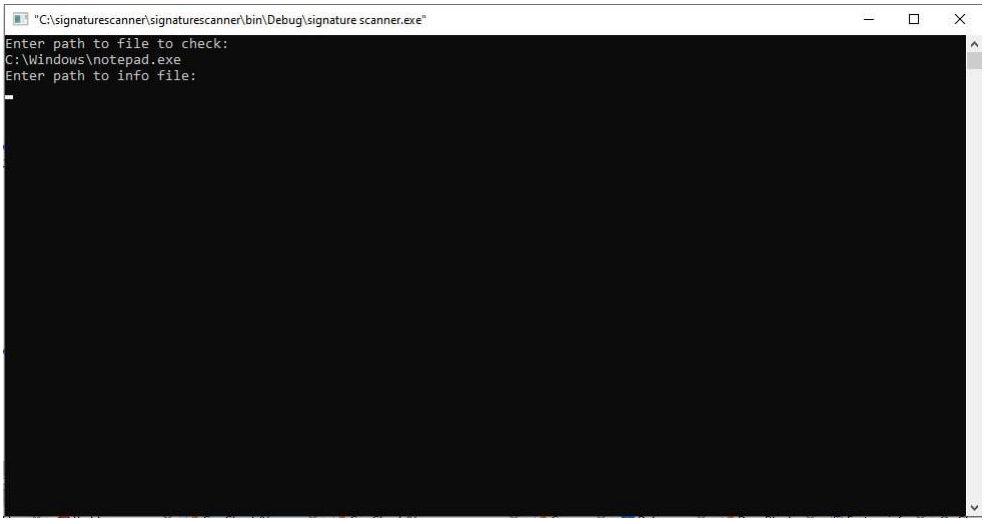


Рисунок 11 – Поля для ввода путей к файлу

Если ввод будет корректным, то программа выведет результат, например, если вредоносный образец найден, то программа оповестит об опасности, как на рисунке 12.

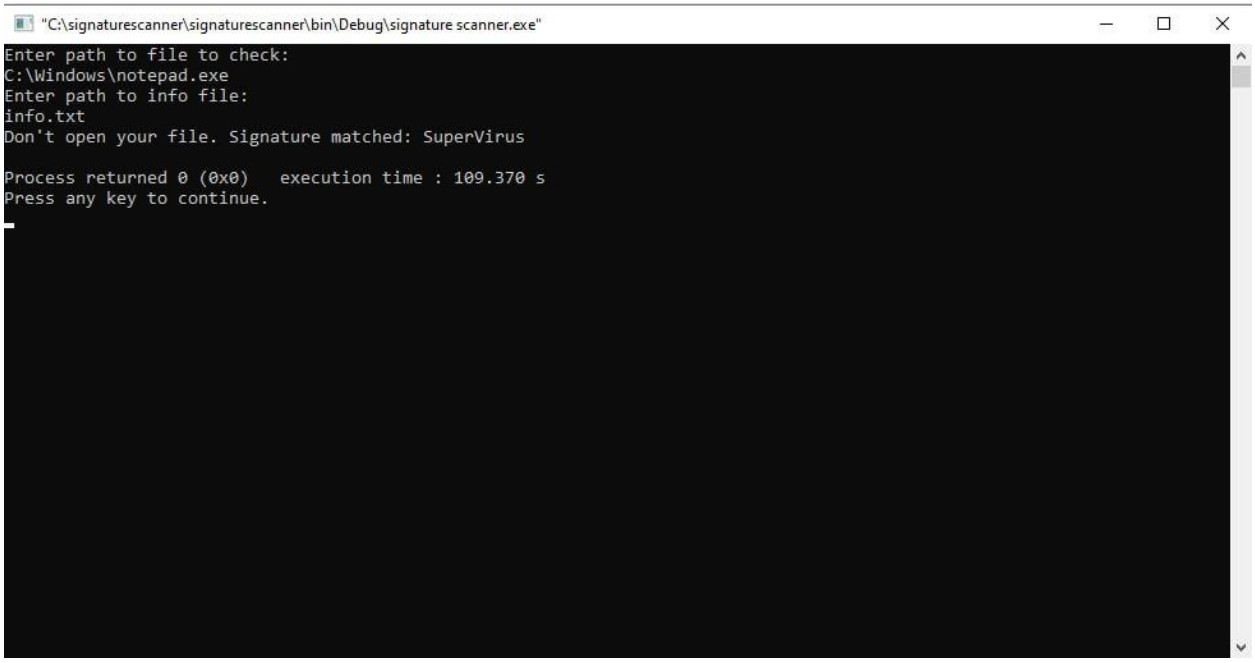


Рисунок 12 – Случай обнаружения вредоносного образца

Если же сигнатура вируса найдена не будет, то программа оповестит, что файл безопасен, как на рисунке 13.

