# 1 ЦЕЛЬ РАБОТЫ

Изучение и практическое освоение техники внедрения собственного кода в целевой процесс для перехвата вызовов функций и изменения его поведение.

# 2 ЗАДАЧИ РАБОТЫ

В рамках выполнения лабораторной работы необходимо разработать программу и dll-библиотеку, которые обеспечивают два режима работы:

1. Отслеживание вызов указанных заранее функций. Каждый вызов функции в целевом процессе должен сопровождаться печатью сообщения на консоль разрабатываемой программы. Распечатанное сообщение должно содержать имя функции и метку времени.

2. Изменение поведение функции FindFirstFile, FindNextFile, CreateFile таким образом, что для целевого процесса пропадает указанный заранее файл. Для изменения поведения достаточно изменять значения, возвращаемые этими функциями в соответствии с определенными входными параметрами.

# 3 ХОД РАБОТЫ

## 3.1 Тестовая программа

На данном этапе была разработана тестовая программа на языке C++, которая использует API-функцию CloseHandle из библиотеки kernel32.dll. Программа создает файл test.txt с помощью функции CreateFileA, а затем закрывает его дескриптор с помощью CloseHandle.

Исходный код программы:

```
#include <windows.h>
#include <iostream>

int main() {
    HANDLE hFile = CreateFileA("test.txt", GENERIC_WRITE, 0, NULL,
CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
    if (hFile != INVALID_HANDLE_VALUE) {
        std::cout << "File created successfully." << std::endl;
```

```
        CloseHandle(hFile);
    }
    else {
        std::cerr << "Failed to create file." << std::endl;
    }
    return 0;
}
```

Для анализа вызова функции CloseHandle была использована IDA Pro. Результаты показаны на Рисунок 1.

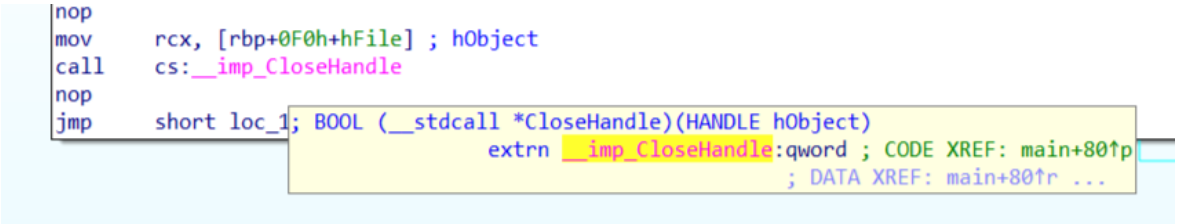


Рисунок 1 – Анализ вызова функции CloseHandle

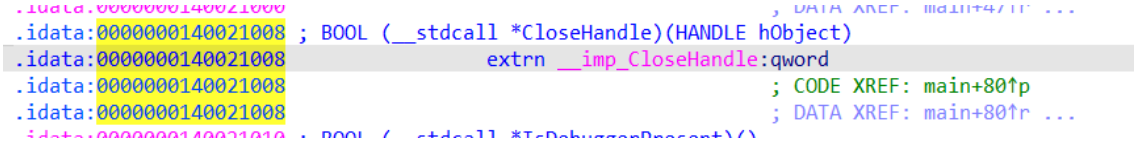


Рисунок 2 – Анализ вызова функции CloseHandle

Для осуществления перехвата вызова функции необходимо переписать первые несколько байт по адресу функции – поместить туда вызов своей функции или прыжок по некоторому адресу.

### 3.2 Разработка класса HookPatch

Для того чтобы осуществить перехват, был разработан класс HookPatch и класс, управляющий перехватами HookManager. Класс HookManager выполняет загрузку библиотеки kernel32.dll и поиск в ней адреса целевой (CloseHandle) функции. Далее она передаёт адрес целевой функции и адрес функции-логирования на вход конструктору HookPatch, который выполняет создание патча в теле функции – заменяет первые байты на jmp [rip + 0] hook_addr – это прыжок на адрес начала функции-логирования. Такой подход был выбран так как адреса целевой и лог-функций находились слишком далеко и прыжок по смещению не показал желаемого результата. Важным при создании патча было также

сохранение оригинальных байт из начала целевой функции для восстановления при снятии патча, а также изменение прав доступа к региону памяти при помощи VirtualProtect для того, чтобы иметь возможность изменять ассемблерный код целевой функции.

