**Федеральное агентство связи**

**Ордена Трудового Красного Знамени**

**Федеральное государственное бюджетное образовательное учреждение**

**Высшего образования**

**«Московский технический университет связи и информатики»**

Кафедра Информационная безопасность

**Отчет по проекту**

По дисциплине «Защита информации от вредоносного программного обеспечения»

Выполнил: студент группы БВТ1801

Митрохин Ярослав Игоревич

Проверил:

Барков Вячеслав Валерьевич

Москва 2021

Оглавление

[ВВЕДЕНИЕ 3](#_Toc68120129)

[Сервис 4](#_Toc68120130)

[Описание методов 4](#_Toc68120131)

[Скриншоты 6](#_Toc68120132)

[Клиент 7](#_Toc68120133)

[Описание методов 11](#_Toc68120134)

[Библиотека классов 12](#_Toc68120135)

[Методы и описание классов 12](#_Toc68120136)

[Базы данных 13](#_Toc68120137)

[Скриншоты работы программы 16](#_Toc68120138)

[Выводы 18](#_Toc68120139)

[Код программы 19](#_Toc68120140)

[Приложение 1 19](#_Toc68120141)

[Приложение 2 19](#_Toc68120142)

[Приложение 3 19](#_Toc68120143)

# ВВЕДЕНИЕ

Задача данного проекта – разработать комплекса антивирусной защиты. В моем случае проект антивируса состоит из трех разделов:

* AntiService – Windows сервис для обработки большинства запросов пользователя;
* AntiClient2 – проект Windows Forms, который содержит основной UI для упрощения управления программой;
* AntiLibrary – библиотека классов, содержащая все основные методы необходимые для работы антивируса;

Функционал проекта должен содержать:

* Запустить сканирование папки
* Запустить сканирование файла
* Остановить сканирование
* Запустить мониторинг директории
* Остановить мониторинг директории.
* Запланировать проверку по расписанию

Для реализации данного проекта был выбран язык – C#, а в качестве программного интерфейса для обеспечения обмена данными между процессами были выбраны сокеты. При разработке приложения, по возможности, компоненты были разбиты на модули, общий функционал был внесен в общие библиотеки и большинство компонентов были построены с помощью архитектурных паттернов. Далее будут описаны как проекты, из которых состоит программа, так и сами методы.

# Сервис

Проект сервиса разрабатывался на основе консольного приложения Windows, в последствии был изменен на Windows сервис. Сервис начинает работу непосредственно после начала запуска. Каждые две секунды происходит проверка соединения, если клиент пытается подключиться к сервису, то сервис образует связь между двумя приложениями. После присоединения клиента к сервису, пользователь вводит\выбирает путь файла\папки, которую нужно проверить. Сервер, в свою очередь, обрабатывает входные данные и проводит определенные действия, зависящие от поступившей информации. Также сервис использует методы, которые были реализованы в библиотеке классов.

В сервисе реализованы такие метода, как:

* MainClass
* SwitchedC
* FilesPE\FilesPE2
* MonitoringThread
* StartScan\StartScan2
* StartScanFile
* OnTimedEvent

## Описание методов

Подробнее о каждом:

*MainClass* – основной метод в сервисе, который отвечает за соединения процессов, выбора выполнения задач и отловление ошибок. Также в данном методе происходит выгрузка проверенных файлов в базу данных (см. главу о БД). Помимо этого, метод *MainClass* запускает таймер для выполнения сканирования по расписанию и, по сути, является исполняемым.

*SwitchedC* – метод для разделения сканирования по расписанию и сканирование папки, а также для выгрузки путей для мониторинга. Данный метод является разделением других методов и разработан для удобства чтения кода и упрощения работы с ним.

*FilesPE\FilesPE2* – методы для определения PE файлов. Можно считать, что данный метод один из самых важных, потому что без него невозможно сканирование вообще. Помимо определения PE файлов, данный метод отделяет zip архивы и просматривает их содержимое. Эти методы используются как в сканировании по расписанию, так и в сканирование папки.

*MonitoringThread* – из названия понятно, что данный метод существует для мониторинга выбранной папки. Основная часть метода – заимствованные методы из библиотеки классов. В нем происходит обнаружение путей для мониторинга и непосредственный мониторинг. Также при изменении файлов в директории происходит мгновенная проверка измененных файлов и отлавливание вирусов.

*StartScan\StartScan2 ­*– метод для сканирования файлов для обнаружения вирусов. Как и методы FilesPE\FilesPE2 основная часть метода – заимствованные методы из библиотеки классов. Также как и в остальных методах сканирования происходит проверка на вирусы, после нахождения файла с определенной сигнатурой происходит помещение вируса в карантин.

*StartScanFile* – сканирование отдельного файла, путь к которому нужно прописать вручную. Также происходит проверка на PE файл и на вирусы. Каждая проверка в любом методе заканчивается выгрузкой проверяемых файлов в базу данных.

*OnTimedEvent* – метод для сканирования по расписанию. Представляет собой ожидающий таймер, который выполняет метод по достижению определенного времени. Сканирование также происходит внутри данного метода.

Сервис тесно связан с базами данных (см. главу о БД). Все просканированные файлы, вирусы, сигнатуры для поисков вирусов, время для сканирования по расписанию и отчетность записывается в базы данных именно из сервиса. Однако большинство методов для действий, связанных с базами данных используют методы, реализованные в библиотеке классов.

Одна из основных функций сервиса – соединение с клиентом. Как было сказано в начале в качестве интерфейса для реализации связи между процессами были выбраны сокеты. Сервис со своей стороны назначает удаленную точку – имя и порт, по которому будет подключаться клиент.

## Скриншоты



Рисунок 1 - Остановленная служба



Рисунок 2 - Работающая служба

Код службы см. в приложении 1.

# Клиент

Клиент – это сперва UI для удобного управления программой пользователем, а уже потом средство для отправки данных на сервис. Также помимо вышеперечисленных функций клиент может обновлять и очищать базы данных, удалять и добавлять каталоги для мониторинга и удалять файлы из карантина или удалять их вообще из системы. В разработанном клиенте есть несколько вкладок:

* Сканирование по расписанию
* Сканирование файла
* Мониторинг
* Сканирование папки
* Просканированные файлы
* Просканированные файлы
* Вирусы
* Отчетность

Каждая вкладка несет в себе смысловую нагрузку и отвечает за отдельный функционал:

*Сканирование по расписанию*

Вид вкладки:

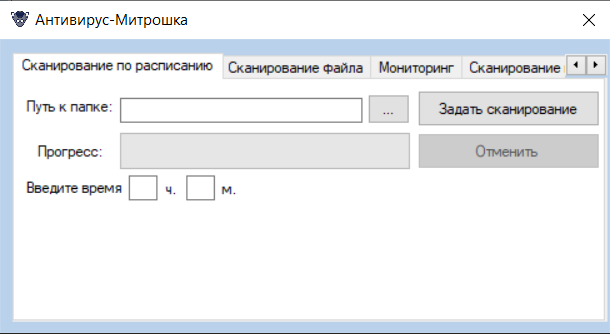
**

Рисунок 3 - Вид вкладки сканирование по расписанию

На данной вкладке пользователь может выбрать путь каталога, который он хочет проверить, задать время в часах и минутах сегодняшнего дня и нажать кнопку «Задать сканирование». После данных манипуляций в назначенной время произойдет сканирование указанного каталога.

*Сканирование файла*

Вид вкладки:

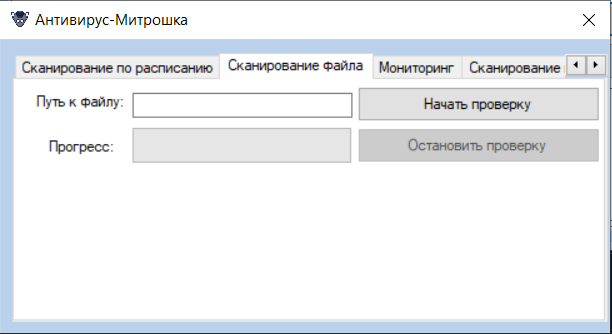
**

Рисунок 4 - Вид вкладки сканирование файла

На данной вкладке пользователь вручную вводит путь к файлу и нажимает кнопку «Начать сканирование». Прогресс сканирование показывает progressBar. После окончания сканирования появляется надпись с последующими действиями.

