**МИНИСТЕРСТВО ТРАНСПОРТА РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА»**

**ИНСТИТУТ ТРАНСПОРТНОЙ ТЕХНИКИ И СИСТЕМ УПРАВЛЕНИЯ (ИТТСУ)**

**Кафедра «Управление и защита информации»**

**Инструменты реверс-инжиниринга и транскомпиляции**

**Учебное пособие**

**Москва – 2024**

**МИНИСТЕРСТВО ТРАНСПОРТА РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА»**

**ИНСТИТУТ ТРАНСПОРТНОЙ ТЕХНИКИ И СИСТЕМ УПРАВЛЕНИЯ (ИТТСУ)**

**Кафедра «Управление и защита информации»**

**Инструменты реверс-инжиниринга и транскомпиляции**

**Учебное пособие**

**по направлению подготовки**

**10.05.01 «Компьютерная безопасность»**

**Москва – 2024**

УДК [004.056](https://www.teacode.com/online/udc/00/004.91.html).53

И 72

Инструменты реверс-инжиниринга и транскомпиляции: Учебное пособие / Ковров А.И., Ляпина Е.П., Савин Л.А., Сидоренко В.Г., Соколов И.Д. – М.: РУТ(МИИТ).2024. – 62 с.

В учебном пособии рассмотрены инструменты, реализующие функции дизассемблирования, декомпиляции и транскомпиляции и применяемые для анализа и синтеза программного кода, созданного на различных языках программирования высокого уровня.

Учебное пособие предназначено для обучающихся по специальности 10.05.01 «Компьютерная безопасность», а также других направлений, связанных с информационными технологиями.

Рецензенты:

Начальник центра искусственного интеллекта АО «ВНИИЖТ» М.А. Кулагин

Заведующий кафедрой [«Вычислительные системы, сети и информационная безопасность»](https://rut-miit.ru/depts/29) РУТ (МИИТ) Б.В. Желенков

© РУТ (МИИТ), 2024

Оглавление

[Введение 4](#_Toc167437523)

[1. РЕВЕРС-инжиниринг программного кода, созданного на ЯП Python 5](#_Toc167437524)

[2. РЕВЕРС-инжиниринг программного кода, созданного на ЯП Python 8](#_Toc167437525)

[3. Реверс инжиниринг программного кода, созданного на языке программирования C# 19](#_Toc167437526)

[3.1. применение Инструмента ILSpy 19](#_Toc167437527)

[3.2. применение Инструмента .NET Reflector 25](#_Toc167437528)

[4. Реверс инжиниринг программного кода, созданного на языке программирования C и С++ 29](#_Toc167437529)

[4.1. Применение инструмента GHIDRA 29](#_Toc167437530)

[4.2. Применение инструмента x64dbg 49](#_Toc167437531)

[5. Конвертация программного кода с одного языка программирования на другой 55](#_Toc167437532)

[5.1. Применение языковой модели с искуственным интеллектом в Telegram 55](#_Toc167437533)

[5.2. применение сервиса AI Code converter 62](#_Toc167437534)

[заключение 67](#_Toc167437535)

[Список литературы 69](#_Toc167437536)

# Введение

Инструментарий реверс-инжиниринга позволяет исследовать функционал программного обеспечения (ПО) и выявлять его недекларированные возможности. Делать это на основе анализа исходного или восстановленного программного кода на языке программирования (ЯП) высокого уровня удобнее, чем с использованием дизассемблированного кода. Получаемые при этом результаты скорее всего будут более полными. Возможности инструментов реверс-инжиниринга зачастую достаточно обширны. Среди них не только просмотр дизассемблированного кода, но и поиск строковых вхождений, построение графов программы, декомпиляция в высокоуровневые языки, плагины для скриптинга и многое другое. Выбор инструментария зависит от того, на каком ЯП было создано ПО. В учебном пособии рассмотрены инструменты, реализующие функции дизассемблирования и декомпиляции программного кода, созданного на различных ЯП высокого уровня, что позволяет провести сравнение особенностей результатов компиляции ПО и обратных к ней действий дизассемблирования и декомпиляции в зависимости от исходного ЯП.

В работах [1] проведена оценка применимости конкретного инструмента к исходному коду исполняемых файлов, созданному на конкретном ЯП с точки зрения читаемости дизассемблированного кода. Полученные результаты представлены в Таблице В.1.

При решении задач реверс-инжиниринга следует учитывать возможное наличие обфускации исходного кода. Программа с обфускацией должна быть предварительно деобфусцирована с помощью специального ПО. Существует множество таких программ [2].

В учебном пособии рассмотрен инструментарий конвертации исходного кода с одного ЯП на другой. Разбор программного кода основан на знании синтаксиса ЯП без учета бизнес-логики ПО. Такой подход можно ограниченно применять для анализа программ. Основное же предназначение конвертации – продление жизнеспособности проектов, которые по тем или иным причинам устарели и нуждаются в обновлении.

Таблица В.1 – Сравнительная характеристика ЯП с точки зрения возможностей реверс-инжиниринга

|  |  |  |  |
| --- | --- | --- | --- |
| Критерии сравнения | Язык исходного кода программы | | |
| Python | C# | C++ |
| Набор ПО для обратной разработки | PyInstaller Extractor, uncompyle6 | .NET Reflector,  ILSpy | IDA Pro,  Radare2 |
| Сложность анализа | Легкий | Средней сложности | Затруднённый |
| Возможность получить исходный код | Да | Частично | Нет |
| Уровень необходимых навыков для обратной разработки | Знание языка Python, базовые умения работы с командной строкой | Понимание логики промежуточного кода, умение работы с разнообразным  ПО для обратной разработки. Навыки определения обфускации  программного кода | Глубокие знания устройства языка ассемблера, логики языка C++, навыки работы с hex редакторами, разнообразным ПО для обратной разработки |
| Пригодность языка для использования в ПО систем управления | Нет | Пригоден в некоторых случаях | Рекомендуется к применению |

Представленный в учебном пособии инструментарий применяется в ходе выполнения лабораторных работ по дисциплине «Технологии реверс-инжинирирнга».

# РЕВЕРС-инжиниринг программного кода, созданного на ЯП Python

Рассмотрим возможный способ восстановления исходного кода ПО, разработанного на ЯП Python. Обобщенный алгоритм выглядит следующим образом:

1. Установка необходимых инструментов (PyInstaller Extractor, uncompyle6).
2. Перенос PyInstaller Extractor в папку с исполняемым файлом (\*.exe), запуск со ссылкой на него.
3. Обработка файлов из созданной папки программой PyInstaller Extractor вручную и с помощью hex редактора.
4. Перевод байт-кода Python (\*.pyc) в исходный код с использованием модуля uncompyle6.

Python – интерпретируемый ЯП (в отличие от С-подобных языков), следовательно после упаковки с помощью каких-либо инструментов (например, PyInstaller) во время запуска и компиляции исполняемого файла обязательно создается новая папка, находящаяся по адресу user/AppData/Local/Temp и содержащая дополнительные вспомогательные файлы для компиляции. Данные файлы возможно найти после компиляции exe-файла.

ПРИМЕЧАНИЕ: при выполнении последующих действий лучше использовать версию Python до 3.10 (в данном примере используется версия 3.7.9), т.к. инструмент uncompyle6 (см. ниже) может работать некорректно на версиях выше.

Для того, чтобы не использовать папку Temp, создадим виртуальное окружение в папке с проектом с помощью команды python -m venv venv. Активируется окружение командой .\venv\Scripts\activate.bat. После активации в терминале будет отображаться метка (venv), свидетельствующая о том, что окружение запущено. В этом случае после установки пакеты PyInstaller, uncompyle6 будут храниться в директории venv.

Для примера рассмотрим программу авторизации пользователя в системе (Рисунок 1.1).

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 1.1 – Пример программы на Python

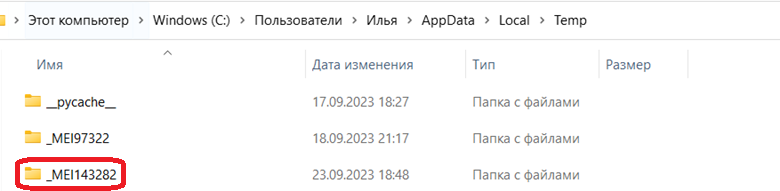
Данная программа была упакована в exe-файл c помощью инструмента PyInstaller. Далее производится запуск и компиляция exe-файла с помощью командной строки (Рисунок 1.2).

Изображение выглядит как текст, Шрифт, снимок экрана, черный

Автоматически созданное описание

Рисунок 1.2 – Компиляция exe-файла

После запуска и компиляции в папке AppData\Local\Temp появляется новая папка, которая содержит в себе файлы, необходимые для запуска и компиляции exe-файла (Рисунок 1.3, Рисунок 1.4).

Рисунок 1.3 – Расположение новой папки, созданной во время запуска и компиляции

Изображение выглядит как текст, снимок экрана, число, меню

Автоматически созданное описание

Рисунок 1.4 – Дополнительные файлы для компиляции и запуска, созданные в новой папке

Во время запуска и компиляции Python распаковывает все библиотеки и коды в данную папку, значит для их получения, а также встроенных ключей, лицензий, паролей достаточно обработать те файлы, которые были созданы во время запуска и компиляции exe-файла. Это говорит о том, что программы, созданные на языке Python и упакованные с помощью PyInstaller Extractor, не защищены от получения исходного кода. Любой пользователь с достаточными знаниями сможет извлечь исполняемые файлы из папки, созданной во время запуска. Дополнительные меры защиты не имеют смысла, так как злоумышленник, у которого будет исполняемый файл, сможет получить доступ к исходному коду программы.

Следующий шаг – установка специального вспомогательного инструмента PyInstaller Extractor, который при запуске и компиляции exe-файла извлечет все созданные файлы из папки Temp и сохранит их, чтобы в дальнейшем возможно было работать с ними уже без необходимости запускать и компилировать exe-файла. Скачать инструмент можно по следующей ссылке: [https://sourceforg.net/projects/pyinstallerextractor/](https://sourceforge.net/projects/pyinstallerextractor/).

Еще понадобится дополнительный модуль uncompyle6, который можно установить с помощью команды pip install (Рисунок 1.5). uncompyle6 переводит байт-код Python обратно в эквивалентный исходный код Python.

Изображение выглядит как текст, электроника, снимок экрана, программное обеспечение

Автоматически созданное описание

Рисунок 1.5 – Установка uncompile6

Далее необходимо переместить PyInstaller Extractor в папку с exe файлом (Рисунок 1.6).

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

Рисунок 1.6 – Итоговое содержимое папки

Из этой папки в командной строке необходимо выполнить команду *python pyinstxtractor.py <имя файла>* (Рисунок 1.7).

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 1.7 – Запуск PyInstaller Extractor в командной строке

После окончания работы PyInstaller Extractor в папке с ним и exe-файлом создается новая папка, в которой сохранились исполняемые файлы, созданные при запуске и компиляции (Рисунок 1.8, Рисунок 1.9).

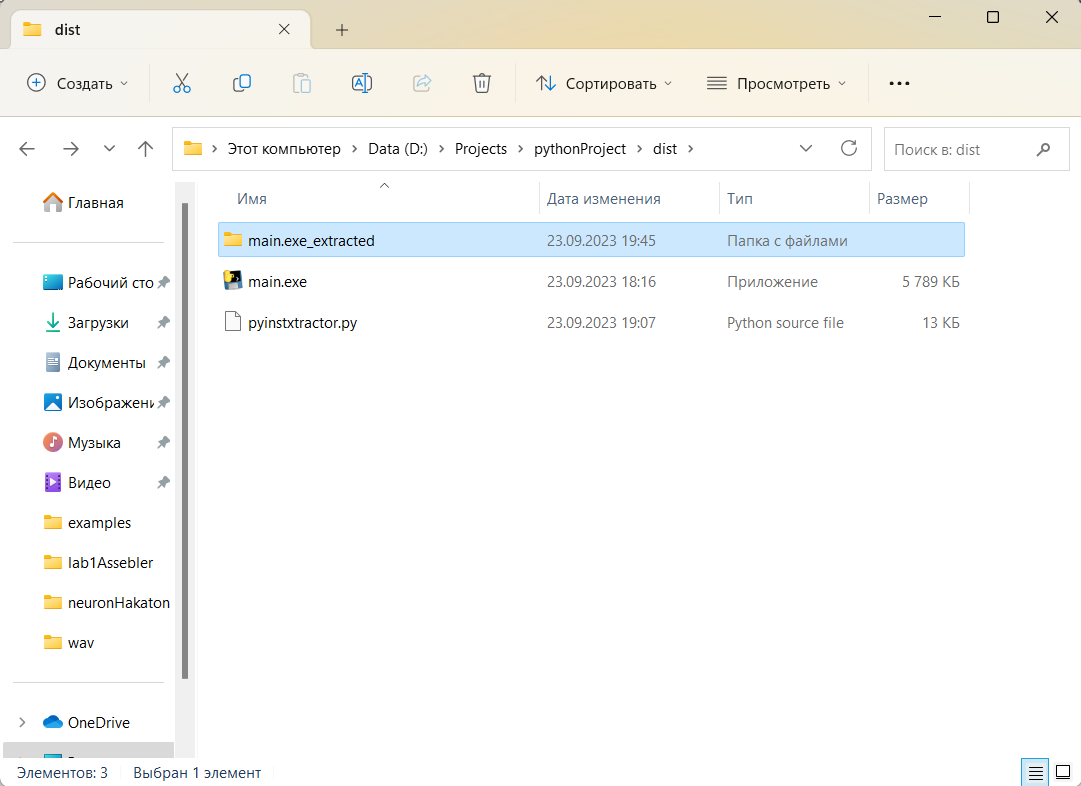


Рисунок 1.8 – Новая папка с файлами, созданными в ходе компиляции

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

Рисунок 1.9 – Исполняемы файлы, созданные во время запуска и компиляции exe и сохраненные с помощью PyInstaller Extractor

Далее в новой папке необходимо найти файл с таким же названием, как и у исполняемого файла (Рисунок 1.10).

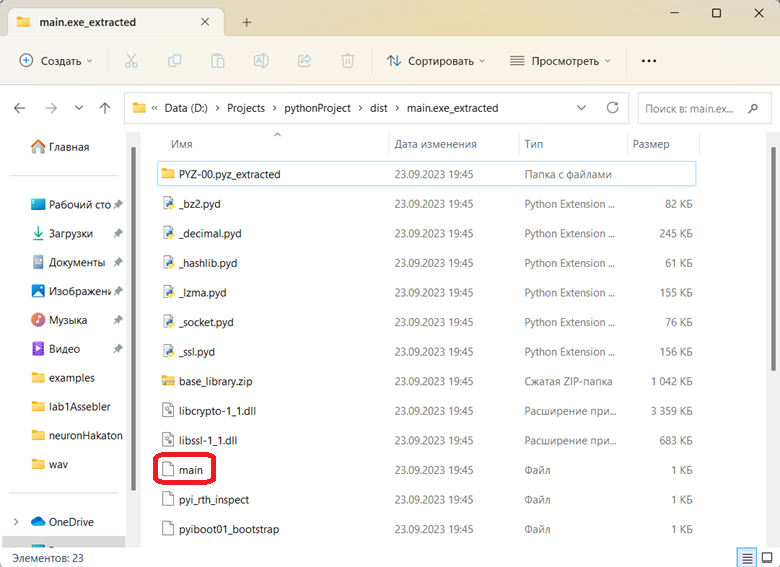


Рисунок 1.10 – Искомый файл

Далее необходимо задать формат .pyc этому файлу (Рисунок 1.11).



Рисунок 1.11 – Изменение формата файла

Далее в этой же папке необходимо найти архив base\_library (Рисунок 1.12), в нем необходимо найти файл abc.pyc (Рисунок 1.13).

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание

Рисунок 1.12 – Найденный архив

Изображение выглядит как текст, программное обеспечение, Значок на компьютере, веб-страница

Автоматически созданное описание

Рисунок 1.13 – Искомый файл в архиве

Далее нужно открыть этот файл через hex редактор и скопировать первую строку (Рисунок 1.14).

Изображение выглядит как текст, число, программное обеспечение, Шрифт

Автоматически созданное описание

Рисунок 1.14 – Hex-редактор

Эту строку нужно вставить в файл .pyc (см. выше рисунки 1.10, 1.11) и сохранить (Рисунок 1.15).

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

Рисунок 1.15 – Изменение файла main.pyc

Далее через консоль необходимо запустить uncompyle6 с указанием на файл main.pyc, с сохранение в новый файл, где будет находиться исходный код программы (Рисунок 1.16).

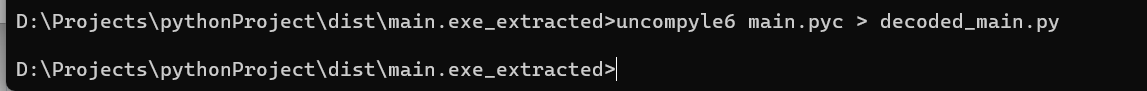


Рисунок 1.16 – Запуск uncompyle6

В папке появился новый файл с кодом на python (рисунок 1.17).

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

Рисунок 1.17 – Файл с исходным кодом

Открыв этот файл в любом IDE (в данном примере использовалась Visual Studio 2019) можно увидеть исходный код на Python, написанный изначально, до упаковки exe (Рисунок 1.18). Таким образом, удалось получить из exe файла исходный код, т.е. выполнить процедуру реверс-инжиниринга.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание

Рисунок 1.18 – Полученный исходный код

Таким образом, использовать Python для создания ПО небезопасно, т.к исходный код можно получить без серьезного анализа (с помощью IDA или RADARE2), что делает бесполезным защиту внутри кода путем шифрования или хеширования. Для создания ПО более безопасно использовать язык С++ и ему подобные, т.к. они компилируемые и, следовательно, для запуска и компиляции ПО системе не придется создавать новые вспомогательные файлы, которые можно обнаружить в папке Temp простым способом.

# Реверс инжиниринг программного кода, созданного на ЯП C#

# применение Инструмента ILSpy

В данном блоке рассмотрим инструмент ILSpy, который позволяет нам декомпилировать приложения, написанные на языке программирования C#.

Приступим к установке данного инструмента. Для этого необходимо скачать официальный репозиторий проекта с GitHub [3]. В данных методических указаниях рассмотрим установку и запуск при помощи Microsoft Visual Studio 2022 (далее – MVS). После успешной загрузки найдём в папке файл «ILSpy.sln», который является файлом решения MVS. После успешной загрузки решения, MVS предложит установить необходимые зависимости, с чем мы соглашаемся (рисунок 2.1).

