

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДІВ
ЗАХИСТУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ВІД РЕФАКТОРИНГУ
RESEARCH AND SOFTWARE IMPLEMENTATION OF
METHODS FOR PROTECTING SOFTWARE FROM REFACTORING

Виконав(ла): студент(ка) 2 курсу, групи КНТ-223м
Спеціальності 122 Комп'ютерні науки
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Системи штучного інтелекту

Г АНОШЕНКО П.О.

(ПРИЗВИЩЕ та ініціали)

Керівник ЗАЙКО Т.А.

(ПРИЗВИЩЕ та ініціали)

Рецензент СКРУПСЬКИЙ С.Ю.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 122 Комп'ютерні науки
(код і найменування)

Освітня програма (спеціалізація) Системи штучного інтелекту
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2024 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ГАНОШЕНКА Пилипа Олександровича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація методів захисту програмного забезпечення від рефакторингу. Research and Software Implementation of Methods for Protecting Software from Refactoring

керівник проєкту (роботи) к.т.н., доцент, ЗАЙКО Тетяна Анатоліївна,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “07” жовтня 2024 року № 398

2. Строк подання студентом проєкту (роботи) 02 грудня 2024 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Матеріали і методи. 3. Опис програми. 4. Експлуатація, тестування та експериментальне дослідження програмного забезпечення.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів) Слайди презентації

(підпис)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
71 с., 1 табл., 32 рис., 3 дод., 13 джерел.

C, LINUX, COMMAND LINE INTERFACE, АЛГОРИТМ,
ОБФУСКАЦІЯ, РЕФАКТОРИНГ, СТАТИЧНИЙ АНАЛІЗ.

Об'єкт дослідження – процес розробки програмного забезпечення для захисту інтелектуальної власності у сфері розробки програмного забезпечення.

Предмет дослідження – програмні засоби для захисту програмного забезпечення від рефакторингу.

Мета роботи – дослідження методів захисту програмного забезпечення від рефакторингу та розробка програмного забезпечення для захисту програмного коду від дослідження та аналізу.

Матеріали, методи та технічні засоби: мова програмування C, середовище розробки Gedit, відладчик Valgrind.

Результати. Розроблено програмне забезпечення для захисту програмного забезпечення від статичного аналізу та рефакторингу за допомогою алгоритму обфускації.

Практична цінність роботи полягає у розробці програмного забезпечення, що забезпечує захист програмного коду від дослідження.

Висновки. Виконано проєктування програмного забезпечення для захисту програмного коду від дослідження за допомогою обфускації. Розроблено програмне забезпечення для захисту програмного коду від статичного аналізу та рефакторингу. Здійснено тестування розробленого програмного забезпечення.

Галузь використання – розробка та захист програмного забезпечення.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 71 pages, 1 tables, 32 figures, 3 appendixes, 13 sources.

C, LINUX, COMMAND LINE INTERFACE, ALGORITHM, OBFUSCATION, REFACTORING, STATIC ANALYSIS.

The object of research is the process of developing software to protect intellectual property in the field of software development.

The subject of the research is software tools for protecting software from refactoring.

The purpose of this work is to study methods for protecting software from refactoring and develop software to protect program code from analysis.

Materials, methods and technical tools: C programming language, Gedit development environment, Valgrind debugger.

Results. The software to protect software from static analysis and refactoring using obfuscation has been developed.

The practical value of the work lies in the development of software that provides protection of software from research.

Conclusions. The software for protecting software from research using obfuscation has been designed. The software for protecting software from static analysis and refactoring has been developed. The developed software has been tested.

Scope of use – software development and protection.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок	8
Вступ	9
1 Аналіз предметної області	11
1.1 Основні поняття обфускації	11
1.2 Аналіз програмного забезпечення для захисту від дослідження за допомогою обфускації	16
1.2.1 Stunnix	19
1.2.2 The Tigress C Diversifier/Obfuscator	20
1.2.3 Mangle-It C/C++ Code Obfuscator	21
1.2.4 Порівняльна таблиця існуючих засобів обфускації	22
1.3 Висновки за розділом 1	24
2 Матеріали і методи	26
2.1 Вибір мови програмування	26
2.2 Вибір середовища розробки програмного забезпечення обфускатора	27
2.3 Висновки за розділом 2	27
3 Опис програми	29
3.1 Структура програмного забезпечення для захисту програмного коду від дослідження	29
3.1.1 Структура функції main()	29
3.1.2 Структура функції skipincludes()	32
3.1.3 Структура функції inttodefne()	34
3.1.4 Структура функції defines()	37
3.1.5 Структура функції clearlines()	38
3.1.6 Структура функцій s_cmnt() та m_cmnt()	40
3.2 Функціонування програмного забезпечення для запобігання рефакторінгу	42
3.3 Опис функцій алгоритму обфускації	43
3.4 Висновки за розділом 3	44

4 Експлуатація, тестування та експериментальне дослідження програмного забезпечення	45
4.1 Призначення та умови застосування програми.....	45
4.2 Характеристики програми для захисту програмного коду від дослідження.....	45
4.3 Звернення до програми	46
4.4 Вхідні й вихідні дані.....	46
4.5 Повідомлення	47
4.6 Висновки за розділом 4.....	47
Висновки.....	48
Перелік джерел посилання	50
Додаток А Технічне завдання.....	52
Додаток Б Фрагмент тексту програми.....	56
Додаток В Слайди презентації	64

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

ASCII – American Standard Code for Information Interchange;

CLI – Command Line Interface;

GNU – GNU's Not Unix;

ОС – операційна система;

ПЗ – програмне забезпечення;

ПП – програмний продукт.

ВСТУП

Інтелектуальна власність має широке поширення завдяки Інтернету, нерідкі випадки розкрадання, модифікації і підміни походження власності. Програмне забезпечення, як форма інтелектуальної власності, досить часто представляє інтерес для шахраїв. Шахраї мають на меті розповсюдження чужого програмного коду, шукають помилки у вихідному коді для подальшої їх експлуатації та самозбагачення, тощо. Для запобігання шахрайським схемам, що мають на меті дослідження та розповсюдження чужого програмного коду, існують автоматизовані алгоритми перетворення програмного коду у складний текст для розуміння як людиною так і комп'ютером без втрати функціональності, які називаються обфускацією.

Обфускація представляє собою один з методів захисту програмного забезпечення від реверсивної інженерії.

Реверсивна інженерія це дослідження програмного забезпечення з метою розуміння принципів його роботи.

Існує велика кількість алгоритмів роботи обфускаторів, однак більша частина з них базується на двох алгоритмах: алгоритм Коллберга та алгоритм Chenxi Wang's.

Алгоритми обфускації призначені для захисту програмного вихідного коду від дослідження.

Обфускація вихідного коду вважається успішною:

- якщо ціна процедури деобфускації перевищує номінальну вартість самого програмного продукту;
- якщо час, витрачений на злом коду, більше часу, протягом якого збереження таємниці алгоритму роботи програми або вихідного коду програми має значимість.

Як і кожний алгоритм – обфускація має і свої недоліки: наприклад, проведення обфускації для віртуальних об'єктів і абстрактних методів практично неможливе.

Звісно що обфускація не гарантує неможливість процедури дизасемблювання і зламу вихідного програмного коду. Буває також, що захист обфускатору заважає користувачам програмного продукту проводити тестування та налагодження програмного забезпечення.

Для досягнення поставленої мети у кваліфікаційній роботі магістра необхідно розв'язати такі задачі:

- провести аналіз предметної області – з початку виникнення обфускації та еволюцію алгоритмів;
- провести аналіз програмних засобів захисту програмного коду за допомогою алгоритму обфускації;
- здійснити проєктування програмного забезпечення алгоритму обфускації: визначити структуру та функції алгоритму, що розробляється, вибрати середовище та інструменти розробки;
- розробити алгоритм обфускації, його функції та методи шифрування, перевірити алгоритм на стійкість до статичного аналізу;
- спроектувати системи взаємодії алгоритму з файлами через інтерфейс командної строки;
- виконати тестування програмного забезпечення.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Основні поняття обфускації

Обфускація – це процес, що перетворює текст або програмний код у вигляд, що зберігає його функціональність але ускладнює аналіз та модифікацію при декомпіляції (рефакторинг). Трансформація може виконуватися на рівні алгоритмів, машинного коду та байт-коду та мати на меті як ускладнення аналізу коду, так і оптимізацію його роботи при обмеженому просторі [1].

Вихідна інформація для обфускації:

- вимоги ISO/IEC 27038:2014;
- перелік вимог користувача для обфускації;
- програма статичного аналізу;
- цільовий файл з критичною інформацією.

Вимоги ISO/IEC 27038:2014 є міжнародними стандартами та рекомендаціями щодо обфускації критичної інформації [2]. Вимоги користувача містять довідку щодо критичних частин коду або тексту а також вимоги до складності декомпіляції обфускованого файлу. Програма статичного аналізу застосовується для тестування алгоритму обфускації на предмет складності зашифрованого їм цільового файлу. Цільовий файл повинен мати відповідний формат для роботи програми обфускації та по закінченню роботи алгоритму має отримати обфусковану копію.

Можна виділити наступні групи вимог до складання алгоритму:

- користувальницькі;
- міжнародні стандарти;
- програмні та аналітичні.

Користувальницькі вимоги є найважливішими, саме користувач визначає наскільки виконання міжнародних стандартів важливе у його алгоритмі та наскільки складною для аналізу статичним аналізатором або аналітиком-людиною має бути обфускований файл. Міжнародні стандарти описують мету та структуру для розробника алгоритму обфускації. Програмні

та аналітичні вимоги є критерієм ресурсозатратності (тобто часу та знань/технологій) для розшифрування обфускованої інформації.

Головні методи обфускації:

- dead-code-insertion – введення до коду невиконуваних інструкцій, що ускладнюють його аналіз;
- лексична обфускація, що може бути виконана мануально, тобто, власноруч, або за допомогою алгоритму обфускації. Націлена на перейменування змінних та функцій, видалення коментарів автора, переносів строк та пробілів [3].

Другорядні вимоги до обфускатора:

- перетворення коду асиметричною односторонньою функцією, що є простою операцією для прямого розрахунку, але складною для зворотнього рішення комп'ютером. Для людини аналіз подібних функцій без застосування комп'ютера є майже неможливим. Даний метод може значно вплинути на швидкість роботи обфускованого файлу, тому використовується у випадках, коли швидкістю зчитування даних можна знехтувати заради їхнього захисту;
- розбиття цілих конструкцій програми на підконструкції, що дозволяє виконувати обфусковану програму поступово, не викриваючи її повний функціонал. Цей метод також значно впливає на швидкість зчитування даних, тому мало використовується.

Головні організаційні вимоги до цільового файлу:

- відповідність формату файлу з даними (*.C);
- некомпільований код, що містить інструкції на мові C.

Головні вимоги щодо інтерфейсу:

- простий інтерфейс користувача для взаємодії користувача, алгоритму обфускації та цільового файлу на основі CLI;
- наявність інформаційних повідомлень у випадку несправності роботи алгоритму.

Схему обфускації зображено на рис. 1.1.