В результате работы программы были получены следующие сообщения при неоднократном вызове целевой функции:

```
[2025-02-27 14:24:52] Function called: 'CloseHandle'
[2025-02-27 14:31:37] Function called: 'CloseHandle'
[2025-02-27 14:41:17] Function called: 'CloseHandle'
```

### 3.3 Разработка библиотеки winhooklib.dll

Далее были разработаны библиотека winhooklib.dll и программа-монитор monitor.exe, осуществляющая загрузку библиотеки в целевой процесс.

Процесс внедрения библиотеки состоит в следующем:

1. Монитор (monitor.exe) открывает целевой процесс с помощью функции OpenProcess. Для этого требуется PID (идентификатор процесса), который может быть получен либо напрямую от пользователя (через аргумент --pid), либо путем поиска процесса по имени (с использованием NtQuerySystemInformation).

2. Для загрузки библиотеки в целевой процесс необходимо выделить память под путь к DLL. Это делается с помощью функции VirtualAllocEx.

3. Путь к библиотеке записывается в выделенную память с помощью функции WriteProcessMemory.

4. Для загрузки библиотеки в целевом процессе создается удаленный поток с помощью функции CreateRemoteThread. Этот поток выполняет функцию LoadLibraryA, передавая ей путь к DLL.

5. Монитор ожидает завершения удаленного потока с помощью функции WaitForSingleObject.

6. После завершения работы удаленного потока освобождается выделенная память и закрываются дескрипторы.

Для передачи сообщений между библиотекой и монитором был выбран механизм именованных каналов. Это механизм взаимодействия IPC, предоставляемый Windows, который позволяет передавать данные между процессами.

Монитор принимает параметры командной строки, такие как PID целевого процесса, его имя и список функций, которые необходимо отслеживать. После получения этих данных монитор внедряет библиотеку winhooklib.dll в целевой процесс. Библиотека (winhooklib.dll) внедряется в целевой процесс, после внедрения библиотека создает именованный канал с именем \\.\pipe\MonitorPipe, который служит для обмена данными с монитором. Далее монитор подключается к именованному каналу \\.\pipe\MonitorPipe, созданному библиотекой, и отправляет в него конфигурацию, например, список функций для перехвата. В процессе работы монитор получает сообщения от библиотеки, содержащие информацию о вызовах функций, и выводит эту информацию на экран, предоставляя пользователю возможность отслеживать активность в целевом процессе.

Так как библиотека не позволяет снимать хук каждый раз по завершении выполнения целевой функции, а также требуется сохранить функционал оригинальных целевых функций, для реализации перехвата класс HookPatch был доработан посредством добавления трамплина. Теперь оригинальные байты записываются в начало выделенной исполняемой памяти, в конце функции-логирования управление передаётся на них, а после оригинальных байт при помощи той же схемы прыжка, что и в перехвате – на продолжение оригинальной целевой функции.

### 3.4 Изменения в реализации монитора и библиотеки

Выбранный способ связи библиотеки и монитора показал себя неудобным, поэтому было принято решение перейти на механизм сокетов.

Программа-монитор создает сокет и устанавливает соединение с сервером (локальный хост 127.0.0.1 на порту 12345). После успешного соединения

программа отправляет конфигурационные данные (например, функцию для мониторинга или путь к файлу для сокрытия) через сокет. Программа запускает отдельный поток, который читает сообщения от DLL через сокет. Если соединение разрывается или целевой процесс завершается, программа завершает свою работу.

При загрузке DLL инициализирует Winsock и создает сокет для прослушивания входящих соединений. В отдельном потоке (SocketCommunicationThread) программа читает данные из сокета. Сокет привязывается к локальному адресу 127.0.0.1 и порту 12345. Программа ожидает подключения внешнего приложения (программы, которая внедрила эту DLL).

Если получено сообщение с командой hook:, программа устанавливает хук на указанную функцию с помощью HookManager. Если получено сообщение с командой hide:, программа скрывает указанный файл (реализация скрытия файла находится в HookManager).

Целью режима hook является перехват вызовов определенных функций. В методе HookManager::set_hook программа получает адрес оригинальной функции с помощью вызова GetProcAddress (из kernel32.dll для перенаправления jmp и из kernelbase.dll для выполнения оригинальных инструкций). Затем создается объект HookPatch и вызывается метод install_hook для замены инструкций.

Программа выделяет память рядом с адресом из kernel32.dll, сохраняет к себе оригинальный прыжок и заменяет его на прыжок в выделенную память. В выделенной памяти помещается прыжок на ассемблерную вставку, которая вызывает адрес с байтами оригинальной функции, сохраняет результат исполнения, вызывает логгер и восстанавливает результат исполнения после него.

Чтобы учесть сохраненные оригинальные байты и исполнить в том числе их, при вызове оригинальной функции управление передается сначала на них, потом на kernelbase, это всё позволяют делать предварительно расставленные инструкции прыжков.

