Федеральное государственное автономное образовательное учреждение высшего образования

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ «ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Московский институт электроники и математики им. Тихонова

Департамент электронной инженерии

ОТЧЕТ
О ПРАКТИЧЕСКОЙ РАБОТЕ №3

по дисциплине «Системное программирование»

«Основы дизассемблирования»

Студент гр. БИБ201

Шадрунов Алексей

Дата выполнения: 12 января 2023 г.

Преподаватель:

Смирнов Д. В.

«___» _____ 2023 г.

Москва, 2023

Содержание

1 Задание на практическую работу

Скомпилировать программу на языке C. Дизассемблировать полученный исполняемый файл с помощью x64dbg. Проследить за работой программы с помощью отладчика и Process Monitor и Process Hacker.

2 Ход работы

2.1 Выделение памяти с помощью malloc в main

Установим точку прерывания на месте последнего вызова функции перед завершением программы (cexit). Этот вызов соответствует функции main (Рисунок 1).

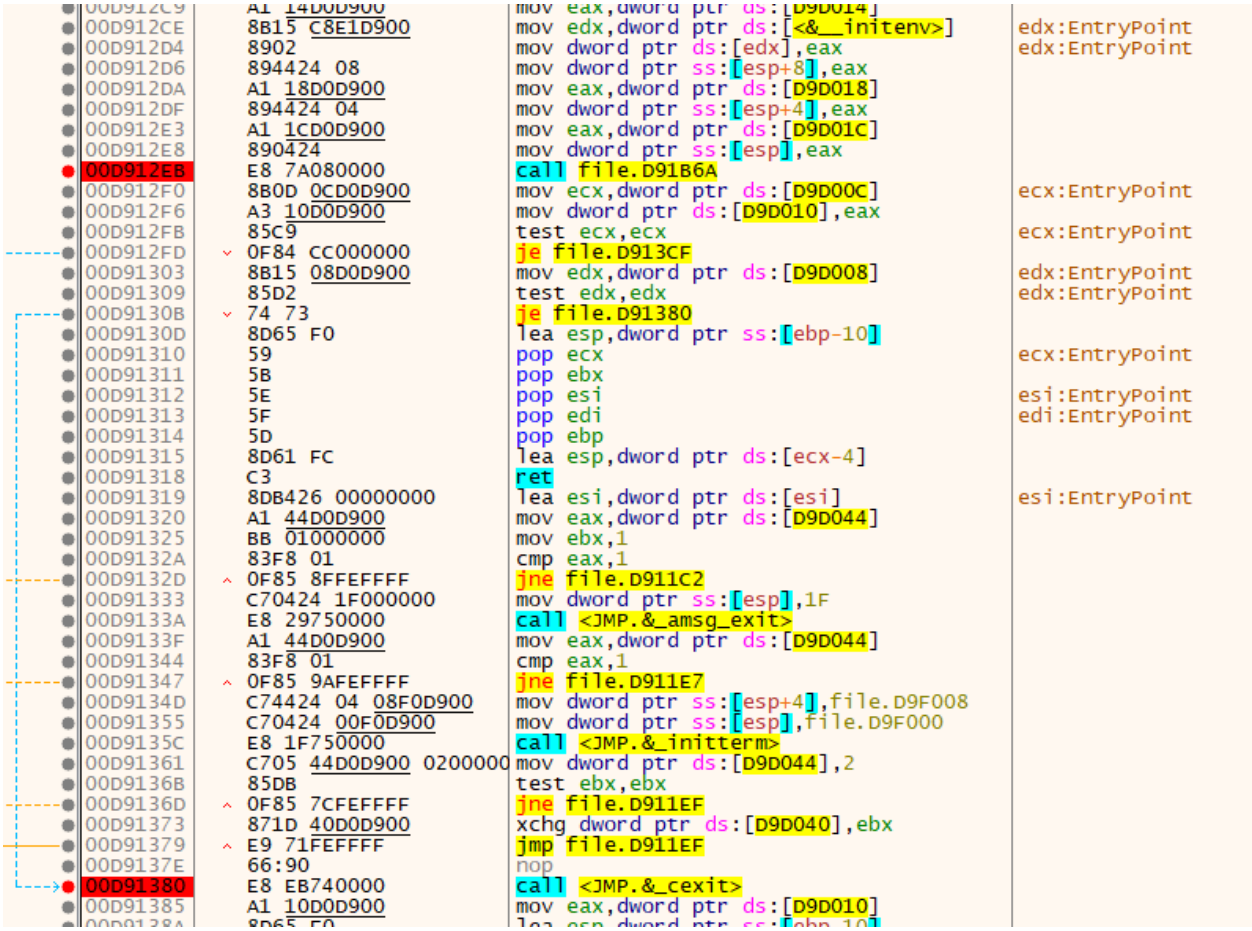

Рисунок 1 – Последний вызов перед выходом

После перехода по адресу функции мы видим два вызова (соответствуют функциям printf в исходном файле), инициализацию переменных и функции выделения памяти malloc (Рисунок 2). Выделяется память под переменную buf (куда попадают вводимые данные с консоли) и переменную command (первое слово из buf).