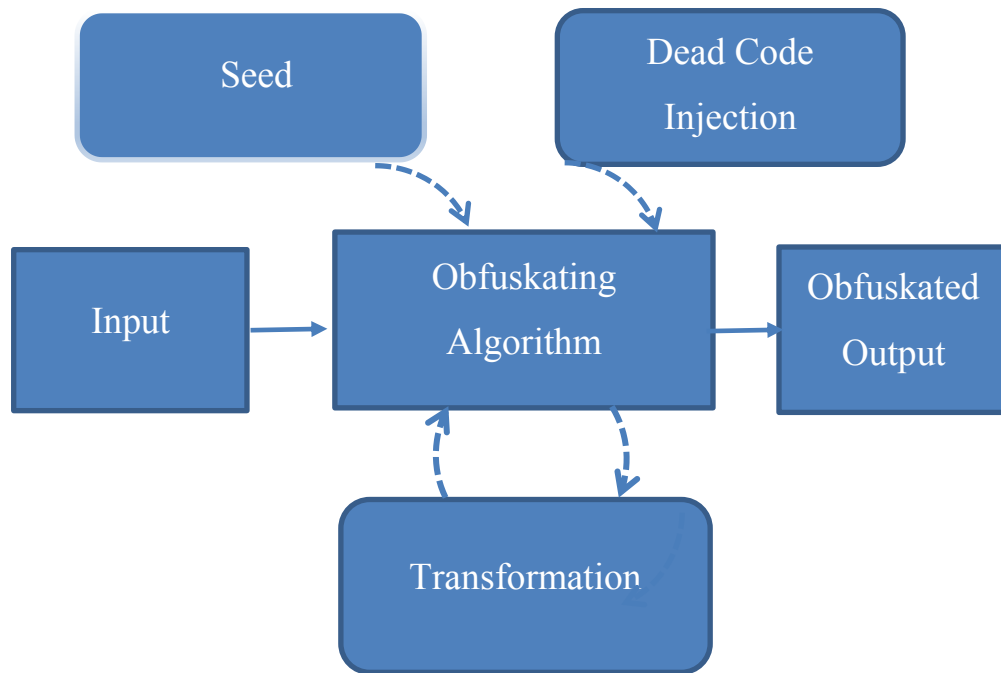


Рисунок 1.1 – Схема обфускації файлів

Процес обфускації – це складна математична і системна задача. Проте, навіть у наш час розробники обфускують та заплутують власні розробки власноруч, тобто, без використання алгоритмів. Мануальний метод обфускації є технічно слабким, а розробка програми такого роду для створення конкуренції раніше зауваженим продуктам на ринку неможлива.

Обфускація програми зазвичай відбувається на останньому етапі перед реалізацією програмного продукту. Саме тоді розробник обробляє незахищені дані обфускатором та отримує копії обфускованих даних. Також, важливим фактором є збереження функціональності на рівні необфускованої програми. Якщо розробник невпевнений в алгоритмі обфускації, то він може додатково протестувати його та порівняти функціонал з необфускованою версією. Після цього програмний продукт може бути реалізований користувачем.

Діаграма станів обфускатора має вигляд наведений на рис. 1.2.

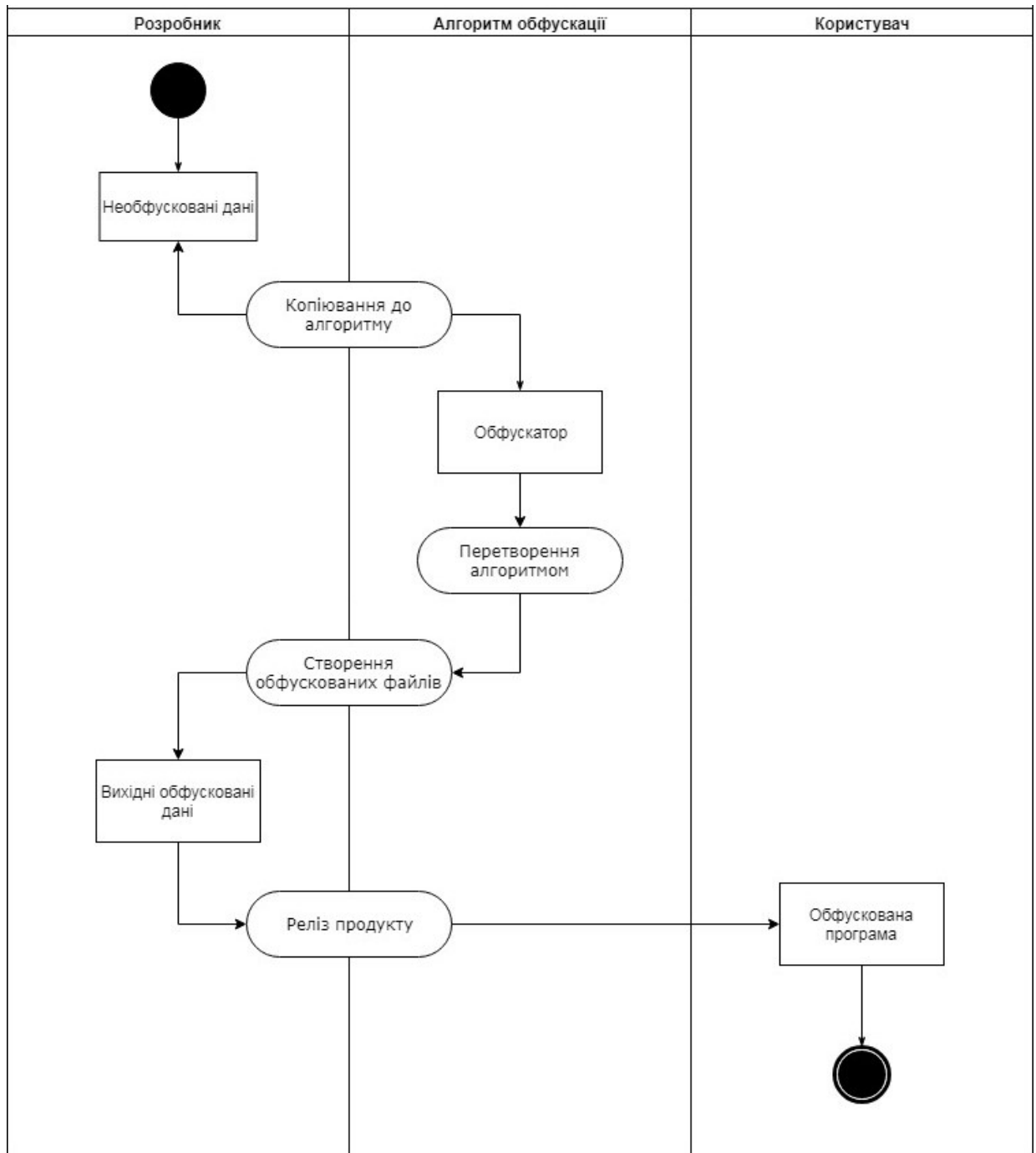


Рисунок 1.2 – Діаграма станів обфускатора

Так як обфускатори створені для використання досвідченими розробниками, що мають на меті приховати свою інтелектуальну власність, найзручнішим методом взаємодії є command line interface. Алгоритм обфускації отримує цільовий файл, створює його копію, у яку пізніше буде записано обфускований код файлу. Наступні кроки алгоритму залежать від вимог користувача і представляють структуру чорного ящика (Black Box Obfuscation).

Тобто користувач бачить лише вхідні дані та результати роботи. Такими складовими алгоритму є лексична обфускація: очищення коментарів розробника, переносів строк, пробілів. Для більшої заплутаності змінюються назви змінних, функцій, тощо. Для більшої заплутаності код модифікується рекурсивними викликами функцій та «засмічується» невиконуваними операціями. На основі цих кроків алгоритму можна зрозуміти, що мета обфускації – порушення якомога більше стандартів програмування зі збереженням функціональності коду.

По закінченню роботи алгоритму, обфусковані дані записуються до файлу, який для більшої заплутаності має нестандартний формат [4].

Копіювання даних включає в себе:

- перевірку цільового файлу;
- створення тимчасових файлів;
- розбиття та передача даних до тимчасових файлів.

Знаходження препроцесорних інструкцій включає в себе:

- перевірка рядків коду на наявність інструкцій;
- виключення препроцесорних рядків з алгоритму;
- запис у тимчасовий файл для зберігання.

Inline-методи обфускації представляють собою перший етап роботи алгоритму обфускації. Він перетворює числові та цифрові константи на сгенеровані вирази для ускладнення аналізу коду. Метод включає в себе:

- перевірка рядків на наявність констант;
- генерацію виразу для заміни константи у код;
- заміну константи на сгенерований вираз;
- компіляцію у препроцесорну інструкцію константи та заміни;
- запис до тимчасового файлу препроцесорної інструкції.

Видалення табуляції та коментарів включає:

- символний перебір коду на наявність табуляції та коментарів;
- пропуск символів табуляції;
- ідентифікацію однострокових та багаторокових коментарів;

- видалення кожного виду коментарів;
- перезапис коду без символів табуляції і коментарів.

Створення обфускованої копії включає:

- створення копії вихідного файлу з приставкою «obf»;
- копіювання вхідних препроцесорних інструкцій;
- копіювання препроцесорних інструкцій модифікації констант;
- копіювання модифікованого коду;
- закриття потоків та видалення тимчасових файлів.

1.2 Аналіз програмного забезпечення для захисту від дослідження за допомогою обфускації

Вперше постановка задачі і алгоритм обфускації без вживання терміна «обфускація» згадані в 1976 році в роботі Діффі і Хеллмана [5]. Десятиліття по тому, в січні 1986, обфускація знайшла своє практичне застосування у вірусі-шифрувальнику Brain. Важливо, що сам вірус був розроблений і інтегрований в обфускованому вигляді в код для захисту авторського права в програмах братів Альві з Пакистану. Коли піратські копії їх продукту розходилися по світу через дискети та Інтернет, вірус записував себе в пристрій зберігання записів, переносючи справжню інформацію в розділ пошкоджених файлів. Цей вірус був створений не з поганими цілями, а лише служив покаранням для нечесних користувачів.

В цьому ж році відбулося масове зараження комп'ютерів IBM обфускованим вірусом Cascade, який містив енкриптор / декриптор в одному файлі з зашифрованим виконуваним кодом. Важливо відзначити, що саме Cascade став в 80-х зразковим вірусом, який уникав виявлення завдяки обфускації. Так було до 1992 року, поки анонімний користувач Dark Avenger не реалізував поліморфний генератор вірусів The Mutation Engine. Його розробка не була вірусом, а була лише доповненням до нього. Викликаний генератор шифрував тіло вірусу випадковим ключем і генерував випадковий працюючий

дешифратор. Дана розробка дозволяла створювати унікальні копії вірусів, що не збігаються між собою жодним байтом і мають різну довжину [6].

Поняття і визначення процесу шифрування вихідного коду ввели Коллберг, Томборсон, Лоу в своїй науковій роботі в 1997, до них наукового обґрунтування не існувало. Вони запропонували перші види обфускації, класифікували їх і провели зв'язок з завданнями системного програмування. У наступних роботах була розширена сфера застосування обфускації до застосування в пошукових системах, базах даних, захисту проектних рішень. Науковими працями та публікаціями були описані теореми в області обфускації.

1. В процесі створення алгоритму Чорного ящика було доведено [7], що неможливо створити універсальний алгоритм обфускатора для вирішення всіх проблем разом. Для потреб криптографії необхідно маскувати дані (секретні ключі), коли системне програмування вимагало приховування алгоритмів від статичного і динамічного аналізу.

2. Дослідники відділили теоретичні вимоги до обфускації від практичних [8]. Це означало, що реалізовані в 90-х роках в комерційних програмах алгоритми були слабкими, а теоретично ідеальні методи мали неадекватно завищені захисні вимоги до практичних завдань, що могло вплинути на функціонал в негативну сторону.

3. Обфускацію можна було використовувати як захист від вірусних атак, але зворотним боком обфускації була визнана можливість її використання для створення і приховування вразливостей-бекдорів в системах і ускладнення виявлення шкідливих програм [8]-[9].

Твердження теореми 1 і її докази[7] стримали ентузіазм дослідників обфускації програм. Тому протягом цілих 6 років алгоритм Чорного ящика вважався найкращим з відомих в обфускації програм, поки в 2007 Голдвассер С. і Гай Н. теоретично не спростували цю думку [10]. У своїй роботі дослідники порівнювали теоретичні алгоритми з метою знаходження

обфускованої програми, яка розголошувала найменшу кількість інформації, ніж будь-яка інша програма з аналогічною функціональністю.

Подальший розвиток дослідження отримало в працях 2013 року [11]. Новий вид обфускатора отримав назву «обфускатор нерозрізненості» і його опис починався з двох програм, які отримують один результат різними методами. Наприклад, в обчисленні функцій, при будь-якому наборі змінних, «обфускатор нерозрізненості» пропонував шифрування, при якому користувач не міг визначити, яка функція знаходиться в його розпорядженні. Більш того, в поєднанні з односторонніми функціями обфускатор надавав можливість доказу з нульовим розголошенням, а гомоморфне шифрування надавало функціональне шифрування.

Саме «обфускатор нерозрізненості» став примітивом для обфускації і криптографії на рівні програмного забезпечення за останні 5 років.

Звичайно, обфускація – це не гарантований захист від злому програмного коду [12]. Перш за все процедура призначена для того, щоб максимально ускладнити шлях до розуміння програмного коду і алгоритму роботи програми передбачуваному зломщику.

