# цель работы

1. Изучить основные типы ошибок в программном обеспечении (целочисленное переполнение, отсутствие проверки длины копируемых данных, переполнение буфера и другие).
2. Получить у преподавателя файлы в соответствии со своим вариантом.
3. Реализовать программу, осуществляющую фаззинг формата файла.
4. Реализованная программа должна осуществлять следующие действия:

* осуществлять изменение оригинального файла (однобайтовая замена, замена нескольких байт, дозапись в файл);
* заменять байты на граничные значения (0x00, 0xFF, 0xFFFF, 0xFFFFFF, 0xFFFFFFFF, 0xFFFF/2, 0xFFFF/2+1, 0xFFFF/2-1 и т.д.);
* иметь автоматический режим работы, при котором производится последовательная замена байт в файле;
* находить в файле символы, разделяющие поля (“,:=;”);
* расширять значения полей в файле (дописывать в конец, увеличивать длину строк в файле);
* находить границы полей в файле на основании анализа нескольких конфигурационных файлов;
* осуществлять запуск исследуемой программы;
* обнаруживать возникновение ошибки в исследуемом приложении;
* получать код ошибки и состояния стека, регистров и другую информацию на момент возникновения ошибки;
* логировать в файл информацию о произошедших ошибках и соответствующих им входных параметрах (произведенные замены).

1. Разработать IDC/IDAPython-скрипт, осуществляющий следующие действия:
   * поиск в программе функций ввода данных (fread, fscanf, read, fgets, …);
   * поиск вызовов небезопасных функций (strcpy, sprintf, strncpy, memcpy, memmove, …).
2. Разработанный IDC/IDAPython-скрипт должен выводить следующую информацию:
   * название найденной функции;
   * адрес, откуда вызывается данная функция.
3. Используя разработанную программу для фаззинга формата файлов осуществить поиск в выданной программе уязвимости (целочисленное переполнение, отсутствие проверки длины данных), приводящей к переполнению буфера.
4. Сформировать файл, приводящий к краху исследуемого приложения.
5. Изучить с использованием дизассемблера и отладчика структуру программы и найти участок кода с уязвимостью. Провести исследование найденной уязвимости и условий ее эксплуатации.

# ход работы

## Фаззер

В ходе выполнения лабораторной работы был разработан фаззер со следующими функциями (рисунок 1):

1. Автофаззинг – Последовательное подставленные байт 0x00 0xFF, 0xFF/2, 0xFF/2+1, 0xFF/2-1, 0xFFFF/2+1, 0xFFFF/2-1 и т.д.
2. Изменение определённых байт, как на случайные значения, так и на вводимые
3. Поиск разделителей полей
4. Дописывание в конце файла байтов

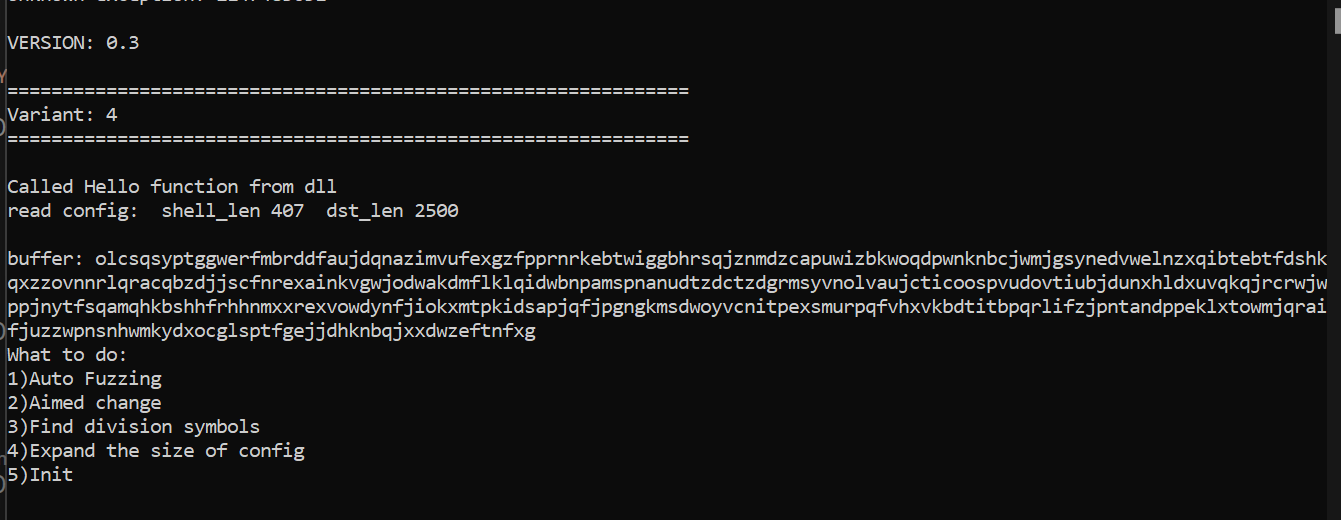


Рисунок 1 – Разработанный фаззер

Фаззер изменяет конфик и запускает программу, в случае получения ошибки в ходе исполнения программы, фаззер сохраняет конфиг файл, а также записывает состояние стека и регистров (рисунок 2)

