|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | Министерство науки и высшего образования Российской Федерации  Калужский филиал  федерального государственного бюджетного  образовательного учреждения высшего образования  ***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»***  ***(КФ МГТУ им. Н.Э. Баумана)*** |

**ФАКУЛЬТЕТ** \_ ***ИУК «Информатика и управление»\_\_\_\_\_\_\_\_\_***

**КАФЕДРА** \_\_ ***ИУК5 «Системы обработки информации»***

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовой работе на тему:**

***реализация технологии Reflective DLL injection с помощью Windows API***

по дисциплине ***Системное программирование***

Студент гр.ИУК5-42Б \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (\_\_\_\_\_\_*Фролов К. Д.* \_\_\_)

(подпись) (Ф.И.О.)

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (\_\_\_\_\_\_*Фролов П. В.* \_\_\_)

(подпись) (Ф.И.О.)

Оценка руководителя \_\_\_\_\_ баллов\_\_\_\_\_\_\_\_\_\_\_\_\_\_

30-50 (дата)

Оценка защиты \_\_\_\_\_ баллов \_\_\_\_\_\_\_\_\_\_\_

30-50 (дата)

Оценка проекта \_\_\_\_\_ баллов \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(оценка по пятибалльной шкале)

Комиссия: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

(подпись) (Ф.И.О.)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

(подпись) (Ф.И.О.)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

(подпись) (Ф.И.О.)

Калуга, 2021

Калужский филиал   
федерального государственного бюджетного образовательного учреждения высшего образования

***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»   
(КФ МГТУ им. Н.Э. Баумана)***

УТВЕРЖДАЮ

Заведующий кафедрой **\_\_ИУК5\_\_\_**

\_\_\_\_\_\_\_\_\_\_\_(Е.В. Вершинин)

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_г.

**ЗАДАНИЕ**

**на выполнение курсовой работы**

по дисциплине ***Системное программирование***

Студент\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(фамилия, инициалы, индекс группы)

Руководитель\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(фамилия, инициалы)

График выполнения проекта: 25% к\_4\_нед., 50% к\_7\_нед., 75% к\_10\_нед., 100% к\_14\_нед.

***1. Тема курсового проекта***

***реализация технологии Reflective DLL injection с помощью Windows API***

***2. Техническое задание***

Разработать приложение с использованием функций Windows API для внедрения кода в целевой процесс\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

***3. Оформление курсового проекта***

3.1. Расчетно-пояснительная записка на\_\_\_\_\_\_\_\_ листах формата А4.

3.2. Перечень графического материала КП (плакаты, схемы, чертежи и т.п.)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата выдачи задания «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_г.

Руководитель курсового проекта \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /\_\_ \_Фролов П. В.\_\_\_\_/

(подпись) (Ф.И.О.)

Задание получил\_\_\_\_\_\_\_\_\_\_\_\_\_\_/\_\_\_\_\_ Фролов К. Д.\_\_\_\_\_\_/ «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_2021г.

(подпись) (Ф.И.О.)

**Содержание**

[1. ТЕХНИЧЕСКОЕ ЗАДАНИЕ 9](#_Toc73033577)

[1.1. Наименование 9](#_Toc73033578)

[1.2. Основание для разработки 9](#_Toc73033579)

[1.3. Исполнитель 9](#_Toc73033580)

[1.4. Цель разработки 9](#_Toc73033581)

[1.5. Содержание работы 10](#_Toc73033582)

[1.5.1. Задачи, подлежащие решению: 10](#_Toc73033583)

[1.5.2. Требования к архитектуре АСОИ 10](#_Toc73033584)

[1.5.3. Требования к составу программных компонентов 11](#_Toc73033585)

[1.5.4. Требования к прикладным программам 11](#_Toc73033586)

[1.5.5. Требования к входным/выходным данным 11](#_Toc73033587)

[1.5.6. Требования к временным характеристикам 11](#_Toc73033588)

[1.5.7. Требования к составу технических средств 12](#_Toc73033589)

[1.6. Этапы разработки 12](#_Toc73033590)

[1.7. Техническая документация, предъявляемая по окончании работы 12](#_Toc73033591)

[1.8. Дополнительные условия 13](#_Toc73033592)

[2. НАУЧНО – ИССЛЕДОВАТЕЛЬСКАЯ ЧАСТЬ 14](#_Toc73033593)

[2.1 Постановка задачи проектирования 14](#_Toc73033594)

[2.2 Описание предметной области 14](#_Toc73033595)

[2.3 Анализ аналогов и прототипов 17](#_Toc73033596)

[2.3.1 Injection DLL 17](#_Toc73033597)

[2.3.2 Injection PE 17](#_Toc73033598)

[2.3.3 Process Hollowing 18](#_Toc73033599)

[2.3.4 Hook injection 19](#_Toc73033600)

[2.3.5 Преимущества выбранной технологии. 20](#_Toc73033601)

[2.4 Перечень задач подлежащих решению в процессе разработки 20](#_Toc73033602)

[2.5 Обоснование выбора инструментов и платформы для разработки 20](#_Toc73033603)

[2.5.1 Операционная система Windows 10. 21](#_Toc73033604)

[2.5.2 Язык программирования C++ 22](#_Toc73033605)

[2.5.3 Среда разработки Microsoft Visual Studio 2019 24](#_Toc73033606)

[3. ПРОЕКТНО – КОНСТРУКТОРСКАЯ ЧАСТЬ 25](#_Toc73033607)

[3.1 Разработка структуры приложения 25](#_Toc73033608)

[3.2 Используемые функции Win32API 26](#_Toc73033609)

[3.2.1 User32.lib 26](#_Toc73033610)

[3.2.2 Advapi32.lib 28](#_Toc73033611)

[3.2.3 Kernel32.lib 29](#_Toc73033612)

[3.3 Разработка и реализация алгоритмов приложения 33](#_Toc73033613)

[3.3.1 Общий алгоритм работы приложения. 33](#_Toc73033614)

[3.3.2 Основные алгоритмы приложения: 34](#_Toc73033615)

[3.4 Разработка интерфейса взаимодействия пользователя с системой 36](#_Toc73033620)

[4. ПРОЕКТНО-ТЕХНОЛОГИЧИСКАЯ ЧАСТЬ 38](#_Toc73033621)

[4.1. Тестирование и отладка макета рабочей программы. 38](#_Toc73033622)

[4.2. Разработка руководства пользователя и администратора. 42](#_Toc73033623)

[4.2.1 Разработка руководства администратора 42](#_Toc73033624)

[4.2.2 Разработка руководства пользователя 44](#_Toc73033632)

[5. ЗАКЛЮЧЕНИЕ 51](#_Toc73033633)

[6. СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТУРАТУРЫ 52](#_Toc73033634)

# 1. ТЕХНИЧЕСКОЕ ЗАДАНИЕ

## 1.1. Наименование

Программная реализация технологии внедрения DLL библиотеки в целевой процесс «Reflective DLL Injection.

## 1.2. Основание для разработки

В системном программировании внедрение кода в целевой процесс играет важную роль для решения прикладных задач по поддержке существующих приложений. Процесс внедрения стороннего кода, ожидает запуска конкретного приложения и вводит дополнительный код в запущенный процесс, чтобы изменить или дополнить его функциональные возможности. Например: программная платформа сценариев, такая как «AutoHotkey», с помощью добавления небольшого кода, предоставляет пользователям возможность создавать сценарии, которые запускаются в фоновом режиме с помощью комбинаций, предварительно заданных, горячих клавиш.

Для внедрения кода в целевой процесс существует технология Reflective dll injection. Для применения на практике необходимо разработать её программную реализацию, которая бы эффективно осуществляла внедрение полезной нагрузки и не обнаруживалась антивирусами как вредоносное программное обеспечение.

## 1.3. Исполнитель

Студент группы ИУК5-42Б Фролов Кирилл Дмитриевич

## 1.4. Цель разработки

Целью курсовой работы является формирование практических навыков по разработке и реализации программного приложения с использованием интерфейса прикладного программирования (АРI) операционных систем.

Задачи проектирования:

1. овладение первичными навыками ведения научно-исследовательской, проектной и производственно-технологической деятельности, развитие творческих способностей индивидуально для каждого студента;
2. подготовка к выполнению выпускной квалификационной работы;
3. усвоение методов грамотного ведения, оформления и редактирования технической документации.

Целью разработки является изучение принципов системного программирования и работы Windows API, а также программная реализация технологии Reflective dll injection для внедрения dll библиотеки в целевой процесс.

## 1.5. Содержание работы

### 1.5.1. Задачи, подлежащие решению:

1. исследование существующих методов внедрения программного кода в целевой процесс;
2. изучение методов работы системных библиотек и WinAPI функций;
3. реализация метода Reflective dll injection;
4. Тестирование реализованной технологии;
5. подготовка расчетно-пояснительной записки и графических листов;
6. подготовка презентации и речи для защиты курсовой работы;
7. защита курсовой работы.

### 1.5.2. Требования к архитектуре АСОИ

К архитектуре предъявляются следующие требования:

1. тип приложения – оконное;
2. модуль полезной нагрузки, представляющий из себя DLL библиотеку;
3. модуль загрузчика, представляющий из себя оконное приложение;
4. модуль загрузчика должен быть представлен в виде отдельного класса;

### 1.5.3. Требования к составу программных компонентов

Программный комплекс должен состоять из следующих программных компонентов:

1. исполняемый файл PE формата с расширением .exe, реализующий метод Reflective dll injection;
2. полезная нагрузка в виде динамической библиотеки .dll ;

### 1.5.4. Требования к прикладным программам

Для работы программного комплекса необходимы:

1. Microsoft Windows 8/10;
2. Microsoft Visual Studio 2019;
3. минимальный набор драйверов, обеспечивающих   
   работоспособность ПК.

### 1.5.5. Требования к входным/выходным данным

Входные данные:

1. PID процесса, в который будет производится внедрение;
2. путь к полезной нагрузке.

Выходные данные:

1. всплывающее окно типа «MessageBox» с сообщением об успешном внедрении.

### 1.5.6. Требования к временным характеристикам

Требования к временным характеристикам программы не предъявляются

### 1.5.7. Требования к составу технических средств

Для функционирования системы необходимы:

* Процессор: не менее 1 ГГц
* ОЗУ: 1 ГБ для 32-разрядной системы или 2 ГБ для 64-разрядной системы
* Место на жестком диске:16 ГБ для 32-разрядной ОС или 20 ГБ для 64-разрядной ОС
* Видеоадаптер: DirectX 9 или более поздняя версия с драйвером WDDM 1.0
* Экран: 800 x 600
* монитор;
* клавиатура;
* мышь.

## 1.6. Этапы разработки

1. создание пользовательского интерфейса;
2. реализация метода Reflective dll injection;
3. тестирование реализованной технологии;

## 1.7. Техническая документация, предъявляемая по окончании работы

По окончанию работы предъявлена расчетно-пояснительная записка в состав которой входят:

* техническое задание;
* научно-исследовательская часть;
* проектно-конструкторская часть;
* проектно-технологическая часть.

