Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2022 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за другим (магістерським) рівнем вищої освіти
на тему
“Дослідження та програмна реалізація системи для протидії декомпіляції коду”

Виконав здобувач вищої освіти
ІІ курсу, групи КІ-21М-1,4
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Федоров Б.С.
«____» _____ 2022 р.

Керівник проекту
кандидат технічних наук, доцент
_____ Смірнов С.А.
«____» _____ 2022 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Рівень вищої освіти *магістр*

Галузь знань _12 _____*"Інформаційні технології"*_____

Спеціальність 123 *"Комп'ютерна інженерія"*

Освітньо-професійна (освітньо-наукова) програма _*"Комп'ютерна інженерія"*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 6 » вересня 2022 року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ДРУГИМ (МАГІСТЕРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Федорову Богдану Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Дослідження та програмна реалізація системи для протидії декомпіляції коду*

2. Керівник роботи *Смірнов Сергій Анатолійович, канд. техн. наук, доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 19-13 від 17.08.2022 року

3. Строк подання студентом роботи до захисту *10.12.2022 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою розробки є дослідження та програмна реалізація системи для протидії декомпіляції коду*

5. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Призначення та область використання.	*6. Наукова новизна.*
2.Перегляд аналогічних існуючих систем.	*7. Економічна ефективність розробленої програми.*
3. Опис і обґрунтування проектних рішень.	*8. Заходи з охорони праці та техніки безпеки.*
4. Етапи програмування системи.	*9. Висновки.*
5. Впровадження системи в промислову експлуатацію	

6. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Наукова новизна	*1 аркуш*
Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*
Показники економічної ефективності	*1 аркуш*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Савеленко Г.В.	05.10.2022	14.11.2022
Охорона праці	Оришака О.В.	06.10.2022	16.11.2022

7. Дата видачі завдання « 6 » вересня 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.10.2022 р.	
2.	Постановка задачі, оформлення ТЗ	15.10.2022 р.	
3.	Розробка моделі компонента	20.10.2022 р.	
4.	Розробка структур даних	25.10.2022 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.10.2022 р.	
6.	Програмування алгоритмів	10.11.2022 р.	
7.	Розрахунок економічної ефективності	13.11.2022 р.	
8.	Розрахунки з охорони праці та техніки безпеки	15.11.2022 р.	
9.	Оформлення ПЗ	17.11.2022 р.	
10.	Попередній захист роботи	10.12.2022 р.	

Дата видачі завдання
« 6 » вересня 2022 р.

Підпис керівника
_____ Смірнов С.А.
(прізвище та ініціали)

Завдання прийнято до виконання
« 6 » вересня 2022 р.

Підпис здобувача
_____ Федоров Б.С.
(прізвище та ініціали)

АНОТАЦІЯ

Федоров Б.С. Дослідження та програмна реалізація системи для протидії декомпіляції коду. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2022.

В даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи для протидії декомпіляції коду.

Метою розробки є дослідження та програмна реалізація системи для протидії декомпіляції коду.

Об'єктом дослідження є процес для протидії декомпіляції коду.

Предметом дослідження є методи для протидії декомпіляції коду.

Методи дослідження базуються на методах захисту інформації, методах математичної статистики, методах розробки програмного забезпечення.

Результат роботи – програмна реалізація системи для протидії декомпіляції коду.

В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на ПЕОМ архітектури IBM PC з ОС Windows 10/11.

Програму розроблено в середовищі RAD Studio Delphi 10.4.

Ключові слова: комп'ютерна інженерія, протидія декомпіляції коду

ABSTRACT

Fedorov B.S. Research and software implementation of a system to counteract code decompilation. 123 Computer engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2022.

In this graduation thesis for the second (master's) level of higher education, software is developed, which is intended for a system to counteract code decompilation.

The goal of the development is the research and software implementation of a system to counteract code decompilation.

The object of research is a process to counteract code decompilation.

The subject of the study is methods for countering code decompilation.

Research methods are based on information protection methods, mathematical statistics methods, and software development methods.

The result of the work is a software implementation of the system to counteract code decompilation.

In the process of working on the software model, an analysis of existing hardware and software was performed. All components of the developed software are fully described.

A convenient user interface has been developed. Instructions for working with software tools are provided.

The program can be used on PCs of IBM PC architecture with Windows 10/11 OS.

The program was developed in the RAD Studio Delphi 10.4 environment.

Keywords: computer engineering, code decompilation prevention

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

ПЗ	–	програмне забезпечення
ПП	–	програмній продукт

ВСТУП

Актуальність теми. Сучасний етап розвитку суспільства характеризується інтенсивним розвитком інформатизації. Як наслідок цього розвивається комп'ютерне піратство. Для протидії комп'ютерному піратству активно використовуються технології автоматичного захисту програмного забезпечення від аналізу й несанкціонованої модифікації. Ці технології широко використовуються в системах керування цифровими правами, а також незамінні при рішенні завдань приховання шкідливого коду.

Разом з тим існуючі методики розраховані або на наявність вихідного коду програми, що надзвичайно утрудняє рішення завдання контролю цілісності програми – або мають надзвичайно високу складність. Більше того, для більшості методик автоматичного захисту програм, що не вимагають наявності вихідного коду, існують механізми автоматичної деактивації захисту. Дана обставина визначає необхідність наявності адекватних методик протидії такого роду механізмам. Ці методики повинні бути автоматичними, не вимагати наявності вихідного коду й не повинні опиратися на недокументовані особливості платформи, на якій виконується захищена програма.

У цей час найпоширеніша методика автоматичного захисту програм від аналізу на основі технології віртуалізації машинного коду. Дана методика ставить у відповідність кожній машинній інструкції одну або кілька інструкцій автоматично згенерованого віртуального процесора, називаних байт-кодом або псевдокодом. У тіло програми, що захищається, вбудовується захищений від аналізу інтерпретатор, завданням якого є виконання згенерованого на етапі захисту байта-коду. Основним недоліком такого підходу є низька швидкість роботи захищеного в такий спосіб коду. Більше того, у більшості випадків все-таки можливе створення автоматичного декомпілятора псевдокоду у вихідний машинний код.

У зв'язку із цим для підвищення стійкості до автоматичних засобів деактивації захисту й забезпечення більш високої швидкодії захищеної програми в порівнянні з існуючими методиками необхідне створення нових підходів. В основі їх лежить принцип програмного «чорного ящика», реалізація якого припускає використання технологій заплутування коду й даних програми або обфускації

Обфускація або заплутування коду – приведення вихідного тексту вихідного виконуємого коду програми до виду, що зберігає її функціональність, але утрудняє аналіз, розуміння алгоритмів роботи й модифікацію при декомпіляції.

«Заплутування» коду може здійснюватися на рівні алгоритму, на рівні вихідного тексту, ассемблерного тексту. Для створення заплутаного ассемблерного тексту можуть використовуватися спеціалізовані компілятори, які використовують неочевидні або недокументовані можливості середовища виконання програми. Існують також спеціальні програми, що роблять обфускацію, називані обфускаторами

Дана обставина визначає актуальність розробки методик обфускації коду й даних програми, що не вимагають для своєї роботи вихідного коду програми й забезпечуючих стійкість стосовно автоматичних утиліт деактивації захисту.

Мета й завдання дослідження. Метою роботи є дослідження та програмна реалізація системи для протидії декомпіляції коду.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Огляд існуючих систем для протидії декомпіляції коду.
– Дослідження системи для протидії декомпіляції коду.
– Програмна реалізація системи для протидії декомпіляції коду.

Об’єктом дослідження є процес для протидії декомпіляції коду.

Предметом дослідження є методи для протидії декомпіляції коду.

Методи дослідження базуються на методах захисту інформації, методах математичної статистики, методах розробки програмного забезпечення.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

– Удосконалено метод для протидії декомпіляції коду.

– Розроблено вітчизняний продукт для протидії декомпіляції коду, який має більш широкі можливості, на відміну від існуючих аналогів.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі для протидії декомпіляції коду.

Достовірність наукових результатів підтверджена теоретичними викладеннями, даними комп'ютерного моделювання, коректними дослідженнями параметрів на функціонуючій обчислювальній мережі, а також відповідністю отриманих результатів окремим результатам, наведеним у науковій літературі.

Робота апробована на LVI Науково-технічній конференції здобувачів вищої освіти «Наука – виробництву», 2022, основні положення випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти надруковані у статті збірника праць молодих науковців ЦНТУ, випуск №13.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи для протидії декомпіляції коду, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Система призначена для захисту програмного забезпечення методом обфускації коду.

У більшості випадків для обходу захисту, взломщикові потрібно вивчити принцип роботи її коду, і те, як вона взаємодіє із самою програмой, що захищається, цей процес вивчення називається процесом реверсивної (зворотної) інженерії. Цей процес часто залежить від властивостей людської психіки, тому використання цих властивостей дозволяє знизити ефективність самого процесу реверсивної інженерії.

Обфускація, це один з методів захисту програмного коду, що дозволяє ускладнити процес реверсивної інженерії коду програмного продукту (ПП), що захищається.

Обфускація може застосовуватися не тільки для захисту ПП, вона має більше широке застосування, наприклад вона, може бути використана творцями вірусів, для захисту їхніх створінь і т.д.

Суть процесу обфускації полягає в тому, щоб заплутати програмний код і усунути більшість логічних зв'язків у ньому, тобто трансформувати його так, щоб він був дуже важкий для вивчення й модифікації сторонніми особами (будь то взломщики, або програмісти які збираються довідатися унікальний алгоритм роботи программы, що захищається).

Із цього слідує, що обфускація одна не призначена для забезпечення найбільш повного й ефективного захисту програмних продуктів, так як вона не надає можливості запобігання нелегального використання програмного продукту. Тому обфускацію звичайно використовують разом з одним з існуючих методів

захисту (шифрування програмного коду й т.д.), це дозволяє значно підвищити рівень захисту ПП у цілому.

Процес обфускації як метод захисту, можна вважати порівняно новим (перші статті, присвячені обфускації, як методу захисту коду програмних продуктів, з'явилися приблизно три-чотири роки тому), і перспективним.

Обфускація відповідає принципу економічної доцільності, так як її використання не сильно, збільшує вартість програмного продукту, і дозволяє при цьому знизити втрати від піратства, і зменшити можливість плагіату в результаті крадіжки унікального алгоритму роботи програмного продукту, що захищається.

1.2 Область застосування

Обфускація програм може мати численні додатки в криптографії: її можна використовувати для перетворення криптосистем із секретним ключем у криптосистеми з відкритим ключем [2-4], для проведення обчислень над зашифрованими даними (гомоморфних обчислень) [2,5], для заміщення моделі випадкового оракула в криптографічних протоколах [2,5], для забезпечення безпеки в ієрархічних системах розподіленого доступу [5], для створення верифікуємих систем таємного голосування [6] і ін.. У всіх перерахованих криптографічних додатках від обфускації програм потрібно настільки ж висока стійкість, як і від традиційних криптографічних систем, використовуваних для рішення подібних завдань. Уперше строге математичне визначення стійкості обфускації програм було запропоновано в роботі [2]: обфускація вважається стійкою, якщо всякий супротивник може витягти з тексту обфускованої програми за розумне (поліноміальне) час не більше інформації, ніж можна було б одержати, проводячи тестові випробування цієї програми як «чорного ящика». У цій же роботі було встановлене існування таких програм, для яких подібна стійкість обфускації в принципі не досяжна. Згодом у ряді робіт [3,4] були запропоновані й інші, менш вимогливі визначення стійкості обфускації, однак, і

для них була показана неможливість побудови ефективного транслятора, що гарантує стійку обфускацію довільних програм. Разом з тим, у роботах [4,5,8] було показано, що для окремих класів функцій (т.зв. «точкових функцій») стійка обфускація їхніх програм, що обчислюють, можлива при тих або інших стандартних криптографічних припущеннях. Найбільш значне просування було досягнуто в роботі [1]; її автори довели реалізуємість стійкої обфускації програм перешифрування повідомлень. У цілому, однак, результати досліджень у цьому напрямку не дають більших підстав для оптимізму: дотепер не вдалося виявити досить складний криптографічної функції, програми обчислення якої допускають доказово стійку обфускацію.

Застосування обфускації не обмежується криптографією; інший напрямок досліджень проблеми обфускації програм пов'язане з її застосуванням в області комп'ютерної безпеки. Обфускуючі перетворення можуть бути використані для захисту інтелектуальної власності на програмне забезпечення [1, 2], для інформаційного захисту мобільних агентів [3] і мікроелектронних схем на етапі проектування [4], для забезпечення приватности інформаційного пошуку [5]. На жаль, виявилося, що техніка обфускації програм може бути також використана й у зловмисних цілях для розробки важко виявляємих «вірусів» [6,7]. Перспективи успішної розробки й застосування обфускуючих перетворень у цьому напрямку представляються більше певними: для багатьох комерційних додатків проведення обфускації програм стає доцільним, якщо витрати, які змушений понести супротивник для подолання наслідків обфускації, істотно перевершують витрати, викликані застосуванням обфускації. Так, наприклад, значне зниження продуктивності обчислювальної системи, обумовлене тривалою роботою антивірусного сканера в спробі виявити потайливий метаморфний «вірус», може бути прирівняне до збитку, заподіяному самим «вірусом».

Методи обфускації програм, застосовувані для забезпечення інформаційної безпеки програмного забезпечення, можна умовно класифікувати по трьох основних характеристиках залежно від того:

– чи допускається використання високонадійного обчислювального пристрою, недоступного для супротивника, або доступ до обфускованої програми вважається необмеженим;

– у якій формі програма передбачається доступною для супротивника (у вигляді вихідного тексту мовою високого рівня або у вигляді коду, що виконується,);

– чи передбачається, що код програми повинен залишатися незмінним протягом її виконання.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи для протидії декомпіляції коду, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти

Призначення обфускатора

Обфускатор аналізує метадані, так як не всі члени складання він може обфусцирувати. Наприклад обфускатору не варто заміняти імена конструкторів типу – це може бути чревате. Хоча в деяких випадках це буває можливо. Також іноді неможлива обфускація віртуальних або абстрактних методів.

Список членів складання для обфускації готовий. Обфускатор привласнює їм нові імена, базуючись на певному алгоритмі. Одні обфускатори привласнюють імена такої ж довжини що й були але позбавлені колишнього змісту, інші базуються на нумерації всіх членів і обфускують та відповідності номеру члена складання, треті – базуючись на токен (token) члена складання – унікальний ідентифікатор члена складання в MSIL, четверті намагаються мінімізувати довжину ім'я й частіше використовувати те саме ім'я серед членів складання (в dotfuscator це називається overload induction) – це дає максимальний ефект обфускації (уявіть собі що у вашого класу всі методи й поля названі як «1»).

Після цього відбувається запис даних назад у складання або генерація нового складання – все це в руках авторів обфускаторів, її оптимізація – так як багато хто з обфускаторів здатні видалити непотрібну інформацію зі складання (дебаг-інформація, невикористовувані методи, поля, класи). Складання готове. Фактично складання після обфускації відрізняється від вихідною однією деталлю – модифікованим розділом рядків (String Heap або #Strings) – одним з п'яти розділів метаданих, це підсумок символьної обфускації, коли обфускатор не модифікує тіла методів. Може бути модифікований ще один розділ (User Strings

або #US) – але поки не зустрічав таких обфускаторів. Відкриваємо в ildasm або Reflector або в іншому навігаторі по складаннях і оцінюємо результат. Деякі обфускатори мають додаткові настроювання: заплутування namespaces (зміна приналежності різних класів певним namespace), шифрування строкових і графічних ресурсів, контроль обфускації на базі спеціальних атрибутів, якими позначаються члени класів у вихідному коді, деякі намагаються загравати з MSIL-кодом методів складання.

Достоїнства**:**

– Обфускатори роблять дизассембльований код важким для вивчення, перетворюючи IsLicensed() в x().

– Деякі обфускатори використовують баги ILDASM для захисту від дизассемблінгу в ньому (Salamander).

– Деякі обфускатори навіть конвертують код в native код, роблячи марним дизассемблінг (Salamander).

– Деякі обфускатори шифрують і пакують exe referenced складання в один exe-файл, так що розмір проги може зменшитися 2-4 рази й не піддається дизассемблінгу (Thinstall).

Недоліки:

– Продукт залишається дизасемблюємим.

– Зібрати складання після дизассемблінгу не важко буде.

– IL код – доступний для читання й розуміння в порівнянні с ассемблерним.

– «Захист» обфускаторів, які використовують баги ILDASM будуть неспроможні перед дизассемблерами інших розроблювачів.

– Захист обфускаторів, які використовують шифрування символьної частини метаданих, строкових і бінарних ресурсів заважає користувачам продукту, налагоджувати й тестувати свої продукти. Крім цього – це ризик, так як деякі символьні дані використовуються в Reflection – для одержання типу

(GetType(“MyType”)), або завантаження ресурсу (GetManifestResourceStream(“MyResource”)).

– Найчастіше обфускатори мають купу настроювань, незрозумілих або складних для розуміння звичайному користувачу. Неінформованість користувача може привести до так як його обфускована програма буде працювати не так як хотілося й іноді приводити до помилки, так ще й до такої, що не відловлюється видладником.

– Ціна – порядок цін – від 40 до 1500 доларів за програму. І ціна може нічого не говорити про якість обфускації. Може так статися що 40-доларовий обфускатор захистить ваш куди краще чим більш дорогий.

– Щодо 4-його достоїнства. Захищена в такий спосіб програма «жорстко прив'язана» до використовуваного .Net Framework, і сервіс-пак установлений вами, «порушить» коректність роботи захищеної програми. Та й такий захист можливий тільки для Intel-процесорів.

– Щодо 5-го достоїнства. Так дійсно захист сильний і позбавлена недоліків 4-го достоїнства. Продукт завантажує з ресурсу потрібні складання до памяті й managed exe файл і передає йому керування, займаючись тільки рішенням проблем зі складаннями, типами, ресурсами (через AssemblyResolve, TypeResolve, ResourceResolve). Але – не всі складання потрібні відразу, завантаження їх вимагає дешифрування й розпакування – додаткового часу й навантаження на процесор. Не всі роблять exe-файли. А Thinstall буде працювати тільки з exe, так як dll-складання вже не мають як раніше процесорного DllMain, з якого це було можливо робити. Але – зламати її проблем також не становить праці. Є така програма ProcDump – вона може продампить запущений процес і відповідно легко одержати розшифрованими й розпакованими захищений exe і referenced складання. Thinstall буде мати проблеми із завантаженням на згадку managed C++ складання.

– Деякі обфускатори створюють замкнуту систему обфускованих складань, де необфускованими залишаються складання сторонніх виробників і

MS-Вские. Але й тут все далеко від досконалості – обфускатор не знає як використовуються ресурси складання, тому може привести вашу програму до неробочого стану. Хоча в цьому випадку якість обфускації стає ідеальним – ні до чого не підкопатися. Знову ж – це можливо тільки для exe-програм. Якщо вам необхідно захищати бібліотеку класів, то вам необхідно буде залишати необфускованими публічні члени класів і публічні класи – звідси крэкер може почати свою роботу.

Огляд обфускаторів і їхніх можливостей

RemoteSoft Slamander Obfuscator

Має навігатор, що дозволяє переміщуватися не тільки по метаданим, але й за структурою PE-файлу, аналізувати ресурси, бінарні дані. Багаті можливості для аналізу метаданих. Немає аналізу callgraph методів. Є свій дизассемблер, але дизасемблює відразу весь клас, не менше – доводиться чекати. Може використовуватися як редактор вихідних текстів. Непоганий символьний обфускатор, декларуємий захист від дизассемблера не працює – у мене спокійно відкривався метод, обфускований ім. Не працює із проектами й з наборами складань. Навігатор (explorer) має плагінну архітектуру що дозволяє підключати модулі від Salamander до нього. Застосовують overload induction – тобто максимально короткі й часто застосовувані назви для членів складання. Також пропонують Protector – який перетворює ваші методи в native embedded code – не дизасемблюються, але залежні від поточної версії .Net Framework. Декларуєма оптимізація складань не була зроблена. Написаний в native code.

Preemptive Solutions Dotfuscator

Community Edition цього обфускатора поширюється з VS .Net 2003. Бідний інтерфейс обфускує як і всі, настроювання важкі, але інтегрований в VS як addin. Працює з тілами методів заплутуючи control flow, тобто саме всілякі for і while. . Працює із проектами через інтеграцію зі студією. Можливо інші версії його мають більшу кількість функцій – але Community Edition мені не сподобалася.

9Rays.Net ILObfuscator

Нова версія 2.5 разюче відрізняється від попередньої 2.0. Дуже багато якісних змін. По-перше з'явився GUI. Що багатьом полегшує завдання роботи зі складаннями. ILOGUI – це й навігатор по складаннях, дизассемблер, система роботи із проектом обфускації, у ній можна переглядати й екстрагувати ресурси, що втримуються в складанні. Чим відрізняється від інших? Managed code – C# и managed C++, робота з набором складань, а значить і крос-обфускація, що дозволяє досягти максимального рівня обфускації, непоганим і зрозумілий GUI с VS IDE-like MDI інтерфейсом, з ILO поставляється SDK, що дозволяє вам створити свою власну систему обфускації, зручна робота із властивостями проекту, є колекції замін і виключень, гнучкі можливості по заплутуванню – скажемо додаток по заплутуванню namespaces (можна вибрати один з варіантів),также пропонуються на вибір варіанти найменування обфускованих класів. Є функція оптимізації складання після обфускації, правда тільки для managed складань, без embedded native code. ILO самообфускован – ви можете переконатися в якості його обфускації. Із заявлених на наступну версію налаштувань – підтримка solutions – тобто наборів проектів, генерація карт обфускації й імпорт їх при обфускації проектів, ці фичи вже дозволять фіксувати результат обфускації й використовувати його в наступної обфускації проекту й зменшувати час тестування складання після обфускації й максимально обфускувати проект. Крім цього обіцяно функцію оптимізації доповнити додатками видалення невикористовуваних членів складань і вставити функцію аналізу проекту для виявлення їх, доповнити навігацію закладками й history.

Lesser Software LSW IL-Obfuscator

Проста й швидка програма, що для багатьох підійде. Небагато настроювань, все понятно й просто. Бере як IL-файли, так і скомпільовані складання. Правда не все – іноді видає якусь дивну помилку й скопилированную складання не піднімає. Таке враження що він її зпочатку дизасемблює. Не працює

з наборами складань, але підтримує заміни й виключення з обфускації. Native code. Але за $40.

Thinstall

Тепер уже теж від Lesser Software – ця штуковина саме працює тільки з exe-файлами. Створює самоустановлювальний exe-файл, з усіма файлами на борті, закриваючи від дизасемблювання складання. Такий файл не без зусиль але розкривається, не варто обманюватися. Як зізнаються самі виробники – найкраще використовувати разом з обфускатором.

Wise Owl Demeanor

Багато чого може – і видаляє непотрібну інформацію з метаданих, шифрує строкові змінні, працює з мультимодульними складаннями (рідко використовувана річ) і інтегрується з VS.Net. Але коштує $1250.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Embarcadero Delphi, раніше Borland Delphi і Codegear Delphi, – інтегроване середовище розробки ПЗ для Microsoft Windows, Mac OS, iOS і Android мовою Delphi (що раніше носила назву Object Pascal), створена спочатку фірмою Borland і на даний момент приналежна й розроблювальна Embarcadero Technologies. Embarcadero Delphi є частиною пакета Embarcadero RAD Studio і поставляється в чотирьох редакціях: Community (поширюється безкоштовно й має обмежену ліцензію на використання в комерційних цілях), Professional, Enterprise і Architect.

Delphi 10.4 Sydney

Випущено 26 травня 2020 року. RAD Studio Delphi 10.4 забезпечує значно поліпшену високопродуктивну нативну підтримку Windows, кращу продуктивність розробки, миттєві підказки code completion, прискорення виконання коду із синтаксисом керованих записів, поліпшення виконання паралельних завдань на сучасних багатоядерних CPU, а також містить більш 1000

виправлень багів, поліпшення продуктивності середовища й бібліотек і багато чого крім того.

Основні можливості Delphi 10.4.1:

− Істотні розширення для Windows: поліпшення для застосунків на моніторах 4K High DPI, інтеграція з новим WebView2 на базі Chromium, використання розширених title bars, таких же, як в Office, Explorer, Google Chrome.

− Керування пам'яттю в Delphi тепер стандартизоване на всіх підтримуваних платформах – мобільних, настільних і серверних – використовучи класичну реалізацію керування пам'яттю об'єктів.

− Істотне поліпшення Delphi Code Insight (без можливого блокування IDE – в окремому процесі), що допоможе при роботі з великими проектами.

− Тип даних Delphi «record» тепер підтримуть довільні ініціалізацію, фіналізацію й операції копіювання.

− Розширена підтримка бібліотек C++: ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode.

− Відладник Win 64 (на LLDB) і збирач для C++.

− Поліпшення для C++: Включена велика кількість поліпшень STL з Dinkumware.

− Підтримка Metal Driver GPU для macOS і iOS.

− Вбудований Fmxlinux.

− Компонент Twebbrowser для iOS тепер реалізований на Wkwebview API. Реалізація компонента Media Player для macOS тепер використовує Avfoundation. Реалізований заново стилізуємий FMX компонент TMemo на платформі Windows значно поліпшений і тепер має відмінну підтримку IME.

− Численні поліпшення швидкості й стабільності роботи нашої бібліотеки The Parallel Programming Library (PPL).

− Додані оновлені драйвери для FireBird, PostgreSQL і SQLite.

− Клієнтські бібліотеки HTTP і REST Client розширені застосунковими можливостями роботи з HTTPS. Також були розширені можливості підтримки Amazon AWS services

− У технологію Visual LiveBindings внесена безліч поліпшень, у тому числі швидкодії, що стосуться, застосунків на VCL і FireMonkey

RAD Studio 10.4 Короткий огляд:

– Істотні розширення для Windows. Створення застосунків, що чудово виглядають, із чіткими елементами інтерфейсу на 4k моніторах High DPI за допомогою нової гнучкої підтримки стилів елементів керування на екрані. Інтеграція із сучасними, безпечними web-технологіями від Microsoft – новим WebView2 на базі Chromium. Використання сучасних розширених title bars, таких же, як в Office, Explorer, Google Chrome, у своїх проектах. Істотні поліпшення надійності налагодження в новому відладнику для C++ Windows 64-bit.

– Зросла продуктивність розробки. Ріст продуктивності за рахунок миттєвої реакції підказок code completion у середовищі IDE. Краща сумісність із уже наявною кодовою базою, і спрощення програмування за рахунок уніфікованої архітектури керування пам'яттю. Швидке зв'язування даних і візуальних елементів за допомогою розширеної технології Visual LiveBindings з підвищеною швидкодією. Просте використання розповсюджених бібліотек C++, наприклад, ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode. Оновлена підтримка Amazon AWS cloud.

– Поліпшення швидкодії і якості. Більш 1000 поліпшень швидкодії і якості. Краща ефективність коду за допомогою нового синтаксису custom managed records. Більш швидке виконання паралельних завдань на сучасних багатоядерних CPU. Переконаєтеся в прискоренні відображення на екрані з підтримкою Metal API на macOS і iOS. Краща сумісність із уже наявною кодовою базою й спрощення програмування за рахунок уніфікованої архітектури керування пам'яттю.

Істотне поліпшення Delphi Code Insight

Як найбільше й головне поліпшення інструментів програмування Delphi за багато років, в 10.4 Delphi Code Insight реалізований через Language Server Protocol (LSP). LSP – це технологія генерації результатів для code completion, навігації й інших сервісів в окремому процесі. Це значить, що code completion і Code Insight одержать більш точні результати без блокування IDE. 10.4 забезпечує набагато більш високу продуктивність розроблювачів, які працюють із більшими проектами, що містять мільйони рядків коду.

Delphi Custom Managed Records

Ключове розширення мови Delphi: тип даних Delphi «record» тепер підтримуть довільні ініціалізацію, фіналізацію й операції копіювання. Управляйте тем, як ці структури створюються, копіюються й звільняються з допомогу вашого коду, який буде виконуватися у відповідний момент.

Це розширює потужність конструкцій records в Delphi, які використовуться щоб одержати більшу ефективність у порівнянні із класами.

Єдине керування пам'яттю

Керування пам'яттю в Delphi тепер стандартизоване на всіх підтримуваних платформах – мобільних, настільних і серверних – використовучи класичну реалізацію керування пам'яттю об'єктів.

У порівнянні з Automatic Reference Counting (ARC), це дає кращу сумісність із існучим кодом і спрощує написання компонентів, бібліотек і застосунків.

ARC модель керування пам'яттю model залишилася для керування рядками й посиланнями на тип інтерфейсу на всіх платформах. Для C++ це означає, що при створенні й звільненні Delphi-style класів в C++ використовується звичайне керування пам'яттю, як у будь-якого heap-allocated класу C++, що значно знижує складність коду.

Розширена підтримка бібліотек C++

В 10.4 ми портували багато популярних бібліотек C++ у C++Builder.

Забезпечивши оптимізовану підтримку бібліотек ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode, поряд із уже підтримуваними Boost і Eigen, які можуть бути додані за допомогою менеджера пакетів Getit.

Win 64-відладник і збирач для C++

В 10.4 з'явився новий відладник C++ для Windows 64-bit. Відладник заснований на LLDB і показує значне збільшення стабільності при налагодженні 64-bit застосунків поряд з новими відладочними можливостями, такими як перегляд і інспекція типів начебто рядків C++ і Delphi, а також колекцій STL, включаючи std::vector, std::map і інших. Крім того, згенерована для застосунку відладочна інформація має інший внутрішній формат, сприяючи більш стабільному й багатому на можливості процесу налагодження, більш докладним перегляду й інспекції в debug-time.

Підвищення якості й швидкодії інструментів

- Велика кількість поліпшень STL від Dinkumware.
- Поліпшені деякі найважливіші методи й області RTL, на базі поліпшень сумісності з популярними бібліотеками C++.
- Поліпшена підтримка Cmake.
- Велика кількість виправлень для підвищення стабільності і якості.
- Відновлення Windows API – Обновлено й додали безліч декларацій API щоб добитися ще більшої інтеграції із платформою Windows.
- Загальні вдосконалення в бібліотеці доступу до БД FireDAC, включаючи оновлені драйвера для FireBird, PostgreSQL і SQLite. Вибір статичного або динамічного підключення SQLite до застосунку.

Змінені стилі VCL для High DPI

В 10.4, архітектура стилізації VCL була суттєво розширена для підтримки High DPI і 4K моніторів. Тепер усі елементи UI на формі VCL автоматично масштабуться під відповідне до монітора дозвіл для показу форми. Був оновлений API стилізації для підтримки стилів high DPI.

Кожний графічний елемент UI може бути обраний з наборів різних масштабів і масштабований до потрібного DPI, що дає чітке зображення елементів UI на всіх моніторах.

Нові High DPI стилі й стилізація окремих VCL компонент

Обновлено велике число вбудованих і преміальних VCL стилів для підтримки нового режиму стилізації High-dpi. Це дозволяє вам створювати застосунку з відмінним дизайном для всіх моніторів.

Розроблювачі VCL застосунків тепер можуть використовувати трохи VCL стилів на різних формах в одному застосунку або в різних компонентів на одній формі. Це також включає стилізацію компонентів загальною темою для платформи. Крім застосункової гнучкості використання стилів, це дозволяє використовувати нестилізуємі компоненти із зовнішніх бібліотек в VCL застосунках, що використовуть стиль.

Поліпшена кроссплатформеність

– Додана підтримка Metal Driver GPU для macOS і iOS.

– Крім підтримки останнього iOS SDK, в RAD Studio 10.4 розроблювачі можуть задовольнити нові вимоги Apple до набору стартових екранів.

– Реалізований заново стилізуємий FMX компонент TMemo на платформі Windows значно поліпшений і тепер має відмінну підтримку IME.

– Користувачам редакцій Enterprise або Architect доступна повна інтеграція Fmxlinux з IDE для створення клієнтських застосунків Linux з GUI.

– Компонент Twebbrowser для iOS тепер реалізований на Wkwebview API.

– Реалізація компонента Media Player для macOS тепер використовує Avfoundation.

Оновлений менеджер пакетів Getit

Менеджер пакетів Getit в IDE був значно вдосконалений.

Дати випуску релізів пакетів тепер видні, і можливе сортування списку по цих датах; відбір тільки встановлених пакетів, контенту, доступного тільки при наявності підписки, багато чого іншого.

Універсальний інсталятор для установки Online і Offline

В 10.4 включений новий універсальний інсталятор, який використовує технологію на базі Getit. Цей інсталятор підтримує як online, так і offline (з ISO) варіанти установки.

Тепер обоє варіанта установки дозволяють вам указати початковий набір можливостей RAD Studio для установки, наприклад, свою комбінацію мов програмування й цільових платформ, мов інтерфейсу, і додавати до нього або видаляти непотрібне в будь-який момент.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи для протидії декомпіляції коду.

В процесі розробки випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти необхідно виконати наступний обсяг роботи:

а) провести аналіз існуючих систем-аналогів для виявлення їх позитивних і негативних якостей. Результати аналізу врахувати в подальших розробках;

б) вибрати та обґрунтувати методику побудови системи контролю роботи технологічного обладнання на виробництві в автоматизованому режимі. Розробити функціональну та структурну схеми системи;

в) розробити програмне забезпечення системи, що дозволить реалізувати поставлену технічним завданням задачу. Побудувати блок-схеми алгоритмів програми та підпрограми;

г) організувати інтерфейс користувача з метою формування та виводу на екран ЕОМ повідомлень про некоректні дії користувача та нестандартні ситуації в роботі технологічного обладнання;

д) розробити рекомендації по організаційних та методичних заходах, які забезпечать впровадження системи в промислову експлуатацію та її подальшу успішну експлуатацію;

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

ж) розробити заходи по охороні праці при впровадженні та експлуатації системи, а також розробити заходи з цивільного захисту;

з) сформувати висновки про виконаний обсяг робіт та одержані результати.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Опис технології обфускації

Існують різні визначення процесу обфускації. Розглядаючи даний процес із погляду захисту програмного продукту (ПП), і трансформації коду програми без можливості в наслідку повернутися до його первісного виду (трансформація "в одну сторону"), можна дати таке визначення: нехай "TR" буде процесом, що трансформує, тоді при "PR1 =TR=> PR2" програма "PR2" буде являти собою трансформований код програми "PR1". Процес трансформації "TR" буде вважатися процесом обфускації якщо, будуть задоволені такі вимоги:

– Код програми "PR2" у результаті трансформації буде істотно відрізнятися від коду програми "PR1", але при цьому він буде виконувати ті ж функції що й код програми "PR1", а також буде працездатним.

– Вивчення принципу роботи, тобто процес реверсивної інженерії, програми "PR2" буде більше складним, трудомістким, і буде займати більше часу, чим програми "PR1".

– При кожному процесі трансформації того самого коду програми "PR1", код програм "PR2" будуть різні.

– Створення програми детрансфомуючої програму "PR2" у її найбільш схожий первісний вид, буде неефективно.

Так як код, одержуваний після здійснення обфускації, над однією й тією же програмою, різний то процес обфускації можна використовувати для швидкої локалізації порушників авторських прав (тобто тих покупців, які будуть займатися нелегальним поширенням куплених копій програм). Для цього визначають контрольну суму кожної копії програми яка пройшла обфускацію, і записують її разом з інформацією про покупця, у відповідну базу даних. Після

цього для визначення порушника, досить буде, визначивши контрольну суму нелегальної копії програми, зіставити її з інформацією, що зберігається в базі даних.

Програмний код може бути представлений у двійковому виді (послідовність байтів представляючих собою так званий машинний код, що виходить після компіляції вихідного коду програми) або вихідному виді (текст утримуючу послідовність інструкцій якоїсь мови програмування, що зрозумілий людині, цей текст у наслідку буде підданий компіляції або інтерпретації на комп'ютері користувача).

Далі в тексті вживаються такі поняття: Вихідна програма – програма, яка захищається, що піддається процесу обфускації. Зловмисник – особистість, що займається вивченням алгоритму роботи вихідної програми (реверсивною інженерією), для будь яких своїх корисливих цілей. Об'єкт – ім'я якогось сховища даних, наприклад змінної, масиву й т.д. Деобфускація – процес, що дозволяє обійти обфускацію, використовується зловмисниками (деобфускація описана в окремому розділі даного матеріалу).

Оцінка процесу обфускації

Існує багато методів визначення ефективності застосування того або іншого процесу обфускації, до конкретного програмного коду.

Ці методи прийнятий розділяти на дві групи: аналітичні й емпіричні. Аналітичні методи ґрунтуються на трьох величинах характеризуючі наскільки ефективний той або інший процес обфускації:

– Стійкість – указує на ступінь складності здійснення реверсивної інженерії над кодом минулий процес обфускації.

– Еластичність – указує на те наскільки добре даний процес обфускації, захистить програмний код від застосування деобфускаторів.

– Вартість перетворення – дозволяє оцінити, наскільки більше потрібно системних ресурсів для виконання коду минулий процес обфускації, ніж для виконання оригінального коду програми.

Їх найбільше ефективно застосовувати при порівнянні різних алгоритмів обфускації, але при цьому вони не можуть дати абсолютної відповіді на питання наскільки ефективне застосування того або іншого алгоритму, саме до даного програмного коду.

Емпіричні ж методи можуть дати прийнятна відповідь на таке питання, так як вони ґрунтуються на статистичних даних одержувані в результаті досліджень. Для проведення одного з таких досліджень потрібна група людей (якнайкраще знайомих, з реверсивною інженерією), фрагмент коду програми, яка захищається, і набір різних алгоритмів обфускації.

Результати такого дослідження будуть містити в собі мінімальну кількість часу, що треба було групі людей, для того щоб вивчити кожний фрагмент коду програми над якою реалізувався один з алгоритмів обфускації.

Процес деобфускації

Коли ми говоримо про процес обфускації, з'являється питання: є чи процес зворотний йому, що дозволив би зловмисникові повернути найбільш схожий первісний код програми, тобто код до обфускації? На це питання важко дати однозначну відповідь, але такий процес існує й носить він назва деобфускації. Але інше не менш важливе питання, це як його можна реалізувати.

З однієї сторони до процесу деобфускації можна віднести процес оптимізації програмного коду, так як вони обоє, у тому або іншому ступені, протилежні процесу обфускації. У процесі обфускації в програмний код часто вироблятися додавання зайвих операцій, вони звичайно не яким образом не впливають на результати роботи самої програми, і призначені для збиття з пантелику й ускладнення процесу вивчення коду програми потойбічними (BEYOND) особами.

У свою чергу процес оптимізації програмного коду спрямований на ліквідацію зайвих операцій, тому в окремих випадках він може виступати як квінтесенція процесу деобфускації.

Слід зазначити, що більшість компіляторів у процесі компіляції вихідного коду, автоматично здійснюють процес оптимізації, тому якщо обфускація здійснюється над вихідним кодом програми (обфускація високого рівня), виникає певна ймовірність, того, що її ефективність після, компіляції знизитися. Якщо ж такий вихідний код буде оброблятися інтерпретатором (тобто не буде підданий компіляції), ефективність здійсненого процесу обфускації, не змінитися.

До процесу деобфускації, також можна віднести й процес декомпіляції, що дозволяє, маючи двійковий код програми одержати найбільш схоже вихідне подання цього коду мовою високого рівня, що більше зрозумілий людині, це дозволить спростити процес реверсивної інженерії. (Слід зазначити, що здійснення обфускації на нижчому рівні, дозволяє найбільше повно ускладнити можливий процес декомпіляції програмного коду.)

На сьогоднішній день існує багато матеріалу дотичного як процесу оптимізації, так і процесу декомпіляції, тому він може бути використаний для початкового вивчення процесу деобфускації.

Нижче представлений простий зразок класифікації методів процесу деобфускації:

– Знаходження й оцінка непрозорих конструкцій (предикатів), статичний аналіз, яких дуже складний.

– Зіставлення зі зразком. Здійснюється різними способами, найпоширеніші два з них. Перший, це коли береться декілька тих самих програм, що пройшли процес обфускації (так як процес обфускації в більшості випадків унікальний, то їхній код також буде різний, хоча вони й будуть виконувати ідентичні дії), і вироблятися порівняння фрагментів їхнього коду, для виявлення вставленого в процесі здійснення обфускації зайвого коду, що у наслідку просто убирається. Другий спосіб зіставлення зі зразком, здійснюється шляхом пошуку в коді програми найпоширеніших конструкцій, застосовуваних у процесі обфускації. Такі конструкції можуть, наприклад, зберігатися й обновлятися у відповідній базі даних, або бути отримані шляхом вивчення роботи самого обфускатора.

– Виділення в програмі фрагментів коду, які жодним чином не зв’язані з основними завданнями, які повинна виконувати програма, тобто виявлення непотрібних (зайвих) ділянок коду.

– Статистичний аналіз, полягає в динамічному аналізі коду програми. Наприклад, знаходження непрозорих предикат може здійснюватися шляхом виділення й подальшого вивчення в аналізованому коді програми тих предикат, які в процесі його виконання повертають завжди одне й теж значення. Статистичний аналіз також може бути використаний для оцінки коректності здійсненого процесу деобфускації, для цього паралельно запускається програма "a" і програма, отримана в результаті деобфускації "a'", їм передаються еквівалентні вхідні дані, і відбувається порівняння вихідних. Якщо вихідні дані однакові, то можна припустити, що процес деобфускації був здійснений правильно.

– Аналіз потоку даних, ґрунтується на вивченні того, як у процесі роботи програми змінюються в ній дані (змінні, масиви).

Статичний аналіз – це сімейство технологій аналізування програм, де аналізовану програму фактично не потрібно запускати, при цьому необхідну інформацію про неї одержують за допомогою спеціальних програм. Наприклад, статичний аналіз програм, представлених у двійковому виді, можна здійснити, використовуючи декомпілятор, а представлених у вихідному виді, використовуючи будь який текстовий редактор. Технології статичного аналізу відрізняються від більшості існуючих, її основні якості полягають в тому, що вона є більше комплексною, і базується на семантиці (визначає значеннєве значення пропозицій алгоритмічної мови) самого коду програми.

Статичний аналіз дозволяє досліджувати програму, і виявити деякі причини її можливого поводження під час її роботи, тобто результати статичного аналізу не можна вважати абсолютно точними.

У свою чергу динамічний аналіз полягає в аналізі/тестуванні програми під час її виконання. Він вважається точним, так як він досліджує фактичне поводження програми, під час її роботи.

Динамічний аналіз звичайно здійснюється швидше, ніж статичний, так як час його виконання найчастіше залежить від швидкості виконання аналізованої програми. Статичний же аналіз звичайно вимагає багато обчислень і є тривалим, особливо коли аналізуються великі програми. Недолік динамічного аналізу полягає в тому, що отримані результати можуть не відповідати результатам, одержуваним при наступних запусках однієї й тої ж програми.

Основні проблеми деобфускації, пов'язані з великою кількістю обчислень, і складністю її алгоритмів.

3.2 Розробка структурної схеми

На рисунку 3.1 зображена структурна схема програмного забезпечення, яке реалізує процес обфускації коду. Розглянемо цю схему більш детально.

Процес обфускації може бути здійснений над кожним з вище перерахованих видів подання програмного коду, тому прийнято виділяти наступні рівні процесу обфускації:

– нижчий рівень, коли процес обфускації здійснюється над ассемблерним кодом програми, або навіть безпосередньо над двійковим файлом програми, що зберігає машинний код;

– вищий рівень, коли процес обфускації здійснюється над вихідним кодом програми написаному мовою високого рівня.

Здійснення обфускації на нижчому рівні вважається менш комплексним процесом, але при цьому більш важко реалізованим з ряду причин. Одна із цих причин полягає в тому, що повинні бути враховані особливості роботи більшості процесорів, так як спосіб обфускації, прийнятний на одній архітектурі, може виявитися неприйнятним на іншій.

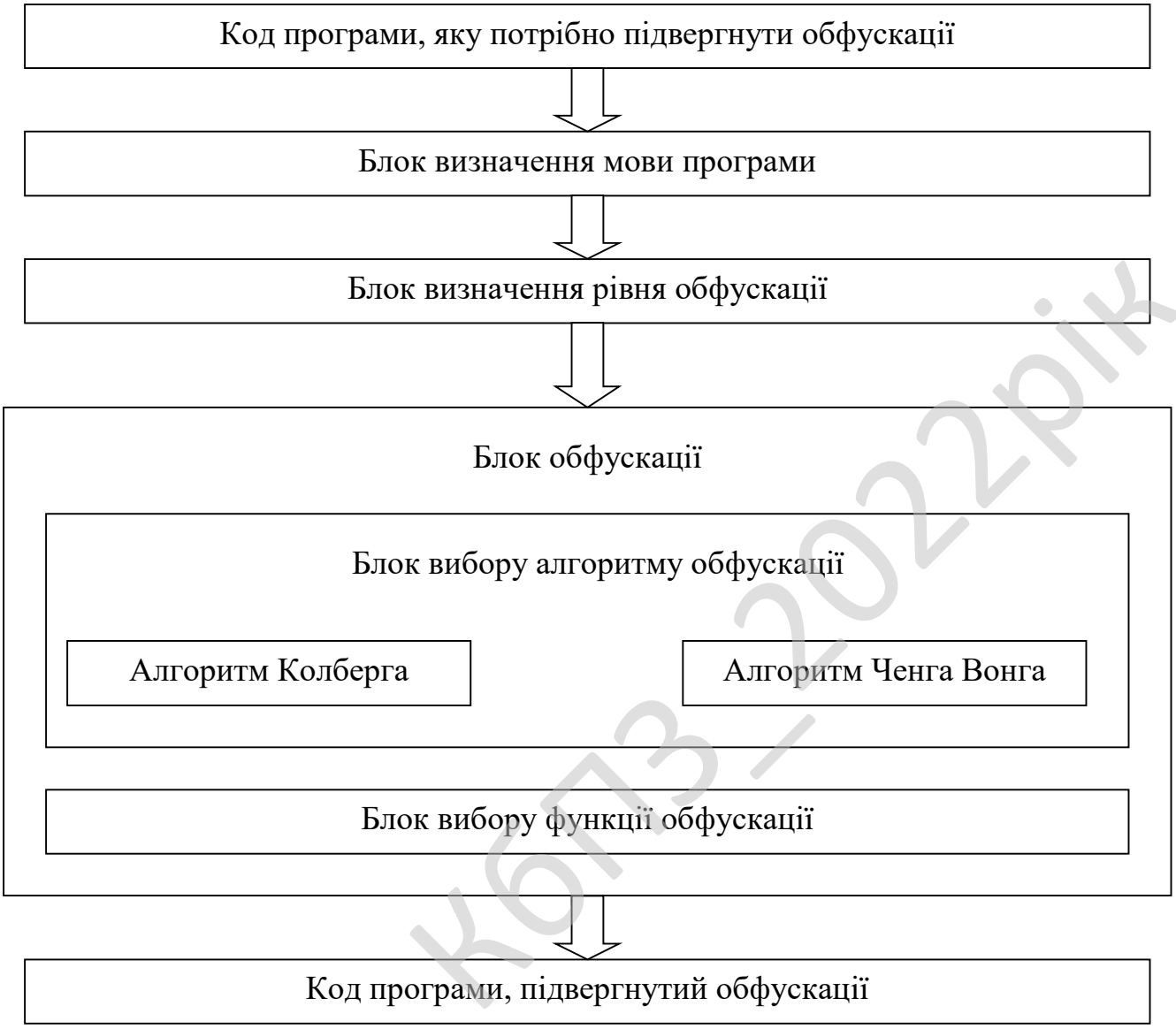

Рисунок 3.1 – Структурна схема системи

Також на сьогоднішній день процес низькорівневої обфускації досліджений мало (напевно так як не встиг одержати широкої популярності).

Більшість існуючих алгоритмів і методів обфускації (включаючи ті які будуть розглянуті нижче) можуть бути застосовані для здійснення процесу обфускації як на нижчому, так і на вищому рівні.

Також іноді може бути неефективно, піддавати обфускації весь код програми (наприклад, через те, що в результаті може значно знизиться час

виконання програми), у таких випадках доцільно здійснювати обфускацію тільки найбільш важливих ділянок коду.

Алгоритми процесу обфускації

Алгоритм обфускації в більшості випадків розглядається як алгоритм, який повинен дотримуватися обфускатор (незалежна програма, що здійснює процес обфускації над переданим їй кодом).

На даний момент існують різні алгоритми здійснення процесу обфускації, починаючи від загальних (абстрактних) алгоритмів процесу обфускації й закінчуючи більше просунутими. Ці алгоритми створювалися відповідно до можливостей тої або іншої мови програмування, і на сьогодні більшість із них адаптовано безпосередньо під мови програмування високого рівня. Нижче представлене короткий опис деяких з них.

Алгоритм Колберга ("Collberg`s algorithm").

Даний алгоритм оперує наступними вхідними значеннями:

– Програма "A", яка складається з вихідних або об'єктних (двійкових) файлів "{C1,C2}".

– Стандартні бібліотеки, використовувані програмою "{L1,L2}".

– Набір процесів, що трансформують, "T{T1,T2}".

– Певний фрагмент коду "S", що витягає із програми "A", і який безпосередньо буде підданий трансформації.

– Набір функцій "E{E1,E2}" які будуть визначати ефективність застосування певних процесів, що трансформують, "{T1,T2}" до фрагмента коду "S".

– Набір функцій "I{I1,I2}" які будуть визначати важливість фрагмента коду "S", і залежно від цього будуть задавати певне значення змінної "RequireObfuscation" (чим "S" важливіше тим ця змінна буде зберігати більше значення).

– Дві числові змінні "AcceptCost" > 0, "RequireObfuscation" > 0, де перше зберігає інформацію про доступне максимальне збільшення системних ресурсів

які потрібні для програмі "A" після того як вона піддасться обфускації, а друг змінна буде зберігати значення необхідного рівня здійснення обфускації (чим важливіше фрагмент коду "S", тим це значення повинне бути більше).

Алгоритм Колберга має таку послідовність операцій:

– Завантаження елементів "{C1,C2}" програми "A".

– Завантаження бібліотек "{L1,L2}".

– Здійснення обфускації над програмою "A", шляхом виділення фрагмента коду "S" і визначення найбільш ефективного процесу трансформації для нього. Цей етап повторюється доти, поки не буде, досягнуть необхідний рівень обфускації "RequireObfuscation" або припустиме збільшення ресурсів "AcceptCost".

– Генерація программ, що трансформується, "A'".

Алгоритм Колберга вважається загальним алгоритмом здійснення процесу обфускації (тобто він не визначає, як саме повинен здійснюватися, той або інший метод обфускації), нижче буде розглянутий більше спеціалізований алгоритм, так як він описує послідовність здійснення одного з методів обфускації, а саме обфускації керування.

Алгоритм Ченги Ванга.

У якості вхідних даних алгоритм приймає типову процедуру, написану мовою високого рівня. Процес обфускації кожної такої процедури складається із трьох етапів:

– Створення графа потоку керування цієї процедури (граф задається безліччю блоків і безліччю зв'язків з'єднуючих їх), після чого граф розбивається, шляхом заміни циклічних конструкцій у ньому на конструкції типу "if (умова) goto".

– Нумерація всіх блоків у графі, і додавання в код процедури змінної (наприклад "swVar"), яка зберігає номер наступного виконуваного блоку.

– Приведення графа до однорідного ("плоскому") виду.

Вище описаний варіант алгоритму обфускації ("Chenxi Wang's algorithm") є не сильно стійким, так як визначити наступний виконуваний блок, неважко (він у нашім випадку буде зберігатися в змінної "swVar"). Тому для підвищення його стійкості вводять масив (наприклад "@gg"), що містить крім номерів блоків, не потрібну інформацію, у результаті запис "$swVar = S6", можна замінити на щось подібне "$swVar = $gg[$gg[1] + $gg[3]]".

3.3 Розробка функціональної схеми

На рисунку 3.2 зображена функціональна схема системи. Нижче розглянемо її більш докладно.

З цього рисунку ми бачимо, що програма обфускації виконує наступні дії над кодом, які визначаються заданими видами обфускації:

1. Лексична обфускація:

– Видалення всіх коментарів у коді програми.

– Видалення різних пробілів.

– Заміна імен ідентифікаторів.

– Додавання різних зайвих операцій.

– Зміна розташування блоків.

2. Обфускація даних.

а) Обфускація зберігання:

– Зміна інтерпретації даних певного типу.

– Зміна строку використання сховищ даних.

– Перетворення статичних даних у процедурні.

– Поділ змінних.

– Зміна подання (або кодування).

б) Обфускація з'єднання:

– Об'єднання змінних.

– Реструктурування масивів.

– Зміна ієрархій спадкування класів

в) Обфускація переупорядкування.

3. Обфускація керування.

а) Обчислювальна обфускація:

– Розширення умов циклів.

– Додавання недосяжного коду.

– Усунення бібліотечних викликів.

– Додавання надлишкових операцій.

– Розпаралелювання коду.

б) Обфускація з'єднання:

– Вбудовування функцій.

– Добування функцій.

– Чергування, об'єднання фрагментів коду програми.

– Клонування.

– Трансформація циклів.

– Розгорнення циклів.

– Поділ циклів.

в) Обфускація послідовності.

Перейдемо до більш детального опису функціональної схеми розробленого програмного продукту обфускації коду програми.

Процеси обфускації можна класифікувати по видах, залежно від способу модифікації коду програми.

Лексична обфускація

Найбільш проста, полягає у форматуванні коду програми, зміні його структури, таким чином, щоб він став нечитабельним, менш інформативним, і важким для вивчення.

Рисунок 3.2 – Функціональна схема системи

Обфускація такого виду містить у собі:

– Видалення всіх коментарів у коді програми, або зміна їх на ті, що дезінформують.

– Видалення різних пробілів, відступів які звичайно використовують для кращого візуального сприйняття коду програми.

– Заміну імен ідентифікаторів (імен змінних, масивів, структур, хешів, функцій, процедур і т.д.), на довільні довгі набори символів, які важко сприймати людині.

– Додавання різних зайвих (сміттєвих) операцій.

– Зміна розташування блоків (функцій, процедур) програми, таким чином, щоб це не яким образом не вплинуло на її працездатність.

Зміна глобальних імен ідентифікаторів варто робити в кожній одиниці трансляції (один файл вихідного коду), так щоб вони мали однакові імена (у противному випадку програма, яка захищається, може стати не функціональною). Також варто враховувати специфічні ідентифікатори, прийняті в тій мові програмування, на якому написана програма, яка захищається, імена таких ідентифікаторів, краще не змінювати.

Дана обфускація програмного коду, у порівнянні з іншими, дозволяє порівняно швидко привести вихідний код програми, у нечитабельний стан. Один з її недоліків полягає в тому, що вона ефективна тільки для здійснення високорівневої обфускації.