После вызова malloc в регистр EAX сохраняется адрес выделенной ячейки памяти. На рисунке 3 видно, что этот участок памяти заполнен случайными значениями.

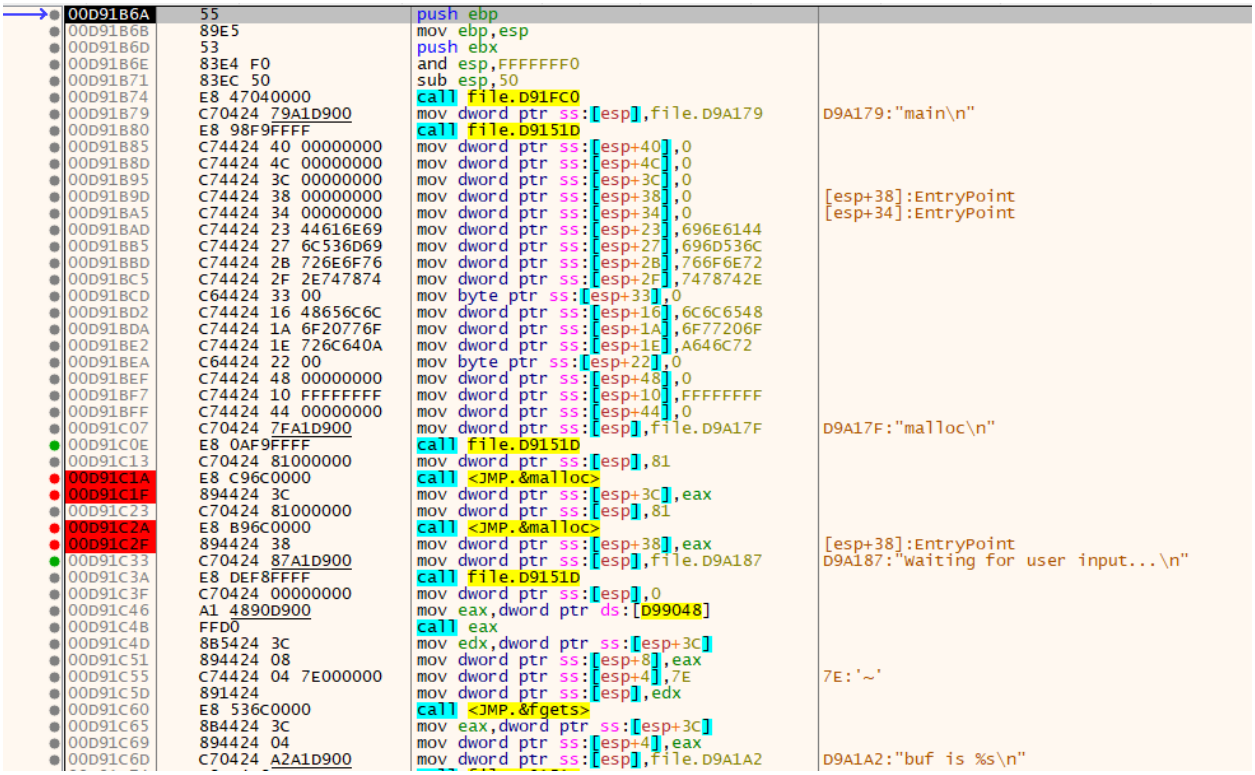

Рисунок 2 – Выделение памяти с помощью malloc