Также должна быть предоставлена графическая часть работы, выполненная формате А1 на 2 листах, в которую входят:

* демонстрационные чертежи;
* алгоритмические схемы.

## 1.8. Дополнительные условия

1. язык программирования С/С++;
2. использование Windows API функций для реализации метод внедрения программного кода в сторонний процесс;
3. интерфейс должен предоставлять пользователю текстовые поля для ввода PID в который необходимо внедриться и путь к полезной нагрузке, и кнопку «внедриться». По завершении работы должен появиться MessegeBox с информацией что внедрение произошло успешно.

# 2. НАУЧНО – ИССЛЕДОВАТЕЛЬСКАЯ ЧАСТЬ

## 2.1 Постановка задачи проектирования

Задача курсовой работы состоит в программной реализации технологии внедрения DLL библиотеки в целевой процесс «Reflective DLL Injection», которая позволяет улучшить функционал уже существующего программного обеспечения. А также необходимо разработать удобный и понятный интерфейс.

## 2.2 Описание предметной области

Предметной областью являются технологии внедрения программного кода в процессы. Рефлексивная DLL инъекция дает возможность выполнять свой код в адресном пространстве уже запущенного процесса. Такое внедрение часто используются внешними программами, чтобы повлиять на поведение другой программы так, как её авторы не задумывали и не предполагали. Например, внедрённый код может перехватывать системные вызовы функций или прочитать содержимое текстовых полей пароля, что невозможно сделать обычным способом. Но данный прием не обязательно применять в плохих целях и, например рефлексивное внедрение может применяться для обновления своего приложения или для его усовершенствования.

Внедрение кода используется для выполнения всевозможных функций в Windows. Эту технологию используют легитимные программы, а также используют вредоносные программы. Например:

Антивирусные программы часто внедряют код в веб-браузеры. Они могут использовать его, например, для отслеживания сетевого трафика и блокировки опасного веб-контента.

Вредоносные программы могут добавлять код в ваш веб-браузер, чтобы лучше отслеживать ваши действия в Интернете, красть защищенную информацию, такую как пароли и номера кредитных карт, а также изменять настройки браузера.

WindowBlinds Stardock, который задает темы для рабочего стола, внедряет код в изменить способ отображения окон .

AutoHotkey, который позволяет создавать сценарии и назначать им общесистемные горячие клавиши , внедряет код для этого.

Графический драйвер подобен встроенным DLL от NVIDIA для выполнения множества задач, связанных с графикой.

Некоторые программы внедряют библиотеки DLL для добавления дополнительных пунктов меню в приложение.

Инструменты для мошенничества в компьютерных играх часто вводят в игры код, чтобы изменить их поведение и получить несправедливое преимущество перед другими игроками.

Рефлексивное внедрение происходит по этапам:

1. Некоторый исполняемый файл считывает DLL-библиотеку с диска в адресное пространство своего процесса и передает управление на ее экспортируемую функцию ReflectiveLoader.

2. Поскольку теперь библиотека существует в произвольном месте в памяти, ReflectiveLoader вычисляет текущее местоположение самой DLL-библиотеки в памяти. Для этого ReflectiveLoader получает адрес текущей инструкции и, двигаясь в обратном направлении, ищет байты 4D5A, соответствующие MZ-сигнатуре. Это нужно, чтобы дать библиотеке возможность анализировать свои собственные заголовки для дальнейшего запуска.

3. ReflectiveLoader определяет адрес библиотеки kernel32.dll в текущем процессе, после чего анализирует таблицу ее экспорта и находит функции LoadLibrary, GetProcAddress и VirtualAlloc, необходимые для дальнейшей загрузки.

4. ReflectiveLoader выделяет непрерывный участок памяти (VirtualAlloc), где и размещает код DLL-библиотеки (заголовки и секции) в соответствии с виртуальными адресами. Затем ReflectiveLoader обрабатывает свою вновь загруженную таблицу импорта: загружает необходимые библиотеки (LoadLibrary) и импортируемые из них функции (GetProcAddress).

5. ReflectiveLoader вызывает DllEntryPoint внедряемой библиотеки.

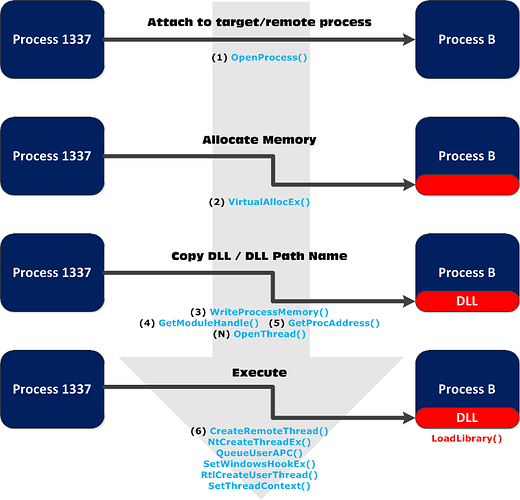


Рис. 1 – Как происходит рефлексивное внедрение в целевой процесс.

## 2.3 Анализ аналогов и прототипов

Существует большое количество методов внедрения такие как:

### 2.3.1 Injection DLL

Injection DLL - это один из наиболее распространенных методов внедрения в целевой процесс. Перед внедрением вредоносной программе потребуется копия вредоносной DLL, уже сохраненная в системе жертвы. Такая технология имеет преимущество, такое что в DLL библиотеке храниться все методы и функции, тем самым можно вызывать эти функции из внедряемого процесса. Такое внедрение происходит по этапам:

1. Программа вызывает функцию Windows API (OpenProcess) для подключения к процессу

2: Часть памяти выделяется в процессе с помощью VirtualAllocEx. Эта память выделяется с использованием доступа «для записи».

3. Программа должна найти адрес функции LoadLibrary в пространстве процесса.

4: Программа вызывает функцию CreateRemoteThread, создавая новый поток в процессе, передавая адрес LoadLibrary, найденный на шаге 3. Когда загружается вредоносная DLL, вызывается метод записи DLL, DLLMain. Здесь будут происходить все действия.

### 2.3.2 Injection PE

Portable Execution (PE) - это формат файла Windows для исполняемого кода. Это структура данных, содержащая всю необходимую информацию, чтобы Windows знала, как ее выполнять.

PE-injection - это метод, при котором программа внедряет PE-образ в уже запущенный процесс. Преимущество этого метода перед внедрением DLL состоит в том, что это бездисковая операция, то есть программе не нужно записывать свои полезные данные на диск перед внедрением. Внедрение происходит так же по этапам:

1. Программа получает базовый адрес и размер процесса-жертвы.

2. Программа выделяет достаточно памяти в процессе жертвы для вставки своего PE-образа.

3. Поскольку вставленное изображение будет иметь другой базовый адрес после его внедрения в затронутый процесс, программе необходимо сначала найти смещение таблицы перемещения процесса-жертвы. С этим смещением программа изменит изображение так, чтобы любые абсолютные адреса в изображении указывали на нужные функции. После обновления PE-образа программа копирует его в процесс.

4. Программа ищет функцию входа для выполнения и запускает ее с помощью CreateRemoteThread.

### 2.3.3 Process Hollowing

Process Hollowing- это метод, при котором Программа запускает процесс, и заменяет код процесса на свой. Преимущество этого метода заключается в том, что программа становится независимым от того, что в данный момент запущено в системе у пользователя. Более того, запустив системный процесс (например, Блокнот), пользователи не будут видеть исполняемый код во внедряемом процессе. Пример того как происходит внедрение:

1. Вредоносная программа создает новый процесс, например «Блокнот», но инструктирует Windows создать его как приостановленный процесс. Это означает, что новый процесс не запустится.

2. Программа делает процесс пустым, отключая связанные с ним области памяти.

3. Программа выделяет память для собственного кода и копирует его в пространство памяти процесса. Затем он вызывает SetThreadContext для процесса-пользователя, который изменяет контекст выполнения процесса на только что созданный.

4. Программа возобновляет процесс; тем самым выполняя свой код.

### 2.3.4 Hook injection

Hook injection это метод который использует ловушку WH\_GETMESSAGE , он устанавливаем процесс, который будет следить за сообщениями, обрабатываемыми системными окнами. Чтобы установить ловушку, мы вызываем функцию SetWindowsHookEx.

WH\_GETMESSAGE аргумент определяет тип сообщения, а functionAddress определяет адрес функции (в адресном пространстве процесса) , программа должна сообщить, когда система пытается обработать сообщение. Таким образом, этот метод можно применить для перехвата определенного процесса или всех процессов в системе. Пример работы hook injection

1. Поток application.exe собирается отправить сообщение в какое-то окно.

2. Система проверяет, установлен ли для этого потока ловушка WH\_GETMESSAGE.

3. Затем система выясняет, отображается ли Inject.dll, которая содержит обратный вызов сообщений.

4. Если Inject.dll еще не сопоставлен, система сопоставляет его с адресным пространством процесса application.exe и увеличивает счетчик блокировок библиотеки DLL в этом процессе.

5.Функция DllMain Inject.dll вызывается с параметром DLL\_PROCESS\_ATTACH .

6. Затем в адресном пространстве процесса application.exe вызывается обратный вызов.

### 2.3.5 Преимущества выбранной технологии.

Reflective DLL Injection позволяет внедрить код DLL-библиотеки в процесс из памяти. Основное преимущество такого подхода заключается в том, что библиотека не регистрируется в системе. В результате ее практически невозможно обнаружить ни на уровне системы, ни на уровне процесса.

### 2.4 Перечень задач подлежащих решению в процессе разработки

1. разработка пользовательского интерфейса;
2. реализация технологии Reflective DLL injection с помощью WinAPI на языке С++;
3. реализация DLL библиотеки, с вызовом MessageBox;
4. Запись заголовков DLL в память;
5. Тестирование и проверка разработанной программы на виртуальной машине;
6. Вывод всплывающих окон с информацией о внедрении;

### 2.5 Обоснование выбора инструментов и платформы для разработки

Для выполнения курсовой работы были выбраны:

### 2.5.1 Операционная система Windows 10.

В операционной системе Windows реализована объектно-ориентированная идеология. Базовый объект системы – окно, поведение которого определяется методом, называемым функцией окна. Графический образ окна на экране дисплея – прямоугольная рабочая область. Независимо от своего типа любой объект Windows идентифицируется описателем или дескриптором (handle). Все взаимоотношения программного кода с объектом осуществляются только через его дескриптор. Интерфейс прикладного программирования (API – Application Programming Interface) представляет собой совокупность 32-битных функций (Win32 API), которые предназначены для создания приложений (программ), работающих под управлением Microsoft Windows. Функции объявлены в заголовочных файлах. Главный из них − файл windows.h, в котором содержатся ссылки на другие заголовочные файлы.

