

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. 3363

Минко Д.А.

Преподаватель

Халиуллин Р.А.

Санкт-Петербург

2024

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент Минко Д.А.

Группа 3363

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, общие сведения о компьютерных вирусах, что такое «компьютерный вирус», какие бывают вирусы, классификация по способу использования ресурсов, классификация по типу заражаемых объектов, классификация по принципам активации, классификация по способу организации программного кода, классификация вирусов-червей, прочие классификации, реализации программы, основные сведения о программном обеспечении, реализованные функции, результаты тестирования программы, сведения о результатах 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 30 страниц.

Дата выдачи задания: 31.03.2024

Дата сдачи реферата: 03.06.2024

Дата защиты реферата: 08.06.2024

Студент

Минко Д.А.

Преподаватель

Халиуллин Р.А.

АННОТАЦИЯ

Курсовая работа посвящена разработке приложения для обнаружения заданного образца в исполняемых файлах по определенной последовательности байтов (сигнатуре) и смещению на языке программирования C.

Работа охватывает процесс проектирования, написание и тестирование программного кода для создания функционального сигнатурного сканера. В работе представлены основные принципы проектирования и написания программного кода для функционирования сканера, включая хранение сигнатур, работы со структурами и внешними файлами, а также обработки ошибок ввода и вывода данных. Приложение оснащено консольным интерфейсом с интуитивно понятными подсказками для пользователя. Помимо этого, в рамках данной работы создано руководство по использованию итогового приложения, включающее в себя пошаговые инструкции, продемонстрирована работа с файлами и предоставлена информация об обработке ошибок в процессе использования программы.

SUMMARY

The course work is devoted to the development of an application for detecting a given pattern in executable files by a certain sequence of bytes (signature) and offset in the C programming language.

The work covers the design process, writing and testing program code to create a functional signature scanner. The work presents the basic principles of designing and writing program code for the operation of the scanner, including storing signatures, working with structures and external files, as well as handling data input and output errors. The application is equipped with a console interface with intuitive user tips. In addition, as part of this work, a manual for using the final application was created, which includes step-by-step instructions,

demonstrated how to work with files, and provided information about error handling while using the program.

СОДЕРЖАНИЕ

Введение	7
1. Общие сведения о компьютерных вирусах	9
1.1. Что такое «компьютерный вирус»	9
1.2. Какие бывают вирусы	9
1.2.1 Классификация по способу использования ресурсов	9
1.2.2 Классификация по типу заражаемых объектов	10
1.2.3 Классификация по принципам активации	11
1.2.4 Классификация по способу организации программного кода	11
1.2.5 Классификация вирусов-червей	12
1.2.6 Прочие классификации	13
2. Реализация программы	14
2.1. Основные сведения о программном обеспечении	14
2.2. Реализованные функции	14
3. Результаты тестирования программы	17
3.1. Сведения о результатах тестирования программы	17
Заключение	22
Список использованных источников	23
Приложение 1. Руководство пользователя	24
Приложение 2. Блок-схема алгоритма	27
Приложение 3. Исходный код программы	28

ВВЕДЕНИЕ

Цель работы заключается в написании программы, которая будет обнаруживать заданный образец в исполняемых файлах по определенной последовательности байтов (сигнатуре) и их смещению. Для реализации был выбран язык программирования С.

Для достижения поставленной цели поставлены следующие задачи:

1. Создать блок-схему, описывающую алгоритм работы данной программы;
2. Реализовать функцию для считывания пути к проверяемому файлу и антивирусной базе, в которой хранятся исходные данные про вирус;
3. Реализовать проверки аргументов реализованных функций;
4. Реализовать проверки возвращаемых функциями значений для всех функций;
5. Реализовать корректную обработку ошибок, которые могут возникать в результате выполнения программы;
6. Разработать «Руководство пользователя».

1. ОБЩИЕ СВЕДЕНИЯ О КОМПЬЮТЕРНЫХ ВИРУСАХ

1.1. Что такое «компьютерный вирус»

Если углубиться в историю происхождения слова «вирус», то можно отметить, что «настоящие» болезнетворные вирусы, то есть сложные молекулы, паразитирующие на живых клетках растений и организмов, получили свое наименование в соответствии с латинским словом «*vīrus*», которое дословно переводится как «яд». Этот термин принадлежит голландцу Мартину Бейерингу, который в самом конце XIX века в научной дискуссии с первооткрывателем вирусов русским ученым Д. И. Ивановским отстаивал гипотезу, что обнаруженные незадолго до этого странные микроскопические объекты являются ядовитыми веществами. Ивановский же считал, что они «живые» и поэтому представляют собой не «вещества», но «существа». В настоящее время признано, что вирусы и не «вещества», и не «существа». Это автономные «обломки» и «испорченные детали» наследственного аппарата клеток, способные внедряться в живую клетку и «перепрограммировать» ее таким образом, чтобы она воспроизводила не себя, а все новые и новые «обломки» и «детали».