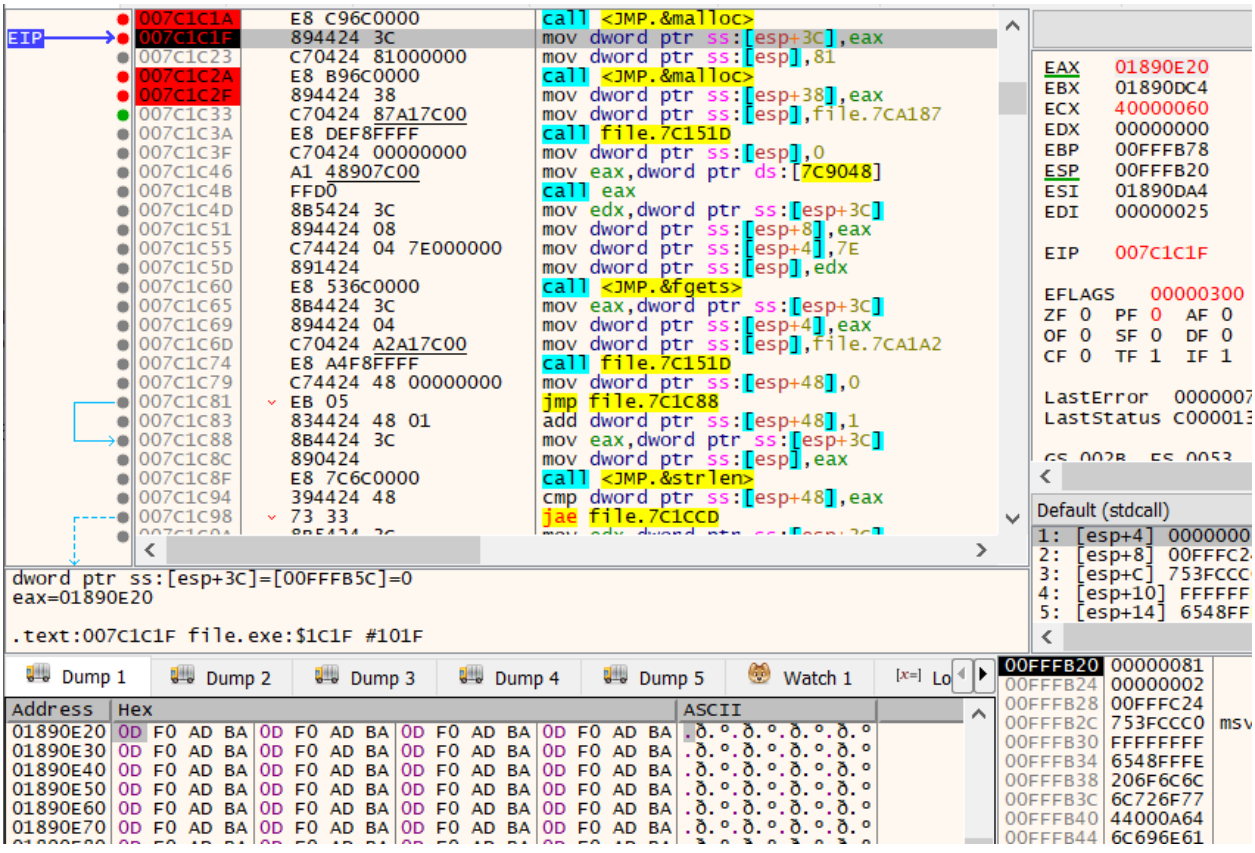

Рисунок 3 – Memory dump

2.2 Обработка команды в fgets

В консоль введём слово open для открытия файла. На рисунке 4 видно, что введённое значение было записано в ячейку памяти.