Обфускація має і свої недоліки: наприклад, проведення обфускації для віртуальних об'єктів і абстрактних методів практично неможливе.

Варто звернути увагу на The International Obfuscated C Code Contest (IOCCC) – щорічний конкурс на обфускування C коду, що проходить з 1984. Ідея конкурсу з'явилася у двох розробників – Ларрі Бассела та Ландона Курт Нолла, що мусили оптимізувати роботу старих програм. Варто зауважити, що обидві програми були настільки нестандартно та неграмотно написані, що натхнули розробників створити власний конкурс на найскладніший код. Саме ці події дали початок міжнародному конкурсу, котрий до наших часів є актуальним.

Основні переваги розробок цього конкурсу – рефері відбирають найкращі роботи з невідомої кількості конкурсантив (згідно традиціям кількість конкурсантив не оголошується) а після публікують їх у відкритому доступі. За

більше ніж 30 років розробники зібрали величезний архів унікальних робіт з обфускації. Кожна робота містить короткі інструкції щодо роботи програми та усі необхідні файли, тому будь-який розробник може знайти концептуальні рішення для їх розробок, або переробити алгоритми під власні вимоги.

Проаналізуємо найвідоміші методи розробки обфускаторів.

1.2.1 Stunnix

Програмне забезпечення Stunnix є одним із найвідоміших, має випробувальний період з обмеженим функціоналом. Використовує inline-методи: прибирає пробіли та табуляцію, видаляє коментарі розробника або шифрує їх md5 хешем, об'єднує стрічки коду в єдине ціле для погіршення сприйняття коду. Включає заміну символьних та числових констант на хешовані вирази.

До переваг Stunnix відноситься велика кількість опцій для перетворення тексту. У меню програми можна налаштувати списки виключень, обрати інструменти, додати «водяні знаки», спеціальні вимоги до обробки або обрати варіації хешування.

Також програмний продукт має власну службу підтримки. Тому користувач завжди зможе вирішити будь-які проблеми зв'язані з його освоєнням або недоліками.

У Інтернеті немає «зламаних» версій обфускатора – отже алгоритм обробки на даний момент є безпечним для використання і невивченим реверс-інженерами. Вікно налаштувань обфускатора Stunnix на рисунку. 1.3.

До недоліків програми можна віднести застарілий користувацький інтерфейс та високу ціну за копію для одного користувача. Також існують пропозиції для корпоративних клієнтів – ціна якої не розголошується.

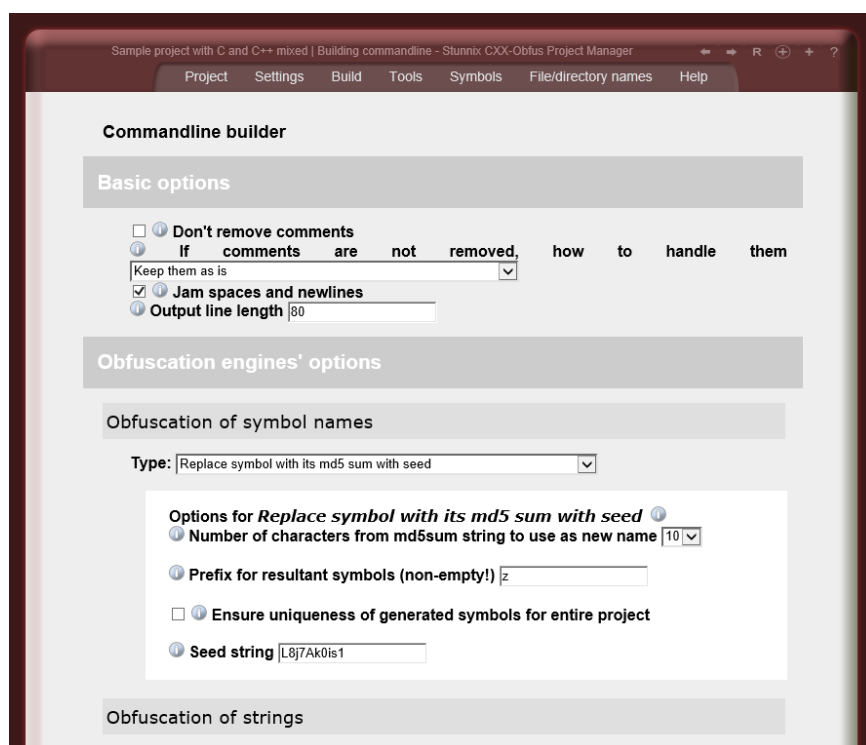


Рисунок 1.3 – Вікно налаштувань обфускатора Stunnix

До недоліків програми можна віднести застарілий користувацький інтерфейс та високу ціну за копію для одного користувача. Також існують пропозиції для корпоративних клієнтів – ціна якої не розголошується.

1.2.2 The Tigress C Diversifier/Obfuscator

Програмне забезпечення The Tigress C Diversifier/Obfuscator – одна з опублікованих розробок студентів The University of Arizona. Його розробка почалася у 2014 році і по 2017 рік розробники випускали оновлені версії свого продукту, що може свідчити про актуальність алгоритму на наш час. Поширюється безкоштовно через вебсайт на базі університету. Взаємодія з користувачем реалізована через CLI та включає в себе налаштування виключень. Основні функції програми – це модифікація коду для запобігання статичному аналізу. Це реалізовано за допомогою додавання у код рекурсивних викликів функцій, шифрування символьних констант та змінних.

Цикл роботи The Tigress C Diversifier/Obfuscator наведено на рис. 1.4.

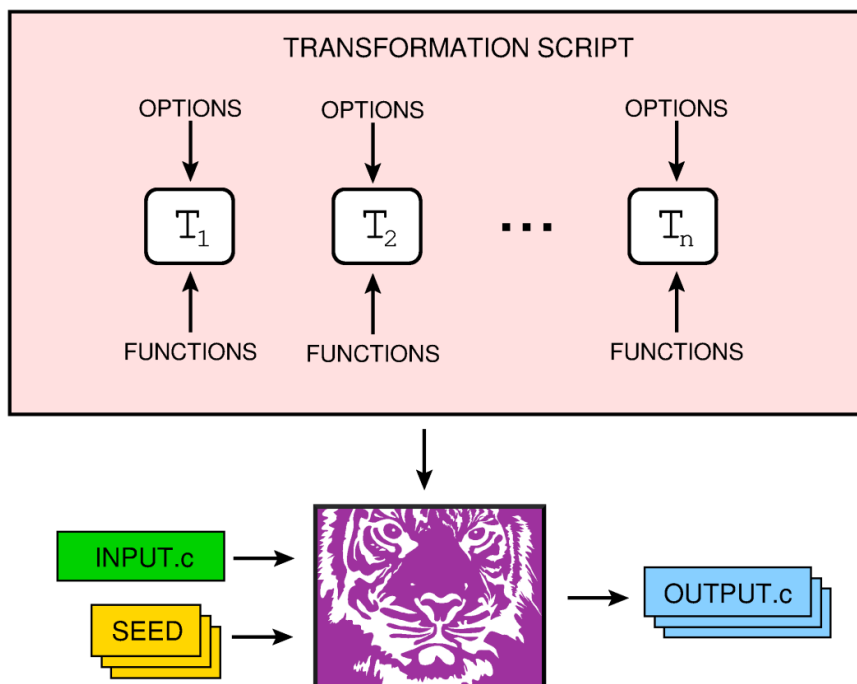


Рисунок 1.4 – Цикл роботи The Tigress C Diversifier/Obfuscator

З недоліків програми можна навести відсутність користувацького інтерфейсу та технічної підтримки. Продукт поки не має широкого поширення, тому невідомо чи алгоритм шифрування був розібраний реверс-інженерами.

1.2.3 Mangle-It C/C++ Code Obfuscator

Програмне забезпечення Mangle-It C/C++ Code Obfuscator – найстаріший з представлених у цьому розділі аналогів. Було реалізовано у 2010 році і з того часу не отримувало жодних оновлень. Програма є популярною, тому є великий ризик, що алгоритми шифрування були проаналізовані реверс-інженерами. Поширюється у вигляді архіву, що для початку роботи потребує активації. Випробувальних версій та корпоративних пропозицій не існує.

Головне вікно програми Mangle-It зображено на рис. 1.5.

До недоліків треба віднести відсутність оновлень і технічної підтримки. Користувацький інтерфейс є мінімалістичним, зважаючи на те, що він був розроблений в 2010 році. Сумісний для роботи лише на ОС Windows XP і молодше.

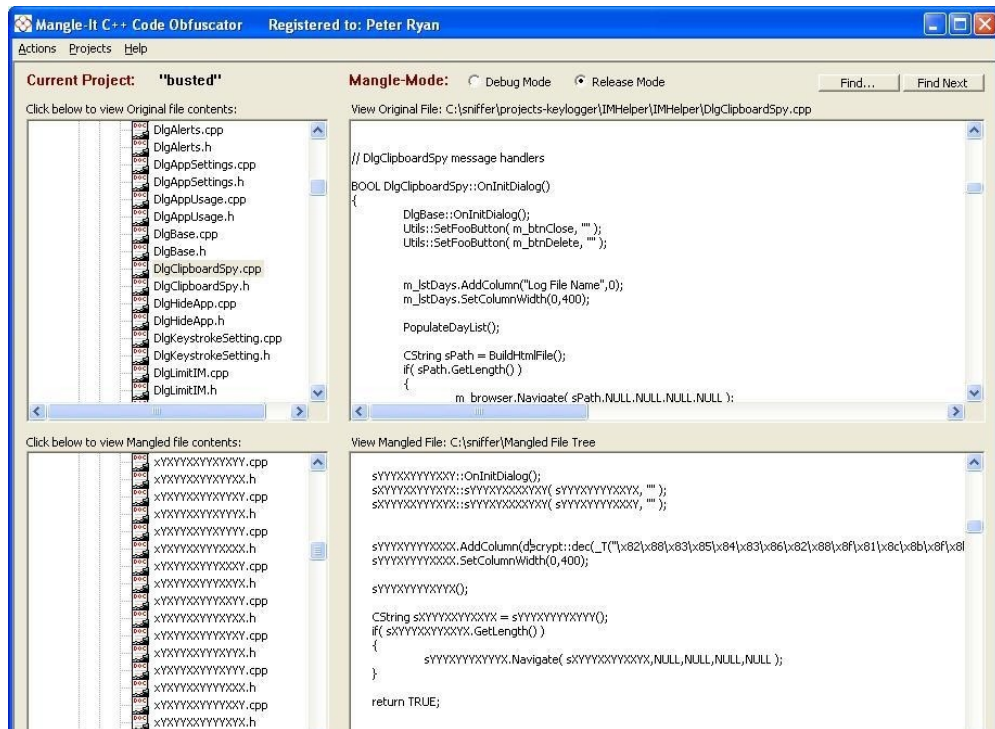


Рисунок 1.5 – Головне вікно програми Mangle-It

Також існує досить багато різноманітних онлайн-обфускаторів та free and open-source обфускаторів.

1.2.4 Порівняльна таблиця існуючих засобів обфускації

Так як більшість користувачів комп'ютерів не мають необхідності у обфускаторах, вони не мають великого вибору на ринку програмного забезпечення. Друга причина полягає у секретності алгоритмів обфускації – якщо обфускатор потрапить у широкий доступ, то реверс-інженери першими вивчать його принципи і технології і обфусковані ним файли будуть розшифровані у короткий період часу.

Основні показники розглянутих вище альтернативних методів обфускації наведені в табл. 1.1.

Таблиця 1.1 – Порівняння програмного забезпечення для захисту програмного коду методом обфускації

Критерій порівняння	Stunnix C Obfuscator	Tigress	Mangle-It
Розробник	Stunnix	The University Of Arizona (США)	PC Sentinel Software
Операційна система	Windows , Linux, Mac OS	Linux, Mac OS	Windows (98, 2000, XP)
Інтерфейс	Віконний	Command Line Interface	Віконний
Методи обфускації	Inline-макриси, хешування, рекурсія	Inline-макриси, відключення API, рекурсія,	Inline-макриси
Налаштування	Гнучкі, через користувацький інтерфейс	Гнучкі, через CLI	Відсутні
Вартість	\$500	Безкоштовно	\$70
Документація	Відсутня	Повна, за виключенням коду алгоритму	Відсутня

За результатами проведеного аналізу можна виділити такі продукти як Stunnix C Obfuscator та Tigress, що відповідають більшості вимог користувачів обфускаторів.