Таким образом, в понятии «вирус» главным сейчас считается не ядовитость и вредоносность, а способность к самовоспроизведению.

Компьютерный вирус — это программа, способная к несанкционированному созданию своих функционально идентичных копий.

В данном определении рассмотрим подробнее три ключевых понятия.

Во-первых, основным определяющим признаком вируса является умение воспроизводиться, генерировать себе подобные объекты. Именно эту часть определения имел в виду в середине 80-х годов американский математик Ф. Коэн, впервые в истории произнеся слова «компьютерный вирус» (хотя сам он уверяет, что авторство термина принадлежит его коллеге Л. Адлеману). В те годы возможность существования вирусов рассматривалась в основном только теоретически, и алгоритмы их

функционирования описывались не на языках программирования, а в терминах системы команд математических формализмов типа «машины Тьюринга» или «нормальных алгоритмов Маркова».

Во-вторых, понятие «функциональной идентичности» копий вируса введено в определение ввиду того, что существует класс так называемых полиморфных вирусов, два различных экземпляра которых внешне могут не иметь ничего общего, но выполняют одни и те же действия в соответствии с одним и тем же алгоритмом. Таким образом, полиморфные вирусы идентичны только с точки зрения выполняемых ими функций.

Наконец, понятие «несанкционированный» означает, что вышеупомянутое создание своих копий происходит вне зависимости от желания пользователя. Любая уважающая себя операционная система (например, MS-DOS) тоже способна копировать самое себя, но вирусом не является, поскольку процесс этот происходит с ведома человека.

От компьютерных вирусов необходимо отличать так называемые троянские программы, не обладающие способностью к саморазмножению и предназначенные исключительно для выполнения несанкционированных (как правило, деструктивных) действий. Журналисты и малоквалифицированные пользователи часто смешивают понятия вируса и троянской программы. А ведь между «вирусами» и «троянами» такая же разница, как между «заразой» и «отравой». Мы же не говорим «отравился гриппом» или «заразился цианистым калием», верно? Вот и не надо путать!

1.2. Какие бывают вирусы

Ранее мы уже использовали ряд терминов, относящихся к различным типам вирусов. Теперь рассмотрим эти классификации подробнее.

1.2.1. Классификация по способу использования ресурсов

В настоящее время целесообразно различать вирусы-паразиты (или просто вирусы) и вирусы-черви (или просто черви).

Первые размножаются, используя ресурсы других программ. Они внедряются внутрь этих программ и активируются вместе с их запуском. Эти вирусы могут заражать исполняемые файлы, документы или другие типы файлов.

Вторые, как правило, используют только ресурсы вычислительных систем, такие как оперативная и долговременная память, а также непрограммные файлы. Они распространяют свои копии по сетям, раскладывают их по носителям информации, буферам памяти и другим местам. Черви автономны и не прикрепляются к другим программам.

1.2.2. Классификация по типу заражаемых объектов

В соответствии с этой классификацией вирусы можно разделить на программные, загрузочные, макровирусы и многоплатформенные вирусы.

Программные вирусы заражают файлы других программ. Пример: вирус Win9X.CIH, паразитирующий на Windows-программах.

Загрузочные вирусы заражают или подменяют маленькие программки, находящиеся в загрузочных секторах жестких дисков, дискет и флэшек. Примером может служить вирус Michelangelo.

Питательной средой для макровирусов служат «макросы» или «скрипты», то есть специализированные программные компоненты, написанные на языках сценариев и находящиеся внутри файлов различных офисных приложений – документов MS Word, электронных таблиц MS Excel, изображений Corel Draw и прочего. Примеры: вирус Concept, заражающий документы MS Word; вирус Laroux, заражающий Excel-таблицы.

Многоплатформенные вирусы паразитируют одновременно на объектах различных типов. Например, вирус OneHalf.3544 заражает как программы MS-DOS, так и загрузочные сектора винчестеров. А вирусы семейства Anarchy, кроме программ MS-DOS и Windows, способны заражать также документы MS Word.

1.2.3. Классификация по принципам активации

По этому признаку вирусы целесообразно разделить на резидентные и нерезидентные.

Резидентные вирусы постоянно находятся в памяти компьютера в активном состоянии, отслеживают попытки обращения к жертвам со стороны других программ и операционной системы и только тогда заражают их. Например, исполнимые программы заражаются в момент запуска, завершения работы или копирования их файлов, а загрузочные сектора – в момент обращения к дискетам. Примерами подобных вирусов являются все те же OneHalf.3544 (в среде MS-DOS) и Win9X.CIH (в среде Windows 95/98/ME).