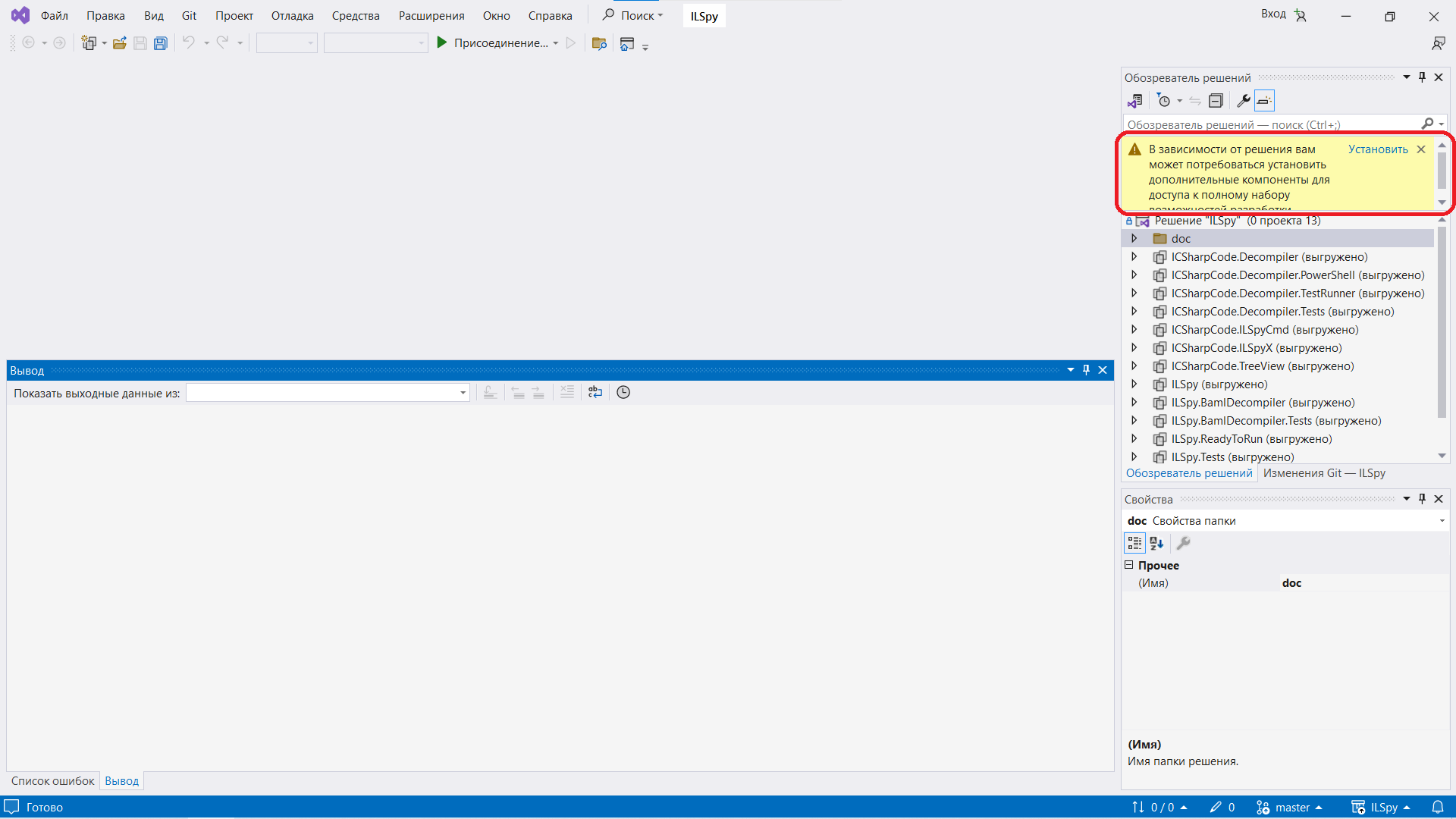


Рисунок 2.1 – Установка необходимых компонентов

Откроется программ установщика MVS (рисунок 2.2). Необходимо дождаться процесса установки компонентов.

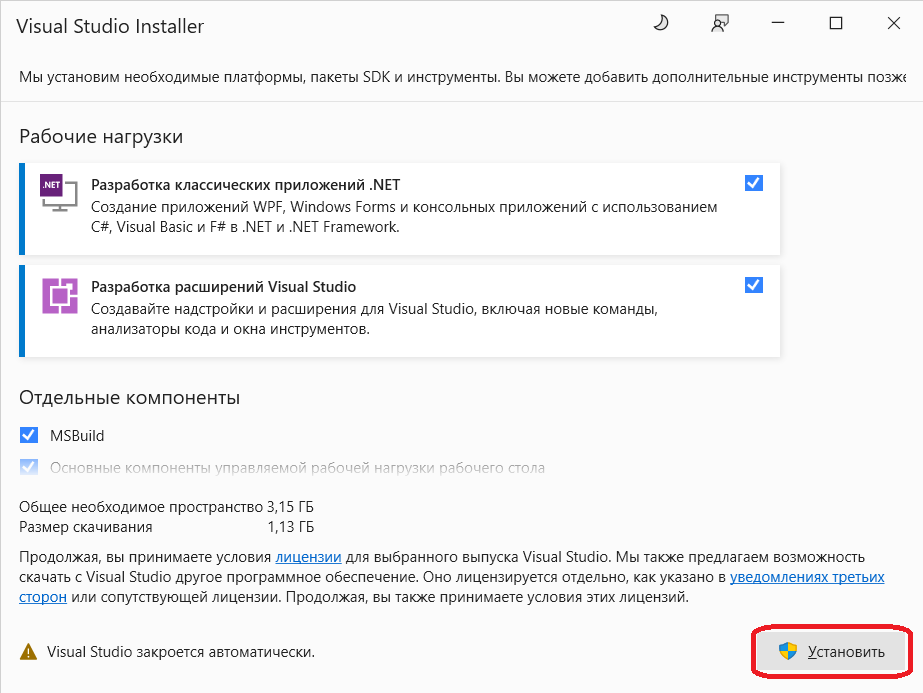


Рисунок 2.2 – Второй шаг установки компонентов

После успешной установки вернёмся к нашему проекту. Теперь, когда все зависимости разрешены, можно запускать приложение. Для этого необходимо нажать на зелёный треугольник вверху (рисунок 2.3).

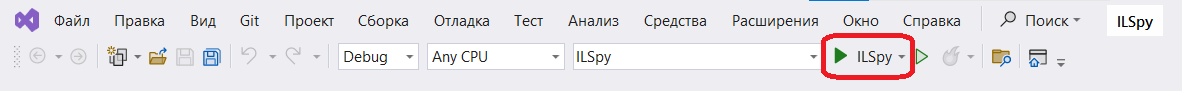


Рисунок 2.3 – Начало сборки проекта

После нажатия начнётся процесс сборки. Это может занять продолжительное время. Наконец, по завершении всех операций, для нас откроется окно программы.

В дальнейшем запуск будет осуществляться простым нажатием на зелёный треугольник, так как проект будет уже собран.

Перейдём к основной части. Начнём процесс реверс инжиниринга с запуска программы для анализа [2], которая изображена на рисунке 2.4. Пароль для архива с программой – «crackmes.one».

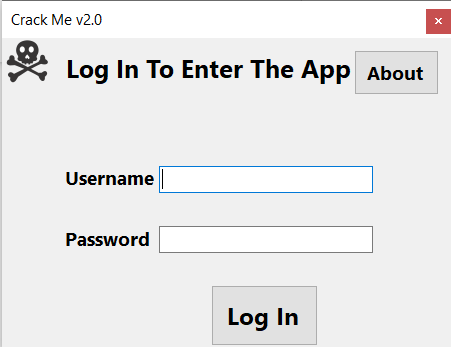


Рисунок 2.4 – Окно запущенной программы

Здесь мы видим два 2 поля, в которые можно ввести данные, и кнопку «Log In». Проверим, что программа действительно имеет логику и не даёт нам произвести авторизацию со случайным логином-паролем (рисунок 2.5).

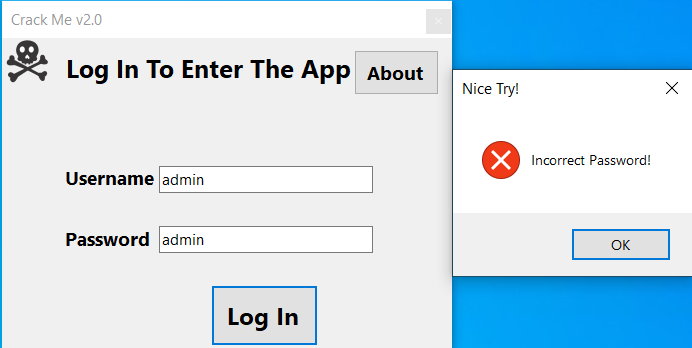


Рисунок 2.5 – Неверные данные на входе

Действительно, программа не позволяет нам просто так войти внутрь.

Программа ILSpy содержит большое число настроек, но наиболее интересны для нас выбор версии языка и язык, на который программа будет декомпилировать код. По умолчанию стоит C#, и С# 11.0 соответственно, что вполне нас устраивает. Помимо этого, программа имеет несколько функциональных областей, с которыми мы будем работать, а именно: навигация по файлам и функциям (отмечено цифрой 1) и сам декомпилированный код (отмечено цифрой 2). Это отображено на рисунке 2.6.

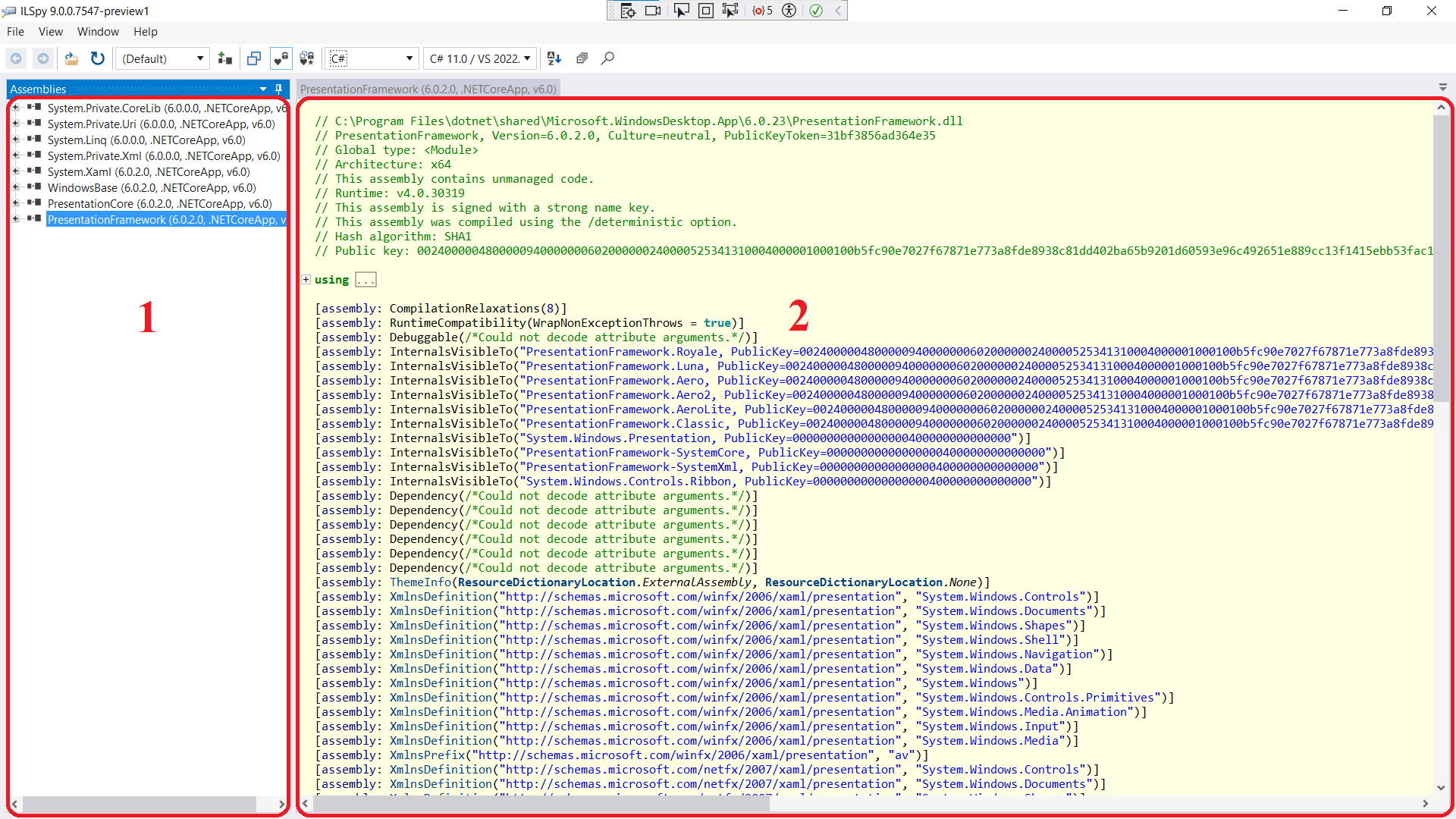


Рисунок 2.6 – Окно программы

В меню навигации найдём класс Form1. В этом классе, как можно видеть на рисунке 2.7, объявлены основные методы и поля нашего приложения.

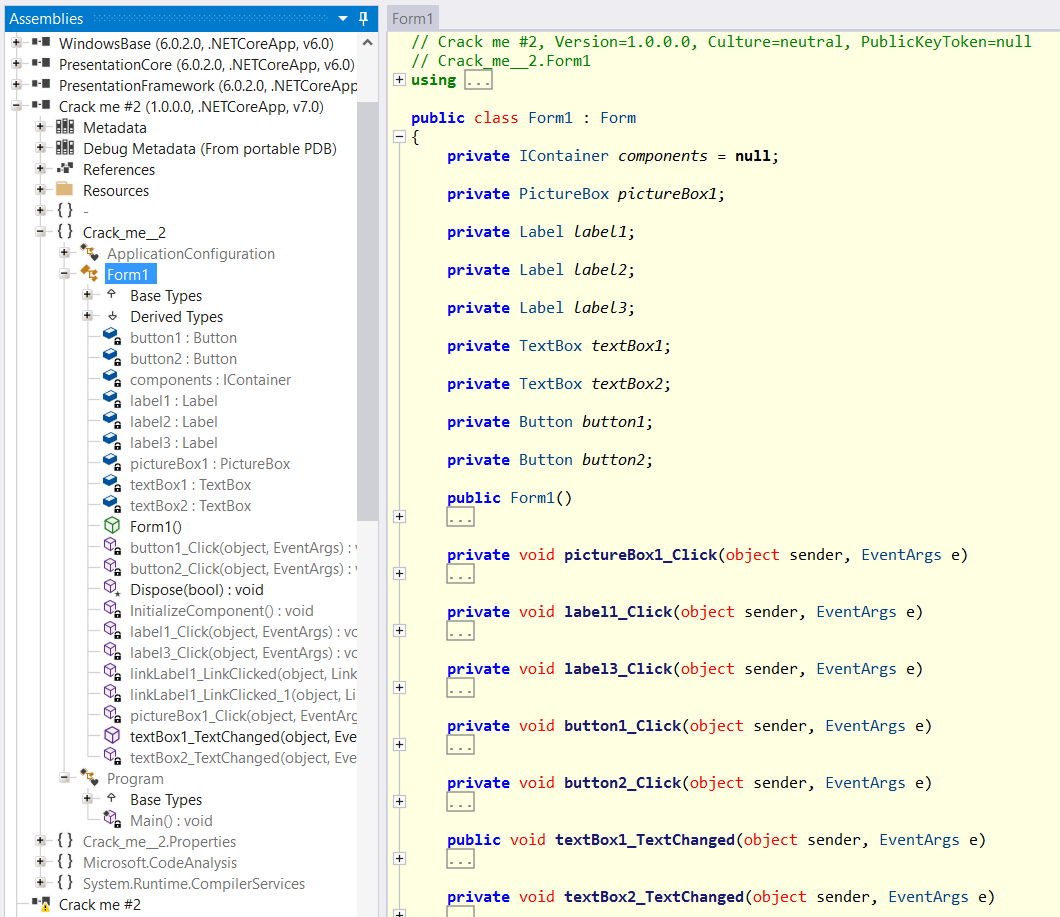


Рисунок 2.7 – Основные методы и поля программы

Просмотрим методы этого класса. В процессе натыкаемся на метод button1\_Click, в которых описан весьма интересный нам функционал, отражённый на рисунке 2.8.

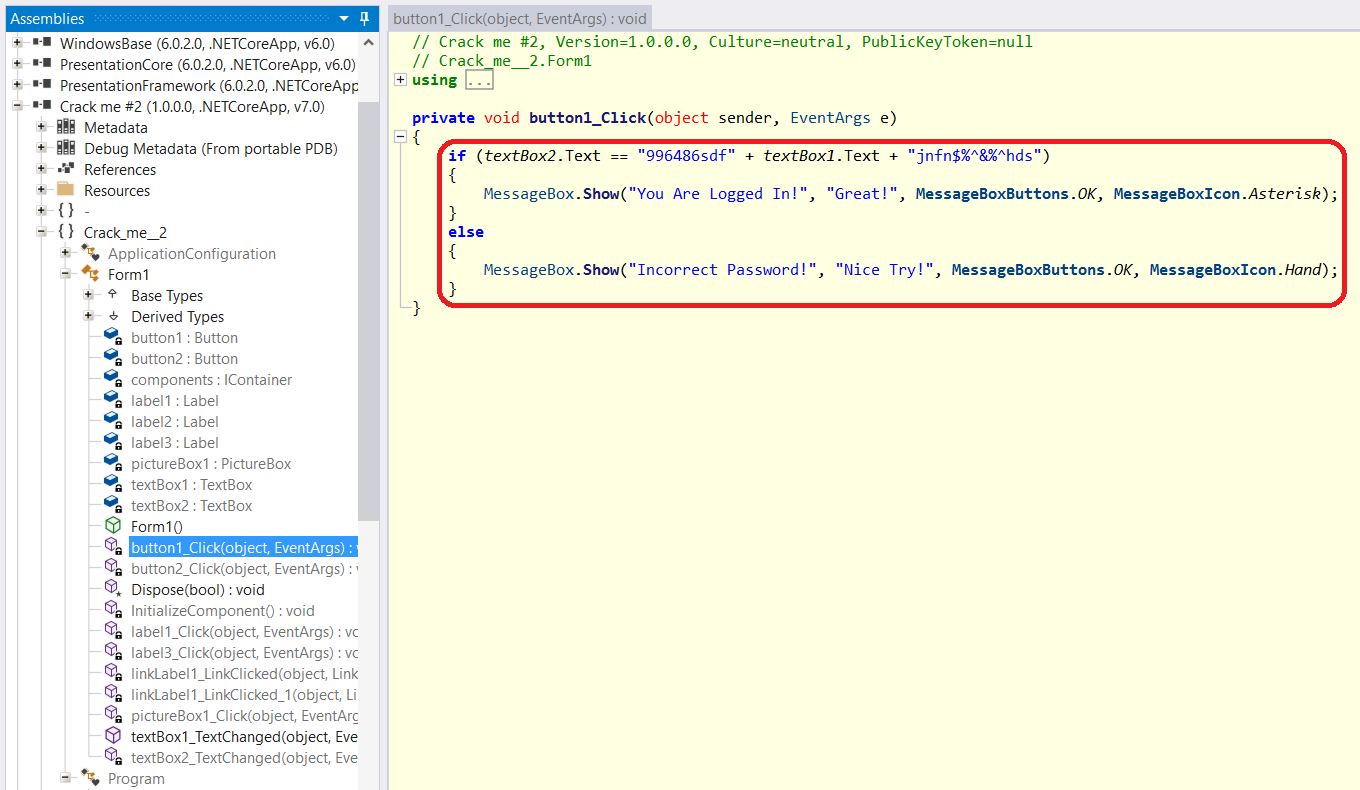


Рисунок 2.8 – Функционал авторизации

Данный код выполняет проверку на равенство содержимого текстового поля textBox2 и строки "996486sdf" + textBox1.Text + "jnfn$%^&%^hds", где textBox1.Text – содержимое поля textBox1. Если условие выполняется, на выходе мы получаем сообщение «You Are Logged In! Great!». В случае провала же – сообщение «Incorrect Password! Nice Try!». Таким образом, программа ожидает, что мы введём некоторый логин, а затем пароль по заданной конструкции, в которой используется логин. Проверим это. Введём в первое поле «admin», а во второе «996486sdfadminjnfn$%^&%^hds». Результат отражён на рисунке 2.9.