*Мониторинг*

Вид вкладки:

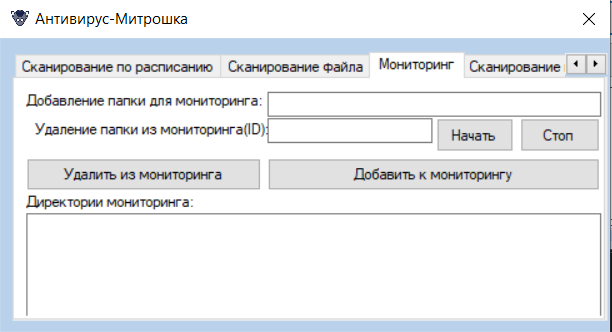
**

Рисунок 5 - Вид вкладки мониторинг

На данной вкладке пользователь может начать и остановить мониторинг соответствующими кнопками. После остановки мониторинга происходит выгрузка проверенных файлов в базу данных. Также здесь можно удалить и добавить директории для мониторинга.

*Сканирование папки*

Вид вкладки:

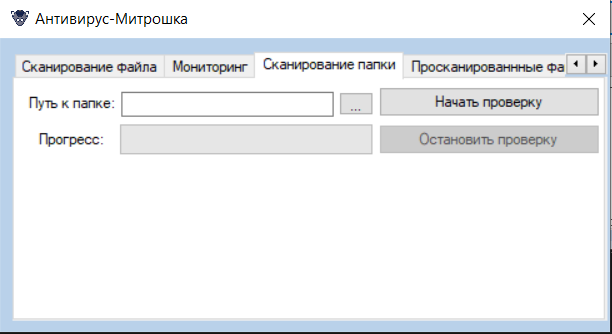
**

Рисунок 6 - Вид вкладки сканирование папки

На данной вкладке пользователь может выбрать путь до каталога и нажать кнопку «начать проверку».

*Просканированные файлы*

Вид вкладки:

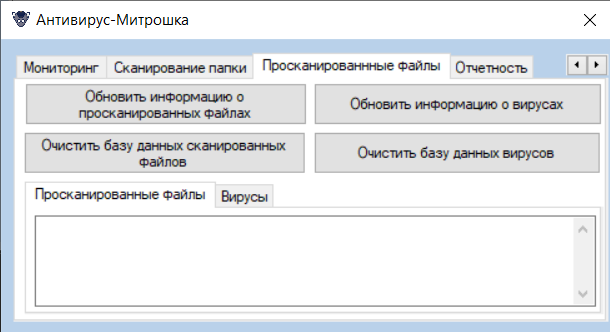


Рисунок 7 - Вид вкладки просканированные файлы

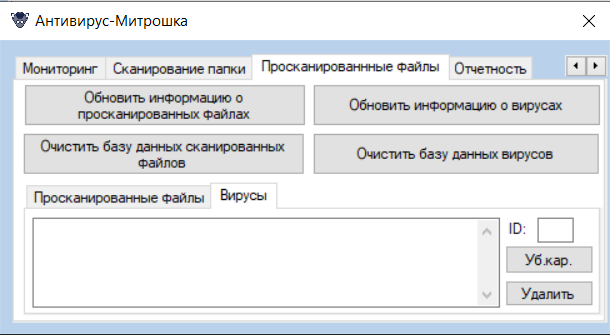


Рисунок 8 - Вид вкладки вирусы

На данных вкладках пользователь можно увидеть просканированные файлы и вирусы, которые были найдены в процессе сканирования. Также пользователь может по желанию убрать файл из карантина, указав id вируса.

*Отчетность*

Вид вкладки:

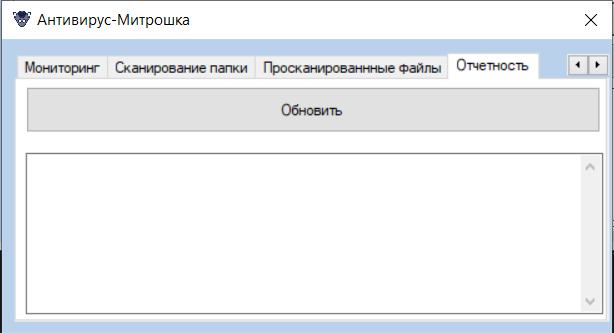
**

Рисунок 9 - Вид вкладки отчетность

На данной вкладке пользователь может увидеть все отчеты по сканированию: дату и время, тип сканирования, кол-во найденных вирусов.

## Описание методов

В клиенте реализованы методы, которые были заимствованы из библиотеки классов. Но также есть такие методы как:

* ScunCatalog
* ScunCatalog2
* ScunMonitoring
* SetProgressbarValue

Данные методы отвечают за отправление входных данных, которые будут использованы сервисом. Среди таких данных: код метода, который будет выполнен, и путь каталога, либо файла. Также в них реализованы визуальные отображения на UI. Что касается остальных методов, в основном они используются для изменения, добавления, удаления данных из баз данных.

Код клиента см. в приложении 2.

# Библиотека классов

В процессе реализации программы была реализована библиотека классов, которая содержит в себе четыре класса:

* PEFILE – класс для определения PE файлов и zip разархивации
* MonitoringO – класс, отвечающий за реализацию действий мониторинга
* DataBaseAction – самый большой класс, который содержит в себе почти все методы для работы с базами данных
* Scaner – класс, отвечающий непосредственно за сканирование файла

Библиотека классов используется для обоих приложений: как для сервиса, так и для клиента. Что упростило работу и реализацию обоих приложений. Теперь конкретнее о каждом классе.

### Методы и описание классов

*PEFILE*

Данный класс содержит методы для чтения байтового содержимого файла, рассмотрение определенной секции байтового файла и разархивация файла. Это один из двух основных классов, необходимых для сканирвания.

*MonitoringO*

Класс, реализованный для мониторинга определенных директорий. Содержит в себе основу для реализации функции мониторинга директории.В нем указаны параметры мониторинга и выводимые значения при вызове определенных событий.

*DataBaseAction*

Класс, содержащий всю реализацию взаимодействий с базами данных. Почти весь клиент наследует методы из этого класса, что показывает его важность. В нем формируются отчеты, даты для сканирования, удаление таблиц и данных из них и т.д.

*Scaner*

Второй из двух важнейших классов для сканирования. В нем реализованы методы для просмотра байтовых фалов и их определенных секций. Также отправление в карантин и обратно также реализовано в этом классе. Работа с базами данных также не обошла стороной этот класс, при нахождении вируса происходит сравнение частей байтового кода с сигнатурой, находящейся в базе данных.

Код библиотеки классов см. приложение 3.

# Базы данных

В качестве средства для хранения данных была выбрана программа sqlite. С помощью нее были созданы следующие таблицы:

* BaseText
* DataAndTime
* FilesPE
* Report
* Signature
* Viruses
* TableofScaner

В каждой базе данных содержится информация, которая используется сервисом или клиентом. При любой выбранной функции антивируса будет выполнена работа с базой данных.

*BaseText*



Рисунок 10 - Поля таблицы

В данной таблице хранятся пути к каталогам мониторинга вместе с их идентификаторами. Поле ID было создано для удобного удаления путей мониторинга пользователем.

*DataAndTime*



Рисунок 11 - Поля таблицы

В данной таблице хранится время для сканирования по расписанию. Каждый раз, когда пользователь нажимает на кнопку старта сканирования, часы и минуты добавляются в базу данных, из которой сервис забирает каждую минуту данные для старта сканирования.

*FilesPE*



Рисунок 12 - Поля таблицы

В данной таблице хранятся все проверенные PE файлы. После любого сканирования в базу данных добавляются пути до исполняемых файлов. Также в клиенте есть функция для очистки данной базы данных.