Нерезидентные вирусы запускаются в момент старта зараженных носителей, время их активности ограничено. Например, вирус Vienna.648 «бодрствует» только несколько мгновений сразу после запуска зараженной им программы, но за это время успевает найти на диске множество новых жертв и прикрепиться к ним, а потом передает управление своему носителю и «засыпает» до следующего запуска.

В многозадачных операционных системах возможны «полу-резидентные» вирусы: они стартуют как нерезидентные, организуют себя в виде отдельного потока запущенной программы, весь срок работы этой программы ведут себя словно резидентные, а потом завершают работу вместе с программой-носителем. Пример – Win32.Funlove.4070.

1.2.4. Классификация по способу организации программного кода

Этот таксономический признак позволяет выделять незашифрованные, зашифрованные и полиморфные вирусы.

Незашифрованные вирусы представляют собой простые программы, код которых не подвергается никакой дополнительной обработке. Такие вирусы (например, Vienna.648) легко обнаруживать в программах, исследовать при помощи дизассемблеров и декомпиляторов и удалять.

Код зашифрованных вирусов, как правило, подвергается некоторым видоизменениям. Вирус заражает жертвы своей зашифрованной копией, а после старта расшифровывает ее в памяти ЭВМ. При обнаружении, изучении и удалении таких вирусов возникают трудности, так как вирусологу необходимо как минимум выполнить обратную операцию – расшифровку кода. Обычно зашифровка вирусов сопровождается использованием в коде специальных антиотладочных приемов. Пример такого вируса – Sayha.Diehard.

Наконец, полиморфные вирусы – это разновидность зашифрованных вирусов, которые меняют свой двоичный образ от экземпляра к экземпляру. Например, полиморфными являются все вирусы семейства OneHalf. Частным случаем полиморфных являются метаморфные вирусы, которые не шифруют двоичный образ своего тела, а просто переставляют местами его команды и заменяют их аналогами, выполняющими те же действия. Пример: Win32.ZMyst.

1.2.5. Классификация вирусов-червей

Чаще всего она выполняется по способу распространения. Почтовые черви (например, E-Worm.Win32.Aliz) распространяются по электронной почте, в виде вложений («аттачей») в электронные письма. Сетевые черви (их еще иногда называют «интернет-червями»), такие как, например, Net-Worm.Win32.Lovesan, используют для своего распространения непосредственно сетевые протоколы и рассылают себя внутри информационных пакетов. «Телефонные», или «мобильные», черви (например, Cabir), являющиеся разновидностью «сетевых», при самораспространении пользуются специфическими протоколами беспроводного информационного обмена, такими как Bluetooth. А известные еще с 1980-х годов файловые черви (например, Mkworm.715) самостоятельно не распространяются с компьютера на компьютер, вместо этого они раскладывают свои многочисленные копии по различным каталогам различных носителей информации и «засовывают» их в ZIP- и RAR-архивы.

1.2.6. Прочие классификации

Существует еще немало вирусных таксономий, порой довольно странных. Например, юристам выгодно делить все вирусы на «вульгарные» (состоящие из единого неделимого фрагмента) и «раздробленные» (состоящие из отдельных фрагментов, не являющихся вирусами, но способных объединяться в одну вирусную программу). А журналисты, не имеющие никакого представления о реальном устройстве и возможностях вирусов, обсуждают «четыре поколения деструктивности», причем вирусы, принадлежащие последнему поколению, якобы способны воздействовать аж на человеческий мозг.

2. РЕАЛИЗАЦИЯ ПРОГРАММЫ

2.1. Основные сведения о программном обеспечении

Для написания программы был выбран язык программирования C, операционная система Windows 11 Pro, version 23H2, среда разработки Code::Blocks, компилятор – GNU GCC.

2.2. Реализованные функции

Реализованные функции: path(), main().

Функция main.

Функция main предназначена для проверки безопасности файла. Для этого она сравнивает сигнатуру проверяемого файла с сигнатурой, хранящейся в файле с информацией о вирусе. Результатом работы функции будет информация о том, является ли файл вирусом с его названием или нет.

Исходный код функции main находится в файле main.c.

Объявление функции:

```
int main ();
```

Тип функции: int.

Аргументы функции:

- Отсутствуют.