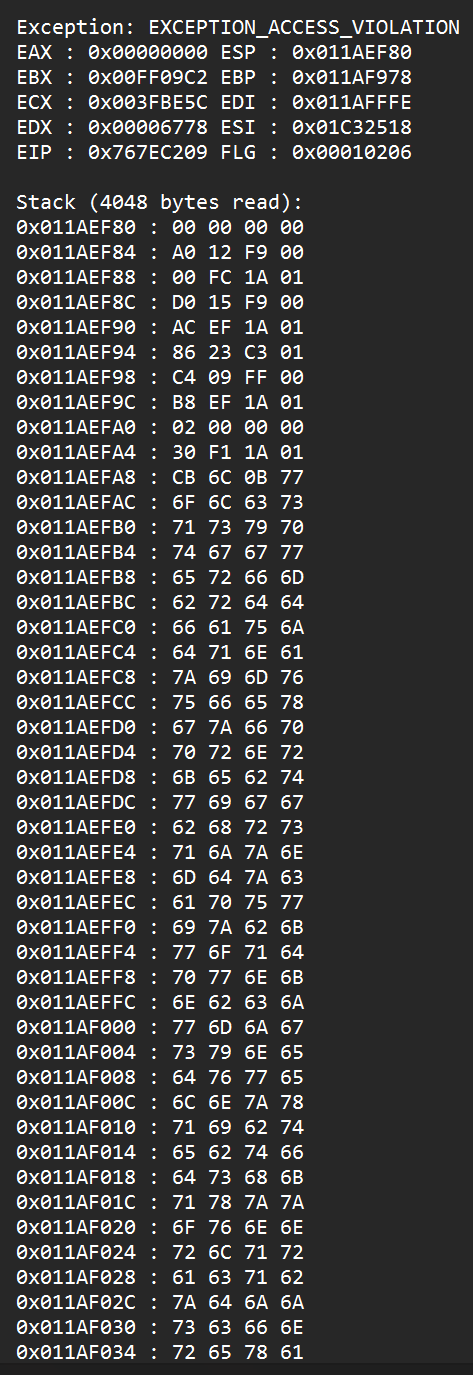


Рисунок 2 – Сохранение состояние стека и регистров

## Поиск уязвимости

В ходе работы фаззера был получен конфиг файл, приводящий к ошибки. При попытки запустить приложения с данным конфиг файлом в дизассемблере IDA Pro было получено следующее исключение (Рисунок 3). Данное исключение означает ошибка записи памяти, а именно инструкция 0x767EC209 пытается обратиться к адресу памяти 0x1400000 и записать в нее переданную информация, но это действие невозможно из-за работающей проверки на целостность памяти.

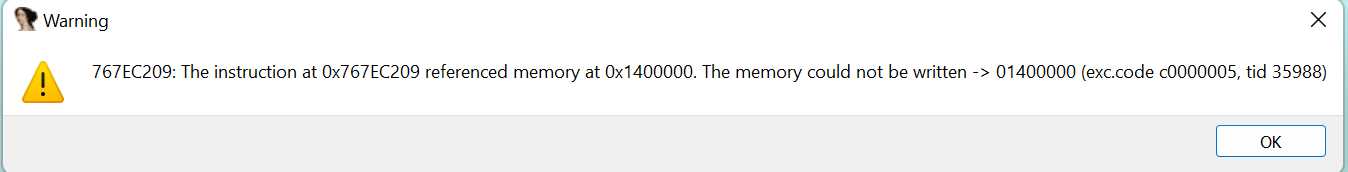


Рисунок 3 – Ошибка, возникающая во время работы приложения

При анализирование дизассемблированного кода была найдена уязвимая функция (Рисунки 4 - 5)