Режим сокрытия реализует хук с трамплином посредством внедрения функций-заместителей, которые внутри по трамплину вызывают оригинальные

функции и исходя из входных параметров принимают решение об изменении возвращаемого значения согласно заданию.

### 3.5 Результат работы программы

Результаты работы программы в каждом из режимов на примере приложения Notepad показано на Рисунок 3-4.

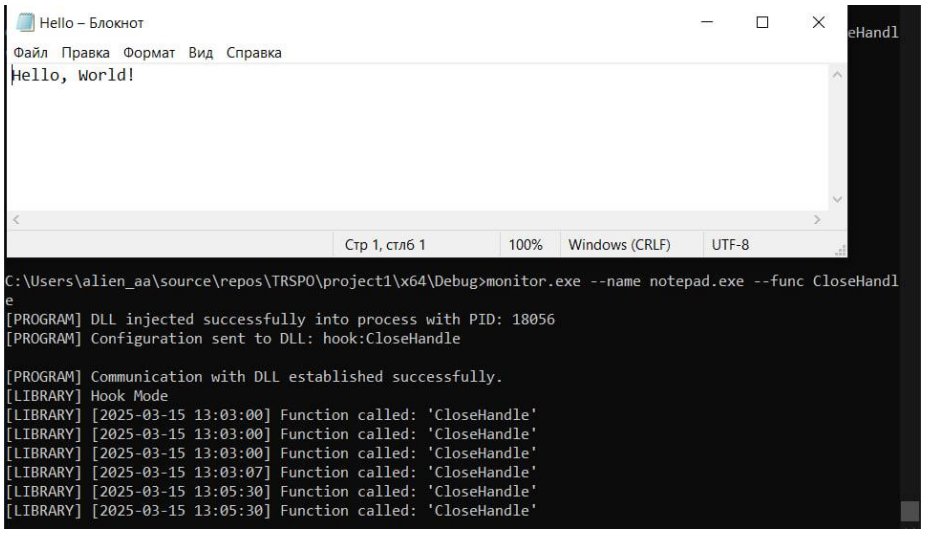


Рисунок 3 – Результат работы программы в режиме логирования

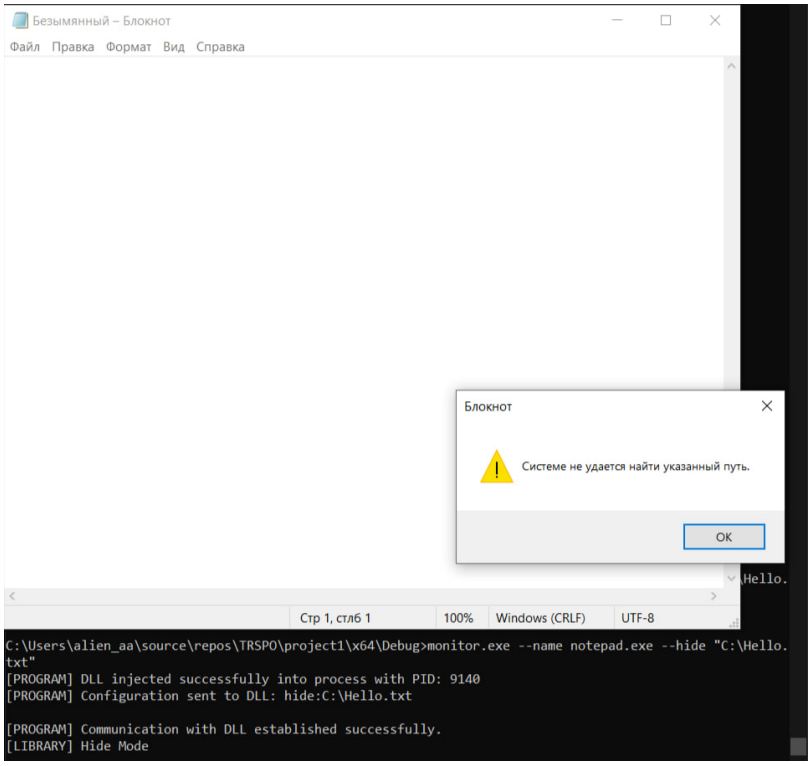


Рисунок 4 – Результат работы программы в режиме сокрытия файла

# 4 ВЫВОД

В ходе лабораторной работы были изучены техники внедрения собственного кода в целевой процесс для перехвата вызовов функций и изменения его поведения. Была разработана библиотека, которая внедряется в целевой процесс и осуществляет перехват. Перехват был выполнен посредством создания трамплина и перезаписи в него оригинальных байтов целевой функции, что позволило сохранить работоспособность программы после внедрения собственного кода.