API — это аббревиатура названия Application Programming Interface (интерфейс прикладного программирования). API представляет собой совокупность функций и инструментов, позволяющих программисту создавать приложения (программы), работающие в некоторой среде.

Win32 API — это набор функций для создания программ, работающих под управлением Microsoft Windows 98, Windows NT или Windows 2000. Все функции этого набора являются 32 битными, что отражено в названии интерфейса

Причины выбора Операционной системы Windows 10:

1. Простота эксплуатации. Даже неподготовленному человеку легко разобраться в принципах работы операционной системы, ведь все интуитивно понятно, также система не требует знаний в программировании.

2. Большое количество программ сторонних компаний выпускают именно под данную операционную систему. Есть как платные, так и бесплатные версии.

3. Большая часть выпускаемого оборудования для ПК выпускается с поддержкой ОС Windows. Веб-камеры, сканеры, принтеры, игровые манипуляторы и так далее. Все драйвера, в первую очередь, выходят под данную ОС, и лишь по прошествии большого промежутка времени выходят под другие операционные системы.

4. Большая распространённость. Данная операционная система, по состоянию на апрель 2020 года, заняла второе место в общей мировой статистике использования ОС.

5. Различные формы и виды оформления. Кроме стандартных вариантов ОС позволяет сторонним программам вносить изменения во внешний вид рабочего стола, папок, заставок и так далее.

6. Можно осуществлять полное взаимодействие с ОС с помощью Windows API и его компонентами

### 2.5.2 Язык программирования C++

C++ — компилируемый, статически типизированный язык программирования общего назначения. Поддерживает такие парадигмы программирования, как процедурное программирование, объектно-ориентированное программирование, обобщённое программирование. Язык имеет богатую стандартную библиотеку, которая включает в себя распространённые контейнеры и алгоритмы, ввод-вывод, регулярные выражения, поддержку многопоточности и другие возможности. C++ сочетает свойства как высокоуровневых, так и низкоуровневых языков. В сравнении с его предшественником — языком C — наибольшее внимание уделено поддержке объектно-ориентированного и обобщённого программирования.

C++ широко используется для разработки программного обеспечения, являясь одним из самых популярных языков программирования. Область его применения включает создание операционных систем, разнообразных прикладных программ, драйверов устройств, приложений для встраиваемых систем, высокопроизводительных серверов, а также игр. Синтаксис C++ унаследован от языка C. Одним из принципов разработки было сохранение совместимости с C. Тем не менее C++ не является в строгом смысле надмножеством C; множество программ, которые могут одинаково успешно транслироваться как компиляторами C, так и компиляторами C++, довольно велико, но не включает все возможные программы на C. Одними из отличительных особенностей и очень важных преимуществ C++ является:

1. Поддерживаются различные стили и технологии программирования, включая традиционное директивное программирование, ООП, обобщённое программирование, метапрограммирование (шаблоны, макросы).

2. Предсказуемое выполнение программ является важным достоинством для построения систем реального времени. Весь код, неявно генерируемый компилятором для реализации языковых возможностей (например, при преобразовании переменной к другому типу), определён в стандарте. Также строго определены места программы, в которых этот код выполняется. Это даёт возможность замерять или рассчитывать время реакции программы на внешнее событие.

3. Автоматический вызов деструкторов объектов при их уничтожении, причём в порядке, обратном вызову конструкторов. Это упрощает (достаточно объявить переменную) и делает более надёжным освобождение ресурсов (память, файлы, семафоры и т. п.), а также позволяет гарантированно выполнять переходы состояний программы, не обязательно связанные с освобождением ресурсов (например, запись в журнал).

4. Пользовательские функции-операторы позволяют кратко и ёмко записывать выражения над пользовательскими типами в естественной алгебраической форме.

5. Windows API представляет собой множество функций, структур данных и числовых констант, следующих соглашениям языка Си. Все языки программирования, способные вызывать такие функции и оперировать такими типами данных в программах, исполняемых в среде Windows, могут пользоваться этим API. В частности, это язык C++

### 2.5.3 Среда разработки Microsoft Visual Studio 2019

Microsoft Visual Studio 2019— это набор инструментов разработки, основанных на использовании компонентов, и других технологий для создания мощных, производительных приложений. Кроме того, среда Visual Studio оптимизирована для совместного проектирования, разработки и развертывания корпоративных решений. Visual Studio предоставляет средства для проектирования, разработки b отладки.

Среда разработки Visual Studio представляет собой полный набор средств разработки для создания веб-приложений ASP.NET, XML (веб-службы), настольных приложений и мобильных приложений. Visual Basic, Visual C# и Visual C++ используют единую интегрированную среду разработки (IDE), которая позволяет совместно использовать средства и упрощает создание решений на базе нескольких языков. Можно создать обычные приложения Microsoft Windows и приложения с архитектурой "клиент-сервер" с помощью использования конструкторов в Visual Studio.

Почему именно эта версия продукта, потому что в нем есть такие удобные нововведения как:

1. Visual Studio IntelliCode повышает эффективность разработки программного обеспечения с помощью искусственного интеллекта (ИИ). Для создания рекомендаций IntelliCode анализирует 2000 проектов с открытым кодом на GitHub

2. Рефакторинг В C++ есть много новых удобных возможностей рефакторинга, которые помогают упорядочить код. Они отображаются как предложения со значком лампочки и включают такие действия, как перемещение элементов в интерфейс или базовый класс, настройку пространств имен в соответствии со структурой папок.

# 3. ПРОЕКТНО – КОНСТРУКТОРСКАЯ ЧАСТЬ

## 3.1 Разработка структуры приложения

Для разработки используется среда Microsoft Visual Studio 2019.

Для представления принципа работы ниже приведена UML диаграмма использования (Рисунок 2).

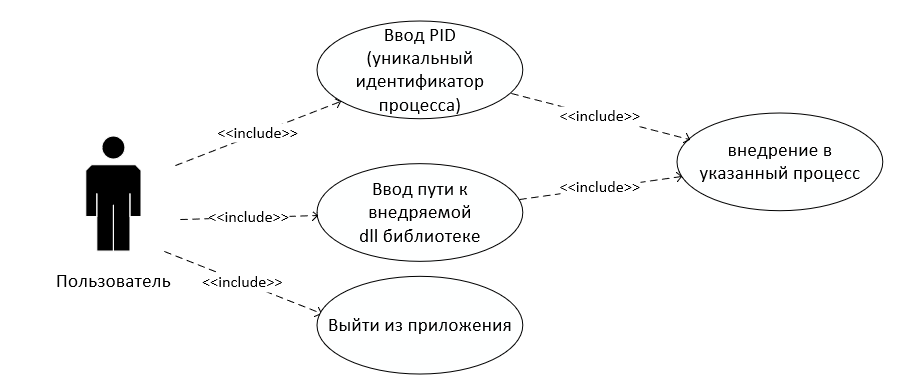


Рисунок 2 - Диаграмма использования приложения «Reflective DLL Injection»

Пользователь должен выбрать идентификатор процесса с помощью диспетчера задач во вкладке подробности, или с помощью любой другой программы, которая предоставляет идентификаторы процессов.

Пользователем заполняется поле ввода идентификатора процесса и путь к dll библиотеке которая должна быть расположена на компьютере пользователя. Чтобы ввести путь пользователю нужно открыть проводник и перейти в папку где расположена dll библиотека и скопировать путь в поле для ввода.

После удачного внедрения в процесс пользователь будет оповещен о удачном внедрении всплывающим окном, в случае если внедрение не удалось пользователь также будет оповещен об ошибке из за которой не удалось внедрить dll библиотеку в процесс.

## 3.2 Используемые функции Win32API

Используемые библиотеки Win32API, функции и их аргументы:

### 3.2.1 User32.lib

* **RegisterClassExW** - Регистрирует класс окна для последующего использования в вызовах функции CreateWindow или CreateWindowExW .

RegisterClassExW(

const WNDCLASSEXW \*unnamedParam1 Указатель на структуру WNDCLASSEX .);

* **PostQuitMessage** - Указывает системе, что поток сделал запрос на завершение

void PostQuitMessage(

int nExitCode Код выхода из приложения

);

* **GetWindowLongPtr** - Получает информацию об указанном окне.

LONG\_PTR GetWindowLongPtrA(

HWND hWnd, Дескриптор окна

int nIndex Устанавливаемое значение смещения от 0 (GWLP\_USERDATA)

);

GWLP\_USERDATA - Устанавливает пользовательские данные, **DefWindowProc** - Вызывает оконную процедуру по умолчанию, чтобы обеспечить обработку по умолчанию для любых оконных сообщений, которые приложение не обрабатывает.

LRESULT LRESULT DefWindowProcA(

HWND hWnd, Дескриптор оконной процедуры

UINT Msg, Сообщение.

WPARAM wParam, Дополнительная информация о сообщении.

LPARAM lParam Дополнительная информация о сообщении.

);

* **GetWindowTextW** - Копирует текст строки заголовка указанного окна (если она есть) в буфер.

int GetWindowTextW(

HWND hWnd, Дескриптор окна

LPWSTR lpString, Буфер, в который будет поступать текст

Int nMaxCount); Максимальное количество символов для копирования в буфер.

* **CreateWindowExW** - Создает перекрывающееся, всплывающее или дочернее окно с расширенным стилем окна

HWND CreateWindowExW(

DWORD dwExStyle, Расширенный стиль создаваемого окна

LPCSTR lpClassName, Класс окна

LPCSTR lpWindowName, Имя окна

DWORD dwStyle, Стиль создаваемого окна

int X, Начальное положение окна по горизонтали.

int Y, Начальное положение окна по вертикали.

int nWidth, Ширина окна.

int nHeight, Высота окна.

HWND hWndParent, Дескриптор родительского окна.

HMENU hMenu, Дескриптор меню.

HINSTANCE hInstance, Дескриптор экземпляра модуля

LPVOID lpParam Указатель на значение, которое будет передан в окно);

* **ShowWindow -** Устанавливает состояние показа указанного окна.

BOOL ShowWindow(

HWND hWnd, Дескриптор окна

int nCmdShow Управление отображением окна);

### 3.2.2 Advapi32.lib

* **OpenProcessToken** - открывает токен доступа, связанный с процессом.

BOOL OpenProcessToken(

HANDLE ProcessHandle, Дескриптор процесса

DWORD DesiredAccess, Маска доступа

PHANDLE TokenHandle Указатель на дескриптор);

* **AdjustTokenPrivileges** - включает или отключает привилегии в указанном токене доступа .

BOOL AdjustTokenPrivileges(

HANDLE TokenHandle, Дескриптор доступа

BOOL DisableAllPrivileges, Указывает, отключает ли функция все привилегии токена

PTOKEN\_PRIVILEGES NewState, Указатель на структуру TOKEN\_PRIVILEGES, которая определяет массив привилегий и их атрибуты.