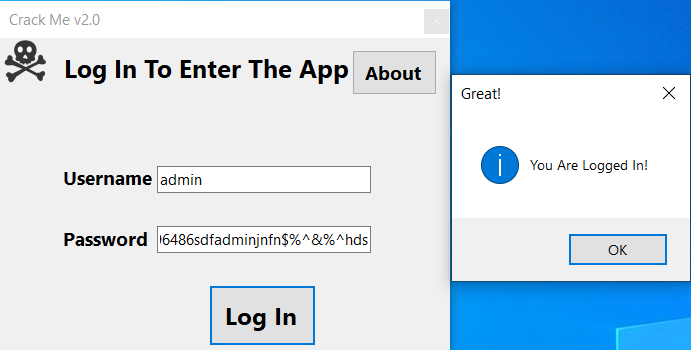


Рисунок 2.9 – Результат нашего анализа

# применение Инструмента .NET Reflector

В данном блоке будет рассмотрен процесс реверс инжиниринга собранной программы, исходный код которой написан на языке программирования C#. Для работы будем использовать ПО от разработчика Redgate под названием .NET Reflector [1]. В качестве примера будет разбирать ту же программу с сайта, содержащего специально собранные программы для практики в искусстве реверс инжиниринга, а конкретнее, программу CrackMe v.2 [2], которая была использована в предыдущем примере.

Запустим .NET Reflector. Главное меню программы представлено на рисунке 2.10.

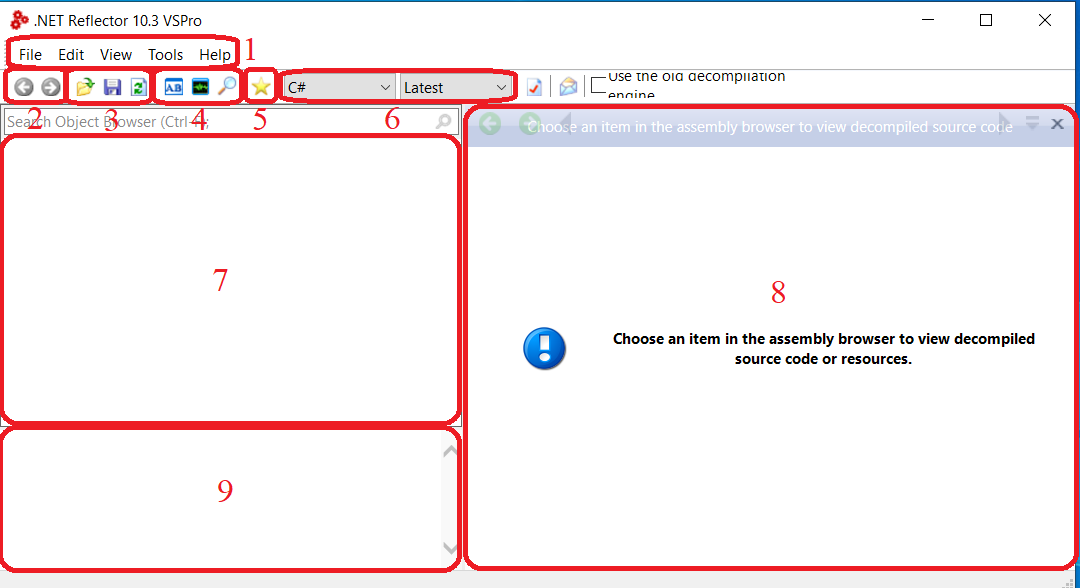


Рисунок 2.10 – Главное меню программы

В этом меню расположены несколько позиций, рассмотрим каждую подробнее:

Блок 1 – это основное контекстное меню, предоставляющее доступ ко всему функционалу программы.

Блок 2 отвечает за навигацию по файлу и по коду.

Блок 3 отвечает за загрузку, сохранение и обновление полей программы.

Блок 4 содержит в себе инструменты декомпиляции, анализа и поиска.

Блок 5 позволяет перейти в закладки в коде, которые пользователь сам определит.

Блок 6 позволяет выбрать язык декомпиляции программы и версию.

Самые полезные блоки для нас начинаются с 7-ого.

Блок 7 отвечает за список загруженных файлов и навигацию по ним. Здесь можно перейти в методы и функции программы, что будет рассмотрено далее.

Блок 8 позволяет нам просмотреть декомпилированный код, где мы, собственно, и будем работать.

Блок 9 отвечает за вывод информации о файле или библиотеке.

Для того, чтобы начать работу, откроем библиотеку нашего приложения в программе. Для этого выберем .dll файл, который необходим для работы данной программы, он находится в той же папке, что и программа. Пройдёмся по файлу в поиске «интересных» методов. Как видим, в программе есть класс Form1, который содержит основные поля и методы программы (рисунок 2.11). Как показала практика, ключевой функционал находится в подобных классах. Пройдём глубже.

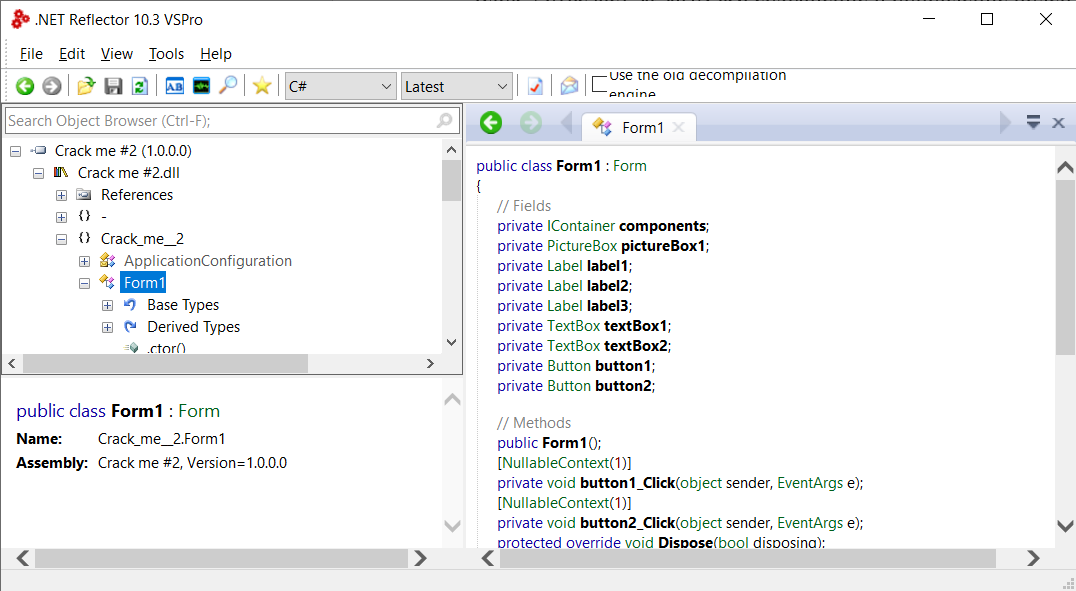


Рисунок 2.11 – Класс Form1

Зайдём в метод обработки нажатия button1\_Click. Как можно видеть, внутри находится декомпилированный код метода, и он содержит несколько полезных нам строк (рисунок 2.12).

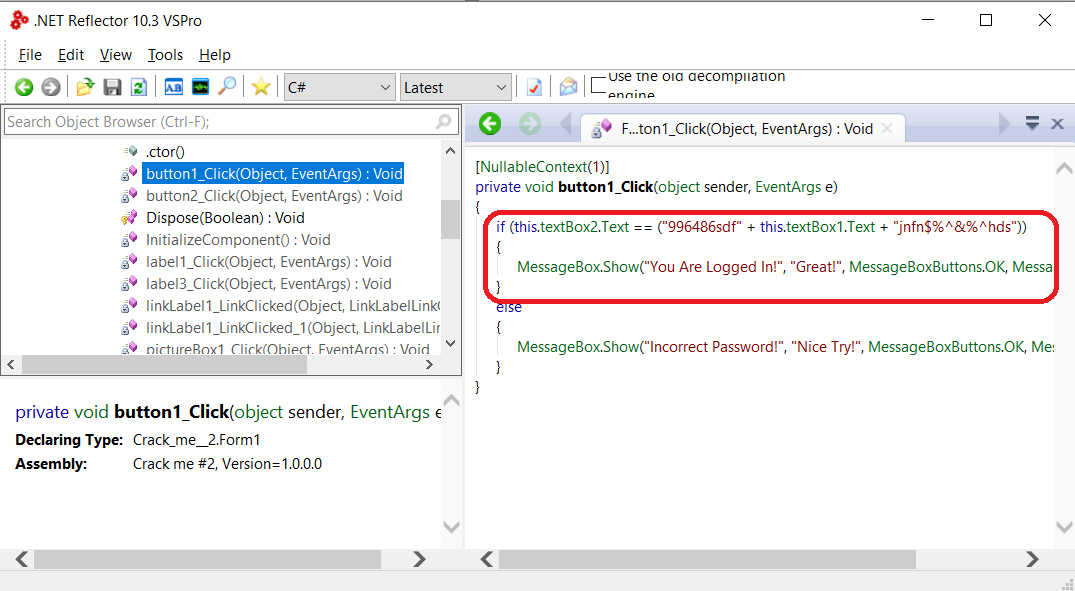


Рисунок 2.12 – Метод button1\_Click

Видим, что обнаруженный метод, как и в разделе 2.1, делает проверку, по результату которой показывает сообщение «You Are Logged In! Great» в случае успеха, и «Incorrect Password! Nice Try» в случае провала. Разберём механизм проверки. Он сравнивает значение поля textBox2 с некоторым значением строки, в которой фигурирует поле textBox1. Таким образом, попробуем ввести в первое поле логина «usr», а в поле пароля введём «996486sdf» + usr + «jnfn$%^&%^hds». Результат – успешный вход, приведён на рисунке 2.13.

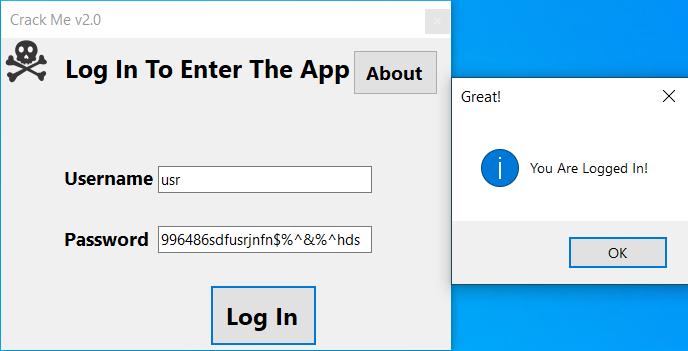


Рисунок 2.13 – Ввод подобранного пароля

**Вывод:** Написание программ на языке программирования C# является небезопасной практикой, так как любой человек, имеющий необходимые минимальные знания в этой области и подходящее ПО, способен получить и проанализировать исходный код любой программы.

# Реверс инжиниринг программного кода, созданного на языке программирования C и С++

# Применение инструмента GHIDRA

В данном разделе рассмотрен инструмент отладки программ под названием GHIDRA. В основные возможности этой программы входит декомпиляция проектов, анализ кода, построение разнообразных графов для анализа программ. В целом, инструмент является открытым и легко расширяемым, что даёт возможность любому пользователю написать плагин, реализующий функционал под его нужды, и интегрировать его в среду GHIDRA. Также данное ПО позволяет выполнять удалённое подключение к серверу. В свою очередь, это даёт возможность сразу нескольким людям работать над одним проектом. Перейдём к установке данной программы.

Первым шагом необходимо скачать архив, содержащий программу, с сайта GitHub [4]. Нужен файл, указанный на рисунке 3.1.

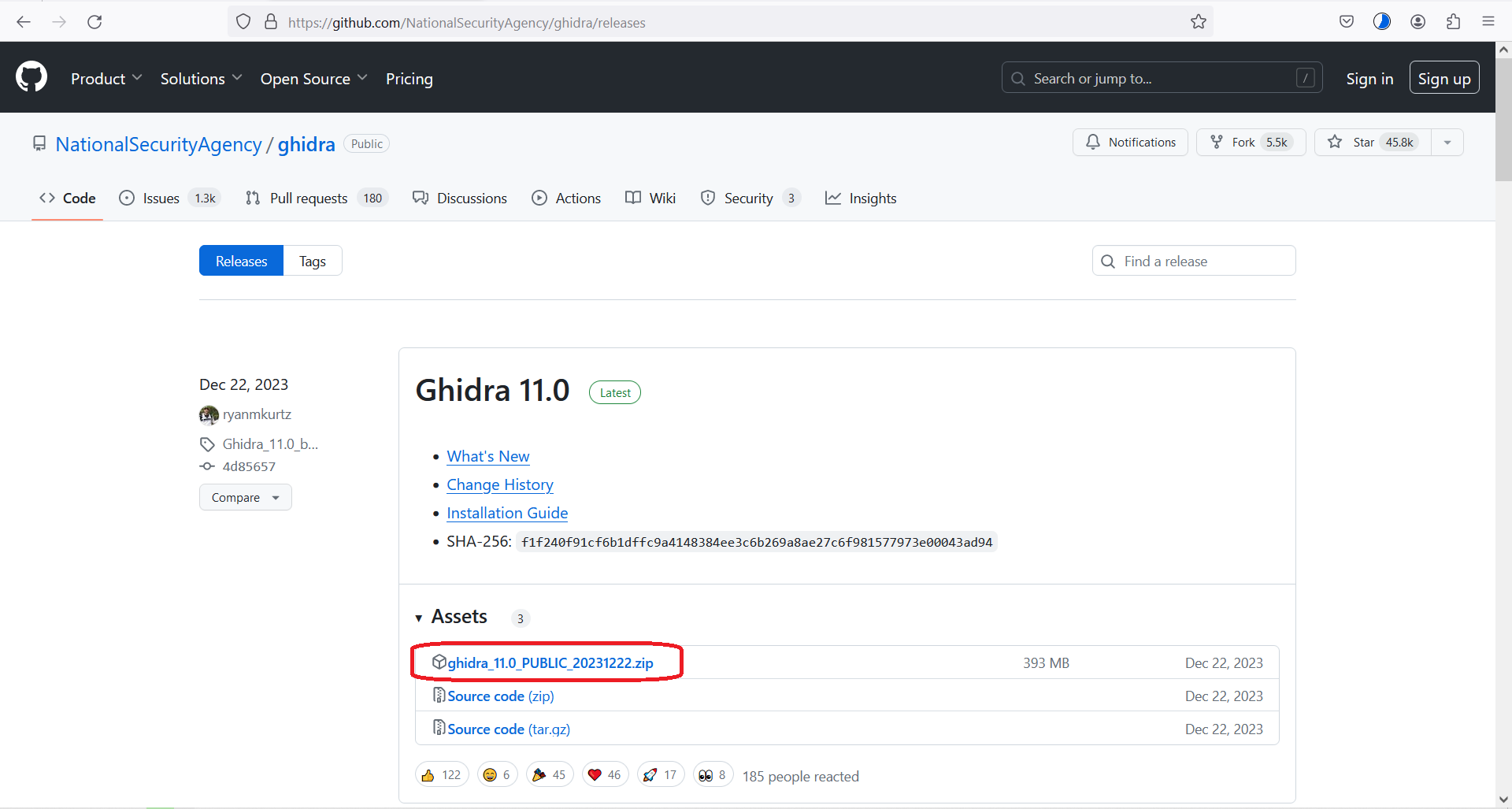


Рисунок 3.1 – Необходимый файл

После загрузки необходимо распаковать архив и перейти в папку с программой. Следующим шагом будет установка JDK [5], если она ещё не имеется на вашем ПК. Для этого необходимо загрузить файл, указанный на рисунке 3.2. После загрузки распаковать архив в удобную директорию. Как результат, JDK будет находиться в ПК и у нас появится возможность работы с GHIDRA. Но перед этим необходимо указать, где находится наша JDK. Для этого будем использовать переменные среды. Для начала, необходимо перейти в дополнительные параметры системы. Сделать это можно, нажав ПКМ по кнопке «Пуск», выбрав «Система», что указано на рисунке 3.3, и перейдя в соответствующий раздел, указанный на рисунке 3.4.