Возвращаемые функцией значение:

- 0 — функция завершилась без ошибок;
- 1 — ошибка текста вывода в консоль;
- 2 — ошибка функции path() для ввода путей файлов;
- 3 — ошибка открытия антивирусной базы;
- 4 — ошибка считывания имени вируса из антивирусной базы;
- 5 — ошибка считывания смещения сигнатуры вируса из антивирусной базы;
- 6 — ошибка считывания сигнатуры вируса из антивирусной базы;

- 7 — ошибка закрытия антивирусной базы;
- 8 — ошибка открытия бинарной версии проверяемого файла;
- 9 — ошибка проверки является ли проверяемый файл EXE файлом;
- 10 — ошибка вывода текста в консоль;
- 11 — ошибка закрытия проверяемого файла;
- 12 — ошибка смещения в конец бинарной версии проверяемого файла;
- 13 — ошибка получения значения смещения;
- 14 — ошибка вывода текста в консоль;
- 15 — ошибка закрытия бинарной версии проверяемого файла;
- 16 — ошибка смещения в точку исходного смещения бинарной версии проверяемого файла;
- 17 — ошибка считывания сигнатуры из бинарной версии проверяемого файла;
- 18 — ошибка вывода текста в консоль;
- 19 — ошибка закрытия бинарной версии проверяемого файла;
- 20 — ошибка вывода текста в консоль;
- 21 — ошибка закрытия бинарной версии проверяемого файла.

Функция path.

Функция path определяет значение целочисленного деления двух чисел.

Исходный код функции path находится в файле main.c.

Объявление функции:

```
int path (char* PathFileTxt, char* PathFileExe);
```

Тип функции: int.

Аргументы функции:

- PathFileTxt — указатель на массив элементов, в котором хранится путь к антивирусной базе, тип аргумента: char *;

- PathFileExe — указатель на массив элементов, в котором хранится путь к проверяемому файлу, тип аргумента: char *;

Возвращаемое функцией значение:

- 0 — функция завершилась без ошибок;
- 1 — значение первого аргумента функции является некорректным;
- 2 — значение второго аргумента функции является некорректным;
- 3 — ошибка вывода текста в консоль;
- 4 — ошибка считывания пути к антивирусной базе;
- 5 — ошибка вывода текста в консоль;
- 6 — ошибка считывания пути проверяемого файла;
- 7 — ошибка вывода текста в консоль.

3. РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ ПРОГРАММЫ

3.1. Сведения о результатах тестирования программы

1. Ввод пути до антивирусной базы и проверяемого файла.

При запуске программы на экран выводится сообщение, о названии приложения, его авторе, и о том, что нужно ввести путь до антивирусной базы (рисунок 1).

```
"Signature scanner" App (by Minko Dmitriy)
Enter the path for the "Antivirus database":
```

Рисунок 1 – Главный экран при запуске программы

От пользователя требуется ввести путь до антивирусной базы. После ввода пути до антивирусной базы, необходимо ввести путь до проверяемого файла (рисунок 2).

```
Enter the path for the "Antivirus database":
C:\Users\midmi\OneDrive\Study\Sem2\Coursach\{p.txt
Enter the path for the "File to be checked":
```

Рисунок 2 – Запрос ввода пути до проверяемого файла

В случае ввода неправильного пути до антивирусной базы программа выдаст ошибку и завершит работу с возвращенным значением «3» (рисунок 3).

```
"Signature scanner" App (by Minko Dmitriy)
Enter the path for the "Antivirus database":
nfdsfds

Enter the path for the "File to be checked":
C:\Users\midmi\OneDrive\Study\Sem2\Coursach\Notepad.exe

ERROR opening the "Antivirus database" for reading!

Process returned 3 (0x3)   execution time : 16.561 s
Press any key to continue.
```

Рисунок 3 – Ввод неправильного пути до антивирусной базы

В случае ввода неправильного пути до проверяемого файла программа выдаст ошибку и завершит работу с возвращенным значением «8» (рисунок 4).

```
"Signature scanner" App (by Minko Dmitry)

Enter the path for the "Antivirus database":
C:\Users\midmi\OneDrive\Study\Sem2\Coursach\fp.txt

Enter the path for the "File to be checked":
Неправильный путь

ERROR opening the "File to be checked" for reading!

Process returned 8 (0x8)   execution time : 27.028 s
Press any key to continue.
```

Рисунок 4 – Ввод неправильного пути до проверяемого файла

2. Завершение работы программы.

Программа анализирует только исполняемые файлы формата «EXE» в случае, если файл таковым не является, программа выдаст информацию, о том, что такой файл не является вирусом (рисунок 5).

```
Enter the path for the "File to be checked":
C:\Users\midmi\OneDrive\Study\Sem2\Coursach\Screenshot_8.png

It's not a virus.

Process returned 0 (0x0)   execution time : 30.763 s
Press any key to continue.
```

Рисунок 5 – Проверяемый файл не является файлом формата «EXE»