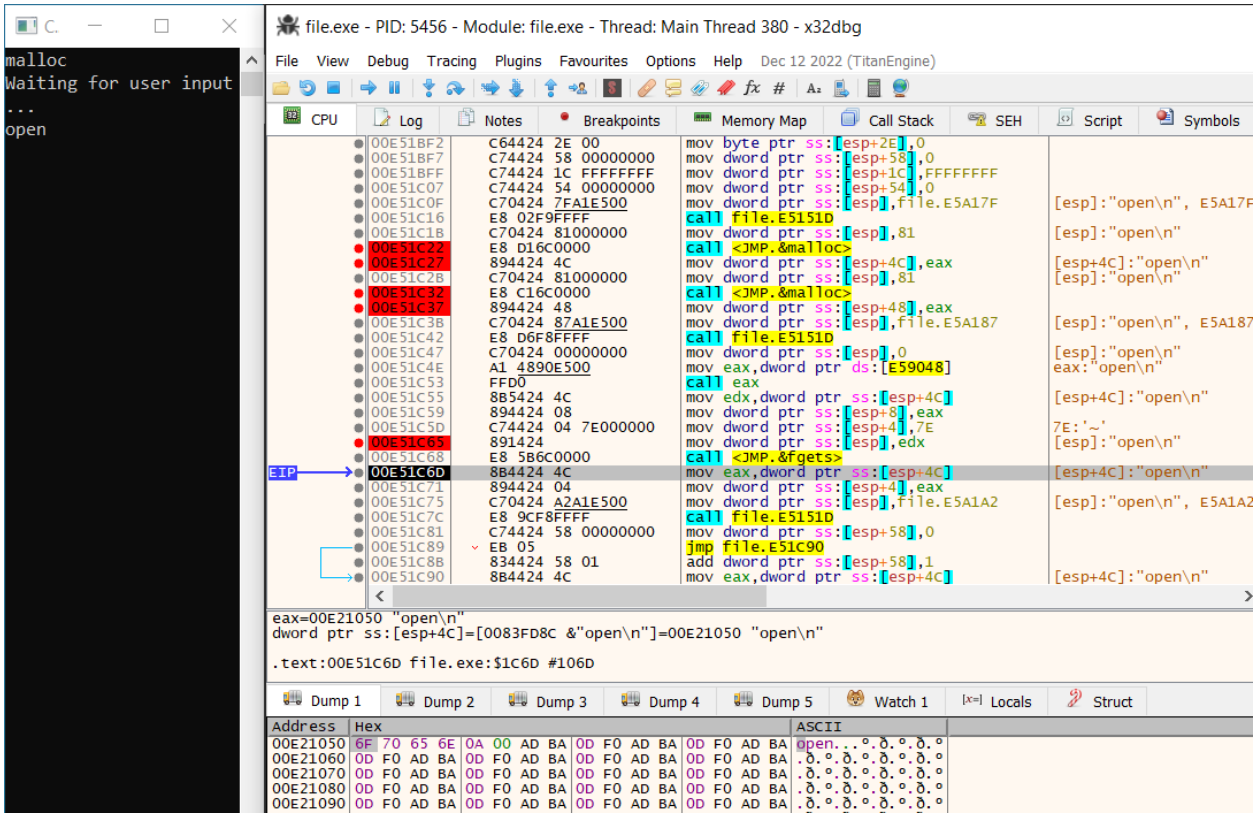

Рисунок 4 – Содержимое памяти

2.3 Вызов API-функции CreateFile

Далее вызывается функция CreateFileA. На рисунке 5 приведён нужный фрагмент библиотеки kernel32.dll.

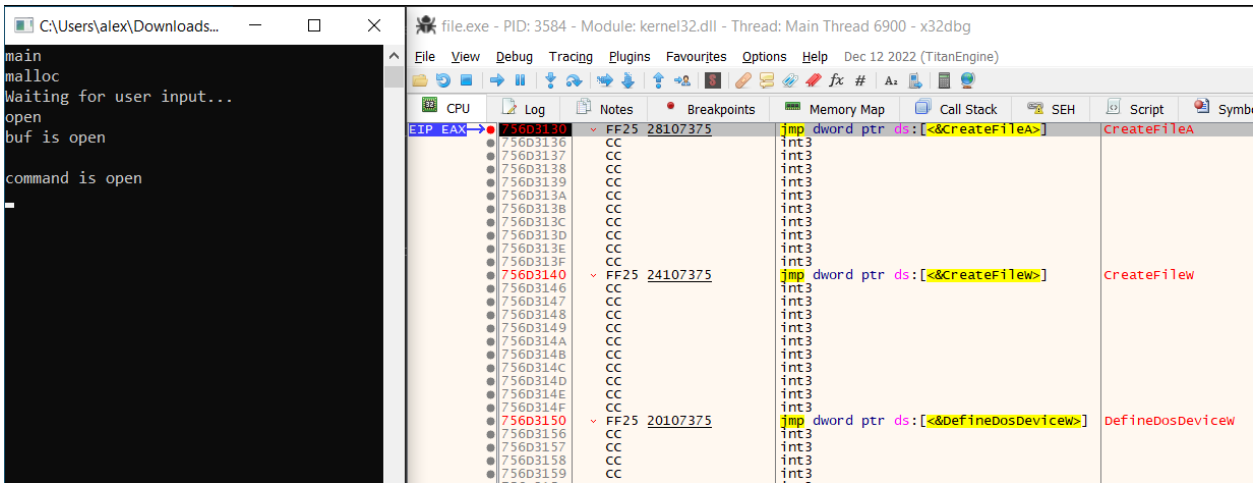

Рисунок 5 – Остановка на функции CreateFileA

На рисунке 6 видно, что в стеке сохранены аргументы функции, например,

название файла (AlekseyShadrunov.txt).