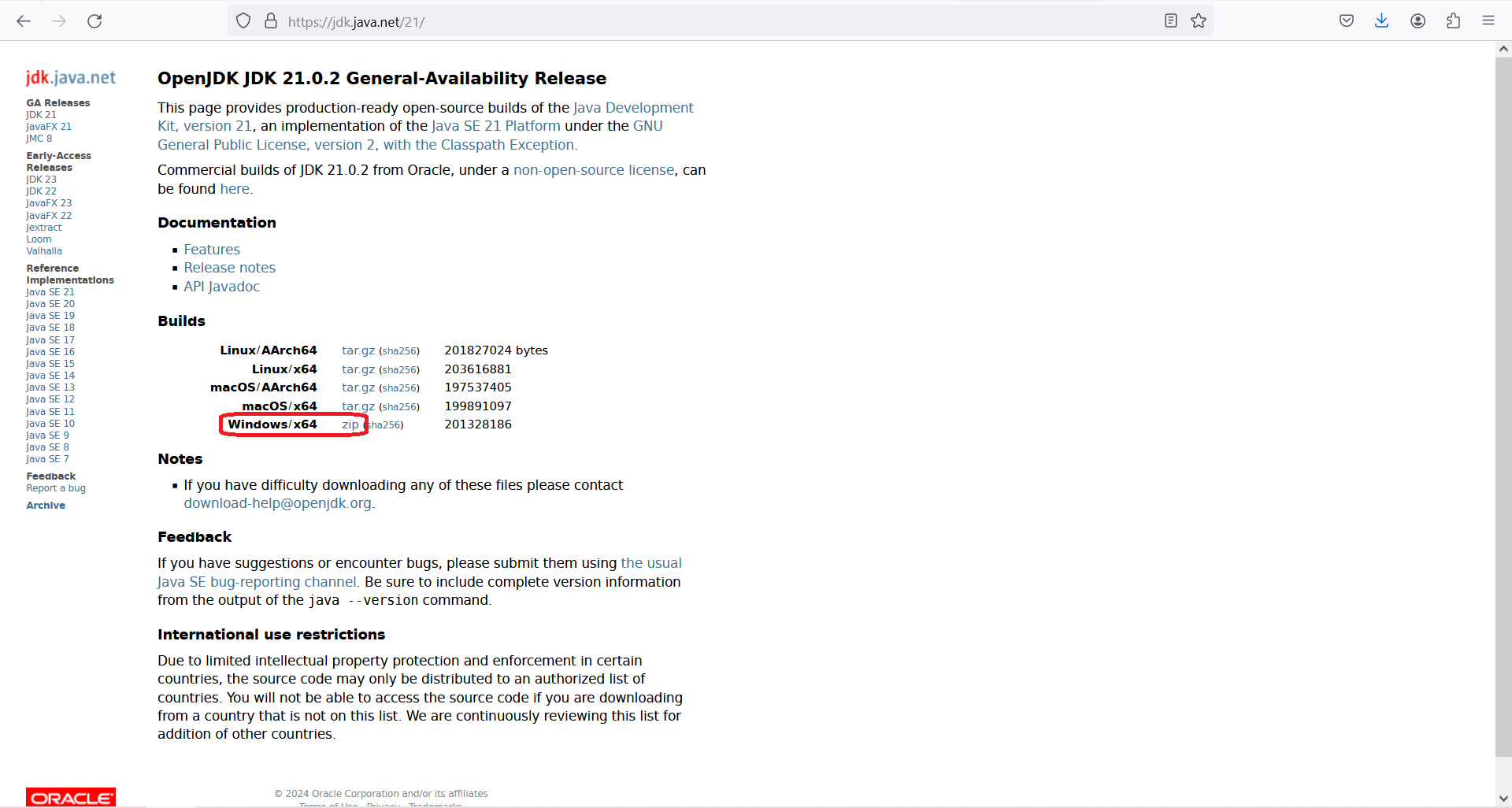


Рисунок 3.2 – Страница загрузки

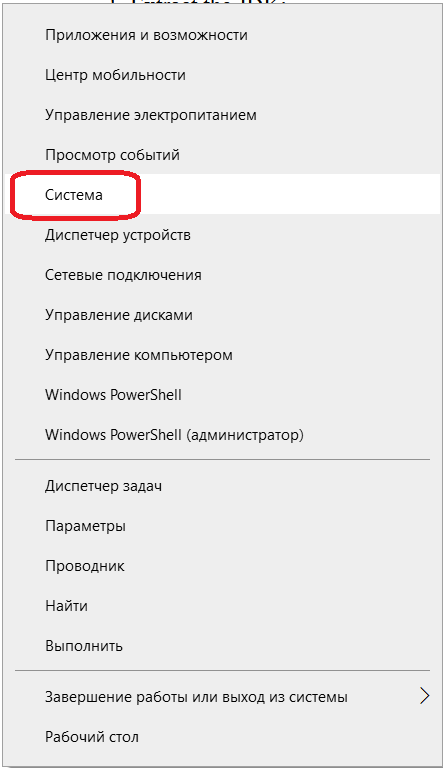


Рисунок 3.3 – Выбор подменю кнопки «Пуск»

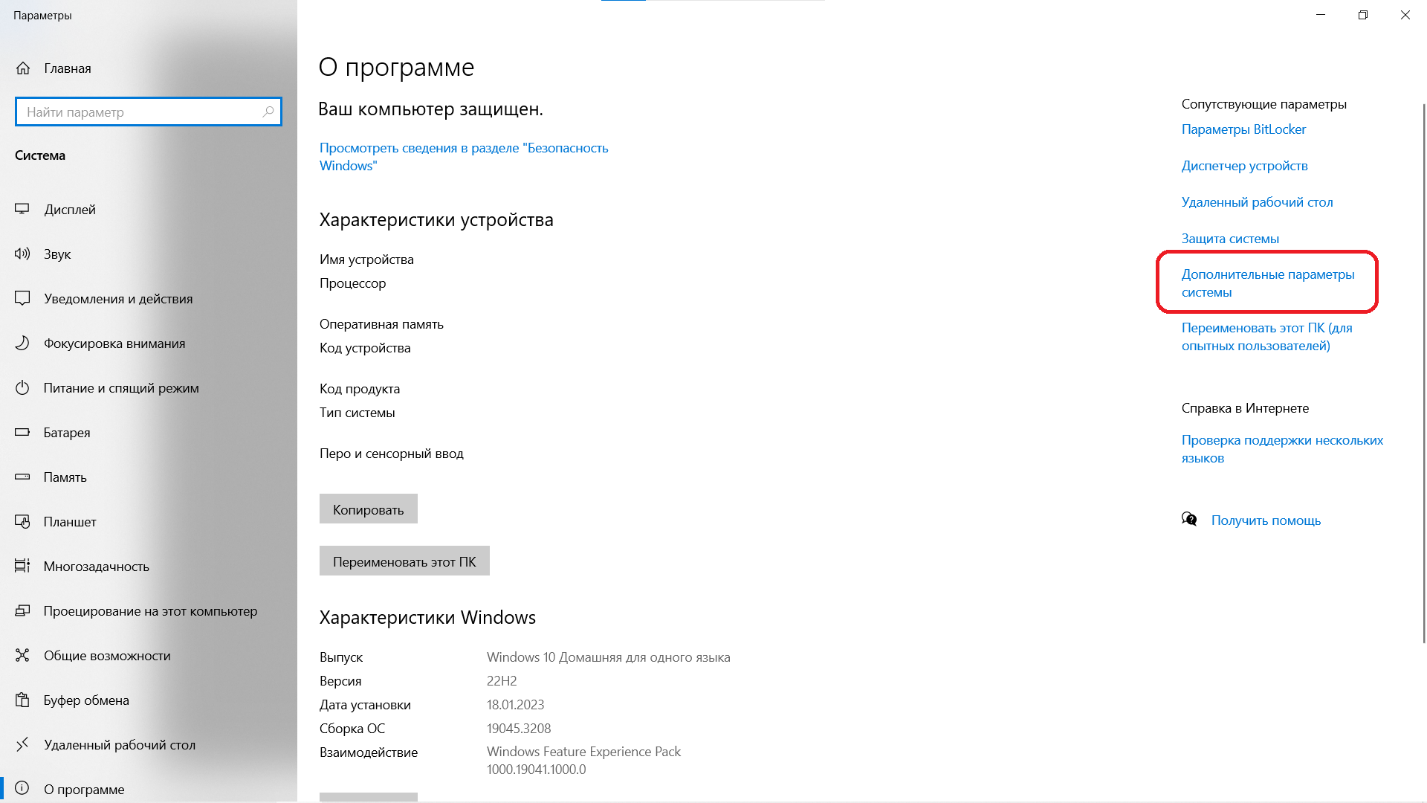


Рисунок 3.4 – Расположение меню дополнительных параметров

Далее – выбор меню переменных среды. Это показано на рисунке 3.5.

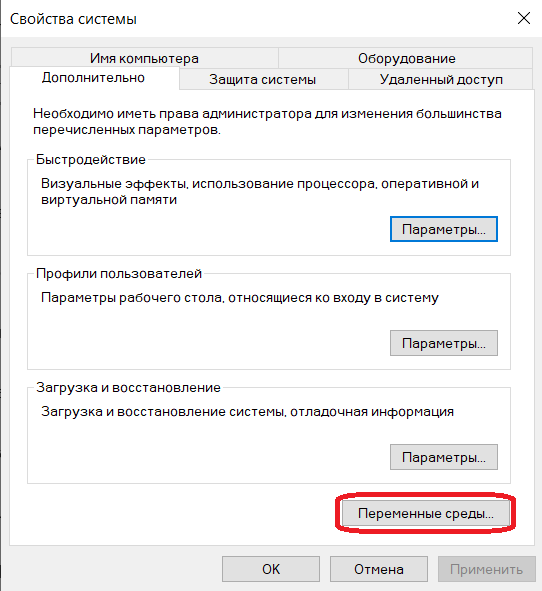


Рисунок 3.5 – Подменю «Переменные среды»

Теперь необходимо выбрать внизу окна переменную Path, после чего нажать «Изменить». Процесс на рисунке 3.6.

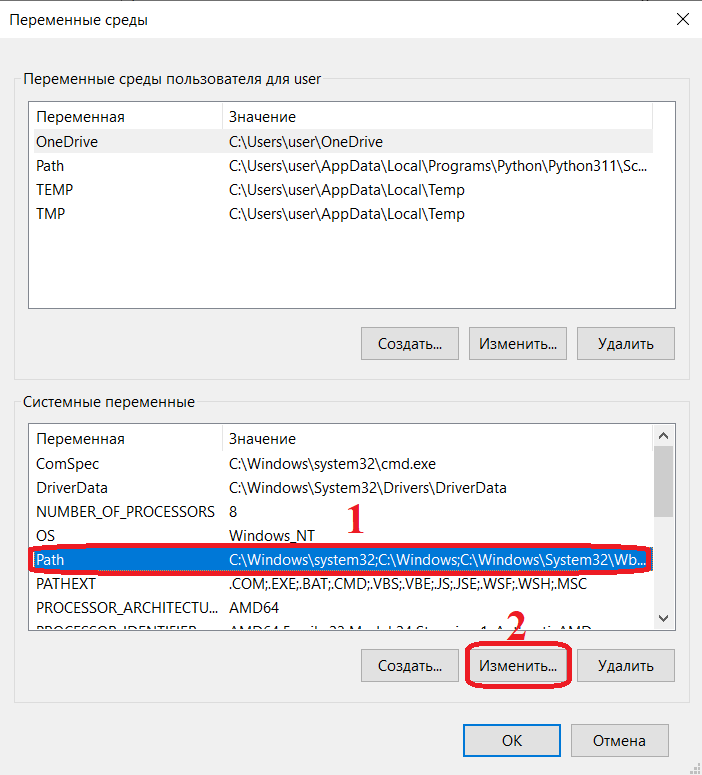


Рисунок 3.6 – Необходимая переменная среды

Далее, в появившемся окне требуется нажать «Создать». Появится возможность добавить строку, в которую необходимо вписать путь к распакованному архиву JDK, а точнее, вложенной в него папке bin/. К примеру, должен получится путь «C:\Users\user\Downloads\jdk-21.0.2\bin». Затем необходимо нажать Enter, после – кнопку «ОК». Процесс формализован на рисунке 3.7.

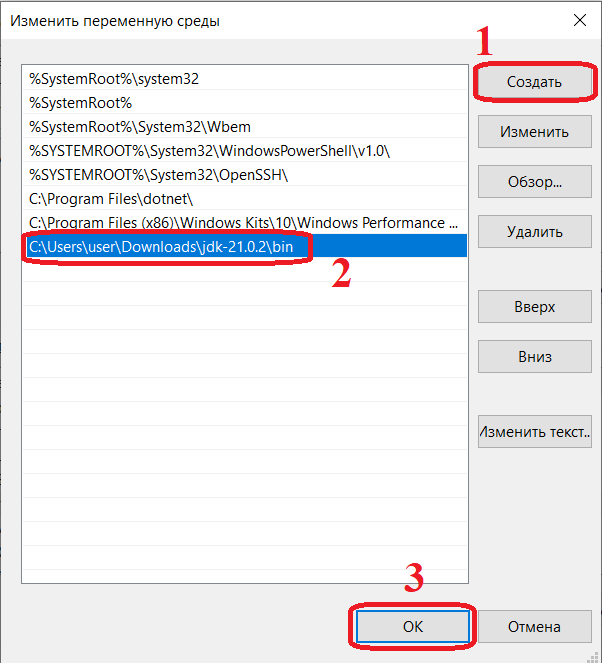


Рисунок 3.7 – Процесс добавления переменной среды

Когда предыдущие этапы выполнены, можно начать работу с GHIDRA. Для этого необходимо перейти в папку с программой, после чего запустить файл ghidraRun.bat, что показано на рисунке 3.8.

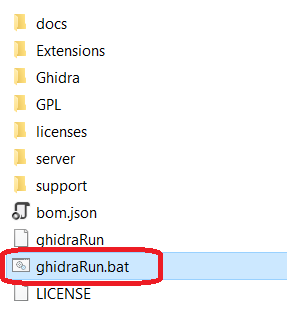


Рисунок 3.8 – Необходимый файл

После запуска необходимо прочесть соглашение и принять его, показано на рисунке 3.9.

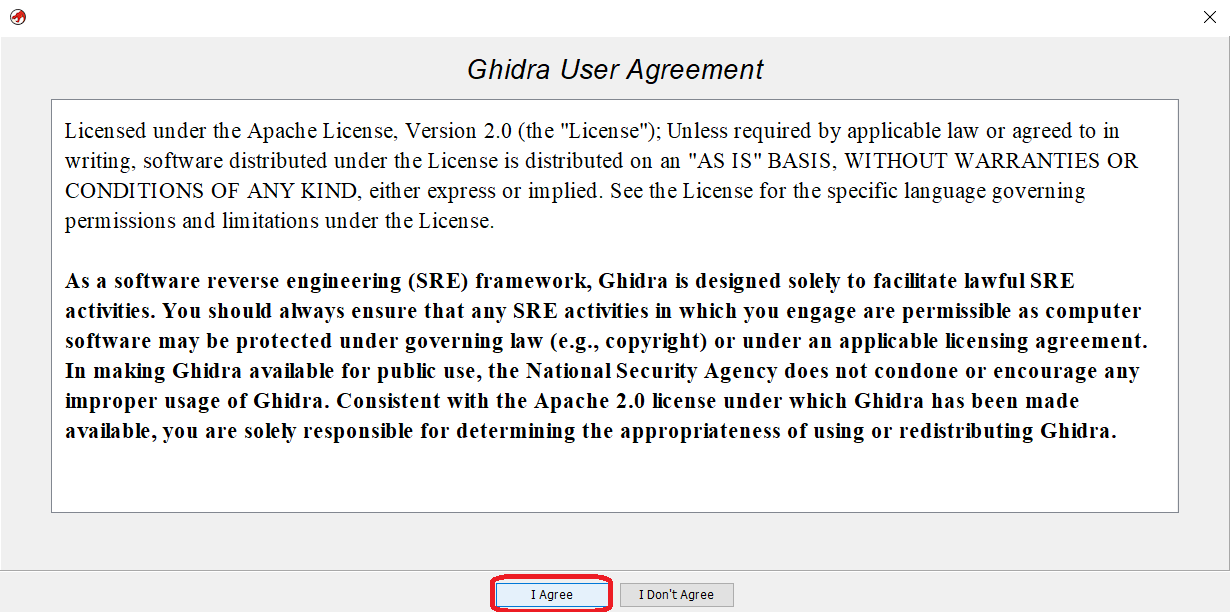


Рисунок 3.9 – Пользовательское соглашение

После соглашения откроется интерфейс помощи, где можно подробно прочесть про функционал программы, и интерфейс выбора проекта.

В интерфейсе проекта необходимо создать новый проект, что указано на рисунке 3.10, 3.11, 3.12. Для проекта рекомендуется выделить отдельную папку во избежание неразберихи с файлами. Для практики реверс‑инжиниринга выберем файл [6] с сайта‑сборника тренировочных проектов. Пароль для скачанного архива – «crackmes.one». Получим один файл, который при запуске просит ввести нас пароль. Произведём ввод наугад. Как видим (рисунок 3.13), программа не приняла пароль.

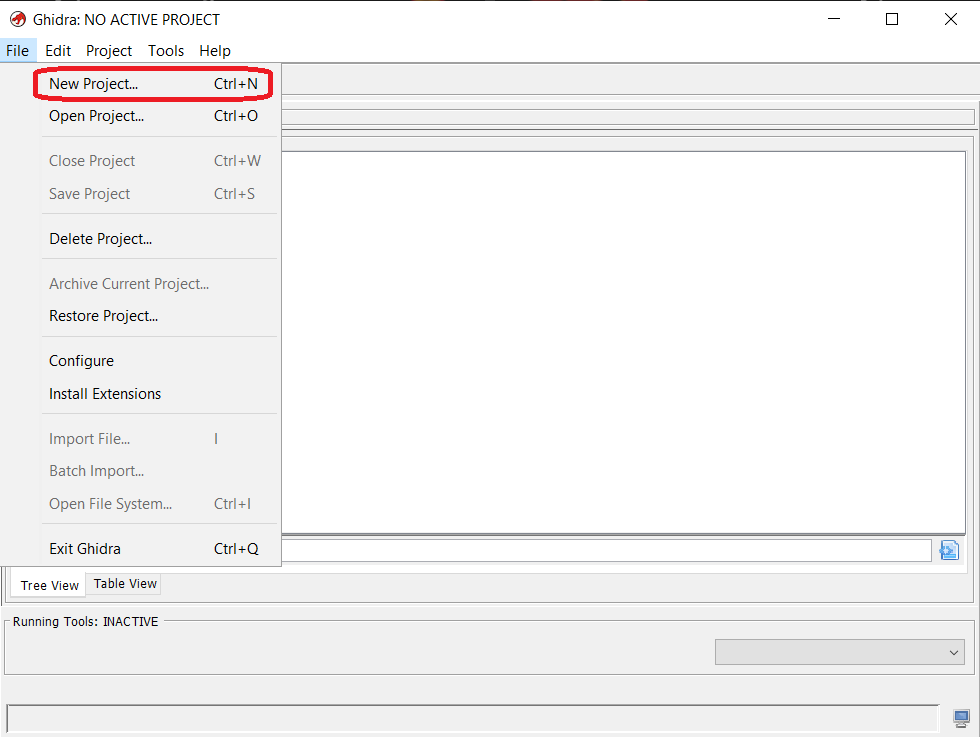


Рисунок 3.10 – Меню создания проекта

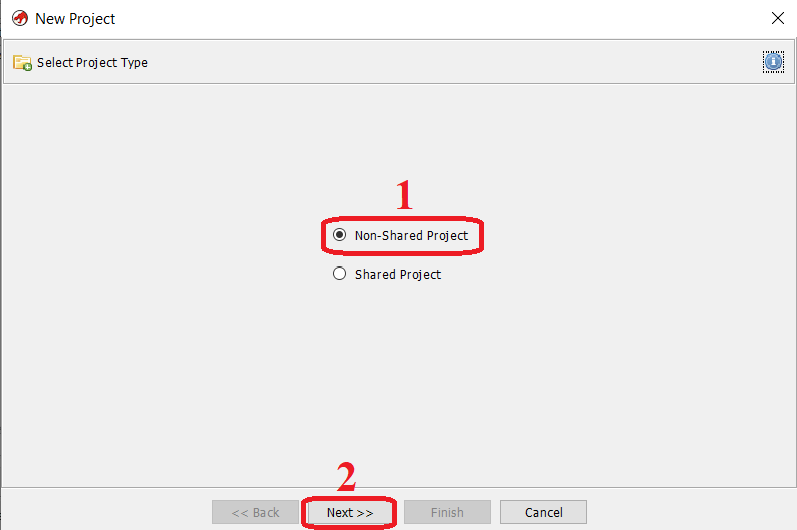


Рисунок 3.11 – Меню выбора режима (одиночный, общий)

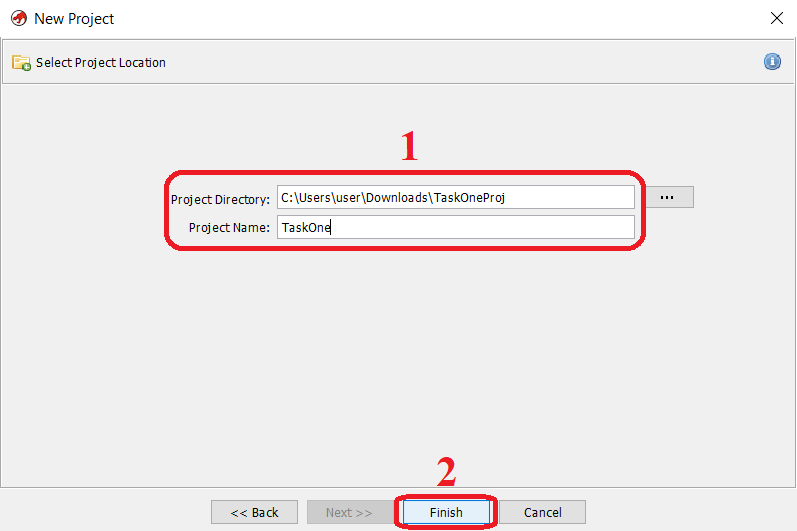


Рисунок 3.12 – Выбор директории проекта

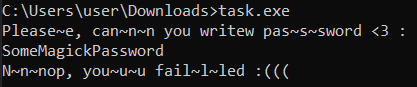


Рисунок 3.13 – Неверный пароль

Откроем этот файл в проекте. Для этого в окне GHIDRA выберем «Import file», после выберем нашу программу. Расположение указано на рисунке 3.14.