DWORD BufferLength, Задает размер в байтах буфера

PTOKEN\_PRIVILEGES PreviousState, NULL

PDWORD ReturnLength NULL);

### 3.2.3 Kernel32.lib

* **HeapFree** - Освобождает блок памяти, выделенный функцией HeapAlloc

BOOL HeapFree(

HANDLE hHeap, Блок памяти которой должен быть освобожден

DWORD dwFlags,

LPVOID lpMem Указатель на освобождаемый блок памяти.);

* **OpenProcess** - Открывает существующий локальный объект процесса.

HANDLE OpenProcess(

DWORD dwDesiredAccess, Доступ к объекту процесса.

BOOL bInheritHandle, true\false наследование дескриптора

DWORD dwProcessId Идентификатор открываемого процесса.);

* **CloseHandle** - Закрывает дескриптор открытого объекта.

BOOL CloseHandle(

HANDLE hObject дескриптор открытого объекта.);

* **GetCurrentProcess**() - Извлекает идентификатор текущего процесса.
* **ReadFile** - Считывает данные из указанного файла

BOOL ReadFile(

HANDLE hFile, Дескриптор устройства

LPVOID lpBuffer, Указатель на буфер, который получает данные

DWORD nNumberOfBytesToRead, Максимальное количество байтов для чтения.

LPDWORD lpNumberOfBytesRead, Указатель на переменную, которая получает количество прочитанных байтов

LPOVERLAPPED lpOverlapped NULL);

* **HeapAlloc** - Выделяет блок памяти.

LPVOID HeapAlloc(

HANDLE hHeap, Дескриптор из которого будет выделена память.

DWORD dwFlags, Параметры распределения памяти.

SIZE\_T dwBytes ); Количество выделяемых байтов.

* **GetProcessHeap**() - Извлекает дескриптор вызывающего процесса.
* **GetFileSize** - Получает размер указанного файла в байтах.

DWORD GetFileSize(

HANDLE hFile, Дескриптор файла.

LPDWORD lpFileSizeHigh NULL.);

* **CreateFileA** - открывает файл. Функция возвращает дескриптор, который можно использовать для доступа к файлу

CreateFileA(

LPCSTR lpFileName, Имя файла или устройства

DWORD dwDesiredAccess, Запрошенный доступ к файлу или устройству

DWORD dwShareMode, Запрошенный режим общего доступа к файлу

LPSECURITY\_ATTRIBUTES lpSecurityAttributes, NULL

DWORD dwCreationDisposition, Действие над файлом

DWORD dwFlagsAndAttributes, Атрибуты и флаги файла или устройства

HANDLE hTemplateFile NULL);

* **CreateRemoteThread** - Создает поток, который выполняется в виртуальном адресном пространстве другого процесса.

CreateRemoteThread(

HANDLE hProcess, Дескриптор процесса

LPSECURITY\_ATTRIBUTES lpThreadAttributes, Указатель на структуру, которая определяет дескриптор безопасности

SIZE\_T dwStackSize, Начальный размер стека в байтах

LPTHREAD\_START\_ROUTINE lpStartAddress, Указатель на определяемую приложением функцию представляет начальный адрес потока в удаленном процессе

LPVOID lpParameter, Указатель на переменную, передаваемую в функцию потока

DWORD dwCreationFlags, Флаги, управляющие созданием потока.

LPDWORD lpThreadId Указатель на переменную, которая получает идентификатор потока.

);

* **WriteProcessMemory** - Записывает данные в область памяти указанного процесса. Вся область для записи должна быть доступна, иначе операция завершится ошибкой.

BOOL WriteProcessMemory(

HANDLE hProcess, Дескриптор изменяемой памяти процесса.

LPVOID lpBaseAddress, Указатель на базовый адрес в указанном процессе

LPCVOID lpBuffer, Указатель на буфер, содержащий данные

SIZE\_T nSize) Количество байтов, которое будет записано в указанный процесс

* **VirtualAllocEx** - Резервирует, фиксирует или изменяет состояние области памяти в виртуальном адресном пространстве указанного процесса

LPVOID VirtualAllocEx(

HANDLE hProcess, Дескриптор процесса

LPVOID lpAddress, Указатель на начальный адрес

SIZE\_T dwSize, Размер выделяемой области памяти

DWORD flAllocationType, Тип распределения памяти

DWORD flProtect); Защита памяти для выделяемой области страниц

* **VirtualProtect** - Изменяет защиту области зафиксированных страниц в виртуальном адресном пространстве вызывающего процесса. VirtualProtect(

LPVOID lpAddress, - Адрес начальной страницы атрибуты доступа которых необходимо изменить

SIZE\_T dwSize, Размер области

DWORD flNewProtect, Опция защиты памяти

PDWORD lpflOldProtect Указатель на значение защиты доступа

## 3.3 Разработка и реализация алгоритмов приложения

### 3.3.1 Общий алгоритм работы приложения.

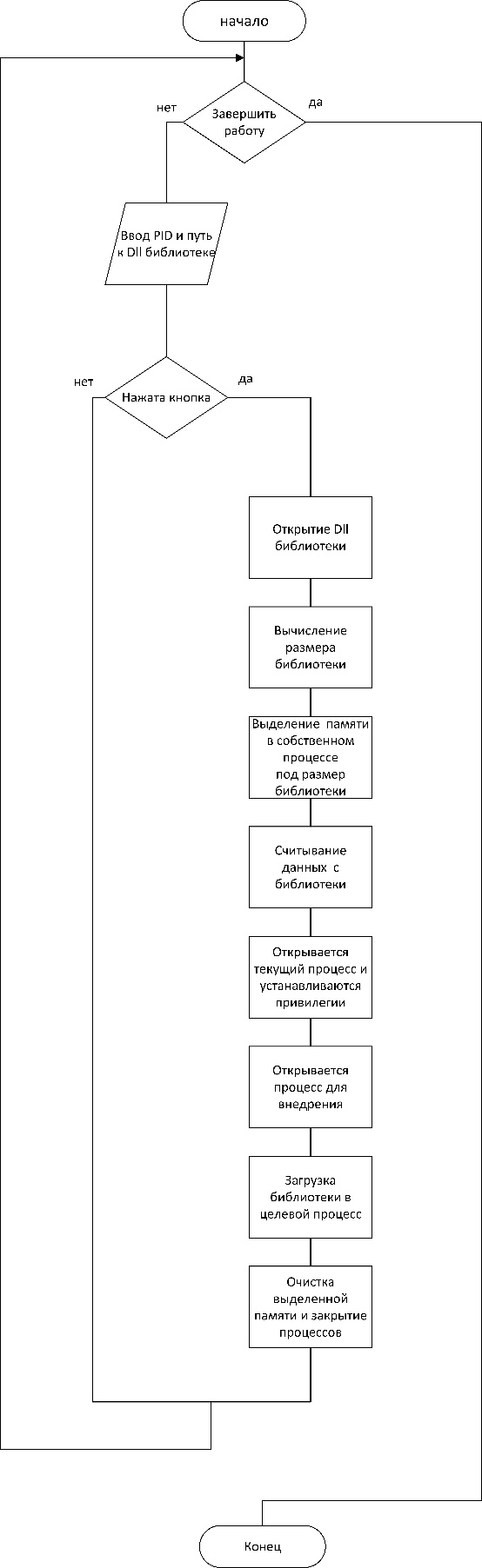


Рисунок 4 – Алгоритм работы приложения

### 3.3.2 Основные алгоритмы приложения:

LoadRemoteLibraryR- Загрузка удаленной библиотеки в целевой процесс (Рисунок 5).

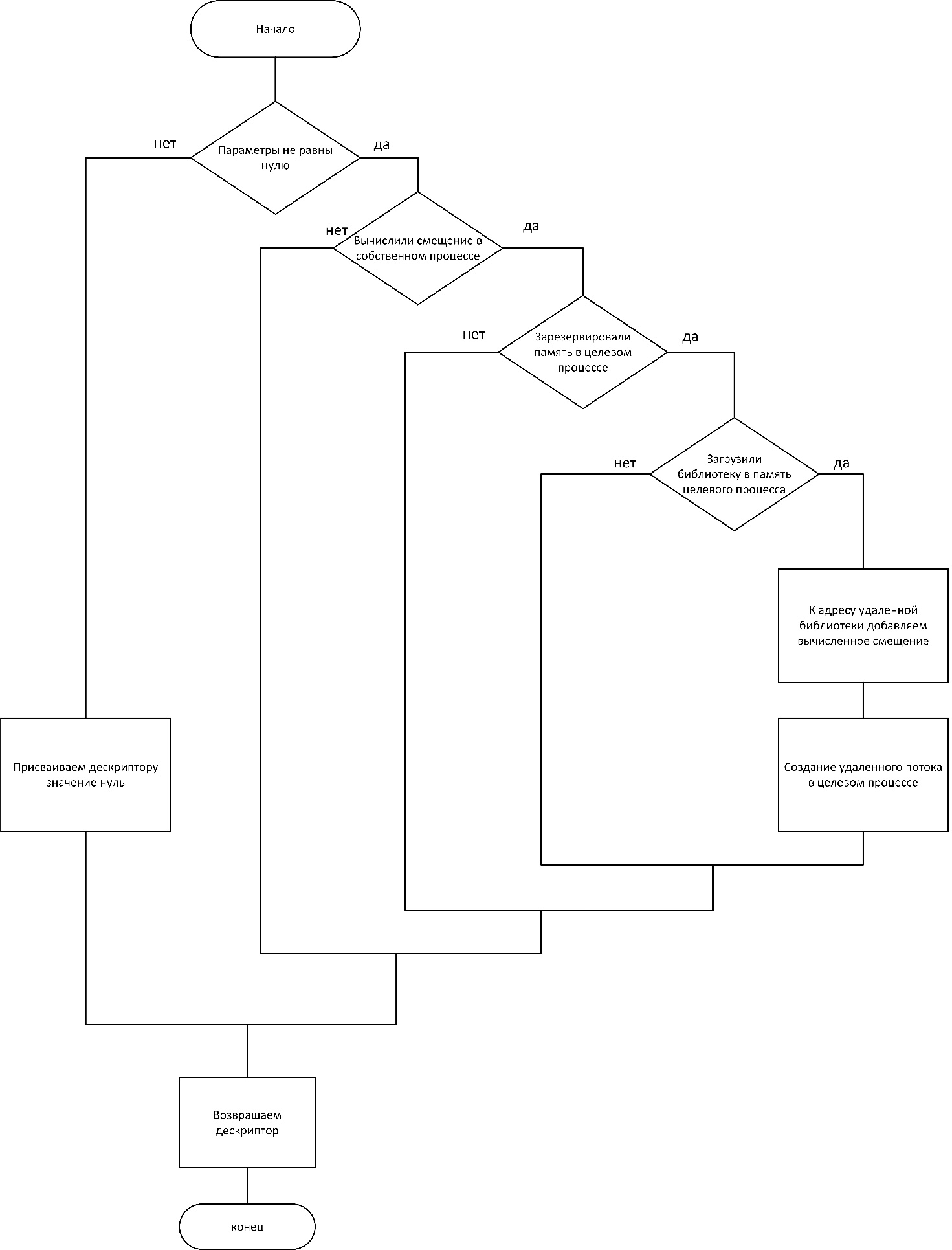


Рисунок 5 – Загрузка удаленной библиотеки

GetReflectiveLoader – проверяет есть ли в библиотеке функция ReflectiveLoader и возвращает смещение в памяти процесса где находиться эта функция (Рисунок 6).