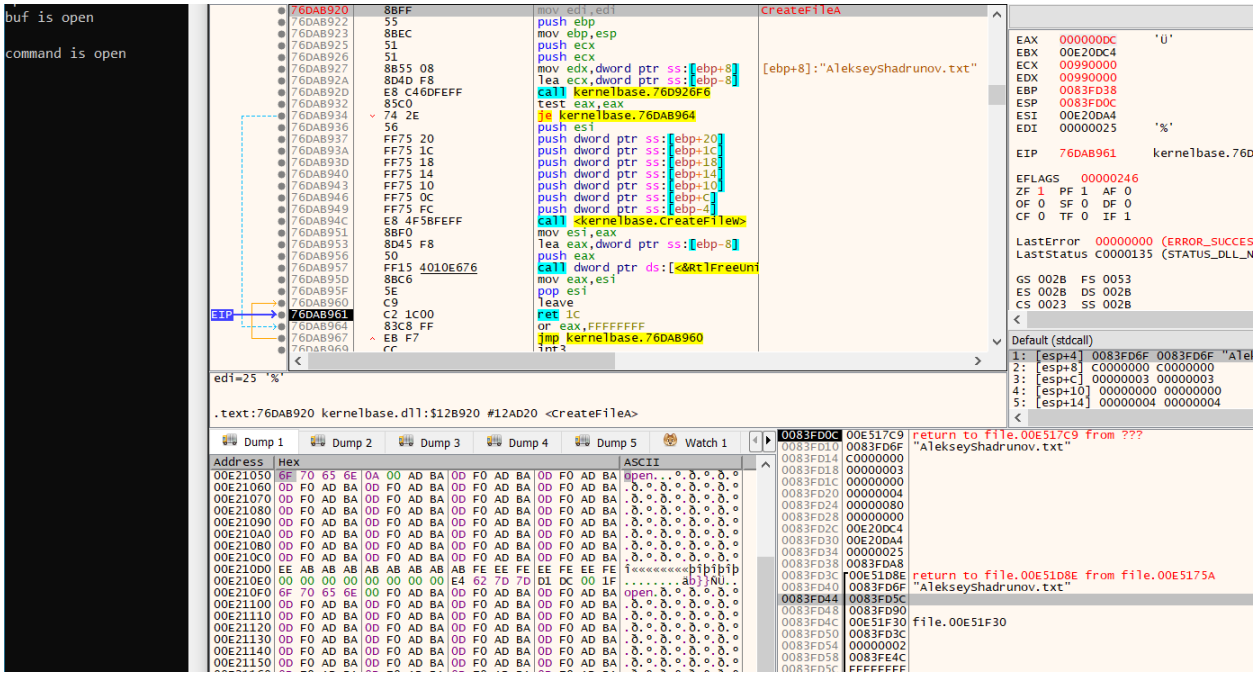

Рисунок 6 – Стек

После вызова API-функции CreateFileA можно увидеть, что в EAX записано значение 0xE0. Это файловый дескриптор или хэндл открытого файла (Рисунок 7).

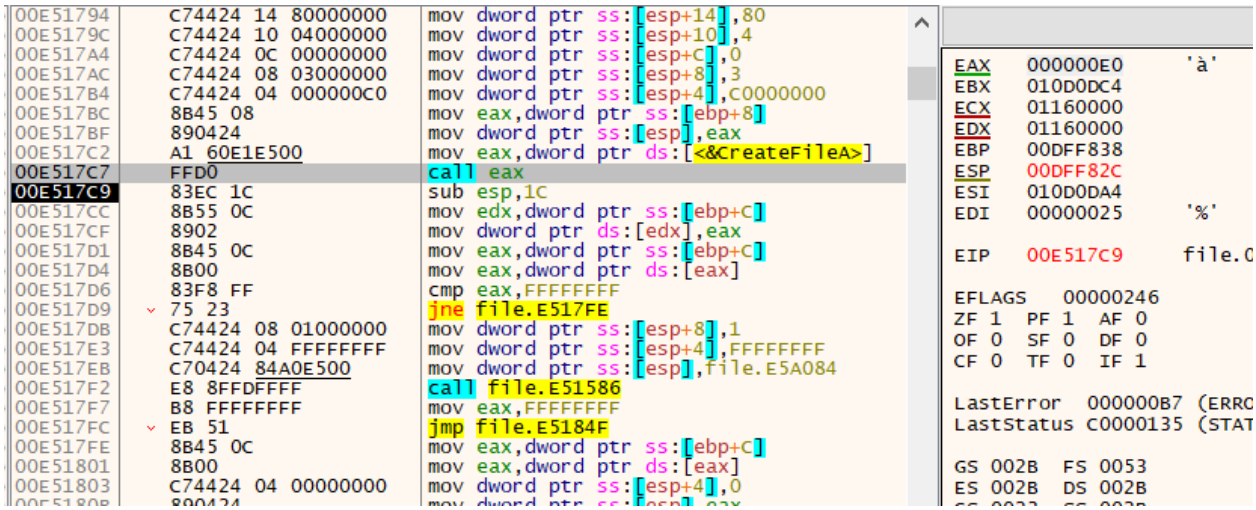

Рисунок 7 – Хэндл открытого файла

Этот же хэндл можно пронаблюдать в Process Hacker (Рисунок 8).

