# 3 ДОСЛІДЖЕННЯ З ЗАСТОСУВАННЯМ СТАТИЧНОГО МЕТОДУ

## 3.1 Опис експерименту та застосованої програмної розробки

В якості досліджуваної програми будемо розглядувати просту програму своєї розробки, яка шифрує та дешифрує рядок введений з клавіатури з використанням алгоритму перестановки по діагоналі справа наліво та використанням операції XOR над символами рядка. Статичний аналіз будемо проводити за допомогою дизасемблеру IDA, та декомпілятору HexRays. В ході експерименту будуть шукатися функції шифрування та дешифрування та проводитися аналіз їх алгоритмів.

## 3.2 Результати дослідження з застосуванням статичного методу

При початку роботи з дизасембльованою програмою, дизасемблер IDA поміщає нас до функції start. Функція start є єдиною експортованою функцією при використанні налаштувань компілятору Micorsoft Visual C++ за замовучванням, вона є точкой входу до програми (рис 3.1).

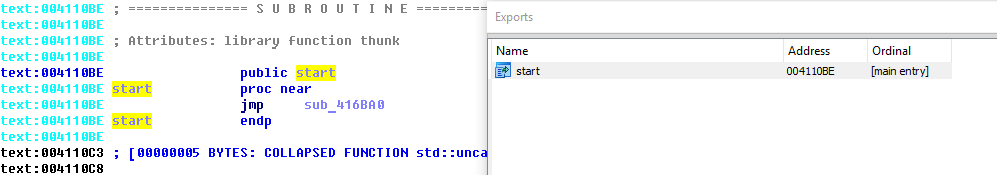


Рисунок 3.1 – Вікно перегляду експортованих функцій

В функції start є лише одна команда, виклик функції sub\_416BA0. Ймовірно, що функція main або виклик до неї знаходиться саме в цій функції. Нажавши двічи на ім’я функції переходимо до її команд. Функція sub\_416BA0 включає до себе виклик функції \_scrt\_common\_main, що є частиною статично імпортованої бібліотеки CRT (С Runtime Library). Оскільки функція sub\_416BA0 більше нічого не викликає, переіменуємо ії на зрозуміле нам ім’я, commonMainCall, за допомогою натисканню правої кнопки миші на ії заголовку (рис.3.2).

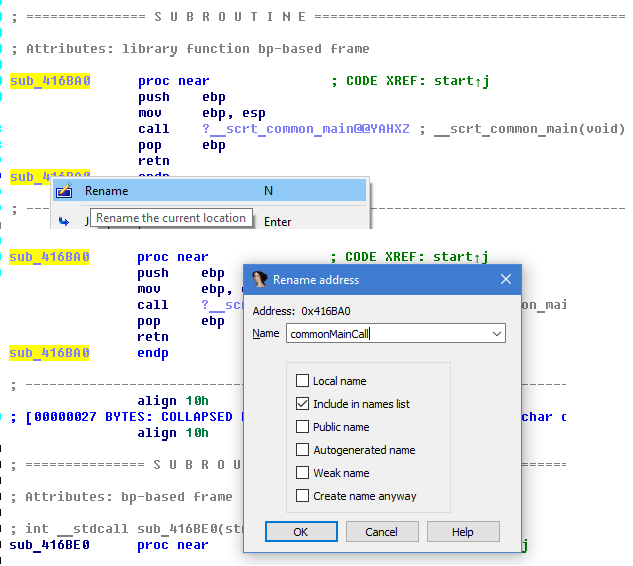


Рисунок 3.2 – Переіменування функції

Наступним кроком перейдемо до функції \_scrt\_common\_main. Ця функція викликає 2 функції (рис 3.3): j\_\_\_\_security\_init\_cookie, та \_\_scrt\_common\_main\_seh. j\_\_\_\_security\_init\_cookie є функцією захисту від перевантаження буферів та є вбудованою функцією CRT. Ця функція не є об’єктом даного аналізу. Функція \_\_scrt\_common\_main\_seh також є вбудованою функцією CRT та ймовірно викликає подальші функції ініціалізації програми, тому перейдемо до неї.