Обфускація даних

Така обфускація пов'язана із трансформацією структур даних. Вона вважається більше складною, і є найбільш просунутою й часто використовуваною. Її прийнято ділити на три основні групи, які описані нижче.

Обфускація зберігання. Полягає в трансформації сховищ даних, а також самих типів даних (наприклад, створення й використання незвичайних типів даних, зміна подання існуючих і т.д.). Нижче наведені основні методи, що дозволяють здійснити таку обфускацію:

– Зміна інтерпретації даних певного типу. Як відомо збереження, будь яких даних у сховищах (змінних, масивах і т.д.) певного типу (ціле число, символ) у процесі роботи програми, дуже розповсюджене явище. Наприклад, для переміщення по елементах масиву дуже часто використовують змінну типу "ціле число", що виступає в ролі індексу. Використання в цьому випадку змінних іншого типу можливо, але це буде не тривіально й може бути менш ефективно. Інтерпретація комбінацій розрядів даних, що втримуються в сховищі, здійснюється залежно від його типу. Так, наприклад, можна сказати, що 16-розрядна змінна цілого типу утримуючої комбінації розрядів 0000000000001100 представляє ціле число 12, але це проста угода, дані в такий змінній можна інтерпретувати по-різному (не обов'язково як 12, а, наприклад як 1100 і т.д.).

– Зміна строку використання сховищ даних, наприклад перехід від локального їхнього використання до глобального й навпаки.

– Перетворення статичних (незмінюванних) даних у процедурні. Більшість програм, у процесі роботи, виводять різну інформацію, що найчастіше в коді програми представляється у вигляді статичних даних таких як рядка, які дозволяють візуально орієнтуватися в її коді й визначати виконувані операції. Такі рядки також бажано змінити обфускації, це можна зробити, просто записуючи кожний символ рядка, використовуючи його ASCII код, наприклад символ "A" можна записати як 16-річне число "0x41", але такий метод банальний. Найбільш ефективний метод, це коли в код програми в процесі здійснення обфускації додається функція, що генерує необхідний рядок відповідно до переданими їй аргументами, після цього рядка в цьому коді віддаляються, і на їхнє місце записується виклик цієї функції з відповідними аргументами. Також до статичних даних відносяться числові константи, які можуть бути також трансформоване, наприклад число 1 можна представити як: (a + 1 – b), де a = b.

– Поділ змінних. Змінні фіксованого діапазону можуть бути розділені на дві й більше змінних. Для цього змінну "V" що має тип "x" розділяють на "k" змінних "v1,...,vk" типу "y" тобто "V == v1,...,vk". Потім створюється набір

функцій що дозволяють витягати змінну типу "x" зі змінних типу "y" і записувати змінну типу "x" у змінні типу "y".

– Зміна подання (або кодування).

Обфускація з'єднання. Один з важливих етапів, у процесі реверсивної інженерії програм, заснований на вивченні структур даних. Тому важливо постаратися, у процесі обфускації, ускладнити подання використовуваних програмою структур даних. Наприклад, при використанні обфускації з'єднання це досягається завдяки з'єднанню незалежних даних, або поділу залежних. Нижче наведені основні методи, що дозволяють здійснити таку обфускацію:

– Об'єднання змінних. Дві або більше змінних "v1,...,vk" можуть бути об'єднані в один змінну "V", якщо їхній загальний розмір ("v1,...,vk") не перевищує розмір змінної "V". Наприклад, розглянемо простий приклад об'єднання двох коротких цілочисельних змінних "X","Y" (розміром 16 біт) в одну цілочисельну змінну "Z" (розміром 32 біта). Для цього скористаємося формулою: Z(X,Y) = 2^16 * Y + X, що дозволить, зневажаючи додаванням, визначати значення Y, тобто нехай X = 12, Y = 4 => Z = 65536 * 4 + 12 = 262156, тепер знаючи "Z" для знаходження "Y" можна 262156 / 65536 = 4.000183105 або приблизно 4. При здійсненні арифметичних операцій над значеннями змінних "X", "Y" зберігаються в "Z" потрібно враховувати вище наведену формулу.

– Реструктурування масивів, полягає в заплутуванні структури масивів, шляхом поділу одного масиву на декілька підмасивів, об'єднання декількох масивів в один, згортання масиву (збільшуючи його розмірність) і навпаки, розвертання (зменшуючи його розмірність). Наприклад, один масив "@A" можна розділити на декілька підмасивів "@A1, @A2", при цьому один масив "@A1" буде містити парні позиції елементів, а другий "@A2" непарні позиції елементів масиву "@A". Під згортанням масиву розуміється створення з одномірного масиву, двовимірного. Наприклад, одномірний масив "A" з попереднього приклада, що має розмір 5 можна замінити двовимірним масивом "B" розміром 2.

– Зміна ієрархій спадкування класів, здійснюється шляхом ускладнення ієрархії спадкування за допомогою створення додаткових класів або використання помилкового поділу класів.

Обфускація переупорядкування. Полягає в зміні послідовності оголошення змінних, внутрішнього розташування сховищ даних, а також переупорядкуванні методів, масивів (використання нетривіального подання багатомірних масивів), певних полів у структурах і т.д.

Обфускація керування

Обфускація такого виду здійснює заплутування потоку керування, тобто послідовності виконання програмного коду.

Більшість її реалізацій ґрунтується на використанні непрозорих предикат, у якості, який виступають, послідовності операцій, результат роботи яких складно визначити (саме поняття "предикат" виражає властивість одного об'єкта (аргументу), або відносини між декількома об'єктами).

Визначення. Предикат "P" уважається непрозорим предикатом, якщо його результат відомий тільки в процесі обфускації, тобто після здійснення процесу обфускації, визначення значення такого предиката, стає важким.

Позначимо непрозорий предикат, що повертає завжди значення TRUE як "P(t)", а повертаючий значення FALSE, як "P(f)", тоді непрозорий предикат, що може повернути кожне із цих двох значень (тобто або TRUE, або FALSE, що нам невідомо) як "P(t,f)". Ці позначення, будуть використовуватися далі в контексті опису обфускації керування. Непрозорі предикати можуть бути:

– Локальними – обчислення втримуватися усередині одиночного вираження (умови), наприклад (запис "(f)" після умови перевірки, указує, що це предикат типу "P(f)").

– Глобальними – обчислення втримуватися усередині однієї процедури (функції.

– Міжпроцедурними – обчислення втримуватися усередині різних процедур (функцій).

Ефективність обфускації керування в основному залежить від використовуваних непрозорих предикат, це змушує створювати як можна складні для вивчення, і прості, гнучкі у використанні непрозорі предикати, але рівною мірою також не маловажну роль має час їхнього виконання, а також кількість виконуваних операцій, крім усього цього предикат не сильно повинен відрізнятися від тих функцій, які виконує сама програма, і не повинен містити надмірну кількість обчислень, у противному ж випадку зловмисник, зможе відразу його виявити. Так як часто для деобфускації використовують технологію статичного аналізу, а одним з її недоліків є складність (трудомісткість) статичного аналізу структур покажчиків, то звичайно в процесі обфускації керування використовують стійкі непрозорі предикати, які дозволяють використовувати недоліки технології статичного аналізу.

Основна ідея стійких непрозорих предикатів полягає в тому, що в програму, у процесі обфускації додається код, що створює набір динамічних структур, а також глобальних покажчиків, які будуть посилатися на різні елементи усередині цих структур. Крім цього, даний код повинен іноді обновляти ці структури (додавати нові елементи в них, поєднувати або розділяти деякі їх, змінювати значення глобальних покажчиків, і т.д.), але таким чином, щоб при цьому минулому збережені деякі умови, наприклад "покажчик p і q ніколи не будуть указувати на той самий елемент" або "покажчик p може посилатися (указувати) на покажчик q" і таке інше. Ці умови в наслідку дозволяють створювати необхідні непрозорі предикати.

Методи що дозволяють здійснити обфускацію керування, класифікуються на три основних групи:

Обчислювальна обфускація. Зміна дотичного головної структури потоку керування. До них можна віднести:

– Розширення умов циклів. Для цього звичайно використовують непрозорі предикати, таким чином, щоб вони не яким образом не впливали на кількість виконань циклічного коду.

– Додавання недосяжного коду, (який не буде виконуватися в процесі роботи програми).

– Усунення бібліотечних викликів. Більшість програм, використовують функції, які визначені в стандартних бібліотеках вихідної мови, на якому писалася програма (наприклад, у Сі це бібліотека "libc"), робота таких функцій добре документована й часто відома зловмисникам, отже, їхня присутність у коді програми, може допомогти в процесі її реверсивної інженерії. Тому імена функцій зі стандартних бібліотек, також бажано додати обфускації, тобто змінити на найбільш безглузді, які потім будуть фігурувати в коді програми, яка захищається. Один зі способів рішення такої проблеми, полягає у використанні в програмі власної версії стандартних бібліотек (які виходять у результаті перейменовування всіх функцій в оригінальній стандартній бібліотеці), це не змінить істотно час виконання програми. Але для того, щоб така програма була стерпної, і могла використовуватися багатьма користувачами, її потрібно буде поставляти разом зі зміненою версією стандартної бібліотеки, що значно збільшить розмір програми. Тому такий спосіб рішення проблеми неефективний. При здійсненні такої обфускації треба в першу чергу ґрунтуватися на особливостях стандартної бібліотеки вихідної мови (те, як у ній взаємозалежні імена функцій з їхніми кодами й т.д.).

– Додавання надлишкових операцій (мертвого коду) у ті ділянки програмного коду, які найбільш важкі (визначально) для вивчення. Часто надлишкові операції, використовуються для розширення арифметичних виражень (наприклад, у непрозорих предикатах)

– Розпаралелювання коду, полягає в поділі коду на окремі незалежні ділянки, які під час роботи програми будуть виконуватися паралельно (тобто одночасно), така обфускація також може полягати в імпровізації розпаралелювання коду програми, для цього створюється так званий макет процесу, що насправді не буде виконувати ніяких корисних операцій.

Обфускація з'єднання. Об'єднання або поділ певних фрагментів коду програми, для того щоб забрати логічні зв'язки між ними. Нижче наведені основні методи, що дозволяють здійснити таку обфускацію:

– Вбудовування функцій, здійснюється шляхом вбудовування коду функції, у місця її виклику (якщо її код буде убудований в усі місця її виклику, тоді саму функцію можна забрати з коду програми).

– Добування функцій, є зворотною дією, стосовно вбудовування функцій. Здійснюється в результаті об'єднання деякої групи взаємозалежних операторів у коді вихідної програми в окрему функцію (при необхідності для цієї функції можна визначити деякі аргументи), що потім заміщають ці групи операторів. Але варто врахувати, що таке перетворення може бути знято компілятором у процесі компіляції коду програми.

– Чергування, об'єднання фрагментів коду програми (функцій наприклад), що виконують різні операції, воєдино (в одну функцію, при цьому в таку функцію, варто додати об'єкт, залежно від значення якого, буде виконуватися код однієї з об'єднаних функцій).

– Клонування, даний метод дозволяє ускладнити аналіз контексту використання функцій, і об'єктів використовуваних у коді вихідної програми. Процес клонування функцій складається у виділенні певної функції "F", часто використовуваної в коді програми, після чого над кодом цієї функції здійснюється трансформація, і створюється її клон "F'", що також буде доданий у код вихідної програми, при цьому частина викликів функції "F" у коді вихідної програми, буде заміщена на виклик функції "F'". У результаті цього в зловмисника створиться подання про те, що функції "F", і "F'" різні. Клонування об'єктів здійснюється аналогічним способом.

– Трансформація циклів. Цикли зустрічаються в коді різних програм, і їх також можна додати трансформації. Блокування циклів, полягає в додаванні вкладених циклів в існуючі, у результаті робота існуючих циклів буде заблокована, на якийсь діапазон значень.

– Розгорнення циклів, повторення тіла циклу один або більше раз (якщо кількість виконуваних циклів відомо в процесі здійснення обфускації (наприклад, дорівнює "N"), то цикл, може бути, розгорнуть повністю, у результаті повторення його тіла в коді N раз).

– Поділ циклів, цикл, що складається з більш ніж однієї незалежної операції можна розбити на кілька циклів (які повинні виконуватися однакову кількість разів), попередньо розбивши на кілька частин, його тіло.

Бажано здійснювати над вихідним циклом послідовно всі перераховані вище трансформації циклів, це дозволить ускладнити його статичний аналіз.

Обфускація послідовності. Полягає в переупорядкуванні блоків (інструкцій переходів), циклів, виражень.

Розглянувши усі блоки функціональної схеми перейдемо до розгляду діаграми взаємодії процесів, які відбуваються у системі.

3.4 Розробка діаграми процесів

Діаграма взаємодії процесів системи, розробленої у результаті виконання магістерського проектування, наведена на рисунку 3.3.

Як ми бачимо з діаграми першим процесом, який запускається у системі, є процес початку/кінця роботи програми.

Після ініціалізації цього процесу відбувається запуск процесу визначення мови програмування на якій написаний програмний продукт.

Коли мова визначена, то запускається процес визначення рівня обфускації.

Після визначення рівня обфускації завантажується процес вибору алгоритму обфускації.

Цей процес дозволяє завантажити такі наступні процеси як:

– Процес який реалізує алгоритм Колберга.

– Процес який реалізує алгоритм Чена Вонга.

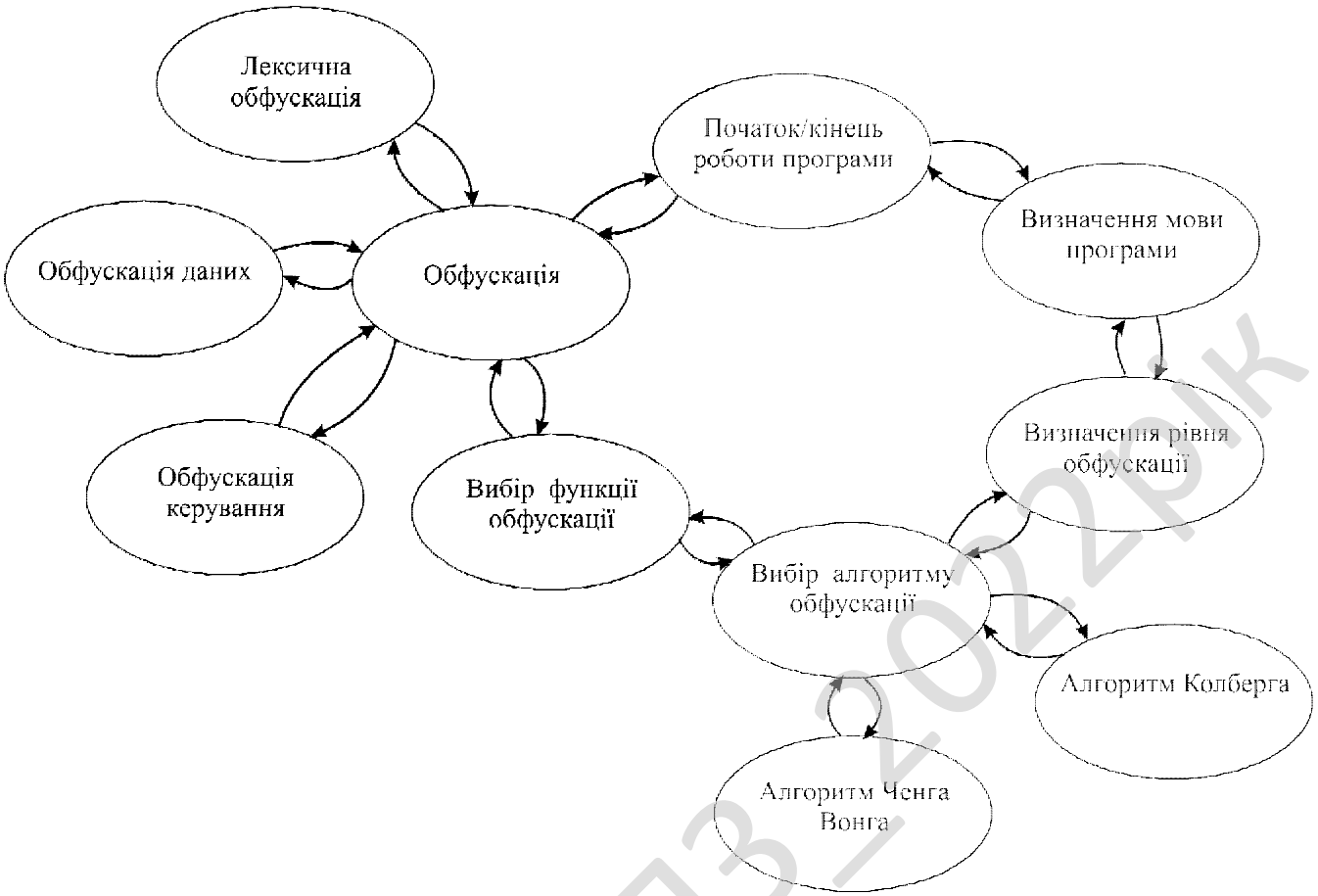

Рисунок 3.3 – Діаграма процесів системи

Крім того процес вибору алгоритму обфускації взаємодіє з процесом вибору функції обфускації.

Осатаній процес взаємодіє вже з суто процесом обфускації.

Процес обфускації взаємодіє з наступними процесами:

– Процес вибору функції обфускації.

– Процес лексичної обфускації.

– Процес обфускації даних.

– Процес обфускації керування.

Крім того процес обфускації є завершуючим у системі, тому, логічно, він взаємодіє з процесом початку/кінця роботи програмного забезпечення.

Таким чином, розглянувши опис системи, структурну, функціональну схеми системи, та діаграму взаємодії процесів перейдемо до опису блок-схем основної програми, та підпрограм, які використовуються, для реалізації системи.

4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

На рисунку 4.1 наведено блок-схему основної програми. Її робота складається з поетапного виконання наступних кроків. Спершу відбувається виведення основного вікна програми. Після цього відбувається відкриття файлу, який необхідно захистити. Наступною дією, якою виконує програма є визначення мови програми, яку необхідно піддати обфускації.

Після цього відбувається визначення рівня обфускації, тобто задається яка складність заплутування вихідного коду, буде встановлення користувачем, для утруднення процесу дизасемблювання програми.

За визначенням рівня обфускації відбувається вибір алгоритму обфускації.

Для обраного алгоритму відбувається вибір функції обфускації.

Таким чином, після того, як користувач встановить вихідні параметри для процесу обфускації, а саме:

– відкриття файлу, який необхідно захистити;

– визначення мови програми;

– визначення рівня обфускації;

– вибір функції обфускації;

відбувається підвергнення коду програми обфускації.

Наступною дією є виведення звіту.

Після виведення настає етап на якому користувач може обрати, перевірити працездатність програми після обфускації, або ні.

Якщо користувач обирає пункт перевірки працездатності програми після обфускації, тоді відбувається виконання наступних дій:

– Запускається програма, яка підверглась обфускації.

– Виводяться результати тесту.

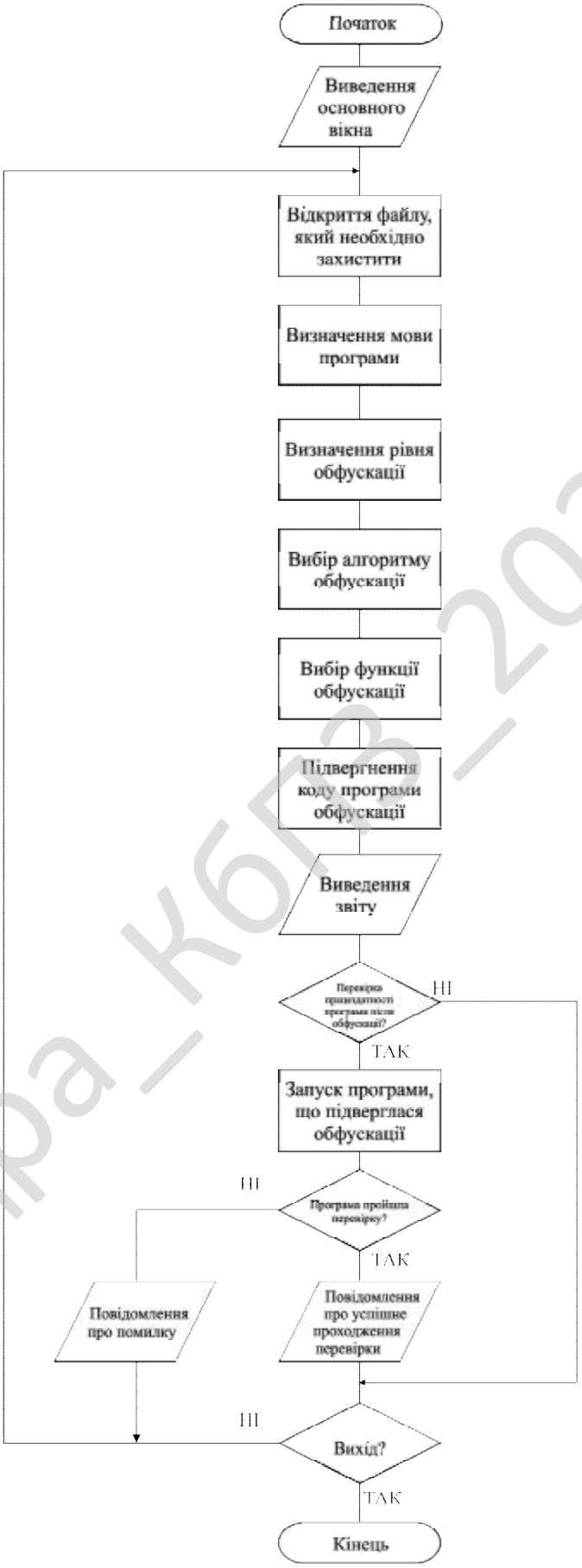

Рисунок 4.1 – Блок-схема алгоритму роботи основної програми

Якщо програма робить після проведення над нею процесу обфускації, то виводиться повідомлення про вдале проходження програмою процесу заплутування коду, й програма завершає свою роботу.

У противному випадку, тобто, коли програма не робить після проведення над нею процесу обфускації, то відбувається перехід на початок програми.

Розглянувши роботу основної програми перейдемо до розгляду роботи підпрограми здійснення обфускації.

Підпрограма починає працювати з визначення користувачем типу обфускації, тобто, що саме буде піддаватися обфускації.

У програмі реалізовані наступні типи обфускації:

– Лексична обфускація.

– Обфускація даних.

– Обфускація керування.

Розглянемо, як працює кожний з цих типів обфускації.

При виборі користувачем лексичної обфускації відбувається виконання наступних послідовних кроків реалізації програмного продукту.

Спершу відбувається видалення усіх коментарів у коді програми.

Після цього відбувається видалення різних пробілів.

Наступним шагом є заміна імен ідентифікаторів.

За заміною імен ідентифікаторів відбувається додавання різних зайвих операцій.

Останнім кроком лексичної обфускації є зміна розташування блоків.

При виборі користувачем обфускації даних відбувається виконання наступних послідовних кроків реалізації програмного продукту.

Спершу відбувається зміна інтерпретації даних певного типу.

Після цього відбувається зміна строку використання сховищ даних.

Наступним шагом є перетворення статистичних даних у процедурні.

За перетворенням статистичних даних у процедурні відбувається поділ змінних.

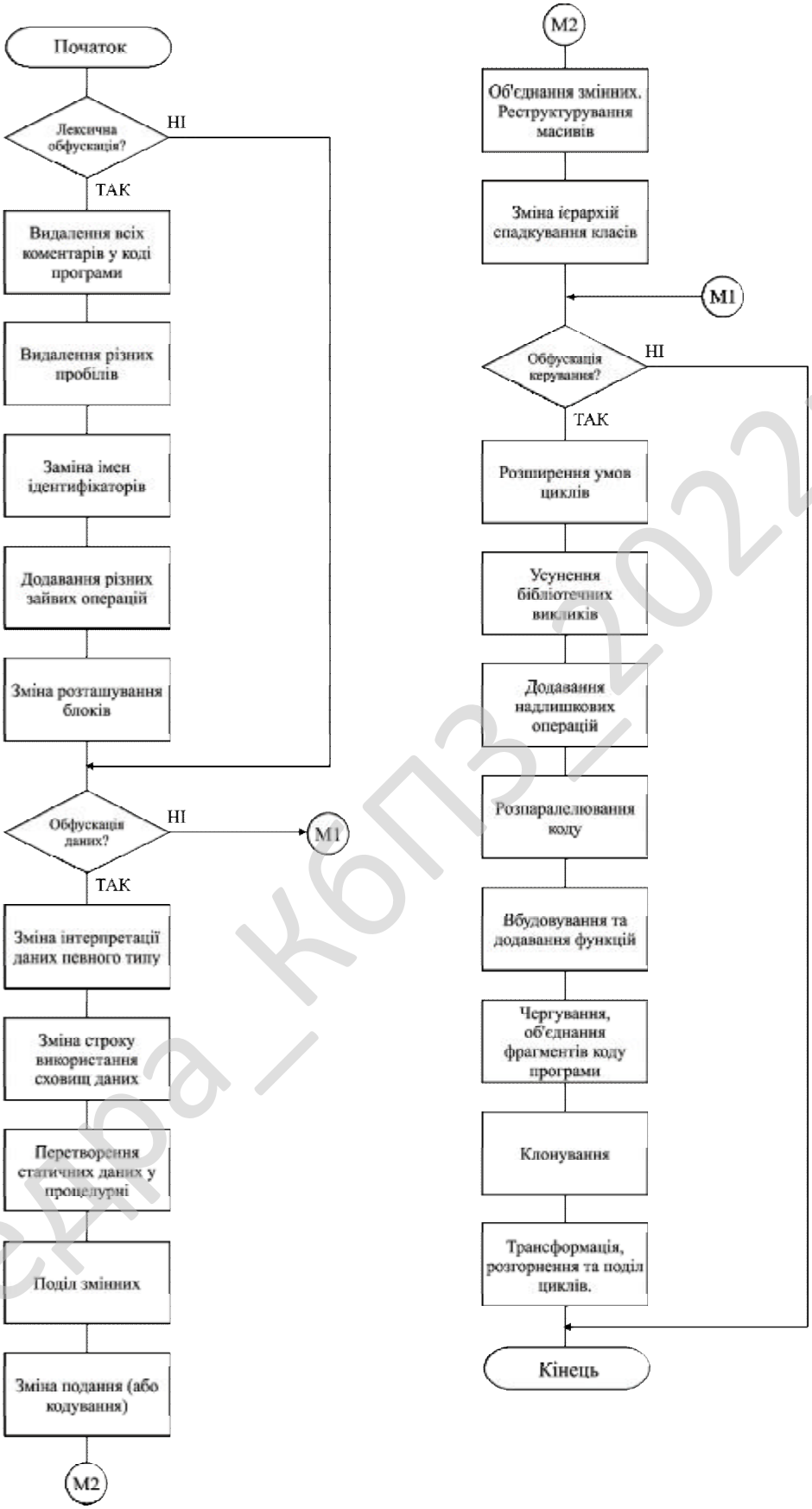

Рисунок 4.2 – Блок-схема алгоритму роботи підпрограми здійснення обфускації

За поділом змінних відбувається об’єднання змінних та реструктурування масивів.

Останнім кроком обфускації даних є зміна ієрархій спадкування класів.

При виборі користувачем обфускації керування відбувається виконання наступних послідовних кроків реалізації програмного продукту.

Спершу відбувається розширення умов циклу.

Після цього відбувається усунення бібліотечних викликів.

Наступним шагом є додавання надлишкових операцій.

За додаванням надлишкових операцій відбувається розпаралелювання коду.

Після виконання розпаралелювання коду відбувається вбудовування та додавання функцій.

Наступним кроком є встановлення чергування та об’єднання фрагментів коду програми.

Після цього відбувається клонування цих блоків.

Останнім кроком обфускації керування є трансформація, розгорнення та поділ циклів.

Розглянувши та описавши блок-схеми основної програми та підпрограми, яка реалізує процедуру обфускації, перейдемо до розгляду практичної реалізації обфускації.

Існують різні типи обфускаторів: одні займаються інтерпретуємими мовами типу Perl або PHP і «кореблять» вихідні тексти (видаляють коментарі, дають змінним безглузді імена, шифрують строкові константи й т.д.), інші «перемелюють» байт-код віртуальних машин Java і .NET, що технічно зробити набагато сутужніше. Більше розвинені обфускатори вламуються безпосередньо в машинний код, «розбавляючи» його сміттєвими інструкціями й виконуючи цілий ряд структурних (рідше математичних) перетворень, що змінюють програму до невпізнанності.

Фактично, це поліморфні генератори. Проблема в тому, що поліморфний генератор може за лічені секунди згенерувати хоч мільярд безглуздих команд, перемішавши їх з декількома кілобайтами корисного коду, що дозволяють сучасні процесори й жорсткі диски. Нехай навіть із втратою ефективності, але всім вже давно наплювати на ефективність.

Дизассемблери ще не навчилися видаляти «сміття» в автоматичному режимі, а проаналізувати мегабайти коду вручну нереально. Потрібні передові методики реконструкції потоку керування, що розплавляють «засмічений» код і поділяючі його на «корисні» і «марні» фракції. Їх немає навіть на рівні «теоретичного розуміння». Хоча деякі ідеї на цей рахунок є (наприклад накладення маршруту трасування на графи залежностей за даними), до практичної реалізації ще далеко.

Обфускація здійснює протидію зловмисникам, перешкоджаючи реконструкції алгоритмів і швидкому взлому захистів, але ці проблеми мерхнуть перед ситуацією в антивірусній індустрії. Щоб зламати програму, аналізувати її алгоритм у загальному-то необов'язково. Зате виявити шкідливий код (він же malware) без цього вже не вдасться!

У найпростішому випадку поліморфний генератор просто «накачує» програму купою нічого не значущих команд типу nop, xchg reg,reg, що ніколи не виконуються переходами типу xor reg,reg/jnz junk, де xor – значима команда, а junk – «мертвий код».

– or ch, ch ; «сміття», що не впливає на регістр ch, але, що впливає на регістр прапорів, однак цей вплив перекривається наступним xor;

– xor eax,eax ; потенційно значима команда;

– seto bl ; «сміття», що встановлює bl в 1, якщо є переповнення, а після xor його завжди немає;

– repne jnz short loc_43409A ; «сміття», що передає керування, якщо не нуль, але після xor прапор нуля завжди встановлений, плюс безглуздий префікс repne;

– rep jnp short loc_43408D ; «сміття», що передає керування, якщо непара, але після xor прапор парності завжди встановлений;

– jo short loc_434094 ; «сміття», що передає керування, якщо прапор переповнення встановлений, а він скинутий xor;

– xchg ebx,ebx ; «сміття», обмін регістрів ebx місцями.

От приклад вихідного методу (перед обфускацією):

Код:

```
private void CalcPayroll(SpecialList employeeGroup)
{
   while(employeeGroup.HasMore())
   {
     employee = employeeGroup.GetNext(true);
     employee.UpdateSalary(); DistributeCheck(employee);
   }
}
```

А от результат (після):

Код:

```
private void _1(_1 _2)
{
  while(_2._1())
  {
    _1 = _2._1(true);
    _1._1();
    _1(_1);
   }
}
```

Спробуйте зрозумійте зміст алгоритму після обфускації. На це піде час – треба довідатися, що за класи беруть участь, спробувати зрозуміти їхнє призначення, у загальному має бути велика праця.

У цьому завдання обфускації – утруднити для розуміння вихідний код, заплутати й усунути логічні зв'язки в коді.

Плюси:

– Обфускатори роблять дизассембльований код важким для вивчення, перетворюючи IsLicensed() в x().

– Деякі обфускатори використовують баги ILDASM для захисту від дизассемблінгу в ньому (Salamander).

– Деякі обфускатори навіть конвертують код в native код, роблячи марним дизассемблінг (Salamander).

– Деякі обфускатори шифрують і пакують ваш exe і referenced збірки в один exe-файл, так що розмір програми може зменшитися 2-4 рази й не піддається дизассемблінгу (Thinstall).

Мінуси:

– Продукт залишається дизассембльованим.

– Зібрати складання після дизассемблінгу не буде важко.

– IL код – доступний для читання й розуміння в порівнянні с ассемблерним.

– «Захист» обфускаторів, які використовують баги ILDASM будуть неспроможні перед дизассемблерами інших розроблювачів.

Більш складні обфускатори «перемішують» код, закручуючи потік керування в заплутану спіраль умовних/безумовних переходів, що використовують техніку «перекриття» команд. Деякі байти належать відразу двом, а в деяких випадках і трьом (!) машинним інструкціям, що «засліплює» дизассемблери, змушуючи їх генерувати неповний і неправильний лістинг.

Однак в інтерактивному режимі все-таки можна дизасемблювати код, але дуже вже утомливо. Краще скористатися трассером, що генерує лістинг реально виконуваних машинних команд. Заодно позбуваємося від частини сміття й «мертвого» коду.

Фрагмент лістингу, сгенерированный розробленою програмою. Демонстрація техніки «перекриття» машинних команд, використовуваної обфускаторами

```
.adata:0043400E loc_43400E: ; CODE XREF: .adata:00434023j
.adata:0043400E ; .adata:loc_43401A j
.adata:0043400E mov eax, 0EBB907EBh
.adata:00434013
.adata:00434013 loc_434013: ; CODE XREF: .adata:loc_43401Dj
```

```
.adata:00434013 seto bl ; стрибок у середину команди
.adata:00434016 or ch, bh
.adata:00434018 jmp short loc_434025
.adata:00434018
.adata:0043401A loc_43401A: ; CODE XREF: .adata:00434009j
.adata:0043401A repne jmp short near ptr loc_43400E+4
.adata:0043401D
.adata:0043401D loc_43401D: ; CODE XREF: .adata:loc_43400Cj
.adata:0043401D jmp short near ptr loc_434013+2
```

Зверни увагу на команду «043401Dh:jmp short loc_434013+2», що стрибає за адресою 434013h+2h == 434015h, тобто в середину інструкції 434013h:seto bl. Саме що в середину! З погляду дизассемблера, команда є «атомарною» структурною одиницею, тобто неподільної. Насправді ж усяка машинна інструкція складається з послідовності байт і може бути виконана з будь-якого місця! У всякому разі, x 86-процесори не вимагають вирівнювання коду. Інакше кажучи, не існує «команд» – існують тільки байти. Якщо почати виконання інструкції не з першого байта, одержимо зовсім іншу команду! Щоб виконати перехід «043401Dh:jmp short loc_434013+2», необхідно підвести курсор до мітки loc_434013 і нажати <U> (так ми «розкришимо» дизассемблерний код на байти), а після перейти за адресою 434015h і нажати <C>, тим самим перетворивши байти в дизассемблерний код.

«Розкриття» накладеної команди

```
.adata:0043400E unk_43400E db 0B8h ; ¬ ; CODE XREF: .adata:loc_434023j
.adata:0043400F db 0EBh ;
.adata:00434010 db 7
.adata:00434011 db 0B9h ; |
.adata:00434012 loc_434012: ; CODE XREF: .adata:loc_43401Aj
.adata:00434012 jmp short loc_434023
.adata:00434014
.adata:00434014 nop
.adata:00434015
.adata:00434015 loc_434015: ; CODE XREF: .adata:loc_43401Dj
.adata:00434015 jmp short loc_43401F ; стрибок сюди
.adata:00434017
.adata:00434017 std
.adata:00434018 jmp short loc_434025
```

```
.adata:0043401A
.adata:0043401A loc_43401A: ; CODE XREF: .adata:00434009j
.adata:0043401A repne jmp short loc_434012
```

На місці seto bl виникла пари інструкцій jmp loc_43401F/std. Який із двох лістингів правильний? Окремо – ні той, ні інший. Вони стають «правильними» тільки вдвох! Однак удержати ці подробиці в голові нереально. Залишається заганяти «альтернативний» лістинг у коментарі. Якщо та сама машинна команда має три й більше «точки входу», то коментарі вже не рятують і виникає плутанина, що змушує використовувати замість дизассемблера трассер.

Витончені обфускатори відслідковують залежності за даними, впроваджуючи осмислені інструкції з «нульовим ефектом». Пояснимо на конкретному прикладі. Допустимо, обфускатору зустрілася конструкція:

Оригінальний код до обфускації

```
PUSH EAX ; останнє звертання до eax
MOV EAX,EBX ; реинициализация eax
```

Легко показати, що між останнім звертанням до eax і його реініціалізацією можна як завгодно модифікувати регістр eax без шкоди для виконання програми, оскільки будь-які операції присвоєння однаково будуть перекриті командою mov eax,ebx.

Код після обфускації

```
push eax ; останнє значиме звертання до eax
xor eax,eax ; сміття
l1:
inc eax ; сміття
jz l2 ; сміття
cmp eax, ebx ; сміття
jnz l1 ; сміття
cmp eax, ecx ; сміття
jge l1 ; сміття
l2:
sub eax, 666h ; сміття
shl eax, 1 ; сміття
mov eax, ebx ; значима реініціалізація eax
```

Також обфускатори можуть тимчасово зберігати регістр на стеці, а потім, позмінюючи його значення, відновлювати колишнє значення.

Тимчасове збереження регістрів на стеці з наступним відновленням:

```
001B:0043402C 50 PUSH EAX ; зберігаємо eax
001B:0043402D 51 PUSH ECX ; зберігаємо ecx
001B:0043402E EB0F JMP 0043403F
001B:0043403F F2EBF5 REPNZ JMP 00434037
001B:00434037 EB0F JMP 00434048
001B:00434048 EBE9 JMP 00434033
001B:00434033 B8EB07B9EB MOV EAX,EBB907EB ; «змінюємо» в eax
001B:0043403B 08FD OR CH,BH ; «змінюємо» в ch
001B:0043403D EB0B JMP 0043404A
001B:0043404A F3EBE4 REPZ JMP 00434031
001B:00434031 EB0F JMP 00434042
001B:00434042 EBF6 JMP 0043403A
001B:0043403A EB08 JMP 00434044
001B:00434044 F2EB08 REPNZ JMP 0043404F
001B:0043404F 59 POP ECX ; відновлюємо ecx
001B:00434050 58 POP EAX ; відновлюємо eax
```

Команда MOV EAX,EBB907EBh на перший погляд виглядає «значимою», але насправді це «сміття», нейтралізуємо командами push eax/pop eax. По суті, весь цей конгломерат робить нульовий ефект, тобто є зовсім бездіяльним кодом. Так що робити вивід про «значимість» команд потрібно з дуже великою обережністю. Поки не буде доведено, що даний шматок коду дійсно має якийсь ефект, він повинен уважатися «сміттям» за замовчуванням.

Деякі обфускатори люблять впроваджувати підроблені розшифровувачі, які розшифровують і відразу зашифровують довільні фрагменти пам'яті.

«Підроблений» розшифровувач, впроваджений оббфускатором

```
00434105 83ED 06 SUB EBP,6
00434108 B8 3B010000 MOV EAX,13B
0043410D 03C5 ADD EAX,EBP
0043410F 33DB XOR EBX,EBX
00434111 81C3 01010101 ADD EBX,1010101
00434117 3118 XOR DWORD PTR DS:[EAX],EBX ; розшифровуємо
00434119 8138 78540000 CMP DWORD PTR DS:[EAX],5478
0043411F 74 04 JE SHORT app_test.00434125
00434121 3118 XOR DWORD PTR DS:[EAX],EBX ; зашифровуємо
00434123 ^EB EC JMP SHORT app_test.00434111
```

Зрозуміло, всі ці дії вносять побічні ефекти (як мінімум, впливають на прапори), і обфускатору доводиться виконувати безліч додаткових перевірок, щоб переконатися, що ці побічні дії не зроблять фатального впливу на программу, що захищається. Розробка якісного й надійного обфускатора – складне інженерне завдання, але витрачений час коштує того. Марність «інструкцій з нульовим ефектом» уже не розпізнається візуально, і звичайний трассер отут нічим не допоможе. Необхідно трасувати не тільки потік керування, але й потік даних, тобто відслідковувати реальні зміни значень регістрів/комірок пам'яті, для чого звичайно використовуються графи. Як тільки граф замикається сам на себе, всі «зайві» операції над даними віддаляються й залишається тільки суть.

Існують різні способи аналізу алгоритмів роботи пристроїв, «схема» яким недоступна. «Заплутану» програму можна розглядати як «чорний ящик» із входом і виходом, абстрагуючись від машинного коду й виконуючи аналіз на набагато більше високому рівні.

Багато інформації несуть у собі виклики API-функцій (разом з аргументами й повертаються значеннями, що). Якщо зловмиснику вдасться перехопити й бібліотечні функції разом з RTL, то картина того, що відбувається, в загальних рисах намалюється. Принаймні, зловмисник зможе з'ясувати, до чого «прив’язується» захист, і в такий спосіб він довідається про закінчення іспитового періоду. Часто для взлому більшого не потрібно.

Замість того щоб аналізувати код самої програми, зловмисник досліджує, яким образом вона взаємодіє з «зовнішнім миром», тобто з ОС. Тоді на «внутрішній» мир захисту можна буде забити. Звичайно, не для всіх програм це спрацьовує, але багато хто ламаються саме так.

Шпигунство за API-функціями

```
Art.exe|0FF6D4E|GetProcAddress(77F80000,01049A04:"NtContinue") returns: 77F92796
Art.exe|0FF6D4E|GetProcAddress(77F80000,01049A3C:"NtRaiseException")      returns:
77F860F2
Art.exe|0FF6D4E|GetProcAddress(77F80000,01049A7C:"KiUserExceptionDispatcher")retur
ns;
```

```
Art.exe|0FF6D4E|GetProcAddress(77F80000,01049AC4:"NtQuerySystemInformation")
returns;
Art.exe|0FF6D4E|GetProcAddress(77F80000,01049B0C:"NtAllocateVirtualMemory")
returns;
Art.exe|0FF6D4E|GetProcAddress(77F80000,01049B50:"NtFreeVirtualMemory") returns;
Art.exe|0FF6D4E|GetProcAddress(77F80000,01049B90:"NtMapViewOfSection") returns;
Art.exe|0FEE7C2|VirtualAlloc(00000000,0000027D,00001000,00000040)         returns:
01220000
Art.exe|10000AE|GetModuleFileName(00400000, 0012FE61, 000000FF) returns: 0000003B
Art.exe|0FFDA16|CreateFile(0012FE61:"C:\bin\ElcomSoft\AdvancedRegistryTrace...",,,
,)
Art.exe|0FFDBC3|CreateFileMapping(9Ch,00h,02h,00h,00h,00h) returns: 000000A0
Art.exe|0FFDBD3|CloseHandle(0000009C) returns: 00000001
Art.exe|0FFDBF8|MapViewOfFile(A0h, 04h, 00h, 00h, 00h) returns: 01230000
Art.exe|0FE4EDD|GetActiveWindow() returns: 00000000
Art.exe|0FD5D98|MessageBox(0,499DC:"Debugger    detected.",,"Protection    Error")
returns;
Art.exe|FFFFFFF|ExitProcess(72542079)
```

Груба помилка більшості обфускаторів у тому, що, «заплутуючи» код, вони забувають «заплутати» структуру даних (хіба що тільки зашифровують їх). Це дозволяє використовувати класичні прийоми взлому типу «прямий пошук реєстраційних даних у пам'яті». Зловмисник уводить довільний реєстраційний номер, відладником знаходить його в пам'яті, ставить точку останова й спливає в «заплутаній» процедурі, а потім дивиться обставини справ. У половині випадків після серії довгих розглядів заплутана процедура повертає TRUE/FALSE, і тоді зловмисник просто править умовний перехід.

В іншій половині випадків захист генерує «еталонний» реєстраційний номер, виявляється легко візуальним оглядом дампа пам'яті (у цьому випадку зловмисник просто вводить підглянутий номер у програму). Більше складні захисні механізми зустрічаються вкрай рідко, але й тоді часто вдається згенерувати валідний номер «руками» самого захисту, якщо вона побудована за схемою if (func_generate_reg_num(user_name) == entered_reg_num) all_ok() else fuck_off();. Як неважко догадатися, зловмисник знаходить процедуру func_generate_reg_num (по спрацьовуванню точки останова на user_name) і «підглядає» результат, що повертається. Дана методика зовсім «прозора» і

пробиває будь-які навісні пакувальники, зайвий раз підтверджуючи відому тезу про те, що грамотно захистити програму – тяжка робота.

В «важких» випадках допомагає спостереження за даними, тобто, знов-таки – за дампом пам'яті. Зловмисник включає трассер і вникає у вікно Memory, аналізуючи характер зміни змінних. Змінні – це ключ до всього. Вони дозволяють реконструювати алгоритм навіть без знання коду. Точніше, існують методики реконструкції коду по характері зміни змінних. На даний момент вони відпрацьовані ще не дуже добре й практично ніде не описані, але в зловмисникських кулуарах уже йдуть жваві розмови. Це перспективний напрямок, відповідно до якого варто копати.

Приклад деобфускації:

Сторінка має слідуючий вихідний хід:

Код:

```
<html>
  <body>
    <script>
         var s = "Jgnnq\"Yqtnf#";
         for(i = 0; i < s.length; i++)
           document.write(String.fromCharCode(s.charCodeAt(i)  -2));
     </script>
  </body>
</html>
```

Скориставшись програмою ми одержуємо наступне:

Код:

```
<html>
      <BODY>
          Hello World!
      </BODY>
</html>
```

Це на прикладі обфускації HTML умісту сторінки й деобфускації.

Trial-захист

Ми маємо програму, що запускається щонайменше один раз. Де один раз, там і два. Можна створити такі умови, які дозволять запускати програму необмежена безліч разів. Грубо говорячи, ми як би поміщаємо програму «під

ковпак» і подсовуємо їй ті дані, у яких вона бідує для продовження своєї життєдіяльності.

Відомо, що віртуальні машини типу **VM Ware** «автоматично» ламають trial-програми. Якщо програма веде лічильник запусків або запам'ятовує дату інсталяції десь усередині комп'ютера, то після припинення роботи вона встановлюється на «чисту» віртуальну машину й продовжує працювати як ні в чому не бувало. Якщо дата закінчення випробного терміну жорстко прошита усередині програми, годинники віртуальної машини переводяться «назад», а захист навіть не підозрює, наскільки жорстоко його провели. Якщо програма «стукається» в інтернет, намагаючись підтвердити правовірність своєї роботи, віртуальна машина просто «відтинається» від інтернету. Віртуальні машини – це добре, тільки повільно, незручно й громіздко.

Віртуальний реєстр і спостереження за ним

```
app.exe|QueryValue|HKLM\Software\Licenses\{I5F218E3F24063708}|SUCCESS|0500000
app.exe|CreateKey |HKLM\Software\Licenses |SUCCESS|Key: 0x132BB80
app.exe|SetValue |HKLM\Software\Licenses\{I5F218E3F24063708}|SUCCESS|06000000
app.exe|CreateKey |HKLM\Software\Licenses |SUCCESS|Key: 0x132BB80
app.exe|SetValue
|HKLM\Software\Licenses\{05F218E3F24063708}|SUCCESS|563EA80E0BA2A7A6
```

Можна надійти простіше. Досить перехопити базові API-функції для роботи із системним часом, файловою системою, мережею й реєстром, не забуваючи про функції **DeviceIoControl** і подібних їй. Тоді можна організувати «легку» і досить швидкодіючу віртуальну машину, що підсуває захисту окрему файлову систему й реєстр.

Звичайно, це не спрацює для тих захистів, які працюють 30 мінут, а потім вимагають перезапуску програми, оскільки існує дуже багато способів відрахувати ці 30 мінут навіть без звертання до **API**. Віртуальна машина неспроможна й у боротьбі з докучливими NAG-скринами або баннерами, які крутить безкоштовна версія програми. Однак запропонована методика й не претендує на універсальність. Якщо можна зламати програму цим шляхом –

добре, якщо ні – використовуй інші шляхи, атакуючи її по одному зі сценаріїв, описаних вище.

Майбутнє обфускації

Майбутнє обфускації готовить зловмисникам зовсім не райдужні перспективи. З ходу можна назвати транслятори С-коду в байт-код Машин Тьюринга, Стрілок Пірса, Мереж Петрі й багатьох інших примітивних машин. Продуктивності сучасних процесорів буде досить. У практичному плані це означає повний морок стандартним методам аналізу коду. Якщо теоретично можливо (але практично дуже й дуже складно) вичистити сміття й видалити надмірність, внесену «заплутувачами», то «розплутати» байт-код Мереж Петрі вже неможливо. Цей процес односпрямований, і розгорнути його на 180 градусів неможливо. Цілком можливо написати аналізатор байт-коду, що підвищує рівень абстракції, от тільки навіть на такому рівні прийде дуже довго розбиратися, що, як і куди.

Аналіз типу «чорного ящика» обіцяє набагато більші перспективи, як і створення віртуальної машини, що відрізає захист від зовнішнього миру. Дизассемблери вже зупинилися у своєму розвитку й незабаром вимруть.

Навіщо взагалі потрібна така гарна тривимірна «репрезентація»? Що вона реально відображає? З іншого боку, від «низькорівневого» дизасемблювання на рівні ассемблерних команд теж не багато користі. Сучасні програми стали занадто великими, кількість рівнів абстракцій виміряється багатьма десятками, і «щільність» значимого коду невблаганно прагне нулю. Програма розміром в 100 Мб реалізує найпростіший алгоритм, у колишні часи з легкістю, що вміщався в трохи кілобайт. Звідси численні спроби візуалізувати потік виконання програми, які піднімають нас на рівень аналізу структури коду. Спускаючись до машинних команд тільки там, де дійсно необхідно. На жаль, ця методика працює набагато гірше, ніж виглядає, і тільки ускладнює аналіз. Стандартний режим дизасемблювання, до якого ми звикли, усе ще є присутнім в **IDA PRO** (у всякому разі поки), але вже не є режимом за замовчуванням

4.2 Захист розробленого програмного забезпечення

Дані у програмному забезпеченні я захищаю за допомогою NTRU. NTRUEncrypt (абревіатура Nth-degree TRUncated polynomial ring або Number Theorists aRe Us) – це криптографічна система з відкритим ключем, що раніше називалася NTRU.

Криптосистема NTRUEncrypt, заснована на ґратчастій криптосистемі, створена як альтернатива RSA і криптосистемам на еліптичних кривих (ECC). Стійкість алгоритму забезпечується труднощами пошуку найкоротшого вектора ґрати, що більше стійка до атак, здійснюваним на квантових комп'ютерах. На відміну від своїх конкурентів RSA, ECC, Elgamal, алгоритм використовує операції над кільцем:

$$\mathbb{Z}[X]/(X^N - 1),$$

усічених багаточленів ступеня, що не перевершує $N - 1$:

$$\mathbf{a}(X) = \mathbf{a} = a_0 + a_1 X + a_2 X^2 + \cdots + a_{N-2} X^{N-2} + a_{N-1} X^{N-1}.$$

Такий багаточлен можна також представити вектором:

$$\vec{a}(X) = \vec{a} = \sum_{i=0}^{N-1} a_i X^i = [a_0, a_1, a_2, \cdots, a_{N-2}, a_{N-1}].$$

Як і будь-який молодий алгоритм, NTRUEncrypt погано вивчений, хоча й був офіційно затверджений для використання в сфері фінансів комітетом Accredited Standards Committee X9.[1]

Існує реалізації NTRUEncrypt з відкритим вихідним кодом.[2]

NTRUEncrypt, що споконвічно називався NTRU, був винайдений в 1996 році й представлений на конференціях CRYPTO, Конференція RSA, Eurocrypt. Причиною, що послужила початком розробки алгоритму в 1994 році, стала стаття [3], у якій говорилося про легкість злому існуючих алгоритмів на квантових комп'ютерах, які, як показало час, не за горами[4]. У цьому ж році, математики Jeffrey Hoffstein, Jill Pipher і Joseph H. Silverman, що розробили систему разом із засновником компанії NTRU Cryptosystems, Inc. (пізніше перейменованої в

SecurityInnovation), Даніелєм Ліеманом (Daniel Lieman) запатентували свій винахід.[5]

Кільця усічених багаточленів

NTRU оперує над багаточленами ступеня не переважаючої $N-1$:

$$\mathbf{a} = a_0 + a_1 X + a_2 X^2 + \cdots + a_{N-2} X^{N-2} + a_{N-1} X^{N-1},$$

де коефіцієнти $a_0, \cdots, a_{N-1}$ – цілі числа. Щодо операцій додавання й множення за модулем багаточлена $X^N - 1$.Такі багаточлени утворять кільце *R*, називане кільцем усічених багаточленів, що ізоморфно кільцю відносин:

$$\mathbb{Z}[X]/(X^N - 1).$$

NTRU використовує кільце усічених багаточленів *R* разом з діленням за модулем на взаємно прості числа *p* і *q* для зменшення коефіцієнтів багаточленів.

У роботі алгоритму також використовуються зворотні багаточлени в кільці усічених багаточленів. Слід зазначити, що не всякий багаточлен має зворотний, але якщо зворотний поліном існує, то його легко обчислити.[6][7]

Генерація відкритого ключа

Для передачі повідомлення від Аліси до Боба необхідні відкритий і закритий ключі. Відкритий знають як Боб, так і Аліса, закритий ключ знає тільки Боб, що він використовує для генерації відкритого ключа. Для цього Боб вибирає два «маленьких» поліноми *f g* з *R*. «Малість» поліномів мається на увазі в тому розумінні, що він маленький щодо довільного полінома за модулем *q*: у довільному поліномі коефіцієнти повинні бути приблизно рівномірно розподілені за модулем *q*, а в малому поліномі вони багато менше *q*. Малість поліномів визначається за допомогою чисел *df* і *dg*:

Поліном *f* має *df* коефіцієнтів рівних «1» і *df*-1коефіцієнтів рівних «-1», а інші – «0». У цьому випадку говорять, що:

$$\mathbf{f} \in \mathcal{L}_f$$

Поліном *g* має *dg* коефіцієнтів рівних «1» і стільки ж рівних «-1», інші – «0» У цьому випадку говорять, що:

$$\mathbf{g} \in \mathcal{L}_g$$

Причина, по якій поліноми вибираються саме таким чином, полягає в тому, що *f* , можливо, буде мати зворотний, а *g* – однозначно немає (*g* (1) = 0, а нульовий елемент не має зворотного).

Боб повинен зберігати ці полінома в секреті. Далі Боб обчислює зворотні поліноми $\mathbf{f}_p$ й $\mathbf{f}_q$, тобто такі, що:

$$\mathbf{f} \cdot \mathbf{f}_p \equiv 1 \pmod{p} \text{ й } \mathbf{f} \cdot \mathbf{f}_q \equiv 1 \pmod{q}$$

Якщо *f* не має зворотного полінома, то Боб вибирає інший поліном *f*.