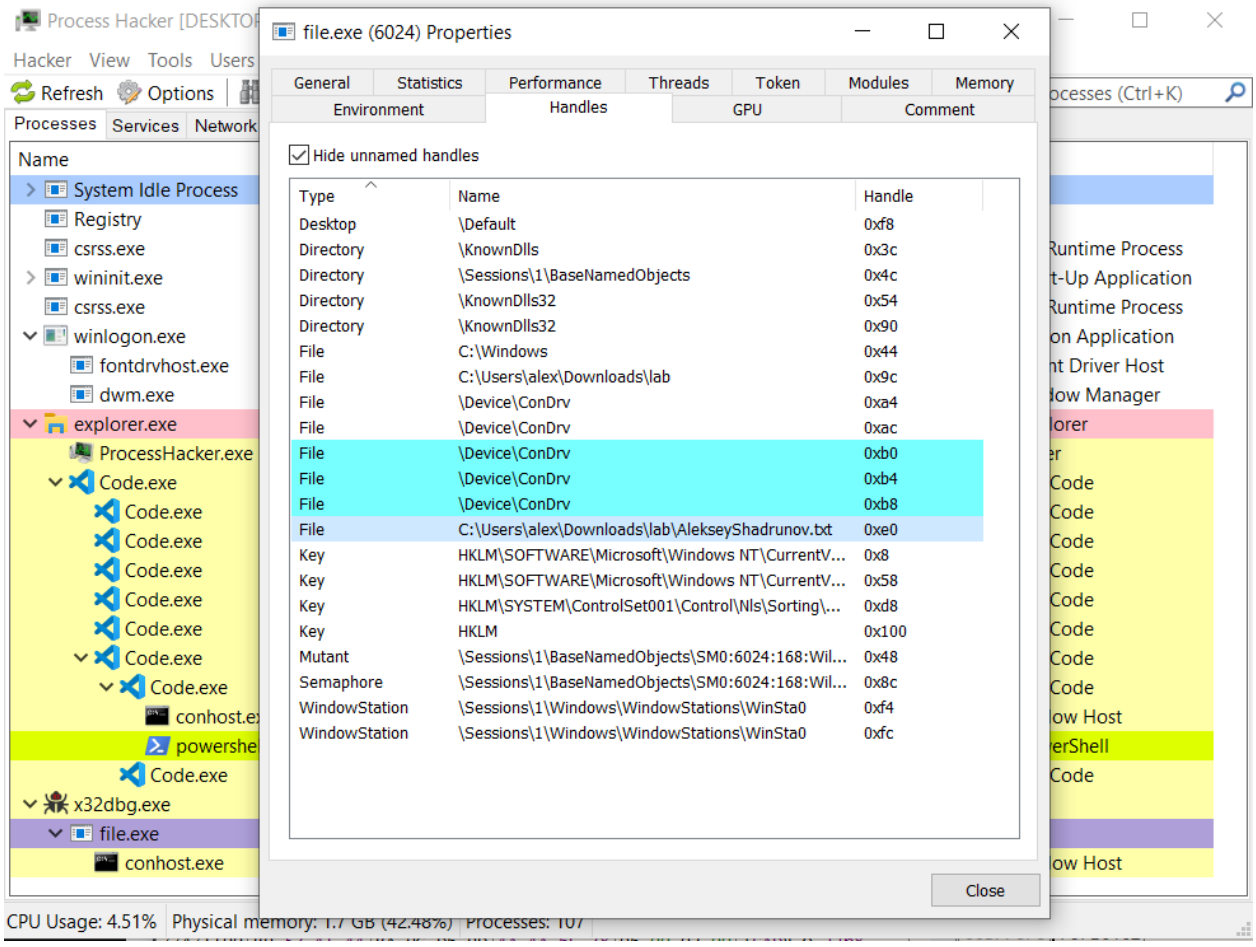

Рисунок 8 – Хэндл в Process Hacker

2.4 Вызов API-функции WriteFile

Вводим в консоль команду write для записи строки в файл. На рисунке 9 показано состояние системы перед вызовом функции WriteFile. Все аргументы записаны в стек: хэндл файла (104), буфер со строкой, количество байт для записи (0xC, то есть 13) и ещё два аргумента.