*Report*



Рисунок 13 - Поля таблицы

В данной таблице несколько полей: ID, time, typeofscan, countofviruses. По названиям понятно, что в таблице сохраняются отчеты по сканированию. После каждого сканирования в базе данных появляется запись, состоящая из времени и даты сканирования, типа сканирования и кол-ва найденных вирусов.

*Signature*



Рисунок 14 - Поля таблицы

В данной таблице два поля: идентификатор и сигнатура. Сигнатура нужна для поиска вирусов в PE файлов и при сканировании происходит обращение к базе данных для того, чтобы найти соответствие в байтовом файле и сигнатуре.

*Viruses*



Рисунок 15 - Поля таблицы

В данной таблице хранятся пути до вирусных файлов. В свою очередь пользователь может удалить файл из базы и из системы, а также убрать из карантина. Данная таблица хранит в себе все вирусы, найдены за все время.

*Tableofscaner*



Рисунок 16 - Поля таблицы

В данной таблице хранятся проверенные файлы после сканирования любым способом.

Пользователь может активно изменять данные базы данных: удалять данные из таблицы, обращаться к файлам через базу данных, удалять файлы через базу данных и т.д.

# Скриншоты работы программы

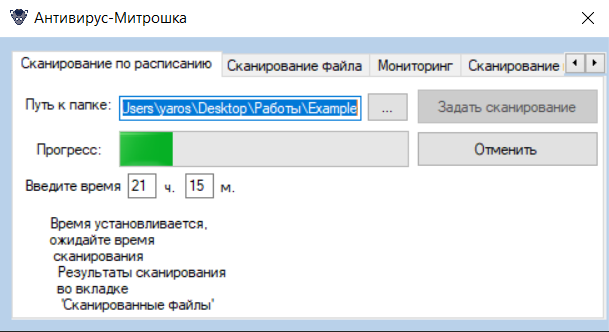


Рисунок 17 - Скриншот работы сканирования по расписанию

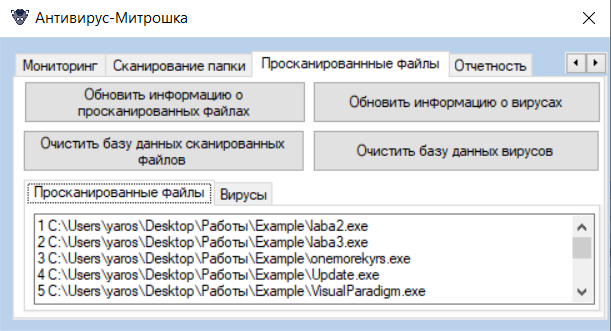


Рисунок 18 - Скриншот информации после сканирования

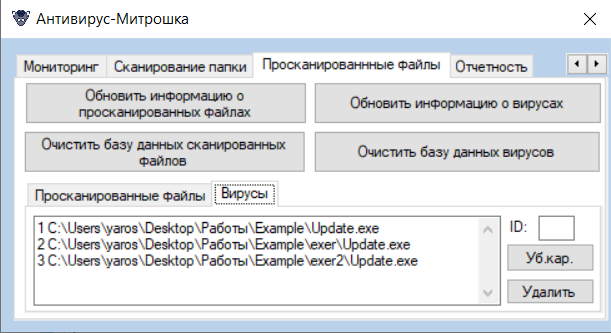


Рисунок 19 - Найденные вирусы

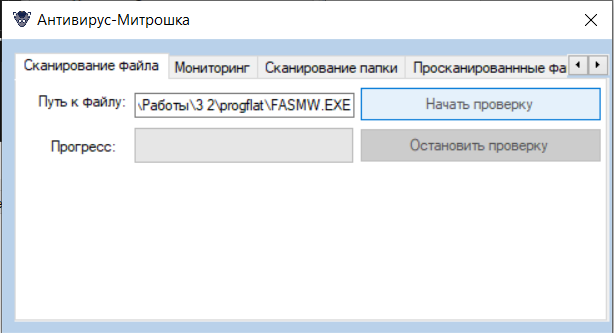


Рисунок 20 - Скриншот работы сканирования файла

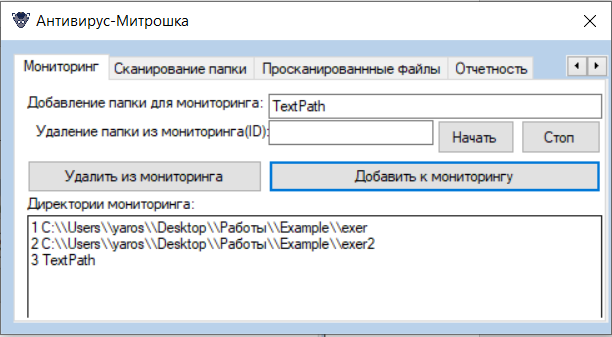


Рисунок 21 - Добавление пути для мониторинга

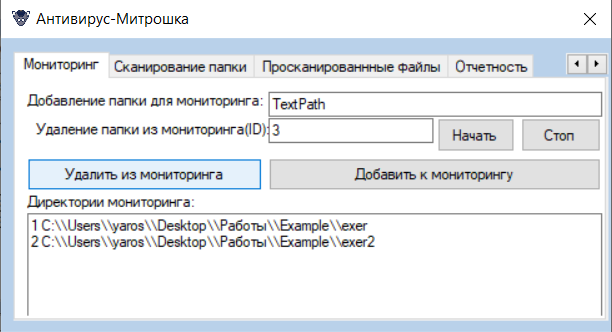


Рисунок 22 - Удаление пути из мониторинга

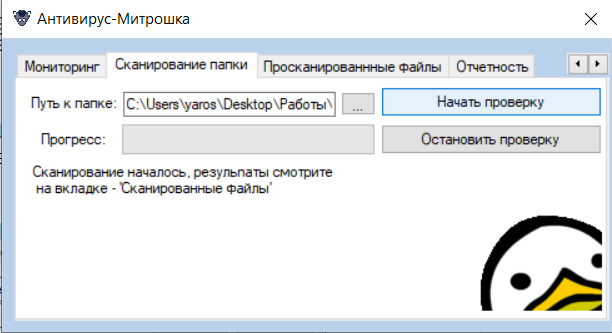


Рисунок 23 - Скриншот сканирования папки

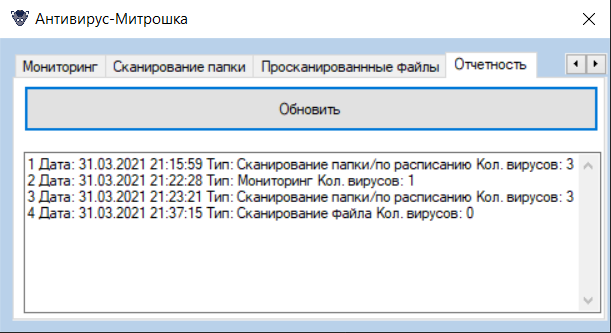


Рисунок 24 - Отчетность

# Выводы

Комплекс антивирусной защиты включает следующие компоненты: антивирусные базы, сервис, осуществляющий проверку заданных объектов, клиентское приложение для управления сервисом и получения результатов (UI), приложение для редактирования содержимого антивирусных баз. Общий функционал вынесен в отдельные библиотеки и использоваться компонентами комплекса совместно. Модули спроектированы с применением принципов дизайна классов и принципов организации компонентов.

