# 3. СТАТИЧНИЙ МЕТОД

# 3.1 Програма - дизасемблер: IDA

IDA Pro Disassembler (англ. Interactive DisAssembler) — інтерактивний дизасемблер, який широко використовується для реверс-інжинірингу. Він відрізняється винятковою гнучкістю, наявністю вбудованої командної мови, підтримує безліч форматів файлів для великого числа процесорів і операційних систем.

Дозволяє будувати блок-схеми, змінювати назви програмних позначок, переглядати локальні процедури в стеку і багато іншого.

В останніх версіях має вбудований налагоджувач (рівнозначні терміни – зневаджувач, деба́ґґер) x86 і ARM.

IDA, до певної міри, вміє автоматично виконувати аналіз коду, використовуючи перехресні посилання, знання параметрів викликів функцій бібліотек, та іншу інформацію. Проте вся його сила проявляється в інтерактивній взаємодії з користувачем. На початку дослідження дизасемблер виконує автоматичний аналіз програми, а потім користувач за допомогою інтерактивних засобів IDA починає давати осмислені імена, коментувати, створювати складні структури даних та іншим чином додавати інформацію в лістинг, згенерований дизасемблером поки не стане ясно, що саме та як робить досліджувана програма.

Дизасемблер має консольну і графічну версії. Підтримує велику кількість форматів файлів. Однією з відмінних особливостей IDA Pro є можливість дизасемблювання байт-коду віртуальних машин Java і .NET. Також підтримує макроси, плагіни і скрипти, а останні версії містять інтегрований налагоджувач.

# 3.2 Дослідження після дизасемблювання

Для того, щоб розбирати файл з допомогою дизасемблеру IDA, слід відкрити файл через пункт меню File / Open, вибрати параметри дизасемблювання (зазвичай можна залишити параметри, задані за замовчуванням) і почекати кілька хвилин, поки не завершиться дизасемблювання. Отриманий лістинг можна почати аналізувати до повного завершення дизасемблювання, але цей лістинг буде менш інформативний.

При практичній реалізації алгоритмів дизасемблювання виникають наступні проблеми.

1. Проблема відновлення символічних імен. У програмі, написаною на мові асемблера, всі змінні, мітки, процедури і сегменти мають символічні імена. При компіляції програм ми ці імена замінюються фізичними адресами. У скомпільованому машинному коді не залишається інформації про символічні імена, якщо, звичайно, вона спеціально не поміщена туди для налагоджування, або для взаємодії з іншими програмами (наприклад, щоб динамічно довантажувати бібліотеки Windows, які містять таблицю імен функцій, що експортуються та імпортуються програмами). Зазвичай ця проблема вирішується шляхом «вигадування» дизасемблером символічних імен типу var1, label2, ргос3 і т. д.

2. Проблема розрізнення команд і даних. У скомпільованому програмою машинному коді дані відрізняються один від одного тільки з контексту використання. Наприклад, машинна команда може безпосередньо слідувати за глобальною змінною. Якщо дизасемблер приймає код за дані або навпаки, текст, що видає дизасемблер, стає безглуздим.

3. Проблема визначення кордону машинної команди. Команди машинного коду, що призначені для виконання на СІЅС-процесорах (процесори x86 і x86-64 відносяться до СІЅС-процесорів), слідують один за одним поспіль безпосередньо, без яких-небудь роздільників. При виконанні коду процесор починає зчитувати чергову команду з байту, який безпосередньо слідує за тільки що виконаної командою. Якщо дізассемблер не правильно визначає кордон команди, він неправильно відновлює цю команду і кілька команд, наступних за ней.