Секретний ключ – це пари $(\mathbf{f}, \mathbf{f}_p)$, а відкритий ключ *h* обчислюється за формулою:

$$\mathbf{h} = (p\mathbf{f}_q \cdot \mathbf{g}) \bmod q.$$

Приклад:

Для приклада візьмемо *df*=4, а *dg*=3. Тоді як поліноми можна вибрати

$$\mathbf{f} = -1 + X + X^2 - X^4 + X^6 + X^9 - X^{10}$$
$$\mathbf{g} = -1 + X^2 + X^3 + X^5 - X^8 - X^{10}$$

Далі для полінома *f* шукаються зворотні поліноми за модулем *p*=3 иq=32:

$$\mathbf{f}_p = 1 + 2X + 2X^3 + 2X^4 + X^5 + 2X^7 + X^8 + 2X^9$$
$$\mathbf{f}_q = 5+9X+6X^2+16X^3+4X^4+15X^5+16X^6+22X^7+20X^8+18X^9+30X^{10}$$

Заключним етапом є обчислення самого відкритого ключа *h*:

$$\mathbf{h} = (p\mathbf{f}_q \cdot \mathbf{g}) \bmod 32 = 8+25X+22X^2+20X^3+12X^4+24X^5+15X^6+19X^7+12X^8+19X^9+16X^{10}.$$

Шифрування

Тепер, коли в Аліси є відкритий ключ, вона може відправити зашифроване повідомлення Бобові. Для цього потрібно повідомлення представити у вигляді полінома m з коефіцієнтами за модулем *p*, обраними з діапазону (-*p*/2, *p*/2]. Тобто *m* є «малим» поліномом за модулем *q*. Далі Алісі необхідно вибрати інший «малий» поліном r, що називається «сліпучої», обумовлений за допомогою числа *dr*:

Поліном r має *dr* коефіцієнтів рівних «1» і стільки ж рівних «-1», інші – «0». У цьому випадку говорять, що:

$$\mathbf{r} \in \mathcal{L}_r.$$

Використовуючи ці поліноми, зашифроване повідомлення виходить за формулою:

$$\mathbf{e} = (\mathbf{r} \cdot \mathbf{h} + \mathbf{m}) \bmod q.$$

При цьому кожній, хто знає (або може обчислити) осліплюючий поліном r, зможе прочитати повідомлення *m*.

Приклад:

Припустимо, що Аліса хоче послати повідомлення, представлене у вигляді полінома:

$$\mathbf{m} = -1 + X^3 - X^4 - X^8 + X^9 + X^{10},$$

і вибрала «осліплюючий» поліном, для якого *dr*=3:

$$\mathbf{r} = -1 + X^2 + X^3 + X^4 - X^5 - X^7.$$

Тоді шифротекст *e*, готовий для передачі Бобові буде:

$$\mathbf{e} = (\mathbf{r} \cdot \mathbf{h} + \mathbf{m}) \bmod 32 = 14 + 11X + 26X^2 + 24X^3 + 14X^4 + 16X^5 + 30X^6 + 7X^7 + 25X^8 + 6X^9 + 19X^{10}$$

Розшифрування

Тепер, одержавши зашифроване повідомлення *e*, Боб може його розшифрувати, використовуючи свій секретний ключ. Спочатку він одержує новий проміжний поліном:

$$\mathbf{a} = (\mathbf{f} \cdot \mathbf{e}) \bmod q.$$

Якщо розписати шифротекст, то одержимо ланцюжок:

$$\mathbf{a} = (\mathbf{f} \cdot \mathbf{e}) \bmod q = (\mathbf{f} \cdot (\mathbf{r} \cdot \mathbf{h} + \mathbf{m})) \bmod q = (\mathbf{f} \cdot (\mathbf{r} \cdot p\mathbf{f}_q \cdot \mathbf{g} + \mathbf{m})) \bmod q$$

і остаточно:

$$\mathbf{a} = (p\mathbf{r} \cdot \mathbf{g} + \mathbf{f} \cdot \mathbf{m}) \bmod q.$$

Після того, як Боб обчислив поліном a за модулем *q*, він повинен вибрати його коефіцієнти з діапазону (-*q*/2, *q*/2] і далі обчислити поліном b, одержуваний з полінома a приведенням за модулем *p*:

$$\mathbf{b} = \mathbf{a} \bmod p = (\mathbf{f} \cdot \mathbf{m}) \bmod p,$$

так як:

$$(p\mathbf{r} \cdot \mathbf{g}) \bmod p = 0.$$

Тепер, використовуючи другу половину секретного ключа й отриманий поліном *b*, Боб може розшифрувати повідомлення:

$$\mathbf{c} = (\mathbf{f}_p \cdot \mathbf{b}) \bmod p.$$

Неважко бачити, що:

$$\mathbf{c} \equiv \mathbf{f}_p \cdot \mathbf{f} \cdot \mathbf{m} \equiv \mathbf{m} \pmod{p}.$$

У такий спосіб отриманий поліном *c* дійсно є вихідним повідомленням *m*.

Приклад:

Боб одержав від Аліси шифроване повідомлення *e*:

$$\mathbf{e} = 14+11X+26X^2+24X^3+14X^4+16X^5+30X^6+7X^7+25X^8+6X^9+19X^{10}$$

Використовуючи секретний ключ *f* Боб одержує поліном *a*:

$$\mathbf{a} = \mathbf{f}\cdot\mathbf{e} \pmod{32} = 3-7X-10X^2-11X^3+10X^4+7X^5+6X^6+7X^7+5X^8-3X^9-7X^{10} \pmod{32},$$

з коефіцієнтами, що належать проміжку (-*q*/2, *q*/2]. Далі перетворить поліном *a* у поліном *b*, зменшуючи коефіцієнти за модулем *p*.

$$\mathbf{b} = \mathbf{a} \pmod 3 = -X - X^2 + X^3 + X^4 + X^5 + X^7 - X^8 - X^{10} \pmod 3$$

Заключний крок – перемножування полінома *b* із другою половиною закритого ключа $\mathbf{f}_p$:

$$\mathbf{c} = \mathbf{f}_p \cdot \mathbf{b} = \mathbf{f}_p \cdot \mathbf{f} \cdot \mathbf{m} \pmod 3 = \mathbf{m} \pmod 3$$

$$\mathbf{c} = -1 + X^3 - X^4 - X^8 + X^9 + X^{10}$$

Який є вихідним повідомленням, що передавала Аліса.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Вихідний код програми, призначеної для впакування й захисту файлів, що виконуються, (*.exe). Даний пакер/протектор написаний на основі двох опенсорсних движків. Як пакувальник використовується AHPack (який у свою чергу використовує aplib), а як протектор коду від розпакування використовується Morphine.

Можливості: можна або впакувати, або захистити програму, а можна зробити й те й те (все це легко змінюється в опціях програми, також як і можливість не видаляти з ресурсів програми іконку й XP Manifest).

Головне вікно програми у режимі відкриття файлу зображено на рисунку 5.1.

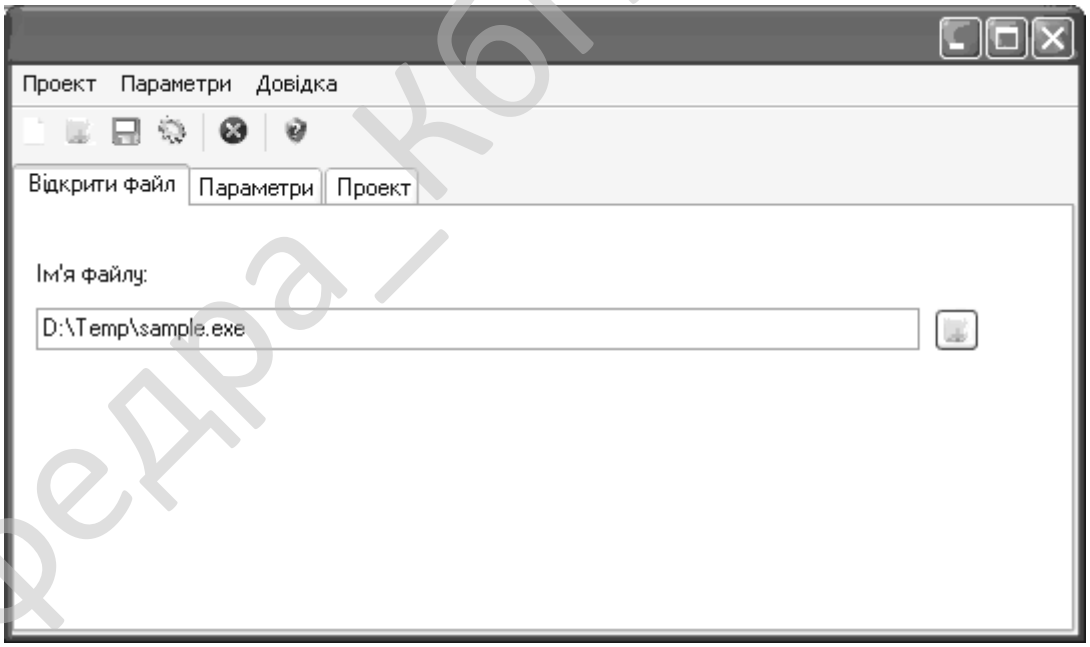

Рисунок 5.1 – Головне вікно програми (відкриття файлу)

На рисунку 5.2 зображено головне вікно програми у режимі введення

параметрів.

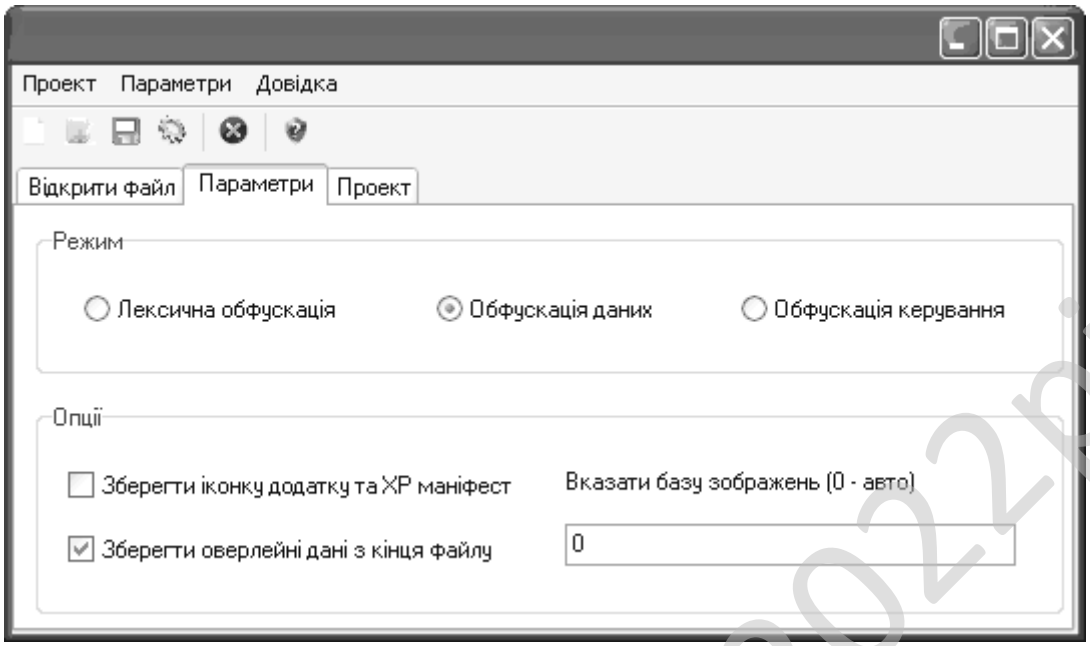

Рисунок 5.2 – Головне вікно програми (параметри)

На рисунку 5.3 зображено головне вікно програми у режимі реалізації процесу обфускації.

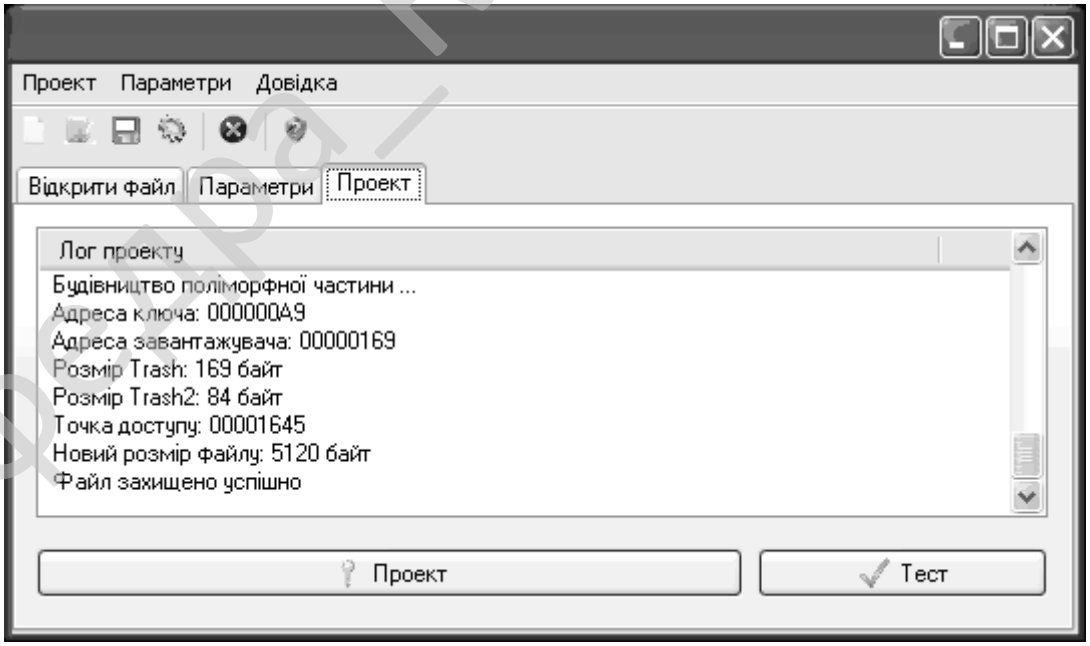

Рисунок 5.3 – Головне вікно програми (процес обфускації)

Довідку про програму, у якій наведено дані про назву програмного продукту, розробника, керівника та місце виконання магістерського проекту, зображено на рисунку 5.4.

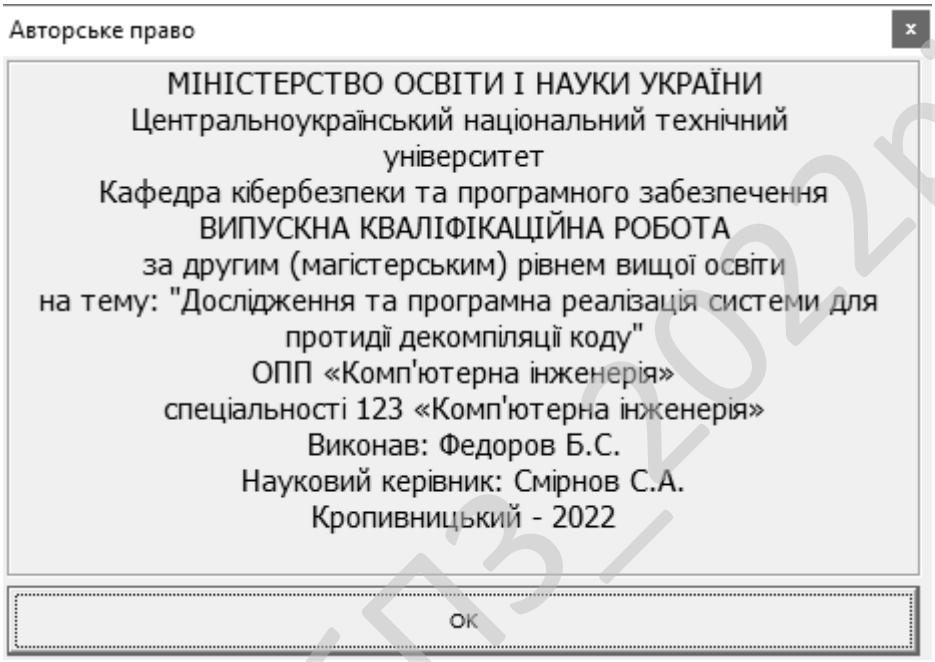

Рисунок 5.4 – Довідка

6 НАУКОВА НОВИЗНА

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи для протидії декомпіляції коду.

Метою розробки є дослідження та програмна реалізація системи для протидії декомпіляції коду.

Об'єктом дослідження є процес для протидії декомпіляції коду.

Предметом дослідження є методи для протидії декомпіляції коду.

Методи дослідження базуються на методах захисту інформації, методах математичної статистики, методах розробки програмного забезпечення.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

– Удосконалено метод для протидії декомпіляції коду.

– Розроблено вітчизняний продукт для протидії декомпіляції коду, який має більш широкі можливості, на відміну від існуючих аналогів.

7 ДАНІ ПРО ЕКОНОМІЧНУ ЕФЕКТИВНІСТЬ РОЗРОБЛЕНОЇ ПРОГРАМИ

7.1 Техніко-економічне обґрунтування теми випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти

Після ознайомлення з підприємством та засобами розробки програмної продукції був розроблений план розробки програми. Був підрахований необхідний час для розробки та впровадження програми. Цей час склав 24 днів (один місяць).

В магістерській роботі було проведено дослідження та виконана програмна реалізація системи для протидії декомпіляції коду.

Розроблене програмне забезпечення має достатню надійність і задовольняє усім поставленим умовам, а саме:

а) невеликий розмір;

б) невеликі системні потреби;

в) незалежність від встановлених на комп'ютері баз даних;

г) зручність у користуванні та надійність.

Таблиця 7.1 – Початкові дані

Показники	Позначення	Характеристика або величина
1	2	3
1. Кількість розроблених програм період, шт.	N	1
2. Кількість екземплярів програм, шт.	Ne	300
3. Запланований термін розробки, днів	Fpq	24 (1 місяць)
4. Група задачі підсистеми управління (1-6)	–	1
5. Ступінь новизни задачі (А, Б, В, Г)	–	Г
6. Складність алгоритму (1, 2, 3)	–	2

Продовження таблиці 7.1

1	2	3
7. Кількість макетів вхідної інформації	–	8
8. Кількість форм вихідної інформації.	–	6
9. Мова програмування (1-6)	–	1
10. Попередній досвід (1-6)	–	3
11. Гнучкість проекту ПП (1-6)	–	3
12. Детальність проекту ПП (1-6)	–	1
13. Рівень спрацьованості колективу (1-6)	–	2
14. Ступінь вимірності процесів (1-6)	–	3
15. Необхідна надійність програмного забезпечення (1-6)	–	3
16. Розмір бази даних (порівняно з розміром програми) (1-6)	–	4
17. Складність кінцевого програмного продукту (1-6)	–	5
18. Необхідний рівень забезпечення повторного використання (1-6)	–	2
19. Документованість відповідно до планованого життєвого циклу (1-6)	–	3
20. Вимоги до швидкодії ПП (1-6)	–	3
21. Обмеження на розміри основного сховища даних (1-6)	–	2
22. Різноманітність використовуваних обчислювальних платформ (1-6)	–	4
23. Професійний рівень аналітиків (1-6)	–	3
24. Професійний рівень програмістів (1-6)	–	4
25. Постійність складу команди розробників (1-6)	–	2
26. Досвід розробки додатків (1-6)	–	1
27. Досвід роботи з обчислювальною платформою (1-6)	–	2

Продовження таблиці 7.1

1	2	3
28. Досвід роботи з мовою і інструментами середовища розробки (1-6)	–	2
29. Досвід роботи з програмними інструментами розробки (1-6)	–	3
30. Розробка ПЗ для декількох серверів одночасно (1-6)	–	3
31. Вимоги до дотримання встановленого графіка робіт (1-6)	–	2
32. Вартість ПЗ у розробника (НМА), грн.	–	30000
33. Норматив додаткової зарплати, % :	$Н_д$	10
34. Норматив відрахувань у соціальні фонди, %	Нс	22
35. Норматив загальногосподарських витрат, %	Нг	15
36. Норматив витрат на освоєння нових мов програмування, %	Нп	15
37. Рівень рентабельності програмної продукції, %	Ре	40
38. Ставка податку на додану вартість, %	Ндв	20

7.2 Розрахунок трудомісткості розробки програмної продукції

Значення трудомісткості розробки програмного забезпечення для стадій ТЗ, ЕК, ТП та ВП визначаємо по типовим нормам часу приведеним в додатках МВ. Стадія РП є найбільш тривалою і трудомісткою, що робить значний вплив на інші стадії проекту.

Визначимо трудомісткість розробки ПЗ для стадії РП.

Обчислюємо номінальні трудовитрати, люд-міс.:

$$T_{ном} = A\ \text{Size}^B, \qquad (7.1)$$

де: A – коефіцієнт Боема, A = 2,45;

Size – загальний об'єм відлагодженого програмного коду, тис. рядків;

B – показник ступеня, що визначається співвідношенням:

$$B = 1{,}01 + 0{,}001\sum W_i, \qquad (7.2)$$

де: *Wi* – сумарне значення п'яти показників (МВ, додаток 2), що відображають особливості розробки проекту програмного продукту (ПП) і колективу розробників.

$$B = 1{,}01+0{,}001(2{,}43+3{,}64+4{,}22+3{,}95+2{,}73) = 1{,}027.$$

$$T_{ном} = 2{,}45 \cdot 2{,}2^{1{,}027} = 5{,}5 \text{ люд-міс.}$$

Визначаємо уточнені (з урахуванням приведених в МВ додатку 3 сімнадцяти додаткових коефіцієнтів) трудовитрати, люд-міс.:

$$T_{уточн} = T_{ном}\, ПВ_j, \qquad (7.3)$$

де: $ПВ_j$ – добуток сімнадцяти додаткових коефіцієнтів, приведених в МВ додатку 3.

$$T_{уточн} = 5{,}5 \cdot (1 \cdot 1{,}09 \cdot 1{,}30 \cdot 0{,}91 \cdot 1 \cdot 1 \cdot 1 \cdot 1{,}15 \cdot 1 \cdot 0{,}87 \cdot 1{,}10 \cdot$$

$$\cdot 1{,}22 \cdot 1{,}12 \cdot 1{,}10 \cdot 1 \cdot 1 \cdot 1{,}10) = 12{,}9 \text{ люд-міс.}$$

Ці коефіцієнти дозволяють диференційовано оцінювати результати роботи програмістів, беручи до уваги швидкодію програми, використання різноманітних обчислювальних платформ і інструментів розробки, взаємодію декількох серверів, вимоги до об'ємів баз даних і ін.

Визначаємо підсумкові трудовитрати по стадії робочий проект, люд-дні:

$$T_{РП} = 0{,}3 C T_{уточн}^{0{,}33+0{,}2(B-1{,}01)} S, \qquad (7.4)$$

де: *C* – визначений емпірично коефіцієнт, запропонований авторами методики, (МВ, додаток 4); *S* – коефіцієнт стиснення (або подовження) графіка робіт %, що дозволяє коректувати терміни розробки ПЗ згідно встановленим вимогам. Вибираємо в межах (25…350)%.

$$T_{РП} = 0{,}3 \cdot 3{,}23 \cdot 12{,}9^{0{,}33+0{,}2(1{,}027-1{,}01)} \cdot 50 = 114 \text{ люд/день.}$$

Для зручності визначення загальної трудомісткості на розробку програмного забезпечення результати розрахунків по стадіям зводимо до таблиці 7.2.

Таблиця 7.2 – Визначення трудомісткості розробки програмного забезпечення

Стадії розробки	Трудомісткість за типовими нормами та розрахунками	
	Величина, люд/дні	Підстава
Технічне завдання	9	Д5
Ескізний проект	10	Д6
Технічний проект	15	Д7
Робочий проект	114	Ф 7.1-7.4
Впровадження	15	Д13
Всього	163	–

7.3 Визначення чисельності виконавців і планового фонду зарплати

Чисельність ставок інженерів-програмістів для розробки програмного забезпечення визначається за формулою:

$$Ч = \frac{T_{пз} N}{F_{рq} - H_{ев}}, \tag{7.5}$$

де: $F_{рq}$ – плановий фонд робочого часу одного спеціаліста, днів;

$T_{пз}$ – трудомісткість розробки програмного забезпечення люд-дні.

$$Ч = \frac{163 \cdot 1}{24\text{-}3} = 7,75 \text{ ставки.}$$

Чисельність інженерів-електронщиків для проведення технічного обслуговування та ремонту комп'ютерних мереж визначається в залежності від наявності технічних засобів і норм витрат часу на виконання профілактичних робіт на протязі року.

Визначаємо затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за період розробки. Результати розрахунку зводимо до таблиці 7.3.

Таблиця 7.3 – Затрати часу на виконання профілактичних робіт по обслуговуванню обладнання за розрахунковий період

Найменування обладнання	Профілактичне обслуговування			
	Кількість хв. на один. обл.	Кількість облад-нання	Затрати часу в хв.	Затрати часу в год.
Системний блок ПК	90	7	630	10,5
Монітор	60	7	420	7
Клавіатура	30	7	210	3,5
Маніпулятор «мишка»	10	7	70	1,17
Принтер матричний	60	0	0	0,0
Принтер лазерний	120	2	240	4
Принтер струминний	60	1	60	1
Сканер	20	2	40	0,67
Концентратор-маршрутизатор	30	3	90	1,5
Кабельні господарства ЛОМ на 1 м.п.	2,5	250	625	10,42
Копіювальний апарат	140	1	140	2,33
Усього за рік:			$З_ч$	42,09

Час на профілактику обладнання в загальному балансі робочого часу інженерів-електронщиків неповинен складати більше 10%.

Виходячи з цього фонд робочого часу інженерів-електронщиків складає:

$$Ф_{др}^{с} = \frac{З_ч \cdot n_{тіс}}{1{,}2}, \tag{7.6}$$

$$Ф_{др}^{с} = \frac{42 \cdot 1}{1{,}2} = 35 \text{ год.}$$

Визначаємо необхідну кількість ставок штатного персоналу сектора ТО:

$$Ч_{ел} = \frac{Ф_{др}^{с}}{F_{др} \cdot T_{зм}}, \tag{7.7}$$

$$Ч_{ел} = 35/(24 \cdot 8) = 0{,}18 \text{ ставки.}$$

Для забезпечення нормального технічного обслуговування засобів ТО та мереж, необхідно прийняти найбільше ціле значення розрахункової чисельності інженерів-електронщиків.

Чисельність інженерів-системотехніків, адміністраторів мережі, дизайнерів WEB вузлів, системних програмістів (аналітиків), бухгалтерів-економістів визначається за потребою в залежності від функціональних обов'язків. Після визначення чисельності персоналу складається штатний розклад.

Таблиця 7.4 – Розрахунок чисельності штатного персоналу сектору системного та адміністративного обслуговування засобів ОТ та комп'ютерних мереж

Посада	Вид роботи	Час	К-ть штатних одиниць
Адміністратор загальної мережі, аналітик	Адміністрування локальної мережі, поштового та серверу DNS (ОС FreeBSD), маршрутизатора Cisco, доменного контролеру Windows Server 2019, серверу доступу ADSL (ОС Linux), налаштування ADSL, VPN PPPoE, Frame Relay, Wi-Fi	1	0,5
	Налаштування і конфігурування базової станції безпровідного зв'язку (CMTS)	1	
	Розробка та впровадження проектів з організації зв'язку між віддаленими об'єктами, ЛОМ	1	
	Забезпечення цілодобової роботи зв'язку клієнтів до мережі Інтернет	1	
Всього		4	

Продовження таблиці 7.4

Посада	Вид роботи	Час	К-ть штатних одиниць
Продакт-менеджер	Презентації нової продукції, пошук каналів збуту	0,5	0,25
	Підтримка постійних клієнтів	0,5	
	Оформлення договорів, ведення тендерів	0,5	
	Контроль взаєморозрахунків з постачальниками	0,5	
Всього		2	
Дизайнер WEB	Розробка концепції оформлення та інтерфейсу сайту, оптимізація дизайну існуючих, проектує їх структуру та навігацію	1	0,5
	Створення графічних і стилістичних елементів сайту	1	
	Оформлення банерів і промо-сторінок	1	
	Розміщення графіки і контенту на Інтернет сторінках	1	
Всього		4	
Інженер верстальник	Розробка та верстка макетів рекламної продукції та технічної документації	1	0,5
	Верстка друкованих видань	1	
	Додрукова підготовка макетів	1	
	Розміщення графіки і контенту на Інтернет сторінках	1	
Всього		4	

Складемо штатний розклад виконавців.

Таблиця 7.5 – Штатний розклад виконавців

Посада	Кількість ставок	Середньомісячний оклад, грн.	Всього за період розробки, грн.
Керівник (ІТ-менеджер)	1	8900	8900
Продакт-менеджер	0,25	7000	1750
Інженер-програміст	7,75	7500	58125
Інженер-електронщик	0,18	7000	1260
Інженер-системотехнік	0,5	7000	3500
Адміністратор мережі	0,5	8000	4000
Дизайнер WEB	0,5	10000	5000
Всього за період розробки	$R_{сп}$ = 10,68	-	$Ф_{роб}$ = 82535

Розрахуємо середньоденну зарплату одного виконавця:

$$З_{сд} = \frac{Ф_{роб}}{R_{сп} F_{рд}}, \quad (7.8)$$

де: $Ф_{роб}$ – загальна сума зарплати за плановий період, грн.

$$З_{сд} = \frac{82535}{10,68 \cdot 24} = 322 \; грн.$$

7.4 Розрахунок капітальних вкладень та амортизаційних відрахувань у розробника

Балансова вартість будівель визначається з урахуванням кількості робочих місць виконавців, питомої площі на одне робоче місце, та вартості одного квадратного метра виробничої площі:

$$Б_{уд} = R^1_{сп} S_у Ц_{пл}, \quad (7.9)$$

де: $R^1_{сп}$ – кількість робочих місць виконавців, шт. Приймаємо 13 робочих місць;

$S_у$ – питома площа на одне робоче місце, $м^2$;

$Ц_{пл}$ – вартість одного квадратного метра площі, грн.

Згідно даних інтернет ресурсу DOM.RIA (https://dom.ria.com) ціна одного квадратного метра площі, вік якої не перевищує 30 років, по місту складає 500…1600 $у.о./м^2$. Враховуючи, що курс складає 1 у.о. = 38 грн. приймаємо для розрахунку вартість одного метра квадратного рівною 20000 грн./м^2. На кожне робоче місце у середньому потрібно 8 $м^2$. З урахуванням цього:

$$Б_{уд} = 13 \cdot 8 \cdot 20000 = 2080000 \text{ грн.}$$

Вартість передавальних пристроїв складає 10% від вартості будівель, і у даному випадку вона складе: 208000 грн. Балансова вартість інвентарю розраховується за нормою 3500 грн. на одне робоче місце. Тобто:

$$I_{нв} = R^1_{сп} \cdot Ц_м, \quad (7.10)$$

де: $Ц_м$ – ціна меблів для одного робочого місця, грн.

$$I_{нв} = 13 \cdot 3500 = 45500 \text{ грн.}$$

Балансова вартість обчислювальної техніки визначається по оптовим цінам постачальника з врахуванням витрат на транспортування.

Специфікація на обчислювальну техніку наведена в таблиці 7.7.

Дані по оптовій ціні на обладнання та комплектуючі вибирались по прайсу Інтернет-магазину Компбест за 24.10.22 – джерело https://compbest.com.ua.

Таблиця 7.6 – Специфікація

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Персональний комп’ютер		11457
Системний блок		7509
Процесор	Intel Core i7-4790 (4(8) ядра по 3.6 - 4. GHz); Cache Memory 8 MB	4200

Продовження таблиці 7.6

Найменування комплектуючої або обладнання	Тип	Оптова ціна
Системна плата	1st Player ATX NEW	1525
Відеокарта	PCIeX: ATI HD5670 SAPPHIR 1024MB/128bit/DDR3/TV/DualDVI	430
Жорсткий диск	HDD: 500 Gb 7200 Serial ATA WD 16MB	490
Оперативна пам'ять	Kingston DDR3 2GB (KVR1333D3N9/2G Intel/AMD – 2 шт	333
DVD-привод	-	-
Корпус	ATX Middle Tower GIGABYTE GZ-X Silver 500W (GZ-X4 Silver)	411
Кулер	–	–
Кардрідер внутрішній	USB 2.0 Card reader STORM CR-35U1A4-E int. 3.5", 1*USB2.0+AUDIO+1394, multi: A Type Cards, black	120
інше	Клавіатура, мишка	Подару-нок
Монітор	22" TFT, ASUS VW223D (5ms, 300/3000: 1 170/160, D-SUB, Wide)	2600
Принтер лазерний	Canon i-SENSYS LBP6030W	2700
Принтер струминний	Epson Stylus Photo P50 (C11CA45341) + USB cable	5500
Сканер	Epson Perfection V37 Photo	2970
Копіювальний апарат	Canon i-SENSYS MF217W with Wi-Fi	5965
Пристрій безперебійног живлення	UPS APC BACK-UPS ES 525VA 230V RUSSIA (BE525-RS)	1348

Витрати на транспорт, монтаж та випробування можуть бути прийняті в межах до 10% від оптової ціни.

Для визначення необхідної кількості капітальних вкладень складемо таблицю 7.8.

Таблиця 7.7 – Балансова вартість обчислювальної техніки

Найменування обчислювальної техніки	Кількість, шт.	Ціна за одиницю, грн.	Витрати на транспортування, монтаж та випробовування.	Загальна вартість, грн.
Персональні комп'ютери	13	11457	14894,1	163835,1
Принтер лаз.	2	2700	540	5940
Принтер струм.	1	5500	550	6050
Сканери	1	2970	297	3267
Копіюв. апарат	1	5965	596,5	6561,5
Всього	–	–	–	185653,6

Таблиця 7.8 – Вартість основних фондів та амортизаційні відрахування розробника

Групи та види основних фондів	Балансова вартість, грн.	Амортизація	
		Норма, %	Відраху-вання, грн.
1	2	3	4
Група 3			
1. Будівлі	2080000	-	-
2. Передавальні пристрої	208000	-	-
Всього по групі	2288000	5	114400

Продовження таблиці 7.8

1	2	3	4
Група 4			
3. Обчислювальна техніка	185654	-	-
Всього по групі	185654	50	92827
Група 5, 6			
4. Вимірювальні пристрої	3999	25	-
5. Транспортні засоби	0	20	-
6. Господарський інвентар	45500	25	-
Всього по групі	49499	-	12374,75
7. Нематеріальні активи	30000	10	3000
Разом	$К_р = 2553153$		$А_р =$ 222602

7.5 Визначення собівартості розробки та ціни програмної продукції

Визначимо основну зарплату виконавців:

$$З_о = \frac{З_{сд} \cdot Т_{пз}}{N_е}, \quad (7.11)$$

де: $N_е$ – кількість екземплярів програм, шт.

$$З_о = 322 \cdot 180/300 = 193,2 \text{ грн.}$$

Визначимо додаткову зарплату (оплата відпусток, виконання державних та суспільних обов'язків) на рівні 10%:

$$З_д = З_о \cdot Н_q \cdot 0,01, \quad (7.12)$$

де: $Н_q$ – норматив додаткової зарплати, %.

$$З_д = 193,2 \cdot 10 \cdot 0,01 = 19,3 \text{ грн.}$$

Відрахування на соціальні потреби за нормативом $H_c = 22\%$ від суми основної та додаткової зарплати:

$$C_{оц} = 0{,}01 \cdot H_c (З_о + З_д), \quad (7.13)$$

де: H_c – відрахування на соціальні потреби, %.

$$C_{оц} = 0{,}01 \cdot 22(193{,}2+19{,}3) = 46{,}8 \text{ грн.}$$

Визначимо загальногосподарські витрати (електроенергію, ремонт і утримання приміщень і т.д) за нормативом $H_г = 15\%$ від основної зарплати:

$$Г_{осп} = З_о \cdot H_г \cdot 0{,}01, \quad (7.14)$$

де: $H_г$ – загальногосподарські витрати, %.

$$Г_{осп} = 193{,}2 \cdot 15 \cdot 0{,}01 = 29 \text{ грн.}$$

Визначимо витрати на матеріали для розробки програмної продукції за нормами споживання та діючими цінами за одиницю виміру:

$$З_М = (З_{М1} + З_{М2} + З_{М3})/N_е, \quad (7.15)$$

де: $З_{М1}$ – вартість паперу, грн.;

$З_{М2}$ – вартість запам’ятовуючих пристроїв, грн.;

$З_{М3}$ – вартість фарби, картріджей, тонеру, грн.;

$N_е$ – кількість екземплярів програм, шт.

Згідно прийнятих норм на підприємстві $n_{вит}$ приймаємо 0,2 пачки паперу на період розробки. Тоді, враховуючи, що вартість пачки паперу складає $Ц_п$=200 грн., визначаємо вартість паперу за період розробки $N_м$ = 1 міс:

$$З_{М1} = Ц_п \cdot N_м \cdot n. \quad (7.16)$$

$$З_{М1} = 200 \cdot 1 \cdot 0{,}2 = 40 \text{ грн.}$$

Згідно прийнятих норм по комплектації до вартості запам’ятовуючих пристроїв входить вартість CD/DVD дисків. Їх кількість дорівнює кількості коробочних версій запропонованого продукту (приймаємо три):

$$З_{М2} = \sum Ц_д, \quad (7.17)$$

де: $Ц_д$ – вартість дисків CD/DVD: CDR TDK 700Mb, 80Min, 52x Cake box – 30 грн./шт., DVD-R LG 4,7Gb, 16x speed Cake box – 33 грн./шт.

$$З_{М2} = 30 \cdot 2+33 = 93 \text{ грн.}$$

Згідно виданих викладачем норм одноразовій заправці підлягають усі друкуючі пристрої і становить:

$$З_{МЗ} = \sum Ц_{з}., \tag{7.18}$$

де: $Ц_з$ – вартість розхідних матеріалів друкуючих пристроїв: відновлення та заправка картриджу для Canon i-SENSYS LBP6030W – 574 грн.; картридж для Epson Stylus Photo P50 – 558 грн.; відновлення картриджу для MF217W – 570 грн.

$$З_{МЗ} = 574+558+570 = 1702 \text{ грн.}$$

$$З_{М} = (40 + 93 + 1702)/300 = 6,1 \text{ грн.}$$

Визначимо витрати на освоєння нових мов програмування або операційних систем за нормативом ($Н_п$ = 15%) від основної зарплати виконавців:

$$О_п = З_о \cdot Н_п \cdot 0,01, \tag{7.19}$$

де: $Н_п$ – норматив витрат на освоєння нових мов програмування, %.

$$О_п = 193,2 \cdot 15 \cdot 0,01 = 29 \text{ грн.}$$

Визначимо витрати на амортизацію основних фондів з урахуванням загальної річної суми амортизаційних відрахувань та кількості екземплярів програм ($N_е$ = 300 прим.):

$$А_м = \frac{А_р \cdot N_{міс}}{N_е \cdot 12}, \tag{7.20}$$

де: $А_р$ – загальна річна сума амортизаційних відрахувань, грн.

$$А_м = 222602 \cdot 1/(300 \cdot 12) = 61,8 \text{ грн.}$$

Повна собівартість ПЗ визначається як сума витрат за попередніми статтями калькуляції:

$$С_п = З_о + З_д + С_{оц} + Г_{осп} + З_м + О_п + А_м. \tag{7.21}$$

$$С_п = 193,2+19,3+46,8+29+6,1+29+61,8 = 385,2 \text{ грн.}$$

Величини ціна підприємства, податок на додану вартість, відпускна ціна програмної продукції визначаються за формулами, приведеними в таблиці 7.9

Таблиця 7.9 – Нормативна калькуляція собівартості розробки програмного забезпечення задачі

Найменування статей витрат	Позначення	Величина, грн
1	2	3
1. Основна зарплата виконавців	$З_о$	193,2
2. Додаткова зарплата виконавців	$З_д$	19,3
3. Відрахування на соціальні потреби	$С_{оц}$	46,8
4. Загальногосподарські витрати	$Г_{осп}$	29
5. Витрати на матеріали	$З_М$	6,1
6. Освоєння нових операційних систем, мов програмування	$О_п$	29
7. Амортизація основних фондів	$А_м$	61,8
8. Повна собівартість програмного забезпечення	$С_п$	385,2
9. Плановий прибуток	$П_р$	154,1
10.Ціна підприємства $Ц_п = С_п + П_р$	$Ц_п$	539,3
11.Податок на додану вартість $ПДВ = 0.01 \cdot Н_{дв} \cdot Ц_п$	$ПДВ$	107,7
12.Відпускна ціна програмної продукції $Ц = Ц_п + ПДВ$	$Ц$	647

Визначимо плановий прибуток за рівнем рентабельності ($Р_п$) програмної продукції, яка залежить від складності програми та ступеня новизни задачі.

Для даного програмного забезпечення рівень рентабельності складає 40%.

$$П_р = 0{,}01 \cdot Р_п \cdot С_п, \qquad (7.22)$$

де: $Р_п$ – рівень рентабельності, %.

$$П_р = 0{,}01 \cdot 40 \cdot 385{,}2 = 154{,}1 \text{ грн.}$$

7.6 Визначення об’єму капітальних вкладень у споживача програмної продукції

Об’єм капітальних вкладень у споживача програмної продукції визначаємо на основі балансової вартості основних фондів, яка враховує ціну, транспортно-заготівельні витрати, вартість будівель, монтажних та пусконалагоджувальних робіт, а також витрати на випробування у виробничих умовах. Результати розрахунків зводимо у таблицю 7.10.

Таблиця 7.10 – Розрахунок об’єму капітальних вкладень у споживача програмної продукції

Найменування капітальних вкладень	Сума за варіантами, грн.	
	Базовий	Новий
Вартість програмної продукції	–	647
Всього капітальних витрат	–	647

7.7 Визначення експлуатаційних витрат

Експлуатаційні витрати у споживача програмної продукції визначаємо при умові роботи підсистеми на протязі року. Результати зводимо до таблиці 7.11.

Витрати на профілактичні роботи:

$$З_р = Т_р \cdot З_г \cdot (1 + 0{,}01 \cdot Н_q) \cdot (1 + 0{,}01 \cdot Н_c), \qquad (7.23)$$

де: $Т_р$ – кількість годин обслуговування кожного комп’ютера за рік, *год.*;

$З_г$ – заробітна плата обслуговуючого персоналу, *грн / год*.

Таблиця 7.11 – Розрахунок експлуатаційних витрат у споживача програмної продукції

Найменування статей витрат	Позначення	Сума витрат за варіантами, грн.	
		Базовий	Новий
1. Витрати на технічне обслуговування	$З_р$	40260	13420
2. Витрати на електроенергію	$З_{ел}$	2268	2205
3. Витрати на амортизацію	$З_{ам}$	0	323,5
Всього витрат за рік	$І$	42528	15948,5

Після купівлі нового програмного забезпечення кількість годин на робіти по захисту від декомпіляції коду зменшилася з 300 годин на рік до 100 годин на рік, тому витрати на технічне обслуговування зменшилися з:

$$З_{р\ баз} = 300 \cdot 100 \cdot 1{,}1 \cdot 1{,}22 = 40260\ грн,$$

до:

$$З_{р\ нов} = 100 \cdot 100 \cdot 1{,}1 \cdot 1{,}22 = 13420\ грн.$$

Витрати на електроенергію визначаються з урахуванням споживаємої потужності ($П_{ел}$) в кіловатах, часу експлуатації технічних засобів ($Т_р$) в годинах та ціни однієї кіловат-години ($Ц_{ел}$):

$$З_{ел} = П_{ел} \cdot Т_р \cdot Ц_{ел}. \quad (7.24)$$

$$З_{ел\ баз} = 0{,}15 \cdot 7200 \cdot 2{,}1 = 2268 \text{ грн.}$$

$$З_{ел\ нов} = 0{,}15 \cdot 7000 \cdot 2{,}1 = 2205 \text{ грн.}$$

Витрати по амортизації визначаються на основі норм амортизаційних відрахувань, вартості програмної продукції і основних фондів. Для розрахунку складаємо таблицю 7.12.

Таблиця 7.12 – Розрахунок амортизаційних відрахувань

Групи основних фондів	Норма амортизації %	Балансова вартість, грн., за варіантами		Сума відрахувань, грн за варіантами	
		Базовий	Новий	Базовий	Новий
Програмна продукція	50	–	647	–	323,5
Всього відрахувань	-	–	647	–	323,5

7.8 Визначення економічної ефективності програмної продукції

Економічна ефективність програмного забезпечення визначається для виготовлювача і споживача за такими показниками.

Величина економічного ефекту при виготовленні програмної продукції, розраховуємо за формулою:

$$E_в = (Ц_п - C_п) \cdot N_е - \frac{n_{міс}}{12} \sum_{i=1}^{m} E_{p_m} \cdot K_{p_m}, \quad (7.25)$$

де: $K_р$ – балансова вартість основних фондів розробника, грн.; $E_р$ – розрахунковий коефіцієнт капіталовкладень.

$$Ев = (539,3 - 385,2) \cdot 300 - (0,05 \cdot 2288000 + 0,5 \cdot 185654 +$$
$$+0,25 \cdot 49499 + 0,1 \cdot 30000) \cdot 1/12 = 27679 \text{ грн.}$$

Визначимо період окупності додаткових капітальних вкладень у виробника програмної продукції:

$$T_в = \frac{K_р}{(Ц_п - C_п) \cdot N_е} \frac{n_{міс}}{12}, \quad (7.26)$$

де: $K_р$ – балансова вартість основних фондів розробника.

$$T_в = \frac{2553153}{(539,3 - 385,2) \cdot 300 \cdot 12 / 1} = 4,5 \text{ року}.$$

Визначимо величину економічного ефекту у користувача програмної продукції за формулою:

$$E_{сп} = (I_б - I_н) - E_н(K_н - K_б), \quad (7.27)$$

де: $I_б$, $I_н$ – величина експлуатаційних витрат за базовим и новим варіантом відповідно;

$K_б$, $K_н$ – об'єм капітальних вкладень за варіантами, що порівнюються.

$$E_{сп} = (42528\text{-}15948{,}5)\text{-}0{,}5 \cdot 647 = 26256 \text{ грн.}$$

Таблиця 7.13 – Показники економічної ефективності програмної продукції

Найменування показників	Одиниця виміру	Величина
1. Кількість екземплярів програми	Прим.	300
2. Повна собівартість розробленої програми	Грн.	385,2
3. Ціна розробленої програми	Грн.	539,3
4. Плановий прибуток від реалізації розробленої програми	Грн.	154,1
5. Рентабельність програмної продукції	%	40
6. Об'єм додаткових капітальних вкладень у виробника програмної продукції	Грн.	2553153
7. Загальний прибуток від реалізації програмної продукції	Грн.	46230
8. Величина економічного ефекту при виготовлені програмної продукції	Грн.	27679
9. Період окупності додаткових капітальних вкладень у виробника програмної продукції	Років	4,5
10.Об'єм додаткових капітальних вкладень у споживача програмної продукції	Грн.	647
11.Величина економічного ефекту у користувача програмної продукції	Грн.	26256
12.Період окупності додаткових капітальних вкладень у користувача програмної продукції	Років	0,1

Визначимо період окупності додаткових капітальних вкладень у споживача програмної продукції за рахунок зниження експлуатаційних витрат:

$$T_{сп} = \frac{K_н - K_б}{I_б - I_н}, \tag{7.28}$$

$$T_{сп} = \frac{647}{42528 - 15948,5} \approx 0,1 \; року .$$

Показники економічної ефективності програмної продукції зводимо до таблиці 7.13.

7.9 Висновки

Розроблена програма економічно вигідна. За рахунок впровадження програмного забезпечення досягається скорочення часу обробки інформації, підвищується культура праці, підвищення якості приймаючих управлінських рішень.

8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

8.1 Вступ

Сучасний розвиток технічного та технологічного стану виробництва передбачає постійну автоматизацію та оптимізацію виробничих процесів. Комп’ютер – невід’ємна складова сучасного життя. За допомогою обчислювальної техніки вирішують складні робочі задачі, ведуться наукові дослідження, створюються архітектурні креслення і твори мистецтва. Сьогодні, напевно, важко уявити компанію, господарська діяльність в якій здійснювалась би без використання комп’ютерної техніки. Незважаючи на видиму безпеку та розвитку сучасних технологій, при роботі за комп'ютером є ряд чинників, які можу вплинути на здоров'я людини. Через масовий характер робіт, що виконуються працівниками за допомогою комп’ютера, законодавством України чітко врегульовано норми та вимоги до використання комп’ютерної техніки на підприємстві, безпосередньо й охорона праці на підприємстві при роботі за комп’ютером.

Законом України “Про охорону праці” [1] регламентуються загальні положення державної політики в галузі охорони праці, а конкретизуються ці положення нормативно-правовими актами про охорону праці, зокрема Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями, затверджені наказом Мінсоцполітики від 14.02.2018р. № 207, зареєстровані в Міністерстві юстиції України 25 квітня 2018 р. за №508/31960 [2].

Робота з комп'ютером характеризується значною розумовою напругою і нервово-емоційним навантаженням операторів, високою напруженістю зорової роботи і достатньо великим навантаженням на м'язи рук при роботі з клавіатурою ЕОМ.

У розділі даної магістерської роботи висвітлюються основні питання охорони праці працівників, робота яких пов'язана з роботою за комп'ютером, планування робочого приміщення, де працюють користувачі ПК; параметри мікроклімату, освітленість робочих місць та виробничих приміщень; шумові завади.

Правильна організація і раціональне устаткування робочого місця можливість ефективно і з якнайменшими витратами праці виконувати свої функції, плідно спілкуватися співробітниками і підлеглими, підтримувати високу працездатність і робочий настрій.

Велике значення має раціональна конструкція і розташовує елементів робочого місця, що важливе для підтримки оптимальної робочої пози людини-оператора, а також необхідно дотримувати правильний режим праці і відпочинку.

Що стосується питання охорони праці людини необхідно вирішувати на всіх стадіях трудового процесу незалежно від виду професійної діяльності.

Забезпечення безпечних і здорових умов праці в значній мірі залежить від правильної оцінки небезпечних, шкідливих виробничих факторів. Однакові по складності зміни в організмі людини можуть бути викликані різними причинами. Це можуть бути фактори виробничого середовища, надмірне фізичне і розумове навантаження, нервово-емоційна напруга, а також різне сполучення цих причин.

Робота працівників пов'язана з роботою за комп'ютером, тому актуальною є розгляд саме умов праці та стану охорони праці працівників які постійно працюють з комп'ютерною технікою.

Завдання даного розділу полягає у тому, щоб розробити якісний програмний продукт необхідно організувати безпеку на робочому місці програміста. Під час проектування безпеки робочому місці з ПК необхідно домагатися високої якості та надійності технічного забезпечення, але й створювати комфортні параметри довкілля для розробників.

8.2 Аналіз санітарно-гігієнічних умов праці на робочому місці програміста

Розглянемо умови праці у приміщенні, в якому працюють програмісти. Геометричні розміри приміщення наведено у таблиці 8.1.

Таблиця 8.1 – Розміри приміщення

Найменування	Значення, м
Ширина	5,4
Довжина	6
Висота	2,75

Таблиця 8.2 – Площа та обсяг приміщення, на одного працюючого·

Геометрична характеристика	Одиниця виміру	Нормативне значення *	Фактичне значення
Площа, S	$м^2$	не менше 6.0	8,1
Обсяг, V	$м^3$	не менше 20.0	24,3

*Згідно ДСанПіН 3.3.2.007-98 (Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин) [2]

У зазначеному приміщенні працюють 4 людей. За даними, які наведено у табл. 8.1, та табл. 8.2, можна зробити висновок, що площа та об'єм приміщення у розрахунку на одно робоче місце програміста не відповідають нормативним вимогам ДСанПіН 3.3.2-007-98 «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» [2], але відповідають нормативним вимогам Наказу Міністерства соціальної політики України № 207, від 14.02.2018 «Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями» [5] та НПАОП 0.00-1.28-10 «Правила охорони праці під час експлуатації електронно-

обчислювальних машин»). Таким чином можна зробити висновок, що санітарно-гігієнічні умови праці на робочому місці програміста відповідають вимогам.

Температура повітря в приміщенні визначається впливом температури зовнішнього повітря і тепловою енергією, яка виділяється всередині приміщення. Джерелами виділення теплоти в даному приміщенні є електроустаткування, освітлювальні прилади, а також люди. У світлий час доби джерелом надлишкового тепла є сонячна радіація. Згідно Постанови № 42 від 01.12.1999 Головного державного санітарного лікаря України, робота, виконувана в даному приміщенні, відноситься до категорії Іа. В цьому випадку людина витрачає енергії до 120 ккал у годину. Вологість повітря в приміщенні визначається впливом багатьох факторів, серед яких: вологість атмосферного повітря, виділення вологи людьми (при диханні та випарами з поверхні шкіри).

Мікроклімат повітряного середовища в приміщенні характеризується запиленістю та загазованістю повітря. Мікроклімат приміщення визначається діючим на організм людини поєднанням, вологості, температури, швидкості руху повітря та інтенсивності теплового випромінювання. Аналіз мікроклімату складається з визначення зазначених вище факторів і порівняння результатів із встановленими нормами.

У таблиці 8.3 наведено оптимальні та фактичні значення параметрів мікроклімату як для категорії ваги робіт Іа, так і розглянутого приміщення. У приміщеннях, де встановлено ЕОМ, рекомендується застосування тільки оптимальних значень показників мікроклімату.

Таблиця 8.3 – Оптимальні і фактичні значення параметрів мікроклімату

Пора року	Оптимальні для Іа			Фактичні		
	Температура, °С	Воло-гість,%	Швидкість повітря, м/с	Температура,°С	Воло-гість%	Швидкість повітря, м/с
Холодна	22-24	40-60	0,1	22-23	40-55	0,1
Тепла	23-25	50-70	0,1	24-25	50-65	0,11

Проведений аналіз показує, що показники мікроклімату в приміщенні відповідають установленим нормам. Штучне опалення застосовується у холодний період року.

В літню пору застосовується кондиціонер.

Для боротьби з пилом робляться регулярні провітрювання та вологі прибирання приміщенні.

У приміщенні знаходяться наступні джерела шуму: принтер HP 1100, електродвигуни вентиляторів ЕОМ.

Одним з найважливіших факторів, які впливають на ефективність трудової діяльності людини, та попереджають травматизм і професійні захворювання програмістів є освітлення на робочому місці.

З 2019 року діють Державні будівельні норми України "Природне і штучне освітлення" – ДБН В.2.5-28:2018 [1], у яких прописані вимоги до використання всіх освітлювальних приладів, у т.ч. світлодіодних.