# Код программы

## Приложение 1

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Diagnostics;

using System.Linq;

using System.ServiceProcess;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Net.Sockets;

using System.Runtime.InteropServices;

using Microsoft.Data.Sqlite;

using System.IO;

using System.Threading;

namespace AntiService

{

public partial class Service1 : ServiceBase

{

public Service1()

{

InitializeComponent();

}

protected override void OnStart(string[] args)

{

Logger.MainClass();

}

protected override void OnStop()

{

}

}

public static class Logger

{

//example

static List<string> list = new List<string>();

static List<string> list2 = new List<string>();

static string resultofmonitoring = "";

static StreamWriter streamWriter;

static StreamReader streamReader;

static string pathKatalog = "";

static bool flagMon = false;

static string pathforrasp = "";

static bool raspflag = true;

static string extactpath = Application.StartupPath + @"\Example2";

//static string strangepath = @"C:\Users\yaros\Desktop\txt.txt";

//static string createText = "";

private static System.Timers.Timer aTimer = new System.Timers.Timer();

public static void MainClass()

{

TcpClient socketForServer;

try

{

var whileThread = new Thread(() =>

{

while (true)

{

try

{

try

{

//Console.WriteLine("Ожидание подключение...");

Thread.Sleep(2000);

socketForServer = new TcpClient("localHost", 10);

}

catch

{

continue;

}

NetworkStream networkStream = socketForServer.GetStream();

streamWriter = new StreamWriter(networkStream);

streamReader = new StreamReader(networkStream);

//Console.WriteLine("\r\n\*Клиент подключенный по порту №10\*");

//Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

//Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

string code = "";

string strvod = streamReader.ReadLine();

if (strvod == "true")

flagMon = true;

else if (strvod == "true1")

{

aTimer.Stop();

}

else if (strvod == "truee")

{

raspflag = false;

}

else

{

var text = strvod.Split(' ');

pathKatalog = text[0];

code = text[1];

}

if (code == "000")

{

string paths = SwitchedC(code);

AntiLibrary.DataBaseAction.ConverttoBase(paths);

paths = "";

DeleteZip();

networkStream.Close();

}

else if (code == "004")

{

SwitchedC(code);

Thread threadMon = new Thread(MonitoringThread);

threadMon.Start();

networkStream.Close();

}

else if (code == "002")

{

string resScun = StartScanFile(pathKatalog);

AntiLibrary.DataBaseAction.ConverttoBase(resScun);

resScun = "";

DeleteZip();

networkStream.Close();

}

else if (code == "003")

{

pathforrasp = pathKatalog;

aTimer.Interval = 60000;

aTimer.Elapsed += OnTimedEvent;

aTimer.AutoReset = true;

aTimer.Enabled = true;

networkStream.Close();

}

else if (code == "005")

{

string flagmonitor = streamReader.ReadLine();

if (flagmonitor != "")

flagMon = true;

}

Thread.Sleep(500);

//Console.WriteLine("Соединение разорвано\r\n");

}

catch

{

Thread.Sleep(100);

//Console.WriteLine("Сканирование завершено");

}

}

});

whileThread.Start();

}

catch (Exception e)

{

}

}

static string SwitchedC(string str)

{

string result = "";

switch (str)

{

case "000":

result = StartScan();

break;

case "001":

result = StartScan2();

break;

case "004":

SqliteConnection sqlite = new SqliteConnection("Data Source=C:\\BaseData\\BasDate.db");

sqlite.Open();

var command = sqlite.CreateCommand();

command.CommandText = @"SELECT \* FROM BaseText";

using (var reader = command.ExecuteReader())

{

for (int i = 0; reader.Read(); i++)

{

list.Add(reader.GetString(1));

}

}

break;

}

return result;

}

static string FilesPE()

{

string res = "";

string[] stri1 = Directory.GetFiles(pathKatalog, "\*.\*", SearchOption.AllDirectories);

foreach (string paths in stri1)

{

if (paths.Substring(paths.LastIndexOf(".") + 1) == "zip")

{

AntiLibrary.PEFILE.ZipUnPacked(paths, extactpath);

list2.Add(paths);

}

if (AntiLibrary.PEFILE.IsPEFile(paths) == true)

{

res += paths + " ";

}

}

string[] stri = Directory.GetFiles(Application.StartupPath + @"\Example2", "\*.\*", SearchOption.AllDirectories);

foreach (string paths1 in stri)

{

if (paths1.Substring(paths1.LastIndexOf(".") + 1) == "zip")

{

AntiLibrary.PEFILE.ZipUnPacked(paths1, extactpath);

list2.Add(paths1);

}

}

string[] stri2 = Directory.GetFiles(Application.StartupPath + @"\Example2", "\*.\*", SearchOption.AllDirectories);

foreach (string paths2 in stri2)

{

if (AntiLibrary.PEFILE.IsPEFile(paths2) == true)

res += paths2 + " ";

}

return res;

}

static string FilesPE2()

{

string res = "";

string[] stri1 = Directory.GetFiles(pathforrasp, "\*.\*", SearchOption.AllDirectories);

foreach (string paths in stri1)

{

Console.WriteLine(paths);

if (paths.Substring(paths.LastIndexOf(".") + 1) == "zip")

{

AntiLibrary.PEFILE.ZipUnPacked(paths, extactpath);

list2.Add(paths);

}

if (AntiLibrary.PEFILE.IsPEFile(paths) == true)

res += paths + " ";

}

string[] stri = Directory.GetFiles(Application.StartupPath + @"\Example2", "\*.\*", SearchOption.AllDirectories);

foreach (string paths1 in stri)

{

if (paths1.Substring(paths1.LastIndexOf(".") + 1) == "zip")

{

AntiLibrary.PEFILE.ZipUnPacked(paths1, extactpath);

list2.Add(paths1);

}

}

string[] stri2 = Directory.GetFiles(Application.StartupPath + @"\Example2", "\*.\*", SearchOption.AllDirectories);

foreach (string paths2 in stri2)

{

if (AntiLibrary.PEFILE.IsPEFile(paths2) == true)

res += paths2 + " ";

}

return res;

}

static void DeleteZip()

{

string[] stri2 = Directory.GetFiles(Application.StartupPath + @"\Example2", "\*.\*", SearchOption.AllDirectories);

foreach (string paths2 in stri2)

{

System.IO.File.Delete(paths2);

}

}

static void MonitoringThread()

{

flagMon = false;

string type = "Мониторинг";

int j = 1;

int i = 0;

while (flagMon == false)

{

do

{

for (int k = 0; k < list.Count; k++)

{

AntiLibrary.MonitoringO.MonitoringON(list[k]);

resultofmonitoring = AntiLibrary.MonitoringO.monitor;

//Console.WriteLine(resultofmonitoring);

if (resultofmonitoring != "")

{

AntiLibrary.DataBaseAction.ConverttoBase(resultofmonitoring);

if (AntiLibrary.Scaner.ScanerFile(AntiLibrary.Scaner.GetFileCode(resultofmonitoring, ref i)) == true)

{

AntiLibrary.DataBaseAction.AddToVirusBase(resultofmonitoring);

bool quar = AntiLibrary.Scaner.MoveToQuarantine(resultofmonitoring, false);

//if (quar == true)

// Console.WriteLine("File moved to quarantine");

//else

// Console.WriteLine("Error");

}

}

}

} while (resultofmonitoring == "");

}

AntiLibrary.DataBaseAction.CreateReport(type, j);

}

static string StartScan()

{

raspflag = true;

string type = "Сканирование папки/по расписанию";

int j = 0;

int i = 0;

string resultofPE = FilesPE();//Все пути PE файлов

var textofPE = resultofPE.Split(' ');

foreach (string str in textofPE)

{

if (str != "")

{

if (AntiLibrary.Scaner.ScanerFile(AntiLibrary.Scaner.GetFileCode(str, ref i)) == true)

{

j++;

AntiLibrary.DataBaseAction.AddToVirusBase(str);

bool quar = AntiLibrary.Scaner.MoveToQuarantine(str, false);

}

}

if (!raspflag)

break;

}

AntiLibrary.DataBaseAction.CreateReport(type, j);

return resultofPE;

}

static string StartScanFile(string path)

{

string type = "Сканирование файла";

int j = 0;

int i = 0;

if (AntiLibrary.PEFILE.IsPEFile(path) == true)

{

if (AntiLibrary.Scaner.ScanerFile(AntiLibrary.Scaner.GetFileCode(path, ref i)) == true)

{

//Console.WriteLine("Virus founded");

j++;

AntiLibrary.DataBaseAction.AddToVirusBase(path);

bool quar = AntiLibrary.Scaner.MoveToQuarantine(path, false);

AntiLibrary.DataBaseAction.CreateReport(type, j);

return path;