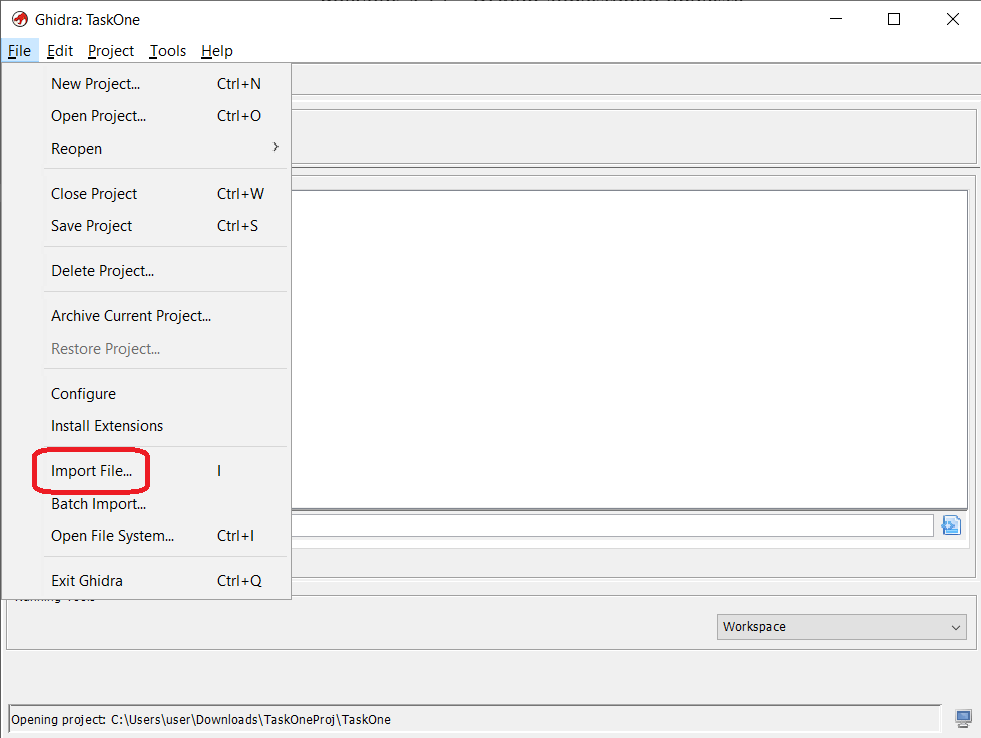


Рисунок 3.14 – Меню импорта файла в проект

Во время импорта файла настройки для анализа рекомендуется оставить по умолчанию. После импорта откроется окно с результатами анализа, где можно ознакомится с различной информацией касаемо загруженного файла. Это отображено на рисунке 3.15. Для дальнейшего анализа необходимо запустить основной инструмент GHIDRA – анализ кода. Для этого требуется нажать на иконку зелёного дракона в папке с проектом (рисунок 3.16).

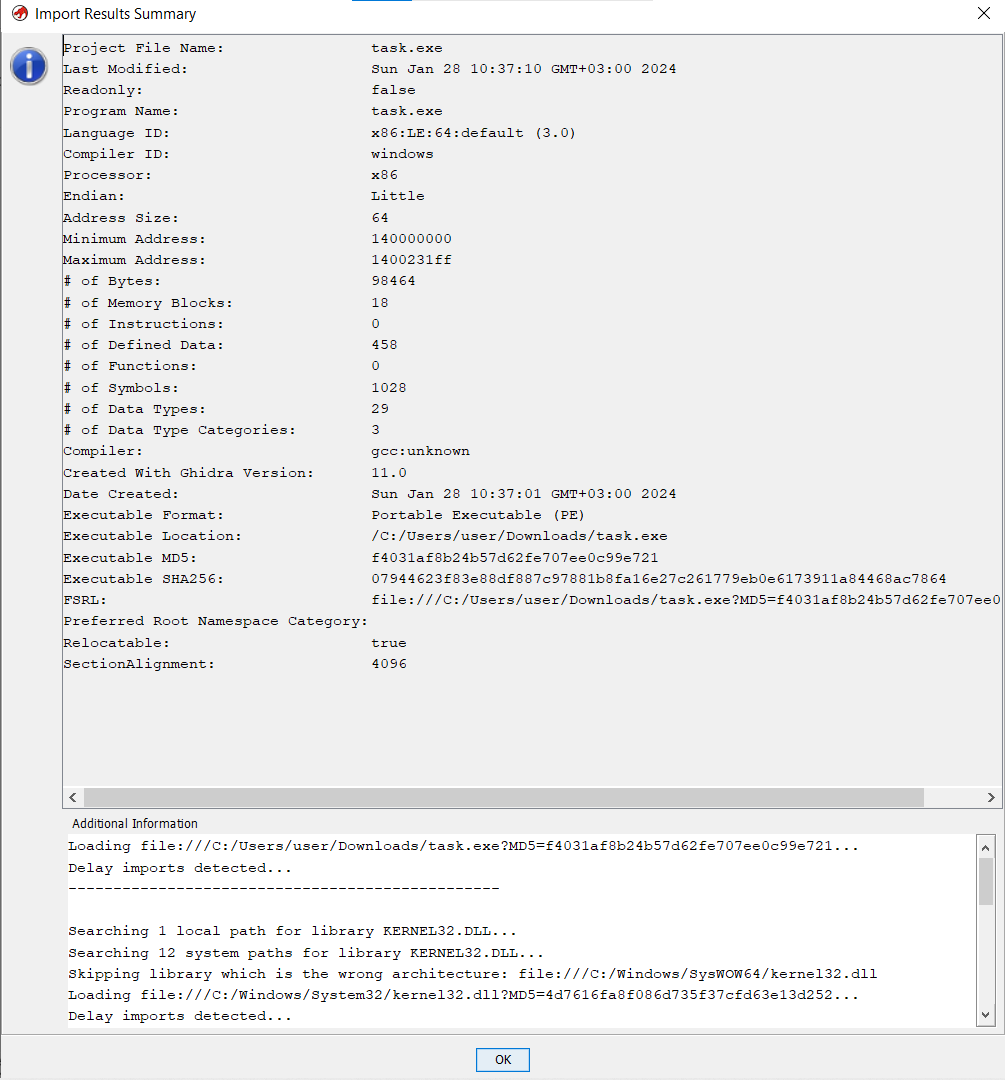


Рисунок 3.15 – Основная информация о файле

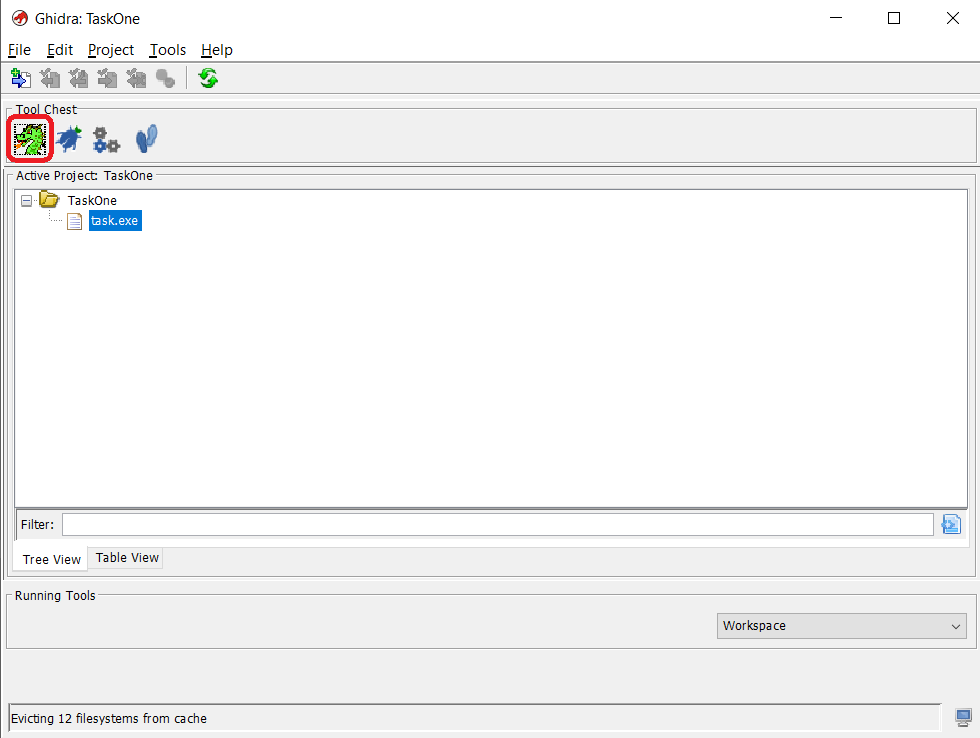


Рисунок 3.16 – Запуск анализа кода

Далее необходимо открыть наш файл. Для этого выберете соответствующий файл в пункте на рисунке 3.17.

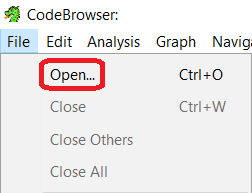


Рисунок 3.17 – Выбор файла для анализа

После выбора программа сообщит, что файл ещё не был проанализирован. Чтобы проанализировать файл, выберете «Yes» на рисунке 3.18. Затем появится окно, в котором можно выбрать, что конкретно необходимо проанализировать в файле. Есть три основных пункта: 1 – общий список опций, 2 – описание, 3 – выбор дополнительных настроек. Всё меню показано на рисунке 3.19. Для начала оставим все пункты так, как есть.

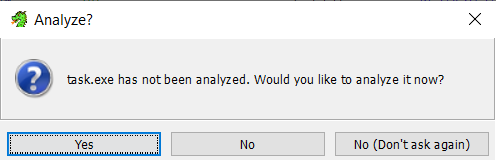


Рисунок 3.18 – Решение об анализе файла

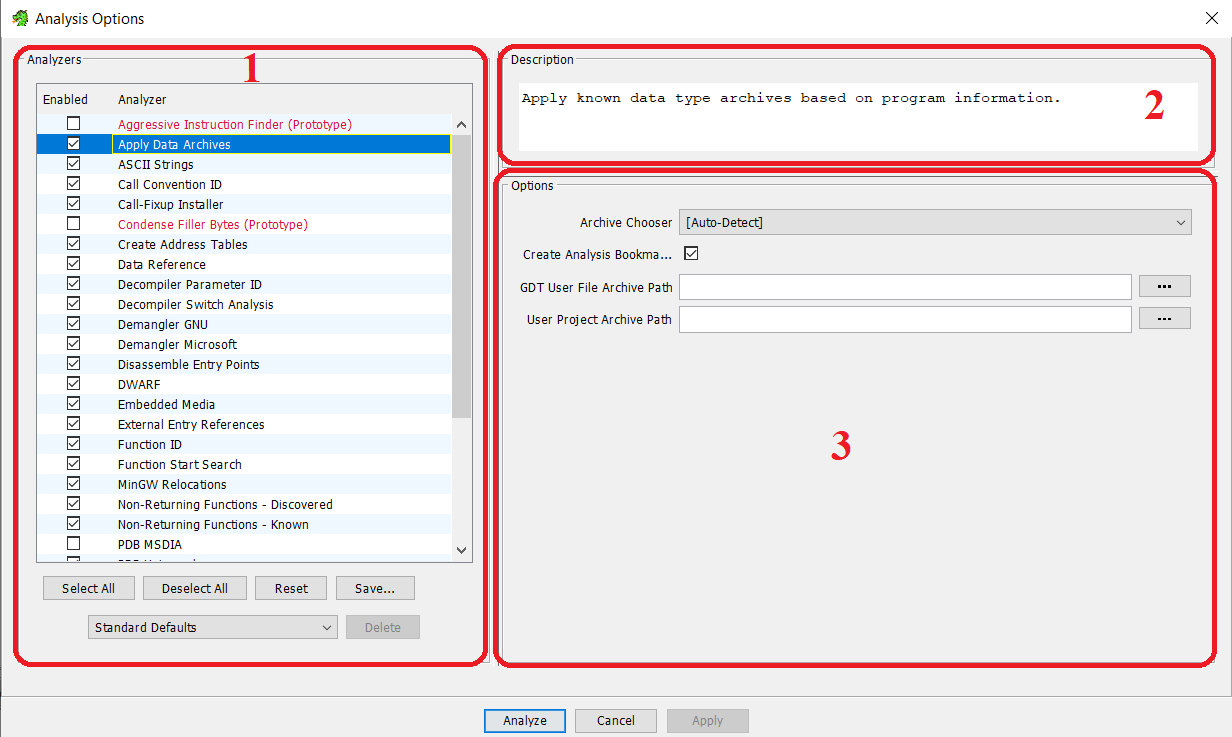


Рисунок 3.19 – Опции анализа

После того, как завершится анализ, откроется основное окно с инструментами отладки. Оно показано на рисунке 3.20 и содержит следующие пункты: 1 – основная панель инструментов, 2 – основные блоки данных программы, 3 – дерево всех вхождений строк и быстрое перемещение по ним, 4 – управление типами данных из программы, 5 – меню листинга с дизассемблированным кодом, 6 – меню с декомпилированым кодом, 7 – консоль.

Попробуем воспользоваться некоторыми из меню для нахождения заветного пароля. Для начала выберем поиск строк. Для этого, как показано на рисунке 3.21, выберем пункт и позднее нажмём «Search», оставив настройки поиска по умолчанию.

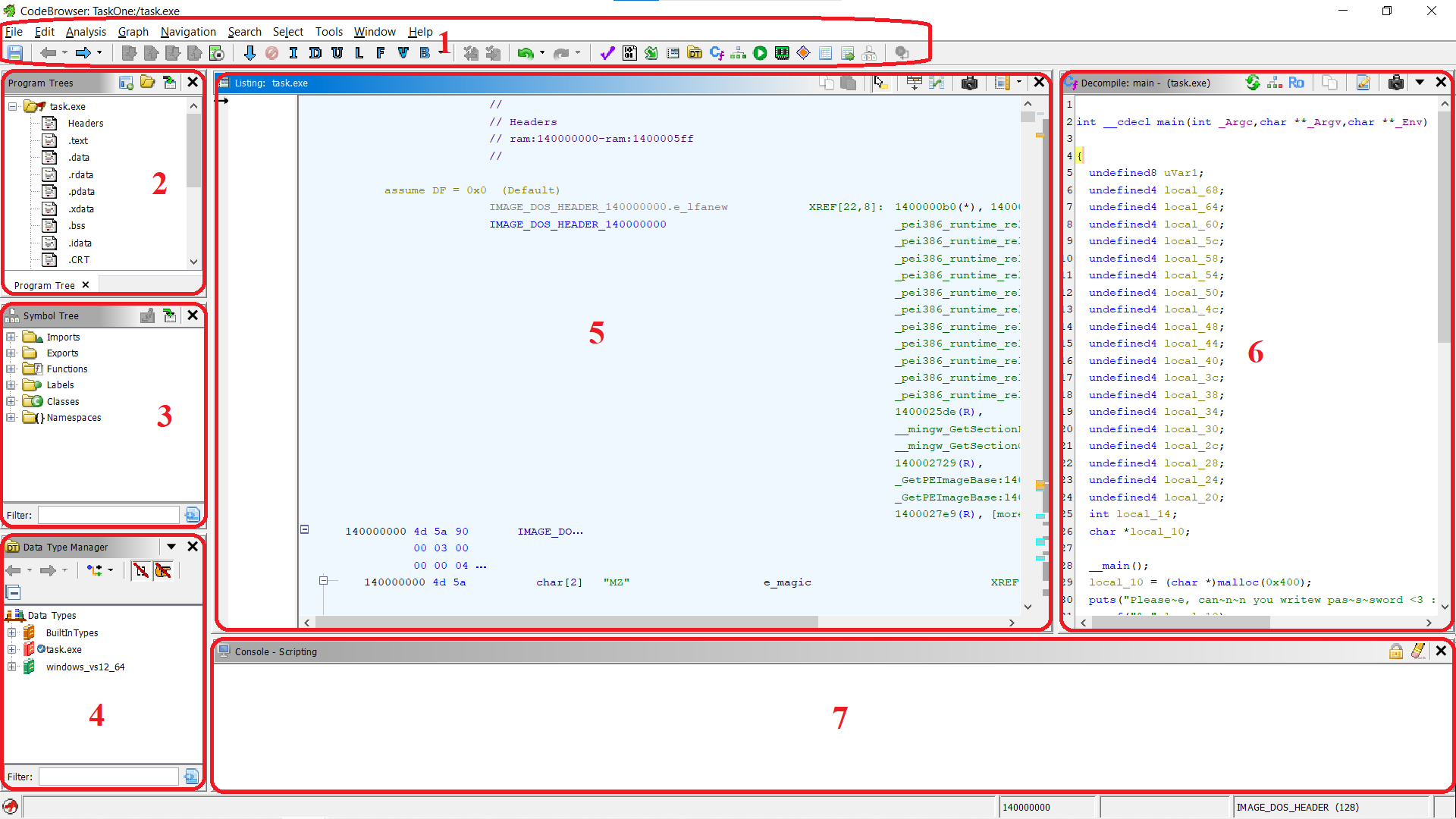


Рисунок 3.20 – Основной интерфейс после анализа

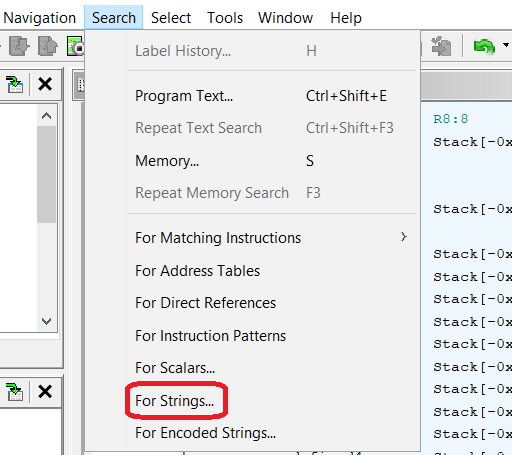


Рисунок 3.21 – Поиск строк в программе

После выполненного поиска получим окно, отражённое на рисунке 3.22, в котором легко заметить несколько интересных строк. Перейдём по адресу первой из интересующих нас строк, дважды кликнув по ней. Как можно видеть на рисунке 3.23, данная строка ссылается на 3 других адреса: неизвестный, и два адреса из функции main. Перейдём по первому адресу для функции main, также дважды нажав по нему.