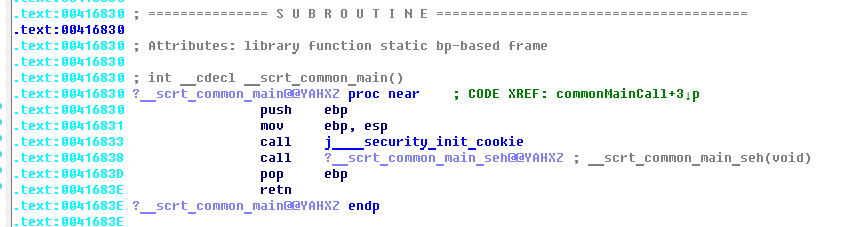


Рисунок 3.3 – Функція \_scrt\_common\_main

Як бачимо, функція \_\_scrt\_common\_main\_seh має багато команд (рис 3.4). Для полегшення аналізу використуємо декомпілятор за допомогою натиснення на клавішу «F5» (рис 3.5).



Рисунок 3.4 – Функція \_\_scrt\_common\_main\_seh



Рисунок 3.5 – Декомпільована функція \_\_scrt\_common\_main\_seh

Виходячи з імен функцій які використовуються, наприклад, j\_\_\_\_scrt\_initialize\_crt, зробимо припущення, що у функції \_\_scrt\_common\_main\_seh проводиться ініціалізація бібліотеки CRT. Продовжимо пошук функції main. На рядку 35 знаходимо виклик функції invoke\_main, судячи з ім’я робимо припущення, що саме там викликається головна функція програми. При переході на цю функцію бачимо (рис 3.6), що в ній проходить сбір аргументів командої строки (argv, argc) та виклик функції \_main. Головна функція аналізуємої програми знайдена.

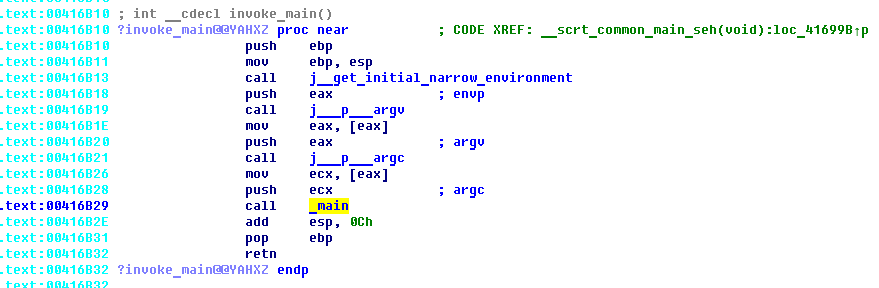


Рисунок 3.6 – Функція invoke\_main

Також головну функцію можна знайти набагато легше, використовуючи перехресні посилання. Знаючи рядки, які програма виводить на екран, можна швидко знайти де вони використовуються. Відкриємо вкладку «Strings» у IDA натиснувши на комбінацію клавіш Shift+F12 (рис 3.7).

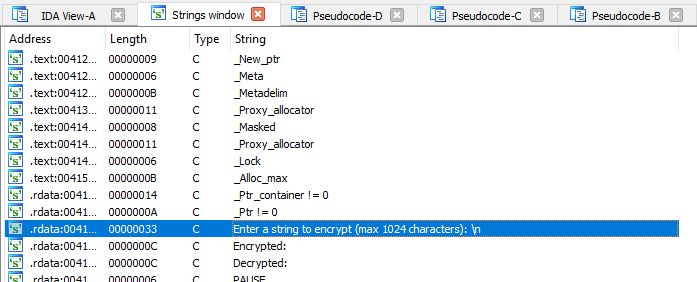


Рисунок 3.7 – Список використовуємих рядків

Як бачимо, у списку присутній рядок який програма виводе на екран. Натиснувши двічі на цей рядок ми переходимо до його розташування у даних програми. Звідти, натиснемо двічі на запис указаний як «DATA XREF» (рис 3.8), це є перехресне посилання, воно приведе нас до функції де використовується ций рядок.