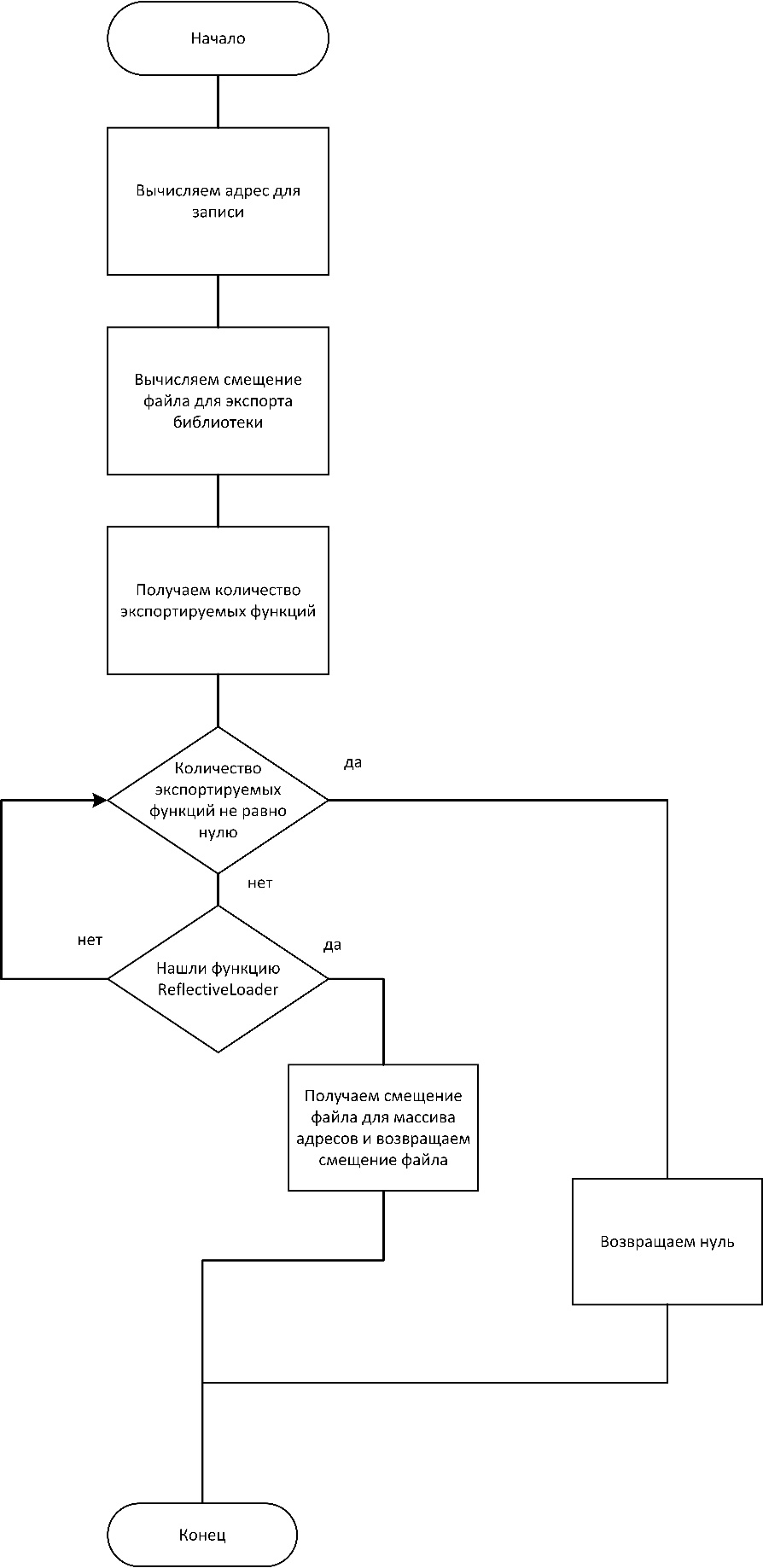


Рисунок 6 – Поиск функции ReflectiveLoader

## 3.4 Разработка интерфейса взаимодействия пользователя с системой

Окно программы должно состоять из двух текстовых полей для ввода информации, где первое поле для ввода идентификатора процесса, а второе для ввода пути где располагается dll библиотека. А также должна быть кнопка при нажатии на которую происходит внедрение в указанный целевой процесс.

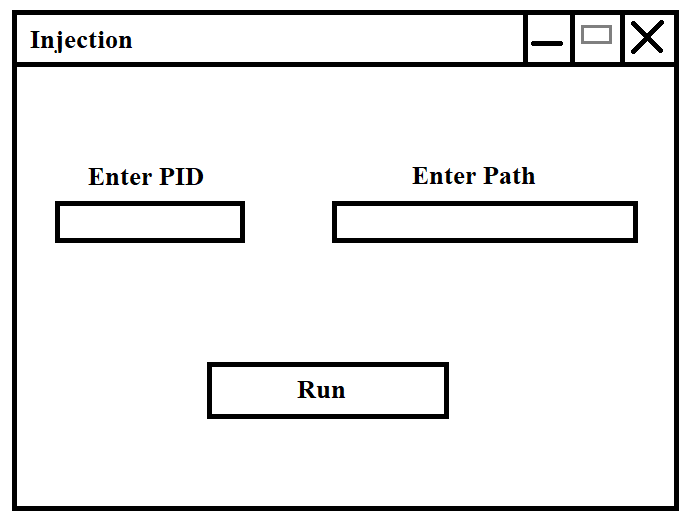


Рисунок 6 – Макет главного окна приложения

При нажатии на кнопку «Run» появляется информационное окно с информацией о внедрении, если внедрение произошло успешно появиться окно (Рисунок 7) в ином случае появиться окно (Рисунок 8) с информацией почему не получилось внедрить dll библиотеку в целевой процесс.

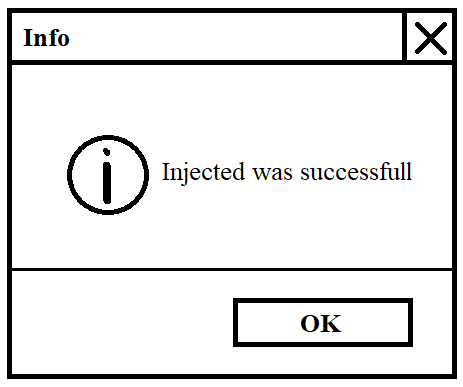


Рисунок 7 – MessegeBox если внедрение произошло успешно

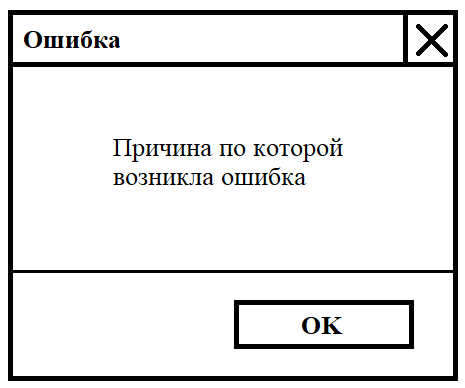


Рисунок 8 – MessegeBox если внедрйение не удалось

# 4. ПРОЕКТНО-ТЕХНОЛОГИЧИСКАЯ ЧАСТЬ

## 4.1. Тестирование и отладка макета рабочей программы.

Пользователь не вводит уникальный идентификатор процесса (Рисунок 9), то ошибка будет обработана и на экране появиться сообщение (Рисунок 10). Такая ошибка появиться в случае, если процесса не существует или не введен уникальный идентификатор процесса.

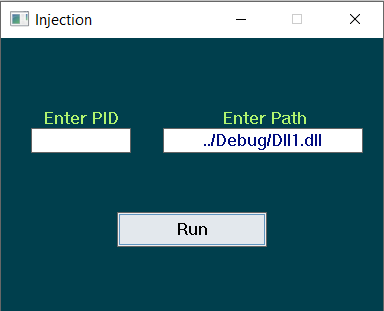


Рисунок 9 – Пользователь не вводит уникальный идентификатор процесса и нажимает на кнопку

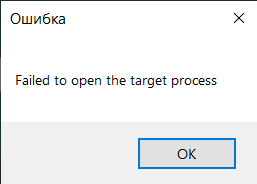


Рисунок 10 – Сообщение о неудачном открытии процесса

Пользователь прописывает путь к несуществующей библиотеке или к файлу, который не имеет расширение .dll (Рисунок 11). Появляется сообщение с ошибкой открытия файла (Рисунок 12).

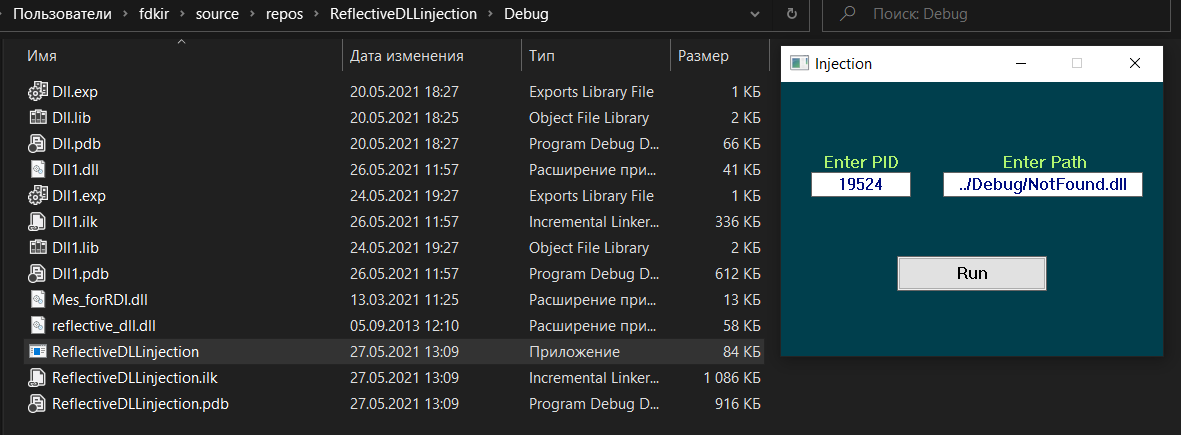


Рисунок 11 – Ввод несуществующей библиотеки

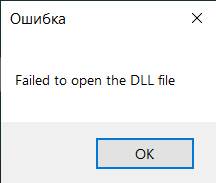


Рисунок 12 – Сообщение при неудачном открытии библиотеки

Пользователь пытается заполнить уникальный идентификатор символами, которые недопустимы при вводе (Рисунок 13). Появляется предупреждение, что это поле можно заполнить только цифрами.