Працю працівника, який постійно працює за комп'ютером, згідно ДБН В.2.5-28:2018 [1], можна віднести до роботи з малою точністю (найменший розмір об'єкта розрізнення від 1 до 5 мм) V-го розряду зорової роботи, з великою контрастністю об'єкта розрізнення (символів на екрані дисплея), з темним тлом (під розряд зорової роботи В). Приміщення можна віднести до 1-ої групи приміщень, у яких проводиться розрізнення об'єктів зорової роботи при фіксованому напрямку лінії зору того, що працює на робочу поверхню. Для такого типу приміщень і розряду зорової роботи нормоване значення коефіцієнта природної освітленості (КПО) робочої поверхні (при поєднаному, спільному освітленні), повинен становити не більше 1,5%, освітленість при штучному висвітленні повинна становити 300 Лк. [1], Крім того все поле зору повинне бути освітлено достатньо рівномірно – ця основна гігієнічна вимога. Так як яскраве світло на ділянці периферійного зору значно збільшує напруженість очей і, як

наслідок, призводить до їх швидкої стомлюваності, ступінь освітлення приміщення і яскравість екрану комп'ютера повинні бути приблизно однаковими.

8.3. Розробка заходів з умов поліпшення охорони праці

Згідно аналізу умов праці в розглянутому приміщенні, ми одержали наступні результати:

– розмірі приміщення, у розрахунку на одному працюючого, відповідають нормативам;

– мікроклімат відповідає нормативному значенню;

– акустичні умови роботи не перевищують нормативних значень.

Таким чином можна припустити, що основною причиною можливого зниження працездатності програміста є психофізіологічний фактор, тому основна пропозиція буде така: дотримання позитивної психологічної атмосфери в колективі та регламентованого режиму праці та відпочинку, організація робочого місця з урахуванням ергономічних вимог.

Рекомендовані заходи: регулярні періодичні наочні огляди персоналом шляхів для евакуації людей із приміщення, відповідно до плану евакуації (який повинен розташовуватись на видному місці у приміщенні), включення до колективного договору мінімально можливого вмісту аптечок з обов’язковою наявністю масок-клапанів, або іншого спорядження для штучного дихання. Регулярна періодична перевірка параметрів заземлення та занулення (вимірювання опору ланцюга). Регулярна наочне знайомство персоналу із шляхами для евакуації людей із приміщення відповідно до плану евакуації, забезпечення розподільних щитів спеціальними розетками з заземлюючими контактами; організація заземлення всіх приладів і пристроїв, які працюють при напрузі вище 36 В. Так як при ураженні електричним струмом у людини може статися фібриляція шлуночків серця, в організації бажано мати дефібрилятор і підготовлений персонал для роботи з ним.

8.4 Розрахункова частина

Для захисного штучного заземлення застосовуються вертикальні електроди: металевий куток 45·45·5 мм., довжиною *L=2,5 м.,* та горизонтальний електрод – металева полоса з перетином 40·4 мм. Напруга – 220/380 В. Розрахункова схема розташування заземлюючих електродів – у ряд.

Розрахунок проводиться за допустимим опором розтіканню струму заземлювача.

Початкові дані для розрахунку захисного заземлення: тип верхнього шару грунта – чорнозем, нижнього шару грунта – глина (питомий опір $\rho_2 = 40$ *Ом·м)*. Умовна товщина верхнього шару грунта: *H=0,5 м*. Відстань між вертикальними заземлювачами (електродами) *A=3 м*. Глибина закладення горизонтального контура заземлення *t=0,8 м*. Опір заземлювача, який нормується: $R_{ЗН} = 4$ *Ом*. Необхідно визначити необхідну кількість вертикальних заземлювачів та длину полоси (горизонтального заземлювача).

Розрахунок захисного заземлення можна автоматизувати за допомогою програми, сирцевий код якої опублікован на стр. 13-16 [6], або аналогічної.

Розрахунок.

Відстань від центра вертикального заземлювача до поверхні землі:

$$T=t+L/2=0,8+2,5/2=2,05 \text{ м.}$$

Розрахунковий питомий опір грунта (з врахуванням того, що фактично вся конструкція заземлювача розташовується у нижньому шарі грунта):

$$\rho = \psi \rho_2 = 1,36 \cdot 40 = 54,4 \text{ Ом·м.}$$

де $\psi = 1,36$ – табличне значення коефіцієнта сезонності для відповідної кліматичної зони у багатошаровому грунті [6]; $\rho = 40$ *Ом·м.* – табличне значення питомого опору нижнього шару грунта (глина) [11].

Еквівалентний діаметр ветикальньного електрода (кутка) [11]:

$$Dв=0,95 \cdot K= 0,95 \cdot 45 = 47,5 \text{ мм.} = 0,04275 \approx 4,3 \text{ м.}$$

де $K = 45$ *мм.* – розмір металевого кутка (задан).

Відношення $A/L=3/2,5=1,2$.

Опір розтіканню електричного струму одного електрода вертикального заземлювача з урахуванням заглиблення заземлювача [11]:

$$R_O=0,366(\rho/L)[lg(2L/Dв)+(1/2)lg((4T+L)/(4T-L))]=$$

$$=0,366(54,5/2,5)[(lg(2\cdot2,5/0,0475)+(1/2\ )lg((4\cdot2,3+2,5)/(4\cdot2,3-2,5))]\ =17,55\ Ом.$$

Визначаємо коефіцієнт екранування вертикальних електродів $Кев=0,8$ при орієнтовній кількості вертикальних електродів, яке дорівнює 4 [11].

Визначаємо необхідну кількість вертикальних заземлювачів (без врахування горизонтального заземлювача), при $R_{ЗН} = 4\ Ом$:

$$N=R_O\ /(\ Кев\ R_{ЗН})=\ 17,55\ /\ (0,8\cdot4)=5,48 \approx 6\ шт.$$

Визначаємо довжину з'єднуючої полоси:

$$L_П\ =1,05\cdot A\cdot N=\ 1,05\cdot2,5\cdot5,48=17,28 \approx 18\ м.$$

Опір розтіканню електричного струму з'єднуючої полоси з урахуванням кліматичного коефіцієнта питомого опору грунта $K_П$ [11]:

$$R_П\ =0,366(\rho_2\cdot K_П/L_П\ )lg(2(L_П\ \cdot L_П\ )/(B\cdot t))=$$

$$=0,366(40\cdot5/17,28)\cdot lg((2\cdot17,28^2)/(0,04\cdot0,8))=18\ Ом.$$

де $K_П=5$ – табличне значення кліматичного коефіцієнта питомого опору грунта для відповідної клвматичної зони для з'єднуючої полоси [11]:

$B = 40\ мм. = 0,04\ м.$- ширина з'єднуючої полоси (задана).

Загальний опір розтіканню електричного струму заземлювача [11]:

$$R\ =(R_O\cdot R_П)/(\ R_O\cdot\eta_П+\ N\cdot R_П\cdot Кев)=$$

$$=(17,55\cdot\ 18)/(17,55\cdot\ 0,75+\ 5,48\cdot\ 18\cdot\ 0,8)=3,43\ Ом.$$

де $\eta_П = 0,75$ – табличне значення коефіцієнта екранування з'єднуючої полоси [11].

Умова $R \le R_{ЗН}$ виконується ($3,43 \le 4$).

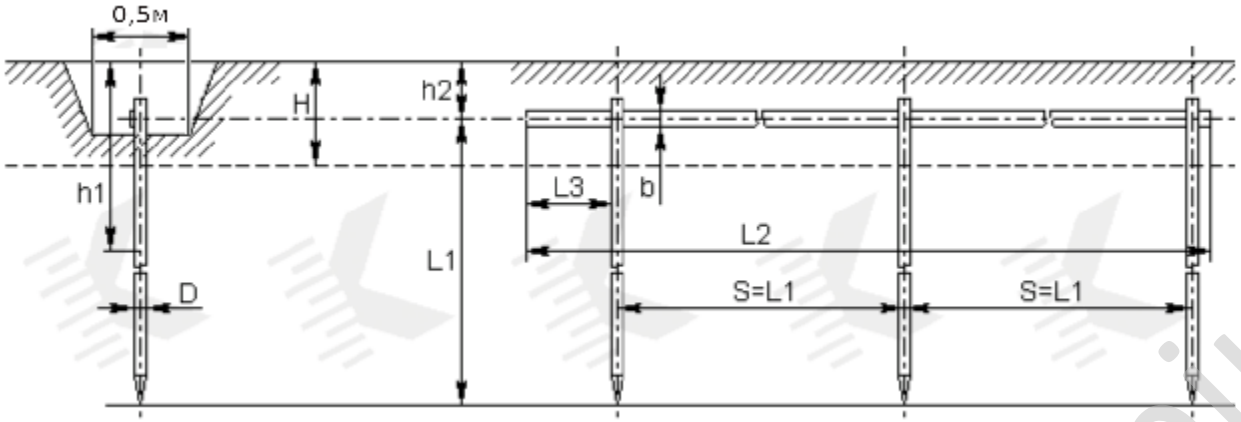

Рисунок 8.1 – Схема штучного заземлення.

При необхідності можна зменшити кількість електродів заземлювача зменшивши загальний опір розтіканню електричного струму заземлювача методом зменшення питомого опору ґрунта, домішуючи у грунт безпосередньо навколи електродів заземлювача розчини солей NaCl, CaCl, сажу, соду, шлак, коксову дрібницю, або спеціальні суміші.

8.5 Висновки до розділу

Дотримання всіх необхідних умов праці не лише сприяє збереженню здоров'я працівників, а також підвищує ефективність виробництва в цілому.

З цих міркувань було здійснено аналіз приміщення, призначеного для праці програмістів, проведено розгляд небезпечних та шкідливих факторів, що негативно впливають на програмістів під час роботи. Виконано розрахунок захисного штучного заземлення, як одного з ключових факторів безпеки програміста. Розроблено заходи з охорони праці.

9 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти, призначено для системи для протидії декомпіляції коду.

В межах України в недостатній мірі представлені вітчизняні розробки в цій області.

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти наведені теоретичне узагальнення й рішення наукового завдання дослідження методів для протидії декомпіляції коду.

Рішення даного завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем для протидії декомпіляції коду.

– Досліджена система для протидії декомпіляції коду.

– На основі отриманих результатів досліджень створена програмна реалізація системи для протидії декомпіляції коду.

Розроблені під час виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання для протидії декомпіляції коду.

Проведено аналіз предметної галузі в ході якого були виявлені об'єкти, взаємодія яких носить істотний характер для функціональної діяльності предметної галузі, і їхні основні характеристики; побудована алгоритм і вибраний середовище розробки.

Розроблене програмне забезпечення має простий, дружній та зручний інтерфейс користувача, що забезпечує легкість у освоєнні роботи програмного продукту, зручність у використанні, і не потребує особливих спеціальних знань.

При створені програмного забезпечення було використано об’єктно-орієнтований підхід, що відповідає сучасним тенденціям у галузі розробки комерційних програмних систем.

Програма реалізована на мові високого рівня RAD Studio Delphi 10.4. Дана мова програмування дозволяє найбільш ефективно обробляти дані. Це дозволило мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне програмне забезпечення, що спеціально розроблене для даної конкретної системи й включає програми, що реалізують її функції.

Програма призначена для виконання під управлінням багатозадачної операційної системи Windows 10/11.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм NTRU.

В цілому створене програмне забезпечення підтверджує правильність використаних проектних рішень та повністю відповідає вимогам технічного завдання. Створене програмне забезпечення має потенційну можливість для подальшого вдосконалення і застосування у різних галузях.

Розроблена програма має реальний економічний ефект від її впровадження у виробництво у сумі 26256 грн. З урахуванням вартості розробки програми та обладнання, строк окупления становить 0,1 роки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Федоров Б.С. Дослідження та програмна реалізація системи для протидії декомпіляції коду // Збірник праць молодих науковців ЦНТУ. – Вип. 13. – Кропивницький: ЦНТУ, 2022.

2. ShokurovA.V. An approach to quantitative analysis of resistance of equivalent transformations of algebraic schemes // Труды Института системного программирования, 2004, т. 6.

3. B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, K. Yang. On the (Im)possibility of Obfuscating Programs. LNCS, 2001, 2139, pp. 1-18.

4. S. Chow, Y. Gu, H. Johnson, V. Zakharov. An approach to the obfuscation of control-flow of sequential computer programs. LNCS, 2001, 2200, pp. 144-155.

5. C. Cifuentes, K. J. Gough. Decompilation of Binary Programs. Technical report FIT-TR-1994-03. Queensland University of Technology, 1994.

6. C. Collberg, C. Thomborson, D. Low. A Taxonomy of Obfuscating Transformations. Department of Computer Science, The University of Auckland, 1997.

7. G. Hachez, C. Vasserot. State of the Art in Software Protection. Project FILIGRANE (Flexible IPR for Software Agent Reliance) deliverable/V2.

8. M. Hind, A. Pioli. Which pointer analysis should I use? In ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 113-123, August 2000.

9. Kovalenko O., Popereshnyak S., Grinenko S., Grinenko O., Radivilova T. «Methods for Assessing the Maturity Levels of Software Ecosystems». *CEUR Workshop Proceedings* Volume 2654, 2019, Pages 251-261. Режим доступу: https://www.scopus.com/record/display.uri?eid=2-s2.0-85083214331&origin=AuthorNamesList&txGid=5633fba897776a6e0f3d5633fbcd3fbe **(Scopus)**.

10. Kovalenko O., Drieieva H., Simakhin V., Bondar S., Drieiev O., Zhumadilova M. «Multifractal Properties of Traffic Generator Based on Markov Chains ». *CEUR Workshop Proceedings* Volume 2588, 2019, Pages 567-579. Режим

доступу: https://www.scopus.com/record/display.uri?eid=2-s2.0-85083214331&origin=AuthorNamesList&txGid=176e2cada8976a6e0f3d5633fbcd3fbe **(Scopus)**.

11. Kovalenko Oleksandr Qualitative risk analysis of software development / Oleksandr Kovalenko, Jamil Al-Azzeh, Oleksii Smirnov, Anna Kovalenko, Serhii Smirnov // Asian Journal of Information Technology. – Volume 17 Issue 3. – Medwell Journals. – 2018. – P. 218-230. ISSN: 1682-3915. URL: http://medwelljournals.com/abstract/?doi=ajit.2018.218.230 Doi: ajit.2018.218.230

12. Kovalenko Oleksandr, The mathematical model of the testing technology for DOM XSS vulnerabilities / O. Kovalenko, O. Smirnov, A.Kovalenko, S. Smirnov, V. Vialkova // Scientific & practical cyber security journal (SPCSJ) Volume 2 Issue 1, P. 22-28. Georgia. Tbilisi. Scientific Cyber Security Association (SCSA), 2018 ISSN: 2587-4667. URL: https://journal.scsa.ge/wp-content/uploads/2018/12/04-3-o.kovalenko-a.kovalenko-o.smirnov-s.smirnov-v.vialkova.pdf

13. Коваленко А.В. Технология тестирования DOM XSS уязвимости / А.В. Коваленко, А.С. Коваленко, А.А. Смирнов, С.А. Смирнов // Scientific & practical cyber security journal (SPCSJ) Volume 1. Issue 1. P. 38-45 Georgia. Tbilisi. Scientific Cyber Security Association (SCSA), 2017 ISSN: 2587-4667. URL: https://journal.scsa.ge/wp-content/uploads/2018/12/8-dom-xss-testing-technology-vulnerabilities.pdf

14. Коваленко О.В. Моделі та методи розроблення програмного забезпечення комп'ютерних систем для підвищення безпеки даних: монографія / О.В. Коваленко // К.: Вид. «КОД» – 2019. – 305 с.

15. Коваленко А.В. Методы качественного анализа и количественной оценки рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Информационные технологии в управлении, образовании, науке и промышленности: монография / Под редакцией профессора В.С. Пономаренко. – Х.: Видавець Рожко С.Г., 2016. – 566 с.

16. Коваленко А.В. Разработка метода управления рисками разработки

программного обеспечения / А.В. Коваленко, А.А. Смирнов // Інформаційні технології: проблеми та перспективи: монографія / За загальною редакцією В.С. Пономаренка. – Х.: Видавець Рожко С.Г., 2017. – 447 с.

17. Коваленко А.В. Комплекс математических моделей технологии тестирования web-приложений / А.В. Коваленко, А.А. Смирнов // Інформаційні технології: сучасний стан та перспективи: монографія / За загальною редакцією В.С. Пономаренка. – Х.: ТОВ «ДІСА ПЛЮС», 2018. – 461 с.

18. Коваленко А.В. Задачи распознавания ситуаций в ERP системах / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Збірник наукових праць "Системи обробки інформації". – Випуск 4(120). – Х.: ХУПС – 2014. – С. 161-164.

19. Коваленко А.В. Методы качественного анализа и количественной оценки рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник наукових праць "Системи обробки інформації". – Випуск 5(142). – Х.: ХУПС – 2016. – С. 153-157.

20. Коваленко А.В. Проблемы анализа и оценки рисков информационной деятельности / А.В. Коваленко, А.А. Смирнов, Н.Н. Якименко, А.П. Доренский // Збірник наукових праць "Системи обробки інформації". – Випуск 3(140). – Х.: ХУПС – 2016. – С. 40-42.

21. Коваленко А.В. Метод качественного анализа рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, Н.Н. Якименко, А.П. Доренский // Наука і техніка Повітряних Сил Збройних Сил України. – Випуск 2(23). – Харків: ХУПС. – 2016. – С. 150-158.

22. Коваленко А.В. Метод количественной оценки рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, Н.Н. Якименко, А.П. Доренский // Збірник наукових праць Харківського університету Повітряних Сил. Випуск 2 (47). – Харків: ХУПС. – 2016. – С. 128-133.

23. Коваленко А.В. Использование псевдобулевых методов бивалентного программирования для управления рисками разработки программного

обеспечения / А.В. Коваленко, А.А. Смирнов // Системи управління, навігації та зв'язку. – Випуск 1 (37). – Полтава: ПолтНТУ. – 2016. – С. 98-103.

24. Коваленко А.В. Метод управления рисками разработки программного обеспечения / А.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 2 (38). – Полтава: ПолтНТУ. – 2016. – С. 93-100.

25. Коваленко А.В. Технология тестирования уязвимости к SQL инъекциям / А.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 5 (45). – Полтава: ПолтНТУ. – 2017. – С. 66-71.

26. Коваленко А.В. Масштабирование имитационной модели технологии тестирования безопасности / А.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 6 (46). – Полтава: ПолтНТУ. – 2017. – С. 181-184.

27. Коваленко А.В. Имитационная модель технологии тестирования безопасности Web-приложений / А.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 1 (47). – Полтава: ПолтНТУ. – 2018. – С. 114-123.

28. Коваленко О.В. Методи якісного аналізу та кількісної оцінки ризиків розроблення програмного забезпечення/ О.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 3 (49). – Полтава: ПолтНТУ. – 2018. – С. 116-125.

29. Коваленко О.В. Управління ризиками розроблення програмного забезпечення за умови обмеженості коштів виділених на усунення помилок безпеки/ О.В. Коваленко // Техніка в сільськогосподарському виробництві, галузеве машинобудування, автоматизація. – Випуск 31. – Кропивницький: ЦНТУ. – 2018. – С. 128-140.

30. Коваленко О.В. GERT-мережевий синтез технології тестування на вразливість WEB-додатків/ О.В. Коваленко // Захист інформації. – Випуск 20(2) – К.: НАУ. – 2018. – С. 89-94.

31. Коваленко О.В. Імітаційна модель технології тестування безпеки на основі положень теорії масштабування / О.В. Коваленко // Безпека інформації. – Випуск 24 (2). – К.: НАУ. – 2018. – С. 110-117.

32. Коваленко О.В. Оцінка ефективності технології тестування

безпеки / О.В. Коваленко // Вчені записки Таврійського національного університету імені В.І. Вернадського. Серія: Технічні науки. Том 29 (68) № 2, 2018. – С. 137-141

33. Коваленко О.В. Методи та засоби управління безпекою додатків / О.В. Коваленко // Інформаційно-керуючі системи на залізничному транспорті. №4, 2018. – С. 41-44.

34. Коваленко О.В. Розробка інформаційної технології передтестової компіляції та розподілу доступу / О.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 4 (50). – Полтава: ПолтНТУ. – 2018. – С. 115-119.

35. Коваленко О.В. Аналіз та дослідження інформаційних технологій розробки програмного забезпечення/ О.В. Коваленко // Вчені записки Таврійського національного університету імені В.І. Вернадського. Серія: Технічні науки. Том 29 (68) № 5, 2018. – С. 131-137.

36. Коваленко О.В. Удосконалений метод управління ризиками розроблення програмного забезпечення на основі напівмарковської моделі прийняття рішень/ О.В. Коваленко // Сучасні інформаційні системи. – Випуск 2(3). – Харків. – 2018. – С. 41-48.

37. Коваленко О.В. Математичні моделі технології тестування DOM XSS вразливості та вразливості до SQL ін'єкцій / О.В. Коваленко // Вісник Черкаського державного технологічного університету. Серія : Технічні науки №4, 2018. – С. 29-36.

38. Коваленко О.В. Математична модель технології тестування вразливості до SQL ін'єкцій / О.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 6 (58). – Полтава: ПолтНТУ. – 2019. – С. 43-47.

39. Коваленко О.В. Математична модель технології тестування комплексу DOM XSS вразливостей для аналітичної оцінки часових витрат / О.В. Коваленко // Центральноукраїнський науковий вісник. Технічні науки. № 2(33). с. 173-180, 2019.

40. Коваленко А.В. Проблемы анализа и оценки рисков

информационной деятельности / А.В. Коваленко, А.А. Смирнов // Збірник наукових праць II міжнародної науково-практичної конференції «Актуальні питання забезпечення кібернетичної безпеки та захисту інформації». м. Київ. 24-27 лютого 2016 р. – Київ: Європейський університет. – 2016. – С. 138-139.

41. Коваленко А.В. Анализ и оценка рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез «Securitea informationala 2015-2016». Conferenta internationala (editia a XII-a). Chisinau. Moldova. 3 martie 2016. – Chisinau: ADSEM. – 2016. – P. 96-102.

42. Коваленко А.В. Исследование источников и причин риска разработки программного обеспечения, этапов и работ, при выполнении которых возникает риск / А.В. Коваленко, А.А. Смирнов // Збірник тез VII всеукраїнської науково-практичної конференції "Інформатика та системні науки (ІСН-2016)". м. Полтава. 10-12 березня 2016 р. – Полтава.: ПУЕТ – 2016. – С. 264-266.

43. Коваленко А.В. Оценка показателя чистой приведенной стоимости для количественной оценки рисков проекта разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез науково-практичної конференції “Проблеми кібербезпеки інформаційно-телекомунікаційних систем”. м. Київ. 10-11 березня 2016 р. – Київ: КНУ ім. Тараса Шевченко – 2016. – С. 81-82.

44. Коваленко А.В. Методика структурной идентификации рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез Міжнародної науково-практичної конференції «Інформаційна безпека та комп'ютерні технології» (IS&CT). м. Кіровоград. 24-25 березня 2016 р. – Кіровоград: КНТУ. – 2016. – С. 71-72.

45. Коваленко А.В. Методы качественного анализа рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез першої міжнародної науково-практичної конференції «Проблеми науково-технічного та правового забезпечення кібербезпеки у сучасному світі» (ПНПЗК-

2016). м. Харків. 30 березня – 1 квітня 2016 р. – Харків: НТУ «ХПІ». – 2016. – С. 6-7.

46. Коваленко А.В. Структурная идентификация рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез XVIII міжнародного науково-практичного семінару «Комбінаторні конфігурації та їх застосування». м. Кіровоград. 15-16 квітня 2016 р. – Кіровоград: КНТУ. – 2016. – С. 175-182.

47. Коваленко А.В. Исследование разработанной методики структурной идентификации рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез VIII міжнародної науково-практичної конференції "Проблеми і перспективи розвитку IT-індустрії". м. Харків. 28-29 квітня 2016 р. – Харків: ХНЕУ. – 2016. – С. 49.

48. Коваленко А.В. Исследование дерева рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез III міжнародної науково-практичної конференції «Інформаційна та економічна безпека» (INFECO-2016)». м. Харків. 28-30 квітня 2016 р. – Харків: ХНМІ ДВНЗ «УБС». – 2016. – С. 174-178.

49. Коваленко А.В. Методы качественного анализа и количественной оценки рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Сборник тезисов XII международной конференции "Стратегия качества в промышленности и образовании". г. Варна. Болгария. 30 мая – 02 июня 2016 г – Варна. ТУВ. – 2016. – С. 585-589.

50. Коваленко А.В. Разработка метода управления рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Матеріали Всеукраїнської науково-практичної конференції «Кібербезпека в Україні: правові та організаційні питання». м. Одеса, 21 жовтня 2016 р. – Одеса : ОДУВС, 2016. – С.146-148.

51. Коваленко А.В. Метод управления рисками разработки программного обеспечения с использованием псевдобулевых методов

бивалентного программирования / А.В. Коваленко, А.А. Смирнов // Матеріали Всеукраїнської науково-практичної конференції «Актуальні задачі та досягнення у галузі кібербезпеки». м. Кропивницький, 23-25 листопада 2016 року – Кропивницький: ЦНТУ, 2016. – С. 162.

52. Коваленко А.В. Псевдобулевые методы бивалентного программирования для управления рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко, С.А. Смирнов // Збірник наукових праць III міжнародної науково-практичної конференції «Актуальні питання забезпечення кібернетичної безпеки та захисту інформації». м. Київ. 22-25 лютого 2017 р. – Київ: Європейський університет. – 2017. – С. 158-162.

53. Коваленко А.В. Метод управления рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез II науково-практичної конференції "Проблеми кібербезпеки інформаційно-телекомунікаційних систем". м. Київ. 23-24 березня 2017 р. – Київ: КНУ ім. Тараса Шевченко – 2017. – С. 203-205.

54. Коваленко А.В. Алгоритмы анализа уязвимостей при управлении рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Conferenta internationala (editia a XIII-a). «Securitea informationala 2017». Chisinau. Republic of Moldova. 4-5 aprilie 2017. – Chisinau: ADSEM. – 2017. – P. 19-22.

55. Коваленко А.В. Алгоритм анализа DOM XSS уязвимости при управлении рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Збірник тез дев'ятнадцятого міжнародного науково-практичного семінару «Комбінаторні конфігурації та їх застосування». м. Кропивницький 7-8 квітня 2017 р. – Кропивницький: ГЛА НАУ. – 2017. – С. 125-127.

56. Коваленко А.В. Алгоритм анализа уязвимости SQL Injection для управления рисками разработки программного обеспечения / А.В. Коваленко,

А.А. Смирнов, А.С. Коваленко // Збірник тез другої міжнародної науково-технічної конференції «Проблеми науково-технічного та правового забезпечення кібербезпеки у сучасному світі» (ПНПЗК-2017). м. Харків.10-12 квітня 2017 р. – Харків: НТУ «ХПІ». – 2017. – С. 27.

57. Коваленко А.В. Метод управления рисками разработки программного обеспечения на основе алгоритмов анализа уязвимостей / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Збірник тез Міжнародної науково-практичної конференції «Інформаційна безпека та комп'ютерні технології» (IS&CT). м. Кіровоград. 20-22 квітня 2017 р. Кіровоград: КНТУ. – 2017. – С. 92.

58. Коваленко А.В. Алгоритмы анализа DOM XSS уязвимости и уязвимости SQL Injection при управлении рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Збірник тез IX міжнародної науково-практичної конференції "Проблеми і перспективи розвитку IT-індустрії". м. Харків. 20-21 квітня 2017 р. – Харків: ХНЕУ. – 2017. – С. 61.

59. Державні будівельні норми України: ДБН В.2.5-28:2018. – Режим доступу до ресурсу: https://goo.su/9AkQ

60. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин: ДСанПІН 3.3.2-007-98. – Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/v0007282-98

61. Закон України «Про охорону праці» від 14.10.1992 р. № 2694-XII. – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/2694-12

62. Зеркалов Д. В. Охорона праці в Галузі: Загальні вимоги: навч. посіб. Київ: Основа. 2011. 551 с.

63. Наказ Міністерства соціальної політики України 14.02.2018 № 207 «Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями». – Режим доступу до ресурсу:

https://zakon.rada.gov.ua/laws/show/z0508

64. Охорона праці. Ч. 1. Захисне заземлення: метод. вказ. до викон. розрахунків з викор. персон. ЕОМ ІВМ сумісного типу / Кіровоград. ін-т с.-г. машинобуд.; [укл. О. В. Оришака, Є. К. Солових, В. О. Оришака]. – Кіровоград: КІСМ, 1997. – 20 с. Режим доступу до ресурсу: http://dspace.kntu.kr.ua/jspui/handle/123456789/4358

65. Постанова № 42 від 01.12.1999 Головного державного санітарного лікаря України «Санітарні норми мікроклімату виробничих приміщень ДСН 3.3.6.042-99. – Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/va042282-99

66. Сакулин В.П., Шептовицкий В.М. Безопасность труда при монтаже и эксплуатации электроустановок / В.П.Сакулин, В.М.Шептовицкий. – Л. : “Колос”, 1973. – 238 с.

67. Центр післядипломної освіти та підвищення кваліфікації. – Режим доступу до ресурсу: https://cpo.stu.cn.ua

68. Оришака, О. В. Основи охорони праці: навч. посіб. / О. В. Оришака, Г. П. Горбачова, К. М. Марченко; М-во освіти і науки України, Центральноукраїн. нац. техн. ун-т. – Кропивницький : ЦНТУ, 2022. – 175 с. – Режим доступу до ресурсу: http://dspace.kntu.kr.ua/jspui/handle/123456789/12161 (дата звернення 19.09.22).

Додаток А
(обов'язковий)

Технічне завдання

Зміст

					ВКРМ-123.22.0025.00.00.ТЗ		
Вим.	Арк.	№ документа	Підпис	Дата			
Розробив		Федоров Б.С.			Дослідження та програмна реалізація системи для протидії декомпіляції коду	Літ.	Аркуш 1 / Аркушів 6
Перевірив		Смірнов С.А.				М	
Н. Контр.		Гермак В.С.				ЦНТУ КІ-21М-1,4	
Затв.		Смірнов О.А.					

1 Найменування та область застосування

Це технічне завдання розповсюджується на дослідження та програмну реалізацію системи для протидії декомпіляції коду.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 19-13 від 17.08.2022 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти є дослідження та програмна реалізація системи для протидії декомпіляції коду.

4 Джерела розробки

Джерелом цієї випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи з ОС та з користувачем;

– техніко-економічне обґрунтування доцільності прийнятого до розробки програмного забезпечення;

– аналіз умов праці;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– програмну реалізацію системи для протидії декомпіляції коду;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ архітектури IBM PC, працювати в ОС Windows 10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпечення.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows 10/11.

5.8.1 Обладнання

Комп’ютер Intel® Celeron/8 Mb/1.2 Gb/SVGA 14” 1Mb або сумісні з ним.

5.8.2 Мова програмування

Середовище RAD Studio Delphi 10.4.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД .

7 Економічні вимоги

7.1 Для ПЗ необхідно виробити функціонально-вартісний аналіз варіантів розробки.

7.2 Виконати розрахунок витрат показників економічного ефекту з урахуванням цін на 3 вересня 2022 року.

8 Вимоги щодо охорони праці

В частині охорони праці випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти повинен бути розглянутий аналіз санітарно-гігієнічних умов праці на робочому місці програміста.

9 Перелік документів, що розробляються

– Наукова новизна – 1 аркуш.
– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуша.
– Показники економічної ефективності – 1 аркуш.
– Пояснювальна записка – 111 аркушів.

10 Етапи розробки

10.1 Збір і обробка інформації по темі випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти. Постановка задачі на виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти (складання ТЗ).

10.2 Проведення досліджень або експериментальних робіт для уточнення основних положень випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти.

10.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

10.4 Побудова схем взаємодії даних.

10.5 Створення прототипу ПЗ.

10.6 Віднаходження ПЗ, аналіз отриманих результатів.

10.7 Робота над питанням охорони праці і техніки безпеки.

10.8 Розрахунок з техніко-економічного обґрунтування.

10.9 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю та приймання

11.1 Подання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти на попередній захист 10.12.2022 р.

11.2 Подання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти на захист 21.12.2022 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за

другим (магістерським) рівнем вищої освіти

_____ Смірнов С.А.

Дослідження та програмна реалізація системи для протидії декомпіляції коду

Лістинг програми

Код документу 12

Носій: CD/DVD-диск / USB-флеш-накопичувач

Загальна кількість аркушів: 101

Літера: РП

Кропивницький – 2022 року

Файл maincode.pas – основна програма

```
unit maincode;

interface
//Підключення бібліотек
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, aPLib, StdCtrls, ComCtrls, PE_Files, Menus,ExtCtrls, shellapi,
morphine,
  Buttons, ImgList, ToolWin, About;
//Опис головного класу
type
  TfrmMain = class(TForm)
    aPLib: TaPLib;
    dlgOpen: TOpenDialog;
    StatusBar1: TStatusBar;
    tabOpen: TTabSheet;
    tabOptions: TTabSheet;
    tabProtect: TTabSheet;
    tabSheet: TPageControl;
    Label1: TLabel;
    txtFileName: TEdit;
    cmdOpen: TBitBtn;
    gpMode: TGroupBox;
    rbPacking: TRadioButton;
    rbProtect: TRadioButton;
    rbFullProtect: TRadioButton;
    gpSettings: TGroupBox;
    cbIcon: TCheckBox;
    cbOverlay: TCheckBox;
    Label2: TLabel;
    txtImageBase: TEdit;
    lstStatus: TListView;
    pb: TProgressBar;
    cmdTest: TBitBtn;
    cmdProtect: TBitBtn;
    ilStatus: TImageList;
    MainMenu1: TMainMenu;
    mnuProject: TMenuItem;
    mnuNew: TMenuItem;
    N1: TMenuItem;
    mnuOpen: TMenuItem;
    mnuSave: TMenuItem;
    N2: TMenuItem;
    mnuExit: TMenuItem;
    mnuHelp: TMenuItem;
    mnuHelpOpen: TMenuItem;
    N3: TMenuItem;
    mnuAbout: TMenuItem;
    ToolBar1: TToolBar;
    ToolButton1: TToolButton;
    ToolButton2: TToolButton;
    ToolButton3: TToolButton;
    ToolButton4: TToolButton;
    ToolButton5: TToolButton;
    ToolButton6: TToolButton;
    ToolButton7: TToolButton;
    ToolButton9: TToolButton;
    dlgSave: TSaveDialog;
    procedure AddLog(sText: string; sImage: integer);
    procedure ClearLog;
    procedure Pack(InputFileName: string);
    procedure cmdOpenClick(Sender: TObject);
    procedure cmdProtectClick(Sender: TObject);
```

```
    procedure FormCreate(Sender: TObject);
    procedure mnuNewClick(Sender: TObject);
    procedure mnuOpenClick(Sender: TObject);
    procedure mnuSaveClick(Sender: TObject);
    procedure mnuExitClick(Sender: TObject);
    procedure cmdTestClick(Sender: TObject);
    procedure mnuAboutClick(Sender: TObject);
    procedure mnuHelpOpenClick(Sender: TObject);
  private
  public
    CurFileSz : DWORD;
  end;

  (*$IFDEF DYNAMIC_VERSION*)
  function CallBack(w0, w1, w2 : DWORD; cbparam : Pointer) : DWORD;stdcall;
(*$ELSE*)
  function CallBack(w0, w1, w2 : DWORD; cbparam : Pointer) : DWORD;cdecl;
(*$ENDIF*)

var
  frmMain: TfrmMain;

const
  packer_ver:string='';

implementation

{$R *.dfm}
{$R windowsxp.res}
//Опис записів заплутування
Type

ProtectorProject = record
  sFileName: string[255];
  sSaveIcon: boolean;
  sSaveOverlay: boolean;
  sImageBase: string[8];
  sPacking: boolean;
  sProtection: boolean;
  sActivePageIndex: integer;
end;


IMAGE_DIR_ITEM=record
                VirtualAddress:DWORD;
                Size:DWORD;
              end;

IMAGE_FILE_HEADER=record
                        Machine:WORD;
                        NumberOfSections:WORD;
                        TimeDateStamp:DWORD;
                        PointerToSymbolTable:DWORD;
                        NumberOfSymbols:DWORD;
                        SizeOfOptionalHeader:WORD;
                        Characteristics:WORD;
                  end;

IMAGE_OPTIONAL_HEADER=record
                            Magic:WORD;
                            MajorLinkerVersion:BYTE;
                            MinorLinkerVersion:BYTE;
                            SizeOfCode:DWORD;
                            SizeOfInitializedData:DWORD;
                            SizeOfUninitializedData:DWORD;
                            AddressOfEntryPoint:DWORD;
                            BaseOfCode:DWORD;
                            BaseOfData:DWORD;
                            ImageBase:DWORD;
```

```
                                блокAlignment:DWORD;
                                FileAlignment:DWORD;
                                MajorOperatingSystemVersion:WORD;
                                MinorOperatingSystemVersion:WORD;
                                MajorImageVersion:WORD;
                                MinorImageVersion:WORD;
                                MajorSubsystemVersion:WORD;
                                MinorSubsystemVersion:WORD;
                                Win32VersionValue:DWORD;
                                SizeOfImage:DWORD;
                                SizeOfHeaders:DWORD;
                                CheckSum:DWORD;
                                Subsystem:WORD;
                                DllCharacteristics:WORD;
                                SizeOfStackReserve:DWORD;
                                SizeOfStackCommit:DWORD;
                                SizeOfHeapReserve:DWORD;
                                SizeOfHeapCommit:DWORD;
                                LoaderFlags:DWORD;
                                NumberOfRvaAndSizes:DWORD;
                                IMAGE_DIRECTORY_ENTRIES:record
                                                        _EXPORT:IMAGE_DIR_ITEM;
                                                        IMPORT:IMAGE_DIR_ITEM;
                                                        RESOURCE:IMAGE_DIR_ITEM;
                                                        EXCEPTION:IMAGE_DIR_ITEM;
                                                        SECURITY:IMAGE_DIR_ITEM;
                                                        BASERELOC:IMAGE_DIR_ITEM;
                                                        DEBUG:IMAGE_DIR_ITEM;
                                                        COPYRIGHT:IMAGE_DIR_ITEM;
                                                        GLOBALPTR:IMAGE_DIR_ITEM;
                                                        TLS:IMAGE_DIR_ITEM;
                                                        CONFIG:IMAGE_DIR_ITEM;
                                                        BOUND_IMPORT:IMAGE_DIR_ITEM;
                                                        IAT:IMAGE_DIR_ITEM;
                                                         end;
                                DUMB:ARRAY [1..24] OF BYTE;
                        end;

SECTION=record
                Name:packed array [0..IMAGE_SIZEOF_SHORT_NAME-1] of Char;
                VirtualSize:DWORD;
                VirtualAddress:DWORD;
                SizeOfRawData:DWORD;
                PointerToRawData:DWORD;
                PointerToRelocations:DWORD;
                PointerToLinenumbers:DWORD;
                NumberOfRelocations:WORD;
                NumberOfLinenumbers:WORD;
                Characteristics:DWORD;
        end;

CONST
MAX_SECTION_NUMBER= $10;

VAR
    PE_HEADER:record
                IMAGE_NT_SIGNATURE:DWORD;
                FILE_HEADER:IMAGE_FILE_HEADER;
                OPTIONAL_HEADER:IMAGE_OPTIONAL_HEADER;
          end;
                блок_HEADER:ARRAY [1..MAX_SECTION_NUMBER] of блок;


    var
    hFile:DWORD;
    e_lfanew:DWORD;
    EXE:WORD;
    i:integer;
    bread:dword;
```

```
    EPreal, EP, imagebase,nv,ns,fa,sa:cardinal;
    num,epsec:integer;
    pe:pe_file;
    PACKEDSECTION:dword;
    PACKEDPOS:pointer;
    temp1:pointer;
    temp2:pointer;

// змінні депакування
    depbegin:dword;
    stra:string;
    iat:array[1..$b1] of byte;
    sizeofsec:dword;
    addrsec:dword;
    iatrva:dword;

Function RVA2Offset(RVA:DWORD):DWORD;
 var i:integer;
     VirtAddr,VA2,szRawData,ptrRawData:DWORD;
 begin
  for i:=1 to PE_HEADER.FILE_HEADER.NumberOfSections do
   begin
    VirtAddr:=SECTION_HEADER[i].VirtualAddress;
    szRawData:=SECTION_HEADER[i].SizeOfRawData;
    ptrRawData:=SECTION_HEADER[i].PointerToRawData;
    if RVA>=VirtAddr then
     begin
      VA2:=VirtAddr+szRawData;
      if RVA<VA2 then
       begin
        RVA:=RVA-VirtAddr;
        RVA:=RVA+ptrRawData;
       end;
     end;
   end;
  RVA2Offset:=RVA;
 end;
//Функція визначення розміру блоку заплутування
function GetLoaderSize(Func:dword):dword;
begin
asm
pushad
mov eax, func
mov esi, 1
@find:
mov dword ptr ebx, [eax]
mov dword ptr ecx, [eax+4]
cmp ebx, $41504544
jnz @notf
cmp ecx, $4E454B43
jnz @notf
mov result, esi
jmp @exit
@notf:
inc esi
inc eax
jmp @find
@exit:
popad
end;
end;
// процедура таблиць підстановки при заплутуванні коду
procedure ImportTable;
begin
{
0045F6A4  E8 F6 05 00 00 00 00 00 00 00 00 00 F8 F6 05 00  иц▯ .........шц▯ .
0045F6B4  E8 F6 05 00 E0 F6 05 00 00 00 00 00 00 00 00 00  иц▯ .ац▯ .........
0045F6C4  05 F7 05 00 E0 F6 05 00 00 00 00 00 00 00 00 00  ▯ч▯.ац▯ .........
0045F6D4  00 00 00 00 00 00 00 00 00 00 00 00 58 12 DA 77  ............X▯Ъw
```

```
0045F6E4  00 00 00 00 79 BB E6 77 1F A0 E6 77 C4 EF ED 77  ....y»жw жwДпнw
0045F6F4  00 00 00 00 4B 45 52 4E 45 4C 33 32 2E 64 6C 6C  ....KERNEL32.dll
0045F704  00 55 53 45 52 33 32 2E 64 6C 6C 00 00 00 47 65  .USER32.dll...Ge
0045F714  74 50 72 6F 63 41 64 64 72 65 73 73 00 00 00 47  tProcAddress...G
0045F724  65 74 4D 6F 64 75 6C 65 48 61 6E 64 6C 65 41 00  etModuleHandleA.
0045F734  00 00 4C 6F 61 64 4C 69 62 72 61 72 79 41 00 00  ..LoadLibraryA..
0045F744  00 4D 65 73 73 61 67 65 42 6F 78 41 00 00 00 00  .MessageBoxA....
}
for i:=1 to $b1 do iat[i]:=0;

iat[1]:=Lo(DEPBEGIN-imagebase+$44);
iat[2]:=Hi(DEPBEGIN-imagebase+$44);
iat[3]:=Lo((DEPBEGIN-imagebase+$44) shr 16);
iat[4]:=Hi((DEPBEGIN-imagebase+$44) shr 16);

iat[13]:=Lo(DEPBEGIN-imagebase+$54);
iat[14]:=Hi(DEPBEGIN-imagebase+$54);
iat[15]:=Lo((DEPBEGIN-imagebase+$54) shr 16);
iat[16]:=Hi((DEPBEGIN-imagebase+$54) shr 16);

iat[17]:=Lo(DEPBEGIN-imagebase+$44);
iat[18]:=Hi(DEPBEGIN-imagebase+$44);
iat[19]:=Lo((DEPBEGIN-imagebase+$44) shr 16);
iat[20]:=Hi((DEPBEGIN-imagebase+$44) shr 16);

iat[21]:=Lo(DEPBEGIN-imagebase+$3C);
iat[22]:=Hi(DEPBEGIN-imagebase+$3C);
iat[23]:=Lo((DEPBEGIN-imagebase+$3C) shr 16);
iat[24]:=Hi((DEPBEGIN-imagebase+$3C) shr 16);

iat[33]:=Lo(DEPBEGIN-imagebase+$61);
iat[34]:=Hi(DEPBEGIN-imagebase+$61);
iat[35]:=Lo((DEPBEGIN-imagebase+$61) shr 16);
iat[36]:=Hi((DEPBEGIN-imagebase+$61) shr 16);

iat[37]:=Lo(DEPBEGIN-imagebase+$3C);
iat[38]:=Hi(DEPBEGIN-imagebase+$3C);
iat[39]:=Lo((DEPBEGIN-imagebase+$3C) shr 16);
iat[40]:=Hi((DEPBEGIN-imagebase+$3C) shr 16);

iat[61]:=Lo(DEPBEGIN-imagebase+$9F);
iat[62]:=Hi(DEPBEGIN-imagebase+$9F);
iat[63]:=Lo((DEPBEGIN-imagebase+$9F) shr 16);
iat[64]:=Hi((DEPBEGIN-imagebase+$9F) shr 16);

iat[69]:=Lo(DEPBEGIN-imagebase+$6C);
iat[70]:=Hi(DEPBEGIN-imagebase+$6C);
iat[71]:=Lo((DEPBEGIN-imagebase+$6C) shr 16);
iat[72]:=Hi((DEPBEGIN-imagebase+$6C) shr 16);

iat[73]:=Lo(DEPBEGIN-imagebase+$7D);
iat[74]:=Hi(DEPBEGIN-imagebase+$7D);
iat[75]:=Lo((DEPBEGIN-imagebase+$7D) shr 16);
iat[76]:=Hi((DEPBEGIN-imagebase+$7D) shr 16);

iat[77]:=Lo(DEPBEGIN-imagebase+$90);
iat[78]:=Hi(DEPBEGIN-imagebase+$90);
iat[79]:=Lo((DEPBEGIN-imagebase+$90) shr 16);
iat[80]:=Hi((DEPBEGIN-imagebase+$90) shr 16);

iat[85]:=byte('K');
iat[86]:=byte('E');
iat[87]:=byte('R');
iat[88]:=byte('N');
iat[89]:=byte('E');
iat[90]:=byte('L');
iat[91]:=byte('3');
iat[92]:=byte('2');
iat[93]:=byte('.');
```

```
iat[94]:=byte('D');
iat[95]:=byte('L');
iat[96]:=byte('L');

iat[98]:= byte('U');
iat[99]:= byte('S');
iat[100]:= byte('E');
iat[101]:=byte('R');
iat[102]:=byte('3');
iat[103]:=byte('2');
iat[104]:=byte('.');
iat[105]:=byte('D');
iat[106]:=byte('L');
iat[107]:=byte('L');

iat[111]:=byte('G');
iat[112]:=byte('e');
iat[113]:=byte('t');
iat[114]:=byte('P');
iat[115]:=byte('r');
iat[116]:=byte('o');
iat[117]:=byte('c');
iat[118]:=byte('A');
iat[119]:=byte('d');
iat[120]:=byte('d');
iat[121]:=byte('r');
iat[122]:=byte('e');
iat[123]:=byte('s');
iat[124]:=byte('s');

iat[128]:=byte('G');
iat[129]:=byte('e');
iat[130]:=byte('t');
iat[131]:=byte('M');
iat[132]:=byte('o');
iat[133]:=byte('d');
iat[134]:=byte('u');
iat[135]:=byte('l');
iat[136]:=byte('e');
iat[137]:=byte('H');
iat[138]:=byte('a');
iat[139]:=byte('n');
iat[140]:=byte('d');
iat[141]:=byte('l');
iat[142]:=byte('e');
iat[143]:=byte('A');

iat[147]:=byte('L');
iat[148]:=byte('o');
iat[149]:=byte('a');
iat[150]:=byte('d');
iat[151]:=byte('L');
iat[152]:=byte('i');
iat[153]:=byte('b');
iat[154]:=byte('r');
iat[155]:=byte('a');
iat[156]:=byte('r');
iat[157]:=byte('y');
iat[158]:=byte('A');

iat[162]:=byte('M');
iat[163]:=byte('e');
iat[164]:=byte('s');
iat[165]:=byte('s');
iat[166]:=byte('a');
iat[167]:=byte('g');
iat[168]:=byte('e');
iat[169]:=byte('B');
iat[170]:=byte('o');
```

```
iat[171]:=byte('x');
iat[172]:=byte('A');

end;

procedure PE_Loader; assembler;
begin
asm
// IAT
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
```

```
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL

ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
NOP //177 BYTES

jmp @next
// Розподіл пам’яті
// GetProcAddress  14

ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL

// GetModuleHandle  15
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL

// GlobalAlloc 11
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
// Kernel32.dll   12
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
// User32.dll    10
ADD BYTE PTR [EAX],AL
```

```
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
// Kernel base
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL
// user32 base
ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL

ADD BYTE PTR [EAX],AL
ADD BYTE PTR [EAX],AL


@next:
PUSHAD
push $11223344
mov eax, $11223344
call [eax]          // GetModuleHandleA
push $11223344     // GlobalAlloc
push eax
mov eax, $11223344 // GetProcaAddr
call [eax]
push $11223344    // Розмір блоків
push $40
call eax
mov [$11223344], eax  // зберігаємо адресу  globalalloc
mov edi,eax
mov esi, $11223344
pushad

    cld
    mov    dl, 80h
    xor    ebx, ebx

@literal:
    movsb
    mov    bl, 2
@nexttag:
    call   @getbit
    jnc    @literal

    xor    ecx, ecx
    call   @getbit
    jnc    @codepair
    xor    eax, eax
    call   @getbit
    jnc    @shortmatch
    mov    bl, 2
    inc    ecx
    mov    al, 10h
@getmorebits:
    call   @getbit
    adc    al, al
    jnc    @getmorebits
    jnz    @domatch
    stosb
    jmp    @nexttag
@codepair:
    call   @getgamma_no_ecx
    sub    ecx, ebx
    jnz    @normalcodepair
    call   @getgamma
    jmp    @domatch_lastpos

@shortmatch:
    lodsb
    shr    eax, 1
```

```
    jz      @donedepacking
    adc     ecx, ecx
    jmp     @domatch_with_2inc

@normalcodepair:
    xchg    eax, ecx
    dec     eax
    shl     eax, 8
    lodsb
    call    @getgamma
    cmp     eax, 32000
    jae     @domatch_with_2inc
    cmp     ah, 5
    jae     @domatch_with_inc
    cmp     eax, 7fh
    ja      @domatch_new_lastpos

@domatch_with_2inc:
    inc     ecx

@domatch_with_inc:
    inc     ecx

@domatch_new_lastpos:
    xchg    eax, ebp
@domatch_lastpos:
    mov     eax, ebp

    mov     bl, 1

@domatch:
    push    esi
    mov     esi, edi
    sub     esi, eax
    rep     movsb
    pop     esi
    jmp     @nexttag

@getbit:
    add      dl, dl
    jnz      @stillbitsleft
    mov      dl, [esi]
    inc      esi
    adc      dl, dl
@stillbitsleft:
   ret
@getgamma:
    xor     ecx, ecx
@getgamma_no_ecx:
    inc     ecx
@getgammaloop:
    call    @getbit
    adc     ecx, ecx
    call    @getbit
    jc      @getgammaloop
    ret
@donedepacking:
    POPAD

// Копіюємо до блоку програми
mov ecx,$11223344
//mov ecx, $11223344 // розмір блоку
@loopim:
// Резервуємо
MOV EBX, [ECX+EAX]  // MOV EAX, [ECX+GlobalMem]
MOV [ECX+$11223344],EBX   // MOV [ECX+SectionAddr+imagebase],EAX
LOOP @loopim
NOP
NOP
```

```
// Імпортуємо відновлення

 MOV EDX, $11223344 //основа коду
 MOV ESI, $11223344 // початковий iat rva
 ADD ESI, EDX
@dum6:
 MOV EAX, [ESI+$0C]
 TEST EAX,EAX
 JE @end
 ADD EAX,EDX
 MOV EBX,EAX
push eax
mov eax, $11223344
call [eax]          // GetModuleHandleA
 TEST EAX,EAX
 JNZ @dum1
 PUSH EBX
mov eax, $11223344
call [eax]          // LoadLibraryA
@dum1:
 MOV [$11223344],EAX  // працюємо з буфером 1
 MOV [$11223344],0     // працюємо з буфером 2
@dum5:
 MOV EDX, $1122344
 MOV EAX, [ESI]
 TEST EAX, EAX
 JNZ @dum2
 MOV EAX,[ESI+$10]
@dum2:
 ADD EAX, EDX
 ADD EAX, [$11223344]  // працюємо з буфером 2
 MOV EBX,[EAX]
 MOV EDI,[ESI+$10]
 ADD EDI,EDX
 ADD EDI,[$11223344]  // працюємо з буфером 2
 TEST EBX, EBX
 JE @dum3
 TEST EBX, $80000000
 JNZ @dum4
 ADD EBX,EDX
 INC EBX
 INC EBX
@dum4:
 AND EBX, $0FFFFFFF
 PUSH EBX
 PUSH [$11223344]  // працюємо з буфером
mov eax, $11223344
call [eax]          // GetProcAddress
 MOV [EDI],EAX
 ADD [$11223344],4  // працюємо з буфером 2
 JMP @dum5
@dum3:
 ADD ESI,$14
 MOV EDX, $11223344 //imagebase
 JMP @dum6
@end:

// Free mem
push $11223344
mov eax, $11223344
call [eax]          // GetModuleHandleA
push $11223344     // GlobalFree
push eax
mov eax, $11223344 // GetProcaAddr
call [eax]
mov edx, [$11223344] // беремо показчик до пам'яті
push edx
call eax
```

```
// Jmp to oep
popad
mov edx, $11223344
jmp edx
nop

//============================
  retn
      INC ESP     //'D'
      INC EBP     //'E'
      PUSH EAX//'P'
      INC ECX     //'A'
      INC EBX     //'C'
      DEC EBX     //'K'
      INC EBP     //'E'
      DEC ESI     //'N'
      INC ESP     //'D'
end;
end;

function CallBack(w0, w1, w2 : DWORD; cbparam : Pointer) : DWORD;
begin
  with frmMain do
  begin
    PB.Position    := Round(w1/CurFileSz*100);
    ClearLog;
    AddLog('Зачекайте будь ласка...',0);
    Application.ProcessMessages;
      Result := aP_pack_continue;
  end;
end;

function PackSection(source1:pointer; size:dword):pointer;
begin

  frmMain.CurFileSz:=size;
  frmMain.aPLib.Source   := source1;
  frmMain.aPLib.Length   := size;
  frmMain.aPlib.CallBack := @CallBack;

 frmMain.aPLib.Pack;
  frmMain.ClearLog;
    PACKEDSECTION:= frmMain.aPLib.Length;

  if frmMain.aPLib.Length = 0 then Exit;

    frmMain.AddLog('Розмір блоків: '+inttostr(frmMain.CurFileSz)+' byte(s)',0);
    frmMain.AddLog(' Упакований  блок: '+inttostr(PACKEDSECTION)+' byte(s)',0);
    frmMain.AddLog(' Розмір депакування:
'+inttostr(GetLoaderSize(dword(@PE_Loader)))+' byte(s)',0);
    frmMain.AddLog(FormatFloat('Ratio: ##%',
(packedsection*100)/frmMain.CurFileSz),0);

result:=frmMain.aPLib.Destination;
end;

procedure InsertString(where:pointer; str:string; offs:dword);
var writ:cardinal;
begin
asm
mov eax, where
add eax, offs
mov where, eax
end;
WriteProcessMemory(GetCurrentProcess(),where,pointer(str),length(str),writ);
end;

procedure InsertBytes(where:pointer; to1:string; size:dword; offs:dword);
```

```
var writ:cardinal;
begin
asm
mov eax, where
add eax, offs
mov where, eax
end;
WriteProcessMemory(GetCurrentProcess(),where,pointer(to1),size,writ);
end;

function Reversed(slovo:dword):dword; assembler;
asm
mov eax, slovo
XCHG AL,AH
ROL EAX,16
XCHG AL,AH
mov result, eax
end;

/// Упаковник коду


procedure TfrmMain.Pack(InputFileName: string);
var wr:cardinal;
begin
if (InputFileName='') or (fileexists(InputFileName)=false) then
   begin
   ClearLog;
   AddLog('Файл не відкрито',2);
     Exit;
   end;
 hFile:=CreateFileA(pchar(InputFileName), GENERIC_READ + GENERIC_WRITE,
FILE_SHARE_READ + FILE_SHARE_WRITE, NIL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL,
0);
   if hFile=INVALID_HANDLE_VALUE then
    begin
   ClearLog;
   AddLog('Файл не відкрито',2);
     Exit;
    end;
  ReadFile(hFile,EXE,2,bread,NIL);
   if EXE<>$5A4D then
    begin
     CloseHandle(hFile);
     Exit;
    end;
   SetFilePointer(hFile,$3C,NIL,FILE_BEGIN);
   ReadFile(hFile,e_lfanew,4,bread,NIL);
   SetFilePointer(hFile,e_lfanew,NIL,FILE_BEGIN);
   ReadFile(hFile,PE_HEADER,SizeOF(PE_HEADER),bread,NIL);
   if PE_HEADER.IMAGE_NT_SIGNATURE<>$00004550 then
    begin
   ClearLog;
   AddLog('Невірний формат виконуваємого файлу',2);
      CloseHandle(hFile);
      Exit;
    end;
   for i:=1 to PE_HEADER.FILE_HEADER.NumberOfSections Do
   ReadFile(hFile,SECTION_HEADER[i],SizeOF(Section),bread,NIL);

   frmMain.StatusBar1.Panels.Items[0].Text:='Packing....';
   ClearLog;


epreal:=PE_HEADER.OPTIONAL_HEADER.AddressOfEntryPoint+PE_HEADER.OPTIONAL_HEADER.
ImageBase;
   ImageBase:=PE_HEADER.OPTIONAL_HEADER.ImageBase;
   EP := RVA2Offset(epreal - ImageBase);
```

```
   num:=PE_HEADER.FILE_HEADER.NumberOfSections;
   fa:=PE_HEADER.OPTIONAL_HEADER.FileAlignment;
   sa:=PE_HEADER.OPTIONAL_HEADER.SectionAlignment;
   блок_HEADER[num].SizeOfRawData:=GetFileSize(hFile,nil)-
SECTION_HEADER[num].PointerToRawData;
   блок_HEADER[num].VirtualSize:=SECTION_HEADER[num].SizeOfRawData;

// Додаємо блок  депакування
PE_HEADER.FILE_HEADER.NumberOfSections:=PE_HEADER.FILE_HEADER.NumberOfSections+1
;
SECTION_HEADER[num+1].Name:='.data';
SECTION_HEADER[num+1].Characteristics:=$C0000040; // NOT EXECUTABLE!
SECTION_HEADER[num+1].PointerToRawData:=((SECTION_HEADER[num].PointerToRawData+S
ECTION_HEADER[num].SizeOfRawData+fa-1) div fa)*fa;
SECTION_HEADER[num+1].VirtualAddress:=((SECTION_HEADER[num].VirtualAddress+SECTI
ON_HEADER[num].VirtualSize+sa-1) div sa)*sa;
SECTION_HEADER[num+1].VirtualSize:=$400;
SECTION_HEADER[num+1].SizeOfRawData:=$400;
PE_HEADER.OPTIONAL_HEADER.SizeOfImage:=SECTION_HEADER[num+1].VirtualAddress+SECT
ION_HEADER[num+1].VirtualSize;

   ns:= блок_HEADER[num].PointerToRawData+SECTION_HEADER[num].SizeOfRawData;
nv:=SECTION_HEADER[num+1].VirtualAddress;


  for i:=1 to PE_HEADER.FILE_HEADER.NumberOfSections do
   begin
   if (ep>=SECTION_HEADER[i].PointerToRawData)and
   (ep<(SECTION_HEADER[i].SizeOfRawData+SECTION_HEADER[i].PointerToRawData))
then begin
   epsec:=i; break;
   end;
   end;

   DEPBEGIN:=nv+imagebase;
   sizeofsec:=SECTION_HEADER[1].SizeOfRawData;


iatrva:=PE_HEADER.OPTIONAL_HEADER.IMAGE_DIRECTORY_ENTRIES.IMPORT.VirtualAddress;

// Записуємо редактуємі дані
stra:='1234';

InsertString(@PE_Loader,'GlobalAlloc',179);
InsertString(@PE_Loader,'GlobalFree',191);
// Записуємо число секторів
// Push Kernel32
asm
mov eax, stra
mov ebx, DEPBEGIN
add ebx, $54
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$101);
// mov eax, GetModuleHandleA
asm
mov eax, stra
mov ebx, DEPBEGIN
add ebx, $48
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$106);

asm
mov eax, stra
mov ebx, DEPBEGIN
add ebx, 179
mov [eax], ebx
end;
```

```
InsertBytes(@PE_Loader,stra,4,$10D);

asm
mov eax, stra
mov ebx, DEPBEGIN
add ebx, $44
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$113);

// завантажуємо розмір  секторів
asm
mov eax, stra
mov ebx, sizeofsec
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$11A);
// зберігаємо адресу  globalallocess
asm
mov eax, stra
mov ebx, DEPBEGIN
add ebx, 202
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$124);
// mov edi, блок_HEADER[1].PointerToRawData
addrsec:=SECTION_HEADER[1].VirtualAddress+imagebase;
asm
mov eax, stra
mov ebx, addrsec
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$12B);
// mov ecx, sizeofsec
asm
mov eax, stra
mov ebx, sizeofsec
sub ebx, 4
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$1C8);

// loop addr
asm
mov eax, stra
mov ebx, addrsec
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$1D1);

// записуємо imagebase
asm
mov eax, stra
mov ebx, imagebase
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$1DA);
// заданий  iat rva
asm
mov eax, stra
mov ebx, iatrva
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$1DF);
// mov eax, GetModuleHandle
asm
mov eax, stra
mov ebx, DEPBEGIN
add ebx, $48
```

```
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$1F6);
// LoadLibraryA
asm
mov eax, stra
mov ebx, DEPBEGIN
add ebx, $4C
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$202);

// mov робота  з буфером 1
asm
mov eax, stra
mov ebx, DEPBEGIN
add ebx, 206
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$20A);

// mov робота  з буфером 2
asm
mov eax, stra
mov ebx, DEPBEGIN
add ebx, 210
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$210);
// записуємо imagebase
asm
mov eax, stra
mov ebx, imagebase
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$219);
InsertBytes(@PE_Loader,stra,4,$26E);
// робота  з буфером 2
asm
mov eax, stra
mov ebx, DEPBEGIN
add ebx, 210
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$22A);
InsertBytes(@PE_Loader,stra,4,$237);
InsertBytes(@PE_Loader,stra,4,$263);

// mov робота  з буфером 1
asm
mov eax, stra
mov ebx, DEPBEGIN
add ebx, 206
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$254);
// GetProcAddress
asm
mov eax, stra
mov ebx, DEPBEGIN
add ebx, $44
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$259);
//======================

// Push Kernel32
asm
mov eax, stra
```

```
mov ebx, DEPBEGIN
add ebx, $54
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$278);
// mov eax, GetModuleHandleA
asm
mov eax, stra
mov ebx, DEPBEGIN
add ebx, $48
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$27d);

asm
mov eax, stra
mov ebx, DEPBEGIN
add ebx, 191
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$284);

asm
mov eax, stra
mov ebx, DEPBEGIN
add ebx, $44
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$28A);

asm
mov eax, stra
mov ebx, DEPBEGIN
add ebx, 202
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$292);

// OEP
asm
mov eax, stra
mov ebx, epreal
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$29B);


ImportTable;
//Записуємо  IAT до депакувальника
WriteProcessMemory(GetCurrentProcess(),@PE_Loader,@iat,sizeof(iat),wr);


// Депакування
   temp1:=pointer(GlobalAlloc(GMEM_ZEROINIT,SECTION_HEADER[1].SizeOfRawData));
   temp2:=pointer(GlobalAlloc(GMEM_ZEROINIT,SECTION_HEADER[1].SizeOfRawData));

   SetFilePointer(hFile,SECTION_HEADER[1].PointerToRawData,NIL,FILE_BEGIN);
   ReadFile(hFile,temp1^,SECTION_HEADER[1].SizeOfRawData,bread,NIL);

   PACKEDPOS:= PackSection(pointer(temp1),SECTION_HEADER[1].SizeOfRawData);

// Очищуємо блок
   SetFilePointer(hFile,SECTION_HEADER[1].PointerToRawData,NIL,FILE_BEGIN);
   WriteFile(hFile,temp2^,SECTION_HEADER[1].SizeOfRawData,bread,NIL);
   GlobalFree(cardinal(temp2));

   SetFilePointer(hFile,SECTION_HEADER[1].PointerToRawData,NIL,FILE_BEGIN);
   WriteFile(hFile,PACKEDPOS^,PACKEDSECTION,bread,NIL);
    GlobalFree(cardinal(temp1));
```

```
    // закінчуємо запис упакованих блоків
PE_HEADER.OPTIONAL_HEADER.AddressOfEntryPoint:=nv+$0FF;
PE_HEADER.OPTIONAL_HEADER.IMAGE_DIRECTORY_ENTRIES.IMPORT.VirtualAddress:=nv;
PE_HEADER.OPTIONAL_HEADER.IMAGE_DIRECTORY_ENTRIES.IMPORT.Size:=$b1;

   SetFilePointer(hFile,ns,NIL,FILE_BEGIN);
   temp2:=@PE_Loader;
   WriteFile(hFile,temp2^,GetLoaderSize(dword(@PE_Loader)),bread,NIL);

   for i:=1 to PE_HEADER.FILE_HEADER.NumberOfSections do
   блок_HEADER[i].Characteristics:=$E00000E0;

   SetFilePointer(hFile,e_lfanew,NIL,FILE_BEGIN);
   WriteFile(hFile,PE_HEADER,SizeOF(PE_HEADER),bread,NIL);
   for i:=1 to PE_HEADER.FILE_HEADER.NumberOfSections do
   WriteFile(hFile,SECTION_HEADER[i],SizeOF(Section),bread,NIL);

   CloseHandle(hFile);
   frmMain.StatusBar1.Panels.Items[0].Text:='Оптимізація...';
   AddLog('Оптимізація...',0);

   pe:=pe_file.Create;
   pe.LoadFromFile(InputFileName);
   pe.OptimizeHeader(true);
   pe.OptimizeFileAlignment;
   pe.FlushFileCheckSum;
   pe.OptimizeFile(true,true,true,false);
   pe.SaveToFile(InputFileName);
   pe.Free;
   AddLog('Файл успішно упаковано',0);

   frmMain.StatusBar1.Panels.Items[0].Text:='Файл успішно упаковано';
end;

procedure TfrmMain.cmdProtectClick(Sender: TObject);
begin
  ClearLog;
  try

CopyFile(pchar(txtFileName.Text),pchar(copy(txtFileName.Text,1,Length(txtFileNam
e.Text)-4)+'.bak'),false)
  except
  end;
  frmMain.StatusBar1.Panels.Items[0].Text:='Робота';
  if (rbPacking.Checked or rbFullProtect.Checked) then Pack(txtFileName.Text);
  if (rbProtect.Checked or rbFullProtect.Checked) then begin
    frmMain.StatusBar1.Panels.Items[0].Text:='Захищено....';
    Protect(txtFileName.Text);
  end;
  frmMain.StatusBar1.Panels.Items[0].Text:='Готово';
  cmdTest.Enabled:=True;
end;

/////// Формування програмного інтерфейсу користувача //////////

procedure TfrmMain.AddLog(sText: string; sImage: integer);
begin
  with lstStatus.Items.Add do begin
    Caption:=sText;
    ImageIndex:=sImage;
  end;
end;

procedure TfrmMain.ClearLog;
begin
  lstStatus.Items.Clear;
end;
```

```
procedure TfrmMain.cmdOpenClick(Sender: TObject);
begin
  frmMain.dlgOpen.Title:='Відкрити EXE файл';
  frmMain.dlgOpen.Filter:='EXE файли (*.exe)|*.exe';
  if frmMain.dlgOpen.Execute then begin
    frmMain.txtFileName.Text:=frmMain.dlgOpen.FileName;
    cmdTest.Enabled:=False;
  end;
end;

procedure TfrmMain.FormCreate(Sender: TObject);
begin
  frmMain.lstStatus.Columns[0].Width:=frmMain.lstStatus.Width-25;
end;

procedure TfrmMain.mnuNewClick(Sender: TObject);
begin
  if (MessageDlg('Ви бажаєте встановити весь пакет програмного
забезпечення?',mtConfirmation,[mbYes, mbNo],0)=mrYes) then begin
    txtFileName.Text:='';
    rbFullProtect.Checked:=True;
    cbIcon.Checked:=True;
    cbOverlay.Checked:=True;
    txtImageBase.Text:='0';
  end;
end;

procedure TfrmMain.mnuOpenClick(Sender: TObject);
var
  strProject: ProtectorProject;
  f: file of ProtectorProject;
begin
  frmMain.dlgOpen.Title:='Open GHF проект';
  frmMain.dlgOpen.Filter:='GHF проекты (*.ghf)|*.ghf';
  if frmMain.dlgOpen.Execute then begin
    AssignFile(f,frmMain.dlgOpen.FileName);
    Reset(f);
    Read(f,strProject);
    CloseFile(f);
    txtFileName.Text:=strProject.sFileName;
    if strProject.sPacking then rbPacking.Checked:=True;
    if strProject.sProtection then rbProtect.Checked:=True;
    if (strProject.sPacking and strProject.sProtection) then
rbFullProtect.Checked:=True;
    cbIcon.Checked:=strProject.sSaveIcon;
    cbOverlay.Checked:=strProject.sSaveOverlay;
    txtImageBase.Text:=strProject.sImageBase;
    tabSheet.ActivePageIndex:=strProject.sActivePageIndex;
  end;
end;

procedure TfrmMain.mnuSaveClick(Sender: TObject);
var
  strProject: ProtectorProject;
  f: file of ProtectorProject;
begin
  frmMain.dlgSave.Title:='Зберегти GHF проект';
  frmMain.dlgSave.Filter:='GHF проекти (*.ghf)|*.ghf';
  if frmMain.dlgSave.Execute then begin
    if (ExtractFileExt(frmMain.dlgSave.FileName)='') then
frmMain.dlgSave.FileName:=frmMain.dlgSave.FileName+'.ghf';
    strProject.sFileName:=txtFileName.Text;
    if rbPacking.Checked then strProject.sPacking:=True;
    if rbProtect.Checked then strProject.sProtection:=True;
    if rbFullProtect.Checked then begin
       strProject.sProtection:=True;
       strProject.sPacking:=True;
    end;
    strProject.sSaveIcon:=cbIcon.Checked;
```

```
    strProject.sSaveOverlay:=cbOverlay.Checked;
    strProject.sImageBase:=txtImageBase.Text;
    strProject.sActivePageIndex:=tabSheet.ActivePageIndex;

    AssignFile(f,frmMain.dlgSave.FileName);
    Rewrite(f);
    Write(f,strProject);
    CloseFile(f);
  end;
end;

procedure TfrmMain.mnuExitClick(Sender: TObject);
begin
  halt;
end;

procedure TfrmMain.cmdTestClick(Sender: TObject);
begin

ShellExecute(frmMain.Handle,'open',pchar(txtFileName.Text),'',pchar(ExtractFileD
ir(txtFileName.Text)),0);
end;

procedure TfrmMain.mnuAboutClick(Sender: TObject);
begin
  frmAbout.ShowModal;
end;

procedure TfrmMain.mnuHelpOpenClick(Sender: TObject);
begin
  if not (ShellExecute(frmMain.Handle,'відкрито',pchar(Application.HelpFile),
  pchar(''),pchar(ExtractFilePath(ParamStr(0))),1)=42) then
  ShowMessage('File "'+Application.HelpFile+'" не знайдено у програмній
директорії');
end;

end.
```