}

else

{

AntiLibrary.DataBaseAction.CreateReport(type, j);

return path;

}

}

AntiLibrary.DataBaseAction.CreateReport(type, j);

return "";

}

static string StartScan2()

{

string type = "Сканирование папки/по расписанию";

int j = 0;

int i = 0;

string resultofPE = FilesPE2();//Все пути PE файлов

//Console.WriteLine("All files: " + resultofPE);

var textofPE = resultofPE.Split(' ');

foreach (string str in textofPE)

{

if (str != "")

{

if (AntiLibrary.Scaner.ScanerFile(AntiLibrary.Scaner.GetFileCode(str, ref i)) == true)

{

//Console.WriteLine("Virus founded: " + str);

j++;

AntiLibrary.DataBaseAction.AddToVirusBase(str);

bool quar = AntiLibrary.Scaner.MoveToQuarantine(str, false);

}

}

}

AntiLibrary.DataBaseAction.CreateReport(type, j);

return resultofPE;

}

private static void OnTimedEvent(Object source, System.Timers.ElapsedEventArgs e)

{

AntiLibrary.DataBaseAction.InfoData();

int hour1 = AntiLibrary.DataBaseAction.hours;

int minute1 = AntiLibrary.DataBaseAction.minutes;

int hour2 = Convert.ToInt32(DateTime.Now.ToString("HH"));

int minute2 = Convert.ToInt32(DateTime.Now.ToString("mm"));

//Console.WriteLine(hour1 + " " + minute1);

//Console.WriteLine(hour2 + " " + minute2);

if (hour1 == hour2 && minute1 == minute2)

{

pathforrasp = SwitchedC("001");

AntiLibrary.DataBaseAction.ConverttoBase(pathforrasp);

pathforrasp = "";

DeleteZip();

AntiLibrary.DataBaseAction.DeleteTableData();

}

}

}

}

## Приложение 2

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Runtime.InteropServices;

using System.Net.Sockets;

using System.Threading;

using System.Threading.Tasks;

using System.Windows.Forms;

using Microsoft.Data.Sqlite;

using AntiLibrary;

using System.IO;

namespace AntiClient2

{

public partial class Form1 : Form

{

static TcpListener tcpListener = new TcpListener(10);

Thread thread;

Thread thread2;

System.Threading.Timer timer = null;

Thread thread3;

static bool flagrasp;

static List<string> list3 = new List<string>();

public Form1()

{

InitializeComponent();

}

private void button2\_Click(object sender, EventArgs e)

{

//Console.WriteLine(Application.StartupPath);

pictureBox1.Visible = true;

pictureBox1.Image = Image.FromFile(Application.StartupPath+@"\xxx.gif");

label3.Text = "Сканирование началось, резульnаты смотрите\r\n на вкладке - 'Сканированные файлы'";

button1.Visible = true;

button3.Enabled = true;

//textBox1.Enabled = false;

progressBar1.Value = 0;

thread2 = new Thread(new ThreadStart(ScunCatalog));

thread2.Start();

}

public void ScunCatalog()

{

tcpListener.Start();

Socket socketForClient = tcpListener.AcceptSocket();

Console.WriteLine("\*\*\*\*");

if (socketForClient.Connected)

{

int i = 0;

NetworkStream networkStream = new NetworkStream(socketForClient);

StreamWriter streamWriter = new StreamWriter(networkStream);

StreamReader streamReader = new StreamReader(networkStream);

string pathtokat = textBox1.Text;

string str = pathtokat + " 000";//Передача кода функции и пути к каталогу

streamWriter.WriteLine(str);

streamWriter.Flush();

while (i != 1)

{

i++;

progressBar1.Minimum = 0;

progressBar1.Maximum = 100;

int value = 0;

for (int j = 0; j < 100; j++)

{

value++;

SetProgressBarValue(progressBar1, value);

Thread.Sleep(100);

}

SetProgressBarValue(progressBar1, 100);

}

streamReader.Close();

networkStream.Close();

streamWriter.Close();

socketForClient.Close();

}

tcpListener.Stop();

pictureBox1.Invoke((ThreadStart)delegate ()

{

pictureBox1.Visible = false;

});

}

public void ScunCatalog2()

{

tcpListener.Start();

Socket socketForClient = tcpListener.AcceptSocket();

if (socketForClient.Connected)

{

int i = 0;

NetworkStream networkStream = new NetworkStream(socketForClient);

StreamWriter streamWriter = new StreamWriter(networkStream);

StreamReader streamReader = new StreamReader(networkStream);

string pathtokat = textBox7.Text;

string str = pathtokat + " 003";//Передача кода функции и пути к каталогу

streamWriter.WriteLine(str);

streamWriter.Flush();

while (i != 1)

{

i++;

progressBar3.Minimum = 0;

progressBar3.Maximum = 100;

int value = 0;

for (int j = 0; j < 100; j++)

{

value++;

SetProgressBarValue(progressBar3, value);

Thread.Sleep(100);

}

SetProgressBarValue(progressBar3, 100);

}

streamReader.Close();

networkStream.Close();

streamWriter.Close();

socketForClient.Close();

}

button9.Invoke((ThreadStart)delegate ()

{

button9.Enabled = true;

});

button10.Invoke((ThreadStart)delegate ()

{

button10.Enabled = false;

});

tcpListener.Stop();

pictureBox3.Invoke((ThreadStart)delegate ()

{

pictureBox3.Visible = true;

pictureBox3.Image = Image.FromFile(Application.StartupPath + @"\rrr.gif");

});

Thread.Sleep(2000);

pictureBox3.Invoke((ThreadStart)delegate ()

{

pictureBox3.Visible = false;

});

}

public void ScunMonitoring()

{

tcpListener.Start();

Socket socketForClient = tcpListener.AcceptSocket();

if (socketForClient.Connected)

{

NetworkStream networkStream = new NetworkStream(socketForClient);

StreamWriter streamWriter = new StreamWriter(networkStream);

StreamReader streamReader = new StreamReader(networkStream);

string str = "C:\\Users\\yaros 004";//Передача кода функции и пути к каталогу

streamWriter.WriteLine(str);

streamWriter.Flush();

streamReader.Close();

networkStream.Close();

streamWriter.Close();

socketForClient.Close();

}

tcpListener.Stop();

}

private void button1\_Click(object sender, EventArgs e)

{

// показать диалог выбора папки

DialogResult result = folderBrowserDialog1.ShowDialog();

// если папка выбрана и нажата клавиша `OK` - значит можно получить путь к папке

if (result == DialogResult.OK)

{

// запишем в нашу переменную путь к папке

textBox1.Text = folderBrowserDialog1.SelectedPath;

}

}

private void button4\_Click(object sender, EventArgs e)

{

label6.Text = "Мониторинг начался. Для результатов просмотрите вкладку - 'Просканированные файлы'";

thread = new Thread(new ThreadStart(ScunMonitoring));

thread.Start();

}

private void button6\_Click(object sender, EventArgs e)

{

AntiLibrary.DataBaseAction.InTableMonitor(textBox3, textBox5);

}

private void button5\_Click(object sender, EventArgs e)

{

AntiLibrary.DataBaseAction.OutTableMonitor(textBox4, textBox5);

}

private void button7\_Click(object sender, EventArgs e)

{

//pictureBox2.Visible = true;

//pictureBox2.Image = Image.FromFile(@"C:\Users\yaros\Desktop\rrr.gif");

button7.Enabled = false;

label16.Text = "Сканирование началось";

progressBar2.Value = 0;

progressBar2.Maximum = 10;

progressBar2.Minimum = 0;

progressBar2.Step = 1;

string theString;

tcpListener.Start();

Socket socketForClient = tcpListener.AcceptSocket();

if (socketForClient.Connected)

{

int i = 0;

NetworkStream networkStream = new NetworkStream(socketForClient);

StreamWriter streamWriter = new StreamWriter(networkStream);

StreamReader streamReader = new StreamReader(networkStream);

theString = textBox6.Text;

string str = theString + " 002";//Передача кода функции и пути к каталогу

streamWriter.WriteLine(str);

streamWriter.Flush();

for (i = 0; i < 10; i++)