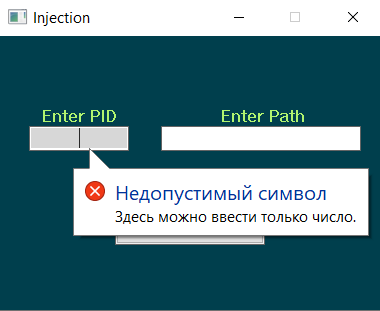


Рисунок 13 – Предупреждение о заполнение полей

Ошибка, при которой не получилось вычислить размер внедряемой библиотеки (Рисунок 14)

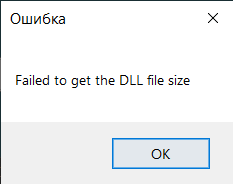


Рисунок 15 – Сообщение при неудачном вычислении размера библиотеки

Ошибка если не получилось выделить память для загрузки библиотеки в процесс (Рисунок 16)

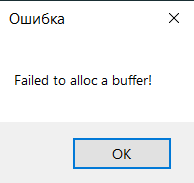


Рисунок 16 – Сообщение при неудачном выделении памяти

Ошибка (Рисунок 17) возникает если не получилось создать поток в целевом процессе, для загрузки библиотеки.

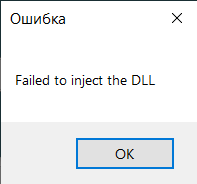


Рисунок 17 – Сообщение при неудачном создании потока

Пользователь не вводит уникальный идентификатор и путь к библиотеке (Рисунок 18). Появляется сообщение об ошибке и с просьбой заполнить поля для ввода (Рисунок 19)

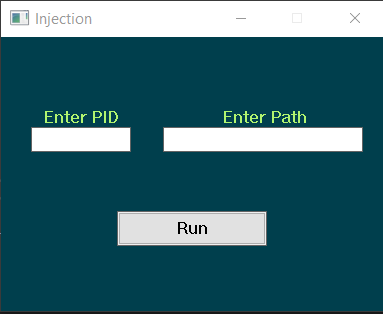


Рисунок 18 – Пользователь не заполнил необходимые поля

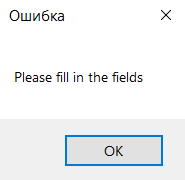


Рисунок 19 – Ошибка с просьбой заполнить нужные поля

Если внедрение произошло успешно, то на экране появиться сообщение с успешным внедрением в целевой процесс (Рисунок 20)

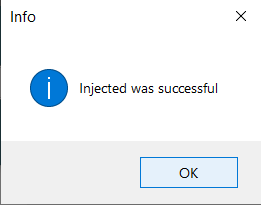


Рисунок 20 – Внедрение произошло успешно

## 4.2. Разработка руководства пользователя и администратора.

### 4.2.1 Разработка руководства администратора

Для того чтобы запустить приложение нужно загрузить архив ReflectiveDllInjection.rar на устройство с операционной системой Windows 10.

Разархивировать архив в любое место на компьютере с помощью программы для архивирования и разархивирования. На рисунке 21 демонстрируется разархивирование с помощью программы WinRar

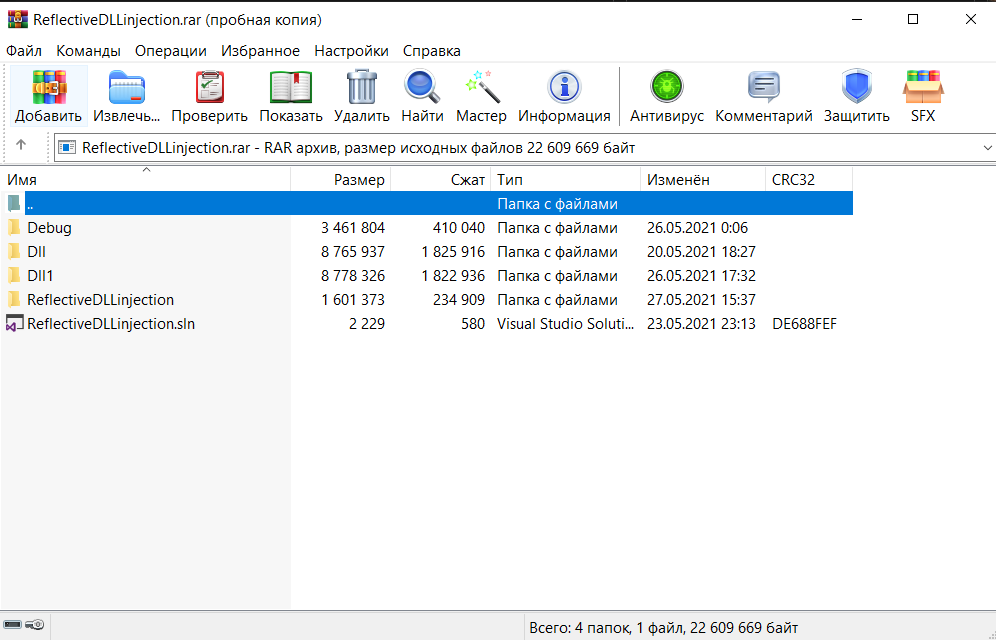


Рисунок 21 – Открытие архива с программой

Для того чтобы извлечь все файлы нужно нажать на кнопку «Извлечь» на верхней панели (Рисунок 22). И выбрать путь для извлечения или указать путь вручную.

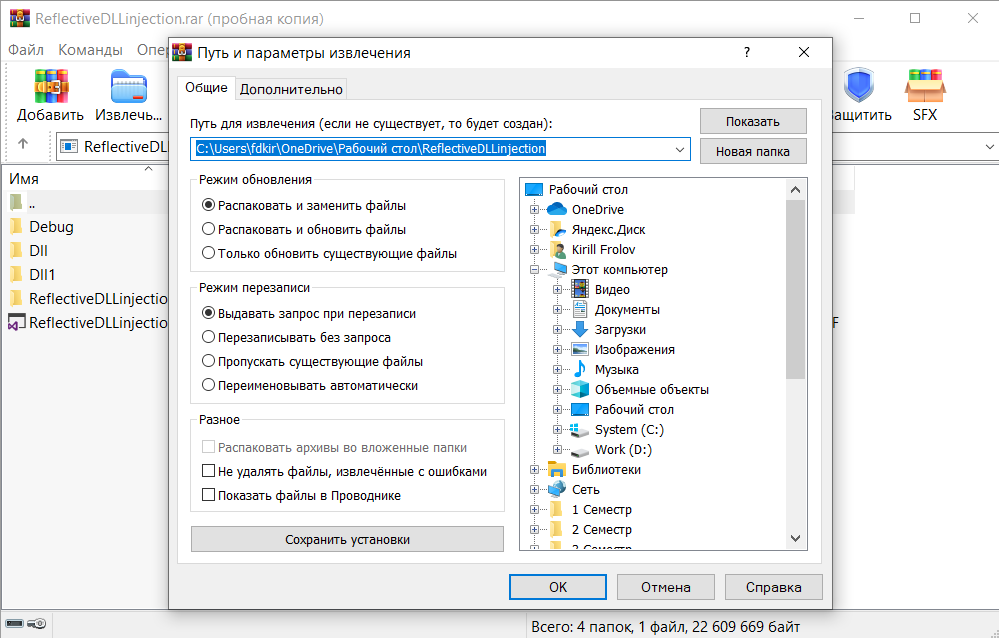


Рисунок 22 – Выбор папки для разархивирования

После разархивирования в папке «Debug» будет лежать исполняемый файл ReflectiveDllInjection.exe (Рисунок 23), его необходимо запустить с правами администратора, для правильной работы приложения.

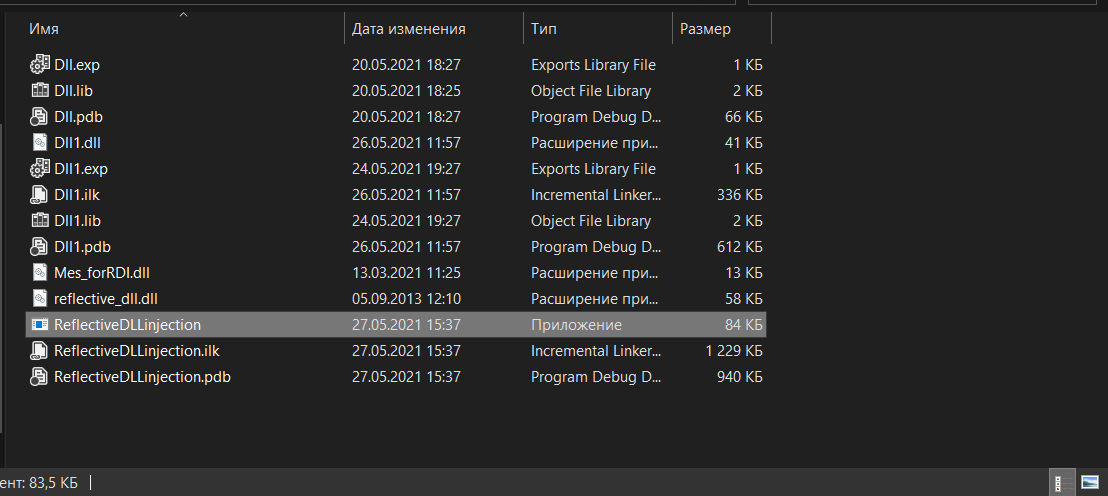


Рисунок 23 – Выбор файла для запуска приложения

### 

### 4.2.2 Разработка руководства пользователя

Чтобы открыть приложение нужно запустить исполняемый файл ReflectiveDLLinjection.exe от имени администратора. После запуска открывается окно приложения (Рисунок 24)

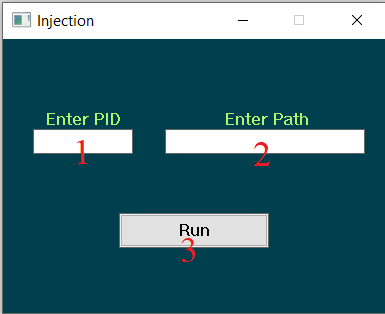


Рисунок 24 – Окно приложения

Под номером 1 расположено поле для ввода уникального идентификатора процесса, который можно увидеть в диспетчере задач во вкладке подробности или в другой иной программе, которая позволяет посмотреть уникальный идентификатор процесса.

Открыть диспетчер задач можно комбинацией клавиш ctrl+shift+esc или в меню поиска ввести «Диспетчер задач» и нажать на иконку (Рисунок 25) или любым другим способом.