Функція Start Рисунок 3.1

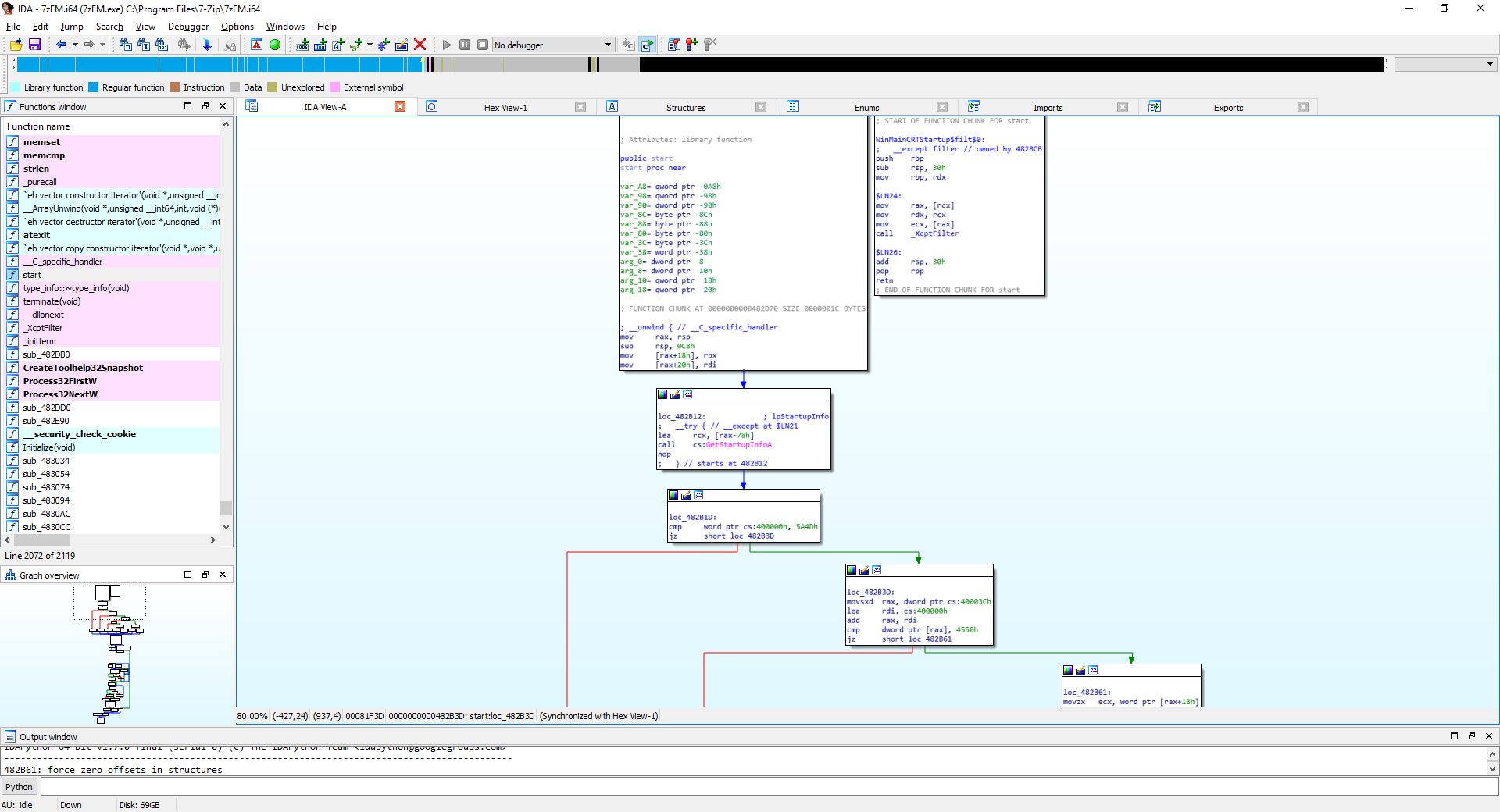
.

Рисунок 3.1 Функція старт

Перехресні посилання — адреси, із яких здійснюються звернення до поточного адресу. Подвійний клацання по такому коментарю викликає перехід на адресу перехресного посилання. За замовчуванням для кожної адреси відображається не більше двох перехресних посилань, це число можна збільшити у вікні параметрів дизасемблювання, що показано на рисунку 9.

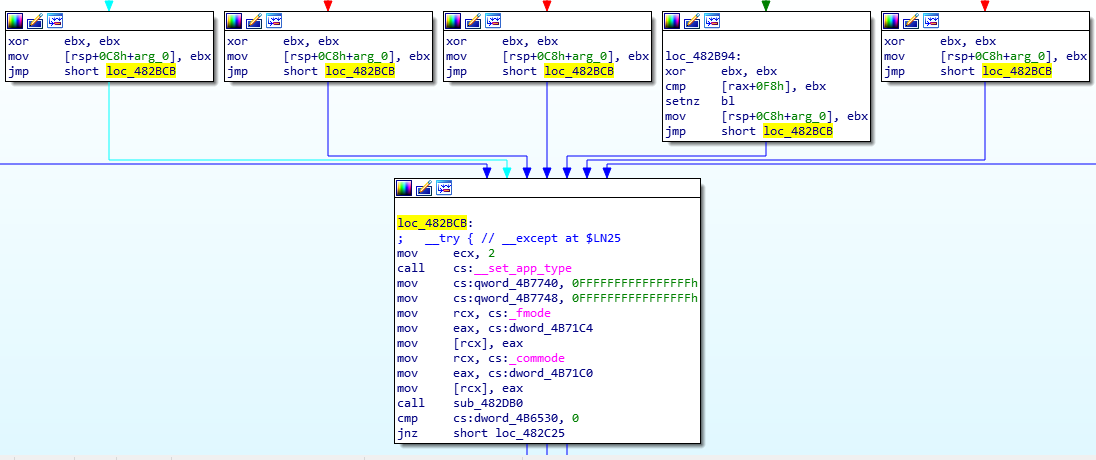


Рисунок 3.2 перехресні посилання

Під час пошуку констант, я буду шукати константи з строковими змінними. Найбільш часто константи знаходяться в сегменті даних, який іменується .date. В старих програмах для MS-DOS ці правила не працюють. Для переходу в сегмент даних нам потрібно, проскролити до низу, і зразу побачимо константи заповнені строковими даними. Рисунок 3.3 Початкова константа зображення 3.4.