Продукт Mangle-It є стабільним та перевіреним часом продуктом. Але технології не стоять на місці і без постійних оновлень продукт залишився застарілим для використання на сучасних ОС.

Декілька слів про Stunnix C Obfuscator. Продукт є поширеним на ринку, має декілька оглядів та рекомендацій. За словами розробників, багато компаній зі списку Fortune500 використовують їх продукт [13]. Так як продукт був орієнтованим на широкий ринок, то він є сумісним з більшістю популярних ОС та має зручний користувацький інтерфейс. Зважаючи на наявність технічної підтримки та гнучких налаштувань ціна продукту в 500 доларів США за копію

відповідає запитам користувачів, а наявність корпоративних пакетів йде лише на користь продукту.

Особливо варто виділити альтернативу Tigress, котрий є фаворитом цього порівняння. Проєкт був розроблений студентами університету та реалізований як умовно-безкоштовний продукт. Має великий перелік методів обфускації, налаштування виключень з правил і гнучкі налаштування. Більшість користувачів обфускаторів це досвідчені розробники ПЗ, тому відсутність віконного користувацького інтерфейсу та сумісність роботи з Unix-подібними ОС не будуть проблемою.

Можна зробити висновок, що існує багато програмних засобів для обфускації. Проте деякі програмні засоби можуть бути схильні до технічних збоїв, тощо, що може негативно вплинути на процес обфускації.

Тому актуальною є розробка обфускатора програмного забезпечення з метою захисту від дослідження.

При розробці програмного забезпечення обфускатора необхідно забезпечити такі функціональні вимоги:

- можливість копіювання даних з цільового файлу;
- знаходження препроцесорних інструкцій та копіювання (тільки в *.c);
- наявність inline-методів обфускації: знаходження та приховання змінних;
- можливість видалення табуляції, пробілів та коментарів;
- об'єднання тимчасових файлів та створення обфускованої копії.

1.3 Висновки за розділом 1

У розділі розглянуто виникнення та основні поняття обфускації. Розглянуто основні принципи алгоритму обфускації. За результатами проведеного аналізу зроблено висновок, що існує багато програмних засобів для захисту програмного кода від рефакторингу методом обфускації.

Проте деякі існуючі програмні засоби можуть бути схильні до технічних збоїв, що може негативно вплинути на процес захисту.

Тому актуальною є розробка програмного забезпечення для захисту програмного коду від статичного аналізу та рефакторингу за допомогою обфускації. Сформульовано функціональні вимоги до програмного забезпечення.

2 МАТЕРІАЛИ І МЕТОДИ

2.1 Вибір мови програмування

Засоби обфускації програмного коду з аналогічним функціоналом розробляються частіше за все на таких мовах програмування як C, C++ та C#.

Результати досліджень розробницьких ресурсів щодо популярності використання мов програмування наведені на рис. 2.1.

IEEE	TIOBE	Packt
1. C 2. Java 3. Python 4. C++ 5. R 6. C# 7. PHP 8. JavaScript 9. Ruby 10. Go	1. Java 2. C 3. C++ 4. C# 5. Python 6. PHP 7. JavaScript 8. VB .NET 9. Perl 10. Assembly	1. JavaScript 2. Python 3. Java 4. C 5. SQL 6. PHP 7. HTML 8. CSS 9. R 10. Bash
PYPL	Redmonk	GitHub
1. Java 2. Python 3. PHP 4. C# 5. JavaScript 6. C++ 7. C 8. Objective-C 9. R 10. Swift	1. JavaScript 2. Java 3. PHP 4. Python 5. C# 6. C++ 7. Ruby 8. CSS 9. C 10. Objective-C	1. JavaScript 2. Java 3. Ruby 4. PHP 5. Python 6. CSS 7. C++ 8. C# 9. C 10. HTML

Рисунок 2.1– Найбільш популярні мови програмування

Оскільки ми розробляємо програму для Unix-подібних систем і будемо взаємодіяти з CLI то перевагу має мова C.

Головна перевага мови C є компіляція, що необхідно для процесу обфускації. З окремих переваг можна навести її швидкість через близькість її інструкцій до машинних.

Варто зауважити що Unix-подібні системи також написані мовою C. З аналогів можна розглянути C++ та C#, проте цей вибір залежить від розробника та його планів на розробку.

2.2 Вибір середовища розробки програмного забезпечення обфускатора

В якості середовища для розробки програми, обрані Unix-подібні ОС, як найбільш зручні середовища для розробників ПО.

До інструментів розробки відносяться наступні програмні продукти:

- середовище розробки (СР);
- стандартні методи мови програмування.

Перевагою мови C є можливість розробки коду зі звичайного текстового редактора. Відомими безкоштовними середовищами розробки є:

- Atom;
- Sublime Text;
- Netbeans;
- Eclipse.

Оскільки C є невибагливою до середовища розробки, то було обрано середовище Gedit.

Gedit має мінімалістичний дизайн та мультязикову підтримку. Також це середовище є портативним і може працювати навіть на віртуальних машинах. У якості відладчика обрано відладчик Valgrind.

2.3 Висновки за розділом 2

Для реалізації програмного забезпечення обрано мову програмування C, яка дозволяє створювати програми, що характеризуються високою продуктивністю. Це дозволяє досягти високої швидкості виконання програм, що є важливим.

Для створення програмного забезпечення для рефакторингу за допомогою алгоритму обфускації обрано середовище розробки Gedit, що представляє собою вільний текстовий редактор робочого середовища GNOME, MacOS та Windows з підтримкою Юнікоду. У якості відладчика обрано Valgrind.

3 ОПИС ПРОГРАМИ

3.1 Структура програмного забезпечення для захисту програмного коду від дослідження

Структурна схема програмного забезпечення для запобігання рефакторингу за допомогою алгоритму обфускації зображена на рис. 3.1.

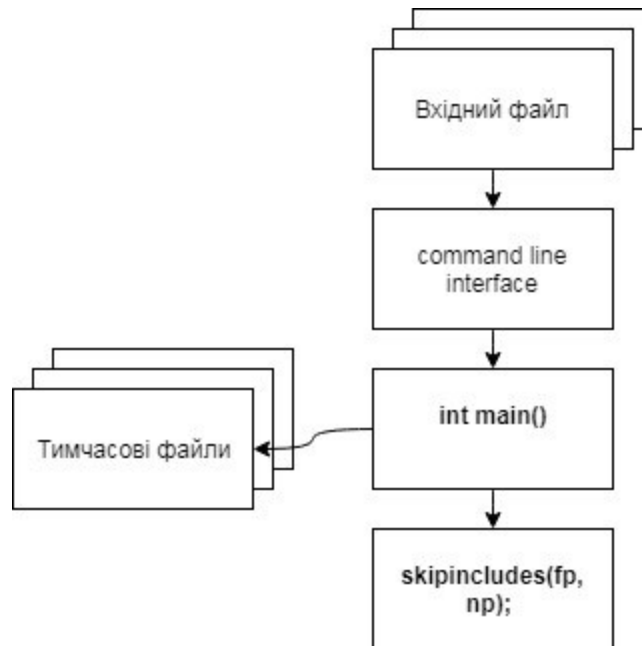


Рисунок 3.1 – Структурна схема програми

3.1.1 Структура функції main()

У алгоритмі ця функція відповідальна за відкриття нових тимчасових файлів, перевірку цільового файлу, поступовий виклик функцій алгоритму та запис і компонування вихідних даних. Структура функції main наведена на рис. 3.2.

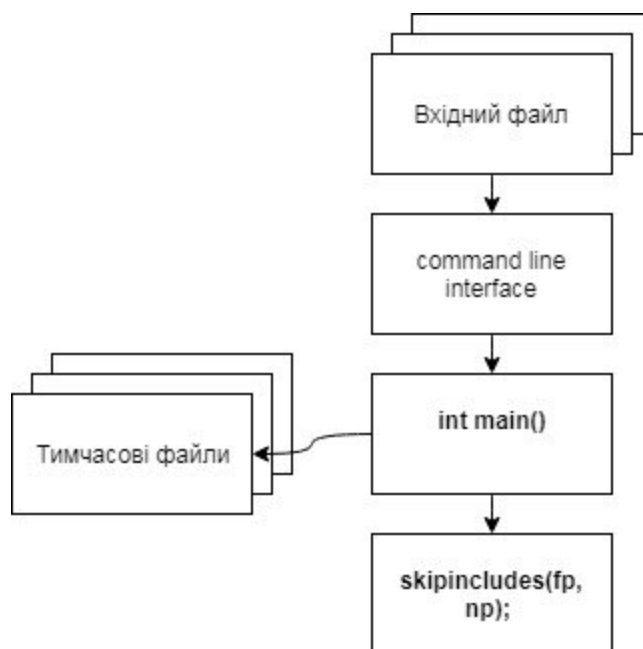


Рисунок 3.2 – Структурна схема функції main

Код функції main():

```

int main(int c, char **arr)
{
    FILE *fp, *np, *np2, *np3, *np4, *tp;
    int ch, prev = 0;
    bool String = 0;
    long checkpoint;
    char incl[100];
    char *replace;

    char rename[100] = "obf.c";
    fp = fopen(arr[1], "r");
    strcat(arr[1], rename);
    np = fopen("temp1.c", "w+");
    np2 = fopen("temp2.c", "w+");
    np3 = fopen("temp3.c", "w+");
    np4 = fopen("temp4.c", "w+");
    tp = fopen(arr[1], "w");
  
```

```

if (fp == NULL)
{
    printf ("Invalid file!\n");
    return 1;
}

```

В даній частині об'явлені змінні необхідні для більшості викликаємих функцій. Також створюються потоки для вхідного файлу, тимчасових файлів та фінального обфускованого файлу.

```

skipincludes(fp, np);

inttodefine(fp, np2, np3);

clearlines(np3, np4);

```

Порядок виклику функцій котрі будуть розглянуті далі по порядку їх виклику. Кожній функції передаються потоки даних для запису або зчитування з файлу.

```

fseek(np, 0, SEEK_SET);
while(fgets(incl, sizeof(incl), np) != NULL)
{
    fputs(incl, tp);
}

fseek(np2, 0, SEEK_SET);
while(fgets(incl, sizeof(incl), np2) != NULL)
{
    fputs(incl, tp);
}
fputs("\n", tp);
fseek(np4, 0, SEEK_SET);

```

```

while(fgets(incl, sizeof(incl), np4) != NULL)
{
    fputs(incl, tp);
}

```

Даний елемент коду виконується по завершенню роботи програми. Він обнуляє покажчики потоків на початок файлу та копіює дані з тимчасових файлів у порядку їх виклику та komponує їх у кінцевий файл.

```

fclose(fp);
fclose(np);
fclose(np2);
fclose(np3);
fclose(np4);
fclose(tp);
remove("temp1.c");
remove("temp2.c");
remove("temp3.c");
remove("temp4.c");
return 0;
}

```

Закриття потоків для зчитування-запису у файли та видалення тимчасових файлів.

3.1.2 Структура функції skipincludes()

Функція що відповідальна за знаходження предпроцесорних інструкцій та перенесення їх до окремого файлу аби запобігти їх модифікації. Структура функції skipincludes наведена на рис. 3.3.

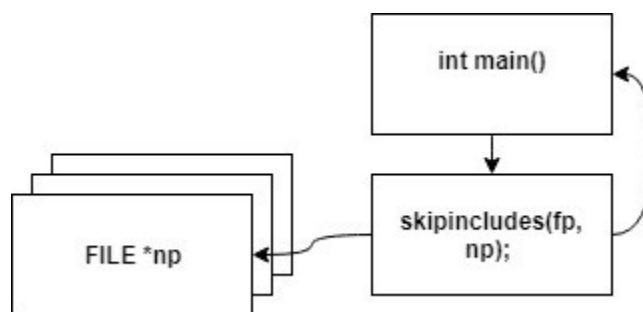


Рисунок 3.3 – Структурна схема функції skipincludes

```

void skipincludes(FILE *fp, FILE *np)
{
    char incl[128];
    char clude[128];
    char *ude = "#include ";

    while(fgets(incl, sizeof(incl), fp) != NULL)
    {
        if (strstr(incl, ude) > 0 && incl != NULL)
        {
            fputs(incl, np);
        }
        else
        {
            fputs(incl, np);
            break;
        }
    }
}
  
```

Функція построково перевіряє вихідний файл `fp` на наявність препроцесорної інструкції такої як `include`. У випадку збігу стрічка переноситься до тимчасового файлу `np`, де буде зберігатися недоторканною іншими алгоритмами.