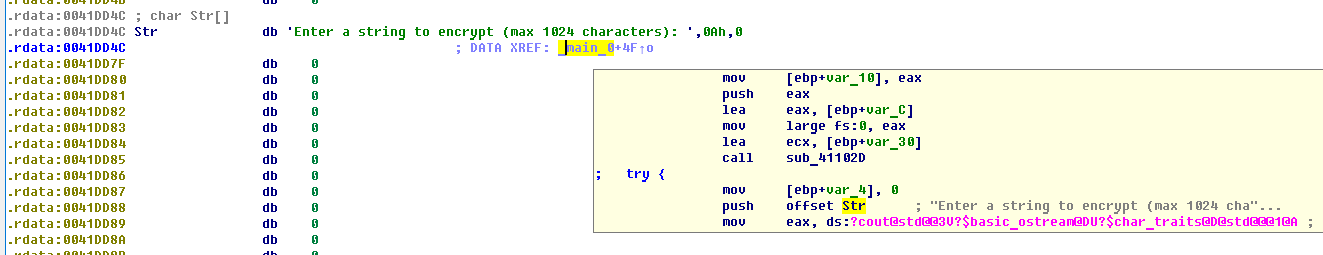


Рисунок 3.8 – Перехресне посилання

Для простоти аналізу, декомпілюємо головну функцію програми main (рис 3.9). Подивившись на використання функції виводу тексту на екран, можна зразу зробити висновки, щодо функцій шифрування та дешифрування.

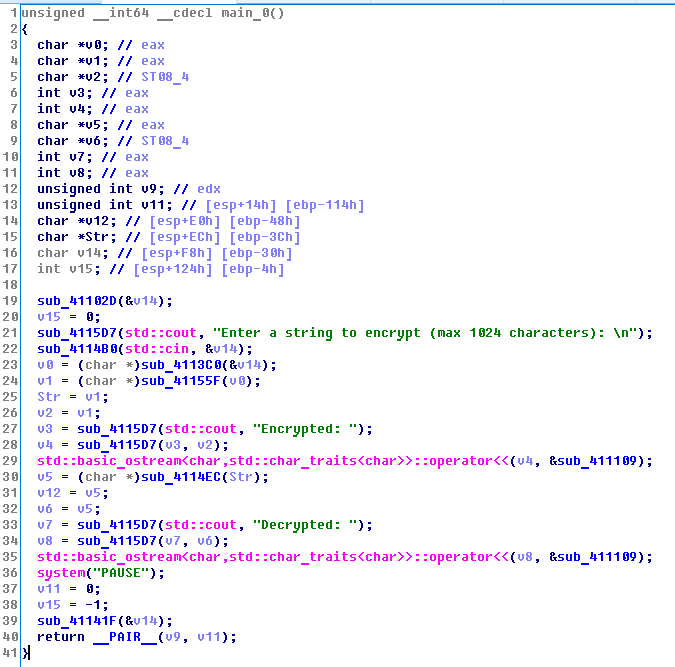


Рисунок 3.9 – Декомпільована функція main

Дивлячись на функцію, що використовує рядок, який пропонує ввести дані для шифрування, можна зробити висновок, що дані з клавіатури заносяться до змінної v14. Далі, змінна v14 викростовується у функції sub\_4113C0 яка повертає значення, яке конвертується до вказівника типу char та заноситься до змінної v0. Припустимо, що функція sub\_4113C0 конвертує введені користувачем дані до звичайного рядка C. Тоді змінна v0 є С-рядком, що користувач ввів з клавіатури. Змінна v0 використовується тільки у функції sub\_41155F. Перейдемо до неї для аналізу (рис 3.10).