Следующим шагом программа проверит по объему файла может ли в нем существовать сигнатура в принципе. В случае, если проверяемый файл меньше смещения «+7» полученного из антивирусной базы, проверяемый файл не будет являться вирусом (рисунок 6).

```
"Signature scanner" App (by Minko Dmitriy)

Enter the path for the "Antivirus database":
C:\Users\midmi\OneDrive\Study\Sem2\Coursach\{p.txt

Enter the path for the "File to be checked":
C:\Users\midmi\OneDrive\Study\Sem2\Coursach\MInkoDA\bin\Debug\MInkoDA.exe

It's not a virus.

Process returned 0 (0x0)   execution time : 27.174 s
Press any key to continue.
```

Рисунок 6 – Объем проверяемого файла меньше минимально возможного файла, который может являться вирусом

Программа проверит совпадение сигнатур из антивирусной базы и бинарной версии проверяемого файла. Если сигнатуры (рисунок 7) не совпадут программа выдаст информацию о том, что данный проверяемый файл не вирус. В случае совпадения сигнатур, в консоль будет выведена информация об имени данного вируса (рисунок 8).

```
"Signature scanner" App (by Minko Dmitriy)

Enter the path for the "Antivirus database":
C:\Users\midmi\OneDrive\Study\Sem2\Coursach\{p.txt

Enter the path for the "File to be checked":
D:\Games\Epic Games\Games\Red Dead Redemption\RedDeadRedemption2\RDR2.exe

It's not a virus.

Process returned 0 (0x0)   execution time : 20.249 s
Press any key to continue.
```

Рисунок 7 – Сигнатура из антивирусной базы и бинарной версии проверяемого файла не совпадают

```
"Signature scanner" App (by Minko Dmitriy)

Enter the path for the "Antivirus database":
C:\Users\midmi\OneDrive\Study\Sem2\Coursach\{p.txt

Enter the path for the "File to be checked":
C:\Users\midmi\OneDrive\Study\Sem2\Coursach\Notepad.exe

This is a virus called "Inferno Virus"

Process returned 0 (0x0)   execution time : 18.858 s
Press any key to continue.
```

Рисунок 8 – Сигнатура из антивирусной базы и бинарной версии
проверяемого файла совпадают

3. Дополнительная информация о возможных ошибках.
 - a. Ошибка указателя.

В случае если в исходном коде, в реализованных функциях, указатель будет ссылаться на несуществующую область памяти, то программа выведет в консоль «Pointer ERROR!» завершит свою работу с возвращённым ненулевым значением (рисунок 9).

```
"Signature scanner" App (by Minko Dmitriy)

Pointer ERROR!
Input file paths ERROR!

Process returned 2 (0x2)   execution time : 0.032 s
Press any key to continue.
```

Рисунок 9 – Окно вывода программы при ошибке указателя

- b. Ошибка вывода в консоль.

В случае если в исходном коде аргумент функции printf(), будет ссылаться на несуществующую область памяти, то программа выведет в консоль «Printf ERROR!» и завершит свою работу с возвращённым ненулевым значением (рисунок 10).

```
"Signature scanner" App (by Minko Dmitriy)

Enter the path for the "Antivirus database":
C:\Users\midmi\OneDrive\Study\Sem2\Coursach\{p.txt

Enter the path for the "File to be checked":
C:\Users\midmi\OneDrive\Study\Sem2\Coursach\Notepad.exe

Printf ERROR!

Process returned 1 (0x1)   execution time : 45.314 s
Press any key to continue.
```

Рисунок 10 – Окно вывода программы при ошибке функции вывода в
консоль

Ошибки открытия и закрытия файла, считывания из него, перемещение курсора смещения и его положения, а также все другие проверяемые функции обрабатываются, для них предусмотрены свои возвращаемые значения, а также подсказки, выводимые в консоли.

ЗАКЛЮЧЕНИЕ

Во время выполнения данной курсовой работы была достигнута поставленная цель, а именно, было создано приложение на языке C, которое обнаруживает заданный образец в исполняемых файлах по сигнатуре и их смещению. По мере достижения цели также были достигнуты поставленные задачи – создана блок-схема, описывающая алгоритм работы программы, написано руководство пользователя, реализованы функции для считывания пути к исследуемому файлу и антивирусной базе, выполнены проверки аргументов функций, реализованы проверки возвращаемых значений для всех функций, а также обеспечена корректная обработка ошибок, которые могут возникать в процессе выполнения программы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Белецкий Я. Энциклопедия языка Си: Пер. с польск.– М.: Мир, 1992.– 687 с.,ил. ISBN 5-03-002113-2;
2. Керниган Б., Ритчи. Д. Язык программирования С, 2-е изд.: Пер. с англ. – СПб.: ООО «Диалектика», 2020.– 288 с.: ил. – Парал. тит. англ. ISBN 978-5-907144-14-9 (рус.);
3. Климентьев К.Е. Компьютерные вирусы и антивирусы: взгляд программиста. – М.: ДМК-Пресс, 2013. – 656 с. – ISBN 978-5-94074-885-4;
4. Подбельский В. В., Фомин С. С. Программирование на языке Си: Учеб. пособие. – 2-е доп. изд. – М.: Финансы и статистика, 2005. 600 с.: ил. ISBN 5-279-02180-6.