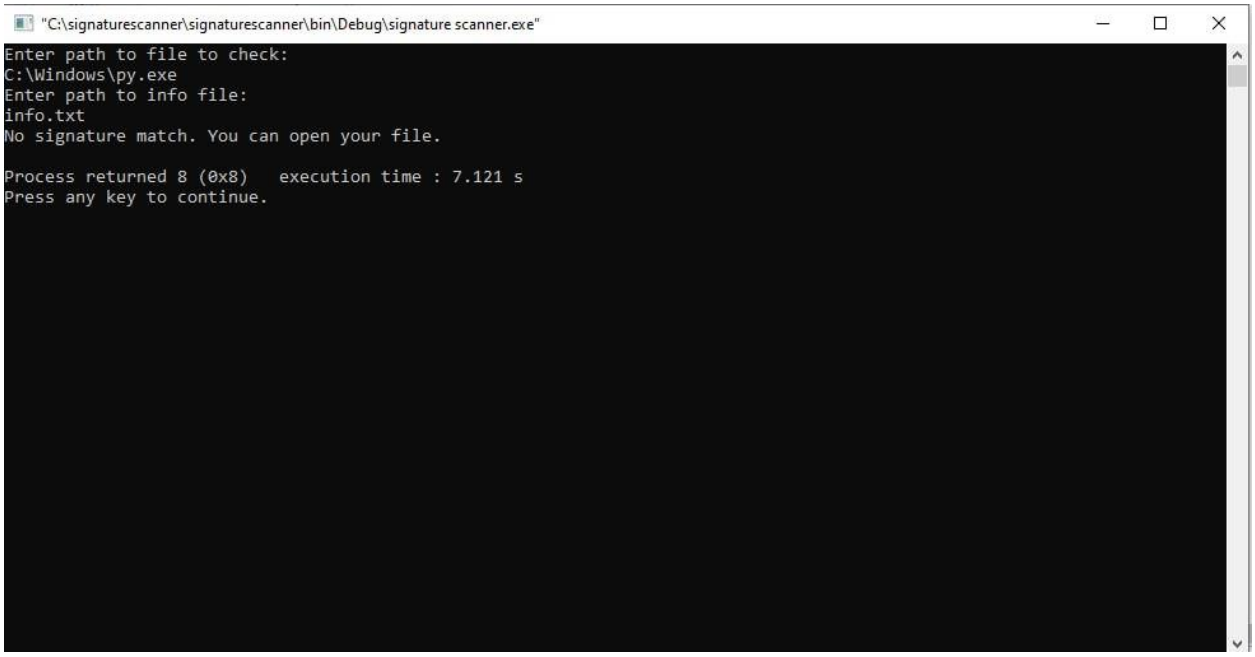


Рисунок 13 – Случай анализа безопасного файла

Кроме случая выше, безопасными считаются, например, неисполняемые файлы. На рисунке 14 показано, что программа выводит в данном случае.