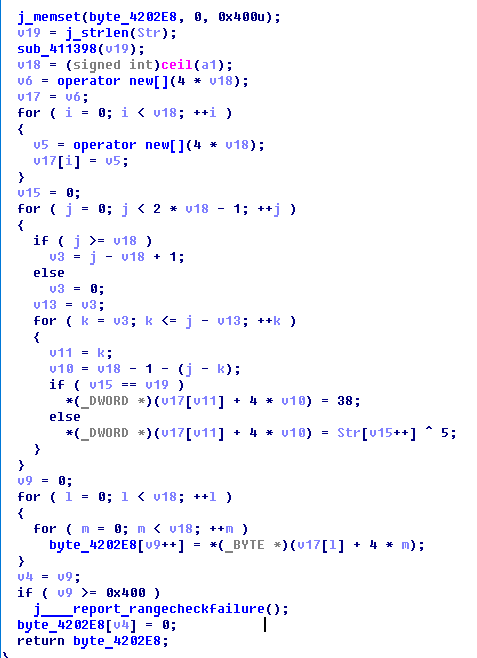


Рисунок 3.10 – Декомпільована функція sub\_41155F

Функція шифрації знайдена. Функція sub\_411398 є обгорткою для функції sqrt. За допомогою функції j\_strlen отримуємо довжину вхідного рядка та заносимо ії до змінної v19. Далі обчислюємо корінь та округлюємо отримане значення за допомогою функції ceil, та заносимо результат до змінної v18, це потрібно для побудови квадратної матриці перестановок. Далі ініціюється динамічний двумірний масив у змінній v6 (v17) за допомогою операторів new та циклу for. Далі вхідний рядок перетворюється по алгоритму перестановки зправа наліво та до кожного символу додається 5 за модулем 2 (XOR). Якщо вхідний рядок менш ніж матриця перестановок, пусті клітини заповнюються символом з кодом 38, тобто «#». Далі перетворена матриця заноситься до статичної змінної byte\_4202E8, та повертаеться з функції.

Виходячи з використання результату функції шифрування, який заноситься до змінної Str у main, можна зробити висновок, що алгоритм дешифрації знаходиться у функції sub\_4114EC, перейдемо до неї (рис 3.11).

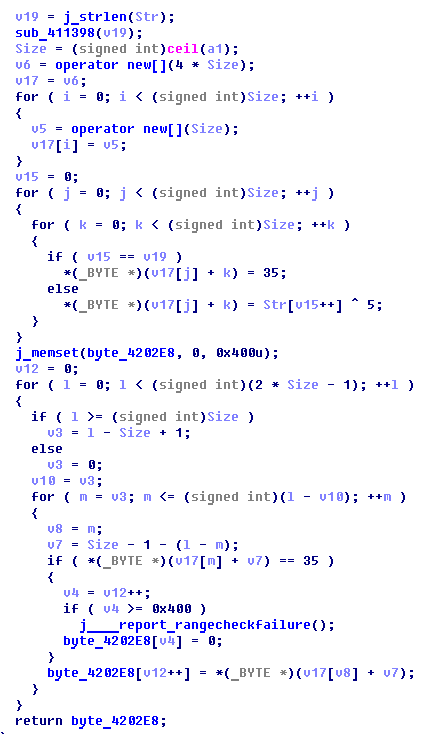


Рисунок 3.11 – Декомпільована функція sub\_4114EC

У цій функції створюється матриця перетворень за раніше описаним алгоритмом та використовуються цикли for, для обходу матриці та дешифрування шифрованої вхідного рядка, який знаходиться у змінній Str. Також присутне додавання 5 по модулю 2, для дешифрування XOR. Бачимо, що всі символи з кодом 35, тобто «#», замінюються на код 0, тобто код кінця рядка. Результат дешифрування заноситься до статичної змінної byte\_4202E8 та повертається до функції main.

Порівняємо результати аналізу з оригінальним вихідним кодом (рис 3.12). Після найменування усіх змінних та функцій, перевіремо чи відповідають усі блоки алгоритму вихідному коду. Як бачимо, декомпілятор повністю відновив алгоритм шифрування та по ньому можна швидко аналізувати його роботу.