ПРИЛОЖЕНИЕ 1. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Программа «Сигнатурный сканер»

Программа последовательно считывает с клавиатуры путь к файлу с информацией о вирусах и путь к проверяемому файлу, а затем выводит результат проверки данного файла на безопасность. Пути до файла необходимо вводить полные, последовательно, через клавишу «Enter». Для упрощения можно нажать правой кнопкой мыши (далее «ПКМ») на сами файлы и выбрать параметр «Копировать как путь». Перед каждым действием пользователя программа выводит инструкцию для следующего действия, требуемого от пользователя.

Минимальные требования:

- Процессор: 1 гигагерц (ГГц) или быстрее с двумя и более ядрами на совместимом 64-разрядном процессоре или системе на микросхеме (SOC);
- ОЗУ: 4 гигабайта (ГБ);
- Требуемое свободное место на жёстком диске: 78 Кб;
- Служба хранилища: 64 Гб или большее хранилище;
- Программное обеспечение системы: UEFI (для единого extensible Firmware Interface, современной версии PC BIOS) и Secure Boot;
- TPM: доверенный модуль платформы (TPM) версии 2.0;
- Видеокарты: совместим с DirectX 12 или более поздней версии с драйвером WDDM 2.0;
- Отображения: дисплей высокой четкости (720p), который больше 9 дюймов по диагонали, 8 бит на цветной канал;
- Операционная система: Windows 11;
- Среда разработки, например, Code::Blocks последней версии.

1. Установка программы.

Установка не требуется, нужно запустить исполняемый файл Inquisitor.exe, скопировав его в любую директорию.

2. Запуск программы.

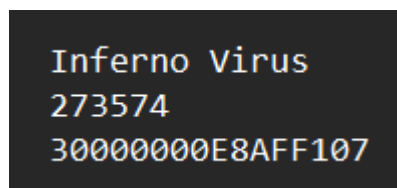
Запустите файл Inquisitor.exe.

3. Работа с программой.

Программа выведет своё название, после запросит пользователя ввести с клавиатуры путь до антивирусной базы в формате строки. Далее программа попросит ввести путь до проверяемого файла в формате строки. В путях до файлов должны отсутствовать символы иноязычных алфавитов, кроме латиницы, так же длина каждого из путей не должна превышать 260 символов. После успешного выполнения двух предыдущих действий будет выведено сообщение о том является проверяемый файл вирусом или нет. В случае, если проверяемый файл является вирусом, выведется его название, которое содержится в антивирусной базе. На протяжении всего времени пользования программой в рабочую консоль выводятся подробные инструкции на английском языке. Ввод каждого пути должен завершаться клавишей «Enter». В случае возникновения ошибки внутри программы, она выдаст номер ошибки, сообщение о ней и досрочно завершит работу. Успешность выполнения программы можно посмотреть в строке return «число». Если оно равно «0» программа выполнена успешно, в ином случае программа завершилась с ошибкой, в этом случае внимательно прочитайте выведенный в консоли комментарий к этой ошибке. В случае несоблюдения правил «Руководства пользователя», пользователь берёт полную ответственность за результат выполнения программы на себя.

Дополнительная информация для разработчиков:

Данные в антивирусной базе должны храниться в строго заданном виде (рисунок 11).



```
Inferno Virus
273574
30000000E8AFF107
```