Файл morphine.pas – захист на основі обфускації коду

```
   unit morphine;

//Якщо RUBBISH_NOPS визначено, додаємо ложні команди та пусті цикли для
погіршання дизасемблювання

interface
{ $DEFINE RUBBISH_NOPS}
{ $DEFINE STATIC_CONTEXT}
uses Windows, SysUtils;

//
//Блок коду:
//0..$10: jmp GetProcAddress+jmp LoadLibrary+pad
//$10..$10+KeySize:Key
//$10+KeySize..$10+KeySize+sizeof(DynLoader):DynLoader
//$10+KeySize+sizeof(DynLoader): code
//
//Блок даних:
//0..sizeof(host)-1: host
//
//Блок імпорту:
//0..$70-1: imports
//
//[TlsSection:]
//0..sizeof(tls): tls
//


//
//переміщуємо функцію jmps (getprocaddress, loadlibrary) в кінець
ініціалізуючого/поліморфічного редактора коду
//для запобігання AV детектування (блок коду стартує з  ..000000FF2534.. у якому
записана сигнатура ):
//реалізовано декілька варіантів для кожного jmp для коду імпорту
(getprocaddress, loadlibrary) та додане фіксування імпортованого коду

//Це новий заплутаний код:
//
//Блок коду:
//$0..KeySize:Key
//KeySize..KeySize+sizeof(DynLoader):DynLoader
//KeySize+sizeof(DynLoader): code
//
//Блок даних:
//0..sizeof(host)-1: host
//
//Блок імпортування:
//0..$70-1: imports
//
//[TlsSection:]
//0..sizeof(tls): tls


//- деякі довільні дані(CoderRoller1) у програму заплутування (DynCoder and
Decoder)
//- блок видалення даних
//- блок незначних помилок
//
//Це новий заплутаний код:
//
//Блок коду:
//0: Rubbish
//KeyPtr..KeyPtr+KeySize:Key
//KeyPtr+KeySize..KeyPtr+KeySize+sizeof(DynLoader):DynLoader
//KeyPtr+KeySize+sizeof(DynLoader): code
```

```
//code+sizeof(code): host
//
//Блок імпортування:
//0..$70-1: imports
//
//[TlsSection:]
//0..sizeof(tls): tls

//
//- заплутаний код імен інструкцій

//- Підтримує DLL
//- введення цього коду дозволяє реалізувати помилковий для дизасемблювання код
//- введення незначних помилок
//+ .edata блок після .tls

//- заплутаний код покращено
//- відбувається шифрування основних даних

//Якщо ви маєте суму PEB, TEB структур (визначених у  DynLoader)

const
 //реалізація заглушки для DOS
 //Ця програма записує  "Ця програма не працює під  DOS режимом"
 DosStub:array[0..$38-1] of Byte=
 ($BA,$10,$00,$0E,$1F,$B4,$09,$CD,$21,$B8,$01,$4C,$CD,$21,$90,$90,
  $54,$68,$69,$73,$20,$70,$72,$6F,$67,$72,$61,$6D,$20,$6D,$75,$73,
  $74,$20,$62,$65,$20,$72,$75,$6E,$20,$75,$6E,$64,$65,$72,$20,$57,
  $69,$6E,$33,$32,$0D,$0A,$24,$37);

 //константи блоку імпорту
 NumberOfDLL=1;                                  //кількість бібліотек
 NumberOfImports=2;                              //кількість функцій
 Kernel32Name='kernel32.dll';                    //ім'я бібліотеки
 NtdllName='ntdll.dll';                          //ім'я ntdll.dll

 GetProcAddressName='GetProcAddress';            //ім'я funct1
 LoadLibraryName='LoadLibraryA';                 //ім'я func2
 Kernel32Size=12;                                //довжина імені dll
 GetProcAddressSize=14;                          //довжина імені func1
 LoadLibrarySize=12;                             //довжина імені func2

 //індекси інструкції заплутування
 PII_BEGIN                    = 0;

 PII_POLY_BEGIN               = PII_BEGIN;
 PII_POLY_MOV_REG_LOADER_SIZE = PII_POLY_BEGIN;
 PII_POLY_MOV_REG_LOADER_ADDR = PII_POLY_MOV_REG_LOADER_SIZE+1;

 PII_CODER_BEGIN              = PII_POLY_MOV_REG_LOADER_ADDR+1;
 PII_CODER_CALL_GET_EIP       = PII_CODER_BEGIN+1;
 PII_CODER_GET_EIP            = PII_CODER_CALL_GET_EIP+1;
 PII_CODER_FIX_DST_PTR        = PII_CODER_GET_EIP+1;
 PII_CODER_KEY_START          = PII_CODER_FIX_DST_PTR+1;
 PII_CODER_MOV_REG_KEY        = PII_CODER_KEY_START;
 PII_CODER_FIX_SRC_PTR        = PII_CODER_MOV_REG_KEY+1;

 PII_CODER_CODE               = PII_CODER_FIX_SRC_PTR+1;
 PII_CODER_LOAD_KEY_TO_REG    = PII_CODER_CODE;
 PII_CODER_TEST_KEY_END       = PII_CODER_LOAD_KEY_TO_REG+1;
 PII_CODER_JZ_CODER_BEGIN     = PII_CODER_TEST_KEY_END+1;
 PII_CODER_ADD_DATA_IDX       = PII_CODER_JZ_CODER_BEGIN+1;
 PII_CODER_XOR_DATA_REG       = PII_CODER_ADD_DATA_IDX+1;
 PII_CODER_STORE_DATA         = PII_CODER_XOR_DATA_REG+1;
 PII_CODER_INC_SRC_PTR        = PII_CODER_STORE_DATA+1;
 PII_CODER_LOOP_CODER_CODE    = PII_CODER_INC_SRC_PTR+1;
 PII_CODER_END                = PII_CODER_LOOP_CODER_CODE+1;

 PII_POLY_JMP_DYNLOADER       = PII_CODER_END+1;
```

```
 PII_POLY_END                    = PII_POLY_JMP_DYNLOADER;
 PII_END                         = PII_POLY_END;

 //інші константи
 MaxPolyCount=20;                                  //максимальна кількість
варіантів для однієї інструкції
 InitInstrCount=PII_END+1;                         //редактор лічильника інструкцій
 RawDataAlignment=$200;                            //вирівнювання SizeOfRawData
 DosStubEndSize=$88;                               //$100 - SizeOf(DosStub)

 //image type const
 IMAGE_TYPE_EXE=0;
 IMAGE_TYPE_DLL=1;
 IMAGE_TYPE_SYS=2;
 IMAGE_TYPE_UNKNOWN=$FFFFFFFF;

 //це двійне слово закінчення  DYN_LOADER у формі декодування
 DYN_LOADER_END_MAGIC=$C0DEC0DE;
 DYN_LOADER_DEC_MAGIC=$1EE7C0DE;

 //регістри
 REG_EAX=0;
 REG_ECX=1;
 REG_EDX=2;
 REG_EBX=3;
 REG_ESP=4;
 REG_EBP=5;
 REG_ESI=6;
 REG_EDI=7;
 REG_NON=255;

 Reg8Count=8;
 Reg16Count=8;
 Reg32Count=8;

 RT_XP_MANIFEST=24;

type
 //тепер декілька типів неможливо знайти у windows.pas

 PImageImportByName=^TImageImportByName;
 TImageImportByName=packed record
  Hint:Word;
  Name:array of Char;
 end;
 PImageThunkData=^TImageThunkData;
 TImageThunkData=packed record
  case Byte of
   0:(ForwarderString:PByte);
   1:(FunctionPtr:PCardinal);
   2:(Ordinal:Cardinal);
   3:(AddressOfData:PImageImportByName);
 end;
 PImageImportDescriptor=^TImageImportDescriptor;
 TImageImportDescriptor=packed record
  case Byte of

0:(Characteristics,cTimeDateStamp,cForwarderChain,cName:Cardinal;cFirstThunk:PIm
ageThunkData);

1:(OriginalFirstThunk:PImageThunkData;oTimeDateStamp,oForwarderChain,oName:Cardi
nal;oFirstThunk:PImageThunkData);
 end;

 PExportDirectoryTable=^TExportDirectoryTable;
 TExportDirectoryTable=packed record
  Flags,TimeStamp:Cardinal;
  MajorVersion,MinorVersion:Word;
```

```
NameRVA,OrdinalBase,AddressTableEntries,NumberOfNamePointers,ExportAddressTableR
VA,
  NamePointerRVA,OrdinalTableRVA:Cardinal;
 end;

 //ось так подібно блоку .tls
 PTlsSectionData=^TTlsSectionData;
 TTlsSectionData=packed record

RawDataStart,RawDataEnd,AddressOfIndex,AddressOfCallbacks,SizeOfZeroFill,Charact
eristics:Cardinal;
 end;

 //мій тип для блоку  tls
 TTlsCopy=record
  Directory:PImageDataDirectory;
  блокData:PTlsSectionData;
  RawData:Pointer;
  RawDataLen,Index:Cardinal;
  Callbacks:Pointer;
  CallbacksLen:Cardinal;
 end;

 //одна псевдо інструкція (p-i) для движка перетворень коду (може містити більш
ніж одну x86 інструкцію)
 TInstruction=packed record
  Len:Byte;                                    //довжина заплутаного коду
  Fix1,Fix2,Fix3,Fix4:Byte;                    //байти індексування для
фіксації
  Code:array[0..30] of Char;                   //заплутаний код
 end;

 //список  p-i, який обирається кожний раз при операції заплутування коду
 TVarInstruction=packed record
  Count,Index:Byte;                            //число p-i та число можливостей
вибору
  VirtualAddress:Cardinal;                     //адреса інструкції у блоці
CODE
  Vars:array[0..MaxPolyCount-1] of TInstruction;//список
 end;

 PResourceDirectoryTable=^TResourceDirectoryTable;
 TResourceDirectoryTable=packed record
  Characteristics:Cardinal;
  TimeDateStamp:Cardinal;
  MajorVersion:Word;
  MinorVersion:Word;
  NumberOfNameEntries:Word;
  NumberOfIDEntries:Word;
 end;

 PResourceDirectoryEntry=^TResourceDirectoryEntry;
 TResourceDirectoryEntry=packed record
  NameID:Cardinal;
  SubdirDataRVA:Cardinal;
 end;

 PResourceDataEntry=^TResourceDataEntry;
 TResourceDataEntry=packed record
  DataRVA:Cardinal;
  Size:Cardinal;
  Codepage:Cardinal;
  Reserved:Cardinal;
 end;

 PResourceTableDirectoryEntry=^TResourceTableDirectoryEntry;
 TResourceTableDirectoryEntry=packed record
  Table:TResourceDirectoryTable;
```

```
  Directory:TResourceDirectoryEntry;
 end;

 PIconDirectoryEntry=^TIconDirectoryEntry;
 TIconDirectoryEntry=packed record
  Width:Byte;
  Height:Byte;
  ColorCount:Byte;
  Reserved:Byte;
  Planes:Word;
  BitCount:Word;
  BytesInRes:Cardinal;
  ID:Word;
 end;

 PIconDirectory=^TIconDirectory;
 TIconDirectory=packed record
  Reserved:Word;
  ResType:Word;
  Count:Word;
  Entries:array[0..31] of TIconDirectoryEntry;
 end;

 TImageType=(itExe,itDLL,itSys);

 TEncoderProc=function(AAddr:Pointer):Cardinal; stdcall;
 procedure Protect(InputFileName: string);


var
 DosHeader:TImageDosHeader;
 DosStubEnd:array[0..DosStubEndSize-1] of Char;
 NtHeaders:TImageNtHeaders;
 FileHandle,MainFile:THandle;
 InputFileName,OutputFileName,Options:string;

NumBytes,TotalFileSize,MainSize,LoaderSize,VirtLoaderData,VirtMainData,VirtKey,I
nitSize,KeyPtr,

AnyDWORD,LoaderPtr,TlsSectionSize,Delta,HostImageBase,HostSizeOfImage,HostCharac
teristics,

ReqImageBase,RandomValue,ExportSectionSize,CurVirtAddr,CurRawData,ExportRVADelta
,
 HostExportSectionVirtualAddress,ExportNamePointerRVAOrg,ExportAddressRVAOrg,
 ImportSectionDataSize,HostImportSectionSize,ImportSectionDLLCount,

HostImportSectionVirtualAddress,InitcodeThunk,CodeSectionVirtualSize,LoaderRealS
ize,

MainRealSize,MainRealSize4,LogCnt,MainDataDecoderLen,DynLoaderDecoderOffset,LdrP
trCode,LdrPtrThunk,

 ResourceSectionSize,HostResourceSectionSize,ResourceIconGroupDataSize,HostResour
ceSectionVirtualAddress,
 ResourceXPMDirSize,AfterImageOverlaysSize:Cardinal;

CodeSection,ExportSection,TlsSection,ImportSection,ResourceSection:TImageSection
Header;
 ImportDesc,NullDesc:TImageImportDescriptor;
 PImportDesc:PImageImportDescriptor;
 ThunkGetProcAddress,ThunkLoadLibrary:TImageThunkData;
 NullWord,KeySize,TrashSize,Trash2Size,HostSubsystem:Word;

MainData,MainDataCyp,LoaderData,Key,InitData,Trash,Trash2,Ptr,ExportData,ImportS
ectionData,ResourceData,
 MainDataEncoder,MainDataDecoder,AfterImageOverlays:Pointer;
 PB,PB2,PB3,PB4,DynLoaderSub,LdrPtr,MainDataDecPtr:PByte;
```

```
TlsSectionPresent,ExportSectionPresent,Quiet,DynamicDLL,ResourceSectionPresent,S
aveIcon,
 SaveOverlay,OverlayPresent:Boolean;
 TlsCopy:TTlsCopy;
 TlsSectionData:TTlsSectionData;
 ImageType:TImageType;
 I:Integer;
 DynLoaderJmp:PCardinal;
 ResourceRoot,ResourceIconGroup,ResourceXPManifest:PResourceDirectoryTable;
 ResourceDirEntry:PResourceDirectoryEntry;
 EncoderProc:TEncoderProc;

implementation

uses maincode;

procedure DynLoader; assembler; stdcall;
//Завантажувач
//він завантажує ре файли до пам'яті з MainData
//фіксуються переміщення
//фіксуються імпортування
//фіксуються експортування
//
asm
  push 012345678h               //LoadLibrary
  push 012345678h               //GetProcAddress
  push 012345678h               //Addr of MainData
  //операція заплутування
  //використовується rva для основниного коду, не знаючи базового образу
  //який отримується  eip та починає робити з 0FFFFF000h
  //по 000401XXXh приблизно 000401000h , саме тому
  //код використовується до 2000h
  call @get_eip
  @get_eip:
  pop eax
  and eax,0FFFFF000h
  add [esp],eax
  add [esp+004h],eax
  add [esp+008h],eax

  call @DynLoader_begin

  //ще один метод заплутування
  //код у LoadLibrary який викликає DllMain зберігає esp у esi
  //якщо змінюється esi то ми додаємо зліва його від esp, та правдиве його
значення
  //додаємо суму 010h для параметрів DllMain + повертаємо адресу заплутаного
коду
  mov esi,esp
  mov [esi+004h],ecx            //змінюємо DllMain.hinstDLL
  add esi,010h
  jmp eax                       //переходимо до наступної точки зміни

  @DynLoader_begin:
  //маємо базовий образ коду у  eax (except ax), зберігаємо його у ebp-050h
  push ebp
  mov ebp,esp
  sub esp,00000200h
  {
   -01F8..-0100 -       NtHeaders:TImageNtHeaders
   -09C         -       MemoryBasicInformation.BaseAddress
   -098         -       MemoryBasicInformation.AllocationBase
   -094         -       MemoryBasicInformation.AllocationProtect
   -090         -       MemoryBasicInformation.RegionSize
   -08C         -       MemoryBasicInformation.State
   -088         -       MemoryBasicInformation.Protect
   -084         -       MemoryBasicInformation.Type
```

```
 -07C         -       IsBadReadPtr:Pointer
 -078         -       VirtualQuery:Pointer
 -074         -       VirtualProtect:Pointer
 -070         -       FirstModule:Cardinal

 -054         -       OrgImageSize:Cardinal
 -050         -       ImageBase:Cardinal
 -04C         -       ImageEntryPoint:Cardinal
 -048         -       ImageSize:Cardinal
 -044         -       ImageType:Cardinal
 -040         -       HintName:Cardinal
 -03C         -       Thunk:Cardinal
 -038..-010   -       блок:TImageSectionHeader
 -00C         -       FileData:Pointer
 -008         -       ImageSizeOrg:Cardinal
 -004         -       ImageBaseOrg:Cardinal
 +008         -       AddrOfMainData:Pointer
 +00C         -       GetProcAddress:Pointer
 +010         -       LoadLibrary:Pointer
}
push ebx                                //зберігаємо ebx, edi, esi
push edi
push esi

and eax,0FFFF0000h

mov [ebp-050h],eax                      //зберігаємо ImageBase

mov ecx,00008000h
@DynLoader_fake_loop:
add eax,0AF631837h
xor ebx,eax
add bx,ax
rol ebx,007h
loop @DynLoader_fake_loop
//Включаємо криптo програму заплутування
//esp та ebp не повинні змінюватися
push dword ptr [ebp+008h]               //AAddr
dd DYN_LOADER_DEC_MAGIC
//\кінець криптоперетворень

call @DynLoader_fill_image_info

push 000h
push 06C6C642Eh
push 032336C65h
push 06E72656Bh                         //kernel32.dll до стеку
push esp                                //lpLibFileName
mov eax,[ebp+010h]                      //ImportThunk.LoadLibrary
call [eax]                              //LoadLibrary
add esp,010h
mov edi,eax

push 000h
push 0636F6C6Ch
push 0416C6175h
push 074726956h                         //VirtualAlloc до стеку
push esp                                //lpProcName
push eax                                //hModule
mov eax,[ebp+00Ch]                      // реалізація ImportThunk.GetProcAddress
call [eax]                              //GetProcAddress
add esp,010h
mov ebx,eax
test eax,eax
jz @DynLoader_end

push 000007463h
push 065746f72h
push 0506C6175h
```

```
push 074726956h                         //VirtualProtect до стеку
push esp                                //lpProcName
push edi                                //hModule
mov eax,[ebp+00Ch]                      // реалізація ImportThunk.GetProcAddress
call [eax]                              //GetProcAddress
add esp,010h
mov [ebp-074h],eax                      //VirtualProtect
test eax,eax
jz @DynLoader_end

push 000h
push 079726575h
push 0516C6175h
push 074726956h                         //VirtualQuery до стеку
push esp                                //lpProcName
push edi                                //hModule
mov eax,[ebp+00Ch]                      // реалізація ImportThunk.GetProcAddress
call [eax]                              //GetProcAddress
add esp,010h
mov [ebp-078h],eax                      //VirtualQuery
test eax,eax
jz @DynLoader_end

push 000h
push 072745064h
push 061655264h
push 061427349h                         //IsBadReadPtr до стеку
push esp                                //lpProcName
push edi                                //hModule
mov eax,[ebp+00Ch]                      // реалізація ImportThunk.GetProcAddress
call [eax]                              //GetProcAddress
add esp,010h
mov [ebp-07Ch],eax                      //IsBadReadPtr
test eax,eax
jz @DynLoader_end


lea edi,[ebp-01F8h]                     //NtHeaders
push edi
mov esi,[ebp+008h]                      //TImageDosHeader
add esi,[esi+03Ch]                      //TImageDosHeader._lfanew
push 03Eh                               //SizeOf(NtHeaders) div 4
pop ecx
rep movsd
pop edi
mov eax,[edi+034h]                      //NtHeaders.OptionalHeader.ImageBase
mov [ebp-004h],eax                      //ImageBaseOrg
mov ecx,[edi+050h]                      //NtHeaders.OptionalHeader.SizeOfImage
mov [ebp-008h],ecx                      //ImageSizeOrg

push ecx
push PAGE_EXECUTE_READWRITE             //flProtect
push MEM_COMMIT or MEM_RESERVE          //flAllocationType
push ecx                                //dwSize
push eax                                //lpAddress
call ebx                                //VirtualAlloc
pop ecx
test eax,eax
jnz @DynLoader_alloc_done

push PAGE_EXECUTE_READWRITE             //flProtect
push MEM_COMMIT                         //flAllocationType
push ecx                                //dwSize
push eax                                //lpAddress
call ebx                                //VirtualAlloc
test eax,eax
jz @DynLoader_end

@DynLoader_alloc_done:
```

```
  mov [ebp-00Ch],eax                    //FileData
  mov edi,eax
  mov esi,[ebp+008h]                    //TImageDosHeader
  push esi
  mov ecx,esi                           //TImageDosHeader
  add ecx,[esi+03Ch]                    //+TImageDosHeader._lfanew = NtHeaders
  mov ecx,[ecx+054h]                    //NtHeaders.SizeOfHeaders
  rep movsb
  pop esi
  add esi,[esi+03Ch]                    //TImageNtHeaders
  add esi,0F8h                          //+SizeOf(TImageNtHeaders) = блок
headers

 @DynLoader_LoadSections:
  mov eax,[ebp+008h]                    //TImageDosHeader
  add eax,[eax+03Ch]                    //TImageDosHeader._lfanew
  movzx eax,[eax+006h]                  //NtHeaders.FileHeader.NumberOfSections

 @DynLoader_LoadSections_do_section:
  lea edi,[ebp-038h]                    //Section
  push edi
  push 00Ah                             //SizeOf(TImageSectionHeader) div 4
  pop ecx
  rep movsd
  pop edi

 @DynLoader_LoadSections_copy_data:
  mov edx,[edi+014h]                    //Section.PointerToRawData
  test edx,edx
  jz @DynLoader_LoadSections_next_section
  push esi
  mov esi,[ebp+008h]                    //AHostAddr
  add esi,edx                           //AHostAddr + блок.PointerToRawData
  mov ecx,[edi+010h]                    //Section.SizeOfRawData
  mov edx,[edi+00Ch]                    //Section.VirtualAddress
  mov edi,[ebp-00Ch]                    //FileData
  add edi,edx                           //FileData + блок.VirtualAddress
  rep movsb
  pop esi
 @DynLoader_LoadSections_next_section:
  dec eax
  jnz @DynLoader_LoadSections_do_section

  mov edx,[ebp-00Ch]                    //FileData
  sub edx,[ebp-004h]                    //Delta = FileData - ImageBaseOrg
  je @DynLoader_PEBTEBFixup

 @DynLoader_RelocFixup:
  mov eax,[ebp-00Ch]                    //FileData
  mov ebx,eax
  add ebx,[ebx+03Ch]                    //TImageDosHeader._lfanew
  mov ebx,[ebx+0A0h]
//IMAGE_DIRECTORY_ENTRY_BASERELOC.VirtualAddress
  test ebx,ebx
  jz @DynLoader_PEBTEBFixup
  add ebx,eax
 @DynLoader_RelocFixup_block:
  mov eax,[ebx+004h]                    //ImageBaseRelocation.SizeOfBlock
  test eax,eax
  jz @DynLoader_PEBTEBFixup
  lea ecx,[eax-008h]                    //ImageBaseRelocation.SizeOfBlock -
SizeOf(TImageBaseRelocation)
  shr ecx,001h                          //(ImageBaseRelocation.SizeOfBlock -
SizeOf(TImageBaseRelocation)) div SizeOf(Word)
  lea edi,[ebx+008h]                    //PImageBaseRelocation +
SizeOf(TImageBaseRelocation)
 @DynLoader_RelocFixup_do_entry:
  movzx eax,word ptr [edi]              //Entry
  push edx
```

```
  mov edx,eax
  shr eax,00Ch                          //Type = Entry shr 12

  mov esi,[ebp-00Ch]                    //FileData
  and dx,00FFFh
  add esi,[ebx]                         //FileData +
ImageBaseRelocation.VirtualAddress
  add esi,edx                           //FileData +
ImageBaseRelocation.VirtualAddress+Entry and $0FFF
  pop edx

 @DynLoader_RelocFixup_HIGH:
  dec eax
  jnz @DynLoader_RelocFixup_LOW
  mov eax,edx
  shr eax,010h                          //HiWord(Delta)
  jmp @DynLoader_RelocFixup_LOW_fixup
 @DynLoader_RelocFixup_LOW:
  dec eax
  jnz @DynLoader_RelocFixup_HIGHLOW
  movzx eax,dx                          //LoWord(Delta)
 @DynLoader_RelocFixup_LOW_fixup:
  add word ptr [esi],ax
  jmp @DynLoader_RelocFixup_next_entry
 @DynLoader_RelocFixup_HIGHLOW:
  dec eax
  jnz @DynLoader_RelocFixup_next_entry
  add [esi],edx

 @DynLoader_RelocFixup_next_entry:
  inc edi
  inc edi                               //Inc(Entry)
  loop @DynLoader_RelocFixup_do_entry

 @DynLoader_RelocFixup_next_base:
  add ebx,[ebx+004h]                    //ImageBaseRelocation +
ImageBaseRelocation.SizeOfBlock
  jmp @DynLoader_RelocFixup_block

 @DynLoader_PEBTEBFixup:
  //існують погані вказівники у InLoadOrderModuleList, ми змінюємо базу нашого
модуля
  //і якщо ми - програма (не dll), ми повинні змінювати базову адресу у PEB
також
  //Для програм написаних на VB, це потрібно робити тут. так як бібліотеки
читають звідси дані
  //у ImportFixup блоку
//  int 3
  mov ecx,[ebp-00Ch]                    //FileData
  mov edx,[ebp-050h]                    //ImageBase
  add [ebp-04Ch],edx                    //ImageEntryPoint

  mov eax,fs:[000000030h]               //TEB.PPEB
  cmp dword ptr [ebp-044h],IMAGE_TYPE_EXE // тип змінених даних= IMAGE_TYPE_EXE
  jnz @DynLoader_in_module_list
  mov [eax+008h],ecx                    //PEB.ImageBaseAddr -> rewrite old
imagebase
 @DynLoader_in_module_list:
  mov eax,[eax+00Ch]                    //PEB.LoaderData
  mov eax,[eax+00Ch]                    //LoaderData.InLoadOrderModuleList

  //тепер неможливо знайти модуль у списку (але та же база, той же розмір та та
жа точка входу)
  mov esi,eax                           //перший запис

 @DynLoader_in_module_list_one:
  mov edx,[eax+018h]                    //InLoadOrderModuleList.BaseAddress
  cmp edx,[ebp-050h]                    //ImageBase
  jnz @DynLoader_in_module_list_next
```

```
  mov edx,[eax+01Ch]                    //InLoaderOrderModuleList.EntryPoint
  cmp edx,[ebp-04Ch]                    //ImageEntryPoint
  jnz @DynLoader_in_module_list_next
  mov edx,[eax+020h]                    //InLoaderOrderModuleList.SizeOfImage
  cmp edx,[ebp-048h]                    //ImageSize
  jnz @DynLoader_in_module_list_next
  mov [eax+018h],ecx                    //InLoadOrderModuleList.BaseAddress ->
перезапис старого запису
  add ecx,[ebp-01D0h]
//+NtHeaders.OptionalHeader.AddressOfEntryPoint
  mov [eax+01Ch],ecx                    //InLoadOrderModuleList.EntryPoint ->
перезапис старої точки входу
  mov ecx,[ebp-01A8h]                   //NtHeaders.OptionalHeader.SizeOfImage
  mov [eax+020h],ecx                    //InLoaderOrderModuleList.SizeOfImage ->
перезапис строго розміру блоку
  jmp @DynLoader_ImportFixup

 @DynLoader_in_module_list_next:
  cmp [eax],esi                         //InLoadOrderModuleList.Flink ?= перший
запис
  jz @DynLoader_ImportFixup
  mov eax,[eax]                         //запис = InLoadOrderModuleList.Flink
  jmp @DynLoader_in_module_list_one


 @DynLoader_ImportFixup:
  mov ebx,[ebp-0178h]
//NtHeaders.OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_IMPORT].VirtualAd
dress
  test ebx,ebx
  jz @DynLoader_export_fixup
  mov esi,[ebp-00Ch]                    //FileData
  add ebx,esi                           //FileData +
NtHeaders.OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_IMPORT].VirtualAddr
ess
 @DynLoader_ImportFixup_module:
  mov eax,[ebx+00Ch]                    //TImageImportDescriptor.Name
  test eax,eax
  jz @DynLoader_export_fixup

  mov ecx,[ebx+010h]                    //TImageImportDescriptor.FirstThunk
  add ecx,esi
  mov [ebp-03Ch],ecx                    // Заглушка
  mov ecx,[ebx]                         //TImageImportDescriptor.Characteristics
  test ecx,ecx
  jnz @DynLoader_ImportFixup_table
  mov ecx,[ebx+010h]
 @DynLoader_ImportFixup_table:
  add ecx,esi
  mov [ebp-040h],ecx                    //HintName
  add eax,esi                           //TImageImportDescriptor.Name + FileData
= ModuleName
  push eax                              //lpLibFileName
  mov eax,[ebp+010h]                    //ImportThunk.LoadLibrary
  call [eax]                            //LoadLibrary
  test eax,eax
  jz @DynLoader_end
  mov edi,eax
 @DynLoader_ImportFixup_loop:
  mov ecx,[ebp-040h]                    //HintName
  mov edx,[ecx]                         //TImageThunkData.Ordinal
  test edx,edx
  jz @DynLoader_ImportFixup_next_module
  test edx,080000000h                   //імпорт порядку?
  jz @DynLoader_ImportFixup_by_name
  and edx,07FFFFFFFh                    //беремо порядок
  jmp @DynLoader_ImportFixup_get_addr
 @DynLoader_ImportFixup_by_name:
```

```
  add edx,esi                           //TImageThunkData.Ordinal + FileData =
OrdinalName
  inc edx
  inc edx                               //OrdinalName.Name
 @DynLoader_ImportFixup_get_addr:
  push edx                              //lpProcName
  push edi                              //hModule
  mov eax,[ebp+00Ch]                    // реалізація ImportThunk.GetProcAddress
  call [eax]                            //GetProcAddress
  mov ecx,[ebp-03Ch]                    //HintName
  mov [ecx],eax
  add dword ptr [ebp-03Ch],004h         // Заглушка -> next Thunk
  add dword ptr [ebp-040h],004h         //HintName -> next HintName
  jmp @DynLoader_ImportFixup_loop
 @DynLoader_ImportFixup_next_module:
  add ebx,014h                          //SizeOf(TImageImportDescriptor)
  jmp @DynLoader_ImportFixup_module

 @DynLoader_export_fixup:
  // перетворюємо усі модулі та шукаємо блок IAT для нашого модуля, потім
змінюємо базу образу  у усіх імпортуємих модулях
//  int 3
  mov eax,fs:[000000030h]               //TEB.PPEB
  mov eax,[eax+00Ch]                    //PEB.LoaderData
  mov ebx,[eax+00Ch]                    //LoaderData.InLoadOrderModuleList
  mov [ebp-070h],ebx                    //FirstModule

 @DynLoader_export_fixup_process_module:
  mov edx,[ebx+018h]                    //InLoadOrderModuleList.BaseAddress
  cmp edx,[ebp-050h]                    //ImageBase
  jz @DynLoader_export_fixup_next

  push edx
  push 004h                             //ucb
  push edx                              //lp
  call [ebp-07Ch]                       //IsBadReadPtr
  pop edx
  test eax,eax
  jnz @DynLoader_export_fixup_next

  mov edi,edx
  add edi,[edi+03Ch]                    //TImageDosHeader._lfanew
  mov edi,[edi+080h]
//TImageNtHeaders.OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_IMPORT].Vir
tualAddress
  test edi,edi
  jz @DynLoader_export_fixup_next
  add edi,edx                           //+ module.ImageBase
 @DynLoader_export_fixup_check_idt:
  xor eax,eax
  push edi
  push 005h                             //sizeof(ImportDirectoryTable)/4
  pop ecx
  rep scasd                             //тест для неіснуючої директорії
  pop edi
  jz @DynLoader_export_fixup_next

  mov esi,[edi+010h]                    //Блок
імпортування.ImportAddressTableRVA
  add esi,[ebx+018h]                    //+ module.ImageBase
  mov eax,[esi]                         //first IAT func address
  sub eax,[ebp-050h]                    //- ImageBase
  jb @DynLoader_export_fixup_next_idir  // це не перетворюється
  cmp eax,[ebp-048h]                    //ImageSize
  jbe @DynLoader_export_fixup_prefixaddr // це перетворюється

 @DynLoader_export_fixup_next_idir:
  add edi,014h                          //+ sizeof(IDT) = next IDT
  jmp @DynLoader_export_fixup_check_idt
```

```
 @DynLoader_export_fixup_prefixaddr:
  push 01Ch                             //dwLength =
sizeof(MemoryBasicInformation)
  lea eax,[ebp-09Ch]                    //MemoryBasicInformation
  push eax                              //lpBuffer
  push esi                              //lpAddress
  call [ebp-078h]                       //VirtualQuery

  lea eax,[ebp-088h]                    //MemoryBasicInformation.Protect
  push eax                              //lpflOldProtect
  push PAGE_READWRITE                   //flNewProtect
  push dword ptr [ebp-090h]             //dwSize =
MemoryBasicInformation.RegionSize
  push dword ptr [ebp-09Ch]             //lpAddress =
MemoryBasicInformation.BaseAddress
  call [ebp-074h]                       //VirtualProtect
  test eax,eax
  jz @DynLoader_export_fixup_next

  push edi
  mov edi,esi
 @DynLoader_export_fixup_fixaddr:
  lodsd
  test eax,eax
  jz @DynLoader_export_fixup_protect_back
  sub eax,[ebp-050h]                    //- ImageBase
  add eax,[ebp-00Ch]                    //+ FileData
  stosd
  jmp @DynLoader_export_fixup_fixaddr

 @DynLoader_export_fixup_protect_back:
  lea eax,[ebp-084h]                    //MemoryBasicInformation.Type (just need
some pointer)
  push eax                              //lpflOldProtect
  push dword ptr [ebp-088h]             //flNewProtect =
MemoryBasicInformation.Protect
  push dword ptr [ebp-090h]             //dwSize =
MemoryBasicInformation.RegionSize
  push dword ptr [ebp-09Ch]             //lpAddress =
MemoryBasicInformation.BaseAddress
  call [ebp-074h]                       //VirtualProtect
  pop edi
  jmp @DynLoader_export_fixup_next_idir

 @DynLoader_export_fixup_next:
  mov ebx,[ebx]
  cmp ebx,[ebp-070h]                    //InLoadOrderModuleList.Flink ?=
FirstModule
  jnz @DynLoader_export_fixup_process_module

 @DynLoader_run:
//  int 3
  mov eax,[ebp-01D0h]
 //NtHeaders.OptionalHeader.AddressOfEntryPoint
  add eax,[ebp-00Ch]
 //NtHeaders.OptionalHeader.AddressOfEntryPoint + FileData = EntryPoint

 @DynLoader_end:
  mov ecx,[ebp-00Ch]                    // нам небхідно FileData
  pop esi
  pop edi
  pop ebx
  leave
  ret 00Ch

 @DynLoader_fill_image_info:
```

```
  //ці величини дають інформацію про наш образ, інформація заповнена раніше, ніж
DynLoader буде поміщений у кінцеву прграму, ми знаходимо їх компенсацію, яка
виходить з DynLoader_end, та шукає ознаку DYN_LOADER_END_MAGIC
  mov [ebp-044h],012345678h             //ImageType
  mov [ebp-048h],012345678h             //ImageSize
  mov [ebp-04Ch],012345678h             //ImageEntryPoint
  mov [ebp-054h],012345678h             //OrgImageSize
  ret
  dd DYN_LOADER_END_MAGIC
end;
procedure DynLoader_end; assembler; asm end;


procedure DynCoder(AAddr:Pointer;ASize:Cardinal;AKey:Pointer); assembler;
stdcall;
//розбиваємо ключ на декілький блоків у пам'яті
asm
 @Coder_begin:
  push edi
  push esi

 @Coder_main_loop:
  mov edi,[ebp+008h]            //AAddr
  mov ecx,[ebp+00Ch]            //ASize
  shr ecx,002h
 @Coder_pre_code:
  mov esi,[ebp+010h]            //AKey
 @Coder_code:
  mov eax,[esi]
  test eax,0FF000000h
  jz @Coder_pre_code
 @Coder_do_code:
  add eax,ecx
  xor eax,[edi]                 // розбиваємо це
  stosd                         // запам'ятовуємо це
  inc esi
  loop @Coder_code

 @Coder_end:
  pop esi
  pop edi
  leave
  ret 00Ch
end;

function
VirtAddrToPhysAddr(ANtHeaders:PImageNtHeaders;AVirtAddr:Pointer):Pointer;
//це повинно підтримувати tls, завантажуючи механізм повернення вказівника у
вихідні дані у старі PE данні о VA визначених AVirtAddr . або нулем, якщо ніякий
блок не містить ці данні
var
 LI:Integer;
 LPSection:PImageSectionHeader;
 LAddr:Cardinal;
begin
 Result:=nil;
 LAddr:=Cardinal(AVirtAddr)-ANtHeaders^.OptionalHeader.ImageBase;

LPSection:=Pointer(Cardinal(@ANtHeaders^.OptionalHeader)+ANtHeaders^.FileHeader.
SizeOfOptionalHeader);
 for LI:=0 to ANtHeaders^.FileHeader.NumberOfSections-1 do
 begin
  if (LPSection^.VirtualAddress<=Cardinal(LAddr)) and
(LPSection^.VirtualAddress+LPSection^.SizeOfRawData>Cardinal(LAddr)) and
(LPSection^.SizeOfRawData<>0) then
  begin
   Result:=Pointer(Cardinal(LPSection^.PointerToRawData)+LAddr-
LPSection^.VirtualAddress);
   Break;
```

```
  end;
  Inc(LPSection);
 end;
end;

function RVA2RAW(ANtHeader,AVirtImage:Pointer;ARVA:Cardinal):Pointer;
//Конвертуємо точку  RVA до RAW
var
 LPB:PByte;
begin
 Result:=nil;

LPB:=VirtAddrToPhysAddr(ANtHeader,Pointer(ARVA+PImageNtHeaders(ANtHeader)^.Optio
nalHeader.ImageBase));
 if LPB=nil then Exit;
 Inc(LPB,Cardinal(AVirtImage));
 Result:=LPB;
end;

function GetTlsCallbacksLen(ACallbacks:Pointer):Cardinal;
//підраховуємо розмір масиву tls який повертається
var
 LPC:PCardinal;
begin
 Result:=4;
 LPC:=ACallbacks;
 while LPC^<>0 do
 begin
  Inc(Result,4);
  Inc(LPC);
 end;
end;

function RoundSize(ASize,AAlignment:Cardinal):Cardinal;
// округляємо у більшу сторону
begin
 Result:=(ASize+AAlignment-1) div AAlignment*AAlignment;
end;

procedure GenerateRandomBuffer(ABuf:PByte;ASize:Cardinal);
//генеруємо буфер псевдовипадкових значень від 1 до 255
var
 LI:Integer;
begin
 for LI:=0 to ASize-1 do
 begin
  ABuf^:=Random($FE)+1;
  Inc(ABuf);
 end;
end;

procedure GenerateKey(AKey:PByte;ASize:Word);
//// генеруємо ключ для кодування даних
 //ключ є псевдовипадковим буфером, який закінчується нулем0
begin
 GenerateRandomBuffer(AKey,ASize);
 PByte(Cardinal(AKey)+Cardinal(ASize)-1)^:=0;
end;

procedure ThrowTheDice(var ADice:Cardinal;ASides:Cardinal=6); overload;
// Закінчення нарізання ний блоки
begin
 ADice:=Random(ASides)+1;
end;

procedure ThrowTheDice(var ADice:Word;ASides:Word=6); overload;
// Закінчення нарізання ний блоки
begin
 ADice:=Random(ASides)+1;
```

```
end;

procedure ThrowTheDice(var ADice:Byte;ASides:Byte=6); overload;
// Закінчення нарізання ний блоки
begin
 ADice:=Random(ASides)+1;
end;

function RandomReg32All:Byte;
// Вибір одного з eax,ecx,edx,ebx,esp,ebp,esi,edi
begin
 Result:=Random(Reg32Count);
end;

function RandomReg16All:Byte;
// Вибір одного з ax,cx,dx,bx,sp,bp,si,di
begin
 Result:=Random(Reg16Count);
end;

function RandomReg8ABCD:Byte;
// Вибір одного з al,cl,dl,bl,ah,ch,dh,bh
begin
 Result:=Random(Reg8Count);
end;

function RandomReg32Esp:Byte;
// Вибір одного з eax,ecx,edx,ebx,-,ebp,esi,edi
begin
 Result:=Random(Reg32Count-1);
 if Result=REG_ESP then Result:=7;
end;

function RandomReg32EspEbp:Byte;
// Вибір одного з eax,ecx,edx,ebx,-,-,esi,edi
begin
 Result:=Random(Reg32Count-2);
 if Result=REG_ESP then Result:=6
 else if Result=REG_EBP then Result:=7;
end;

procedure PutRandomBuffer(var AMem:PByte;ASize:Cardinal);
begin
 GenerateRandomBuffer(AMem,ASize);
 Inc(AMem,ASize);
end;

function Bswap(var AMem:PByte;AReg:Byte):Byte;
begin
 Result:=2;
 AMem^:=$0F;                          //bswap
 Inc(AMem);
 AMem^:=$C8+AReg;                     //reg32
 Inc(AMem);
end;

function Stosd(var AMem:PByte):Byte;
begin
 Result:=1;
 AMem^:=$AB;                          //stosd
 Inc(AMem);
end;

function Movsd(var AMem:PByte):Byte;
begin
 Result:=1;
 AMem^:=$A5;                          //movsd
 Inc(AMem);
end;
```

```
function Ret(var AMem:PByte):Byte;
begin
 Result:=1;
 AMem^:=$C3;                              //повернення
 Inc(AMem);
end;

procedure Ret16(var AMem:PByte;AVal:Word);
begin
 AMem^:=$C2;                              //повернення
 Inc(AMem);
 PWord(AMem)^:=AVal;                      //поверненняval
 Inc(AMem,2);
end;

procedure RelJmpAddr32(var AMem:PByte;AAddr:Cardinal);
begin
 AMem^:=$E9;                              //jmp
 Inc(AMem);
 PCardinal(AMem)^:=AAddr;
 Inc(AMem,4);
end;

procedure RelJmpAddr8(var AMem:PByte;AAddr:Byte);
begin
 AMem^:=$EB;                              //jmp
 Inc(AMem);
 AMem^:=AAddr;                            //Addr8
 Inc(AMem);
end;


procedure RelJzAddr32(var AMem:PByte;AAddr:Cardinal);
begin
 AMem^:=$0F;                              // умовний перехід
 Inc(AMem);
 AMem^:=$84;                              // якщо дорівнює нулю
 Inc(AMem);
 PCardinal(AMem)^:=AAddr;
 Inc(AMem,4);
end;

procedure RelJnzAddr32(var AMem:PByte;AAddr:Cardinal);
begin
 AMem^:=$0F;                              // умовний перехід
 Inc(AMem);
 AMem^:=$85;                              // якщо не дорівнює нулю
 Inc(AMem);
 PCardinal(AMem)^:=AAddr;
 Inc(AMem,4);
end;

procedure RelJbAddr32(var AMem:PByte;AAddr:Cardinal);
begin
 AMem^:=$0F;                              // умовний перехід
 Inc(AMem);
 AMem^:=$82;                              // якщо нижче
 Inc(AMem);
 PCardinal(AMem)^:=AAddr;
 Inc(AMem,4);
end;

procedure RelJzAddr8(var AMem:PByte;AAddr:Byte);
begin
 AMem^:=$74;                              //jz
 Inc(AMem);
 AMem^:=AAddr;                            //addr8
 Inc(AMem);
```

```
end;

procedure RelJnzAddr8(var AMem:PByte;AAddr:Byte);
begin
 AMem^:=$75;                              //jnz
 Inc(AMem);
 AMem^:=AAddr;                            //addr8
 Inc(AMem);
end;

function JmpRegMemIdx8(var AMem:PByte;AReg,AIdx:Byte):Byte;
begin
 Result:=3;
 AMem^:=$FF;                              //jmp
 Inc(AMem);
 AMem^:=$60+AReg;                         //regmem
 InC(AMem);
 if AReg=REG_ESP then
 begin
  Inc(Result);
  AMem^:=$24;                             //esp
  Inc(AMem);
 end;
 AMem^:=AIdx;                             //idx8
 Inc(AMem);
end;

function PushRegMem(var AMem:PByte;AReg:Byte):Byte;
begin
 Result:=2;
 AMem^:=$FF;                              //push
 Inc(AMem);
 if AReg=REG_EBP then
 begin
  Inc(Result);
  AMem^:=$75;                             //ebp
  Inc(AMem);
  AMem^:=$00;                             //+0
 end else AMem^:=$30+AReg;                //regmem
 Inc(AMem);
 if AReg=REG_ESP then
 begin
  Inc(Result);
  AMem^:=$24;                             //esp
  Inc(AMem);
 end;
end;

procedure PushReg32(var AMem:PByte;AReg:Byte);
begin
 AMem^:=$50+AReg;                         //push reg
 Inc(AMem);
end;

function PushReg32Rand(var AMem:PByte):Byte;
begin
 Result:=RandomReg32Esp;
 PushReg32(AMem,Result);
end;

procedure PopReg32(var AMem:PByte;AReg:Byte);
begin
 AMem^:=$58+AReg;                         //pop reg
 Inc(AMem);
end;

function PopReg32Idx(var AMem:PByte;AReg:Byte;AIdx:Cardinal):Byte;
begin
 Result:=6;
```

```
 AMem^:=$8F;                            //pop
 Inc(AMem);
 AMem^:=$80+AReg;                       //reg32
 Inc(AMem);
 if AReg=REG_ESP then
 begin
  AMem^:=$24;                           //esp
  Inc(AMem);
  Inc(Result);
 end;
 PCardinal(AMem)^:=AIdx;                //+ idx
 InC(AMem,4);
end;

procedure RelCallAddr(var AMem:PByte;AAddr:Cardinal);
begin
 AMem^:=$E8;                            //call