{

Thread.Sleep(50);

progressBar2.PerformStep();

}

streamReader.Close();

networkStream.Close();

streamWriter.Close();

}

socketForClient.Close();

tcpListener.Stop();

button7.Enabled = true;

label16.Text = "Сканирование завершено. Результаты смотрите во вкладке \r\n 'Сканированные файлы'";

//pictureBox2.Visible = false;

}

private void button11\_Click(object sender, EventArgs e)

{

// показать диалог выбора папки

DialogResult result = folderBrowserDialog1.ShowDialog();

// если папка выбрана и нажата клавиша `OK` - значит можно получить путь к папке

if (result == DialogResult.OK)

{

// запишем в нашу переменную путь к папке

textBox7.Text = folderBrowserDialog1.SelectedPath;

}

}

private void button9\_Click(object sender, EventArgs e)

{

flagrasp = true;

button9.Invoke((ThreadStart)delegate ()

{

button9.Enabled = false;

});

button10.Invoke((ThreadStart)delegate ()

{

button10.Enabled = true;

});

progressBar3.Invoke((ThreadStart)delegate ()

{

progressBar3.Value = 0;

});

int hours = Convert.ToInt32(textBox8.Text);

int minutes = Convert.ToInt32(textBox9.Text);

AntiLibrary.DataBaseAction.CreateData(hours, minutes);

label15.Text = "Время установливается, \r\nожидайте время\r\n сканирования\r\n Результаты сканирования\r\n во вкладке\r\n 'Сканированные файлы'";

thread3 = new Thread(new ThreadStart(ScunCatalog2));

thread3.Start();

//ScunRaspisanie();

}

private void SetProgressBarValue(ProgressBar prb, int value)

{

if (this.InvokeRequired)

{

this.Invoke((Action<ProgressBar, int>)SetProgressBarValue, prb, value);

}

else

{

prb.Value = value;

}

}

private void button12\_Click(object sender, EventArgs e)

{

AntiLibrary.DataBaseAction.UpdateTableScanedFiles(textBox11);

}

private void button13\_Click(object sender, EventArgs e)

{

label6.Text = " ";

label6.Text = "Мониторинг завершен";

tcpListener.Start();

Socket socketForClient = tcpListener.AcceptSocket();

if (socketForClient.Connected)

{

NetworkStream networkStream = new NetworkStream(socketForClient);

StreamWriter streamWriter = new StreamWriter(networkStream);

string flag = "true";

streamWriter.WriteLine(flag);

streamWriter.Flush();

networkStream.Close();

streamWriter.Close();

socketForClient.Close();

}

tcpListener.Stop();

}

private void button10\_Click(object sender, EventArgs e)

{

label5.Text = " ";

label5.Text = "Сканирование отменено";

tcpListener.Start();

Socket socketForClient = tcpListener.AcceptSocket();

if (socketForClient.Connected)

{

NetworkStream networkStream = new NetworkStream(socketForClient);

StreamWriter streamWriter = new StreamWriter(networkStream);

string flag = "true1";

streamWriter.WriteLine(flag);

streamWriter.Flush();

networkStream.Close();

streamWriter.Close();

socketForClient.Close();

}

tcpListener.Stop();

}

private void button14\_Click(object sender, EventArgs e)

{

AntiLibrary.DataBaseAction.DeleteTable();

}

private void button3\_Click(object sender, EventArgs e)

{

label3.Text = " ";

label3.Text = "Сканирование завершено";

tcpListener.Start();

Socket socketForClient = tcpListener.AcceptSocket();

if (socketForClient.Connected)

{

NetworkStream networkStream = new NetworkStream(socketForClient);

StreamWriter streamWriter = new StreamWriter(networkStream);

string flag = "truee";

streamWriter.WriteLine(flag);

streamWriter.Flush();

networkStream.Close();

streamWriter.Close();

socketForClient.Close();

}

tcpListener.Stop();

}

private void button15\_Click(object sender, EventArgs e)

{

AntiLibrary.DataBaseAction.UpdateTableViruse(textBox2);

}

private void button16\_Click(object sender, EventArgs e)

{

AntiLibrary.DataBaseAction.DeleteTableVirus();

}

private void button18\_Click(object sender, EventArgs e)

{

AntiLibrary.DataBaseAction.DeleteValueFromViruse(textBox10, textBox2);

}

private void button19\_Click(object sender, EventArgs e)

{

AntiLibrary.DataBaseAction.UpdateReport(textBox12);

}

private void button17\_Click(object sender, EventArgs e)

{

list3.Clear();

int j = Convert.ToInt32(textBox10.Text);

SqliteConnection sqlite = new SqliteConnection("Data Source=C:\\BaseData\\BasDate.db");

sqlite.Open();

var command = sqlite.CreateCommand();

command.CommandText = @"SELECT \* FROM Viruses where id = " + j;

using (var reader = command.ExecuteReader())

{

for (int i = 0; reader.Read(); i++)

{

list3.Add(reader.GetString(1));

}

}

string path = list3[0];

AntiLibrary.Scaner.MoveToQuarantine(path, true);

command = sqlite.CreateCommand();

command.CommandText = @"DELETE FROM Viruses WHERE id = " + j;

command.ExecuteNonQuery();

command = sqlite.CreateCommand();

command.CommandText = @"update Viruses

set id = (select count(\*)

from Viruses s2

where s2.id < Viruses.id or

s2.id = Viruses.id and s2.id <= Viruses.id

)";

command.ExecuteNonQuery();

command = sqlite.CreateCommand();

command.CommandText = @"SELECT \* FROM Viruses";

textBox2.Text = "";

using (var reader = command.ExecuteReader())

{

for (int i = 0; reader.Read(); i++)

{

textBox2.Text += reader.GetString(0) + " " + reader.GetString(1) + "\r\n";

}

}

sqlite.Close();

}

private void button8\_Click(object sender, EventArgs e)

{

Console.WriteLine("Jcnfyjdkty");

}

}

}

## Приложение 3

Класс Scaner

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

using Microsoft.Data.Sqlite;

using System.IO;

using System.IO.Compression;

namespace AntiLibrary

{

public static class Scaner

{

static List<string> list1 = new List<string>();

static List<string> list2 = new List<string>();

static List<string> list3 = new List<string>();

public static List<string> list4 = new List<string>();

public static bool ScanerFile(byte[] arrayofbyte)

{

SignatureFind();

for (int i = 0; i < arrayofbyte.Length-10; i++)

{

string temp = ByteString(arrayofbyte, i, 10);

foreach(string str in list2)

{

temp = ByteString(arrayofbyte, i, (str.Length)/2);

if (temp.Equals(str)) return true;

}

}

return false;

}

public static void ZipArchive(List<string> list)

{

ZipFind();

foreach (string str in list)

{

using (ZipArchive archive = ZipFile.OpenRead(str))

{

foreach (ZipArchiveEntry entry in archive.Entries)

{

foreach (string str1 in list3)

{

if (entry.FullName.EndsWith(".exe", StringComparison.OrdinalIgnoreCase).Equals(str1) ||

entry.FullName.EndsWith(".dll", StringComparison.OrdinalIgnoreCase).Equals(str1))

{

list4.Add(str);

}

}

}

}

}

}

static void ZipFind()

{

SqliteConnection sqlite = new SqliteConnection("Data Source=C:\\BaseData\\BasDate.db");

sqlite.Open();

var command = sqlite.CreateCommand();

command.CommandText = @"SELECT \* FROM Viruse";

using (var reader = command.ExecuteReader())

{

for (int i = 0; reader.Read(); i++)

{

list3.Add(reader.GetString(1));

}

}

sqlite.Close();

list3.Clear();

}

static void SignatureFind()

{

SqliteConnection sqlite = new SqliteConnection("Data Source=C:\\BaseData\\BasDate.db");

sqlite.Open();

var command = sqlite.CreateCommand();

command.CommandText = @"SELECT \* FROM Signature";

using (var reader = command.ExecuteReader())

{

for (int i = 0; reader.Read(); i++)

{

list2.Add(reader.GetString(1));