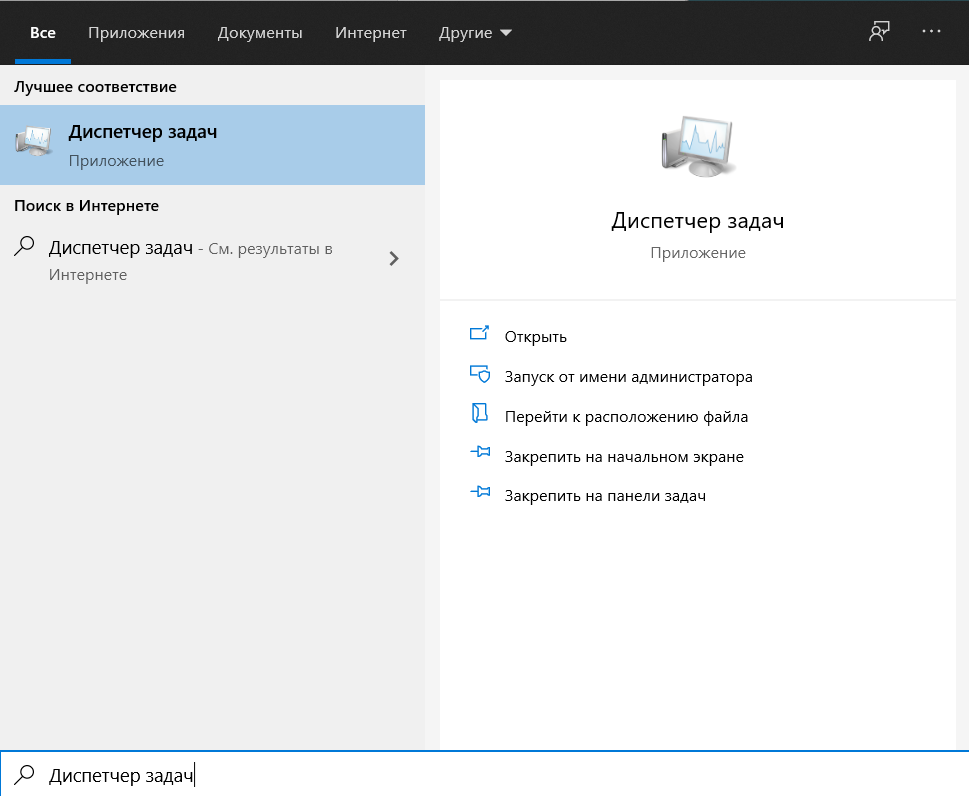


Рисунок 25 – Поиск диспетчера задач

Когда открылся диспетчер задач необходимо перейти на вкладку «Подробности» в верхнем меню

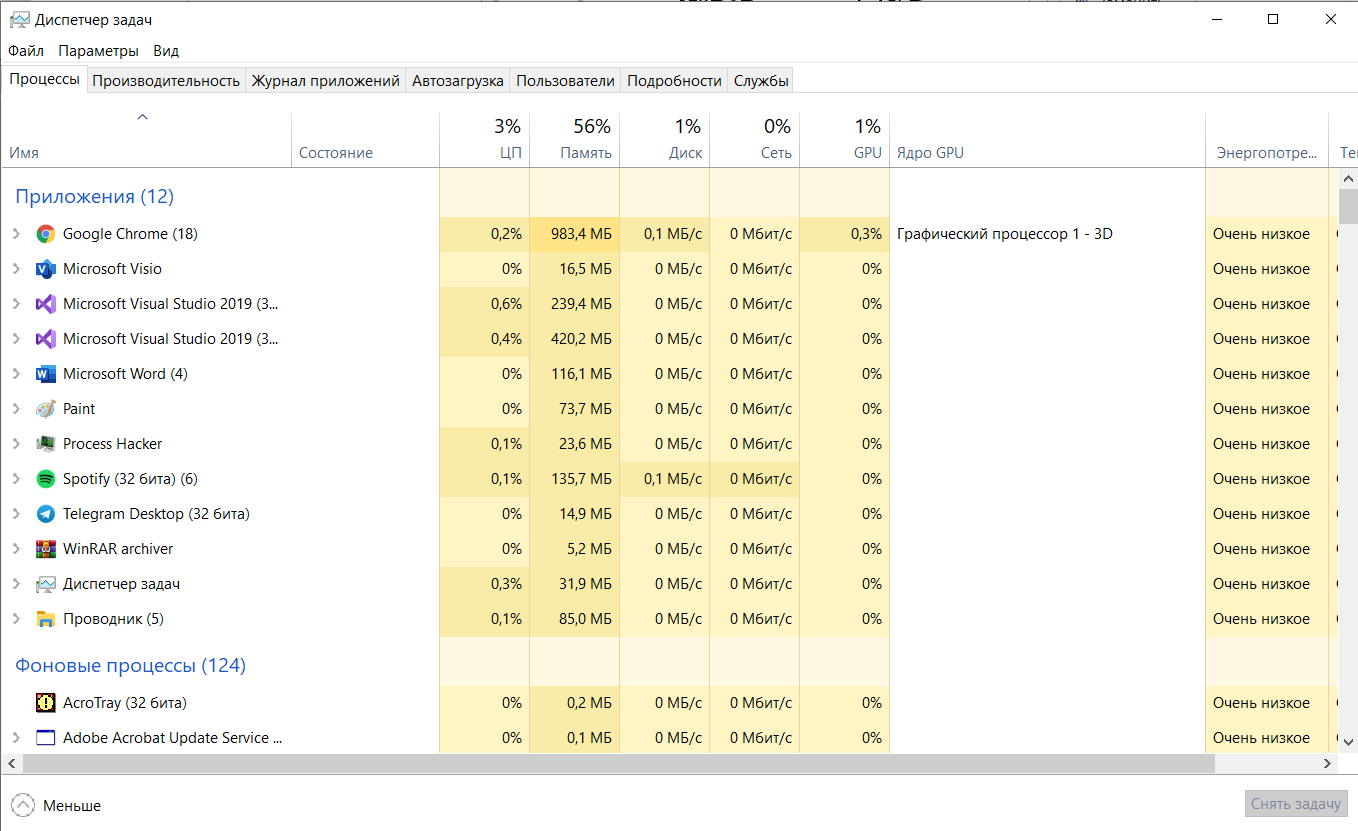


Рисунок 26 – Главное окно диспетчера задач

В столбце с названием «Имя» пишется название процесса, в столбце «ИД процесса» записаны уникальные идентификаторы процессов (Рисунок 27). Для того чтобы выбрать уникальный идентификатор процесса выбираем название в столбце «Имя» и напротив него в столбце «ИД процесса» записываем цифры в приложение в поле 1 с названием «Enter PID»

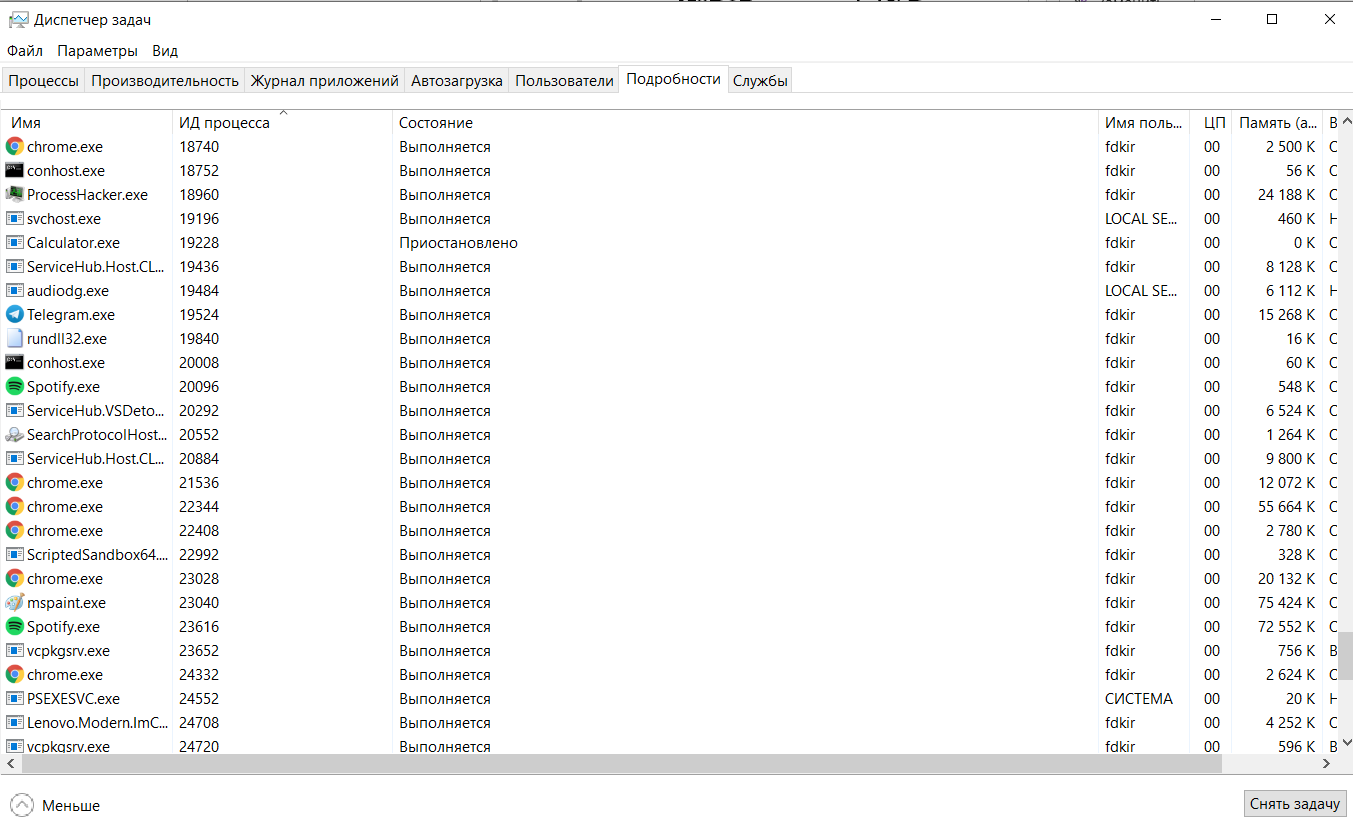


Рисунок 27 – выбор уникального идентификатора процесса

Под номером 2 расположено поле для ввода пути, где расположена внедряемая библиотека. Путь можно посмотреть в проводнике Windows или в любой другой программе, которая позволяет скопировать путь, где лежит внедряемая библиотека. Нажимаем на кнопку (Рисунок 28) поиска которая расположена в левом нижнем углу экрана и вводим в поисковой строке «Проводник» и открываем его с помощью нажатия на иконку проводника.