3.1.3 Структура функції inttodefne()

Алгоритм заміни констант на випадково сгенеровані вирази. Це ускладнює сприйняття тексту і наскільки багато констант буде замінено залежить від вимог розробника. Структура функції inttodefne наведена на рис. 3.4.

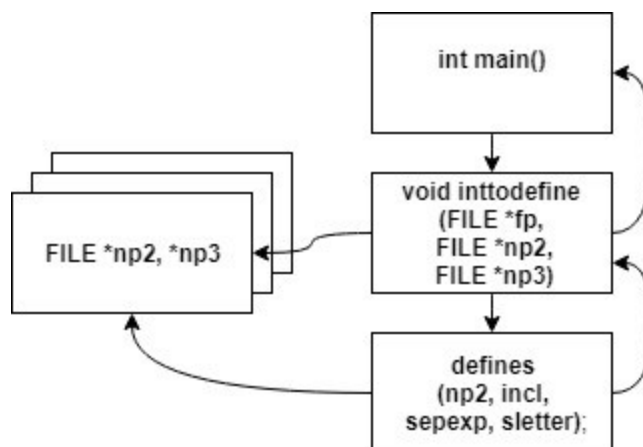


Рисунок 3.4 – Структурна схема функції inttodefne

```

void inttodefne(FILE *fp, FILE *np2, FILE *np3)
{
    char incl[128];

    while(fgets(incl, sizeof(incl), fp) != NULL)
    {
        while ((int)strstr(incl, "int
") > strlen(incl) && (int)strstr(incl, "int main") < strlen(incl))
        {

            int position = strcspn(incl, "int ") + 4;
            char septype[100] = { 0 }, sepexp[100], secesp[100], z
eros[100] = { 0 };

            strncpy(septype, incl, position);

```

```

    strncpy(sepexp, (incl + position), 128);

    int secret = rand() % 5000 + 4999;
    sprintf(secesp, "%d", secret);
    char sletter [10] = "0x0";
    strcat(sletter, secesp);
    puts(sletter);

    defines(np2, incl, sepexp, sletter);

    fputs(septype, np3);
    fputs(sletter, np3);
    fputs(";", np3);

    break;

}

```

Функція построково перевіряє почтаковий файл на наявність ініціалізації чисельних змінних `int`. Заходячи їх, функція генерує випадкове число, що поєднується з приставкою «0x0» та передається у функцію `defines()`, котра буде описана наступною. Наступним кроком функція замінює змісти стрічки на згенерований вираз.

```

while ((int)strstr(incl, "for") > strlen(incl))
{

    int position = strcspn(incl, "for") + 3;
    char septype[100] = { 0 }, sepexp[100], secesp[100], z
eros[100] = { 0 };

    strncpy(septype, incl, position);
    strncpy(sepexp, (incl + position), 128);

```

```

    int secret = rand() % 5000 + 4999;
    sprintf(secesp, "%d", secret);
    char sletter [10] = "0x0";
    strcat(sletter, secesp);
    puts(sletter);

    defines(np2, incl, sepexp, sletter);

    fputs(septype, np3);
    fputs(" ", np3);
    fputs(sletter, np3);
    fputs(" ", np3);

    break;

}

```

Аналогічним чином замінюються цикли for.

```

    while ((int)strstr(incl, "int
") <= strlen(incl) && (int)strstr(incl, "for") <= strlen(incl))
    {
        fputs(incl, np3);
        break;
    }

    while((int)strstr(incl, "int main") != 0)
    {
        fputs(incl, np3);
        break;
    }
}
}

```

Функції, що переносять інші стрічки незмінними до файлу та обробка винятку для `int main()`.

3.1.4 Структура функції `defines()`

Функція що відповідає за перенесення замієнених у `inttodefіne()` виразів. Створює препроцесорну інструкцію та вкладає її у тимчасовий файл. Структура функції `skipіncludes` наведена на рис. 3.5.

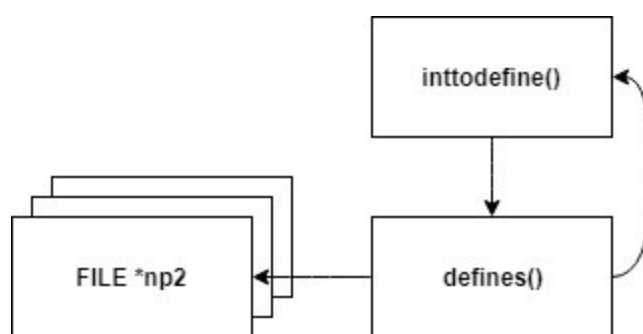


Рисунок 3.5 – Структурна схема функції `skipіncludes`

```

void defines(FILE *np2, char *incl, char *sletter, char *sepexp)
{
    fputs("#define ", np2);
    fputs(sepexp, np2);
    fputs(" ", np2);
    fputs(sletter, np2);
}
  
```

Після вкладання справжнього виразу функція генерує декілька несправжніх виразів та вкладає їх у тойже тимчасовий файл. Цей прийом у обфускації називається `dead code injection`.

```

int secret = rand() % 3 + 1;
for(int i = 0; i < secret; i++)
{
  
```

```

char fakesecesp[100], fakesecesp2[100], fakesecesp3;

int fakesecret = rand() % 5000 + 4999;
long fakesecret2 = rand() % 999999999999 + 1000000000000;
fakesecesp3 = fakesecret * 16;
sprintf(fakesecesp, "%d", fakesecret);

sprintf(fakesecesp2, "%d", fakesecesp3);

char fakesletter [10] = "0x0";
strcat(fakesletter, fakesecesp);

fputs("#define ", np2);
fputs(fakesletter, np2);
fputs(" ", np2);
fprintf(np2, "%lu", fakesecret2);
fputs(";", np2);
while(i < secret)
{
    fputs("\n", np2);
    break;
}
}
}

```

3.1.5 Структура функції clearlines()

Функція що посимвольно шукає табуляцію та коментарі. Знаки табуляції пропускаються, для однострокових і багаторокових коментарів створені окремі функції, котри будуть описані пізніше. Структура функції clearlines() наведена на рис. 3.6.

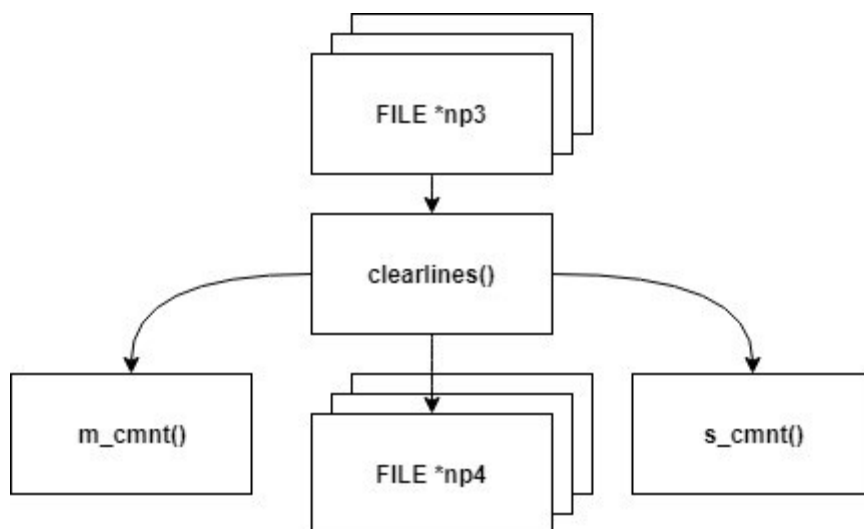


Рисунок 3.6 – Структурна схема функції clearlines

```

void clearlines(FILE *np3, FILE *np4)
{
    fseek(np3, 0, SEEK_SET);
    char incl[128];
    int prev = 0, ch;
    bool String = 0;
    int i = 0;
    while ((ch = getc(np3)) != EOF)
    {
        i++;
        if (!String)
        {
            if (ch == '/')
            {
                ch = getc(np3);

                switch(ch)
                {
                    case '*': m_cmnt(np3); break;
                    case '/': s_cmnt(np3, np4); break;
                    default: putc(prev, np4); putc(ch, np4); break;
                }
            }
        }
    }
}

```

```

    }
}

else if (ch == '\n')
{
    while(ch == EOF)
    {
        break;
    }
}
else if (ch == '\t')
{

}

else putc(ch, np4);
}

else putc(ch, np4);

}
}

```

3.1.6 Структура функцій s_cmnt() та m_cmnt()

Функція що викликаються функцією clearlines(). s_cmnt() спрямована на видалення однострокових коментарів, m_cmnt() займається многостроковими. Структура функцій s_cmnt() та m_cmnt() наведена на рис. 3.7.

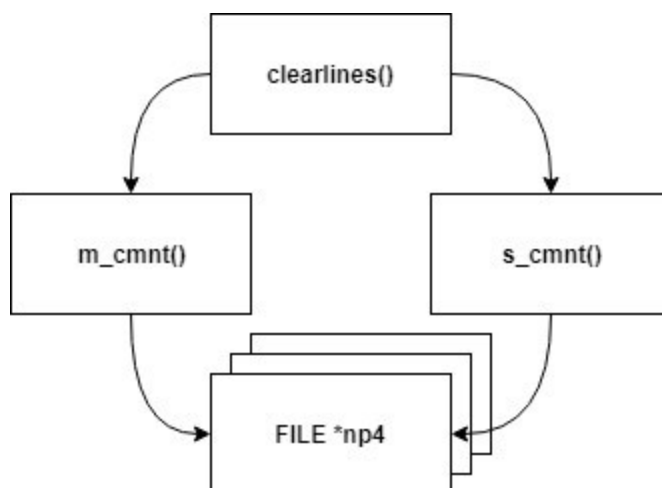


Рисунок 3.7 – Структурна схема функцій s_cmnt() та m_cmnt()

```

void m_cmnt(FILE *np3)
{
    int prev = 0, ch;
    while ((ch = getc(np3)) != EOF)
    {
        if (prev == '*' && ch == '/')
            return;
        else
            prev = ch;
    }
}

void s_cmnt(FILE *np3, FILE *np4)
{
    int ch;
    while ((ch = getc(np3)) != '\n' && ch != EOF);
    while ((ch = getc(np3)) == '\n')
    {
        getc(np3);
    }
    putc(ch, np4);
}

```

3.2 Функціонування програмного забезпечення для запобігання рефакторінгу

Обфускація є процесом заплутування та ускладнення сприйняття вхідних даних. Існує велика кількість методів та інструментів для обфускації.

Для розробки власного алгоритму обфускації було обрано поширені методи, що значно ускладнюють аналіз даних для реверс-інженера.

Один з важливих кроків це очистка коментарів розробника. Наявність коментарів може значно полегшити роботу нападнику своєю інформативністю. Тож, від коментарів обов'язково треба позбавитися. Коментарі бувають одностроковими і багатостроковими. Алгоритм має видаляти обидва види.

Наступним алгоритмом є видалення табуляції. Це руйнує структурованість коду, що може додати часу для розшифрування нападнику. Проте, це прямо залежить від об'єму програми.

Алгоритм заміни констант на випадково сгенеровані вирази. Це ускладнює сприйняття тексту і наскільки багато констант буде замінено залежить від вимог розробника.

Після виклику з CLI користувачем обфускатора та цільового файлу починається копіювання тексту з вхідного файлу `*fr`. Варто зауважити що після цього втручання користувача у роботі програми закінчується. Функція `main()` перевіряє якщо файл має дані та створює тимчасові файли `*np`, `*np2`, `*np3`, `*np4`.

Починається робота функцій. Кожна з функцій зчитує частину файлу за переносить її до тимчасового файлу. Це необхідно для запобігання пошкодження даних при виконанні перетворень. Наприклад, функція `skipincludes()` копіює препроцесорні інструкції `#include` до окремого файлу.