}

}

sqlite.Close();

}

static string ByteString(byte[] arrayofbyte, int index, int length)

{

string result = BitConverter.ToString(arrayofbyte, index, length).Replace("-", "");

return result;

}

public static void DeleteFile(int id)

{

list1.Clear();

SqliteConnection sqlite = new SqliteConnection("Data Source=C:\\BaseData\\BasDate.db");

sqlite.Open();

var command = sqlite.CreateCommand();

command.CommandText = @"SELECT \* FROM Viruses where id = " + id;

using (var reader = command.ExecuteReader())

{

for (int i = 0; reader.Read(); i++)

{

list1.Add(reader.GetString(1));

}

}

sqlite.Close();

string path = list1[0];

File.Delete(path);

}

public static byte[] GetFileCode(string path, ref int offset)

{

Stream stream = File.OpenRead(path);

bool txtIsFound = false;

while (!txtIsFound)

{

switch ((char)stream.ReadByte())

{

case '.': //.text

if ((char)stream.ReadByte() == 't' &&

(char)stream.ReadByte() == 'e' &&

(char)stream.ReadByte() == 'x' &&

(char)stream.ReadByte() == 't') txtIsFound = true;

break;

default:

break;

}

if (stream.Position == stream.Length - 1) break;

}

stream.Position += 10;

byte[] array = new byte[4];

stream.Read(array, 0, 4);

Array.Reverse(array, 0, array.Length);

int rawDataSize = BitConverter.ToInt32(array, 0);

//Console.WriteLine("rawDataSize = " + rawDataSize);

stream.Read(array, 0, 4);

Array.Reverse(array, 0, array.Length);

int rawDataPosition = BitConverter.ToInt32(array, 0);

//Console.WriteLine("rawDataPosition = " + rawDataPosition);

offset = rawDataPosition;

array = new byte[rawDataSize];

stream.Position = rawDataPosition;

stream.Read(array, 0, rawDataSize - 1);

//Console.WriteLine("rawData: " + BitConverter.ToString(array));

stream.Close();

return array;

}

public static bool MoveToQuarantine(string path, bool Back)

{

try

{

using (var stream = File.Open(path, FileMode.Open))

{

stream.Position = 0;

if (Back == true) stream.WriteByte(0x4D);

else stream.WriteByte(0x66);

stream.Close();

}

}

catch (Exception) { return false; }

return true;

}

}

}

Класс DataBaseAction

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO;

using System.Windows.Forms;

using Microsoft.Data.Sqlite;

namespace AntiLibrary

{

public static class DataBaseAction

{

public static int hours;

public static int minutes;

public static void ConverttoBase(string paths2)

{

var text2 = paths2.Split(' ');

SqliteConnection sqlite = new SqliteConnection("Data Source=C:\\BaseData\\BasDate.db");

sqlite.Open();

/\*var command = sqlite.CreateCommand();

command.CommandText = @"DELETE FROM FilesPE";

command.ExecuteNonQuery();\*/

var command = sqlite.CreateCommand();

foreach (string path in text2)

{

if (path != "")

{

command.CommandText = @"INSERT INTO FilesPE" + "(paths)" + "VALUES('" + path + "')";

command.ExecuteNonQuery();

}

}

command = sqlite.CreateCommand();

command.CommandText = @"DELETE FROM FilesPE WHERE id < (SELECT MAX(id) FROM FilesPE AS T1 WHERE FilesPE.paths = T1.paths);";

command.ExecuteNonQuery();

command = sqlite.CreateCommand();

command.CommandText = @"update FilesPE

set id = (select count(\*)

from FilesPE s2

where s2.id < FilesPE.id or

s2.id = FilesPE.id and s2.id <= FilesPE.id

)";

command.ExecuteNonQuery();

sqlite.Close();

}

public static void DeleteTable()

{

SqliteConnection sqlite = new SqliteConnection("Data Source=C:\\BaseData\\BasDate.db");

sqlite.Open();

var command = sqlite.CreateCommand();

command.CommandText = @"DELETE FROM FilesPE";

command.ExecuteNonQuery();

sqlite.Close();

}

public static void DeleteTableData()

{

SqliteConnection sqlite = new SqliteConnection("Data Source=C:\\BaseData\\BasDate.db");

sqlite.Open();

var command = sqlite.CreateCommand();

command.CommandText = @"DELETE FROM DataAndTime";

command.ExecuteNonQuery();

sqlite.Close();

}

public static void DeleteTableVirus()

{

SqliteConnection sqlite = new SqliteConnection("Data Source=C:\\BaseData\\BasDate.db");

sqlite.Open();

var command = sqlite.CreateCommand();

command.CommandText = @"DELETE FROM Viruses";

command.ExecuteNonQuery();

sqlite.Close();

}

public static void AddToVirusBase(string path)

{

var text2 = path.Split(' ');

SqliteConnection sqlite = new SqliteConnection("Data Source=C:\\BaseData\\BasDate.db");

sqlite.Open();

/\*var command = sqlite.CreateCommand();

command.CommandText = @"DELETE FROM FilesPE";

command.ExecuteNonQuery();\*/

var command = sqlite.CreateCommand();

foreach (string paths in text2)

{

if (path != "")

{

command.CommandText = @"INSERT INTO Viruses" + "(path)" + "VALUES('" + paths + "')";

command.ExecuteNonQuery();

}

}

command = sqlite.CreateCommand();

command.CommandText = @"DELETE FROM Viruses WHERE id < (SELECT MAX(id) FROM Viruses AS T1 WHERE Viruses.path = T1.path);";

command.ExecuteNonQuery();

command = sqlite.CreateCommand();

command.CommandText = @"update Viruses

set id = (select count(\*)

from Viruses s2

where s2.id < Viruses.id or

s2.id = Viruses.id and s2.id <= Viruses.id

)";

command.ExecuteNonQuery();

sqlite.Close();

}

public static void CreateData(int hour,int minutes)

{

SqliteConnection sqlite = new SqliteConnection("Data Source=C:\\BaseData\\BasDate.db");

sqlite.Open();

DateTime data = DateTime.Now;

var command = sqlite.CreateCommand();

command.CommandText = @"INSERT INTO DataAndTime" + "(hour,minute)" + "VALUES('" + hour + "','" + minutes + "')";

command.ExecuteNonQuery();

sqlite.Close();

}

public static void InfoData()

{

SqliteConnection sqlite = new SqliteConnection("Data Source=C:\\BaseData\\BasDate.db");

sqlite.Open();

var command = sqlite.CreateCommand();

command.CommandText = @"SELECT \* FROM DataAndTime";

using (var reader = command.ExecuteReader())

{

for (int i = 0; reader.Read(); i++)

{

hours = Convert.ToInt32(reader.GetString(0));

minutes = Convert.ToInt32(reader.GetString(1));

}

}

sqlite.Close();

}

public static void CreateReport(string typrofscan,int numberofviruses)

{

SqliteConnection sqlite = new SqliteConnection("Data Source=C:\\BaseData\\BasDate.db");

sqlite.Open();

DateTime data = DateTime.Now;

var command = sqlite.CreateCommand();

command.CommandText = @"INSERT INTO Report" + "(time,typeofscan,countofviruses)" + "VALUES('" + data + "','" + typrofscan + "'," + numberofviruses + ")";

command.ExecuteNonQuery();

command = sqlite.CreateCommand();

command.CommandText = @"update Report

set id = (select count(\*)

from Report s2

where s2.id < Report.id or

s2.id = Report.id and s2.id <= Report.id

)";

command.ExecuteNonQuery();

sqlite.Close();

}

public static void InTableMonitor(TextBox textbox,TextBox textBox1)

{

string str = textbox.Text;

SqliteConnection sqlite = new SqliteConnection("Data Source=C:\\BaseData\\BasDate.db");

sqlite.Open();

var command = sqlite.CreateCommand();

command.CommandText = @"INSERT INTO BaseText" + "(path)" + "VALUES('" + str + "')";

command.ExecuteNonQuery();

command = sqlite.CreateCommand();

command.CommandText = @"update BaseText

set id = (select count(\*)

from BaseText s2

where s2.id < BaseText.id or

s2.id = BaseText.id and s2.id <= BaseText.id

)";

command.ExecuteNonQuery();

command = sqlite.CreateCommand();

command.CommandText = @"SELECT \* FROM BaseText";

textBox1.Text = "";

using (var reader = command.ExecuteReader())