Рисунок 28 – Кнопка поиска

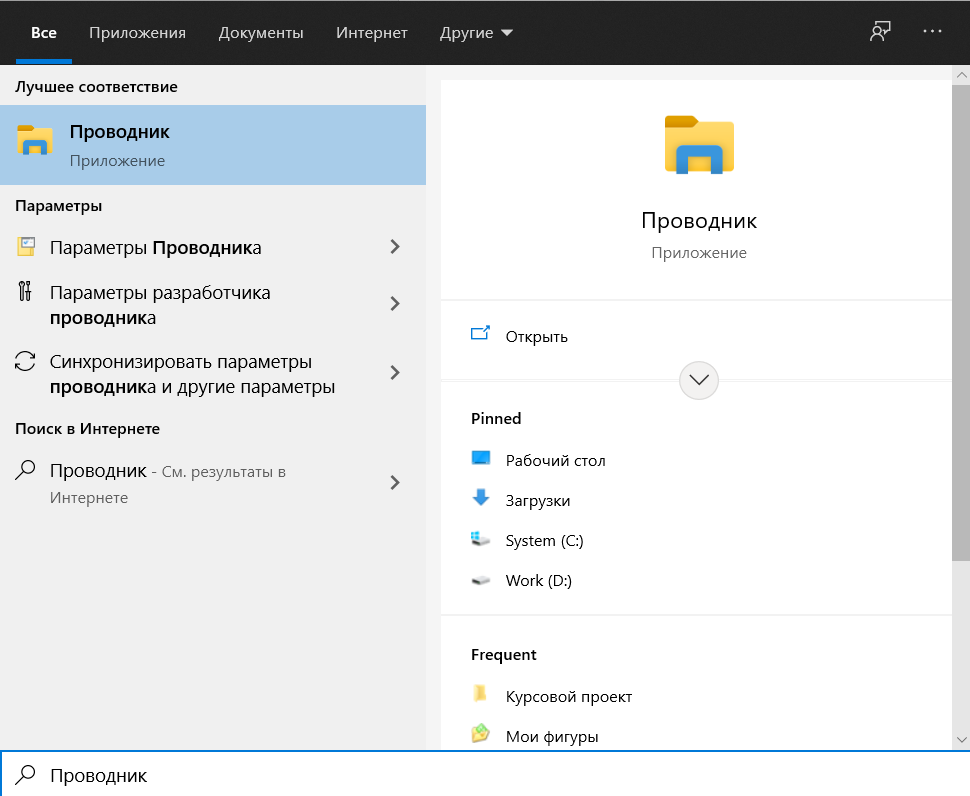


Рисунок 29 – Поиска проводника

Теперь переходим в папку где лежит библиотека с расширением .dll и нажимаем на поле которое расположено в верхней части окна, после нажатия сразу выделяется путь к этой папке. После этого необходимо скопировать этот путь с помощью клавиш ctrl+c или нажать на правую кнопку мыши и выбрать пункт скопировать или записать путь в ручную в приложение в поле номер 2 с названием «Enter path». И добавить символ «/» и название библиотеки с расширением .dll

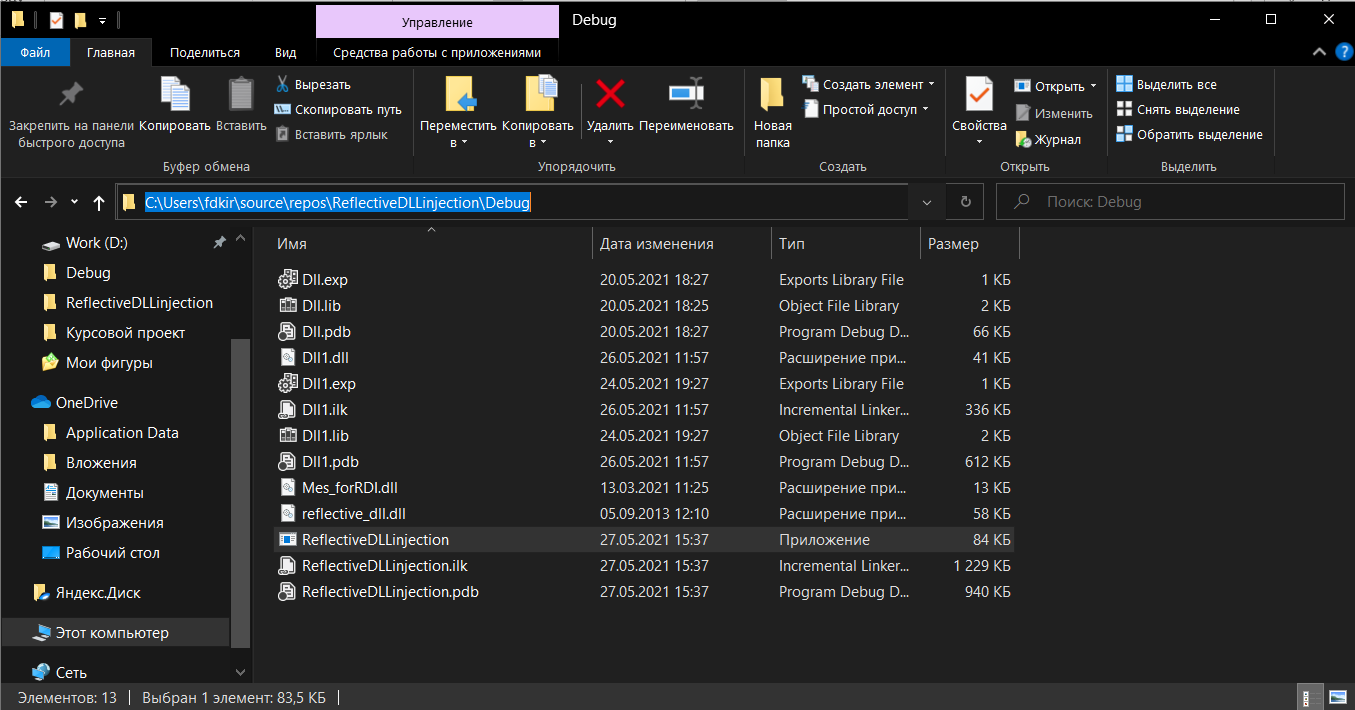


Рисунок 30 – Копирование пути к приложению

Под номером 3 расположена кнопка, при нажатии на которую происходит внедрение в процесс, указанный в поле под номером 1, и библиотека, указанная в поле под номером 2 загрузиться в этот процесс.

Пример: внедрение в мессенджер «Телеграмм» с выводом MessegeBox

1. Вводим PID (уникальный идентификатор процесса) и путь к расположению внедряемой dll библиотеки (Рисунок 31)

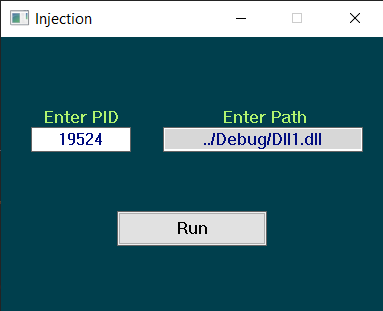


Рисунок 31 – заполнение полей приложения

Видим сообщение об удачном внедрении (Рисунок 32)

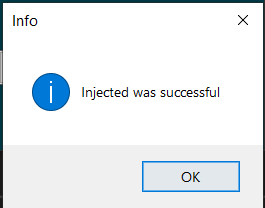


Рисунок 32 – Сообщение об удачном внедрении

При загрузке dll библиотеки в мессенджер «Телеграмм» вызываться MessegeBox (Рисунок 33)

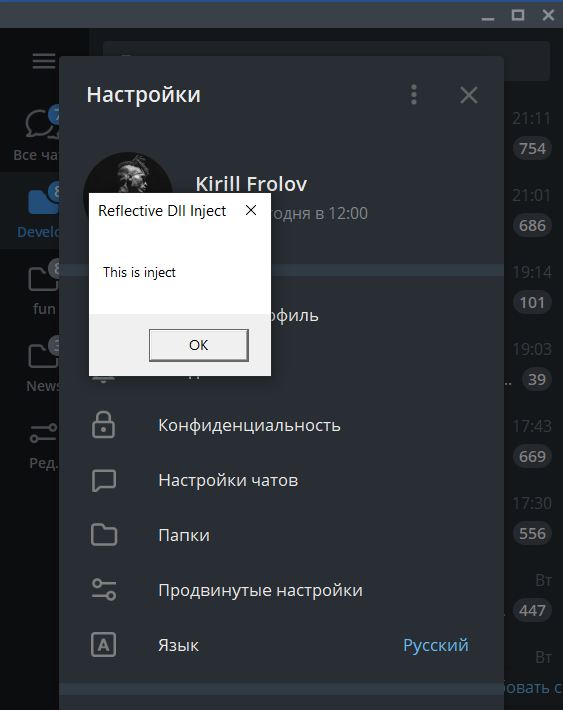


Рисунок 33 – вызов MessegeBox при загрузке dll

На рисунке 34 видно, что MessegeBox был вызван в адресном пространстве



Рисунок 33 – MessegeBox вызван в адрессном пространстве «Телеграмм»

# 5. ЗАКЛЮЧЕНИЕ

В ходе выполнения данной курсовой работы была реализована технология Reflective DLL injection.

Были сформированы навыки по разработке и реализации программного приложения с использованием интерфейса прикладного программирования (АРI) операционных систем.

В данной работе выполнены все поставленные задачи.

Reflective dll injection – уникальная в своем роде технология которая позволяет внедрить dll библиотеку так, что это будет незаметно для системы. А также от внедрения библиотек зависят Windows и многие ее приложения. Такая технология была разработан для законных целей, но может использоваться в плохих целях. Несмотря на то, что противодействовать методам внедрения процессов сложно, глубокая защита по-прежнему эффективна для противодействия другим этапам жизненного цикла заражения вредоносным ПО. Поэтому технология reflective dll injection эффективна для внедрения, потому что она обходит большинство защит процесса.

В будущем можно усовершенствовать разработанное приложение путем добавления дополнительного функционала (например выбор файла при нажатии на поле ввода пути к библиотеке или трансляции запущенных процессов).

# 6. СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТУРАТУРЫ

1. Техническая документация [Электронный ресурс] – URL: <https://docs.microsoft.com/ru-ru/documentation>
2. Справочник программирования приложений Windows (API) [Электронный ресурс] – URL: <https://docs.microsoft.com/en-us/windows/win32/apiindex/windows-api-list>
3. WinAPI создание классов и основные функции [Электронный ресурс] – URL: <https://habr.com/ru/post/130093/>
4. Фостер Дж. - Защита от взлома: сокеты, эксплойты, shell-код: Издательство "ДМК Пресс" г 2008 с 784 – [Электронный ресурс]// Лань: электронно-библиотечная система. - URL: <https://e.lanbook.com/book/1117>
5. Диогенес Ю. , Озкайя Э. - Кибербезопасность. стратегия атак и обороны: Издательство "ДМК Пресс" г 2020 с326 784 – [Электронный ресурс]// Лань: электронно-библиотечная система. - URL: https://e.lanbook.com/book/131717
6. Reflective DLL injection принцип работы [Электронный ресурс] – URL: <https://0x00sec.org/t/reflective-dll-injection/3080>
7. DLL Injection принцип работы [Электронный ресурс] – URL: <https://m.habr.com/ru/post/73324/>