Рисунок 11 – Пример хранения данных в антивирусной базе

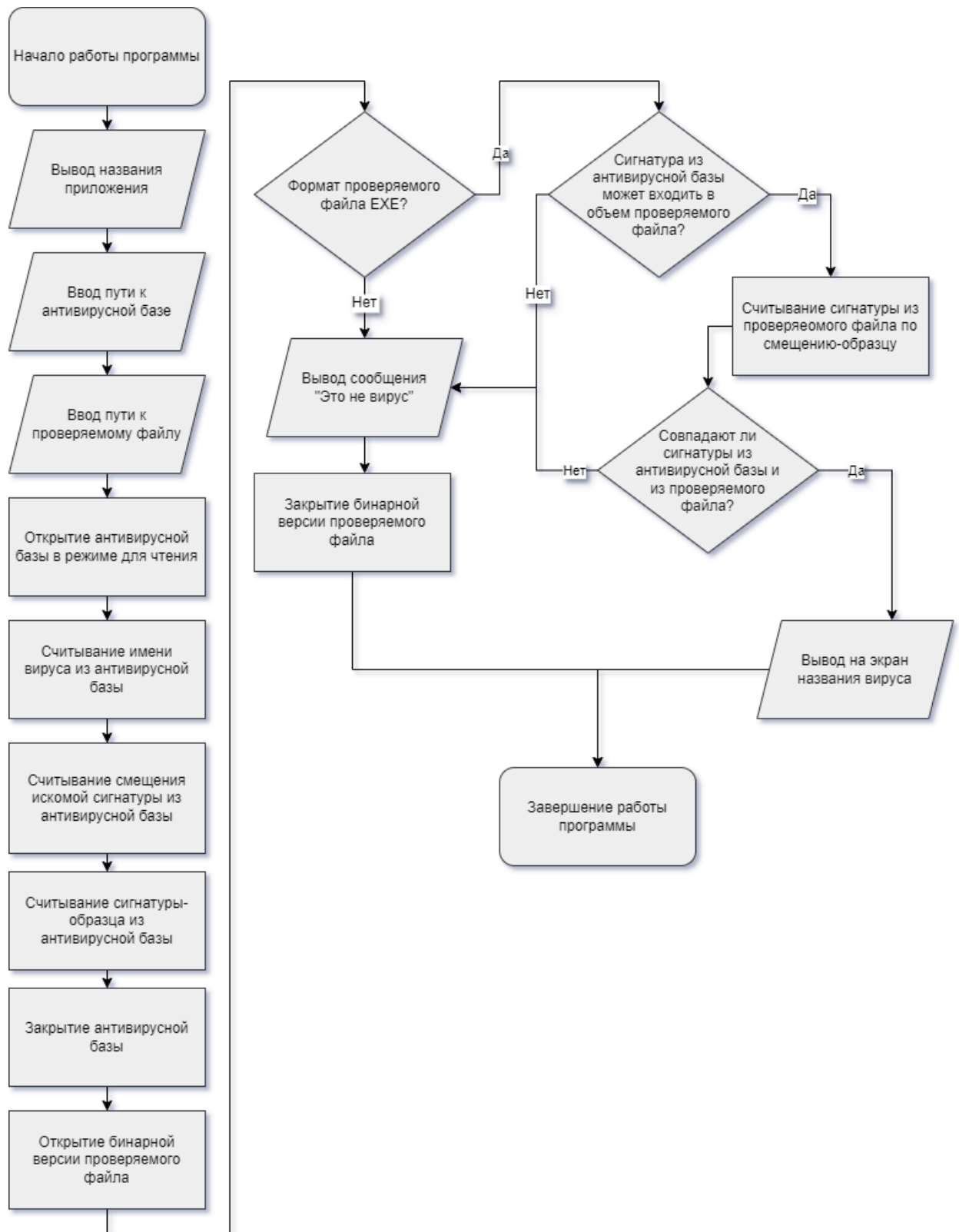
Первая строка: название вируса, не более 49-ти символов, правильная работоспособность программы, в случае наличия в имени алфавитов кроме латинского не гарантируется;

Вторая строка: смещение, указанное в десятичной системе исчисления без пробелов;

Третья строка: сигнатура-образец, указывается 8-символами шестнадцатеричной системы исчисления без пробелов.

Первая и вторая строка должны заканчиваться клавишей «Enter», после третьей строки никаких символов быть не должно.

ПРИЛОЖЕНИЕ 2. БЛОК-СХЕМА АЛГОРИТМА



ПРИЛОЖЕНИЕ 3. ИСХОДНЫЙ КОД ПРОГРАММЫ

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

int path(char* PathFileTxt, char* PathFileExe)
{
    if(PathFileTxt == NULL)
    {
        printf("Pointer ERROR!\n");
        return 1;
    }
    if (PathFileExe == NULL)
    {
        printf("Pointer ERROR!\n");
        return 2;
    }
    if (printf("Enter the path for the \"Antivirus
database\":\n") < 0)
    {
        printf("\nPrintf ERROR!\n");
        return 3;
    }
    if (scanf("%259[^\n]*c", PathFileTxt) != 1)
    {
        printf("\nScanf ERROR!\n");
        return 4;
    }
    if (printf("\nEnter the path for the \"File to be
checked\":\n") < 0)
    {
        printf("\nPrintf ERROR!\n");
        return 5;
    }
    if (scanf("%259[^\n]*c", PathFileExe) != 1)
    {
        printf("\nScanf ERROR!\n");
        return 6;
    }
    if (printf("\n") < 0)
    {
        printf("\nPrintf ERROR!\n");
        return 7;
    }
    return 0;
}

struct VirusInfo
{
    char Name[50];
```

```

        size_t Move;
        unsigned char Sign[8];
    };

    int main()
    {
        FILE *FilenameExe;
        FILE *FilenameTxt;
        unsigned char Signature[8];
        char PathFileTxt[MAX_PATH];
        char PathFileExe[MAX_PATH];
        unsigned char MZ[2];
        long int TMP = 0;
        size_t i = 0;
        struct VirusInfo Virus;

        if(printf("\nSignature scanner\ " App (by Minko
Dmitriy)\n\n") < 0)
        {
            printf("\nPrintf ERROR!\n");
            return 1;
        }
        if(path(PathFileTxt, PathFileExe) != 0)
        {
            printf("Input file paths ERROR!\n");
            return 2;
        }