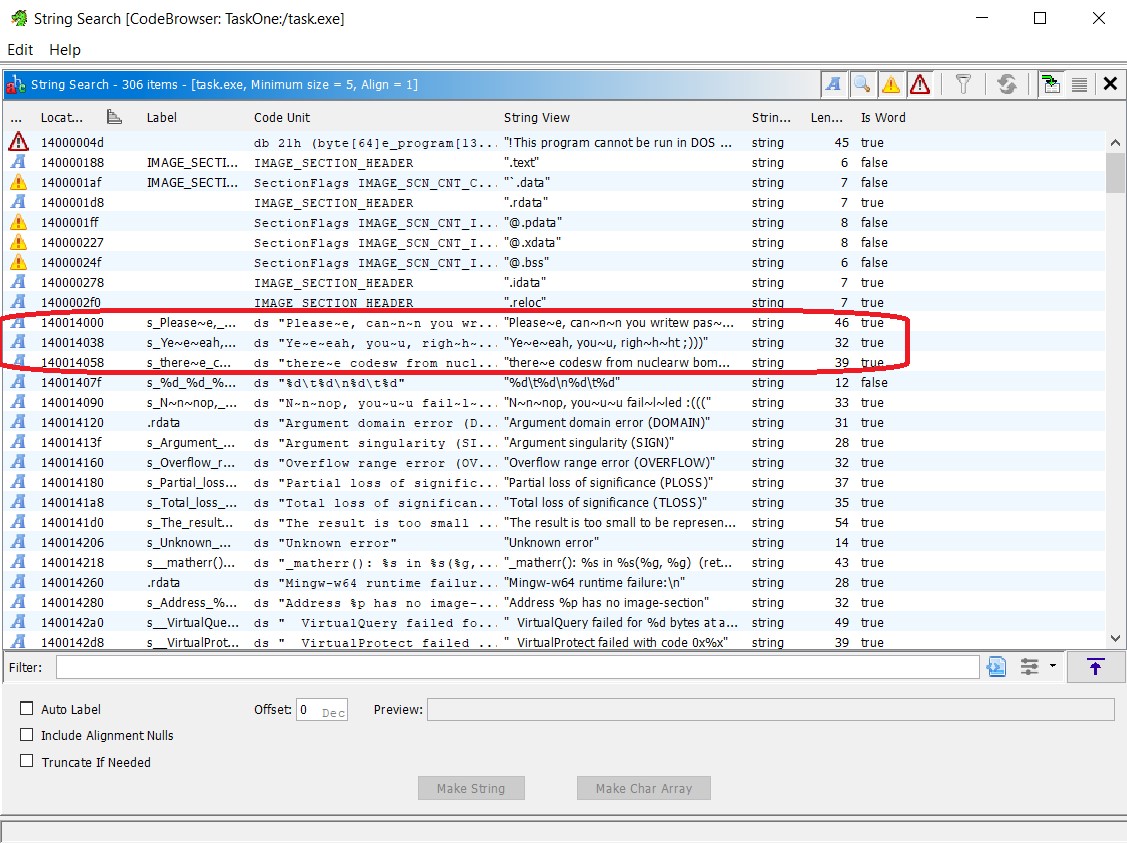


Рисунок 3.22 – Строки-метки

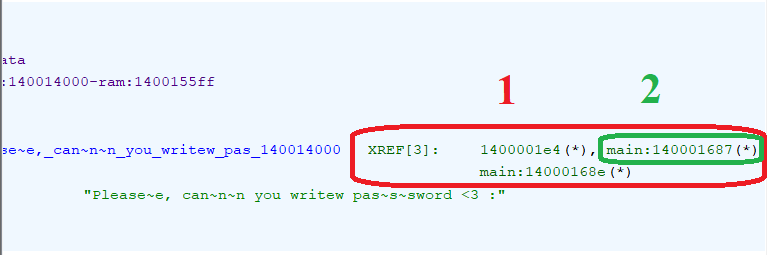


Рисунок 3.23 – Ссылки для данной строки

Теперь дважды нажмём на вызов какой-либо функции в дизассемблированном виде. Это позволит нам перейти к этому расположению в декомпилированном виде, что кратно упростит анализ кода. Место для нажатия указано на рисунке 3.24.



Рисунок 3.24 – Функция, по которой можно перейти

Посмотрим декомпилированный код на рисунке 3.25. Здесь можно увидеть несколько интересных блоков. Так, после вывода строки на экран с помощью функции puts (1), у пользователя запрашивается ввод в переменную local\_10 функцией scanf (2). В следующем блоке (3) объявляется некоторый список постоянных значений. Затем, в блоке (4) происходит вызов функции cmp\_pass, и перевод её результата в булевый вид (true, false). Наконец, последний блок (5), в зависимости от значения переменной прошлого блока, выполняет ветвление и выводит строки на экран, одну в случае успешной проверки, и другую иначе. После этого программа завершается.

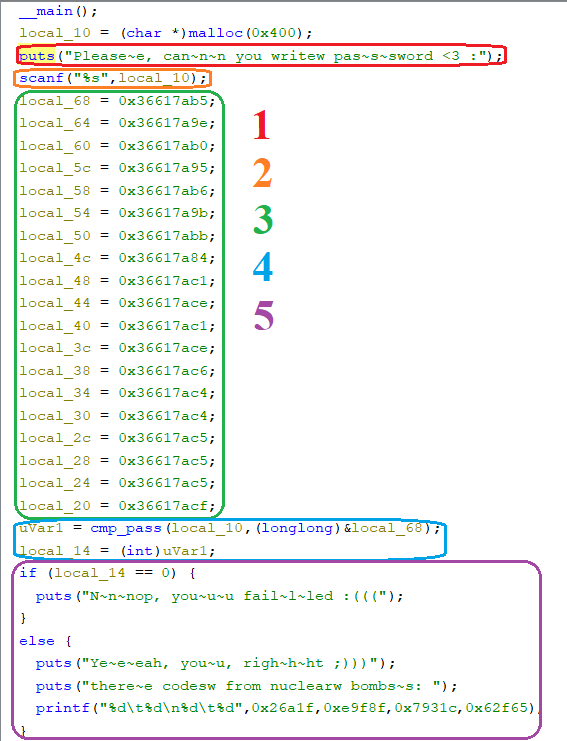


Рисунок 3.25 – Блок функции main

Посмотрим детальнее функцию cmp\_pass. Как видим, в неё передаются введённые пользователем данные и адрес первой переменной блока (2). Перейдём внутрь функции, что отражено на рисунке 3.26. Сначала данная функция подсчитывает число символов в введённой пользователем строке, и если оно не равно 19 (0x13 в шестнадцатеричной системе), то моментально возвращается 0. В противном случае выполняется цикл, в котором возвращается 0 в случае несоответствия. Следует обратить внимание, что, не меняя настроек оформления, один символ из строки с условием переносится на следующую строку.

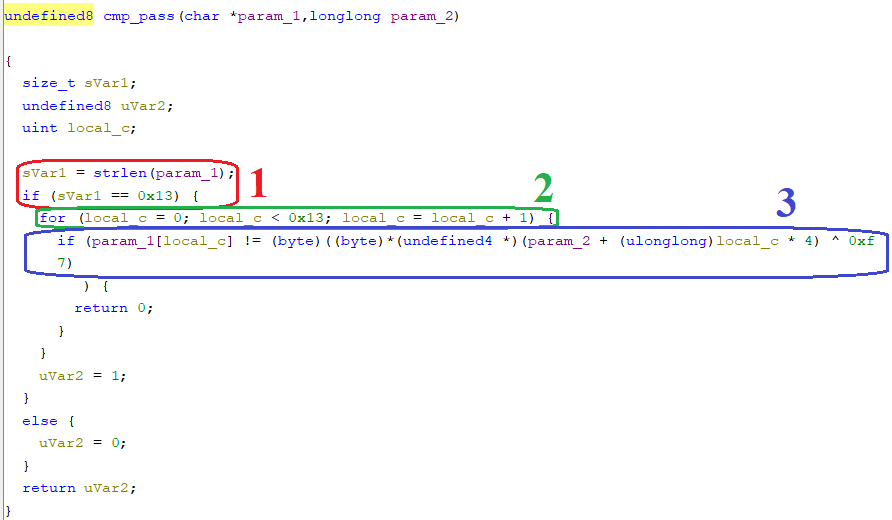


Рисунок 3.26 – Функция cmp\_pass

Подробно разберём условие. В нём выполняется проверка на неравенство символа, введённого пользователем, с неким символом из памяти. Этот символ получается, как байт памяти со смещением по циклу, с применённой к нему операцией XOR (^). Затем полученное преобразуется в байт и подаётся на сравнение. Разберём подробнее этот механизм. Так как мы имеем порядок записи little endian, что можно было заметить на рисунке 3.15, то байты мы должны брать в обратном порядке, с конца. Этот список был виден ранее, но повторим его на рисунке 3.27.

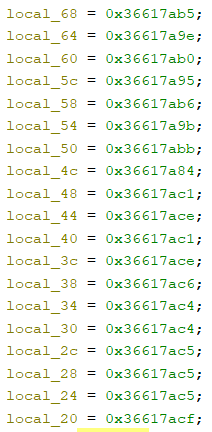


Рисунок 3.27 – Данный программы

Заметим, что значение цикла на каждой итерации умножается на 4, что означает смещение на 4 байта вверх по памяти (в данном случае вниз, из-за обратной записи). Возьмём первые данные. Это будет строка 0x366617AB5. Программа применяет операцию XOR с F7 лишь к последнему байту, давайте проделаем это на рисунке 3.28. Для удобства воспользуемся калькулятором. Как результат получим код символа ASCII 0x42, и по таблице соответствия [7] на рисунке 3.29 найдём нужный нам символ. В данном случае, это английская заглавная «B». Выполнив смещение на 4 байта, получим следующие исходные данные: 9E. Снова проделав XOR F7 получим 0x69, что означает английский символ «i». Проделаем операцию до конца.



Рисунок 3.28 – Операция XOR

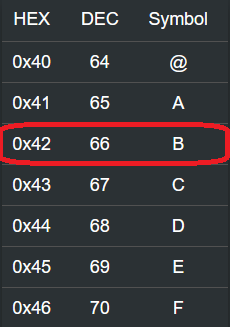


Рисунок 3.29 – Таблица соответствия

По результату операции получим строку «BiGbAlLs69691332228». Попробуем ввести её в нашу программу в качестве пароля. На рисунке 3.30 отражён результат ввода. Как видим, мы успешно обнаружили пароль.

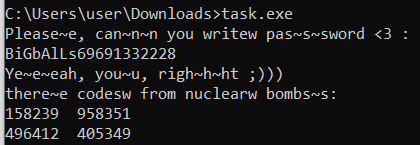


Рисунок 3.30 – Результат с «найденным» паролем

В данном разделе рассмотрены базовые инструменты, предоставляемые GHIDRA для отладки программ. Следует помнить, что данный инструмент является легко расширяемым. Это позволяет значительно увеличить число доступных инструментов, которые и без дополнительных модулей имеют необходимый для отладки функционал. Многое было оставлено без внимания с целью самостоятельного изучения данного инструмента. Как итог, данное ПО позволяет, при должном уровне знаний, обходить защиту вредоносных приложений и исследовать их на скрытые возможности и изъяны логики. Учитывая то, что GHIDRA является бесплатным решением с открытым исходным кодом, и в силу всего вышесказанного, данная среда позволяет полноценно заменить коммерческие решения.

# Применение инструмента x64dbg

Инструмент x64dbg является программой для отладки. Он является решением с открытым исходным кодом. Данная программа подходит для отладки приложений на платформе Windows x32 и Windows x64. Отладчик имеет понятный интерфейс и множество функций, среди которых имеется поддержка плагинов и скриптовых функций.

Подробное описание возможностей программы, равно как и ссылки для её установки, находятся на домашней странице (<https://x64dbg.com/>). Установка производится путём скачивания и распаковки архива с программой. В директории необходимо найти папку release, где предоставлена возможность выбора необходимой версии. Запуск x96dbg.exe приведёт к установке обоих версий на ПК в виде приложений. Запустить версии программы можно без установки по отдельности. Для этого перейдите в необходимую директорию (x32, x64) и запустите в ней соответствующий файл (x32dbg.exe, x64dbg.exe). После этого откроется сама программа.

Для отладки выберем программу [8] с сайта crackmes.one. Запустим её и введём произвольные данные. Как видим, результат нашей попытки войти отрицательный (рисунок 3.31).

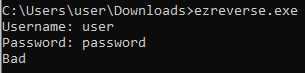


Рисунок 3.31 – Попытка ввода пароля

Запустим отладку, выбрав в x64dbg подменю Файл, а далее – Открыть (рисунок 3.32). Как видим, перед нами открывается множество функциональных областей. Разберём их предназначение более подробно. Сами области отражены на рисунке 3.33.

Первая область – элементы управления ходом программы и анализом. Вторая отвечает за открытие дополнительных окон программы. Третья – основная область, содержащая дизассемблированный код программы. Четвёртая содержит данные о содержимом регистров, а пятая – о содержимом стека. В шестой хранятся дампы памяти программы и прочие данные. Седьмой содержит информацию, а восьмой – командная строка, позволяющая ввести в программу разнообразные команды.

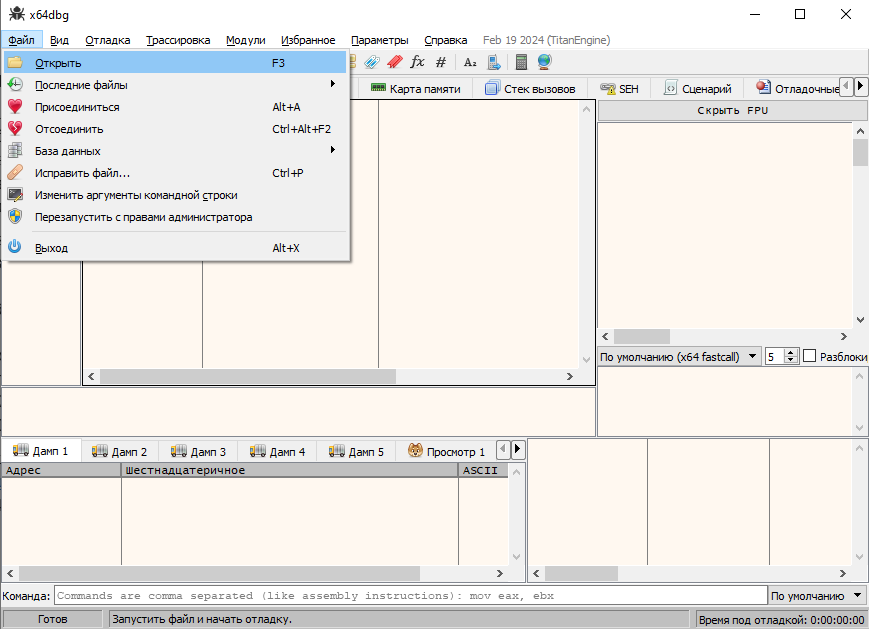


Рисунок 3.32 – Пункт меню для выбора файла

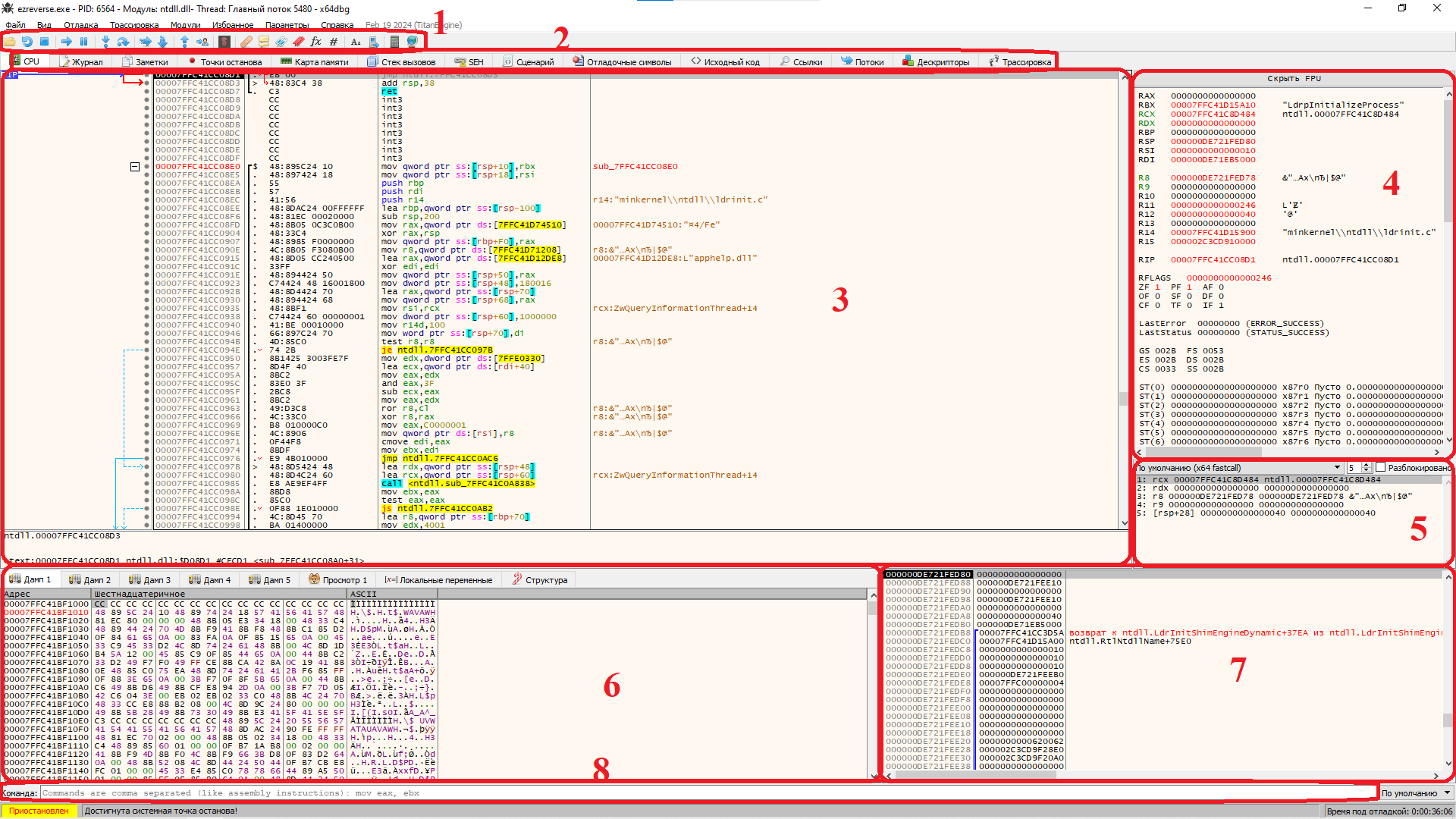


Рисунок 3.33 – Основное содержание программы

Теперь, когда мы разобрались с назначением основных элементов рабочей области, приступим к анализу программы.