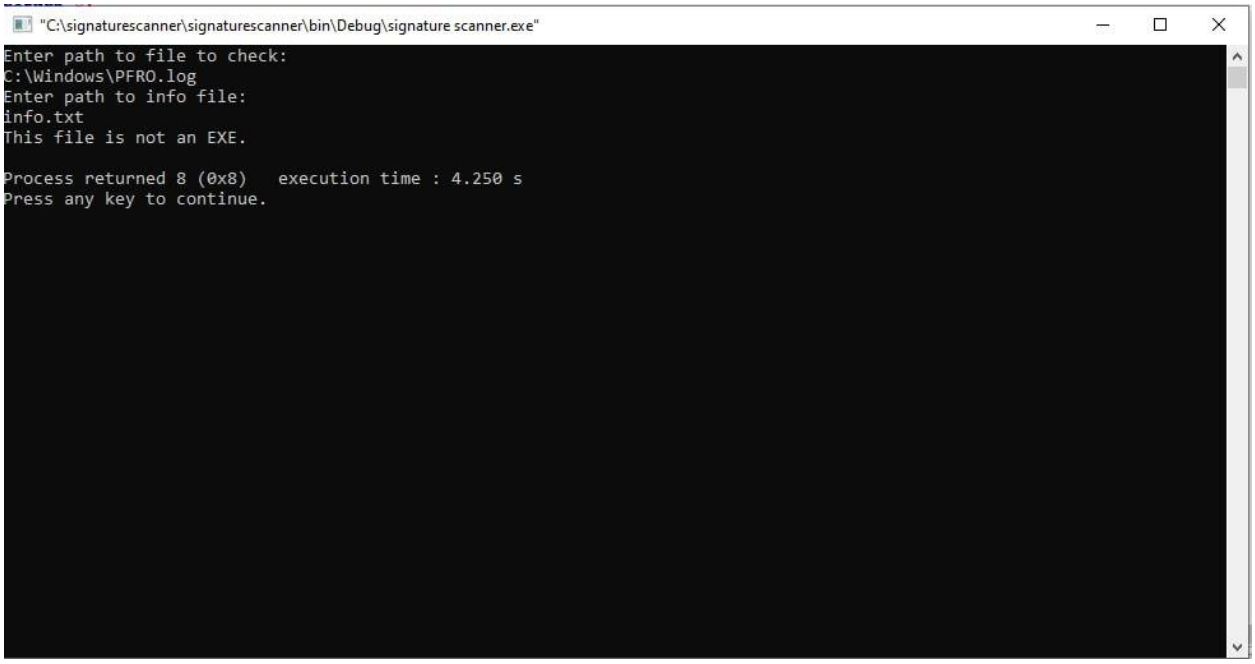


Рисунок 14 – Случай проверки неисполняемого файла

Кроме того, программа проверяет, может ли сигнатура поместиться в проверяемом файле. Случай, если нет, показан на рисунке 15.

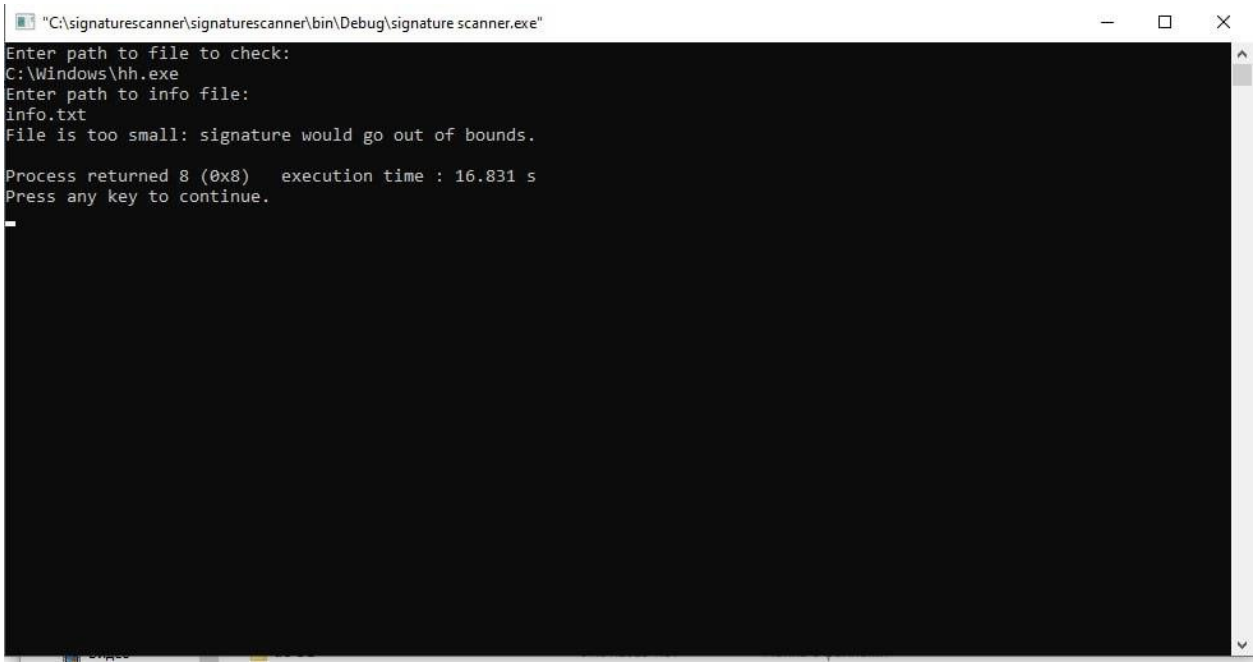


Рисунок 15 – Случай проверки недостаточно большого файла

Как уже было сказано выше, для корректной работы программы необходимо вводить правльные пути к файлам, если же ввести пути с ошибками, программа выдаст ошибку, как на рисунке 16.