 Inc(AMem);
 PCardinal(AMem)^:=AAddr;               //Addr
 Inc(AMem,4);
end;

procedure MovReg32Reg32(var AMem:PByte;AReg1,AReg2:Byte);
begin
 AMem^:=$89;                            //mov
 Inc(AMem);
 AMem^:=AReg2*8+AReg1+$C0;              //reg32,reg32
 Inc(AMem);
end;

procedure AddReg32Reg32(var AMem:PByte;AReg1,AReg2:Byte);
begin
 AMem^:=$01;                            //add
 Inc(AMem);
 AMem^:=AReg2*8+AReg1+$C0;              //reg32,reg32
 Inc(AMem);
end;

function AddReg32RegMem(var AMem:PByte;AReg1,AReg2:Byte):Byte;
begin
 Result:=2;
 AMem^:=$03;                            //add
 Inc(AMem);
 if AReg2=REG_EBP then
 begin
  Inc(Result);
  AMem^:=AReg1*8+$45;                   //reg32,ebp
  Inc(AMem);
  AMem^:=$00;                           //+0
 end else AMem^:=AReg1*8+AReg2;         //reg32,regmem
 Inc(AMem);
 if AReg2=REG_ESP then
 begin
  Inc(Result);
  AMem^:=$24;                           //esp
  Inc(AMem);
 end;
end;

function AddRegMemReg32(var AMem:PByte;AReg1,AReg2:Byte):Byte;
begin
 Result:=2;
 AMem^:=$01;                            //add
 Inc(AMem);
 if AReg1=REG_EBP then
 begin
  Inc(Result);
  AMem^:=AReg2*8+$45;                   //regmem,ebp
  Inc(AMem);
```

```
  AMem^:=$00;                          //+0
 end else AMem^:=AReg2*8+AReg1;        //regmem,reg
 Inc(AMem);
 if AReg1=REG_ESP then
 begin
  Inc(Result);
  AMem^:=$24;                          //esp
  Inc(AMem);
 end;
end;

procedure AddReg32Num8(var AMem:PByte;AReg,ANum:Byte);
begin
 AMem^:=$83;                          //add
 Inc(AMem);
 AMem^:=$C0+AReg;                     //reg32
 Inc(AMem);
 AMem^:=ANum;                         //num8
 Inc(AMem);
end;

procedure MovReg32Num32(var AMem:PByte;AReg:Byte;ANum:Cardinal);
begin
 AMem^:=$B8+AReg;                     //mov reg32
 Inc(AMem);
 PCardinal(AMem)^:=ANum;              //num32
 Inc(AMem,4);
end;

function MovReg32IdxNum32(var AMem:PByte;AReg:Byte;AIdx,ANum:Cardinal):Byte;
begin
 Result:=10;
 AMem^:=$C7;                          //mov
 Inc(AMem);
 AMem^:=$80+AReg;                     //reg32
 Inc(AMem);
 if AReg=REG_ESP then
 begin
  Inc(Result);
  AMem^:=$24;                          //esp
  Inc(AMem);
 end;
 PCardinal(AMem)^:=AIdx;              //+ idx
 Inc(AMem,4);
 PCardinal(AMem)^:=ANum;              //Num32
 Inc(AMem,4);
end;

procedure MovReg32Reg32IdxNum32(var AMem:PByte;AReg1,AReg2:Byte;ANum:Cardinal);
//обидва AReg не повинні бути REG_ESP або REG_EBP
begin
 if AReg1=REG_ESP then begin AReg1:=AReg2; AReg2:=REG_ESP; end;
 if AReg2=REG_EBP then begin AReg2:=AReg1; AReg1:=REG_EBP; end;
 AMem^:=$C7;                          //mov
 Inc(AMem);
 AMem^:=$04;
 Inc(AMem);
 AMem^:=AReg1*8+AReg2;
 Inc(AMem);
 PCardinal(AMem)^:=ANum;              //Num32
 Inc(AMem,4);
end;

function MovReg32RegMem(var AMem:PByte;AReg1,AReg2:Byte):Byte;
begin
 Result:=2;
 AMem^:=$8B;                          //mov
 Inc(AMem);
 if AReg2=REG_EBP then
```

```
 begin
  Inc(Result);
  AMem^:=AReg1*8+$45;                    //reg32,ebp
  Inc(AMem);
  AMem^:=$00;                            //+0
 end else AMem^:=AReg1*8+AReg2;          //reg32,regmem
 Inc(AMem);
 if AReg2=REG_ESP then
 begin
  Inc(Result);
  AMem^:=$24;                            //esp
  Inc(AMem);
 end;
end;

function MovRegMemReg32(var AMem:PByte;AReg1,AReg2:Byte):Byte;
begin
 Result:=2;
 AMem^:=$89;                            //mov
 Inc(AMem);
 if AReg1=REG_EBP then
 begin
  Inc(Result);
  AMem^:=AReg2*8+$45;                    //reg32,ebp
  Inc(AMem);
  AMem^:=$00;                            //+0
 end else AMem^:=AReg2*8+AReg1;          //reg32,regmem
 Inc(AMem);
 if AReg1=REG_ESP then
 begin
  Inc(Result);
  AMem^:=$24;                            //esp
  Inc(AMem);
 end;
end;

function MovReg32RegMemIdx8(var AMem:PByte;AReg1,AReg2,AIdx:Byte):Byte;
begin
 Result:=3;
 AMem^:=$8B;                            //mov
 Inc(AMem);
 AMem^:=AReg1*8+AReg2+$40;              //AReg1,AReg2
 Inc(AMem);
 if AReg2=REG_ESP then
 begin
  Inc(Result);
  AMem^:=$24;                            //esp
  Inc(AMem);
 end;
 AMem^:=AIdx;                           //AIdx
 Inc(AMem);
end;

procedure PushNum32(var AMem:PByte;ANum:Cardinal);
begin
 AMem^:=$68;                            //push
 Inc(AMem);
 PCardinal(AMem)^:=ANum;
 Inc(AMem,4);
end;

procedure JmpReg32(var AMem:PByte;AReg:Byte);
begin
 AMem^:=$FF;                            //jmp | call
 Inc(AMem);
 AMem^:=$E0+AReg;                       //reg32
 Inc(AMem);
end;
```

```
procedure CallReg32(var AMem:PByte;AReg:Byte);
begin
 AMem^:=$FF;                            //jmp | call
 Inc(AMem);
 AMem^:=$D0+AReg;                       //reg32
 Inc(AMem);
end;

procedure Cld(var AMem:PByte);
begin
 AMem^:=$FC;                            //cld
 Inc(AMem);
end;

procedure Std(var AMem:PByte);
begin
 AMem^:=$FD;                            //std
 Inc(AMem);
end;

procedure Nop(var AMem:PByte);
begin
 AMem^:=$90;                            //nop
 Inc(AMem);
end;

procedure Stc(var AMem:PByte);
begin
 AMem^:=$F9;                            //stc
 Inc(AMem);
end;

procedure Clc(var AMem:PByte);
begin
 AMem^:=$F8;                            //clc
 Inc(AMem);
end;

procedure Cmc(var AMem:PByte);
begin
 AMem^:=$F5;                            //cmc
 Inc(AMem);
end;

procedure XchgReg32Rand(var AMem:PByte);
begin
 AMem^:=$87;                            //xchg
 Inc(AMem);
 AMem^:=$C0+RandomReg32All*9;           //reg32
 Inc(AMem);
end;

function XchgReg32Reg32(var AMem:PByte;AReg1,AReg2:Byte):Byte;
begin
 if AReg2=REG_EAX then begin AReg2:=AReg1; AReg1:=REG_EAX end;
 if AReg1=REG_EAX then ThrowTheDice(Result,2)
 else Result:=2;
 if Result=2 then
 begin
  AMem^:=$87;                           //xchg
  Inc(AMem);
  AMem^:=$C0+AReg2*8+AReg1;             //reg32
 end else AMem^:=$90+AReg2;             //xchg eax,reg32
 Inc(AMem);
end;

procedure MovReg32Rand(var AMem:PByte);
begin
 AMem^:=$8B;                            //mov
```

```
 Inc(AMem);
 AMem^:=$C0+RandomReg32All*9;          //reg32
 Inc(AMem);
end;

procedure IncReg32(var AMem:PByte;AReg:Byte);
begin
 AMem^:=$40+AReg;                      //inc reg32
 Inc(AMem);
end;

procedure DecReg32(var AMem:PByte;AReg:Byte);
begin
 AMem^:=$48+AReg;                      //dec reg32
 Inc(AMem);
end;

function IncReg32Rand(var AMem:PByte):Byte;
begin
 Result:=RandomReg32All;
 IncReg32(AMem,Result);
end;

function DecReg32Rand(var AMem:PByte):Byte;
begin
 Result:=RandomReg32All;
 DecReg32(AMem,Result);
end;

function LeaReg32Reg32(var AMem:PByte;AReg1,AReg2:Byte):Byte;
begin
 Result:=2;
 AMem^:=$8D;                           //mov
 Inc(AMem);
 if AReg2=REG_EBP then
 begin
  Inc(Result);
  AMem^:=AReg1*8+$45;                  //reg32,ebp
  Inc(AMem);
  AMem^:=$00;                          //+0
 end else AMem^:=AReg1*8+AReg2;        //reg32,regmem
 Inc(AMem);
 if AReg2=REG_ESP then
 begin
  Inc(Result);
  AMem^:=$24;                          //esp
  Inc(AMem);
 end;
end;

procedure LeaReg32Rand(var AMem:PByte);
begin
 AMem^:=$8D;                           //lea
 Inc(AMem);
 AMem^:=$00+RandomReg32EspEbp*9;       //reg32
 Inc(AMem);
end;

procedure LeaReg32Addr32(var AMem:PByte;AReg,AAddr:Cardinal);
begin
 AMem^:=$8D;                           //lea
 Inc(AMem);
 AMem^:=$05+AReg*8;                    //reg32
 Inc(AMem);
 PCardinal(AMem)^:=AAddr;              //addr32
 Inc(AMem,4);
end;
```

```
procedure TestReg32Rand(var AMem:PByte);
begin
 AMem^:=$85;                            //test
 Inc(AMem);
 AMem^:=$C0+RandomReg32All*9;           //reg32
 Inc(AMem);
end;

procedure OrReg32Rand(var AMem:PByte);
begin
 AMem^:=$0B;                            //or
 Inc(AMem);
 AMem^:=$C0+RandomReg32All*9;           //reg32
 Inc(AMem);
end;

procedure AndReg32Rand(var AMem:PByte);
begin
 AMem^:=$23;                            //and
 Inc(AMem);
 AMem^:=$C0+RandomReg32All*9;           //reg32
 Inc(AMem);
end;

procedure TestReg8Rand(var AMem:PByte);
var
 LReg8:Byte;
begin
 LReg8:=RandomReg8ABCD;
 AMem^:=$84;                            //test
 Inc(AMem);
 AMem^:=$C0+LReg8*9;                    //reg8
 Inc(AMem);
end;

procedure OrReg8Rand(var AMem:PByte);
var
 LReg8:Byte;
begin
 LReg8:=RandomReg8ABCD;
 AMem^:=$0A;                            //or
 Inc(AMem);
 AMem^:=$C0+LReg8*9;                    //reg8
 Inc(AMem);
end;

procedure AndReg8Rand(var AMem:PByte);
var
 LReg8:Byte;
begin
 LReg8:=RandomReg8ABCD;
 AMem^:=$22;                            //and
 Inc(AMem);
 AMem^:=$C0+LReg8*9;                    //reg8
 Inc(AMem);
end;

procedure CmpRegRegNum8Rand(var AMem:PByte);
var
 LRnd:Byte;
begin
 LRnd:=Random(3);
 AMem^:=$3A+LRnd;                       //cmp
 Inc(AMem);
 if LRnd<2 then LRnd:=Random($40)+$C0
 else LRnd:=Random($100);
 AMem^:=LRnd;                           //reg16 | reg32 | num16
 Inc(AMem);
end;
```

```
function CmpReg32Reg32(var AMem:Pbyte;AReg1,AReg2:Byte):Byte;
begin
 Result:=2;
 AMem^:=$39;                              //cmp
 Inc(AMem);
 AMem^:=$C0+AReg1+AReg2*8;                //reg1,reg2
 Inc(AMem);
end;

procedure CmpReg32Num8(var AMem:PByte;AReg,ANum:Byte);
begin
 AMem^:=$83;                              //cmp
 Inc(AMem);
 AMem^:=$F8+AReg;                         //reg32
 Inc(AMem);
 AMem^:=ANum;                             //num8
 Inc(AMem);
end;

procedure CmpReg32RandNum8(var AMem:PByte;AReg:Byte);
begin
 CmpReg32Num8(AMem,AReg,Random($100));
end;

procedure CmpRandReg32RandNum8(var AMem:PByte);
begin
 CmpReg32RandNum8(AMem,RandomReg32All);
end;

procedure JmpNum8(var AMem:PByte;ANum:Byte);
var
 LRnd:Byte;
begin
 LRnd:=Random(16);
 if LRnd=16 then AMem^:=$EB               //jmp
 else AMem^:=$70+LRnd;                    //cond jmp
 Inc(AMem);
 AMem^:=ANum;                             //num8
 Inc(AMem);
end;

procedure SubReg32Reg32(var AMem:PByte;AReg1,AReg2:Byte);
begin
 AMem^:=$29;                              //sub
 Inc(AMem);
 AMem^:=AReg2*8+AReg1+$C0;                //reg32,reg32
 Inc(AMem);
end;


procedure SubReg32Num8(var AMem:PByte;AReg,ANum:Byte);
begin
 AMem^:=$83;                              //sub
 Inc(AMem);
 AMem^:=$E8+AReg;                         //reg32
 Inc(AMem);
 AMem^:=ANum;                             //num8
 Inc(AMem);
end;

function SubReg32Num8Rand(var AMem:PByte;ANum:Byte):Byte;
begin
 Result:=RandomReg32All;
 SubReg32Num8(AMem,Result,ANum);
end;

function AddReg32Num8Rand(var AMem:PByte;ANum:Byte):Byte;
begin
```

```
 Result:=RandomReg32All;
 AddReg32Num8(AMem,Result,ANum);
end;

procedure SubAlNum8(var AMem:PByte;ANum:Byte);
begin
 AMem^:=$2C;                              //sub al
 Inc(AMem);
 AMem^:=ANum;                             //num8
 Inc(AMem);
end;

procedure TestAlNum8(var AMem:PByte;ANum:Byte);
begin
 AMem^:=$A8;                              //test al
 Inc(AMem);
 AMem^:=ANum;                             //num8
 Inc(AMem);
end;

procedure TestAlNum8Rand(var AMem:PByte);
begin
 TestAlNum8(AMem,Random($100));
end;

procedure SubReg8Num8(var AMem:PByte;AReg,ANum:Byte);
begin
 AMem^:=$80;                              //sub
 Inc(AMem);
 AMem^:=$E8+AReg;                         //reg8
 Inc(AMem);
 AMem^:=ANum;                             //num8
 Inc(AMem);
end;

procedure SubReg8Num8Rand(var AMem:PByte;ANum:Byte);
var
 LReg8:Byte;
begin
 LReg8:=RandomReg8ABCD;
 SubReg8Num8(AMem,LReg8,ANum);
end;

procedure TestReg8Num8(var AMem:PByte;AReg,ANum:Byte);
begin
 AMem^:=$F6;                              //test
 Inc(AMem);
 AMem^:=$C0+AReg;                         //reg8
 Inc(AMem);
 AMem^:=ANum;                             //num8
 Inc(AMem);
end;

procedure TestReg8Num8Rand(var AMem:PByte);
begin
 TestReg8Num8(AMem,RandomReg8ABCD,Random($100));
end;

procedure AddReg8Num8(var AMem:PByte;AReg,ANum:Byte);
begin
 AMem^:=$80;                              //add
 Inc(AMem);
 AMem^:=$C0+AReg;                         //reg8
 Inc(AMem);
 AMem^:=ANum;                             //num8
 Inc(AMem);
end;

procedure AddReg8Num8Rand(var AMem:PByte;ANum:Byte);
```

```
var
 LReg8:Byte;
begin
 LReg8:=RandomReg8ABCD;
 AddReg8Num8(AMem,LReg8,ANum);
end;

procedure AddAlNum8(var AMem:PByte;ANum:Byte);
begin
 AMem^:=$04;                          //add al
 Inc(AMem);
 AMem^:=ANum;                         //num8
 Inc(AMem);
end;

procedure FNop(var AMem:PByte);
begin
 AMem^:=$D9;                          //fnop
 Inc(AMem);
 AMem^:=$D0;
 Inc(AMem);
end;

procedure OrReg16Rand(var AMem:PByte);
var
 LReg16:Byte;
begin
 LReg16:=RandomReg16All;
 AMem^:=$66;                          //or | test | and
 Inc(AMem);
 AMem^:=$0B;
 Inc(AMem);
 AMem^:=$C0+LReg16*9;                 //reg16
 Inc(AMem);
end;

procedure TestReg16Rand(var AMem:PByte);
var
 LReg16:Byte;
begin
 LReg16:=RandomReg16All;
 AMem^:=$66;                          //or | test | and
 Inc(AMem);
 AMem^:=$85;
 Inc(AMem);
 AMem^:=$C0+LReg16*9;                 //reg16
 Inc(AMem);
end;

procedure AndReg16Rand(var AMem:PByte);
var
 LReg16:Byte;
begin
 LReg16:=RandomReg16All;
 AMem^:=$66;                          //or | test | and
 Inc(AMem);
 AMem^:=$23;
 Inc(AMem);
 AMem^:=$C0+LReg16*9;                 //reg16
 Inc(AMem);
end;

procedure Cdq(var AMem:PByte);
begin
 AMem^:=$99;
 Inc(AMem);
end;

procedure ShlReg32Num8(var AMem:PByte;AReg,ANum:Byte);
```

```
begin
 AMem^:=$C1;                          //shl | shr | sal | sar
 Inc(AMem);
 AMem^:=$E0+AReg;                     //reg32
 Inc(AMem);
 AMem^:=ANum;                         //num8
 Inc(AMem);
end;

procedure ShlReg32RandNum8FullRand(var AMem:PByte);
begin
 ShlReg32Num8(AMem,RandomReg32All,Random(8)*$20);
end;

procedure ShrReg32Num8(var AMem:PByte;AReg,ANum:Byte);
begin
 AMem^:=$C1;                          //shl | shr | sal | sar
 Inc(AMem);
 AMem^:=$E8+AReg;                     //reg32
 Inc(AMem);
 AMem^:=ANum;                         //num8
 Inc(AMem);
end;

procedure ShrReg32RandNum8FullRand(var AMem:PByte);
begin
 ShrReg32Num8(AMem,RandomReg32All,Random(8)*$20);
end;

procedure SalReg32Num8(var AMem:PByte;AReg,ANum:Byte);
begin
 AMem^:=$C1;                          //shl | shr | sal | sar
 Inc(AMem);
 AMem^:=$F0+AReg;                     //reg32
 Inc(AMem);
 AMem^:=ANum;                         //num8
 Inc(AMem);
end;

procedure SalReg32RandNum8FullRand(var AMem:PByte);
begin
 SalReg32Num8(AMem,RandomReg32All,Random(8)*$20);
end;

procedure SarReg32Num8(var AMem:PByte;AReg,ANum:Byte);
begin
 AMem^:=$C1;                          //shl | shr | sal | sar
 Inc(AMem);
 AMem^:=$F8+AReg;                     //reg32
 Inc(AMem);
 AMem^:=ANum;                         //num8
 Inc(AMem);
end;

procedure SarReg32RandNum8FullRand(var AMem:PByte);
begin
 SarReg32Num8(AMem,RandomReg32All,Random(8)*$20);
end;

procedure RolReg8Num8(var AMem:PByte;AReg,ANum:Byte);
begin
 AMem^:=$C0;                          //rol | ror
 Inc(AMem);
 AMem^:=$C0+AReg;                     //reg8
 Inc(AMem);
 AMem^:=ANum;                         //num8
 Inc(AMem);
end;
```

```
procedure RolReg8RandNum8FullRand(var AMem:PByte);
begin
 RolReg8Num8(AMem,RandomReg8ABCD,Random($20)*8);
end;

procedure RorReg8Num8(var AMem:PByte;AReg,ANum:Byte);
begin
 AMem^:=$C0;                              //rol | ror
 Inc(AMem);
 AMem^:=$C8+AReg;                         //reg8
 Inc(AMem);
 AMem^:=ANum;                             //num8
 Inc(AMem);
end;

procedure RorReg8RandNum8FullRand(var AMem:PByte);
begin
 RorReg8Num8(AMem,RandomReg8ABCD,Random($20)*8);
end;

procedure RolReg32Num8(var AMem:PByte;AReg,ANum:Byte);
begin
 AMem^:=$C1;                              //rol | ror
 Inc(AMem);
 AMem^:=$C0+AReg;                         //reg32
 Inc(AMem);
 AMem^:=ANum;                             //num8
 Inc(AMem);
end;

procedure RolReg32RandNum8FullRand(var AMem:PByte);
begin
 RolReg32Num8(AMem,RandomReg32All,Random(8)*$20);
end;

procedure RorReg32Num8(var AMem:PByte;AReg,ANum:Byte);
begin
 AMem^:=$C1;                              //rol | ror
 Inc(AMem);
 AMem^:=$C8+AReg;                         //reg32
 Inc(AMem);
 AMem^:=ANum;                             //num8
 Inc(AMem);
end;

procedure RorReg32RandNum8FullRand(var AMem:PByte);
begin
 RorReg32Num8(AMem,RandomReg32All,Random(8)*$20);
end;

procedure TestAxNum16(var AMem:PByte;ANum:Word);
begin
 AMem^:=$66;                              //test ax
 Inc(AMem);
 AMem^:=$A9;
 Inc(AMem);
 PWord(AMem)^:=ANum;                      //num16
 Inc(AMem,2);
end;

procedure TestAxNum16Rand(var AMem:PByte);
begin
 TestAxNum16(AMem,Random($10000));
end;

procedure CmpAxNum16(var AMem:PByte;ANum:Word);
begin
 AMem^:=$66;                              //cmp ax
 Inc(AMem);
```

```
 AMem^:=$3D;
 Inc(AMem);
 PWord(AMem)^:=ANum;                    //num16
 Inc(AMem,2);
end;

procedure CmpAxNum16Rand(var AMem:PByte);
begin
 TestAxNum16(AMem,Random($10000));
end;

procedure PushNum8(var AMem:PByte;ANum:Byte);
begin
 AMem^:=$6A;                            //push
 Inc(AMem);
 AMem^:=ANum;                           //num8
 Inc(AMem);
end;

procedure PushNum8Rand(var AMem:PByte);
begin
 PushNum8(AMem,Random($100));
end;

function XorRand(var AMem:PByte):Word;
var
 LRnd:Byte;
 LRes:PWord;
begin
 LRes:=Pointer(AMem);
 LRnd:=Random(5);
 AMem^:=$30+LRnd;                       //xor
 Inc(AMem);
 if LRnd=4 then AMem^:=Random($100)     //num8
 else AMem^:=Random(7)*9+Random(8)+1+$C0;      //reg8 | reg32 but never the same
reg
 Inc(AMem);
 Result:=LRes^;
end;

procedure InvertXor(var AMem:PByte;AXor:Word);
begin
 PWord(AMem)^:=AXor;
 Inc(AMem,2);
end;

procedure DoubleXorRand(var AMem:PByte);
begin
 InvertXor(AMem,XorRand(AMem));
end;

function NotReg32(var AMem:PByte;AReg:Byte):Byte;
begin
 Result:=2;
 AMem^:=$F7;                            //not
 Inc(AMem);
 AMem^:=$D0+AReg;                       //reg32
 Inc(AMem);
end;

function NegReg32(var AMem:PByte;AReg:Byte):Byte;
begin
 Result:=2;
 AMem^:=$F7;                            //not
 Inc(AMem);
 AMem^:=$D8+AReg;                       //reg32
 Inc(AMem);
end;
```

```
function NotRand(var AMem:PByte):Word;
var
 LRes:PWord;
begin
 LRes:=Pointer(AMem);
 AMem^:=$F6+Random(1);                  //not
 Inc(AMem);
 AMem^:=$D0+Random(8);                  //reg8 | reg32
 Inc(AMem);
 Result:=LRes^;
end;

procedure InvertNot(var AMem:PByte;ANot:Word);
begin
 PWord(AMem)^:=ANot;
 Inc(AMem,2);
end;

procedure DoubleNotRand(var AMem:PByte);
begin
 InvertNot(AMem,NotRand(AMem));
end;

function NegRand(var AMem:PByte):Word;
var
 LRes:PWord;
begin
 LRes:=Pointer(AMem);
 AMem^:=$F6+Random(1);                  //neg
 Inc(AMem);
 AMem^:=$D8+Random(8);                  //reg8 | reg32
 Inc(AMem);
 Result:=LRes^;
end;

procedure InvertNeg(var AMem:PByte;ANeg:Word);
begin
 PWord(AMem)^:=ANeg;
 Inc(AMem,2);
end;

procedure DoubleNegRand(var AMem:PByte);
begin
 InvertNeg(AMem,NegRand(AMem));
end;

procedure AddReg16Num8(var AMem:PByte;AReg,ANum:Byte);
begin
 AMem^:=$66;                            //add | or | and | sub | xor | cmp
 Inc(AMem);
 AMem^:=$83;
 Inc(AMem);
 AMem^:=$C0+AReg;                       //reg16
 Inc(AMem);
 AMem^:=ANum;                           //num;
 Inc(AMem);
end;

procedure AddReg16Num8Rand(var AMem:PByte;ANum:Byte);
begin
 AddReg16Num8(AMem,RandomReg16All,ANum);
end;

procedure OrReg16Num8(var AMem:PByte;AReg,ANum:Byte);
begin
 AMem^:=$66;                            //add | or | and | sub | xor | cmp
 Inc(AMem);
 AMem^:=$83;
 Inc(AMem);
```

```
 AMem^:=$C8+AReg;                        //reg16
 Inc(AMem);
 AMem^:=ANum;                            //num
 Inc(AMem);
end;

procedure OrReg16Num8Rand(var AMem:PByte;ANum:Byte);
begin
 OrReg16Num8(AMem,RandomReg16All,ANum);
end;

procedure AndReg16Num8(var AMem:PByte;AReg,ANum:Byte);
begin
 AMem^:=$66;                             //add | or | and | sub | xor | cmp
 Inc(AMem);
 AMem^:=$83;
 Inc(AMem);
 AMem^:=$E0+AReg;                        //reg16
 Inc(AMem);
 AMem^:=ANum;                            //num
 Inc(AMem);
end;

procedure AndReg16Num8Rand(var AMem:PByte;ANum:Byte);
begin
 AndReg16Num8(AMem,RandomReg16All,ANum);
end;

procedure SubReg16Num8(var AMem:PByte;AReg,ANum:Byte);
begin
 AMem^:=$66;                             //add | or | and | sub | xor | cmp
 Inc(AMem);
 AMem^:=$83;
 Inc(AMem);
 AMem^:=$E8+AReg;                        //reg16
 Inc(AMem);
 AMem^:=ANum;                            //num
 Inc(AMem);
end;

procedure SubReg16Num8Rand(var AMem:PByte;ANum:Byte);
begin
 SubReg16Num8(AMem,RandomReg16All,ANum);
end;

procedure XorReg16Num8(var AMem:PByte;AReg,ANum:Byte);
begin
 AMem^:=$66;                             //add | or | and | sub | xor | cmp
 Inc(AMem);
 AMem^:=$83;
 Inc(AMem);
 AMem^:=$F0+AReg;                        //reg16
 Inc(AMem);
 AMem^:=ANum;                            //num
 Inc(AMem);
end;

procedure XorReg16Num8Rand(var AMem:PByte;ANum:Byte);
begin
 XorReg16Num8(AMem,RandomReg16All,ANum);
end;

procedure CmpReg16Num8(var AMem:PByte;AReg,ANum:Byte);
begin
 AMem^:=$66;                             //add | or | and | sub | xor | cmp
 Inc(AMem);
 AMem^:=$83;
 Inc(AMem);
 AMem^:=$F8+AReg;                        //reg16
```

```
 Inc(AMem);
 AMem^:=ANum;                          //num
 Inc(AMem);
end;

procedure CmpReg16Num8RandRand(var AMem:PByte);
begin
 CmpReg16Num8(AMem,RandomReg16All,Random($100));
end;

procedure RolReg16Num8(var AMem:PByte;AReg,ANum:Byte);
begin
 AMem^:=$66;                           //rol | ror
 Inc(AMem);
 AMem^:=$C1;
 Inc(AMem);
 AMem^:=$C0+AReg;                      //reg16
 Inc(AMem);
 AMem^:=ANum;                          //num8
 Inc(AMem);
end;

procedure RolReg16RandNum8FullRand(var AMem:PByte);
begin
 RolReg16Num8(AMem,RandomReg16All,Random($10)*$10);
end;

procedure RorReg16Num8(var AMem:PByte;AReg,ANum:Byte);
begin
 AMem^:=$66;                           //rol | ror
 Inc(AMem);
 AMem^:=$C1;
 Inc(AMem);
 AMem^:=$C1+AReg;                      //reg16
 Inc(AMem);
 AMem^:=ANum;                          //num8
 Inc(AMem);
end;

procedure RorReg16RandNum8FullRand(var AMem:PByte);
begin
 RorReg16Num8(AMem,RandomReg16All,Random($10)*$10);
end;

function XchgRand(var AMem:PByte):Word;
var
 LRes:PWord;
 LRnd:Byte;
begin
 LRes:=Pointer(AMem);
 LRnd:=Random(4);
 case LRnd of
  0,1:AMem^:=$66+LRnd;                 //xchg
  2,3:AMem^:=$86+LRnd-2;               //xchg
 end;
 Inc(AMem);
 case LRnd of
  0,1:AMem^:=$90+Random(8);            //reg16 | reg32
  2,3:AMem^:=$C0+Random($10);          //reg8 | reg32
 end;
 Inc(AMem);
 Result:=LRes^;
end;

procedure InvertXchg(var AMem:PByte;AXchg:Word);
begin
 PWord(AMem)^:=AXchg;
 Inc(AMem,2);
end;
```

```
procedure DoubleXchgRand(var AMem:PByte);
begin
 InvertXchg(AMem,XchgRand(AMem));
end;

procedure LoopNum8(var AMem:PByte;ANum:Byte);
begin
 AMem^:=$E2;                          //loop
 Inc(AMem);
 AMem^:=ANum;                         //ANum
 Inc(AMem);
end;

procedure JecxzNum8(var AMem:PByte;ANum:Byte);
begin
 AMem^:=$E3;                          //jecxz
 Inc(AMem);
 AMem^:=ANum;                         //ANum
 Inc(AMem);
end;

procedure MovzxEcxCl(var AMem:PByte);
begin
 AMem^:=$0F;                          //movzx
 Inc(AMem);
 AMem^:=$B6;
 Inc(AMem);
 AMem^:=$C9;                          //ecx:cx
 Inc(AMem);
end;

procedure MovReg32Reg32Rand(var AMem:PByte;AReg:Byte);
begin
 AMem^:=$8B;                          //mov
 Inc(AMem);
 AMem^:=$C0+8*AReg+RandomReg32All;    //reg32
 Inc(AMem);
end;

procedure CmpEaxNum32(var AMem:PByte;ANum:Cardinal);
begin
 AMem^:=$3D;                          //cmp eax
 Inc(AMem);
 PCardinal(AMem)^:=ANum;              //num32
 Inc(AMem,4);
end;

procedure CmpEaxNum32Rand(var AMem:PByte);
begin
 CmpEaxNum32(AMem,Random($FFFFFFFF));
end;

procedure TestEaxNum32(var AMem:PByte;ANum:Cardinal);
begin
 AMem^:=$A9;                          //test eax
 Inc(AMem);
 PCardinal(AMem)^:=ANum;              //num32
 Inc(AMem,4);
end;

procedure TestEaxNum32Rand(var AMem:PByte);
begin
 TestEaxNum32(AMem,Random($FFFFFFFF));
end;

procedure SubEaxNum32(var AMem:PByte;ANum:Cardinal);
begin
 AMem^:=$2D;                          //sub eax
```

```
 Inc(AMem);
 PCardinal(AMem)^:=ANum;                //num32
 Inc(AMem,4);
end;

procedure AddEaxNum32(var AMem:PByte;ANum:Cardinal);
begin
 AMem^:=$05;                            //add eax
 Inc(AMem);
 PCardinal(AMem)^:=ANum;                //num32
 Inc(AMem,4);
end;

procedure AndEaxNum32(var AMem:PByte;ANum:Cardinal);
begin
 AMem^:=$25;                            //and eax
 Inc(AMem);
 PCardinal(AMem)^:=ANum;                //num32
 Inc(AMem,4);
end;

procedure OrEaxNum32(var AMem:PByte;ANum:Cardinal);
begin
 AMem^:=$0D;                            //or eax
 Inc(AMem);
 PCardinal(AMem)^:=ANum;                //num32
 Inc(AMem,4);
end;

procedure XorEaxNum32(var AMem:PByte;ANum:Cardinal);
begin
 AMem^:=$35;                            //xor eax
 Inc(AMem);
 PCardinal(AMem)^:=ANum;                //num32
 Inc(AMem,4);
end;

procedure AddReg32Num32(var AMem:PByte;AReg:Byte;ANum:Cardinal);
begin
 AMem^:=$81;                            //add | or | and | sub | xor | cmp
 Inc(AMem);
 AMem^:=$C0+AReg;                       //reg32
 Inc(AMem);
 PCardinal(AMem)^:=ANum;                //num32
 Inc(AMem,4);
end;

procedure OrReg32Num32(var AMem:PByte;AReg:Byte;ANum:Cardinal);
begin
 AMem^:=$81;                            //add | or | and | sub | xor | cmp
 Inc(AMem);
 AMem^:=$C8+AReg;                       //reg32
 Inc(AMem);
 PCardinal(AMem)^:=ANum;                //num32
 Inc(AMem,4);
end;

procedure AndReg32Num32(var AMem:PByte;AReg:Byte;ANum:Cardinal);
begin
 AMem^:=$81;                            //add | or | and | sub | xor | cmp
 Inc(AMem);
 AMem^:=$E0+AReg;                       //reg32
 Inc(AMem);
 PCardinal(AMem)^:=ANum;                //num32
 Inc(AMem,4);
end;

procedure SubReg32Num32(var AMem:PByte;AReg:Byte;ANum:Cardinal);
begin
```

```
 AMem^:=$81;                            //add | or | and | sub | xor | cmp
 Inc(AMem);
 AMem^:=$E8+AReg;                       //reg32
 Inc(AMem);
 PCardinal(AMem)^:=ANum;                //num32
 Inc(AMem,4);
end;

procedure XorReg32Num32(var AMem:PByte;AReg:Byte;ANum:Cardinal);
begin
 AMem^:=$81;                            //add | or | and | sub | xor | cmp
 Inc(AMem);
 AMem^:=$F0+AReg;                       //reg32
 Inc(AMem);
 PCardinal(AMem)^:=ANum;                //num32
 Inc(AMem,4);
end;

procedure XorReg32Reg32(var AMem:PByte;AReg1,AReg2:Byte);
begin
 AMem^:=$31;                            //xor
 Inc(AMem);
 AMem^:=$C0+AReg2*8+AReg1;              //reg32,reg32
 Inc(AMem);
end;

function XorReg32RegMem(var AMem:PByte;AReg1,AReg2:Byte):Byte;
begin
 Result:=2;
 AMem^:=$33;                            //xor
 Inc(AMem);
 if AReg2=REG_EBP then
 begin
  Inc(Result);
  AMem^:=AReg1*8+$45;                   //reg32,ebp
  Inc(AMem);
  AMem^:=$00;                           //+0
 end else AMem^:=AReg1*8+AReg2;         //reg32,regmem
 Inc(AMem);
 if AReg2=REG_ESP then
 begin
  Inc(Result);
  AMem^:=$24;                           //esp
  Inc(AMem);
 end;
end;

function XorRegMemReg32(var AMem:PByte;AReg1,AReg2:Byte):Byte;
begin
 Result:=2;
 AMem^:=$31;                            //xor
 Inc(AMem);
 if AReg1=REG_EBP then
 begin
  Inc(Result);
  AMem^:=AReg2*8+$45;                   //reg32,ebp
  Inc(AMem);
  AMem^:=$00;                           //+0
 end else AMem^:=AReg2*8+AReg1;         //reg32,regmem
 Inc(AMem);
 if AReg1=REG_ESP then
 begin
  Inc(Result);
  AMem^:=$24;                           //esp
  Inc(AMem);
 end;
end;

procedure CmpReg32Num32(var AMem:PByte;AReg:Byte;ANum:Cardinal);
```

```
begin
 AMem^:=$81;                          //add | or | and | sub | xor | cmp
 Inc(AMem);
 AMem^:=$F8+AReg;                     //reg32
 Inc(AMem);
 PCardinal(AMem)^:=ANum;              //num32
 Inc(AMem,4);
end;

function TestReg32Num32(var AMem:PByte;AReg:Byte;ANum:Cardinal):Byte;
begin
 if AReg=REG_EAX then ThrowTheDice(Result,2)
 else Result:=2;
 Inc(Result,4);
 if Result=6 then
 begin
  AMem^:=$F7;                          //test
  Inc(AMem);
  AMem^:=$C0+AReg;                     //reg32
  Inc(AMem);
  PCardinal(AMem)^:=ANum;              //num32
  Inc(AMem,4);
 end else TestEaxNum32(AMem,ANum);
end;

procedure TestReg32Reg32(var AMem:PByte;AReg1,AReg2:Byte);
begin
 AMem^:=$85;                          //test
 Inc(AMem);
 AMem^:=AReg2*8+AReg1+$C0;            //reg32,reg32
 Inc(AMem);
end;

function TestRegMemNum32(var AMem:PByte;AReg:Byte;ANum:Cardinal):Byte;
begin
 Result:=6;
 AMem^:=$F7;                          //test
 Inc(AMem);
 if AReg=REG_EBP then
 begin
  Inc(Result);
  AMem^:=$45;                         //ebp
  Inc(AMem);
  AMem^:=$00;                         //+0
 end else AMem^:=AReg;                //reg32
 Inc(AMem);
 if AReg=REG_ESP then
 begin
  Inc(Result);
  AMem^:=$24;                         //esp
  Inc(AMem);
 end;
 PCardinal(AMem)^:=ANum;              //num32
 Inc(AMem,4);
end;


procedure AddReg32RandNum32(var AMem:PByte;ANum:Cardinal);
begin
 AddReg32Num32(AMem,RandomReg32All,ANum);
end;

procedure OrReg32RandNum32(var AMem:PByte;ANum:Cardinal);
begin
 OrReg32Num32(AMem,RandomReg32All,ANum);
end;

procedure AndReg32RandNum32(var AMem:PByte;ANum:Cardinal);
begin
```

```
 AndReg32Num32(AMem,RandomReg32All,ANum);
end;

procedure SubReg32RandNum32(var AMem:PByte;ANum:Cardinal);
begin
 SubReg32Num32(AMem,RandomReg32All,ANum);
end;

procedure XorReg32RandNum32(var AMem:PByte;ANum:Cardinal);
begin
 XorReg32Num32(AMem,RandomReg32All,ANum);
end;

procedure CmpReg32RandNum32Rand(var AMem:PByte);
begin
 CmpReg32Num32(AMem,RandomReg32All,Random($FFFFFFFF));
end;

procedure TestReg32RandNum32Rand6(var AMem:PByte);
var
 LLen:Byte;
begin
 LLen:=TestReg32Num32(AMem,RandomReg32All,Random($FFFFFFFF));
 if LLen=5 then
 begin
  AMem^:=$90;
  Inc(AMem);
 end;
end;

procedure MovReg32Num32Rand(var AMem:PByte;AReg:Byte);
begin
 MovReg32Num32(AMem,AReg,Random($FFFFFFFF));
end;

procedure MovReg16Num16(var AMem:PByte;AReg:Byte;ANum:Word);
begin
 AMem^:=$66;                          //mov
 Inc(AMem);
 AMem^:=$B8+AReg;                     //reg16
 Inc(AMem);
 PWord(AMem)^:=ANum;                  //num16
 Inc(AMem,2);
end;

procedure MovReg16Num16Rand(var AMem:PByte;AReg:Byte);
begin
 MovReg16Num16(AMem,AReg,Random($10000));
end;



procedure GenerateRubbishCode(AMem:Pointer;ASize,AVirtAddr:Cardinal); stdcall;
//генерує буфер інструкцій, нічого не роблять, та не потрібно забувати, що флаги
зазвичай змінюються тут й не використані nops

 procedure InsertRandomInstruction(var AMem:PByte;ALength:Byte;var
ARemaining:Cardinal);
 var
  LRegAny:Byte;
  LMaxDice,LXRem:Cardinal;
 begin
  case ALength of
   1:begin
    ThrowTheDice(LMaxDice,50);
{$IFDEF RUBBISH_NOPS}
    LMaxDice:=11;
{$ENDIF}
    case LMaxDice of
```

```
     001..010:Cld(AMem);
     011..020:Nop(AMem);
     021..030:Stc(AMem);
     031..040:Clc(AMem);
     041..050:Cmc(AMem);
    end;
   end;
   2:begin
    ThrowTheDice(LMaxDice,145);
    case LMaxDice of
     001..010:XchgReg32Rand(AMem);
     011..020:MovReg32Rand(AMem);
     021..030:begin LRegAny:=IncReg32Rand(AMem); DecReg32(AMem,LRegAny); end;
     031..040:begin LRegAny:=DecReg32Rand(AMem); IncReg32(AMem,LRegAny); end;
     041..050:begin LRegAny:=PushReg32Rand(AMem); PopReg32(AMem,LRegAny); end;
     051..060:LeaReg32Rand(AMem);
     061..070:TestReg32Rand(AMem);
     071..080:OrReg32Rand(AMem);
     081..090:AndReg32Rand(AMem);
     091..100:TestReg8Rand(AMem);
     101..110:OrReg8Rand(AMem);
     111..120:AndReg8Rand(AMem);
     121..130:CmpRegRegNum8Rand(AMem);
     131..132:begin Std(AMem); Cld(AMem); end;
     133..134:JmpNum8(AMem,0);
     135..138:SubAlNum8(AMem,0);
     139..140:TestAlNum8Rand(AMem);
     141..142:AddAlNum8(AMem,0);
     143..145:FNop(AMem);
    end;
   end;
   3:begin
    ThrowTheDice(LMaxDice,205);
    case LMaxDice of
     001..010:begin JmpNum8(AMem,1); InsertRandomInstruction(AMem,1,LXRem); end;
     011..020:SubReg32Num8Rand(AMem,0);
     021..030:AddReg32Num8Rand(AMem,0);
     031..040:begin LRegAny:=PushReg32Rand(AMem); IncReg32(AMem,LRegAny);
PopReg32(AMem,LRegAny); end;
     041..050:begin LRegAny:=PushReg32Rand(AMem); DecReg32(AMem,LRegAny);
PopReg32(AMem,LRegAny); end;
     051..060:CmpRandReg32RandNum8(AMem);
     061..070:TestReg8Num8Rand(AMem);
     071..080:SubReg8Num8Rand(AMem,0);
     081..090:AddReg8Num8Rand(AMem,0);
     091..100:AndReg16Rand(AMem);
     101..110:TestReg16Rand(AMem);
     111..120:OrReg16Rand(AMem);
     121..130:ShlReg32RandNum8FullRand(AMem);
     131..140:ShrReg32RandNum8FullRand(AMem);
     141..150:SalReg32RandNum8FullRand(AMem);
     151..160:SarReg32RandNum8FullRand(AMem);
     161..170:RolReg8RandNum8FullRand(AMem);
     171..180:RorReg8RandNum8FullRand(AMem);
     181..190:RolReg32RandNum8FullRand(AMem);
     191..200:RorReg32RandNum8FullRand(AMem);
     201..203:begin PushReg32(AMem,REG_EDX); Cdq(AMem); PopReg32(AMem,REG_EDX);
end;
     204..205:begin LRegAny:=PushReg32Rand(AMem);
InsertRandomInstruction(AMem,1,LXRem); PopReg32(AMem,LRegAny); end;
    end;
   end;
   4:begin
    ThrowTheDice(LMaxDice,170);
    case LMaxDice of
     001..020:begin JmpNum8(AMem,2); InsertRandomInstruction(AMem,2,LXRem); end;
     021..040:begin LRegAny:=PushReg32Rand(AMem);
InsertRandomInstruction(AMem,2,LXRem); PopReg32(AMem,LRegAny); end;
     041..050:TestAxNum16Rand(AMem);
```

```
     051..060:CmpAxNum16Rand(AMem);
     061..063:DoubleXorRand(AMem);
     064..066:DoubleNegRand(AMem);
     067..070:DoubleNotRand(AMem);
     071..080:AddReg16Num8Rand(AMem,0);
     081..090:OrReg16Num8Rand(AMem,0);
     091..100:AndReg16Num8Rand(AMem,$FF);
     101..110:SubReg16Num8Rand(AMem,0);
     111..120:XorReg16Num8Rand(AMem,0);
     121..130:CmpReg16Num8RandRand(AMem);
     131..140:RolReg16RandNum8FullRand(AMem);
     141..150:RorReg16RandNum8FullRand(AMem);
     151..155:DoubleXchgRand(AMem);
     156..160:begin LRegAny:=PushReg32Rand(AMem);
MovReg32Reg32Rand(AMem,LRegAny); PopReg32(AMem,LRegAny); end;
     161..170:begin PushReg32Rand(AMem); AddReg32Num8(AMem,REG_ESP,4); end;
    end;
   end;
   5:begin
    ThrowTheDice(LMaxDice,150);
    case LMaxDice of
     001..030:begin JmpNum8(AMem,3); InsertRandomInstruction(AMem,3,LXRem); end;
     031..060:begin LRegAny:=PushReg32Rand(AMem);
InsertRandomInstruction(AMem,3,LXRem); PopReg32(AMem,LRegAny); end;
     061..070:begin LRegAny:=PushReg32Rand(AMem); PushNum8Rand(AMem);
PopReg32(AMem,LRegAny); PopReg32(AMem,LRegAny); end;
     071..080:begin PushNum8Rand(AMem); AddReg32Num8(AMem,REG_ESP,4); end;
     081..090:AddEaxNum32(AMem,0);
     091..100:OrEaxNum32(AMem,0);
     101..110:AndEaxNum32(AMem,$FFFFFFFF);
     111..120:SubEaxNum32(AMem,0);
     121..130:XorEaxNum32(AMem,0);
     131..140:CmpEaxNum32Rand(AMem);
     141..150:TestEaxNum32Rand(AMem);
    end;
   end;
   6:begin
    ThrowTheDice(LMaxDice,161);
    case LMaxDice of
     001..040:begin JmpNum8(AMem,4); InsertRandomInstruction(AMem,4,LXRem); end;
     041..080:begin LRegAny:=PushReg32Rand(AMem);
InsertRandomInstruction(AMem,4,LXRem); PopReg32(AMem,LRegAny); end;
     081..090:AddReg32RandNum32(AMem,0);
     091..100:OrReg32RandNum32(AMem,0);
     101..110:AndReg32RandNum32(AMem,$FFFFFFFF);
     111..120:SubReg32RandNum32(AMem,0);
     121..130:XorReg32RandNum32(AMem,0);
     131..140:CmpReg32RandNum32Rand(AMem);
     141..150:TestReg32RandNum32Rand6(AMem);
     151..161:begin LRegAny:=PushReg32Rand(AMem);
MovReg16Num16Rand(AMem,LRegAny); PopReg32(AMem,LRegAny); end;
    end;
   end;
   7:begin
    ThrowTheDice(LMaxDice,110);
    case LMaxDice of
     001..050:begin JmpNum8(AMem,5); InsertRandomInstruction(AMem,5,LXRem); end;
     051..100:begin LRegAny:=PushReg32Rand(AMem);
InsertRandomInstruction(AMem,5,LXRem); PopReg32(AMem,LRegAny); end;
     101..110:begin LRegAny:=PushReg32Rand(AMem);
MovReg32Num32Rand(AMem,LRegAny); PopReg32(AMem,LRegAny); end;
    end;
   end;
   8:begin
    ThrowTheDice(LMaxDice,120);
    case LMaxDice of
     001..060:begin JmpNum8(AMem,6); InsertRandomInstruction(AMem,6,LXRem); end;
     061..120:begin LRegAny:=PushReg32Rand(AMem);
InsertRandomInstruction(AMem,6,LXRem); PopReg32(AMem,LRegAny); end;
```

```
    end;
   end;
   9..10:begin
    ThrowTheDice(LMaxDice,200);
    case LMaxDice of
     001..100:begin JmpNum8(AMem,ALength-2);
InsertRandomInstruction(AMem,ALength-2,LXRem); end;
     101..200:begin LRegAny:=PushReg32Rand(AMem);
InsertRandomInstruction(AMem,ALength-2,LXRem); PopReg32(AMem,LRegAny); end;
    end;
   end;
  end;
  if ALength<11 then Dec(ARemaining,ALength);
 end;

var
 LPB:PByte;
 LReg:Byte;
 LDice,LDecSize,LSize,LAddr:Cardinal;

begin
 LPB:=AMem;
 LSize:=ASize;

 while LSize>0 do
 begin
  ThrowTheDice(LDice,6);                //1-5 генерує одну маленьку інструкцію
                                        //6 генерує повно розмірні інструкції

  if LSize<32 then LDice:=1;            //для навеликого використання невеликих
інструкцій буферів
  if AVirtAddr=0 then LDice:=1;         //декілька додаткових інструкцій це
використовують

{$IFDEF RUBBISH_NOPS}
  LDice:=1;
{$ENDIF}
  if LDice<6 then                       //генерує повнорозмірні інструкції
  begin                                 //генерує малі інструкції
   ThrowTheDice(LDice,LSize*100);       //001..100 для однобайтних інструкцій
                                        //011..200 для двобайтних інструкцій

{$IFDEF RUBBISH_NOPS}
   LDice:=1;
{$ENDIF}
   if LSize=1 then LDice:=1;

   case LDice of
    001..002:InsertRandomInstruction(LPB,1,LSize);      //однобайтні інструкції
    101..104:InsertRandomInstruction(LPB,2,LSize);      //двобайтні інструкції
    201..208:InsertRandomInstruction(LPB,3,LSize);      //трибайтні інструкції
    301..316:InsertRandomInstruction(LPB,4,LSize);      //чотирьохбайтні
інструкції
    401..432:InsertRandomInstruction(LPB,5,LSize);      //п'ятибайтні інструкції
    501..564:InsertRandomInstruction(LPB,6,LSize);      //шостибайтні інструкції
    else InsertRandomInstruction(LPB,(LDice+99) div 100,LSize); //інструкції
більшої довжини
   end;