Программа начинает своё выполнение с инициализации системных данных. Чтобы понять, какой модуль исполняется в текущий момент, достаточно взглянуть на название программы. На рисунке 3.34 видно, что сейчас идёт выполнение ntdll.dll – библиотеке, которая вызывает прерывания на системе с ядром NT. Чтобы выполнить модуль до следующего ключевого элемента, необходимо нажать соответствующую кнопку (рисунок 3.35), либо клавишу F9. Как видим, информация в окне обновилась на новую (рисунок 3.36), а значит можно найти строковые вхождения для оптимизации процессе (рисунок 3.37).



Рисунок 3.34 – Текущий исполняемый модуль

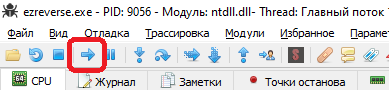


Рисунок 3.35 – Кнопка выполнения ключевого блока



Рисунок 3.36 – Новый исполняемый модуль

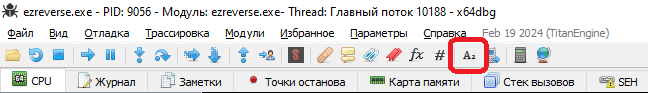


Рисунок 3.37 – Поиск строк в программе

После этого увидим, что открылся раздел ссылки. Здесь найдём интересующие нас строки и адрес, по которому они используются (рисунок 3.38). Перейдём по адресу строки «Password». Для этого нажмём ПКМ по адресу и выберем пункт меню (рисунок 3.39).

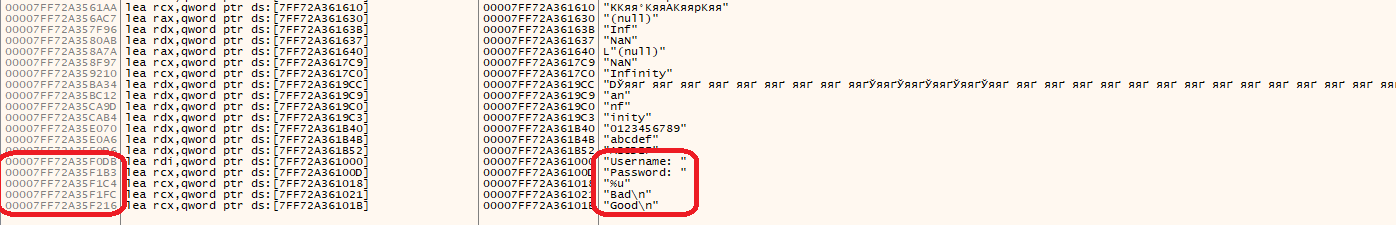


Рисунок 3.38 – Строковые вхождения

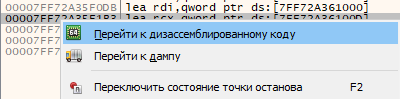


Рисунок 3.39 – Переход по адресу строкового вхождения

После перехода поставим точку останова по произвольному адресу чуть дальше строки. Для этого нажмём ЛКМ по какой-либо строке, после чего нажмём F2. Адрес строки окрасится в красный цвет (рисунок 3.40)



Рисунок 3.40 – Выставление точки останова

После этого выполним несколько раз программу, до тех пор, пока она не запросит нас ввести имя и пароль. Введём произвольные значения: 123 в качестве имени, и 321 в качестве пароля. Затем снова перейдём к дизассемблированному виду. Рассмотрим назначение действий в цикле на рисунке 3.41.

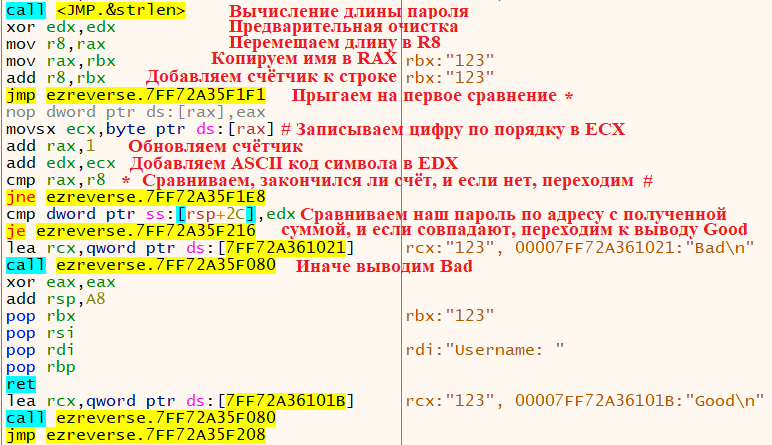


Рисунок 3.41 – Описание механизма проверки

Можно посмотреть переходы более детально, построив граф программы по нажатию кнопки G, вводу в поле 8 команды «graph», или через контекстное меню. Пример графа для данного участка кода предоставлен на рисунке 3.42.



Рисунок 3.42 – Граф механизма проверки

Теперь, обладая данными знаниями, попробуем обойти проверку пароля. Для этого в качестве имени введём «usr», а в качестве пароля сумму ASCII‑символов имени (u – 117, s – 115, r – 114), равную 346. Результат приведён на рисунке 3.43.



Рисунок 3.43 – Ввод полученных данных

Как видим, нам успешно удалось обойти проверку.

В результате знакомства с инструментом x64dbg можно утверждать, что в умелых руках это не только отладчик программ, но и полноценная среда для выполнения задач реверс‑инжиниринга. Инструмент обладает богатым функционалом, полностью ознакомиться с которым можно, как было сказано ранее, на официальном сайте. x64dbg даёт возможность качественно и быстро анализировать программы. Благодаря своим преимуществам рекомендуется использовать этот отладчик для решения задач реверс‑инжиниринга на системах под управлением Windows.

# Конвертация программного кода с одного языка программирования на другой

# Применение языковой модели с искуственным интеллектом в Telegram

В настоящее время для конвертации одного языка программирования в другой не существует универсального рабочего решения, за исключением написания кода с нуля. В данной части методического указания рассмотрим пример конвертации кода путём применения нейронных сетей. Конкретно в данном пункте воспользуемся ChatGPT v3.5 в мессенджере Telegram (<https://t.me/GPT4Telegrambot>).

Для того, чтобы перевести код, необходимо обратиться к боту с запросом «Перепиши код на C++» или с похожим, при этом выбор языка программирования не ограничен. Попробуем перевести простую программу, написанную на языке Python и выполняющую проверку на палиндромы, на язык C++. Сначала проверим работу программы на языке Python (Рисунок 4.1):

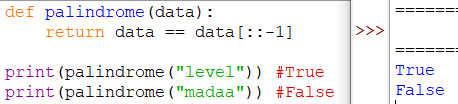


Рисунок 4.1 – Запуск кода программы на Python

Как видим, код работает именно так, как предполагается. Теперь попросим чат-бота переписать наш код на язык C++, и проверим, насколько хорошо он справится с этим (Рисунок 4.2).

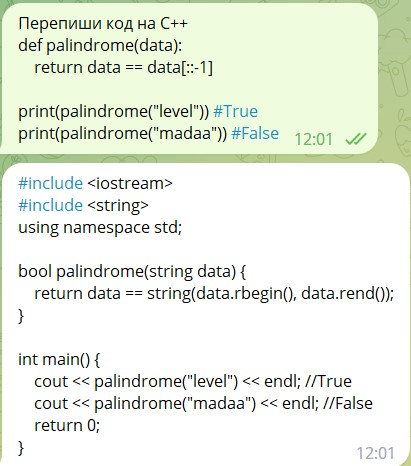


Рисунок 4.2 – Запрос боту на конвертацию кода

Выходной код загрузим в Visual Studio 2022, и там проверим его работоспособность (Рисунок 4.3).

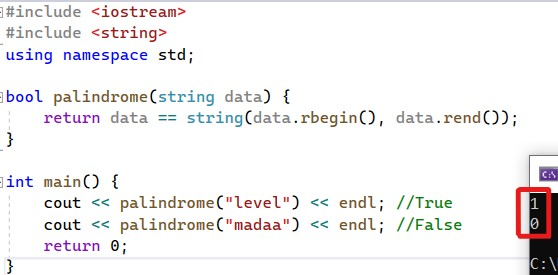


Рисунок 4.3 – Загрузка и выполнение сгенерированного кода на C++

Как можно видеть, программа работает, как и задумывалось. Это означает, что нейронная сеть смогла правильно конвертировать код.

Этот пример является достаточно простым, поэтому попробуем ещё два – перевести программу среднего и высокого класса сложности. В качестве «средней» возьмём консольную программу для игры в «виселицу», код приведён на рисунке 4.4. Проверим её работу на Python (Рисунок 4.5):

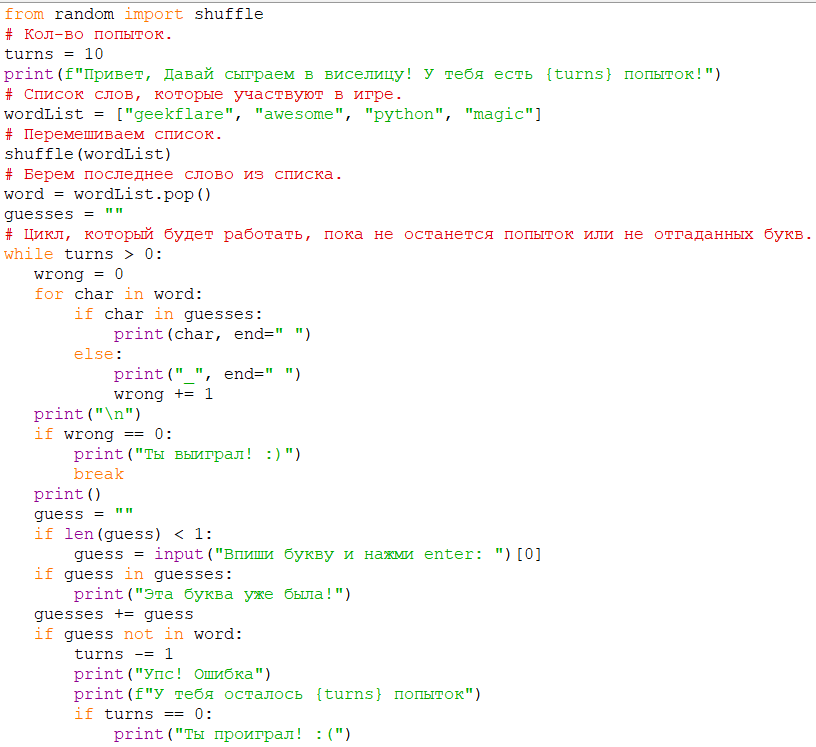


Рисунок 4.4 – Код «виселицы» на Python

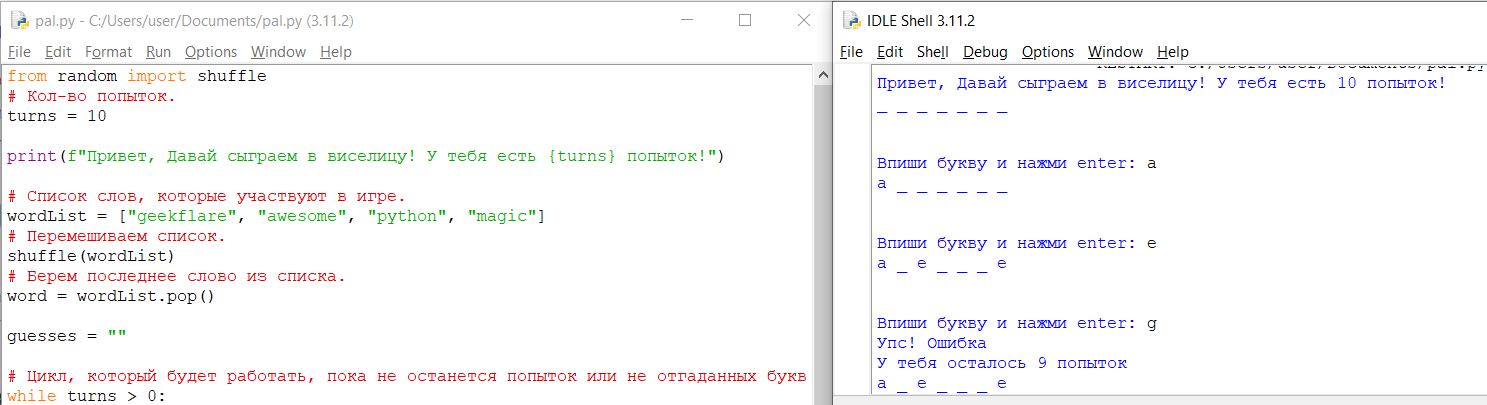


Рисунок 4.5 – Проверка кода «виселицы»

Как видим, код успешно выполняется. Теперь попросим ChatGPT переписать его. Результат запуска переписанной программы на рисунке 4.6.

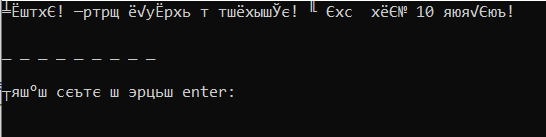


Рисунок 4.6 – Запущенная «виселица» на C++

Как можно видеть, появились первые проблемы. В данном случае проблема несерьёзная, так как просто отсутствует определение локализации. Попробуем исправить это с помощью соответствующей строки: setlocale(LC\_ALL, “ru”); После того, как добавим данную строку, попробуем снова запустить программу (Рисунок 4.7).

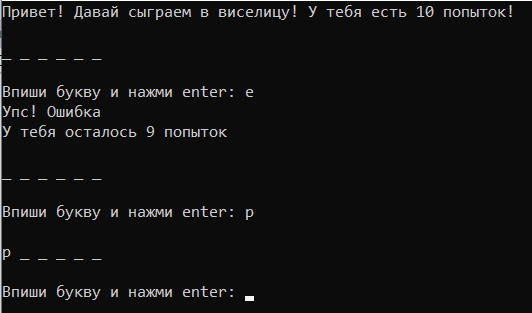


Рисунок 4.7 – Запуск отредактированной программы на C++

Как видим, тут уже потребовалось вносить правки в код, предоставленный сетью, для того чтобы получить желаемый результат.

Наконец, для «сложной» программы выберем ту, что работает с графикой. Для примера возьмём программу, которая рисует движущийся квадрат. Проверим её (Рисунок 4.8).

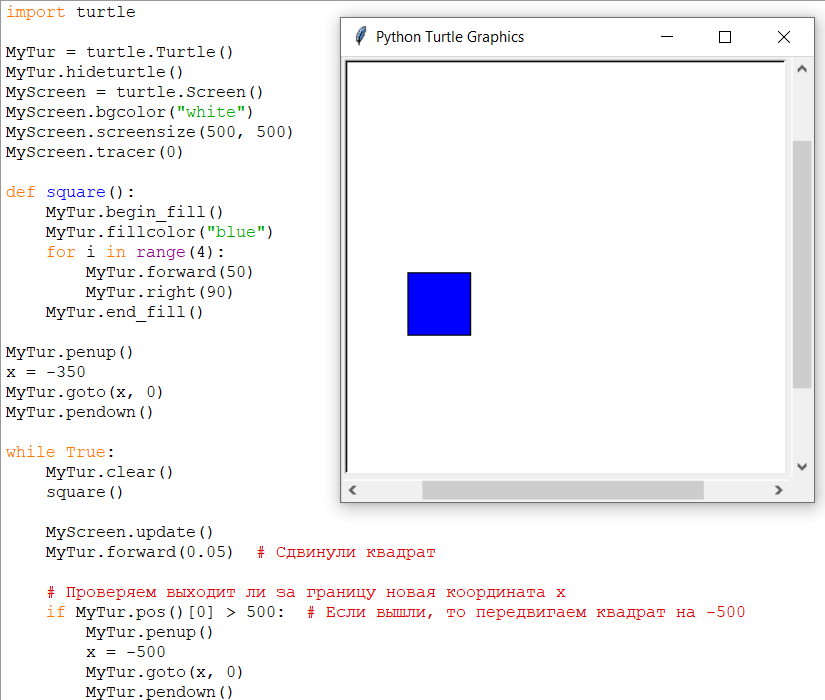


Рисунок 4.8 – Запуск «сложной» программы на Python

Попросим нейронную сеть снова переписать за нас код. Переписанный код приведён на рисунке 4.9.

Следует заметить, что при копировании кода компилятор, вероятнее всего, «пожалуется» на отсутствие необходимой библиотеки SFML. В данном же примере уже произведена её установка и настройка решения для использования этой библиотеки по инструкции с сайта <https://habr.com/ru/articles/703500/> (дата обращения – 22.10.23). В рамках данного пособия не станем рассматривать подробную настройку, так как это не является целью данного раздела.

После копирования кода и настройки необходимых библиотек, попробуем собрать наше решение. Как видим, эта попытка провалилась, и компилятор выдал нам множество ошибок (Рисунок 4.10). Таким образом, перевод кода с языка Python на язык С++ провалился, а конечный код требует глубокого анализа, понимания языка и серьёзных доработок.



Рисунок 4.9 – Переписанный на C++ код

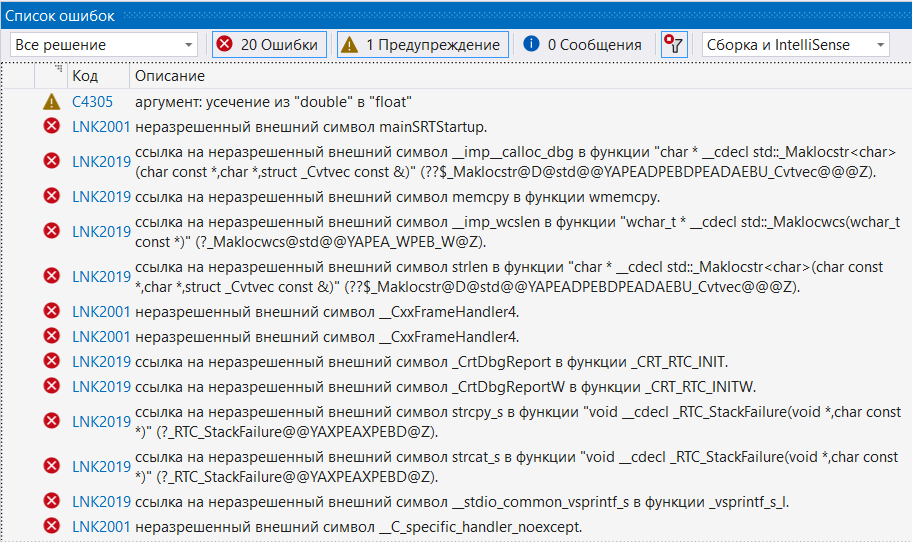


Рисунок 4.10 – Ошибки при попытке запуска скомпилированного кода на C++

# применение сервиса AI Code converter

В данном разделе рассмотрим возможность конвертации кода с помощью сервиса AI Code Converter. Данный сервис позволяет переводить код не только с языка программирования Python, но также и со множества других языков. В нашем случае попробуем перевести 3 программы из предыдущей главы в код на язык программирования C++.