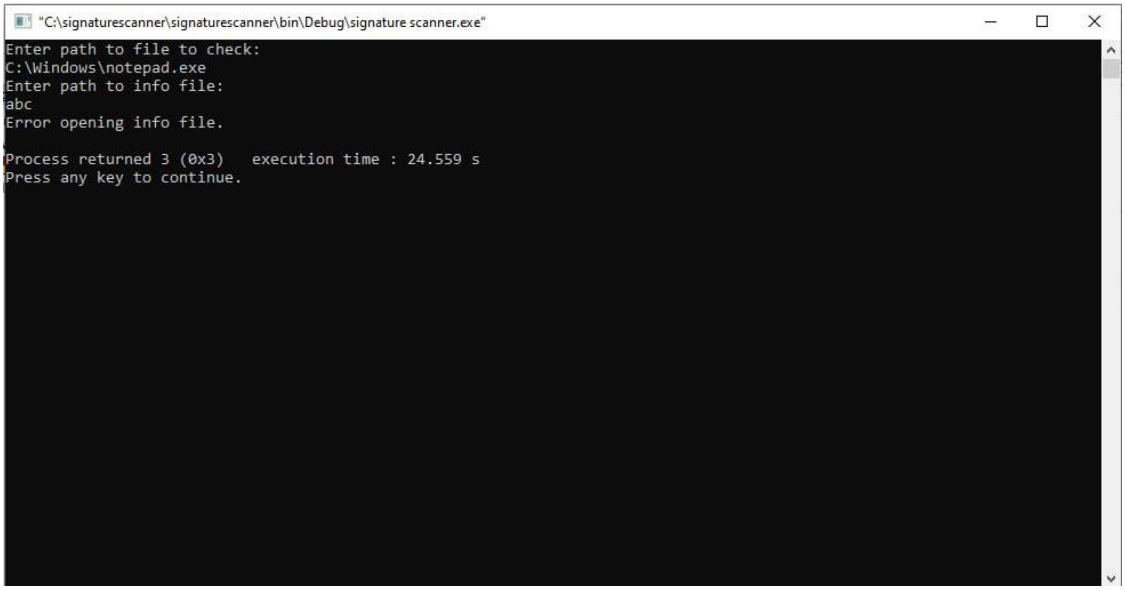


Рисунок 16 – Обработка ошибки ввода пути

Кроме того, если пользователь введет корректный путь, но к файлу, в котором не хранится информация о вредоносных образцах, программа уведомит и об этом, как на рисунке 17.