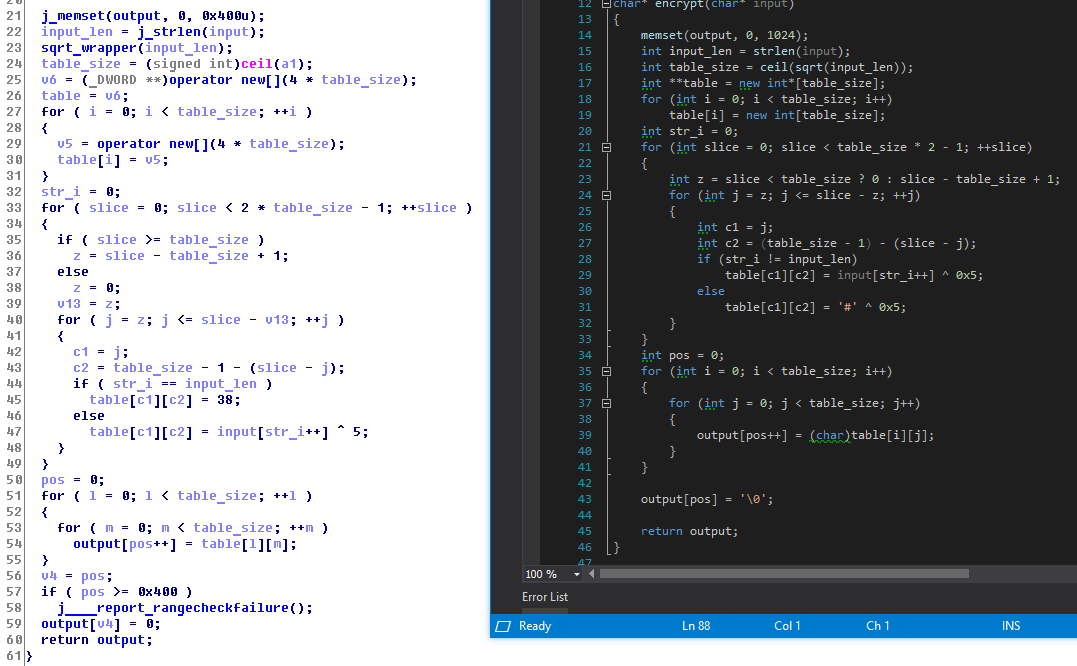


Рисунок 3.12 – Порівняння функцій шифрування

Алгоритм дешифрування також був повністю відновлений (рис 3.13) та може бути використаний для повного аналізу.

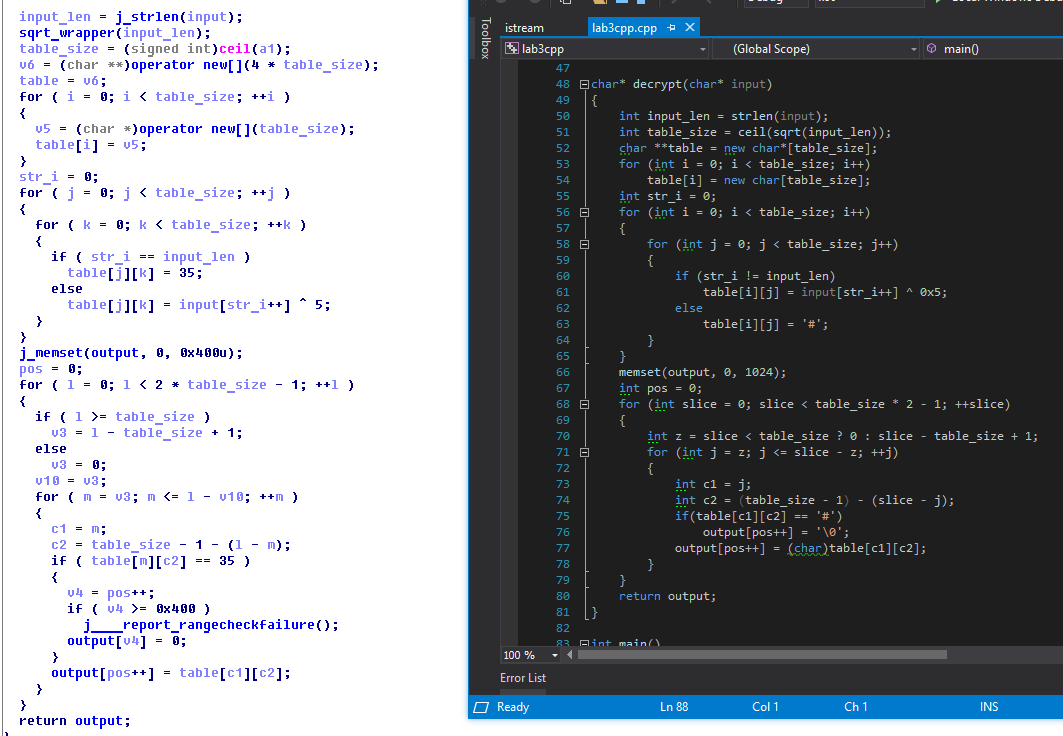


Рисунок 3.13 – Порівняння функцій дешифрування