Первой переведём программу, проверяющую, является ли слово палиндромом. Код программы и результат выполнения были приведены на рисунке 4.1. Теперь запросим сервер перевести этот код на язык C++ (Рисунок 4.11).

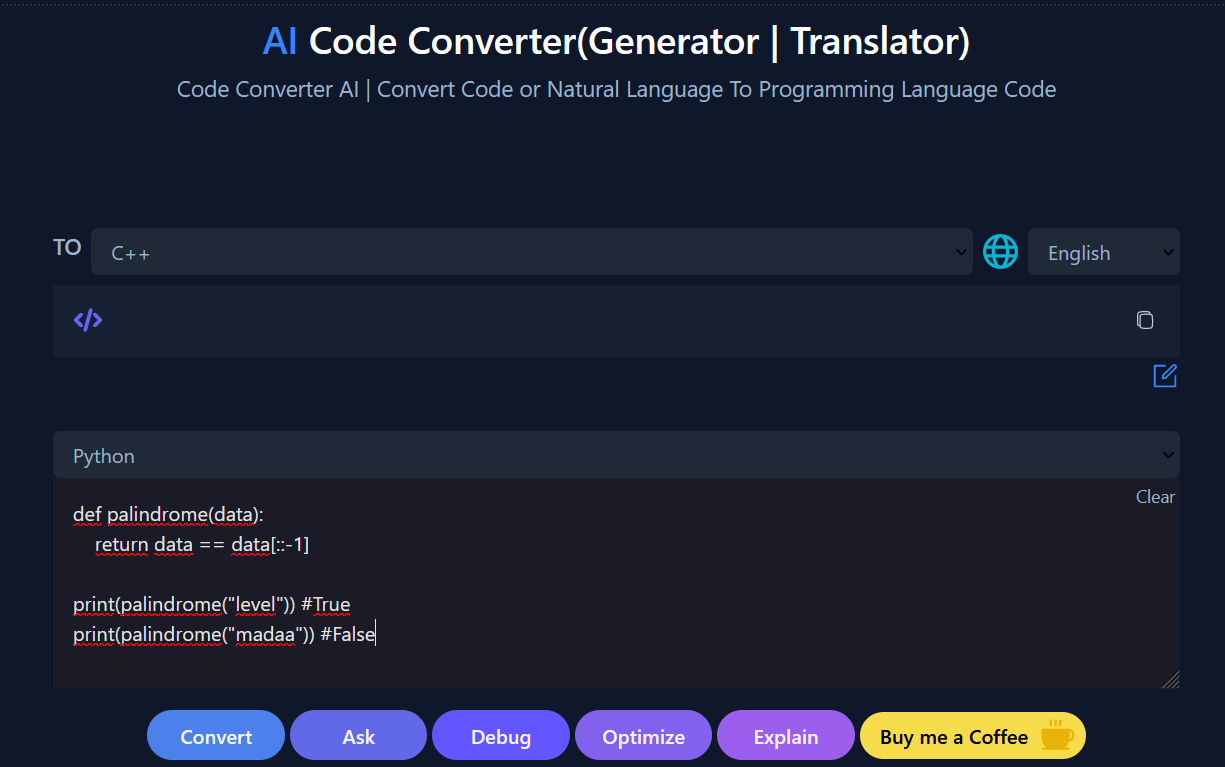


Рисунок 4.11 – Общий вид сервиса

Для конвертации необходимо ввести свой код в поле внизу, после чего нажать кнопку «Convert». В результате в поле вверху получим готовый код (Рисунок 4.12). Запишем полученный код в IDE и выполним. Как видно из рисунка 4.13, программа успешно выполнилась и выдала ожидаемый результат.



Рисунок 4.12 – Готовый код в сервисе

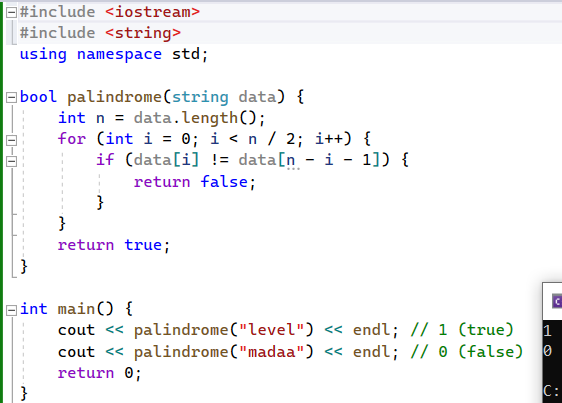


Рисунок 4.13 – Выполненный код в IDE

Таким образом, простую программу сервис легко переводит.

Возьмём следующую по сложности программу – консольная программа для игры в «виселицу», код которой изображён на рисунке 4.4. По аналогии, получим код на выходе сервиса и запустим его в IDE. Результат приведён на рисунке 4.14.

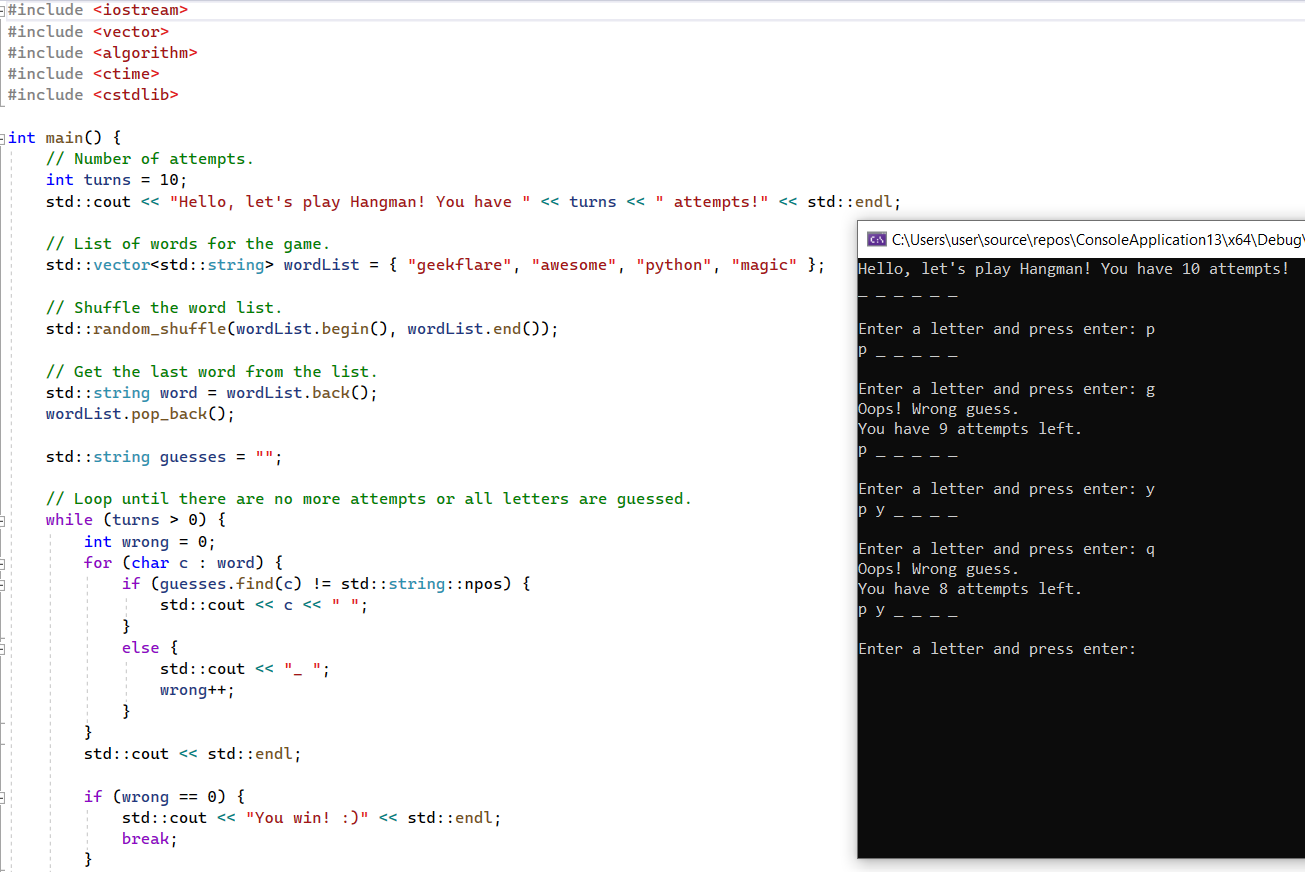


Рисунок 4.14 – Запуск конвертированной программы

Как можно заметить, программа прекрасно скомпилировалась и весь функционал, заложенный в неё, работает. Следует отдельно отметить, что в отличии от предыдущего способа, описанного в разделе 4.1, программа автоматически перевела текст нашей программы на английский язык, тем самым, избавив нас от исправления ошибки, с который мы сталкивались ранее. Однако, если посмотреть внимательнее, то конвертация на другой язык происходит по желанию пользователя, и по умолчанию выбран английский. Давайте исправим это. Поле с данной настройкой выделено на рисунке 4.15.



Рисунок 4.15 – Поле выбора языка

Снова выполним конвертацию кода с обновлёнными настройками. Результат, также как и предыдущий, скомпилирует и запустим. Полученный вывод изображён на рисунке 4.16.

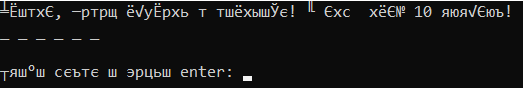


Рисунок 4.16 – Вывод программы со строками на русском языке

Как видно, мы столкнулись с той же проблемой. Сервис не добавил нам локализацию на русский язык. Исправим это добавив необходимую строку кода setlocale(LC\_ALL, “ru”); После того, как добавим данную строку, попробуем снова запустить программу, что отобразим на рисунке 4.17.

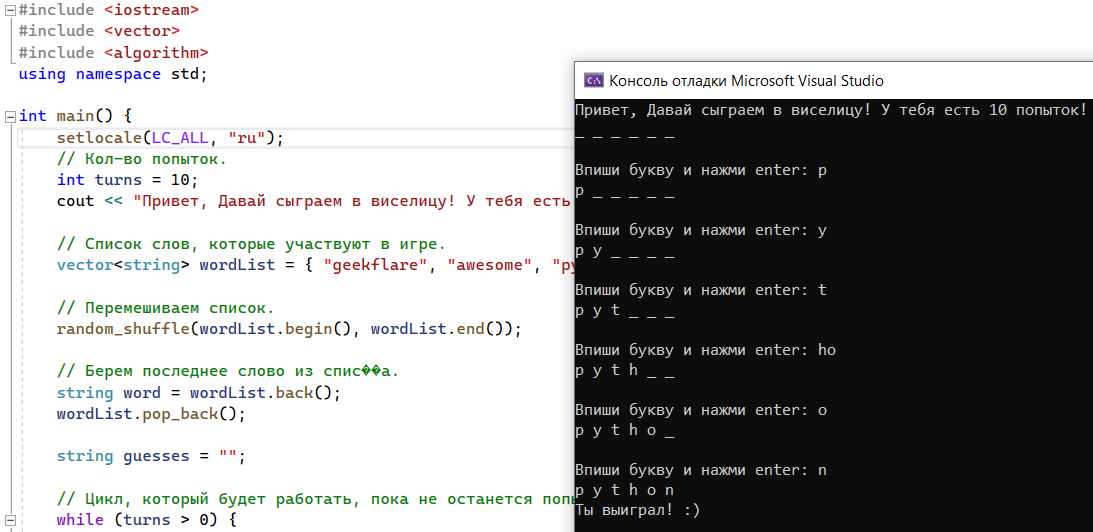


Рисунок 4.17 – Запуск сконвертированной программы с русской локализацией

Наша программа работает так, как и ожидается.

Перейдём к самому сложному – конвертации программы «трудного» уровня, использующей графику и стороннюю библиотеку. Этот пример возьмём также, как и предыдущие, из раздела 4.1. Код приведён ранее на рисунке 4.8. Результат конвертации приведён на рисунке 4.18. Как можно видеть, конвертация оказалась провальной. Несмотря на попытку использовать работу с графикой, в данном случае сервис лишь изменил размер окна консоли. При этом программа работает без нареканий, с оговоркой на то, что утерян весь исходный функционал. На смену ему появился бесконечный цикл, обновляющий переменную х.

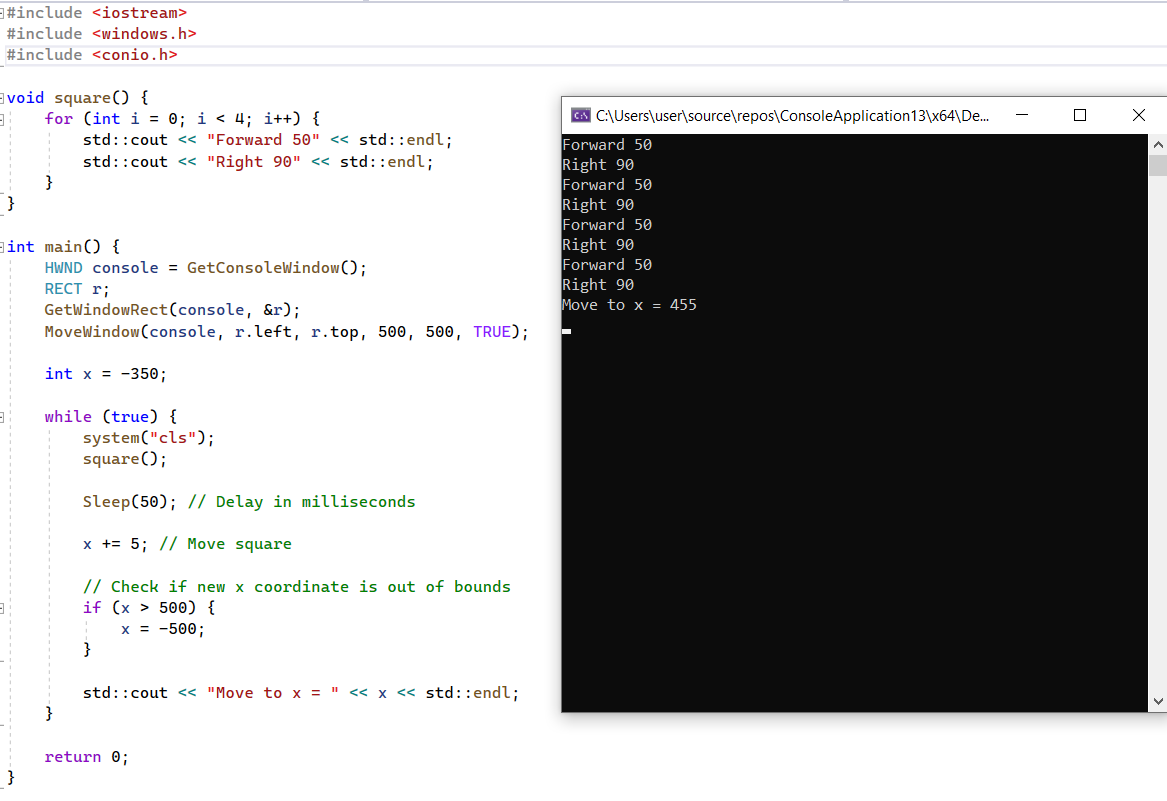


Рисунок 4.18 – Результат выполнения сконвертированного «сложного» кода

В рамках данной главы рассмотрен способ конвертации кода с языка программирования Python на язык программирования C++. Было рассмотрено три примера разной сложности. По результатам можно утверждать, что конвертация кода с помощью стороннего инструментария способна качественно помочь перенести простой по логике код на другой язык программирования с минимальными доработками или вовсе без них. С другой же стороны, конвертация кода со сложной логикой до сих пор является трудной задачей, с решением которой не способен справиться инструментарий, рассматриваемый в данной главе, и для качественного перевода требуется квалифицированный человек.

# заключение

Подведём итоги. В данном учебном пособии были рассмотрены несколько способов проведения реверс-инжиниринга программ. Одни созданы исключительно для конкретных языков программирования, тогда как другие являются универсальными. Затронута тема конвертации программного кода с одного языка на другой. На примере показано, что данное направление в текущее время слабо развито, однако имеет огромные перспективы в будущем, в частности, за счёт возможностей нейронных сетей.

Рассмотрен основной функционал программ. Базовые знания, изложенные в пособии, помогут снизить порог вхождения в работу с инструментарием обратной разработки. Рассмотрены эффективные методы поиска ключевых мест в программах. Отдельно освещены дополнительные темы, такие как источники для практики, документация к программам, перспективы в изучении данных инструментов.

В результате проделанной работу студенты получили навыки работы с инструментами получения исходного кода, реверс‑инжиниринга программ, конверторами кода, а также прикладные навыки работы с языком Ассемблера и переменными среды.

Стоит понимать, что методы, используемые в данном учебном пособии, хоть и распространены, но не являются универсальными для всех отлаживаемых программ. Полное понимание процессов, происходящих в программе – зачастую затратное по времени, или в целом невозможное для человека действие. Однако хорошее понимание механик, основанных на изложенных в данном пособии, способствует ускорению работы и отладки программ.

Наконец, происходит постоянная «гонка вооружений», заключающаяся в создании новых способов защиты и их обхода. Для того, чтобы выполнить комплекс работ и в будущем стать квалифицированным специалистом, необходимо постоянное совершенствование в описанной области.

# Список литературы

1. Савин Л.А. Применение инструментов обратной разработки для выявления недекларированных возможностей программного обеспечения систем управления/
2. Веб‑сервис для хостинга IT‑проектов GitHub [Электронный ресурс] // GitHub : The complete developer platform to build, scale, and deliver secure software. – 2023. –URL: <https://github.com/NotPrab/.NET-Deobfuscator> (дата обращения 24.10.23).
3. NET Decompiler: Decompile Any .NET Code | .NET Reflector // RedGate URL: 1. https://www.red-gate.com/products/reflector/ (дата обращения: 18.11.23).
4. Crackmes // crackmes.one URL: https://crackmes.one/crackme/64d5ef92b25df8732eebc74c (дата обращения: 18.11.23).
5. icsharpcode/ILSpy: .NET Decompiler with support for PDB generation, ReadyToRun, Metadata (&more) // GitHub URL: https://github.com/icsharpcode/ILSpy (дата обращения: 28.01.24).
6. NationalSecurityAgency/ghidra // GitHub URL: https://github.com/NationalSecurityAgency/ghidra/releases (дата обращения: 28.01.24).
7. OpenJDK JDK 21.0.2 GA Release // jdk.java.net URL: https://jdk.java.net/21/ (дата обращения: 28.01.24).
8. Crackmes // crackmes.one URL: https://crackmes.one/crackme/65ad74aceef082e477ff5f24 (дата обращения: 28.01.24).
9. Dark Theme ASCII Table // ByteTool URL: https://bytetool.web.app/en/ascii/ (дата обращения: 28.01.24).
10. Crackmes // crackmes.one URL: <https://crackmes.one/crackme/65ba60e0eef082e477ff6482> (дата обращения: 29.02.24).

Ковров Артём Игоревич

Ляпина Елизавета Павловна

Савин Лев Андреевич

Сидоренко Валентина Геннадьевна

Соколов Илья Дмитриевич

Инструменты реверс‑инжиниринга и транскомпиляции

Учебное пособие