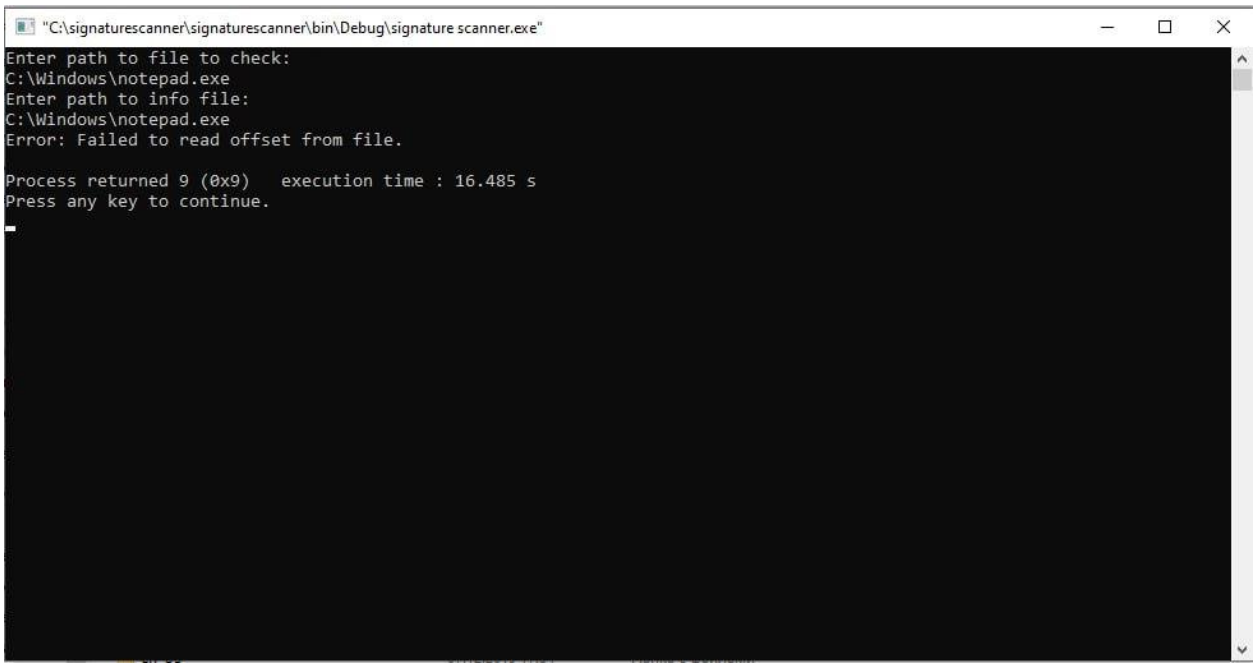
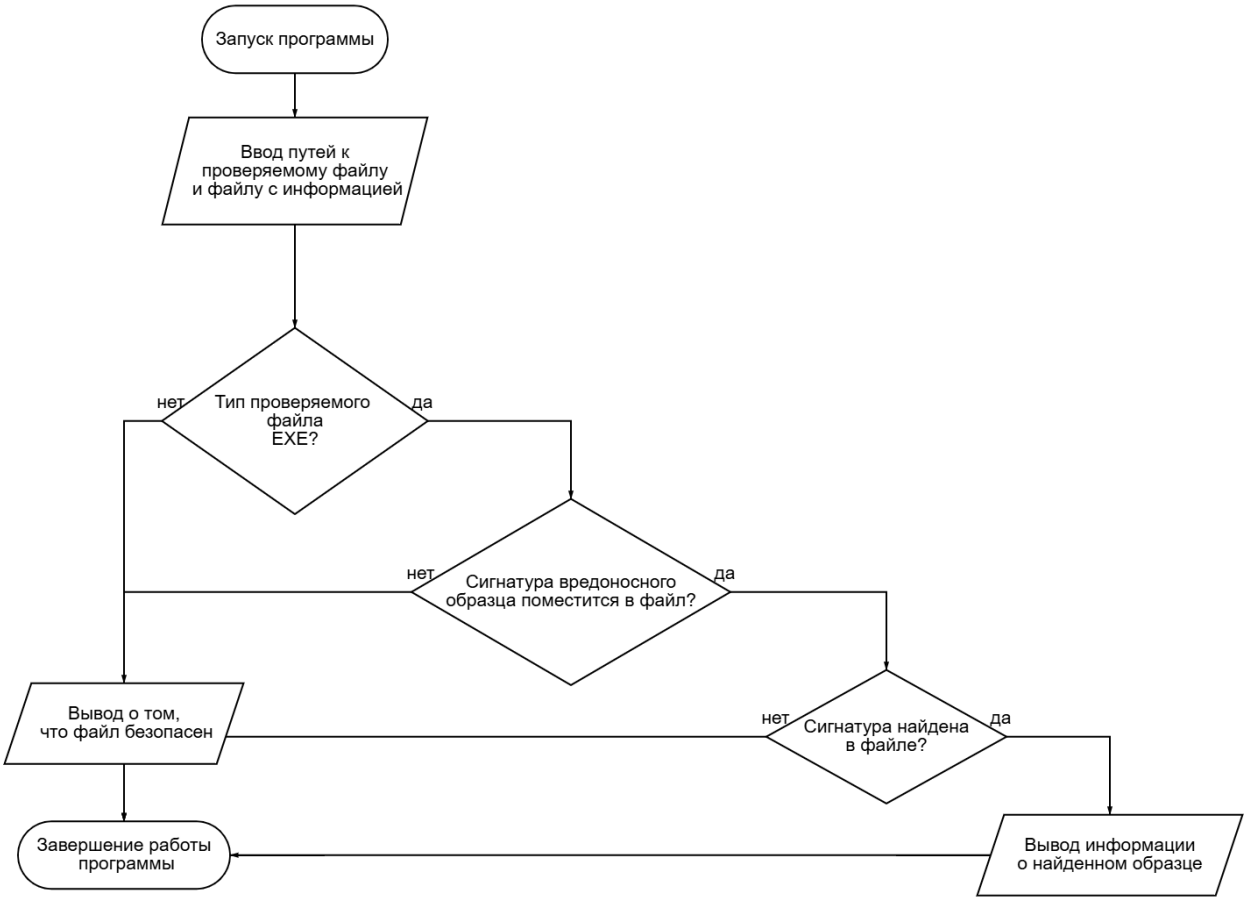