  end else
  begin
//   ThrowTheDice(LDice,100);
   ThrowTheDice(LDice,63);
//   if LDice<76 then LDecSize:=LSize
   if LDice<57 then LDecSize:=LSize
   else LDecSize:=0;
   case LDice of
    1..18:begin                         // використовує rel jump
     RelJmpAddr32(LPB,LSize-5);           //5 jump
     PutRandomBuffer(LPB,LSize-5);
```

```
    end;
    19..37:begin                          //використовує rel call
     LReg:=PushReg32Rand(LPB);
     ThrowTheDice(LDice);
     if LDice>3 then LAddr:=LSize-8       //1 push, 5 call, 1 pop, 1 pop
     else LAddr:=LSize-10;                //1 push, 5 call, 3 add, 1 pop

     RelCallAddr(LPB,LAddr);
     PutRandomBuffer(LPB,LAddr);
     if LDice>3 then PopReg32(LPB,LReg)
     else AddReg32Num8(LPB,REG_ESP,4);
     PopReg32(LPB,LReg);
    end;
(*
 цей код не може бути використаний для dll, так як нам потрібно переходи для
цього
    38..56:begin                          // використовує reg jmp
     LReg:=PushReg32Rand(LPB);
     ThrowTheDice(LDice);
     LAddr:=AVirtAddr+ASize-1;            //1 pop
                                          // використовує ASize cuz of not rel jmp
     if LDice>3 then
     begin
      MovReg32Num32(LPB,LReg,LAddr);
      LAddr:=LSize-9;                     //1 push, 5 mov, 2 jmp, 1 pop
     end else
     begin
      PushNum32(LPB,LAddr);
      PopReg32(LPB,LReg);
      LAddr:=LSize-10;                    //1 push, 5 push, 1 pop, 2 jmp, 1 pop
     end;
     JmpReg32(LPB,LReg);
     PutRandomBuffer(LPB,LAddr);
     PopReg32(LPB,LReg);
    end;

    57..75:begin                          // використовує reg call
     LReg:=PushReg32Rand(LPB);
     ThrowTheDice(LDice,8);               //1,2 - push,mov,call,pop,pop
                                          //3,4 - push,mov,call,add,pop
                                          //5,6 - push,push,pop,call,pop,pop
                                          //7,8 - push,push,pop,call,add,pop

     case LDice of
      1,2,5,6:LAddr:=AVirtAddr+ASize-2; //1 pop, 1 pop
      else LAddr:=AVirtAddr+ASize-4;    //1 pop, 3 add
     end;

     if LDice<5 then
     begin
      MovReg32Num32(LPB,LReg,LAddr);
      if LDice<3 then LAddr:=LSize-10   //1 push, 5 mov, 2 call, 1 pop, 1 pop
      else LAddr:=LSize-12;             //1 push, 5 mov, 2 call, 3 add, 1 pop
     end else
     begin
      PushNum32(LPB,LAddr);
      PopReg32(LPB,LReg);
      if LDice<7 then LAddr:=LSize-11   //1 push, 5 push, 1 pop, 2 call, 1 pop,
1 pop
      else LAddr:=LSize-13;             //1 push, 5 push, 1 pop, 2 call, 3 add,
1 pop
     end;
     CallReg32(LPB,LReg);
     PutRandomBuffer(LPB,LAddr);
     case LDice of
      1,2,5,6:PopReg32(LPB,LReg);
      else AddReg32Num8(LPB,REG_ESP,4);
     end;
     PopReg32(LPB,LReg);
```

```
    end;
    *)

//    76..94:begin                          // використовує loop + jeczx
     38..56:begin                          // використовує loop + jeczx
     if LSize-3<$7D then LAddr:=LSize-4
     else LAddr:=$7C;
     LAddr:=Random(LAddr)+2;
     LoopNum8(LPB,LAddr);
     JecxzNum8(LPB,LAddr-2);
     PutRandomBuffer(LPB,LAddr-2);
     IncReg32(LPB,REG_ECX);
     LDecSize:=LAddr+3;
    end;
    //95..100:begin                         // використовує back loop
    57..63:begin                          // використовує back loop
     if LSize-7<$7D then LAddr:=LSize-7
     else LAddr:=$75;
     LAddr:=Random(LAddr)+3;
     PushReg32(LPB,REG_ECX);
     MovzxEcxCl(LPB);                                    //не є чеканням, якщо
ecx = 0
     GenerateRubbishCode(LPB,LAddr-3,0);
     Inc(LPB,LAddr-3);
     LoopNum8(LPB,$FE-LAddr);
     PopReg32(LPB,REG_ECX);

     LDecSize:=LAddr+4;
    end;
   end;
   Dec(LSize,LDecSize);
  end;
 end;
end;

procedure
GenerateInitCode(ACodePtr,AKeyPtr,AData1Ptr,ASize1,AData2Ptr,ASize2,ADynLoadAddr
,
                            AMainPtr,AEntryPointAddr,AImpThunk:Cardinal);
//Це - полідешифратор. Завантажувач бачучи кінець цієї функції, не забуває
додавати фіксуючі вказівники для деяких інструкцій. що дозволяє додавати більше
варіантів для кожної інструкції

type
 TPolyContext=record
  DataSizeRegister:Byte;
  DataAddrRegister:Byte;
  EipRegister:Byte;
  KeyAddrRegister:Byte;
  KeyBytesRegister:Byte;
  FreeRegisters:array[0..1] of Byte;
 end;

 var
  LInitInstr:array[0..InitInstrCount-1] of TVarInstruction;
  LI:Integer;
  LVirtAddr,LRubbishSize,LDelta,LDelta2,LRemaining,LCodeStart,LEIPSub:Cardinal;
  LPB:PByte;
  PolyContext:TPolyContext;
{$IFNDEF STATIC_CONTEXT}
  LRegUsed:array[0..Reg32Count-1] of Boolean;
  LNotUsed:Integer;
  LReg:Byte;
{$ENDIF}

  function InstructionAddress(AInstruction:Cardinal):PByte;
  //повертає точку виклику інструкції
  begin
```

```
  Result:=Pointer(Cardinal(InitData)+LInitInstr[AInstruction].VirtualAddress-
LCodeStart);
 end;

 function CallAddress(AFromInstruction,AToInstruction:Cardinal):Cardinal;
 //повертає відносну дельту між двоиа інструкціями для виклику
 begin
  Result:=LInitInstr[AToInstruction].VirtualAddress-
(LInitInstr[AFromInstruction].VirtualAddress+5);
 end;

 function JcxAddress(AFromInstruction,AToInstruction:Cardinal):Cardinal;
 //повертає відносну дельту між двоиа інструкціями для умовного переходу
 begin
  Result:=LInitInstr[AToInstruction].VirtualAddress-
(LInitInstr[AFromInstruction].VirtualAddress+6);
 end;

 function InsVAddr(AInstr:Cardinal):Cardinal;
 begin
  Result:=LInitInstr[AInstr].VirtualAddress;
 end;

 function InsFix1(AInstr:Cardinal):Byte;
 begin
  Result:=LInitInstr[AInstr].Vars[LInitInstr[AInstr].Index].Fix1;
 end;

 function InsFix2(AInstr:Cardinal):Byte;
 begin
  Result:=LInitInstr[AInstr].Vars[LInitInstr[AInstr].Index].Fix2;
 end;

 function InsFix3(AInstr:Cardinal):Byte;
 begin
  Result:=LInitInstr[AInstr].Vars[LInitInstr[AInstr].Index].Fix3;
 end;

 function InsFix4(AInstr:Cardinal):Byte;
 begin
  Result:=LInitInstr[AInstr].Vars[LInitInstr[AInstr].Index].Fix4;
 end;

 procedure FixInstr(AInstr:Cardinal;AFix1:Cardinal;AFix2:Cardinal=Cardinal(-1));
 begin
  if InsFix1(AInstr)<>Byte(-1) then
  begin
   LPB:=InstructionAddress(AInstr);
   Inc(LPB,InsFix1(AInstr));
   PCardinal(LPB)^:=AFix1;
  end;
  if (InsFix2(AInstr)<>Byte(-1)) and (AFix2<>Cardinal(-1)) then
  begin
   LPB:=InstructionAddress(AInstr);
   Inc(LPB,InsFix2(AInstr));
   PCardinal(LPB)^:=AFix2;
  end;
 end;

 procedure GeneratePolyInstruction(AInstruction,ARegister:Byte);
 var
  LPB:PByte;
  LReg,LFreeReg,LFreeRegOther,LAnyReg:Byte;

   function CtxFreeReg:Byte;
   var
    LIdx:Byte;
   begin
    LIdx:=Random(10) mod 2;
```

```
    LFreeReg:=PolyContext.FreeRegisters[LIdx];
    LFreeRegOther:=PolyContext.FreeRegisters[(LIdx+1) mod 2];
    Result:=LFreeReg;
   end;

   function CtxAnyRegEsp:Byte;
   begin
    LAnyReg:=RandomReg32Esp;
    Result:=LAnyReg;
   end;

begin
 case AInstruction of
  PII_POLY_MOV_REG_LOADER_SIZE,PII_POLY_MOV_REG_LOADER_ADDR,
  PII_CODER_MOV_REG_KEY:
   with LInitInstr[AInstruction] do
   begin
    Count:=4;
    Vars[0].Len:=5;
    Vars[0].Fix1:=1;
    LPB:=@Vars[0].Code;
    MovReg32Num32(LPB,ARegister,$12345678);

    Vars[1].Len:=6;
    Vars[1].Fix1:=1;
    LPB:=@Vars[1].Code;
    PushNum32(LPB,$12345678);
    PopReg32(LPB,ARegister);

    Vars[2].Len:=5;
    Vars[2].Fix1:=1;
    LPB:=@Vars[2].Code;
    MovReg32Num32(LPB,CtxFreeReg,$12345678);
    Inc(Vars[2].Len,XchgReg32Reg32(LPB,LFreeReg,ARegister));

    Vars[3].Len:=6;
    Vars[3].Fix1:=2;
    LPB:=@Vars[3].Code[0];
    LeaReg32Addr32(LPB,ARegister,$12345678);
   end;

  PII_POLY_JMP_DYNLOADER:
   with LInitInstr[AInstruction] do
   begin
    Count:=3;
    Vars[0].Len:=5;
    Vars[0].Fix1:=1;
    Vars[0].Fix2:=0;
    LPB:=@Vars[0].Code;
    RelJmpAddr32(LPB,$12345678);

    Vars[1].Len:=8;
    Vars[1].Fix1:=4;
    Vars[1].Fix2:=3;
    LPB:=@Vars[1].Code;
    LReg:=RandomReg32Esp;
    XorReg32Reg32(LPB,LReg,LReg);
    RelJzAddr32(LPB,$12345678);

    Vars[2].Len:=7;
    Vars[2].Fix1:=3;
    Vars[2].Fix2:=2;
    LPB:=@Vars[2].Code;
    DecReg32(LPB,PolyContext.EipRegister);
    RelJnzAddr32(LPB,$12345678);
   end;

  PII_CODER_CALL_GET_EIP:
   with LInitInstr[AInstruction] do
```

```
 begin
  Count:=4;
  Vars[0].Len:=5;
  Vars[0].Fix1:=1;
  Vars[0].Fix2:=0;
  Vars[0].Fix3:=5;
  LPB:=@Vars[0].Code;
  RelCallAddr(LPB,$12345678);

  Vars[1].Len:=12;
  Vars[1].Fix1:=3;
  Vars[1].Fix2:=2;
  Vars[1].Fix3:=12;
  LPB:=@Vars[1].Code;
  RelJmpAddr8(LPB,5);
  RelJmpAddr32(LPB,$12345678);
  RelCallAddr(LPB,Cardinal(-10));

  Vars[2].Len:=5;
  Vars[2].Fix1:=Byte(-1);
  Vars[2].Fix2:=Byte(-1);
  Vars[2].Fix3:=5;
  LPB:=@Vars[2].Code;
  RelCallAddr(LPB,0);

  Vars[3].Len:=9;
  Vars[3].Fix1:=Byte(-1);
  Vars[3].Fix2:=Byte(-1);
  Vars[3].Fix3:=9;
  LPB:=@Vars[3].Code;
  RelJmpAddr8(LPB,2);
  RelJmpAddr8(LPB,5);
  RelCallAddr(LPB,Cardinal(-7));
 end;

PII_CODER_GET_EIP:
 with LInitInstr[AInstruction] do
 begin
  Count:=4;
  Vars[0].Len:=1;
  LPB:=@Vars[0].Code;
  PopReg32(LPB,ARegister);

  Vars[1].Len:=3;
  LPB:=@Vars[1].Code;
  Inc(Vars[1].Len,MovReg32RegMem(LPB,ARegister,REG_ESP));
  AddReg32Num8(LPB,REG_ESP,4);

  Vars[2].Len:=3;
  LPB:=@Vars[2].Code;
  AddReg32Num8(LPB,REG_ESP,4);
  Inc(Vars[2].Len,MovReg32RegMemIdx8(LPB,ARegister,REG_ESP,Byte(-4)));

  Vars[3].Len:=4;
  LPB:=@Vars[3].Code;
  Inc(Vars[3].Len,MovReg32RegMem(LPB,ARegister,REG_ESP));
  IncReg32(LPB,REG_ESP);
  IncReg32(LPB,REG_ESP);
  IncReg32(LPB,REG_ESP);
  IncReg32(LPB,REG_ESP);
 end;

PII_CODER_FIX_DST_PTR,PII_CODER_FIX_SRC_PTR:
 with LInitInstr[AInstruction] do
 begin
  Count:=4;
  Vars[0].Len:=2;
  LPB:=@Vars[0].Code;
  AddReg32Reg32(LPB,ARegister,PolyContext.EipRegister);
```

```
  Vars[1].Len:=6;
  LPB:=@Vars[1].Code;
  PushReg32(LPB,PolyContext.EipRegister);
  AddReg32Reg32(LPB,PolyContext.EipRegister,ARegister);
  MovReg32Reg32(LPB,ARegister,PolyContext.EipRegister);
  PopReg32(LPB,PolyContext.EipRegister);

  Vars[2].Len:=6;
  LPB:=@Vars[2].Code;
  PushReg32(LPB,PolyContext.EipRegister);
  AddReg32Reg32(LPB,PolyContext.EipRegister,ARegister);
  PushReg32(LPB,PolyContext.EipRegister);
  PopReg32(LPB,ARegister);
  PopReg32(LPB,PolyContext.EipRegister);

  Vars[3].Len:=2;
  LPB:=@Vars[3].Code;
  PushReg32(LPB,PolyContext.EipRegister);
  Inc(Vars[3].Len,AddReg32RegMem(LPB,ARegister,REG_ESP));
  PopReg32(LPB,PolyContext.EipRegister);
 end;

PII_CODER_LOAD_KEY_TO_REG:
 with LInitInstr[AInstruction] do
 begin
  Count:=4;
  LPB:=@Vars[0].Code;
  Vars[0].Len:=MovReg32RegMem(LPB,ARegister,PolyContext.KeyAddrRegister);

  Vars[1].Len:=1;
  LPB:=@Vars[1].Code;
  Inc(Vars[1].Len,PushRegMem(LPB,PolyContext.KeyAddrRegister));
  PopReg32(LPB,ARegister);

  LPB:=@Vars[2].Code;
  Vars[2].Len:=LeaReg32Reg32(LPB,ARegister,PolyContext.KeyAddrRegister);
  Inc(Vars[2].Len,MovReg32RegMem(LPB,ARegister,ARegister));

  Vars[3].Len:=2;
  LPB:=@Vars[3].Code;
  XorReg32Reg32(LPB,ARegister,ARegister);
  Inc(Vars[3].Len,AddReg32RegMem(LPB,ARegister,PolyContext.KeyAddrRegister));
 end;

PII_CODER_TEST_KEY_END:
 with LInitInstr[AInstruction] do
 begin
  Count:=4;
  LPB:=@Vars[0].Code;
  Vars[0].Len:=TestReg32Num32(LPB,ARegister,$FF000000);

  LPB:=@Vars[1].Code;
  Vars[1].Len:=TestRegMemNum32(LPB,PolyContext.KeyAddrRegister,$FF000000);

  LPB:=@Vars[2].Code;
  Vars[2].Len:=7;
  MovReg32Reg32(LPB,CtxFreeReg,ARegister);
  ShrReg32Num8(LPB,LFreeReg,$18);
  TestReg32Reg32(LPB,LFreeReg,LFreeReg);

  LPB:=@Vars[3].Code;
  Vars[3].Len:=11;
  PushReg32(LPB,ARegister);
  PopReg32(LPB,CtxFreeReg);
  AndReg32Num32(LPB,LFreeReg,$FF000000);
  CmpReg32Num8(LPB,LFreeReg,0);
 end;
```

```
PII_CODER_JZ_CODER_BEGIN:
 with LInitInstr[AInstruction] do
 begin
  Count:=2;
  LPB:=@Vars[0].Code;
  Vars[0].Len:=6;
  Vars[0].Fix1:=2;
  Vars[0].Fix2:=0;
  RelJzAddr32(LPB,$12345678);

  LPB:=@Vars[1].Code;
  Vars[1].Len:=7;
  Vars[1].Fix1:=3;
  Vars[1].Fix2:=1;
  RelJnzAddr8(LPB,5);
  RelJmpAddr32(LPB,$12345678);
 end;

PII_CODER_ADD_DATA_IDX:
 with LInitInstr[AInstruction] do
 begin
  Count:=4;
  LPB:=@Vars[0].Code;
  Vars[0].Len:=2;
  AddReg32Reg32(LPB,ARegister,PolyContext.DataSizeRegister);

  LPB:=@Vars[1].Code;
  Vars[1].Len:=2;
  PushReg32(LPB,PolyContext.DataSizeRegister);
  Inc(Vars[1].Len,AddReg32RegMem(LPB,ARegister,REG_ESP));
  PopReg32(LPB,PolyContext.DataSizeRegister);

  LPB:=@Vars[2].Code;
  Vars[2].Len:=6;
  PushReg32(LPB,CtxFreeReg);
  MovReg32Reg32(LPB,LFreeReg,PolyContext.DataSizeRegister);
  AddReg32Reg32(LPB,Aregister,LFreeReg);
  PopReg32(LPB,LFreeReg);

  LPB:=@Vars[3].Code;
  Vars[3].Len:=2;
  PushReg32(LPB,ARegister);
  Inc(Vars[3].Len,AddRegMemReg32(LPB,REG_ESP,PolyContext.DataSizeRegister));
  PopReg32(LPB,ARegister);
 end;

PII_CODER_XOR_DATA_REG:
 with LInitInstr[AInstruction] do
 begin
  Count:=4;
  LPB:=@Vars[0].Code;
  Vars[0].Len:=XorReg32RegMem(LPB,ARegister,PolyContext.DataAddrRegister);

  LPB:=@Vars[1].Code;
  Vars[1].Len:=3;
  PushReg32(LPB,CtxFreeReg);
  Inc(Vars[1].Len,PushRegMem(LPB,PolyContext.DataAddrRegister));
  Inc(Vars[1].Len,XorReg32RegMem(LPB,ARegister,REG_ESP));
  PopReg32(LPB,LFreeReg);
  PopReg32(LPB,LFreeReg);

  LPB:=@Vars[2].Code;
  Vars[2].Len:=4;
  PushReg32(LPB,CtxFreeReg);
  Inc(Vars[2].Len,MovReg32RegMem(LPB,LFreeReg,PolyContext.DataAddrRegister));
  XorReg32Reg32(LPB,ARegister,LFreeReg);
  PopReg32(LPB,LFreeReg);

  LPB:=@Vars[3].Code;
```

```
     Vars[3].Len:=4;
     PushReg32(LPB,CtxFreeReg);
     Inc(Vars[3].Len,MovReg32RegMem(LPB,LFreeReg,PolyContext.DataAddrRegister));
     PushReg32(LPB,LFreeReg);
     Inc(Vars[3].Len,XorRegMemReg32(LPB,REG_ESP,ARegister));
     PopReg32(LPB,ARegister);
     PopReg32(LPB,LFreeReg);
    end;

   PII_CODER_STORE_DATA:
    with LInitInstr[AInstruction] do
    begin
     Count:=4;
     LPB:=@Vars[0].Code;
     Vars[0].Len:=1;

     if (PolyContext.DataAddrRegister<>REG_EDI) or
(PolyContext.KeyBytesRegister<>REG_EAX) then
     begin
      if (PolyContext.DataAddrRegister<>REG_EDI) then Inc(Vars[0].Len,6);
      if (PolyContext.DataAddrRegister<>REG_EDI) then PushReg32(LPB,REG_EDI);

      if (PolyContext.DataAddrRegister<>REG_EAX) then Inc(Vars[0].Len,2);
      if (PolyContext.DataAddrRegister<>REG_EAX) then PushReg32(LPB,REG_EAX);

      if (PolyContext.DataAddrRegister<>REG_EDI) then
PushReg32(LPB,PolyContext.DataAddrRegister);
      PushReg32(LPB,PolyContext.KeyBytesRegister);
      PopReg32(LPB,REG_EAX);
      if (PolyContext.DataAddrRegister<>REG_EDI) then PopReg32(LPB,REG_EDI);
      Inc(Vars[0].Len,4);
     end;
     Stosd(LPB);
     if (PolyContext.DataAddrRegister<>REG_EDI) or
(PolyContext.KeyBytesRegister<>REG_EAX) then
     begin
      PushReg32(LPB,REG_EAX);
      if (PolyContext.DataAddrRegister<>REG_EDI) then PushReg32(LPB,REG_EDI);
      if (PolyContext.DataAddrRegister<>REG_EDI) then
PopReg32(LPB,PolyContext.DataAddrRegister);
      PopReg32(LPB,PolyContext.KeyBytesRegister);
      if (PolyContext.DataAddrRegister<>REG_EAX) then PopReg32(LPB,REG_EAX);
      if (PolyContext.DataAddrRegister<>REG_EDI) then PopReg32(LPB,REG_EDI);
     end;

     LPB:=@Vars[1].Code;
     Vars[1].Len:=4;

Inc(Vars[1].Len,MovRegMemReg32(LPB,PolyContext.DataAddrRegister,ARegister));
     IncReg32(LPB,PolyContext.DataAddrRegister);
     IncReg32(LPB,PolyContext.DataAddrRegister);
     IncReg32(LPB,PolyContext.DataAddrRegister);
     IncReg32(LPB,PolyContext.DataAddrRegister);

     LPB:=@Vars[2].Code;
     Vars[2].Len:=5;
     PushReg32(LPB,CtxFreeReg);
     Inc(Vars[2].Len,XchgReg32Reg32(LPB,REG_ESP,PolyContext.DataAddrRegister));
     PopReg32(LPB,LFreeReg);
     PushReg32(LPB,ARegister);
     PopReg32(LPB,LFreeReg);
     Inc(Vars[2].Len,XchgReg32Reg32(LPB,PolyContext.DataAddrRegister,REG_ESP));
     PopReg32(LPB,LFreeReg);

     LPB:=@Vars[3].Code;
     Vars[3].Len:=2;

     if ARegister=REG_EDI then
     begin
```

```
      MovReg32Reg32(LPB,CtxFreeReg,REG_EDI);
      Inc(Vars[3].Len,2);
     end;

     if PolyContext.DataAddrRegister<>REG_EDI then
     begin
      PushReg32(LPB,REG_EDI);
      MovReg32Reg32(LPB,REG_EDI,PolyContext.DataAddrRegister);
      Inc(Vars[3].Len,6);
     end;
     if ARegister=REG_EDI then PushReg32(LPB,LFreeReg)
     else PushReg32(LPB,ARegister);
     Inc(Vars[3].Len,XchgReg32Reg32(LPB,REG_ESI,REG_ESP));
     Movsd(LPB);
     Inc(Vars[3].Len,XchgReg32Reg32(LPB,REG_ESP,REG_ESI));
     if PolyContext.DataAddrRegister<>REG_EDI then
     begin
      MovReg32Reg32(LPB,PolyContext.DataAddrRegister,REG_EDI);
      PopReg32(LPB,REG_EDI);
     end;
    end;

   PII_CODER_INC_SRC_PTR:
    with LInitInstr[AInstruction] do
    begin
     Count:=4;
     LPB:=@Vars[0].Code;
     Vars[0].Len:=1;
     IncReg32(LPB,ARegister);

     LPB:=@Vars[1].Code;
     Vars[1].Len:=3;
     AddReg32Num8(LPB,ARegister,1);

     LPB:=@Vars[2].Code;
     Vars[2].Len:=3;
     SubReg32Num8(LPB,ARegister,Byte(-1));

     LPB:=@Vars[3].Code;
     Vars[3].Len:=7;
     PushReg32(LPB,CtxFreeReg);
     PushNum8(LPB,1);
     PopReg32(LPB,LFreeReg);
     AddReg32Reg32(LPB,ARegister,LFreeReg);
     PopReg32(LPB,LFreeReg);
    end;

   PII_CODER_LOOP_CODER_CODE:
    with LInitInstr[AInstruction] do
    begin
     Count:=1;
     LPB:=@Vars[0].Code;
     Vars[0].Len:=7;
     Vars[0].Fix1:=3;
     Vars[0].Fix2:=1;
     DecReg32(LPB,ARegister);
     RelJnzAddr32(LPB,$12345678);
    end;

  end;
 end;

begin
 ASize1:=ASize1 shr 2;
// ASize2:=ASize2 shr 2;

 ZeroMemory(@LInitInstr,SizeOf(LInitInstr));

 //генеруємо випадковий контекст
```

```
 with PolyContext do
 begin
{$IFNDEF STATIC_CONTEXT}
  DataSizeRegister:=REG_NON;
  DataAddrRegister:=REG_NON;
  EipRegister:=REG_NON;
  KeyAddrRegister:=REG_NON;
  KeyBytesRegister:=REG_NON;
  FreeRegisters[0]:=REG_NON;
  FreeRegisters[1]:=REG_NON;
{$ELSE}
//  DataSizeRegister:=REG_ESI;
//  DataAddrRegister:=REG_EBP;
//  EipRegister:=REG_ECX;
//  KeyAddrRegister:=REG_EAX;
//  KeyBytesRegister:=REG_EBX;
//  FreeRegisters[0]:=REG_EDI;
//  FreeRegisters[1]:=REG_EDX;
  DataSizeRegister:=REG_EAX;
  DataAddrRegister:=REG_EBX;
  EipRegister:=REG_ECX;
  KeyAddrRegister:=REG_EDX;
  KeyBytesRegister:=REG_ESI;
  FreeRegisters[0]:=REG_EDI;
  FreeRegisters[1]:=REG_EBP;
{$ENDIF}
 end;

{$IFNDEF STATIC_CONTEXT}
 for LI:=0 to Reg32Count-1 do LRegUsed[LI]:=False;
 LNotUsed:=Reg32Count-1;
 while LNotUsed>0 do
 begin
  LReg:=Random(Reg32Count);
  while LRegUsed[LReg] or (LReg=REG_ESP) do LReg:=(LReg+1) mod Reg32Count;
  LRegUsed[LReg]:=True;

  with PolyContext do
  case LNotUsed of
   1:DataSizeRegister:=LReg;
   2:DataAddrRegister:=LReg;
   3:EipRegister:=LReg;
   4:KeyAddrRegister:=LReg;
   5:KeyBytesRegister:=LReg;
   6:FreeRegisters[0]:=LReg;
   7:FreeRegisters[1]:=LReg;
  end;
  Dec(LNotUsed);
 end;
{$ENDIF}

// ці рядки добрі для відлагодження
// PolyContext.DataSizeRegister:=REG_ESI;
// PolyContext.DataAddrRegister:=REG_EBX;
// PolyContext.EipRegister:=REG_EDX;
// PolyContext.KeyAddrRegister:=REG_EBP;
// PolyContext.KeyBytesRegister:=REG_EAX;
// PolyContext.FreeRegisters[0]:=REG_ECX;
// PolyContext.FreeRegisters[1]:=REG_EDI;


GeneratePolyInstruction(PII_POLY_MOV_REG_LOADER_SIZE,PolyContext.DataSizeRegiste
r);

GeneratePolyInstruction(PII_POLY_MOV_REG_LOADER_ADDR,PolyContext.DataAddrRegiste
r);

 GeneratePolyInstruction(PII_CODER_CALL_GET_EIP,REG_NON);
 GeneratePolyInstruction(PII_CODER_GET_EIP,PolyContext.EipRegister);
```

```
 GeneratePolyInstruction(PII_CODER_FIX_DST_PTR,PolyContext.DataAddrRegister);
 GeneratePolyInstruction(PII_CODER_MOV_REG_KEY,PolyContext.KeyAddrRegister);
 GeneratePolyInstruction(PII_CODER_FIX_SRC_PTR,PolyContext.KeyAddrRegister);

GeneratePolyInstruction(PII_CODER_LOAD_KEY_TO_REG,PolyContext.KeyBytesRegister);
 GeneratePolyInstruction(PII_CODER_TEST_KEY_END,PolyContext.KeyBytesRegister);
 GeneratePolyInstruction(PII_CODER_JZ_CODER_BEGIN,REG_NON);
 GeneratePolyInstruction(PII_CODER_ADD_DATA_IDX,PolyContext.KeyBytesRegister);
 GeneratePolyInstruction(PII_CODER_XOR_DATA_REG,PolyContext.KeyBytesRegister);
 GeneratePolyInstruction(PII_CODER_STORE_DATA,PolyContext.KeyBytesRegister);
 GeneratePolyInstruction(PII_CODER_INC_SRC_PTR,PolyContext.KeyAddrRegister);

GeneratePolyInstruction(PII_CODER_LOOP_CODER_CODE,PolyContext.DataSizeRegister);
 GeneratePolyInstruction(PII_POLY_JMP_DYNLOADER,REG_NON);

 //
 //вписуємо деякий текст який нічого не означає, вибираємо інструкцію, й до неї
його дописуємо, після цього вибираємо наступну інструкцію. й до неї дописуємо
також деяку нічого не значущу інформацію, й так далі…
 //
 //але повинні ураховувати, що такі інструкції, які нічого не виконують
неможливо вписувати між PII_CODER_TEST_KEY_END  та PII_CODER_JZ_CODER_BEGIN
 //

 ZeroMemory(InitData,InitSize);
 LRemaining:=InitSize;

 LPB:=InitData;

LCodeStart:=NtHeaders.OptionalHeader.ImageBase+NtHeaders.OptionalHeader.AddressO
fEntryPoint;
 LVirtAddr:=LCodeStart;

 for LI:=0 to InitInstrCount-1 do
 with LInitInstr[LI] do
 begin
  LDelta:=InitInstrCount-LI;
  LDelta2:=LRemaining-LDelta*10;
  LRubbishSize:=Random(LDelta2 div LDelta);
  if (LI<>PII_CODER_JZ_CODER_BEGIN) and (LRubbishSize>0) then           //не
змінювати флаги після тестування
  begin
   GenerateRubbishCode(LPB,LRubbishSize,LVirtAddr);
   Inc(LPB,LRubbishSize);
   Inc(LVirtAddr,LRubbishSize);
   Dec(LRemaining,LRubbishSize);
  end;

  VirtualAddress:=LVirtAddr;
  Index:=Random(LInitInstr[LI].Count);
  with Vars[Index] do
  begin
   CopyMemory(LPB,@Code,Len);
   Inc(LPB,Len);
   Inc(LVirtAddr,Len);
   Dec(LRemaining,Len);
  end;
 end;

 LRubbishSize:=Random(LRemaining);
 GenerateRubbishCode(LPB,LRubbishSize,LVirtAddr);
 Dec(LRemaining,LRubbishSize);
 Inc(LPB,LRubbishSize);
 LRubbishSize:=LRemaining;
 GenerateRandomBuffer(LPB,LRubbishSize);


 //
 //тепер скоректуємо вказівники
```

```
 //

 //викликаємо та відновлюємо для отримання eip
 //але потрібно тільки базовий образ, так як потрібно вирахувати rva цього
виклику
 LEIPSub:=InsVAddr(PII_CODER_CALL_GET_EIP)-
ACodePtr+InsFix3(PII_CODER_CALL_GET_EIP);

 FixInstr(PII_POLY_MOV_REG_LOADER_SIZE,ASize1);
 FixInstr(PII_POLY_MOV_REG_LOADER_Addr,AData1Ptr-LEIPSub);

 FixInstr(PII_CODER_MOV_REG_KEY,AKeyPtr-LEIPSub);

FixInstr(PII_CODER_CALL_GET_EIP,CallAddress(PII_CODER_CALL_GET_EIP,PII_CODER_GET
_EIP)-InsFix2(PII_CODER_CALL_GET_EIP));

FixInstr(PII_CODER_JZ_CODER_BEGIN,JcxAddress(PII_CODER_JZ_CODER_BEGIN,PII_CODER_
KEY_START)-InsFix2(PII_CODER_JZ_CODER_BEGIN));

FixInstr(PII_CODER_LOOP_CODER_CODE,JcxAddress(PII_CODER_LOOP_CODER_CODE,PII_CODE
R_CODE)-InsFix2(PII_CODER_LOOP_CODER_CODE));

 FixInstr(PII_POLY_JMP_DYNLOADER,ADynLoadAddr-
(InsVAddr(PII_POLY_JMP_DYNLOADER)+5)-InsFix2(PII_POLY_JMP_DYNLOADER));


//
//повідомлення про кінець роботи
//
//

//                                                  // PII_BEGIN

//                                                  // PII_POLY_BEGIN
// mov ecx,0WWXXYYZZh                 //редагування розміру        //
PII_POLY_MOV_REG_LOADER_SIZE
// mov edi,0WWXXYYZZh                 //редагування адреси         //
PII_POLY_MOV_REG_LOADER_ADDR

//                                                  // PII_CODER_BEGIN
// виклик PII_CODER_GET_EIP                           //
PII_CODER_CALL_GET_EIP
// pop edx                                          // PII_CODER_GET_EIP
// add edi,edx                                      //
PII_CODER_FIX_DST_PTR
//                                                  // PII_CODER_KEY_START
// mov esi,0WWXXYYZZh                 //адреса ключа            //
PII_CODER_MOV_REG_KEY
// add esi,edx                                      //
PII_CODER_FIX_SRC_PTR
//                                                  // PII_CODER_CODE
// mov eax,[esi]                      //редагування байт ключа     //
PII_CODER_LOAD_KEY_TO_REG
// test eax,0FF000000h                //тестування кінця ключа    //
PII_CODER_TEST_KEY_END
// jz PII_CODER_KEY_START             //перезавантаження  ключа        //
PII_CODER_JZ_CODER_BEGIN
// add eax,ecx                        //додавання деякого непотрібного коду
// PII_CODER_ADD_DATA_IDX
// xor [edi],eax                      //декодування             //
PII_CODER_XOR_DATA_REG
// stosd                              //зберігання даних         //
PII_CODER_STORE_DATA
// inc esi                            //зміна ключа         //
PII_CODER_INC_SRC_PTR
// loop PII_CODER_CODE                                      //
PII_CODER_LOOP_CODER_CODE
//                                                  // PII_CODER_END
```

```
// jmp @DynLoader_begin                                    //
PII_POLY_JMP_DYNLOADER
//                                                         // PII_POLY_END

//                                                         // PII_END

end;

function ExtractFileName(APath:string):string;
//повертаємо ім'я файлу з повним шляхом
var
 LI,LJ:Integer;
begin
 if Length(APath)<>0 then
 begin
  LJ:=0;
  for LI:=Length(APath) downto 1 do
   if APath[LI]='\' then
   begin
    LJ:=LI;
    Break;
   end;
  Result:=Copy(APath,LJ+1,MaxInt);
 end else Result:='';
end;


procedure Usage;
var
 LStr:string;
begin
end;

procedure ErrorMsg(AErrorMsg:string);
begin
 frmMain.AddLog(AErrorMsg,2);
end;

function UpperCase(AStr:string):string;
//вибір для рядка
var
 LI:Integer;
begin
 SetLength(Result,Length(AStr));
 for LI:=1 to Length(AStr) do Result[LI]:=UpCase(AStr[LI]);
end;

{$R-}
function IntToHex(ACard:Cardinal;ADigits:Byte):string;
//перетворення числа у рядок hex
const
 HexArray:array[0..15] of
Char=('0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F');
var
 LHex:string;
 LInt,LHInt:Cardinal;
begin
 LHex:=StringOfChar('0',ADigits);
 LInt:=ADigits;
 LHInt:=16;
 while ACard>0 do
 begin
  LHex[LInt]:=HexArray[(ACard mod LHint) mod 16];
  ACard:=ACard div 16;
  LHInt:=LHInt*16;
  if LHInt=0 then LHInt:=$FFFFFFFF;
  Dec(LInt);
 end;
 Result:=LHex;
```

```
end;

function HexToInt(AHex:string):Cardinal;
//перетворення hex рядку у число
var
 LI,LO:Byte;
 LM:Cardinal;
begin
 LM:=1;
 Result:=0;
 AHex:=UpperCase(AHex);
 if (Length(AHex)>2) and (AHex[2]='X') then AHex:=Copy(AHex,3,MaxInt);

 for LI:=Length(AHex) downto 1 do
 begin
  if not ((AHex[LI] in ['0'..'9']) or (AHex[LI] in ['A'..'F'])) then
  begin
   Result:=0;
   Exit;
  end;
  if AHex[LI] in ['0'..'9'] then LO:=48 else LO:=55;
  LO:=Ord(AHex[LI])-LO;
  Result:=Result+LO*LM;
  LM:=LM shl 4;
 end;
end;
{$R+}


function CheckPEFile(AData:PByte):Boolean;
//повертає True якщо крапка AData валідного файлу образу
var
 LPNtHdr:PImageNtHeaders;
begin
 Result:=False;
 try
  if PImageDosHeader(AData)^.e_magic<>PWord(PChar('MZ'))^ then Exit;

  LPNtHdr:=Pointer(Cardinal(AData)+Cardinal(PImageDosHeader(AData)^._lfanew));

  if LPNtHdr^.Signature<>PCardinal(PChar('PE'))^ then Exit;
  if LPNtHdr^.FileHeader.Machine<>IMAGE_FILE_MACHINE_I386 then Exit;
  if LPNtHdr^.OptionalHeader.Magic<>IMAGE_NT_OPTIONAL_HDR_MAGIC then Exit;
  Result:=True;
 except
 end;
end;

function ProcessCmdLine:Boolean;
//командний рядок процесу, повертає True якщо аргументи відповідають дійсності
var
 LI:Integer;
 LPar,LUpArg:string;
begin
 Result:=False;

 Options:='';
 Quiet:=False;
 DynamicDLL:=False;
 SaveIcon:=True;
 SaveOverlay:=False;
 ReqImageBase:=0;
 InputFileName:='';
 OutputFileName:='';

 if (ParamCount<1) or (ParamCount>5) then Exit;
 LI:=1;
 while LI<=ParamCount do
 begin
```

```
  LPar:=ParamStr(LI);
  LUpArg:=UpperCase(LPar);
  if LUpArg[1]='-' then
  begin
   if Length(LUpArg)=1 then Break;
   case LUpArg[2] of
    'Q':Quiet:=True;
    'D':DynamicDLL:=True;
    'I':SaveIcon:=False;
    'A':SaveOverlay:=True;
    'B','O':begin
     if Length(LUpArg)<4 then Break;
     if LUpArg[3]<>':' then Break;
     if LUpArg[2]='B' then
     begin
      ReqImageBase:=HexToInt(Copy(LUpArg,4,MaxInt));
      if ReqImageBase=0 then Break;
     end else OutputFileName:=Copy(LPar,4,MaxInt);
    end;
    else Break;
   end;
  end else
  begin
   InputFileName:=LPar;
   Inc(LI);
   Break;
  end;
  Inc(LI);
 end;
 if Length(OutputFileName)=0 then OutputFileName:=InputFileName;
 Result:=(LI-1=ParamCount) and (Length(InputFileName)>0);
end;

function MyAlloc(ASize:Cardinal):Pointer;
//виділяє пам'ять для VirtualAlloc
begin
 Result:=VirtualAlloc(nil,ASize,MEM_COMMIT,PAGE_EXECUTE_READWRITE);
end;

function MyFree(APtr:Pointer):Boolean;
// визволяє пам'ять для VirtualAlloc
begin
 if APtr<>nil then Result:=VirtualFree(APtr,0,MEM_RELEASE)
 else Result:=False;
end;

procedure PrepareResourceSectionData;
//розпаковує та заповнює блок ресурсу
var
 LTypeTable:record
  Directory:TResourceDirectoryTable;
  IconsEntry,IconGroupEntry,XPEntry:TResourceDirectoryEntry;
 end;

 LXPManifest:record
  NameDir:TResourceTableDirectoryEntry;
  LangDir:TResourceTableDirectoryEntry;
  DataEntry:TResourceDataEntry;
 end;

 LIconGroup:record
  GroupNameDir:TResourceTableDirectoryEntry;
  GroupLangDir:TResourceTableDirectoryEntry;
  GroupData:TResourceDataEntry;

  IconCount:Integer;

  NameDir:TResourceDirectoryTable;
  NameEntries:array[0..31] of TResourceDirectoryEntry;
```

```
  LangDirs:array[0..31] of TResourceTableDirectoryEntry;

  DataEntries:array[0..31] of TResourceDataEntry;
 end;

 LResourceStrings:array[0..1023] of Char;
 LResourceData:array[0..65535] of Char;
 LResourceStringsPtr,LResourceDataPtr:Cardinal;

 LIconDirectory:PIconDirectory;

 LNameEntry:PResourceDirectoryEntry;
 LLangEntry:PResourceTableDirectoryEntry;
 LDataEntry:PResourceDataEntry;

LNameRVA,LSubEntryRVA,LSize,LResStringsRVA,LResDataRVA,LManifestSize,LResRawRVA:
Cardinal;
 LNameLen:Word;
 LPB,LPBManifest:PByte;
 LI:Integer;
 LImage:HMODULE;
 LIcoRes:HRSRC;

begin
 LImage:=LoadLibraryEx(PChar(InputFileName),0,LOAD_LIBRARY_AS_DATAFILE);
 ResourceSectionSize:=0;
 ZeroMemory(@LTypeTable,SizeOf(LTypeTable));
 if ResourceIconGroup<>nil then Inc(LTypeTable.Directory.NumberOfIDEntries,2);
 if ResourceXPManifest<>nil then Inc(LTypeTable.Directory.NumberOfIDEntries);
 LTypeTable.IconsEntry.NameID:=Cardinal(RT_ICON);
 LTypeTable.IconsEntry.SubdirDataRVA:=$80000000;
 LTypeTable.IconGroupEntry.NameID:=Cardinal(RT_GROUP_ICON);
 LTypeTable.IconGroupEntry.SubdirDataRVA:=$80000000;
 LTypeTable.XPEntry.NameID:=RT_XP_MANIFEST;
 LTypeTable.XPEntry.SubdirDataRVA:=$80000000;

 LResourceStringsPtr:=0;
 LResourceDataPtr:=0;

 if ResourceIconGroup<>nil then
 begin
  ZeroMemory(@LIconGroup,SizeOf(LIconGroup));
  LPB:=Pointer(ResourceIconGroup);
  Inc(LPB,SizeOf(TResourceDirectoryTable));
  LNameEntry:=Pointer(LPB);

  LNameRVA:=LNameEntry^.NameID and $7FFFFFFF;
  LSubEntryRVA:=LNameEntry^.SubdirDataRVA and $7FFFFFFF;

  if LNameEntry^.NameID and $80000000<>0 then
  begin
   LIconGroup.GroupNameDir.Table.NumberOfNameEntries:=1;
   LPB:=RVA2RAW(Ptr,MainData,HostResourceSectionVirtualAddress+LNameRVA);
   LNameLen:=2*PWord(LPB)^;
   LIconGroup.GroupNameDir.Directory.NameID:=LResourceStringsPtr+$80000000;
   CopyMemory(@LResourceStrings[LResourceStringsPtr],LPB,LNameLen+2);
   Inc(LResourceStringsPtr,LNameLen+2);
  end else
  begin
   LIconGroup.GroupNameDir.Directory.NameID:=LNameEntry^.NameID;
   LIconGroup.GroupNameDir.Table.NumberOfIDEntries:=1;
  end;
  LIconGroup.GroupNameDir.Directory.SubdirDataRVA:=0;


LLangEntry:=RVA2RAW(Ptr,MainData,HostResourceSectionVirtualAddress+LSubEntryRVA)
;
  LSubEntryRVA:=LLangEntry^.Directory.SubdirDataRVA and $7FFFFFFF;
```

```
  LIconGroup.GroupLangDir.Table.NumberOfIDEntries:=1;
  LIconGroup.GroupLangDir.Directory.NameID:=LLangEntry^.Directory.NameID;
  LIconGroup.GroupLangDir.Directory.SubdirDataRVA:=0;


LDataEntry:=RVA2RAW(Ptr,MainData,HostResourceSectionVirtualAddress+LSubEntryRVA)
;
  LPB:=RVA2RAW(Ptr,MainData,LDataEntry^.DataRVA);
  LIconGroup.GroupData.Size:=LDataEntry^.Size;
  LIconGroup.GroupData.DataRVA:=LResourceDataPtr;
  LIconGroup.GroupData.Codepage:=LDataEntry^.Codepage;

  CopyMemory(@LResourceData[LResourceDataPtr],LPB,LDataEntry^.Size);
  Inc(LResourceDataPtr,LDataEntry^.Size);

  LIconDirectory:=Pointer(LPB);
  LIconGroup.IconCount:=LIconDirectory^.Count;
  LIconGroup.NameDir.NumberOfIDEntries:=LIconGroup.IconCount;
  for LI:=0 to LIconDirectory^.Count-1 do
  begin
   LIconGroup.NameEntries[LI].NameID:=LIconDirectory^.Entries[LI].ID;
   LIconGroup.NameEntries[LI].SubdirDataRVA:=$80000000;
   LIconGroup.LangDirs[LI].Table.NumberOfIDEntries:=1;
   LIconGroup.LangDirs[LI].Directory.SubdirDataRVA:=$80000000;


LIcoRes:=FindResource(LImage,MakeIntResource(LIconDirectory^.Entries[LI].ID),RT_
ICON);
   LPB:=LockResource(LoadResource(LImage,LIcoRes));
   LSize:=SizeofResource(LImage,LIcoRes);
   LIconGroup.DataEntries[LI].Size:=LSize;
   LIconGroup.DataEntries[LI].DataRVA:=LResourceDataPtr;

   CopyMemory(@LResourceData[LResourceDataPtr],LPB,LSize);
   Inc(LResourceDataPtr,LSize);
  end;

  LSize:=6+LIconDirectory^.Count*SizeOf(TIconDirectoryEntry);
  CopyMemory(@LResourceData[LResourceDataPtr],LIconDirectory,LSize);
  Inc(LResourceDataPtr,LSize);
 end;

 if ResourceXPManifest<>nil then
 begin
  LPB:=Pointer(ResourceXPManifest);
  Inc(LPB,SizeOf(TResourceDirectoryTable));
  LNameEntry:=Pointer(LPB);
  LNameRVA:=LNameEntry^.NameID and $7FFFFFFF;
  LSubEntryRVA:=LNameEntry^.SubdirDataRVA and $7FFFFFFF;

  LXPManifest.NameDir.Table.NumberOfIDEntries:=1;
  LXPManifest.NameDir.Directory.NameID:=LNameRVA;
  LXPManifest.NameDir.Directory.SubdirDataRVA:=$80000000;


LLangEntry:=RVA2RAW(Ptr,MainData,HostResourceSectionVirtualAddress+LSubEntryRVA)
;
  LNameRVA:=LLangEntry^.Directory.NameID and $7FFFFFFF;
  LSubEntryRVA:=LLangEntry^.Directory.SubdirDataRVA and $7FFFFFFF;
  LXPManifest.LangDir.Table.NumberOfIDEntries:=1;
  LXPManifest.LangDir.Directory.NameID:=LNameRVA;
  LXPManifest.LangDir.Directory.SubdirDataRVA:=$80000000;


LDataEntry:=RVA2RAW(Ptr,MainData,HostResourceSectionVirtualAddress+LSubEntryRVA)
;
  LPB:=RVA2RAW(Ptr,MainData,LDataEntry^.DataRVA);
  LXPManifest.DataEntry.DataRVA:=LResourceDataPtr;
  LXPManifest.DataEntry.Size:=LDataEntry^.Size;
```

```
  LXPManifest.DataEntry.Codepage:=LDataEntry^.Codepage;

  CopyMemory(@LResourceData[LResourceDataPtr],LPB,LDataEntry^.Size);
  Inc(LResourceDataPtr,LDataEntry^.Size);
 end;

 LPB:=ResourceData;
 LManifestSize:=0;
 if ResourceXPManifest<>nil then
LManifestSize:=2*SizeOf(TResourceTableDirectoryEntry);

 LSubEntryRVA:=SizeOf(LTypeTable.Directory) or $80000000;
 if ResourceIconGroup<>nil then
Inc(LSubEntryRVA,SizeOf(LTypeTable.IconsEntry)+SizeOf(LTypeTable.IconGroupEntry)
);
 if ResourceXPManifest<>nil then Inc(LSubEntryRVA,SizeOf(LTypeTable.XPEntry));
 if ResourceIconGroup=nil then LTypeTable.XPEntry.SubdirDataRVA:=LSubEntryRVA
 else LTypeTable.IconsEntry.SubdirDataRVA:=LSubEntryRVA;
 LSize:=LSubEntryRVA and $7FFFFFFF;
 Inc(LPB,LSize);

 if ResourceIconGroup=nil then
 begin
  LResDataRVA:=LSubEntryRVA and $7FFFFFFF;
  Inc(LResDataRVA,SizeOf(LXPManifest.NameDir));
  Inc(LResDataRVA,SizeOf(LXPManifest.LangDir));
 end else
 begin
  LResStringsRVA:=LSubEntryRVA;
  Inc(LResStringsRVA,SizeOf(LIconGroup.NameDir));
  Inc(LResStringsRVA,LIconGroup.IconCount*SizeOf(TResourceDirectoryEntry));
  Inc(LResStringsRVA,LIconGroup.IconCount*SizeOf(TResourceTableDirectoryEntry));
  Inc(LResStringsRVA,SizeOf(LIconGroup.GroupNameDir));
  Inc(LResStringsRVA,SizeOf(LIconGroup.GroupLangDir));
  Inc(LResStringsRVA,LManifestSize);
  LResDataRVA:=LResStringsRVA and $7FFFFFFF+LResourceStringsPtr;

  //icons - name directory
  LSize:=SizeOf(LIconGroup.NameDir);
  Inc(LSubEntryRVA,LSize);
  CopyMemory(LPB,@LIconGroup.NameDir,LSize);
  Inc(LPB,LSize);

  //іконки – ім’я введення
  LSize:=LIconGroup.IconCount*SizeOf(TResourceDirectoryEntry);
  Inc(LSubEntryRVA,LSize);
  for LI:=0 to LIconGroup.IconCount-1 do
  begin
   LIconGroup.NameEntries[LI].SubdirDataRVA:=LSubEntryRVA;
   Inc(LSubEntryRVA,SizeOf(TResourceTableDirectoryEntry));
  end;
  CopyMemory(LPB,@LIconGroup.NameEntries,LSize);
  Inc(LPB,LSize);

  //іконки – ім’я директорії + введення
  LSize:=LIconGroup.IconCount*SizeOf(TResourceTableDirectoryEntry);
  for LI:=0 to LIconGroup.IconCount-1 do
  begin
   LIconGroup.LangDirs[LI].Directory.SubdirDataRVA:=LResDataRVA;
   Inc(LResDataRVA,SizeOf(TResourceDataEntry));
  end;
  CopyMemory(LPB,@LIconGroup.LangDirs,LSize);
  Inc(LPB,LSize);

  //іконка групи - ім’я директорії
  LTypeTable.IconGroupEntry.SubdirDataRVA:=LSubEntryRVA;
  LSize:=SizeOf(LIconGroup.GroupNameDir.Table);
  Inc(LSubEntryRVA,LSize);
  CopyMemory(LPB,@LIconGroup.GroupNameDir.Table,LSize);
```

```
 Inc(LPB,LSize);

 //іконка групи - ім'я введення
 if LIconGroup.GroupNameDir.Directory.NameID and $80000000<>0 then
  LIconGroup.GroupNameDir.Directory.NameID:=LResStringsRVA or $80000000;

 LSize:=SizeOf(LIconGroup.GroupNameDir.Directory);
 Inc(LSubEntryRVA,LSize);
 LIconGroup.GroupNameDir.Directory.SubdirDataRVA:=LSubEntryRVA;
 CopyMemory(LPB,@LIconGroup.GroupNameDir.Directory,LSize);
 Inc(LPB,LSize);

 /іконка групи - мова директорії + введення
 LIconGroup.GroupLangDir.Directory.SubdirDataRVA:=LResDataRVA;
 LSize:=SizeOf(LIconGroup.GroupLangDir);
 Inc(LSubEntryRVA,LSize);
 Inc(LResDataRVA,SizeOf(TResourceDataEntry));
 CopyMemory(LPB,@LIconGroup.GroupLangDir,LSize);
 Inc(LPB,LSize);
end;

if ResourceXPManifest<>nil then
begin
 LTypeTable.XPEntry.SubdirDataRVA:=LSubEntryRVA;

 //визначення- ім'я директорії + введення
 LSize:=SizeOf(LXPManifest.NameDir);
 Inc(LSubEntryRVA,LSize);
 LXPManifest.NameDir.Directory.SubdirDataRVA:=LSubEntryRVA;
 CopyMemory(LPB,@LXPManifest.NameDir,LSize);
 Inc(LPB,LSize);

 //визначення- мова директорії + введення
 LSize:=SizeOf(LXPManifest.LangDir);
 LXPManifest.LangDir.Directory.SubdirDataRVA:=LResDataRVA;
 Inc(LResDataRVA,SizeOf(TResourceDataEntry));
 CopyMemory(LPB,@LXPManifest.LangDir,LSize);
 Inc(LPB,LSize);
end;

//рядки
CopyMemory(LPB,@LResourceStrings,LResourceStringsPtr);
Inc(LPB,LResourceStringsPtr);

LResRawRVA:=LResDataRVA and $7FFFFFFF;

if ResourceIconGroup<>nil then
begin
 //іконки - дані
 LSize:=SizeOf(TResourceDataEntry)*LIconGroup.IconCount;
 for LI:=0 to LIconGroup.IconCount-1 do

Inc(LIconGroup.DataEntries[LI].DataRVA,LResRawRVA+ResourceSection.VirtualAddress
);
 CopyMemory(LPB,@LIconGroup.DataEntries,LSize);
 Inc(LPB,LSize);

 /іконка групи - дані
 LSize:=SizeOf(LIconGroup.GroupData);