{

for (int i = 0; reader.Read(); i++)

{

textBox1.Text += reader.GetString(0) + " " + reader.GetString(1) + "\r\n";

}

}

sqlite.Close();

}

public static void OutTableMonitor(TextBox textbox, TextBox textbox1)

{

int j = Convert.ToInt32(textbox.Text);

SqliteConnection sqlite = new SqliteConnection("Data Source=C:\\BaseData\\BasDate.db");

sqlite.Open();

var command = sqlite.CreateCommand();

command.CommandText = @"DELETE FROM BaseText WHERE id = " + j;

command.ExecuteNonQuery();

command = sqlite.CreateCommand();

command.CommandText = @"update BaseText

set id = (select count(\*)

from BaseText s2

where s2.id < BaseText.id or

s2.id = BaseText.id and s2.id <= BaseText.id

)";

command.ExecuteNonQuery();

command = sqlite.CreateCommand();

command.CommandText = @"SELECT \* FROM BaseText";

textbox1.Text = "";

using (var reader = command.ExecuteReader())

{

for (int i = 0; reader.Read(); i++)

{

textbox1.Text += reader.GetString(0) + " " + reader.GetString(1) + "\r\n";

}

}

sqlite.Close();

}

public static void UpdateTableScanedFiles(TextBox textbox)

{

SqliteConnection sqlite = new SqliteConnection("Data Source=C:\\BaseData\\BasDate.db");

sqlite.Open();

var command = sqlite.CreateCommand();

command.CommandText = @"update FilesPE

set id = (select count(\*)

from FilesPE s2

where s2.id < FilesPE.id or

s2.id = FilesPE.id and s2.id <= FilesPE.id

)";

command.ExecuteNonQuery();

command = sqlite.CreateCommand();

command.CommandText = @"SELECT \* FROM FilesPE";

textbox.Text = "";

using (var reader = command.ExecuteReader())

{

for (int i = 0; reader.Read(); i++)

{

textbox.Text += reader.GetString(0) + " " + reader.GetString(1) + "\r\n";

}

}

sqlite.Close();

}

public static void UpdateTableViruse(TextBox textbox)

{

SqliteConnection sqlite = new SqliteConnection("Data Source=C:\\BaseData\\BasDate.db");

sqlite.Open();

var command = sqlite.CreateCommand();

command.CommandText = @"update Viruses

set id = (select count(\*)

from Viruses s2

where s2.id < Viruses.id or

s2.id = Viruses.id and s2.id <= Viruses.id

)";

command.ExecuteNonQuery();

command = sqlite.CreateCommand();

command.CommandText = @"SELECT \* FROM Viruses";

textbox.Text = "";

using (var reader = command.ExecuteReader())

{

for (int i = 0; reader.Read(); i++)

{

textbox.Text += reader.GetString(0) + " " + reader.GetString(1) + "\r\n";

}

}

sqlite.Close();

}

public static void DeleteValueFromViruse(TextBox textbox,TextBox textbox2)

{

int j = Convert.ToInt32(textbox.Text);

AntiLibrary.Scaner.DeleteFile(j);

SqliteConnection sqlite = new SqliteConnection("Data Source=C:\\BaseData\\BasDate.db");

sqlite.Open();

var command = sqlite.CreateCommand();

command.CommandText = @"DELETE FROM Viruses WHERE id = " + j;

command.ExecuteNonQuery();

command = sqlite.CreateCommand();

command.CommandText = @"update Viruses

set id = (select count(\*)

from Viruses s2

where s2.id < Viruses.id or

s2.id = Viruses.id and s2.id <= Viruses.id

)";

command.ExecuteNonQuery();

command = sqlite.CreateCommand();

command.CommandText = @"SELECT \* FROM Viruses";

textbox2.Text = "";

using (var reader = command.ExecuteReader())

{

for (int i = 0; reader.Read(); i++)

{

textbox2.Text += reader.GetString(0) + " " + reader.GetString(1) + "\r\n";

}

}

sqlite.Close();

}

public static void UpdateReport(TextBox textbox)

{

SqliteConnection sqlite = new SqliteConnection("Data Source=C:\\BaseData\\BasDate.db");

sqlite.Open();

var command = sqlite.CreateCommand();

command.CommandText = @"update Report

set id = (select count(\*)

from Report s2

where s2.id < Report.id or

s2.id = Report.id and s2.id <= Report.id

)";

command.ExecuteNonQuery();

command = sqlite.CreateCommand();

command.CommandText = @"SELECT \* FROM Report";

textbox.Text = "";

using (var reader = command.ExecuteReader())

{

for (int i = 0; reader.Read(); i++)

{

textbox.Text += reader.GetString(0) + " Дата: " + reader.GetString(1) + " Тип: " + reader.GetString(2) + " Кол. вирусов: " + reader.GetString(3) + "\r\n";

}

}

sqlite.Close();

}

}

}

Класс PEFILE

using System;

using System.IO.Compression;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.IO;

using System.Windows.Forms;

using System.Threading;

using System.Net;

using System.Net.Sockets;

using System.Runtime.InteropServices;

using System.Diagnostics;

namespace AntiLibrary

{

public static class PEFILE

{

public static void ZipUnPacked(string paths,string extractPath)

{

using (ZipArchive archive = ZipFile.OpenRead(paths))

{

foreach (ZipArchiveEntry entry in archive.Entries)

{

// Gets the full path to ensure that relative segments are removed.

string destinationPath = Path.GetFullPath(Path.Combine(extractPath, entry.FullName));

// Ordinal match is safest, case-sensitive volumes can be mounted within volumes that

// are case-insensitive.

if (destinationPath.StartsWith(extractPath, StringComparison.Ordinal))

{

entry.ExtractToFile(destinationPath);

}

}

}

}

static PEFile ToFileHeader(byte[] ar)

{

GCHandle handle = GCHandle.Alloc(ar, GCHandleType.Pinned);

PEFile pfh = (PEFile)Marshal.PtrToStructure(handle.AddrOfPinnedObject(), typeof(PEFile));

handle.Free();

return pfh;

}

public static bool IsPEFile(string path)

{

using (BinaryReader br = new BinaryReader(File.OpenRead(path), Encoding.ASCII))

{

/// Читаем первые 64 байта. Именно там находится заголовок.

byte[] ar = br.ReadBytes(64);

/// Заносим эти байты в структуру

PEFile sig = ToFileHeader(ar);

/// Переходим по смещению для чтения первых двух символов PE-заголовка

br.BaseStream.Seek(sig.PEHeaderAddress, SeekOrigin.Begin);

/// Читаем два символа и переводим их в строку

string pe = new string(br.ReadChars(2));

/// Возвращаем результат выполнения

return sig.DosHeader == "MZ" && pe == "PE";

}

}

[StructLayout(LayoutKind.Explicit)]

struct PEFile

{

[MarshalAs(UnmanagedType.ByValTStr, SizeConst = 3)]

[FieldOffset(0)]

public string DosHeader;

[MarshalAs(UnmanagedType.I8)]

[FieldOffset(0x3C)]

public long PEHeaderAddress;

}

}

}

Класс MonitoringO

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO;

namespace AntiLibrary

{

public static class MonitoringO

{

public static string monitor = "";

public static void MonitoringON(object obj)

{

var path = (string)obj;

var watcher = new FileSystemWatcher(path);

watcher.NotifyFilter = NotifyFilters.LastWrite;

watcher.Changed += OnChanged;

watcher.IncludeSubdirectories = true;

watcher.EnableRaisingEvents = true;

}

public static void OnChanged(object sender, FileSystemEventArgs e)

{

if (e.ChangeType != WatcherChangeTypes.Changed)

{

return;

}

monitor = $"{e.FullPath}";

}

}

}