По завершенню роботи функцій описаних раніше функцій, `main()` збирає зміст тимчасових файлів у необхідній для компіляції порядку у фінальному файлі `*tp`. Фінальний файл зберігається у одній папці з вхідним файлом та має

приставку «.obf.c». Тимчасові файли очищуються і програма закриває потоки вводу-виводу. Функціональна схема програми зображена на рис.3.8.



Рисунок 3.8 – Функціональна схема програми

3.3 Опис функцій алгоритму обфускації

Функція копіювання та створення взаємодіє з користувачем через CLI. Надалі алгоритм не вимагає від користувача жодних втручань. Функція створює необхідні тимчасові файли та перевіряє наявність вхідних даних.

Функція перевірки препроцесорних інструкцій пострічково перевіряє вхідні дані на наявність будь яких препроцесорних інструкцій, котрі необхідні для функціонування обфускованої програми. Знайдені інструкції копіюються до тимчасових файлів.

Функція створення фінального обфускованого файлу копіює інформацію з тимчасових файлів та komponує їх в правильному порядку для функціонування програми.

3.4 Висновки за розділом 3

У розділі були розглянуті і обрані зручні для розробки програмного забезпечення такі як мова програмування, середовище розробки і відладчик.

Як середовище розробки, обрано середовище gedit, що є мінімалістичним та відповідаючим вимогам розробника.

В якості мови програмування проєкту обрано C – компілюєму мову програмування з близькими до машинних інструкціями.

Була спроектована структурна схема програми, що розробляється та складено опис модулів цієї програми.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Призначення та умови застосування програми

Програма призначена для захисту програмного коду від рефакторингу методом обфускації.

Для експлуатації програмного забезпечення захисту програмного коду методом обфускації необхідно наступне апаратне та програмне забезпечення:

- оперативна пам'ять об'ємом не менше 512Mb;
- процесор із тактовою частотою не менше 700MHz;
- наявність Інтернет браузеру з оновленнями, а також стійкого Інтернет з'єднання;
- наявність монітора з розрішенням не менше 1024 x 768 (рекомендується 1440 x 900 і більше).

Функціональні характеристики розробленого програмного забезпечення приведено у технічному завданні (додаток А). Фрагмент тексту програмного забезпечення для захисту програмного коду від дослідження, приведено у додатку Б.

4.2 Характеристики програми для захисту програмного коду від дослідження

Програмне забезпечення для захисту програмного коду від рефакторингу методом обфускації розроблено у середовищі gedit з відладчиком valgrind.

Зберігання даних під час роботи алгоритму реалізоване тимчасовими файлами, після завершення роботи програма не зберігає жодних даних для наступної сесії.

4.3 Звернення до програми

Через CLI викликається виконуваний файл програми з адресою та іменем цільового файлу. Приклад виклику алгоритму обфускатора через CLI наведено на рис. 4.1.

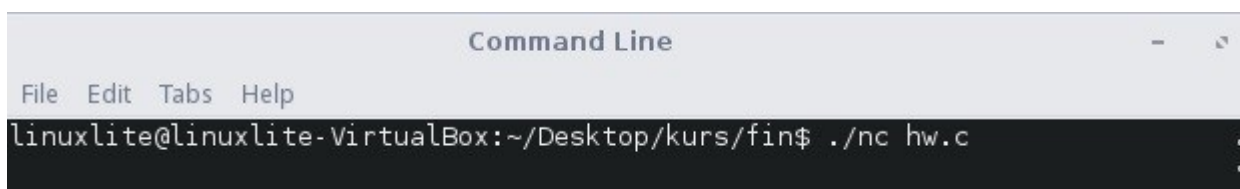


Рисунок 4.1 – Приклад виклику алгоритму обфускатора

4.4 Вхідні й вихідні дані

Вхідними даними програми є *.C файл призначений для обфускації.

Вихідні дані програми – обфускована копія вхідного файлу без втрати функціональності.

Для наявності, вигляд папки з обфускатором зображено на рис. 4.2.

Зміст:

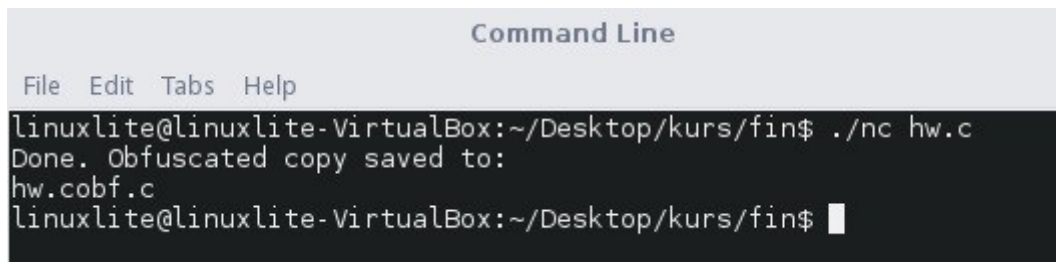
- hw.c – вхідний файл;
- hw.cobf.c – вихідна обфускована копія hw.c;
- nc.c – код обфускатора;
- nc – скомпільований обфускатор.



Рисунок 4.2 – Зміст папки з обфускатором

4.5 Повідомлення

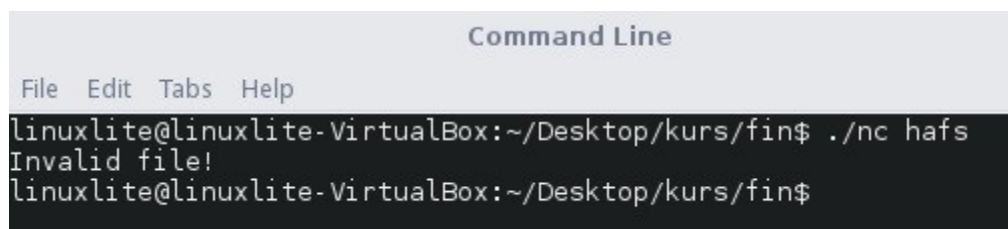
Повідомлення користувачу виводяться через CLI. В результаті успішного виконання програми користувач отримає відповідне повідомлення з ім'ям файлу. Приклад такого повідомлення зображено на рис. 4.3.



```
Command Line
File Edit Tabs Help
linuxlite@linuxlite-VirtualBox:~/Desktop/kurs/fin$ ./nc hw.c
Done. Obfuscated copy saved to:
hw.cobf.c
linuxlite@linuxlite-VirtualBox:~/Desktop/kurs/fin$
```

Рисунок 4.3 – Повідомлення про успішне виконання алгоритму

При помилках користувача видається повідомлення про помилку. Приклад повідомлення некоректного вхідного файлу наведено на рис. 4.4.



```
Command Line
File Edit Tabs Help
linuxlite@linuxlite-VirtualBox:~/Desktop/kurs/fin$ ./nc hafs
Invalid file!
linuxlite@linuxlite-VirtualBox:~/Desktop/kurs/fin$
```

Рисунок 4.4 – Повідомлення некоректного вхідного файлу

4.6 Висновки за розділом 4

Було розглянуто призначення, умови застосування алгоритму, список компонентів ПП та їх опис, спосіб звертання до програми, підготовку програми до запуску, вхідні та вихідні дані і повідомлення про помилки.

ВИСНОВКИ

В ході виконання дипломної кваліфікаційної роботи магістра було розроблено автоматизований алгоритм обфускації програмного коду.

За призначенням алгоритм спрямований допомогти розробникам програмного забезпечення захистити їх інтелектуальну власність, а саме програмний код. В якості ОС для роботи алгоритму обрано GNU/Linux.

Інтелектуальна власність має широке поширення завдяки Інтернету, нерідкі випадки розкрадання, модифікації і підміни походження власності. Програмне забезпечення, як форма інтелектуальної власності, досить часто представляє інтерес для шахраїв. Шахраї мають на меті розповсюдження чужого програмного коду, шукають помилки у вихідному коді для подальшої їх експлуатації та самозбагачення, тощо. Для запобігання шахрайським схемам, що мають на меті дослідження та розповсюдження чужого програмного коду існують автоматизовані алгоритми перетворення програмного коду у складний текст для розуміння як людиною так і комп'ютером без втрати функціональності, які називаються обфускацією.

Алгоритми обфускації призначені для захисту програмного вихідного коду від дослідження.

За результатами проведеного аналізу зроблено висновок, що існує багато програмних засобів для захисту програмного коду від рефакторингу методом обфускації.

Проте деякі існуючі програмні засоби можуть бути схильні до технічних збоїв, що може негативно вплинути на процес захисту.

Тому актуальною є розробка програмного забезпечення для захисту програмного коду від статичного аналізу та рефакторингу за допомогою обфускації. Сформульовано функціональні вимоги до програмного забезпечення.

В якості середовища розробки обрано gedit з відладчиком valgrind. В якості мови програмування проєкту обрано C – компілюєму мову, максимально подібно своїми інструкціями до ОС Linux.

Було розглянуто основні функції, використані у програмі. Також описано логічну структуру та порядок виконання функцій алгоритму.

Було розглянуто призначення, умови застосування програми, список компонентів ПП та їх опис, спосіб звертання до програми, підготовку програми до запуску, вхідні та вихідні дані і повідомлення про помилки.

У рамках роботи були виконані наступні кроки:

- проведено аналіз предметної області з початку виникнення обфускації та еволюцію алгоритмів;
- визначено структуру та функції алгоритму, що розроблюється та обрано середовище та інструменти розробки програмного забезпечення;
- розроблено алгоритм обфускації, його функції та методи шифрування, алгоритм перевірено на стійкість до статичного аналізу;
- спроектовано систему взаємодії алгоритму з файлами через інтерфейс командної строки;
- проаналізовано результати роботи програми.

Проведено тестування розробленого програмного забезпечення, яке показало високу надійність та стабільність програмного продукту.

Результатом роботи вважається розробка автоматизованого алгоритму обфускації для програмного коду C без втрати функціоналу коду.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Obfuscation (software) [Electronic resource]. – Access mode: [https://en.wikipedia.org/wiki/Obfuscation_\(software\)](https://en.wikipedia.org/wiki/Obfuscation_(software)).
2. ISO/IEC 27038:2014 – Information technology – Security techniques – Specification for digital redaction [Electronic resource]. – Access mode: <http://www.iso27001security.com/html/27038.html>.
3. Code obfuscation [Electronic resource]. The National Cyber Security Centre. – Access mode: https://www.ncsc.gov.uk/content/files/protected_files/guidance_files/Code-obfuscation.pdf.
4. Diffie, W., New directions in cryptography / W. Diffie, M. Hellman // IEEE Transactions on Information Theory – 1976, IT-22(6), P. 644-654.
5. A Brief History of Malware Obfuscation: Part 1 of 2 [Electronic resource]. – Access mode: https://blogs.cisco.com/security/a_brief_history_of_malware_obfuscation_part_1_of_2.
6. Barak B. On the (im)possibility of obfuscating programs / B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, Ke Yang. – S.: Springer-Verlag, 2001. – 1-18 p.
7. Collberg C. A Taxonomy of Obfuscating Transformations» / C. Collberg, C. Thomborson, D. Low – A: Technical Report – N148 – S: Univ. of Auckland, 1997. P.64-88.
8. Chess D. An undetectable computer virus / D. Chess, S. White // Proceedings of the 2000 Virus Bulletin Conference : міжнар. наук.-техн. конф – 2000.
9. Goldwasser, S. On best-possible obfuscation / S. Goldwasser, N. R. Guy; [Fourth IACR Theory of Cryptography Conference]. – A.: KNAW Trippenhuis, 2007. P. 194-213.

10. Candidate indistinguishability obfuscation and functional encryption for all circuits. / S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, B. Waters; – X.: UCLA, 2013 – 46 p.

11. Леощенко С.Д. Средства обфускации как инструменты защиты программ и данных от рефакторинга [Текст] / С.Д. Леощенко, Т.А. Зайко // Тиждень науки: щоріч. наук.- практ. конф., 2016 – Запоріжжя : ЗНТУ, с. 1-2.

12. Зайцев М.А. Розробка алгоритму обфускації даних, як засобу захисту програм та даних від рефакторингу [Текст] / М.А. Зайцев, Т.А. Зайко // Тиждень науки: щоріч. наук.- практ. конф., 2018 – Запоріжжя : ЗНТУ, с. 1-2.