Рисунок 17 – Ошибка чтения данных об образцах

Кроме этого, программа также обрабатывает ошибки получаемых операторов в реализованных функциях, а также ошибки ввода и вывода и всех функций для работы с файлами.

Получается, что при корректном вводе прогорамма выводит корректный результат проверки файла, а при некорректном сообщает пользователю, какие проблемы возникли в процессе работы.

ПРИЛОЖЕНИЕ 2. БЛОК-СХЕМА АЛГОРИТМА



ПРИЛОЖЕНИЕ 3. ИСХОДНЫЙ КОД

Файл main.c:

```
#include <stdio.h>
#include <stddef.h>

int isExe(FILE* f, int* not_exe)
{
    unsigned char format_magic[2];
    int result;

    if (f == NULL)
    {
        printf("Error: NULL file pointer passed to
isExe.\n");
        return 1;
    }

    if (not_exe == NULL)
    {
        printf("NULL pointer Error.\n");
        return 2;
    }


    if                          (fread(format_magic,
sizeof(format_magic[0]),
sizeof(format_magic)/sizeof(format_magic[0]),   f)   !=
sizeof(format_magic)/sizeof(format_magic[0]))
```

```
        {
            printf("Error: Failed to read file header.\n");
            return 3;
        }

        if (format_magic[0] != 'M' || format_magic[1] != 'Z')
        {
            *not_exe = 1;
            if (printf("This file is not an EXE.\n") < 0)
            {
                printf("Output Error.\n");
                return 4;
            }
        }
        return 0;
    }

    int isSizeEnough(FILE* f, long offset, size_t signature_size, int* not_enough)
    {
        long size;
        int result;

        if (f == NULL)
        {
            printf("Error: NULL file pointer passed to isSizeEnough.\n");
```

```
            return 1;
        }
        if (offset < 0)
        {
            printf("Error: Negative offset passed to
is-SizeEnough.\n");
            return 2;
        }
        if (signature_size == 0)
        {
            printf("Error: Zero-length signature passed
to isSizeEnough.\n");
            return 3;
        }
        if (not_enough == NULL)
        {
            printf("NULL pointer Error\n");
            return 4;
        }
        if (fseek(f, 0, SEEK_END) != 0)
        {
            printf("Error: Failed to seek to end of
file.\n");
            return 5;
        }

        size = ftell(f);
        if (size == -1)
        {
```

```
            printf("Error:    Failed    to    get    file
size.\n");
            return 6;
        }

        if (size < (offset + (long)signature_size))
        {
            *not_enough = 1;
            if  (printf("File  is  too  small:  signature
would go out of bounds.\n") < 0)
            {
                printf("Output Error.\n");
                return 7;
            }
        }

        return 0;
    }

    int main()
    {
        char pathchecked[100];
        char infopath[100];
        FILE* checkedFile;
        FILE* infofile;
        unsigned char sign[8];
        unsigned char buffer[8];
        long int offset;
        int result;
        size_t i;
```

```
        char signature_name[100];
        int isExeResult;
        int isSizeEnoughResult;
        int not_exe = 0;
        int not_enough = 0;

        if (printf("Enter path to file to check:\n") <
0)
        {
            printf("Output error.\n");
            return 1;
        }
        if (scanf("%s", pathchecked) != 1)
        {
            printf("Error:   Failed   to   read   file
path.\n");
            return 2;
        }

        if (printf("Enter path to info file:\n") < 0)
        {
            printf("Output error.\n");
            return 1;
        }
        if (scanf("%s", infopath) != 1)
        {
            printf("Error:  Failed  to  read  info  file
path.\n");
            return 2;
        }
```

```
checkedFile = fopen(pathchecked, "rb");
if (checkedFile == NULL)
{
    printf("Error opening checked file.\n");
    return 3;
}

infofile = fopen(infopath, "r");
if (infofile == NULL)
{
    printf("Error opening info file.\n");
    fclose(checkedFile);
    return 3;
}
isExeResult = isExe(checkedFile, &not_exe);
if (isExeResult != 0)
{
    fclose(checkedFile);
    fclose(infofile);
    switch (isExeResult)
    {
        case 1:
            return 4;
            break;
        case 2:
            return 5;
            break;
        case 3:
            return 6;
```

```
                    break;
                case 4:
                    return 1;
                    break;
            }
        }
        if (not_exe)
        {