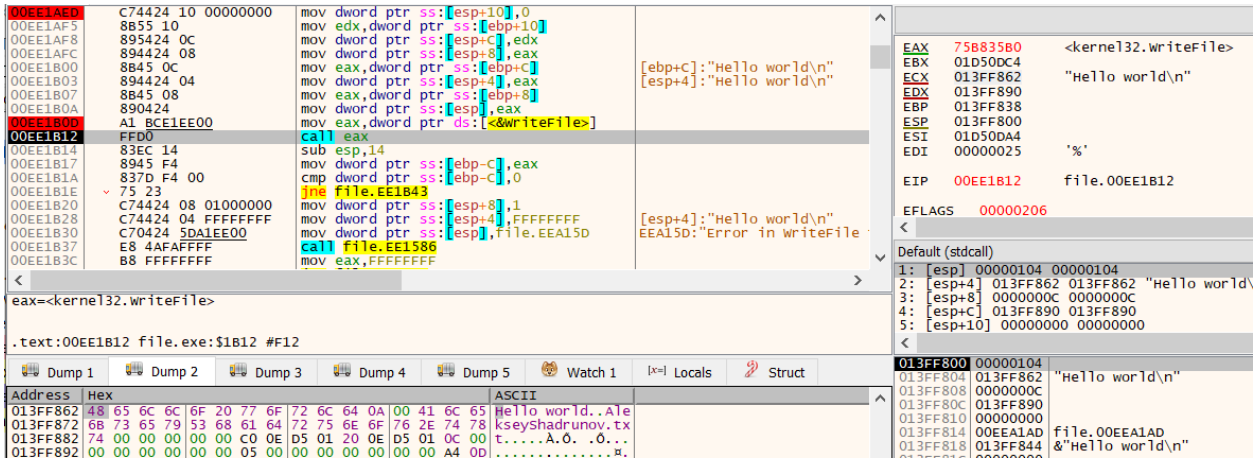

Рисунок 9 – Аргументы в стеке

На рисунке 10 видно тело функции WriteFile, а также строка для записи, которая находится в дампе памяти. На рисунке 11 — Возвращаемое значение 1 (в регистре EAX), что соответствует успешному выполнению функции.

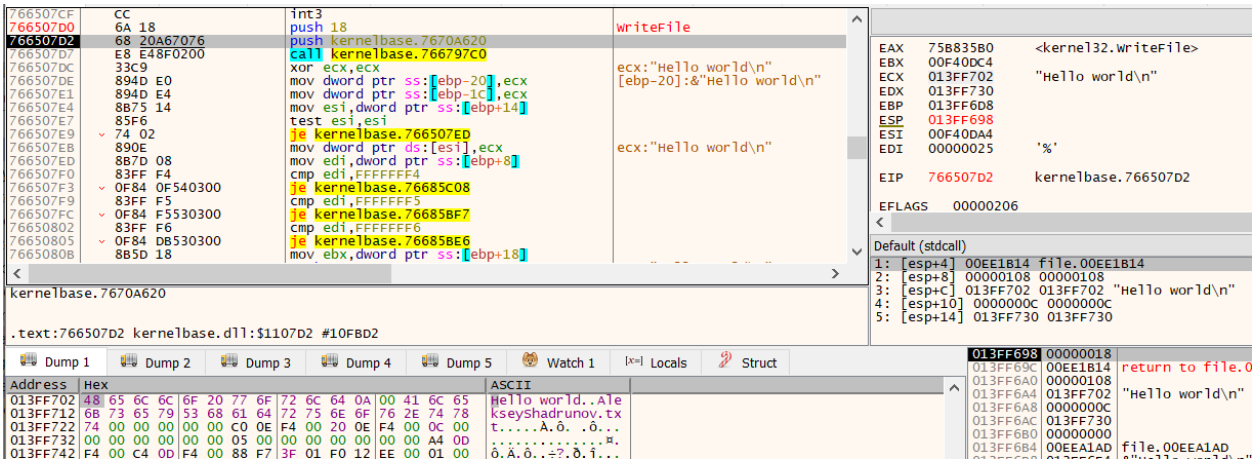

Рисунок 10 – Дамп памяти со строкой

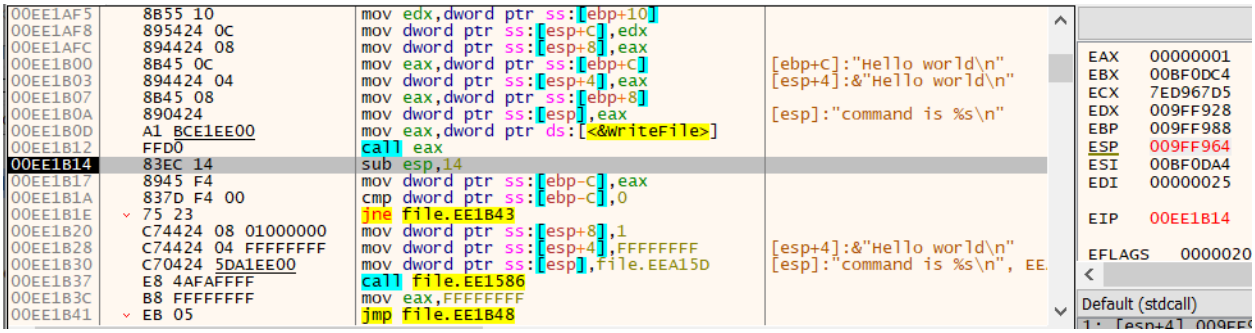

Рисунок 11 – Возвращаемое значение 1

2.5 Вызов API-функции ReadFile

Далее считываем из файла с помощью функции ReadFile. Аргументы функции: хэндл (0x100), адрес выходного буфера (0xDC0F60), количество байт для чтения (0xC) и ещё два опциональных аргумента (Рисунок 12). Аргументы записываются в стек перед вызовом функции.