13. About StunniX. [Electronic resource]. – Access mode: <http://stunniX.com/about/>.

ДОДАТОК А
Технічне завдання

A.1 Вступ

Програма використовується розробниками програмного забезпечення на мові С для захисту інтелектуальної власності.

A.2 Підстава для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Дослідження та програмна реалізація методів захисту програмного забезпечення від рефакторингу», затверджене наказом Національного університету «Запорізька політехніка» № 398 від 07 жовтня 2024 р.

A.2 Призначення розробки

Програма призначена для обфускації програмного коду на мові С без втрати функціоналу.

A.3 Основні вимоги до програми, що розробляється

A.3.1 Вимоги до функціональних характеристик

Виходячи з постановки задачі, були сформовані основні функціональні вимоги алгоритму.

Додаток має надавати користувачеві можливість обфускації некомпільованого коду на С без втрати функціональності.

A.3.2 Вимоги до інтерфейсу програми

Інтерфейс програмного забезпечення для підтримки процесу функціонування Інтернет аукціону повинен бути зручним для користувачів. Повинна бути забезпечена можливість роботи в візуальному режимі.

А.3.3 Вимоги до надійності

Для коректної роботи програми користувач має слідувати написаному у інструкції, коректно вказувати параметри та звертати увагу на повідомлення програми.

При використанні системи можуть виникнути такі незалежні від користувача ризики:

Відмова устаткування. У разі відмови обладнання може призвести до завершення роботи алгоритму без отримання вихідних даних. Проблема виправляється повторним запуском алгоритму.

Втрата або пошкодження даних. Може статися як через відмову обладнання або через неувважність користувача алгоритмом. Первинно проблема вирішується перезавпуском алгоритму. Якщо проблема пошкодження даних повторюється з кожною сесією, то користувачу рекомендовано перевірити вхідні дані на наявність проблем використання пам'яті. Для автоматизації рекомендовано використовувати ПЗ відладчика valgrind.

А.3.4 Умови експлуатації

Для забезпечення роботи системи необхідний комп'ютер з мінімальними вимогами і встановленим ОС Linux Lite.

Особливих вимог до експлуатації системи не пред'являється.

А.3.5 Вимоги до складу та параметрів технічних засобів

Для експлуатації даної системи необхідні такі мінімальні вимоги ПК.

Мінімальна конфігурація:

- ОЗУ: мінімум 512 Мбайт;
- процесор: 700 MHz;
- HDD: мінімум 8 Гбайт;

- версія Linux Lite;
- тип системи: 32 чи 64-розрядна;
- наявність монітора з розрішенням не менше 1366 x 768.

A.3.6 Вимоги до маркування і пакування

Програма для підтримки функціонування Інтернет аукціону може бути записана на будь-якому носії інформації.

На пакуванні повинна бути назва програми – «Програмне забезпечення для обфускації програмного коду»

A.4 Вхідні дані до роботи

Вхідними даними програми є *.С файл призначений для обфускації.

Вихідні дані програми – обфускована копія вхідного файлу без втрати функціональності.

ДОДАТОК Б
Фрагмент тексту програми

Б.1 Текст файлу nc.c

```

#include <stdio.h>
#include <stdbool.h>
#include <string.h>
#include <stdlib.h>

void m_cmnt(FILE *np3)
{
    int prev = 0, ch;

    while ((ch = getc(np3)) != EOF)
    {
        if (prev == '*' && ch == '/')
            return;
        else
            prev = ch;
    }
}

void s_cmnt(FILE *np3, FILE *np4)
{
    int ch;
    while ((ch = getc(np3)) != '\n' && ch != EOF);
    while ((ch = getc(np3)) == '\n')
    {
        getc(np3);
    }
    putc(ch, np4);
}

void clearlines(FILE *np3, FILE *np4)
{
    fseek(np3, 0, SEEK_SET);
    char incl[128];
    int prev = 0, ch;
    bool String = 0;
    int i = 0;
    while ((ch = getc(np3)) != EOF)
    {
        i++;

```

```

if (!String)
{

    if (ch == '/')
    {
        ch = getc(np3);

        switch(ch)
        {
            case '*': m_cmnt(np3); break;
            case '/': s_cmnt(np3, np4); break;
            default: putc(prev, np4); putc(ch, np4); break;
        }
    }

    else if (ch == '\n')
    {
        while(ch == EOF)
        {
            break;
        }
    }
    else if (ch == '\t')
    {

    }

    else putc(ch, np4);
}

else putc(ch, np4);

}
}

void defines(FILE *np2, char *incl, char *sletter, char *sepexp)
{
    fputs("#define ", np2);
    fputs(sepexp, np2);
    fputs(" ", np2);
    fputs(sletter, np2);

```

```

int secret = rand() % 3 + 1;
for(int i = 0; i < secret; i++)
{
    char fakesecesp[100], fakesecesp2[100], fakesecesp3;
    int fakesecret = rand() % 5000 + 4999;
    long fakesecret2 = rand() % 999999999999 + 1000000000000;
    fakesecesp3 = fakesecret * 16;
    sprintf(fakesecesp, "%d", fakesecret);

    sprintf(fakesecesp2, "%d", fakesecesp3);

    char fakesletter [10] = "OxO";
    strcat(fakesletter, fakesecesp);

    fputs("#define ", np2);
    fputs(fakesletter, np2);
    fputs(" ", np2);
    fprintf(np2, "%lu", fakesecret2);
    fputs(";", np2);
    while(i < secret)
    {
        fputs("\n", np2);
        break;
    }
}

void inttodefne(FILE *fp, FILE *np2, FILE *np3)
{
    char incl[128];

    while(fgets(incl, sizeof(incl), fp) != NULL)
    {
        puts("vzyal line");
        puts(incl);

        while ((int)strstr(incl, "int ") > strlen(incl) && (int)strstr(incl, "int main") < strlen(incl))
        {

            int position = strstr(incl, "int ") + 4;
            char septype[100] = { 0 }, sepexp[100], secesp[100], zeros[100] = { 0 };

```

```

    strncpy(septype, incl, position);
    strncpy(sepexp, (incl + position), 128);

    puts(septype);
    puts("EEEEEE");
    puts(sepexp);

    int secret = rand() % 5000 + 4999;
    sprintf(secesp, "%d", secret);
    char sletter [10] = "OxO";
    strcat(sletter, secesp);
    puts(sletter);

    defines(np2, incl, sepexp, sletter);

    fputs(septype, np3);
    fputs(sletter, np3);
    fputs(";", np3);

    break;
}

while ((int)strstr(incl, "for") > strlen(incl))
{

    int position = strcspn(incl, "for") + 3;
    char septype[100] = { 0 }, sepexp[100], secesp[100], zeros[100] = { 0 };

    strncpy(septype, incl, position);
    strncpy(sepexp, (incl + position), 128);

    puts(septype);
    puts("EEEEEE");
    puts(sepexp);

    int secret = rand() % 5000 + 4999;
    sprintf(secesp, "%d", secret);
    char sletter [10] = "OxO";

```

```

    strcat(sletter, secesp);
    puts(sletter);

    defines(np2, incl, sepexp, sletter);

    fputs(septype, np3);
    fputs(" ", np3);
    fputs(sletter, np3);
    fputs(" ", np3);

    break;

}

while ((int)strstr(incl, "int ") <= strlen(incl) && (int)strstr(incl, "for") <= strlen(incl))
{
    puts("nepadhodit");
    fputs(incl, np3);
    break;
}

while((int)strstr(incl, "int main") != 0)
{
    puts("nepadhodit");
    fputs(incl, np3);
    break;
}
}
}

void copylines(FILE *fp, FILE *np)
{
    char incl[100];
    while(fgets(incl, sizeof(incl), fp) != NULL)
    {
        fputs(incl, np);
    }
}

void skipincludes(FILE *fp, FILE *np)
{

```

```

char incl[100];
char clude[100];
char *ude = "#include ";

while(fgets(incl, sizeof(incl), fp) != NULL)
{
    if (strstr(incl, ude) > 0 && incl != NULL)
    {
        fputs(incl, np);

    }
    else
    {
        fputs(incl, np);
        break;
    }
}

}

int main(int c, char **arr)
{
    FILE *fp, *np, *np2, *np3, *np4, *tp;
    int ch, prev = 0;
    bool String = 0;
    long checkpoint;
    char incl[100];
    char *replace;

    char rename[100] = "obf.c";
    fp = fopen(arr[1], "r");
    strcat(arr[1], rename);
    np = fopen("temp1.c", "w+");
    np2 = fopen("temp2.c", "w+");
    np3 = fopen("temp3.c", "w+");
    np4 = fopen("temp4.c", "w+");
    tp = fopen(arr[1], "w");

    if (fp == NULL)

```

```

{
    printf("Invalid file!\n");
    return 1;
}

skipincludes(fp, np);

inttodefne(fp, np2, np3);

clearlines(np3, np4);

fseek(np, 0, SEEK_SET);
while(fgets(incl, sizeof(incl), np) != NULL)
{
    fputs(incl, tp);
}

fseek(np2, 0, SEEK_SET);
while(fgets(incl, sizeof(incl), np2) != NULL)
{
    fputs(incl, tp);
}
fputs("\n", tp);
fseek(np4, 0, SEEK_SET);
while(fgets(incl, sizeof(incl), np4) != NULL)
{
    fputs(incl, tp);
}

fclose(fp);
fclose(np);
fclose(np2);
fclose(np3);
fclose(np4);
fclose(tp);
remove("temp1.c");
remove("temp2.c");
remove("temp3.c");
remove("temp4.c");
return 0;
}

```

ДОДАТОК В
Слайди презентації

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЗАПОРІЗЬКА ПОЛІТЕХНІКА»
КАФЕДРА ПРОГРАМНИХ ЗАСОБІВ

ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДІВ ЗАХИСТУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ВІД РЕФАКТОРИНГУ

Виконав студент групи КНТ-223м
Керівник к.т.н., доцент кафедри програмних засобів

Пилип ГАНОШЕНКО
Тетяна ЗАЙКО

2024

Рисунок В.1 – Слайд 1

Об’єкт дослідження – процес розробки програмного забезпечення для захисту інтелектуальної власності у сфері розробки програмного забезпечення.

Предмет дослідження – програмні засоби для захисту програмного забезпечення від рефакторингу.

Мета роботи – дослідження методів захисту програмного забезпечення від рефакторингу та розробка програмного забезпечення для захисту програмного коду від дослідження та аналізу.

2

Рисунок В.2 – Слайд 2

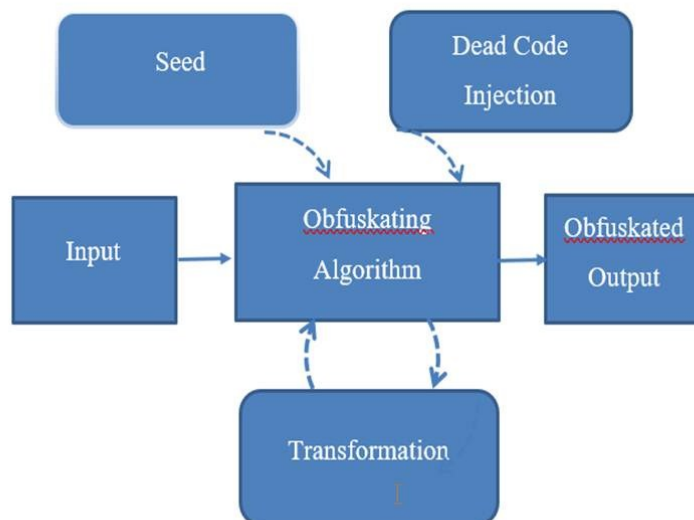
ЗАВДАННЯ ДО РОБОТИ

- ПРОВЕСТИ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ – З ПОЧАТКУ ВИНИКНЕННЯ ОБФУСКАЦІЇ ТА ЕВОЛЮЦІЮ АЛГОРИТМІВ;
- ПРОВЕСТИ АНАЛІЗ ПРОГРАМНИХ ЗАСОБІВ ЗАХИСТУ ПРОГРАМНОГО КОДУ ЗА ДОПОМОГОЮ АЛГОРИТМУ ОБФУСКАЦІЇ;
- ЗДІЙСНИТИ ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ АЛГОРИТМУ ОБФУСКАЦІЇ: ВИЗНАЧИТИ СТРУКТУРУ ТА ФУНКЦІЇ АЛГОРИТМУ, ЩО РОЗРОБЛЯЄТЬСЯ, ВИБРАТИ СЕРЕДОВИЩЕ ТА ІНСТРУМЕНТИ РОЗРОБКИ;
- РОЗРОБИТИ АЛГОРИТМ ОБФУСКАЦІЇ, ЙОГО ФУНКЦІЇ ТА МЕТОДИ ШИФРУВАННЯ, ПЕРЕВІРИТИ АЛГОРИТМ НА СТІЙКІСТЬ ДО СТАТИЧНОГО АНАЛІЗУ;
- СПРОЄКТУВАТИ СИСТЕМИ ВЗАЄМОДІЇ АЛГОРИТМУ З ФАЙЛАМИ ЧЕРЕЗ ІНТЕРФЕЙС КОМАНДНОЇ СТРОКИ;
- ВИКОНАТИ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.