            result = fclose(infofile);
            if (result != 0)
            {
                printf("Error closing file.\n");
                fclose(checkedFile);
                return 7;
            }
            result = fclose(checkedFile);
            if (result != 0)
            {
                printf("Error closing file.\n");
                return 7;
            }
            return 8;
        }
        result   =   fscanf(infofile,   "%s",   signa-
ture_name);
        if (result != 1)
        {
            printf("Error:  Failed  to  read  signature
name from file.\n");
            fclose(checkedFile);
```

```
            fclose(infofile);
            return 9;
        }

        result = fscanf(infofile, "%lx", &offset);
        if (result != 1)
        {
            printf("Error: Failed to read offset from
file.\n");
            fclose(checkedFile);
            fclose(infofile);
            return 9;
        }
        for(i = 0; i < sizeof(sign); i++)
        {
            result    =    fscanf(infofile,    "%hhx",
&sign[i]);
            if (result != 1)
            {
                printf("Error: Failed to read 8-byte
signature from file.\n");
                fclose(checkedFile);
                fclose(infofile);
                return 9;
            }
        }

        isSizeEnoughResult = isSizeEnough(checkedFile,
offset, sizeof(sign), &not_enough);
        if (isSizeEnoughResult != 0)
```

```
{
    fclose(checkedFile);
    fclose(infofile);
    switch(isSizeEnoughResult)
    {
        case 1:
            return 10;
            break;
        case 2:
            return 11;
            break;
        case 3:
            return 12;
            break;
        case 4:
            return 13;
            break;
        case 5:
            return 14;
            break;
        case 6:
            return 15;
            break;
        case 7:
            return 1;
            break;
    }
}
if (not_enough)
{
```

```
            result = fclose(infofile);
            if (result != 0)
            {
                printf("Error closing file.\n");
                fclose(checkedFile);
                return 7;
            }
            result = fclose(checkedFile);
            if (result != 0)
            {
                printf("Error closing file.\n");
                return 7;
            }
            return 8;
        }
        if (fseek(checkedFile, offset, SEEK_SET) != 0)
        {
            printf("Error:  Failed  to  seek  to  off-
set.\n");
            fclose(checkedFile);
            fclose(infofile);
            return 14;
        }
        if      (fread(buffer,      sizeof(buffer[0]),
sizeof(buffer)/sizeof(buffer[0]),    checkedFile)    !=
sizeof(buffer)/sizeof(buffer[0]))
        {
            printf("Error: Could not read from checked
file.\n");
            fclose(checkedFile);
```

```
            fclose(infofile);
            return 6;
        }

        for (i = 0; i < (sizeof(buffer)/sizeof(buffer[0])); i++)
        {
            if (buffer[i] != sign[i])
            {
                if (printf("No signature match. You can open your file.\n") < 0)
                {
                    printf("Output Error.\n");
                    fclose(checkedFile);
                    fclose(infofile);
                    return 1;
                }
                result = fclose(infofile);
                if (result != 0)
                {
                    printf("Error closing file.\n");
                    fclose(checkedFile);
                    return 7;
                }
                result = fclose(checkedFile);
                if (result != 0)
                {
                    printf("Error closing file.\n");
                    return 7;
                }
```

```
                return 8;
            }
        }
        if (printf("Don't open your file. Signature
matched: %s\n", signature_name) < 0)
        {
            printf("Output Error.\n");
            fclose(checkedFile);
            fclose(infofile);
            return 1;
        }
        result = fclose(infofile);
        if (result != 0)
        {
            printf("Error closing file.\n");
            fclose(checkedFile);
            return 7;
        }
        result = fclose(checkedFile);
        if (result != 0)
        {
            printf("Error closing file.\n");
            return 7;
        }
        return 0;
    }
```

Файл info.txt:

```
SuperVirus 1E684 1F 44 00 00 85 C0 0F 85
```