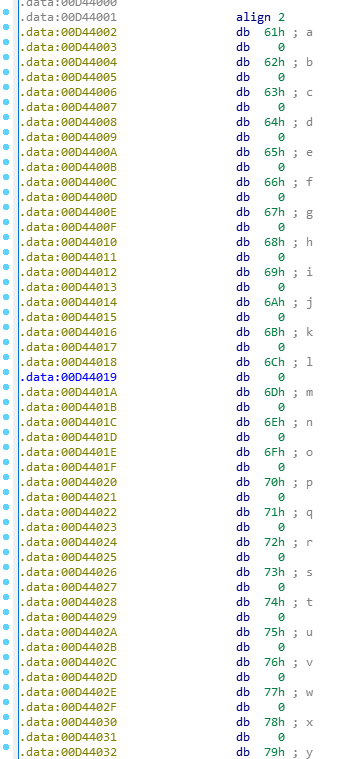


Рисунок 3.3 – Константа з строковими даними



Рисунок 3.4 Початкова константа

Після декомпіляції програми знайдено функцію, що не відповідає початковому стану цієї функції. Під час вивчення декомпільованої функції видно, що тип даних функції декомпільовано невірно, тобто замість типу void стоїть int64. Також був змінений вид логіки функції, тобто цикл у циклу був замінений двома різними циклами. Замість циклу for використовується цикл while в якому була змінена умова замість (i < k – move). Також Замість while використовувався цикл for зі зміненою умовою та шагом. Початковий модуль - рисунок 3.4. Декомпільований модуль - рисунок 3.5.

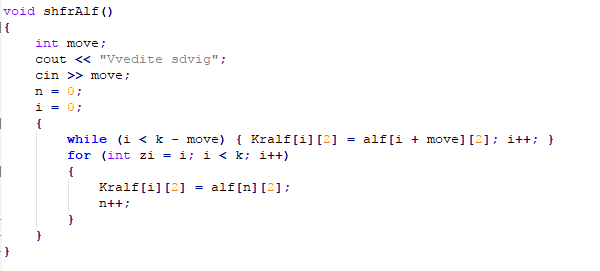


Рисунок 3.4 Початковий модуль

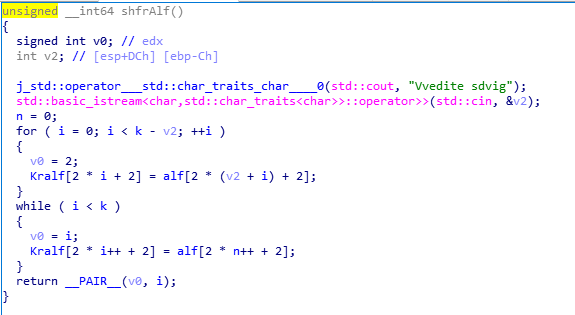


Рисунок 3.5 Декомпільований модуль

Після декомпіляції програми знайдено функцію, що відповідає її початковому стану. Все, крім типу даних функції, було декомпільовано вірно. Функцію початкову і функцію декомпільовану можна побачити на зображенні 3.6 та 3.7 відповідно

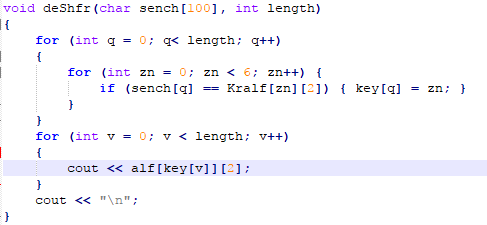


Рисунок 3.6 Початкова функція

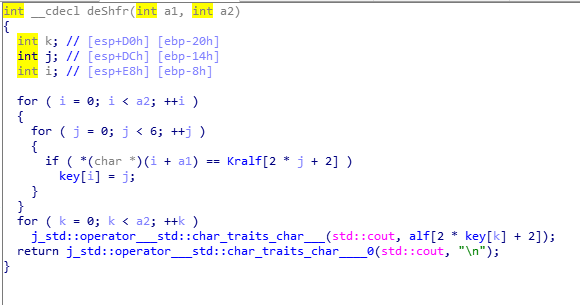


Рисунок 3.7 Декомпільована функція