3

Рисунок В.3 – Слайд 3

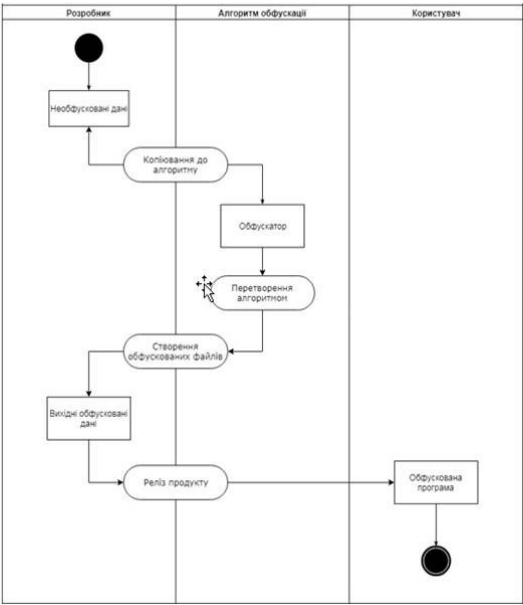
СХЕМА ОБФУСКАЦІЇ



4

Рисунок В.4 – Слайд 4

ДІАГРАМА СТАНІВ ОБФУСКАТОРА



5

Рисунок В.5 – Слайд 5

ПОРІВНЯННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ЗАХИСТУ ПРОГРАМНОГО КОДУ МЕТОДОМ ОБФУСКАЦІЇ

Критерій порівняння	Stunnix C Obfuscator	Tigress	Mangle-It
Розробник	Stunnix	The University Of Arizona (США)	PC Sentinel Software
Операційна система	Windows , Linux, Mac OS	Linux, Mac OS	Windows (98, 2000, XP)
Інтерфейс	Віконний	Command Line Interface	Віконний
Методи обфускації	Inline-макроси, хешування, рекурсія	Inline-макроси, відключення API, рекурсія,	Inline-макроси
Налаштування	Гнучкі, через користувацький інтерфейс	Гнучкі, через CLI	Відсутні
Вартість	\$500	Безкоштовно	\$70
Документація	Відсутня	Повна, за виключенням коду алгоритму	Відсутня

6

Рисунок В.6 – Слайд 6

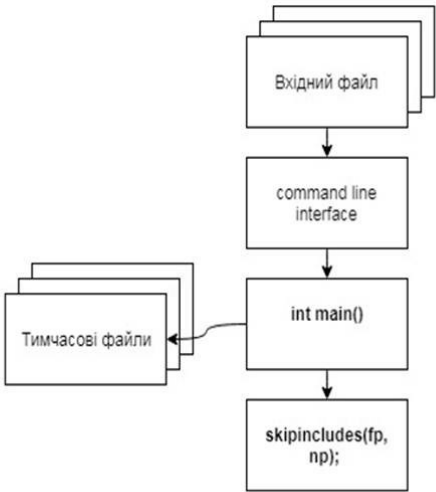
ВИБІР МОВИ ПРОГРАМУВАННЯ

IEEE 1. C 2. Java 3. Python 4. C++ 5. R 6. C# 7. PHP 8. JavaScript 9. Ruby 10. Go	TIOBE 1. Java 2. C 3. C++ 4. C# 5. Python 6. PHP 7. JavaScript 8. VB .NET 9. Perl 10. Assembly	Packt 1. JavaScript 2. Python 3. Java 4. C 5. SQL 6. PHP 7. HTML 8. CSS 9. R 10. Bash
PYPL 1. Java 2. Python 3. PHP 4. C# 5. JavaScript 6. C++ 7. C 8. Objective-C 9. R 10. Swift	Redmonk 1. JavaScript 2. Java 3. PHP 4. Python 5. C# 6. C++ 7. Ruby 8. CSS 9. C 10. Objective-C	GitHub 1. JavaScript 2. Java 3. Ruby 4. PHP 5. Python 6. CSS 7. C++ 8. C# 9. C 10. HTML

7

Рисунок В.7 – Слайд 7

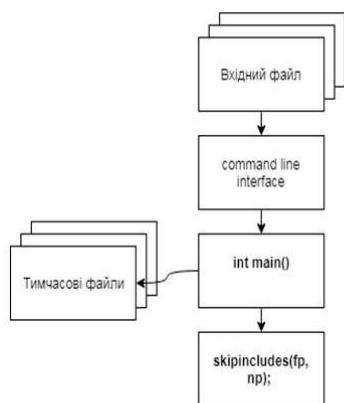
СТРУКТУРНА СХЕМА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



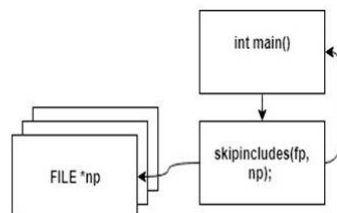
8

Рисунок В.8 – Слайд 8

Структурна схема функції main



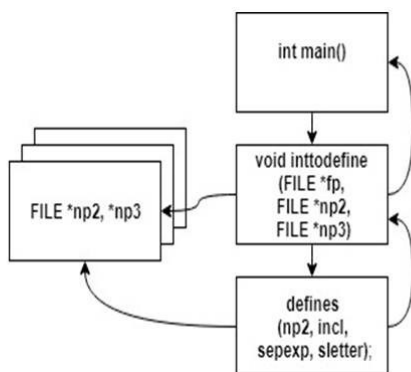
Структурна схема функції skipincludes



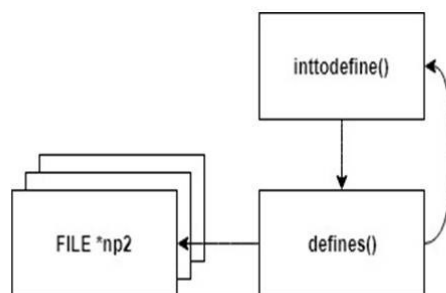
9

Рисунок В.9 – Слайд 9

Структурна схема функції inttodefine



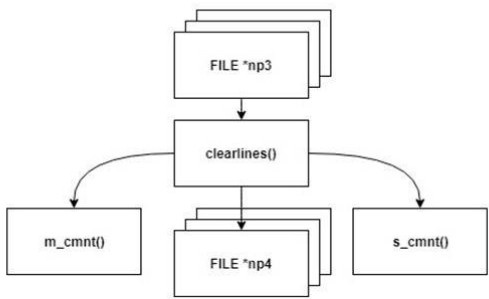
Структурна схема функції skipincludes



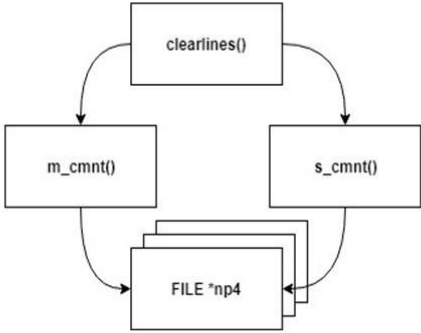
10

Рисунок В.10 – Слайд 10

Структурна схема функції clearlines



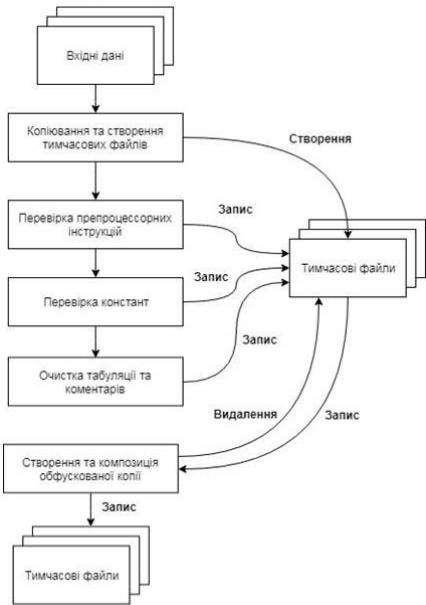
Структурна схема функцій s_cmnt() та m_cmnt()



11

Рисунок В.11 – Слайд 11

ФУНКЦІОНАЛЬНА СХЕМА ПРОГРАМИ ОБФУСКАЦІЇ

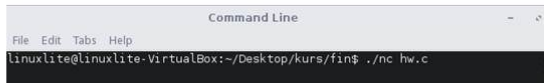


12

Рисунок В.12 – Слайд 12

ПРИКЛАД РОБОТИ З ПРОГРАМОЮ

Приклад виклику алгоритму обфускатора

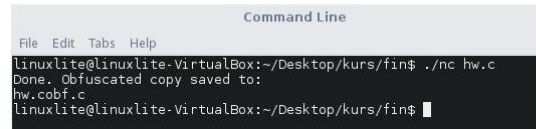


```

Command Line
File Edit Tabs Help
linuxlite@linuxlite-VirtualBox:~/Desktop/kurs/fin$ ./nc hw.c

```

Повідомлення про успішне виконання алгоритму

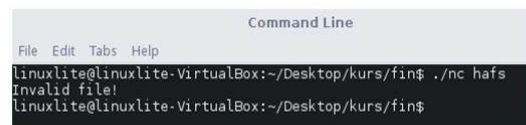


```

Command Line
File Edit Tabs Help
linuxlite@linuxlite-VirtualBox:~/Desktop/kurs/fin$ ./nc hw.c
Done. Obfuscated copy saved to:
hw.cobf.c
linuxlite@linuxlite-VirtualBox:~/Desktop/kurs/fin$

```

Повідомлення некоректного вхідного файлу



```

Command Line
File Edit Tabs Help
linuxlite@linuxlite-VirtualBox:~/Desktop/kurs/fin$ ./nc hafs
Invalid file!
linuxlite@linuxlite-VirtualBox:~/Desktop/kurs/fin$

```

13

Рисунок В.13 – Слайд 13

ВИСНОВКИ

В ході виконання дипломної кваліфікаційної роботи магістра було розроблено автоматизований алгоритм обфускації програмного коду.

За призначенням алгоритм спрямований допомогти розробникам програмного забезпечення захистити їх інтелектуальну власність, а саме програмний код. В якості ОС для роботи алгоритму обрано GNU/Linux.

Інтелектуальна власність має широке поширення завдяки Інтернету, нерідкі випадки розкрадання, модифікації і підміни походження власності. Програмне забезпечення, як форма інтелектуальної власності, досить часто представляє інтерес для шахраїв. Шахраї мають на меті розповсюдження чужого програмного коду, шукають помилки у вихідному коді для подальшої їх експлуатації та самозбагачення, тощо. Для запобігання шахрайським схемам, що мають на меті дослідження та розповсюдження чужого програмного коду існують автоматизовані алгоритми перетворення програмного коду у складний текст для розуміння як людиною так і комп'ютером без втрати функціональності, які називаються обфускацією.

Алгоритми обфускації призначені для захисту програмного вихідного коду від дослідження.

За результатами проведеного аналізу зроблено висновок, що існує багато програмних засобів для захисту програмного коду від рефакторинга методом обфускації.

Проте деякі існуючі програмні засоби можуть бути схильні до технічних збоїв, що може негативно вплинути на процес захисту.

Тому актуальною є розробка програмного забезпечення для захисту програмного коду від статичного аналізу та рефакторингу за допомогою обфускації. Сформульовано функціональні вимоги до програмного забезпечення.

14

Рисунок В.14 – Слайд 14