После вызова функции по адресу выходного буфера содержится строка из файла (Рисунок 13)

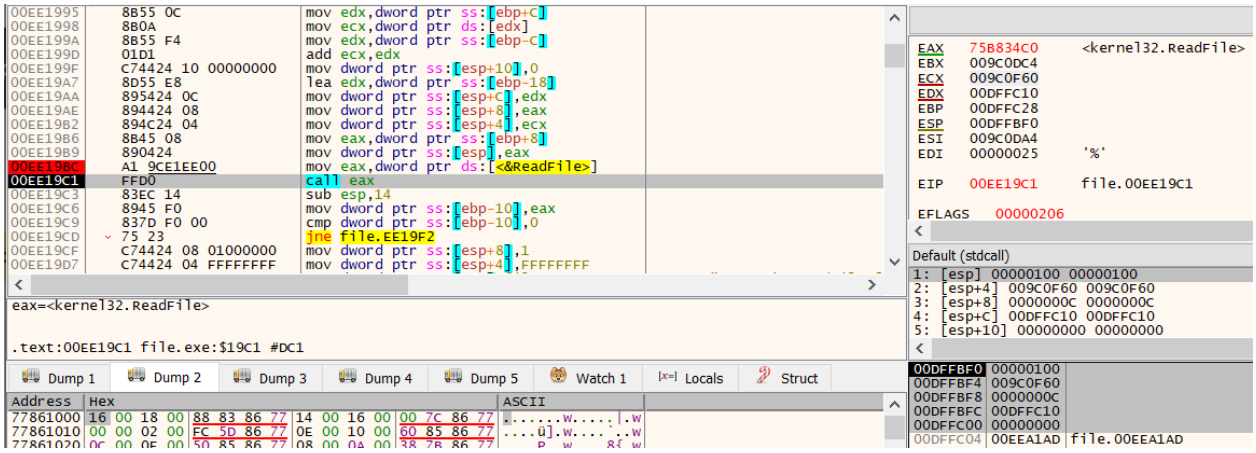

Рисунок 12 – Аргументы в стеке

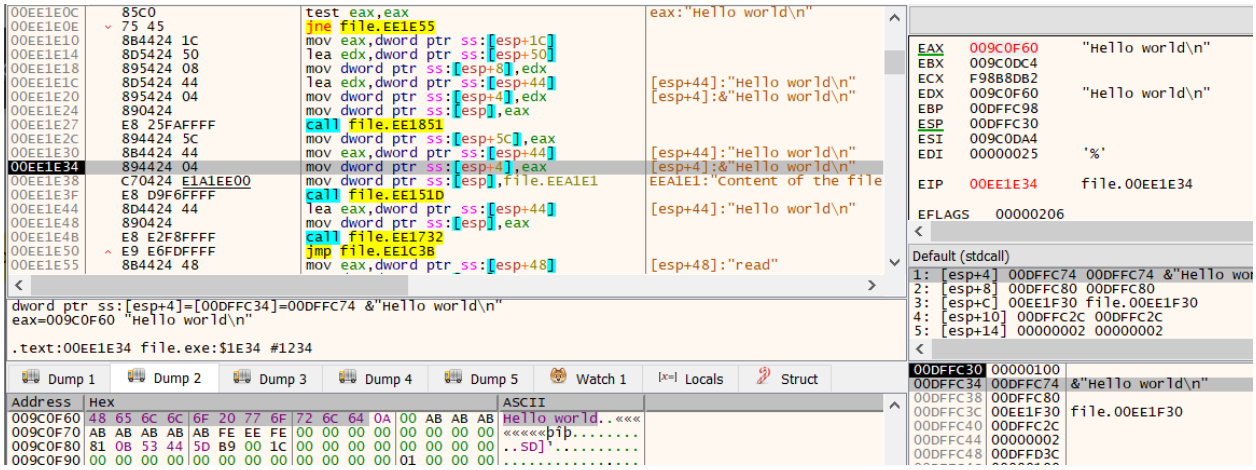

Рисунок 13 – Строка в памяти

2.6 Найти утечку ресурсов в программе

3 Выводы о проделанной работе

В ходе работы я реализовал программу, подсчитывающую сумму остатков от деления на 3 элементов массива длиной 10. Программа написана на ассемблере. В программе используется синтаксис Intel, а также реализован вывод в консоль без подключения библиотек.

Приложение А