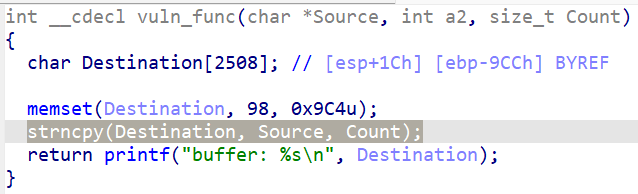


Рисунок 4 – Псевдокод уязвимой функции

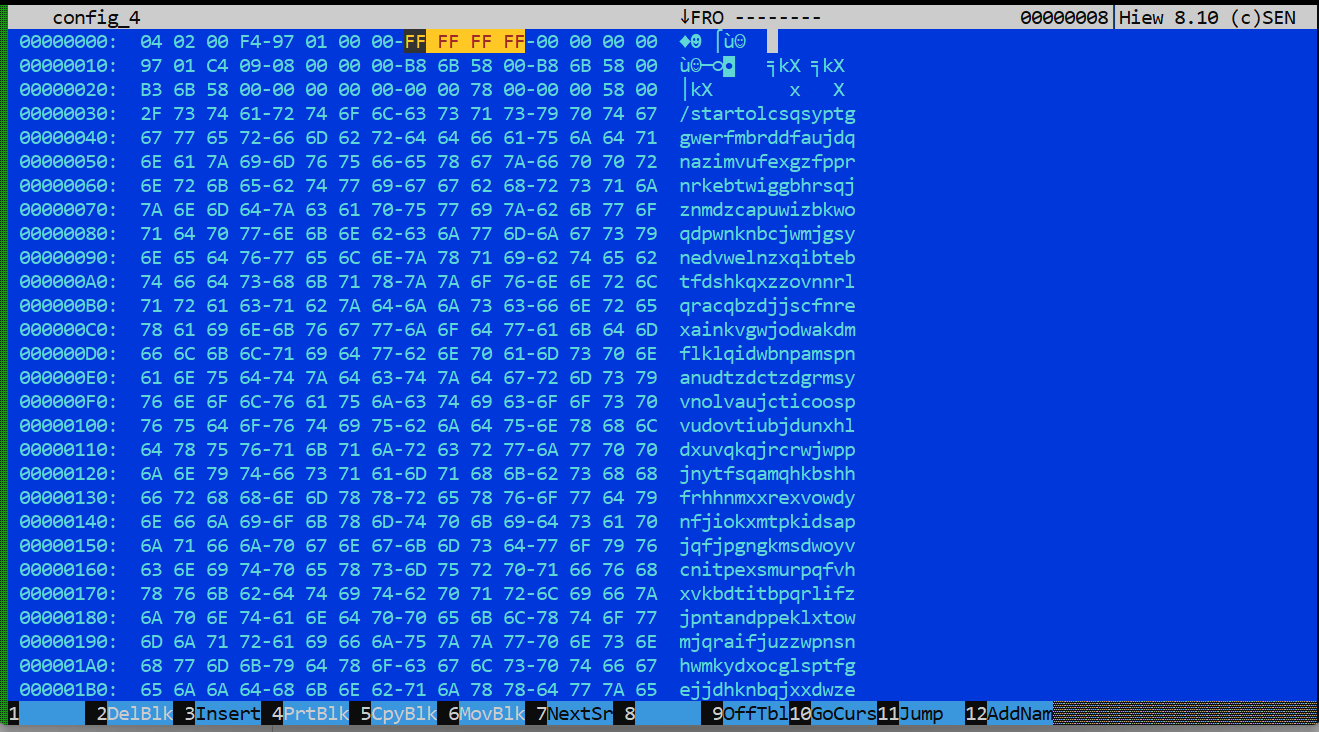


Рисунок 5 – Пример конфиг файла приводящее к ошибке

## Скрипт в IDA Pro

Так же был реализован скрипт для дизассемблера IDA pro. Разработанный скрипт находит адреса небезопасных функций и функций ввода данных (Рисунок 6).

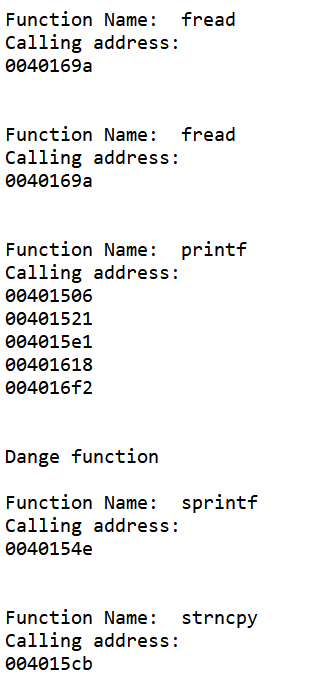


Рисунок 6 – Пример работы скрипта

# Вывод

В ходе выполнения данной лабораторной работы мною были изучены принципы поиска уязвимостей в программном обеспечении без исходных кодов, а также получены навыки работы с IDA Pro.