        FilenameTxt = fopen(PathFileTxt, "r");
        if (FilenameTxt == NULL)
        {
            printf("ERROR opening the \"Antivirus database\" for
reading!\n");
            return 3;
        }
        if(fscanf(FilenameTxt, "%49[^\n]", Virus.Name) != 1)
        {
            printf("Virus name reading ERROR!\n");
            fclose(FilenameTxt);
            return 4;
        }
        if(fscanf(FilenameTxt, "%zu", &Virus.Move) != 1)
        {
            printf("Virus signature offset reading ERROR!\n");
            fclose(FilenameTxt);
            return 5;
        }
        for (i = 0; i <
(sizeof(Virus.Sign)/sizeof(Virus.Sign[0])); i++)
        {
            if (fscanf(FilenameTxt,"%02hhx", &Virus.Sign[i]) !=
1)
            {

```

```

        printf("ERROR reading signature from \"Antivirus
database\\!");
        fclose(FilenameTxt);
        return 6;
    }
}
if (fclose(FilenameTxt) != 0)
{
    printf("ERROR closing the \"Antivirus
database\\!\\n");
    return 7;
}
FilenameExe = fopen(PathFileExe, "rb");
if (FilenameExe == NULL)
{
    printf("ERROR opening the \"File to be checked\" for
reading!\\n");
    return 8;
}

if (fread(MZ, sizeof(MZ)/sizeof(MZ[0]), sizeof(MZ[0]),
FilenameExe) != 1)
{
    printf("ERROR reading signature from binary
file!\\n");
    fclose(FilenameExe);
    return 9;
}
if ((MZ[0] != 'M') || (MZ[1] != 'Z'))
{
    if (printf("It's not a virus.\\n") < 0)
    {
        printf("\\nPrintf ERROR!\\n");
        fclose(FilenameExe);
        return 10;
    }
    if (fclose(FilenameExe) != 0)
    {
        printf("ERROR closing the \"File to be
checked\\!\\n");
        return 11;
    };
    return 0;
}
if (fseek(FilenameExe, 0, SEEK_END) != 0)
{
    printf("ERROR in fseek function!\\n");
    fclose(FilenameExe);
    return 12;
}
TMP = ftell(FilenameExe);
if (TMP == -1)
{

```

```

        printf("ERROR obtaining the size of the \"File to be
checked\\\"!\n");
        fclose(FilenameExe);
        return 13;
    }
    if (TMP < (Virus.Move +
(sizeof(Virus.Sign)/sizeof(Virus.Sign[0]) - 1)))
    {
        if (printf("It's not a virus.\n") < 0)
        {
            printf("\nPrintf ERROR!\n");
            fclose(FilenameExe);
            return 14;
        }
        if(fclose(FilenameExe) != 0)
        {
            printf("ERROR closing the \"File to be
checked\\\"!\n");
            return 15;
        };
        return 0;
    }
    if (fseek(FilenameExe, Virus.Move, SEEK_SET) != 0)
    {
        printf("ERROR in fseek function!\n");
        fclose(FilenameExe);
        return 16;
    }
    if(fread(Signature,
sizeof(Signature)/sizeof(Signature[0]), sizeof(Signature[0]),
FilenameExe) != 1)
    {
        printf("ERROR reading the signature from the binary
file!\n");
        fclose(FilenameExe);
        return 17;
    }
    TMP = 0;

    for (i = 0; i <
(sizeof(Signature)/sizeof(Signature[0])); i++)
    {
        if(Signature[i] != Virus.Sign[i])
        {
            if (printf("It's not a virus.\n") < 0)
            {
                printf("\nPrintf ERROR!\n");
                return 18;
            }
            if (fclose(FilenameExe) != 0)
            {
                printf("ERROR closing the \"File to be
checked\\\" file!\n");

```

```

        return 19;
    }
    return 0;
}
TMP++;
}
if(TMP == 8)
{
    if (printf("This is a virus called \"%s\"\n",
Virus.Name) < 0)
    {
        printf("Printf ERROR!\n");
        return 20;
    }
}
if (fclose(FilenameExe) != 0)
{
    printf("ERROR closing the \"File to be checked\"
file!\n");
    return 21;
}
return 0;
}

```