 Inc(LIconGroup.GroupData.DataRVA,LResRawRVA+ResourceSection.VirtualAddress);
 CopyMemory(LPB,@LIconGroup.GroupData,LSize);
 Inc(LPB,LSize);
end;

if ResourceXPManifest<>nil then
begin
 //визначення - дані
 LSize:=SizeOf(LXPManifest.DataEntry);
 Inc(LXPManifest.DataEntry.DataRVA,LResRawRVA+ResourceSection.VirtualAddress);
```

```
  CopyMemory(LPB,@LXPManifest.DataEntry,LSize);
  Inc(LPB,LSize);
 end;

 CopyMemory(LPB,@LResourceData,LResourceDataPtr);
 Inc(LPB,LResourceDataPtr);
 ResourceSectionSize:=Cardinal(LPB)-Cardinal(ResourceData);

 LPB:=ResourceData;
 CopyMemory(LPB,@LTypeTable,SizeOf(LTypeTable.Directory));
 Inc(LPB,SizeOf(LTypeTable.Directory));
 if ResourceIconGroup<>nil then
 begin
  CopyMemory(LPB,@LTypeTable.IconsEntry,SizeOf(LTypeTable.IconsEntry));
  Inc(LPB,SizeOf(LTypeTable.IconsEntry));
  CopyMemory(LPB,@LTypeTable.IconGroupEntry,SizeOf(LTypeTable.IconGroupEntry));
  Inc(LPB,SizeOf(LTypeTable.IconGroupEntry));
 end;
 if ResourceXPManifest<>nil then
  CopyMemory(LPB,@LTypeTable.XPEntry,SizeOf(LTypeTable.XPEntry));

 FreeLibrary(LImage);
end;

function GenerateEncoderDecoder(AHostSize:Cardinal;out
OEncoder,ODecoder:Pointer):Cardinal;
//генератор шифратора та дешифратора для головного файлу, повертає розмір
декодера
const
 CI_XOR         = 00;
 CI_ADD         = 01;
 CI_SUB         = 02;
 CI_ROR         = 03;
 CI_ROL         = 04;
 CI_NOT         = 05;
 CI_NEG         = 06;
 CI_BSWAP       = 07;
 CI_XOR_OFS     = 08;
 CI_ADD_OFS     = 09;
 CI_SUB_OFS     = 10;
 CI_XOR_SMH     = 11;
 CI_ADD_SMH     = 12;
 CI_SUB_SMH     = 13;
 CI_SMH_ADD     = 14;
 CI_SMH_SUB     = 15;
 CI_MAX         = 16;

type
 TCoderInstruction=packed record
  IType,ILen:Byte;
  IArg1,IArg2,IArg3:Cardinal;
 end;
 TCoderContext=record
  DataSizeRegister:Byte;
  DataAddrRegister:Byte;
  DataRegister:Byte;
  OffsetRegister:Byte;
  SmashRegister:Byte;
  FreeRegister:Byte;
 end;

 TCoder=array[0..255] of TCoderInstruction;

var
 LCoderContext:TCoderContext;
 LEncoder,LDecoder:TCoder;
 LEncoderData,LDecoderData:array[0..511] of Char;
 LInstrCount,LReg:Byte;
 LI,LNotUsed:Integer;
```

```
 LRegUsed:array[0..Reg32Count-1] of Boolean;
 LPB,LPB2:PByte;
 LEncSize,LDecSize,LSmashNum:Cardinal;

 procedure GenerateCoderInstruction(ACoder:TCoder;AInstr:Integer);
 begin
  case ACoder[LI].IType of
   CI_XOR:XorReg32Num32(LPB,LCoderContext.DataRegister,ACoder[LI].IArg1);
   CI_ADD:AddReg32Num32(LPB,LCoderContext.DataRegister,ACoder[LI].IArg1);
   CI_SUB:SubReg32Num32(LPB,LCoderContext.DataRegister,ACoder[LI].IArg1);
   CI_ROR:RorReg32Num8(LPB,LCoderContext.DataRegister,ACoder[LI].IArg1);
   CI_ROL:RolReg32Num8(LPB,LCoderContext.DataRegister,ACoder[LI].IArg1);
   CI_NOT:NotReg32(LPB,LCoderContext.DataRegister);
   CI_NEG:NegReg32(LPB,LCoderContext.DataRegister);
   CI_BSWAP:Bswap(LPB,LCoderContext.DataRegister);

CI_XOR_OFS:XorReg32Reg32(LPB,LCoderContext.DataRegister,LCoderContext.OffsetRegi
ster);

CI_ADD_OFS:AddReg32Reg32(LPB,LCoderContext.DataRegister,LCoderContext.OffsetRegi
ster);

CI_SUB_OFS:SubReg32Reg32(LPB,LCoderContext.DataRegister,LCoderContext.OffsetRegi
ster);

CI_XOR_SMH:XorReg32Reg32(LPB,LCoderContext.DataRegister,LCoderContext.SmashRegis
ter);

CI_ADD_SMH:AddReg32Reg32(LPB,LCoderContext.DataRegister,LCoderContext.SmashRegis
ter);

CI_SUB_SMH:SubReg32Reg32(LPB,LCoderContext.DataRegister,LCoderContext.SmashRegis
ter);
   CI_SMH_ADD:AddReg32Num32(LPB,LCoderContext.SmashRegister,ACoder[LI].IArg1);
   CI_SMH_SUB:SubReg32Num32(LPB,LCoderContext.SmashRegister,ACoder[LI].IArg1);
  end;
 end;

begin
 LInstrCount:=Random(32)+16;                    //число інструкцій у
кодуванні/декодуванні
 LSmashNum:=Cardinal(Random($FFFFFFFF));

 //спершу генеруємо контекст кодувальника
 for LI:=0 to Reg32Count-1 do LRegUsed[LI]:=False;
 LRegUsed[REG_ESP]:=True;
 LRegUsed[REG_EBP]:=True;
 LNotUsed:=Reg32Count-2;
 while LNotUsed>0 do
 begin
  LReg:=Random(Reg32Count);
  while LRegUsed[LReg] do LReg:=(LReg+1) mod Reg32Count;
  LRegUsed[LReg]:=True;

  with LCoderContext do
  case LNotUsed of
   1:DataSizeRegister:=LReg;
   2:DataAddrRegister:=LReg;
   3:DataRegister:=LReg;
   4:OffsetRegister:=LReg;
   5:SmashRegister:=LReg;
   6:FreeRegister:=LReg;
  end;
  Dec(LNotUsed);
 end;

 // генеруємо кодер/декодер
 for LI:=0 to LInstrCount-1 do
 begin
```

```
LEncoder[LI].IType:=Random(CI_MAX);
case LEncoder[LI].IType of
 CI_XOR:begin
  //DataRegister = DataRegister xor IArg1
  LEncoder[LI].IArg1:=Cardinal(Random($FFFFFFFF));
  LDecoder[LI].IType:=CI_XOR;
  LDecoder[LI].IArg1:=LEncoder[LI].IArg1;
 end;

 CI_ADD:begin
  //DataRegister = DataRegister + IArg1
  LEncoder[LI].IArg1:=Cardinal(Random($FFFFFFFF));
  //DataRegister = DataRegister - IArg1
  LDecoder[LI].IType:=CI_SUB;
  LDecoder[LI].IArg1:=LEncoder[LI].IArg1;
 end;

 CI_SUB:begin
  //DataRegister = DataRegister - IArg1
  LEncoder[LI].IArg1:=Cardinal(Random($FFFFFFFF));
  //DataRegister = DataRegister + IArg1
  LDecoder[LI].IType:=CI_ADD;
  LDecoder[LI].IArg1:=LEncoder[LI].IArg1;
 end;

 CI_ROR:begin
  //DataRegister = DataRegister ror IArg1
  LEncoder[LI].IArg1:=Random($100);
  //DataRegister = DataRegister rol IArg1
  LDecoder[LI].IType:=CI_ROL;
  LDecoder[LI].IArg1:=LEncoder[LI].IArg1;
 end;

 CI_ROL:begin
  //DataRegister = DataRegister rol IArg1
  LEncoder[LI].IArg1:=Random($100);
  //DataRegister = DataRegister ror IArg1
  LDecoder[LI].IType:=CI_ROR;
  LDecoder[LI].IArg1:=LEncoder[LI].IArg1;
 end;

 CI_NOT:begin
  //DataRegister = not DataRegister
  LDecoder[LI].IType:=CI_NOT;
 end;

 CI_NEG:begin
  //DataRegister = -DataRegister
  LDecoder[LI].IType:=CI_NEG;
 end;

 CI_BSWAP:begin
  //DataRegister = swaped DataRegister
  LDecoder[LI].IType:=CI_BSWAP;
 end;

 CI_XOR_OFS:begin
  //DataRegister = DataRegister xor OffsetRegister
  LDecoder[LI].IType:=CI_XOR_OFS;
 end;

 CI_ADD_OFS:begin
  //DataRegister = DataRegister + OffsetRegister
  //DataRegister = DataRegister - OffsetRegister
  LDecoder[LI].IType:=CI_SUB_OFS;
 end;

 CI_SUB_OFS:begin
  //DataRegister = DataRegister + OffsetRegister
```

```
   //DataRegister = DataRegister - OffsetRegister
   LDecoder[LI].IType:=CI_ADD_OFS;
  end;

  CI_XOR_SMH:begin
   //DataRegister = DataRegister xor SmashRegister
   LDecoder[LI].IType:=CI_XOR_SMH;
  end;

  CI_ADD_SMH:begin
   //DataRegister = DataRegister + SmashRegister
   //DataRegister = DataRegister - SmashRegister
   LDecoder[LI].IType:=CI_SUB_SMH;
  end;

  CI_SUB_SMH:begin
   //DataRegister = DataRegister + SmashRegister
   //DataRegister = DataRegister - SmashRegister
   LDecoder[LI].IType:=CI_ADD_SMH;
  end;

  CI_SMH_ADD:begin
   //SmashRegister = SmashRegister + IArg1
   LEncoder[LI].IArg1:=Cardinal(Random($FFFFFFFF));
   //SmashRegister = SmashRegister - IArg1
   LDecoder[LI].IType:=CI_SMH_SUB;
   LDecoder[LI].IArg1:=LEncoder[LI].IArg1;
  end;

  CI_SMH_SUB:begin
   //SmashRegister = SmashRegister - IArg1
   LEncoder[LI].IArg1:=Cardinal(Random($FFFFFFFF));
   //SmashRegister = SmashRegister + IArg1
   LDecoder[LI].IType:=CI_SMH_ADD;
   LDecoder[LI].IArg1:=LEncoder[LI].IArg1;
  end;
 end;
end;


LPB:=@LEncoderData;
// заглушка
PushReg32(LPB,REG_EBX);
PushReg32(LPB,REG_ESI);
PushReg32(LPB,REG_EDI);
MovReg32RegMemIdx8(LPB,LCoderContext.DataAddrRegister,REG_ESP,$10);
MovReg32Num32(LPB,LCoderContext.DataSizeRegister,AHostSize);
XorReg32Reg32(LPB,LCoderContext.OffsetRegister,LCoderContext.OffsetRegister);
MovReg32Num32(LPB,LCoderContext.SmashRegister,LSmashNum);
//головний цикл
LPB2:=LPB;
//редагування даних
MovReg32RegMem(LPB,LCoderContext.DataRegister,LCoderContext.DataAddrRegister);
// генеруємо інструкції кодування
for LI:=0 to LInstrCount-1 do
 GenerateCoderInstruction(LEncoder,LI);
//запам'ятовуємо дані
MovRegMemReg32(LPB,LCoderContext.DataAddrRegister,LCoderContext.DataRegister);
//збільшуємо вказівник на дані
AddReg32Num8(LPB,LCoderContext.DataAddrRegister,4);
//збільшуємо зсув
AddReg32Num8(LPB,LCoderContext.OffsetRegister,4);
//кінець даних?
CmpReg32Reg32(LPB,LCoderContext.OffsetRegister,LCoderContext.DataSizeRegister);
RelJnzAddr32(LPB,-((Cardinal(LPB)+6)-Cardinal(LPB2)));
//повернення
MovReg32Reg32(LPB,REG_EAX,LCoderContext.SmashRegister);
PopReg32(LPB,REG_EDI);
PopReg32(LPB,REG_ESI);
```

```
 PopReg32(LPB,REG_EBX);
 Ret16(LPB,4);

 LEncSize:=Cardinal(LPB)-Cardinal(@LEncoderData);
 OEncoder:=MyAlloc(LEncSize);
 if OEncoder=nil then
 begin
  Result:=0;
  Exit;
 end;
 CopyMemory(OEncoder,@LEncoderData,LEncSize);

 EncoderProc:=OEncoder;
 LSmashNum:=EncoderProc(MainDataCyp);

 LPB:=@LDecoderData;
 // заглушка
 PopReg32(LPB,LCoderContext.DataAddrRegister);
 AddReg32Num32(LPB,LCoderContext.DataAddrRegister,AHostSize);
 MovReg32Num32(LPB,LCoderContext.DataSizeRegister,AHostSize);
 MovReg32Num32(LPB,LCoderContext.OffsetRegister,AHostSize);
 MovReg32Num32(LPB,LCoderContext.SmashRegister,LSmashNum);
 //головний цикл
 LPB2:=LPB;
 //dec offset
 SubReg32Num8(LPB,LCoderContext.OffsetRegister,4);
 //dec data ptr
 SubReg32Num8(LPB,LCoderContext.DataAddrRegister,4);
 //редагування даних
 MovReg32RegMem(LPB,LCoderContext.DataRegister,LCoderContext.DataAddrRegister);
 // генеруємо інструкції декодування
 for LI:=LInstrCount-1 downto 0 do
  GenerateCoderInstruction(LDecoder,LI);
 //запам'ятовуємо дані
 MovRegMemReg32(LPB,LCoderContext.DataAddrRegister,LCoderContext.DataRegister);
 //кінець даних?
 TestReg32Reg32(LPB,LCoderContext.OffsetRegister,LCoderContext.OffsetRegister);
 RelJnzAddr32(LPB,-((Cardinal(LPB)+6)-Cardinal(LPB2)));

 LDecSize:=Cardinal(LPB)-Cardinal(@LDecoderData);

 ODecoder:=MyAlloc(LDecSize);
 if ODecoder=nil then
 begin
  MyFree(OEncoder);
  OEncoder:=nil;
  Result:=0;
 end else
 begin
  CopyMemory(ODecoder,@LDecoderData,LDecSize);
  Result:=LDecSize;
 end;
end;

procedure FindAfterImageOverlays;
//переглядаємо оверлейні дані у MainData записані у кінці заповненого файлу
AfterImageOverlays, точка визначення його розміру - AfterImageOverlaysSize
var
 LI:Integer;
 LPSection:PImageSectionHeader;
 LMaxAddr,LDataSize:Cardinal;
 LHdr:PImageNtHeaders;

begin
 AfterImageOverlays:=nil;
 AfterImageOverlaysSize:=0;
 LMaxAddr:=0;
 LHdr:=Pointer(Cardinal(MainData)+Cardinal(PImageDosHeader(MainData)^._lfanew));
```

```
LPSection:=Pointer(Cardinal(@LHdr^.OptionalHeader)+LHdr^.FileHeader.SizeOfOption
alHeader);

 for LI:=0 to LHdr^.FileHeader.NumberOfSections-1 do
 begin
  LDataSize:=RoundSize(LPSection^.SizeOfRawData,RawDataAlignment);
  if LPSection^.PointerToRawData+LDataSize>LMaxAddr then
LMaxAddr:=LPSection^.PointerToRawData+LDataSize;
  Inc(LPSection);
 end;
 if (LMaxAddr>0) and (LMaxAddr<MainRealSize) then
 begin
  AfterImageOverlays:=Pointer(Cardinal(MainData)+LMaxAddr);
  AfterImageOverlaysSize:=MainRealSize-LMaxAddr;
 end;
end;


procedure Protect(InputFileName: string);
begin
 Randomize;
 SaveIcon:=frmMain.cbIcon.Checked;
 DynamicDLL:=False;
 SaveOverlay:=frmMain.cbOverlay.Checked;
 ReqImageBase:=HexToInt(frmMain.txtImageBase.Text);

 OutputFileName:=InputFileName;

MainFile:=CreateFile(PChar(InputFileName),GENERIC_READ,FILE_SHARE_READ,nil,OPEN_
EXISTING,0,0);
 if MainFile<>INVALID_HANDLE_VALUE then
 begin
  MainRealSize:=GetFileSize(MainFile,nil);

  MainRealSize4:=MainRealSize;
  frmMain.AddLog('Захищає...',0);
  if MainRealSize4 mod 4<>0 then Inc(MainRealSize4,4-MainRealSize4 mod 4);

  MainSize:=MainRealSize+Cardinal(Random(100)+10);
  MainData:=MyAlloc(MainSize);
  MainDataCyp:=MyAlloc(MainSize);
  if (MainData<>nil) and (MainDataCyp<>nil) then
  begin
   GenerateRandomBuffer(MainData,MainSize);
   ZeroMemory(MainData,MainRealSize4);
   if ReadFile(MainFile,MainData^,MainRealSize,NumBytes,nil) then
   begin
    CloseHandle(MainFile);
    MainFile:=INVALID_HANDLE_VALUE;
    CopyMemory(MainDataCyp,MainData,MainSize);
    if CheckPEFile(MainData) then
    begin

Ptr:=Pointer(Cardinal(MainData)+Cardinal(PImageDosHeader(MainData)^._lfanew));

     ImageType:=itExe;
     HostCharacteristics:=PImageNtHeaders(Ptr)^.FileHeader.Characteristics;
     if HostCharacteristics and IMAGE_FILE_DLL<>0 then ImageType:=itDLL;

HostExportSectionVirtualAddress:=PImageNtHeaders(Ptr)^.OptionalHeader.DataDirect
ory[IMAGE_DIRECTORY_ENTRY_EXPORT].VirtualAddress;

     HostImageBase:=PImageNtHeaders(Ptr)^.OptionalHeader.ImageBase;
     HostSubsystem:=PImageNtHeaders(Ptr)^.OptionalHeader.Subsystem;
     HostSizeOfImage:=PImageNtHeaders(Ptr)^.OptionalHeader.SizeOfImage;

HostImportSectionSize:=PImageNtHeaders(Ptr)^.OptionalHeader.DataDirectory[IMAGE_
DIRECTORY_ENTRY_IMPORT].Size;
```

```
HostImportSectionVirtualAddress:=PImageNtHeaders(Ptr)^.OptionalHeader.DataDirect
ory[IMAGE_DIRECTORY_ENTRY_IMPORT].VirtualAddress;


HostResourceSectionVirtualAddress:=PImageNtHeaders(Ptr)^.OptionalHeader.DataDire
ctory[IMAGE_DIRECTORY_ENTRY_RESOURCE].VirtualAddress;

     if (HostSubsystem=IMAGE_SUBSYSTEM_WINDOWS_GUI) or
(HostSubsystem=IMAGE_SUBSYSTEM_WINDOWS_CUI) then
     begin
      FindAfterImageOverlays;


TlsSectionSize:=PImageNtHeaders(Ptr)^.OptionalHeader.DataDirectory[IMAGE_DIRECTO
RY_ENTRY_TLS].Size;
      TlsSectionPresent:=TlsSectionSize<>0;

      ExportSectionPresent:=False;
      ExportSectionSize:=0;
      ExportData:=nil;
      if ImageType=itDLL then
      begin

ExportSectionSize:=PImageNtHeaders(Ptr)^.OptionalHeader.DataDirectory[IMAGE_DIRE
CTORY_ENTRY_EXPORT].Size;
       ExportSectionPresent:=ExportSectionSize<>0;
      end;

      ResourceSectionPresent:=False;
      HostResourceSectionSize:=0;
      ResourceData:=nil;
      if (ImageType=itExe) or (ImageType=itDLL) then
      begin

HostResourceSectionSize:=PImageNtHeaders(Ptr)^.OptionalHeader.DataDirectory[IMAG
E_DIRECTORY_ENTRY_RESOURCE].Size;
       ResourceSectionPresent:=HostResourceSectionSize<>0;
      end;

      OverlayPresent:=(AfterImageOverlays<>nil) and (AfterImageOverlaysSize>0);

       if TlsSectionPresent then
       begin
        frmMain.AddLog('.tls блок присутній',0);
        frmMain.AddLog('Початковий  .tls розмір блоку:
'+IntToStr(TlsSectionSize),0);
       end else frmMain.AddLog('.tls блок не присутній',1);

       if ExportSectionPresent then
       begin
        if not DynamicDLL then
        begin
         frmMain.AddLog('Експортуємий блок присутній',0);
         frmMain.AddLog('Початковий  експортуємий розмір блоку:
'+IntToStr(ExportSectionSize),0);
        end else frmMain.AddLog('Динамічна DLL – експортуємий блок не
використовується',1);
       end else frmMain.AddLog('Експортуємий блок не присутній',1);

       if ResourceSectionPresent then
       begin
        if SaveIcon then frmMain.AddLog('Ресурсний блок присутній',0)
        else frmMain.AddLog('Ресурсний блок присутній але не
використовується',1);
       end else frmMain.AddLog('Ресурсний блок не присутній',1);

       if OverlayPresent then
       begin
```

```
        if SaveOverlay then frmMain.AddLog('Оверлейні дані в наявності',0)
        else frmMain.AddLog('Оверлейні дані присутні але не
використовуються',1);
       end else frmMain.AddLog('Оверлейні дані не присутні ',1);

      if DynamicDLL then ExportSectionPresent:=False;
      if not SaveIcon then ResourceSectionPresent:=False;
      if not SaveOverlay then OverlayPresent:=False;

      if ResourceSectionPresent then
      begin
       ResourceRoot:=RVA2RAW(Ptr,MainData,HostResourceSectionVirtualAddress);

ResourceDirEntry:=RVA2RAW(Ptr,MainData,HostResourceSectionVirtualAddress+SizeOf(
TResourceDirectoryTable));
       ResourceIconGroup:=nil;
       ResourceXPManifest:=nil;
       for I:=0 to
ResourceRoot^.NumberOfIDEntries+ResourceRoot^.NumberOfNameEntries-1 do
       begin
        if (ResourceIconGroup=nil) and
(ResourceDirEntry^.NameID=Cardinal(RT_GROUP_ICON)) then

ResourceIconGroup:=RVA2RAW(Ptr,MainData,HostResourceSectionVirtualAddress+Resour
ceDirEntry^.SubdirDataRVA and $7FFFFFFF);

        if (ResourceXPManifest=nil) and
(ResourceDirEntry^.NameID=Cardinal(RT_XP_MANIFEST)) then

ResourceXPManifest:=RVA2RAW(Ptr,MainData,HostResourceSectionVirtualAddress+Resou
rceDirEntry^.SubdirDataRVA and $7FFFFFFF);

        if not ((ResourceIconGroup=nil) or (ResourceXPManifest=nil)) then Break;
        Inc(ResourceDirEntry);
       end;

       if not ((ResourceIconGroup=nil) and (ResourceXPManifest=nil)) then
       begin
        ResourceData:=MyAlloc(HostResourceSectionSize);
        if ResourceData=nil then
        begin
         ErrorMsg('Unable to allocate memore for resource data');
         ResourceSectionPresent:=False;
        end;
       end else ResourceSectionPresent:=False;
       if not ResourceSectionPresent then
         frmMain.AddLog('Ресурсний блок не має іконки або XP виявлення',1);
      end;


MainDataDecoderLen:=GenerateEncoderDecoder(MainRealSize4,MainDataEncoder,MainDat
aDecoder);
       if MainDataDecoderLen<>0 then
      begin
       LoaderRealSize:=Cardinal(@DynLoader_end)-Cardinal(@DynLoader);
       LoaderSize:=LoaderRealSize+MainDataDecoderLen+Cardinal(Random(100))+4;
       if LoaderSize mod 4>0 then Inc(LoaderSize,4-LoaderSize mod 4);
       frmMain.AddLog('Редактуємо розмір: '+IntToStr(LoaderSize),0);

       LoaderData:=MyAlloc(LoaderSize);
       if LoaderData<>nil then
       begin
        GenerateRandomBuffer(LoaderData,LoaderSize);
        CopyMemory(LoaderData,@DynLoader,LoaderRealSize);

        MainDataDecPtr:=LoaderData;
        while PCardinal(MainDataDecPtr)^<>DYN_LOADER_DEC_MAGIC do
Inc(MainDataDecPtr);
        DynLoaderDecoderOffset:=Cardinal(MainDataDecPtr)-Cardinal(LoaderData);
```

```
CopyMemory(Pointer(Cardinal(MainDataDecPtr)+MainDataDecoderLen),Pointer(Cardinal
(@DynLoader)+DynLoaderDecoderOffset+4),LoaderRealSize-DynLoaderDecoderOffset);
        CopyMemory(MainDataDecPtr,MainDataDecoder,MainDataDecoderLen);

        KeySize:=Random(200)+50;
        KeyPtr:=Random(200);
        LoaderPtr:=KeyPtr+KeySize;
        Trash2Size:=Random(256)+20;

        frmMain.AddLog(' Розмір ключа кодування: '+IntToStr(KeySize),0);
        Key:=MyAlloc(KeySize);
        if Key<>nil then
        begin
         GenerateKey(Key,KeySize);

         ZeroMemory(@DosHeader,SizeOf(DosHeader));
         ZeroMemory(@NtHeaders,SizeOf(NtHeaders));
         ZeroMemory(@DosStubEnd,SizeOf(DosStubEnd));
         DosHeader.e_magic:=PWord(PChar('MZ'))^;
         DosHeader.e_cblp:=$0050;
         DosHeader.e_cp:=$0002;
         DosHeader.e_cparhdr:=$0004;
         DosHeader.e_minalloc:=$000F;
         DosHeader.e_maxalloc:=$FFFF;
         DosHeader.e_sp:=$00B8;
         DosHeader.e_lfarlc:=$0040;
         DosHeader.e_ovno:=$001A;
         DosHeader._lfanew:=$0100;

         NtHeaders.Signature:=PCardinal(PChar('PE'))^;
         NtHeaders.FileHeader.Machine:=IMAGE_FILE_MACHINE_I386;
         NtHeaders.FileHeader.NumberOfSections:=2;
         if TlsSectionPresent then Inc(NtHeaders.FileHeader.NumberOfSections);
         if ExportSectionPresent then
Inc(NtHeaders.FileHeader.NumberOfSections);
         if ResourceSectionPresent then
Inc(NtHeaders.FileHeader.NumberOfSections);
         NtHeaders.FileHeader.TimeDateStamp:=Random($20000000)+$20000000;

NtHeaders.FileHeader.SizeOfOptionalHeader:=IMAGE_SIZEOF_NT_OPTIONAL_HEADER;
         NtHeaders.FileHeader.Characteristics:=IMAGE_FILE_EXECUTABLE_IMAGE or
IMAGE_FILE_LINE_NUMS_STRIPPED
                                              or IMAGE_FILE_LOCAL_SYMS_STRIPPED or
IMAGE_FILE_32BIT_MACHINE;
         case ImageType of

itExe:NtHeaders.FileHeader.Characteristics:=NtHeaders.FileHeader.Characteristics
or IMAGE_FILE_RELOCS_STRIPPED;

itDLL:NtHeaders.FileHeader.Characteristics:=NtHeaders.FileHeader.Characteristics
or IMAGE_FILE_DLL;
         end;
         RandomValue:=Random(10);
         if RandomValue>5 then
NtHeaders.FileHeader.Characteristics:=NtHeaders.FileHeader.Characteristics or
IMAGE_FILE_BYTES_REVERSED_LO or IMAGE_FILE_BYTES_REVERSED_HI;

         NtHeaders.OptionalHeader.Magic:=IMAGE_NT_OPTIONAL_HDR_MAGIC;
         NtHeaders.OptionalHeader.MajorLinkerVersion:=Random(9)+1;
         NtHeaders.OptionalHeader.MinorLinkerVersion:=Random(99)+1;
         NtHeaders.OptionalHeader.BaseOfCode:=$00001000;                //може
змінюватися
         if ReqImageBase<>0 then
NtHeaders.OptionalHeader.ImageBase:=RoundSize(ReqImageBase,$00010000)
         else if HostImageBase=$00400000 then
NtHeaders.OptionalHeader.ImageBase:=RoundSize(HostImageBase+HostSizeOfImage+$001
00000,$00010000)
         else NtHeaders.OptionalHeader.ImageBase:=$00400000;
```

```
        NtHeaders.OptionalHeader.SectionAlignment:=$00001000;          //1000h
= 4096
        NtHeaders.OptionalHeader.FileAlignment:=$00000200;             //може
змінюватися 200h = 512
        NtHeaders.OptionalHeader.MajorOperatingSystemVersion:=$0004;
        NtHeaders.OptionalHeader.MajorSubsystemVersion:=$0004;
        NtHeaders.OptionalHeader.SizeOfHeaders:=$00000400;             //може
змінюватися
        NtHeaders.OptionalHeader.Subsystem:=HostSubsystem;
        NtHeaders.OptionalHeader.SizeOfStackReserve:=$00100000;
        NtHeaders.OptionalHeader.SizeOfStackCommit:=$00010000;         //може
змінюватися
        NtHeaders.OptionalHeader.SizeOfHeapReserve:=$00100000;
        NtHeaders.OptionalHeader.SizeOfHeapCommit:=$00010000;
        NtHeaders.OptionalHeader.NumberOfRvaAndSizes:=$00000010;

        frmMain.AddLog('Побудова  .text блоку',0);

        ZeroMemory(@CodeSection,SizeOf(CodeSection));
        CopyMemory(@CodeSection.Name,PChar('.text'),5);       //може
змінюватися -> CODE
        CodeSection.VirtualAddress:=NtHeaders.OptionalHeader.BaseOfCode;
        CodeSection.PointerToRawData:=NtHeaders.OptionalHeader.SizeOfHeaders;

        InitSize:=Random($280)+$280;

CodeSection.SizeOfRawData:=RoundSize(LoaderPtr+LoaderSize+Trash2Size+InitSize+Ma
inSize,RawDataAlignment);

CodeSectionVirtualSize:=RoundSize(CodeSection.SizeOfRawData,NtHeaders.OptionalHe
ader.SectionAlignment);
        if CodeSectionVirtualSize<HostSizeOfImage then
CodeSectionVirtualSize:=RoundSize(HostSizeOfImage,NtHeaders.OptionalHeader.Secti
onAlignment);
        CodeSection.Misc.VirtualSize:=CodeSectionVirtualSize;

        NtHeaders.OptionalHeader.SizeOfCode:=CodeSection.SizeOfRawData;

        CodeSection.Characteristics:=IMAGE_SCN_CNT_CODE or
IMAGE_SCN_MEM_EXECUTE or IMAGE_SCN_MEM_WRITE or IMAGE_SCN_MEM_READ;

        frmMain.AddLog('.text блок, віртуальна адреса:
'+IntToHex(CodeSection.VirtualAddress,8),0);
        frmMain.AddLog('.text блок, віртуальний розмір:
'+IntToHex(CodeSection.Misc.VirtualSize,8),0);

        frmMain.AddLog('Будуємо .idata блок,0);


NtHeaders.OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_IMPORT].VirtualAddr
ess:=CodeSection.VirtualAddress+CodeSection.Misc.VirtualSize;       //може
змінюватися

        ZeroMemory(@ImportSection,SizeOf(ImportSection));

ImportSection.VirtualAddress:=NtHeaders.OptionalHeader.DataDirectory[IMAGE_DIREC
TORY_ENTRY_IMPORT].VirtualAddress;
        ImportSectionData:=MyAlloc(HostImportSectionSize+$70);
        ZeroMemory(ImportSectionData,HostImportSectionSize+$70);
        ImportSectionDLLCount:=1;

        if (HostImportSectionSize<>0) and (ImageType=itDLL) and not DynamicDLL
then
        begin

PB:=VirtAddrToPhysAddr(Ptr,Pointer(HostImportSectionVirtualAddress+PImageNtHeade
rs(Ptr)^.OptionalHeader.ImageBase));
         Inc(PB,Cardinal(MainData));
         PImportDesc:=Pointer(PB);
```

```
          while not ((PImportDesc^.Characteristics=0) and
(PImportDesc^.cTimeDateStamp=0)
            and (PImportDesc^.cForwarderChain=0) and (PImportDesc^.cName=0)
            and (PImportDesc^.cFirstThunk=nil)) do
          begin

PB2:=VirtAddrToPhysAddr(Ptr,Pointer(PImportDesc^.cName+PImageNtHeaders(Ptr)^.Opt
ionalHeader.ImageBase));
           Inc(PB2,Cardinal(MainData));
           if (UpperCase(PChar(PB2))<>UpperCase(Kernel32Name))
            and (UpperCase(PChar(PB2))<>UpperCase(NtdllName)) then
Inc(ImportSectionDLLCount);
           Inc(PImportDesc);
          end;
         end;


PB:=VirtAddrToPhysAddr(Ptr,Pointer(HostImportSectionVirtualAddress+PImageNtHeade
rs(Ptr)^.OptionalHeader.ImageBase));
         Inc(PB,Cardinal(MainData));
         PImportDesc:=Pointer(PB);
         PB2:=ImportSectionData;
         ZeroMemory(@ImportDesc,SizeOf(ImportDesc));

ImportDesc.Characteristics:=ImportSection.VirtualAddress+(ImportSectionDLLCount+
1)*SizeOf(ImportDesc);

ImportDesc.cName:=ImportSection.VirtualAddress+(ImportSectionDLLCount+1)*SizeOf(
ImportDesc)+(NumberOfImports+1+2*(ImportSectionDLLCount-
1))*SizeOf(TImageThunkData)*2;

ImportDesc.cFirstThunk:=Pointer(ImportDesc.Characteristics+(NumberOfImports+1+2*
(ImportSectionDLLCount-1))*SizeOf(TImageThunkData));
         InitcodeThunk:=Cardinal(ImportDesc.cFirstThunk);

         CopyMemory(PB2,@ImportDesc,SizeOf(ImportDesc));
         Inc(PB2,SizeOf(ImportDesc));
         PB3:=ImportSectionData;
         Inc(PB3,(ImportSectionDLLCount+1)*SizeOf(ImportDesc));
         PB4:=ImportSectionData;
         Inc(PB4,ImportDesc.cName-ImportSection.VirtualAddress);
         CopyMemory(PB4,PChar(Kernel32Name),Kernel32Size);
         Inc(PB4,RoundSize(Kernel32Size+1,2));

         PCardinal(PB3)^:=Cardinal(PB4)-
Cardinal(ImportSectionData)+ImportSection.VirtualAddress-2;
         CopyMemory(PB4,PChar(GetProcAddressName),GetProcAddressSize);
         Inc(PB4,RoundSize(GetProcAddressSize+1,2));
         Inc(PB3,SizeOf(DWORD));
         PCardinal(PB3)^:=Cardinal(PB4)-
Cardinal(ImportSectionData)+ImportSection.VirtualAddress-2;
         CopyMemory(PB4,PChar(LoadLibraryName),LoadLibrarySize);
         Inc(PB4,RoundSize(LoadLibrarySize+1,2));
         Inc(PB3,SizeOf(DWORD));
         Inc(PB3,SizeOf(DWORD));


         if (HostImportSectionSize<>0) and (ImageType=itDLL) and not DynamicDLL
then
         for I:=2 to ImportSectionDLLCount do
         begin
          ZeroMemory(@ImportDesc,SizeOf(ImportDesc));
          ImportDesc.Characteristics:=Cardinal(PB3)-
Cardinal(ImportSectionData)+ImportSection.VirtualAddress;
          ImportDesc.cName:=Cardinal(PB4)-
Cardinal(ImportSectionData)+ImportSection.VirtualAddress;

ImportDesc.cFirstThunk:=Pointer(ImportDesc.Characteristics+(NumberOfImports+1+2*
(ImportSectionDLLCount-1))*SizeOf(TImageThunkData));
```

```
          CopyMemory(PB2,@ImportDesc,SizeOf(ImportDesc));
          Inc(PB2,SizeOf(ImportDesc));

          while True do
          begin

PB:=VirtAddrToPhysAddr(Ptr,Pointer(PImportDesc^.cName+PImageNtHeaders(Ptr)^.Opti
onalHeader.ImageBase));
           Inc(PB,Cardinal(MainData));
           if (UpperCase(PChar(PB))<>UpperCase(Kernel32Name))
            and (UpperCase(PChar(PB))<>UpperCase(NtdllName)) then Break;
           Inc(PImportDesc);
          end;
          AnyDWORD:=Length(PChar(PB));
          CopyMemory(PB4,PB,AnyDWORD);
          Inc(PB4,RoundSize(AnyDWORD+1,2));

PB:=VirtAddrToPhysAddr(Ptr,Pointer(Cardinal(PImportDesc^.cFirstThunk)+PImageNtHe
aders(Ptr)^.OptionalHeader.ImageBase));
          Inc(PB,Cardinal(MainData));
          if PCardinal(PB)^ and $80000000=0 then
          begin

PB:=VirtAddrToPhysAddr(Ptr,Pointer(PCardinal(PB)^+PImageNtHeaders(Ptr)^.Optional
Header.ImageBase));
           Inc(PB,Cardinal(MainData));
           Inc(PB,2);
           AnyDWORD:=Length(PChar(PB));
           PCardinal(PB3)^:=Cardinal(PB4)-
Cardinal(ImportSectionData)+ImportSection.VirtualAddress;
           Inc(PB4,2);
           CopyMemory(PB4,PB,AnyDWORD);
           Inc(PB4,RoundSize(AnyDWORD+1,2));
          end else PCardinal(PB3)^:=PCardinal(PB)^;
          Inc(PImportDesc);
          Inc(PB3,SizeOf(DWORD));
          Inc(PB3,SizeOf(DWORD));
         end;

         PB3:=ImportSectionData;
         Inc(PB3,(ImportSectionDLLCount+1)*SizeOf(ImportDesc));
         PB:=PB3;
         AnyDWORD:=(NumberOfImports+1+2*(ImportSectionDLLCount-
1))*SizeOf(TImageThunkData);
         Inc(PB,AnyDWORD);
         CopyMemory(PB,PB3,AnyDWORD);
         ImportSectionDataSize:=Cardinal(PB4)-Cardinal(ImportSectionData);

         CopyMemory(@ImportSection.Name,PChar('.idata'),6);

ImportSection.Misc.VirtualSize:=RoundSize(ImportSectionDataSize,NtHeaders.Option
alHeader.SectionAlignment);

ImportSection.SizeOfRawData:=RoundSize(ImportSectionDataSize,RawDataAlignment);

ImportSection.PointerToRawData:=CodeSection.PointerToRawData+CodeSection.SizeOfR
awData;

NtHeaders.OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_IMPORT].Size:=Impor
tSection.SizeOfRawData;
         ImportSection.Characteristics:=IMAGE_SCN_CNT_CODE or
IMAGE_SCN_CNT_INITIALIZED_DATA or IMAGE_SCN_MEM_WRITE or IMAGE_SCN_MEM_READ;


CurVirtAddr:=ImportSection.VirtualAddress+ImportSection.Misc.VirtualSize;
         CurRawData:=ImportSection.PointerToRawData+ImportSection.SizeOfRawData;

         frmMain.AddLog('.idata віртуальна адреса блоку:
'+IntToHex(ImportSection.VirtualAddress,8),0);
```

```
         frmMain.AddLog('.idata віртуальний розмір блоку:
'+IntToHex(ImportSection.Misc.VirtualSize,8),0);

         // .tls блок
         if TlsSectionPresent then
         begin
          frmMain.AddLog('Побудова  .tls блоку',0);

TlsCopy.Directory:=@PImageNtHeaders(Ptr)^.OptionalHeader.DataDirectory[IMAGE_DIR
ECTORY_ENTRY_TLS];

TlsCopy.SectionData:=RVA2RAW(Ptr,MainData,TlsCopy.Directory.VirtualAddress);
          if TlsCopy.SectionData<>nil then
          begin
           TlsCopy.RawDataLen:=TlsCopy.SectionData^.RawDataEnd-
TlsCopy.SectionData^.RawDataStart;
           TlsCopy.RawData:=MyAlloc(TlsCopy.RawDataLen);

           PB:=RVA2RAW(Ptr,MainData,TlsCopy.SectionData^.RawDataStart-
HostImageBase);
           if PB<>nil then CopyMemory(TlsCopy.RawData,PB,TlsCopy.RawDataLen)
           else ZeroMemory(TlsCopy.RawData,TlsCopy.RawDataLen);

           PB:=RVA2RAW(Ptr,MainData,TlsCopy.SectionData^.AddressOfCallbacks-
HostImageBase);
           if PB=nil then
           begin
            TlsCopy.CallbacksLen:=4;
            TlsCopy.Callbacks:=MyAlloc(TlsCopy.CallbacksLen);
            ZeroMemory(TlsCopy.Callbacks,TlsCopy.CallbacksLen);
           end else
           begin
            TlsCopy.CallbacksLen:=GetTlsCallbacksLen(PB);
            TlsCopy.Callbacks:=MyAlloc(TlsCopy.CallbacksLen);
            CopyMemory(TlsCopy.Callbacks,PB,TlsCopy.CallbacksLen);
           end;

           ZeroMemory(@TlsSection,SizeOf(TlsSection));
           CopyMemory(@TlsSection.Name,PChar('.tls'),4);
           TlsSection.VirtualAddress:=CurVirtAddr;
           TlsSection.PointerToRawData:=CurRawData;
           TlsSection.Characteristics:=IMAGE_SCN_MEM_WRITE or
IMAGE_SCN_MEM_READ;

           ZeroMemory(@TlsSectionData,SizeOf(TlsSectionData));

TlsSectionData.RawDataStart:=NtHeaders.OptionalHeader.ImageBase+TlsSection.Virtu
alAddress+RoundSize(SizeOf(TlsSectionData),$10);

TlsSectionData.RawDataEnd:=TlsSectionData.RawDataStart+TlsCopy.RawDataLen;

TlsSectionData.AddressOfCallbacks:=RoundSize(TlsSectionData.RawDataEnd,$10);

TlsSectionData.AddressOfIndex:=RoundSize(TlsSectionData.AddressOfCallbacks+TlsCo
py.CallbacksLen,$08);

           TlsSection.SizeOfRawData:=RoundSize(TlsSectionData.AddressOfIndex-
TlsSection.VirtualAddress-
NtHeaders.OptionalHeader.ImageBase+$10,RawDataAlignment);

TlsSection.Misc.VirtualSize:=RoundSize(TlsSection.SizeOfRawData,NtHeaders.Option
alHeader.SectionAlignment);


NtHeaders.OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_TLS].VirtualAddress
:=CurVirtAddr;       //може змінюватися

NtHeaders.OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_TLS].Size:=TlsSecti
on.SizeOfRawData;
```

```
           CurVirtAddr:=TlsSection.VirtualAddress+TlsSection.Misc.VirtualSize;
           CurRawData:=TlsSection.PointerToRawData+TlsSection.SizeOfRawData;
          end else TlsSectionPresent:=False;
          frmMain.AddLog('.tls віртуальна адреса блоку:
'+IntToHex(TlsSection.VirtualAddress,8),0);
          frmMain.AddLog('.tls віртуальний розмір блоку:
'+IntToHex(TlsSection.Misc.VirtualSize,8),0);
         end;

         if ExportSectionPresent then
         begin
          frmMain.AddLog('Побудова  .edata блоку',0);
          ZeroMemory(@ExportSection,SizeOf(ExportSection));
          CopyMemory(@ExportSection.Name,PChar('.edata'),6);

          ExportSection.VirtualAddress:=CurVirtAddr;
          ExportSection.PointerToRawData:=CurRawData;
          ExportSection.Characteristics:=IMAGE_SCN_MEM_WRITE or
IMAGE_SCN_MEM_READ;

ExportSection.SizeOfRawData:=RoundSize(ExportSectionSize,RawDataAlignment);

ExportSection.Misc.VirtualSize:=RoundSize(ExportSection.SizeOfRawData,NtHeaders.
OptionalHeader.SectionAlignment);


NtHeaders.OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT].VirtualAddr
ess:=CurVirtAddr;       //може змінюватися

NtHeaders.OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT].Size:=Expor
tSection.SizeOfRawData;


CurVirtAddr:=ExportSection.VirtualAddress+ExportSection.Misc.VirtualSize;

CurRawData:=ExportSection.PointerToRawData+ExportSection.SizeOfRawData;

          ExportData:=MyAlloc(ExportSection.Misc.VirtualSize);
          ZeroMemory(ExportData,ExportSection.Misc.VirtualSize);


PB:=VirtAddrToPhysAddr(Ptr,Pointer(PImageNtHeaders(Ptr)^.OptionalHeader.DataDire
ctory[IMAGE_DIRECTORY_ENTRY_EXPORT].VirtualAddress+PImageNtHeaders(Ptr)^.Optiona
lHeader.ImageBase));
          if PB<>nil then Inc(PB,Cardinal(MainData));
          CopyMemory(ExportData,PB,ExportSectionSize);

          //фіксуємо RVAs у блоку експорту у Export Directory Table

ExportNamePointerRVAOrg:=PExportDirectoryTable(ExportData)^.NamePointerRVA;

ExportAddressRVAOrg:=PExportDirectoryTable(ExportData)^.ExportAddressTableRVA;
          ExportRVADelta:=ExportSection.VirtualAddress-
HostExportSectionVirtualAddress;

PExportDirectoryTable(ExportData)^.NameRVA:=PExportDirectoryTable(ExportData)^.N
ameRVA+ExportRVADelta;

PExportDirectoryTable(ExportData)^.ExportAddressTableRVA:=PExportDirectoryTable(
ExportData)^.ExportAddressTableRVA+ExportRVADelta;

PExportDirectoryTable(ExportData)^.NamePointerRVA:=PExportDirectoryTable(ExportD
ata)^.NamePointerRVA+ExportRVADelta;

PExportDirectoryTable(ExportData)^.OrdinalTableRVA:=PExportDirectoryTable(Export
Data)^.OrdinalTableRVA+ExportRVADelta;

          //+ фіксуємо RVAs у Export Name Pointer Table
```

```
PB2:=VirtAddrToPhysAddr(Ptr,Pointer(ExportNamePointerRVAOrg+PImageNtHeaders(Ptr)
^.OptionalHeader.ImageBase));
          Dec(PB2,Cardinal(PB)-Cardinal(MainData));
          Inc(PB2,Cardinal(ExportData));
          for I:=0 to PExportDirectoryTable(ExportData)^.NumberOfNamePointers-1
do
          begin
           PCardinal(PB2)^:=PCardinal(PB2)^+ExportRVADelta;
           Inc(PB2,SizeOf(DWORD));
          end;
          frmMain.AddLog('Експортуєма віртуальна адреса блоку:
'+IntToHex(ExportSection.VirtualAddress,8),0);
          frmMain.AddLog('Експортуємий віртуальний розмір блоку:
'+IntToHex(ExportSection.Misc.VirtualSize,8),0);
         end;

         if ResourceSectionPresent then
         begin
          frmMain.AddLog('Побудова  .rsrc блоку',0);

          ZeroMemory(@ResourceSection,SizeOf(ResourceSection));
          CopyMemory(@ResourceSection.Name,PChar('.rsrc'),5);

          ResourceSection.VirtualAddress:=CurVirtAddr;
          PrepareResourceSectionData;
          ResourceSection.PointerToRawData:=CurRawData;
          ResourceSection.Characteristics:=IMAGE_SCN_MEM_READ;

ResourceSection.SizeOfRawData:=RoundSize(ResourceSectionSize,RawDataAlignment);

ResourceSection.Misc.VirtualSize:=RoundSize(ResourceSection.SizeOfRawData,NtHead
ers.OptionalHeader.SectionAlignment);


NtHeaders.OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_RESOURCE].VirtualAd
dress:=CurVirtAddr;

NtHeaders.OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_RESOURCE].Size:=Res
ourceSection.SizeOfRawData;


CurVirtAddr:=ResourceSection.VirtualAddress+ResourceSection.Misc.VirtualSize;

CurRawData:=ResourceSection.PointerToRawData+ResourceSection.SizeOfRawData;

          frmMain.AddLog('Ресурсна віртуальна адреса блоку:
'+IntToHex(ResourceSection.VirtualAddress,8),0);
          frmMain.AddLog('Ресурсний віртуальний розмір блоку:
'+IntToHex(ResourceSection.Misc.VirtualSize,8),0);
         end;

         NtHeaders.OptionalHeader.SizeOfImage:=CurVirtAddr;

         frmMain.AddLog('Побудова  дескріптора імпорту ...',0);

         ZeroMemory(@ImportDesc,SizeOf(ImportDesc));

ImportDesc.Characteristics:=ImportSection.VirtualAddress+(NumberOfDLL+1)*SizeOf(
ImportDesc);

ImportDesc.cName:=ImportSection.VirtualAddress+(NumberOfDLL+1)*SizeOf(ImportDesc
)+(NumberOfImports+1)*SizeOf(TImageThunkData)*2;

ImportDesc.cFirstThunk:=Pointer(ImportDesc.Characteristics+(NumberOfImports+1)*S
izeOf(TImageThunkData));

ThunkGetProcAddress.Ordinal:=ImportSection.VirtualAddress+(NumberOfDLL+1)*SizeOf
(ImportDesc)+(NumberOfImports+1)*SizeOf(TImageThunkData)*2+Kernel32Size+2;
```

```
ThunkLoadLibrary.Ordinal:=ThunkGetProcAddress.Ordinal+GetProcAddressSize+2+2;

         ZeroMemory(@NullDesc,SizeOf(NullDesc));

TotalFileSize:=RoundSize(CurRawData,NtHeaders.OptionalHeader.FileAlignment);
         if OverlayPresent then Inc(TotalFileSize,AfterImageOverlaysSize);

         frmMain.AddLog('Побудова  поліморфичної частини ...',0);
         TrashSize:=KeyPtr;

         frmMain.AddLog('Адреса ключа: '+IntToHex(KeyPtr,8),0);
         frmMain.AddLog('Редактуємо адресу: '+IntToHex(LoaderPtr,8),0);
         frmMain.AddLog('Розмір байтів доданого сміття:
'+IntToStr(TrashSize),0);
         frmMain.AddLog('Розмір байтів доданого сміття2:
'+IntToStr(Trash2Size),0);
         Trash:=MyAlloc(TrashSize);
         Trash2:=MyAlloc(Trash2Size);
         if (Trash<>nil) and (Trash2<>nil) then
         begin
          GenerateRandomBuffer(Trash,TrashSize);
          GenerateRandomBuffer(Trash2,Trash2Size);


NtHeaders.OptionalHeader.AddressOfEntryPoint:=CodeSection.VirtualAddress+LoaderP
tr+LoaderSize+Trash2Size;
          frmMain.AddLog(' Виконуєма точка входу:
'+IntToHex(NtHeaders.OptionalHeader.AddressOfEntryPoint,8),0);
          InitData:=MyAlloc(InitSize);
          if InitData<>nil then
          begin

VirtLoaderData:=NtHeaders.OptionalHeader.ImageBase+CodeSection.VirtualAddress+Lo
aderPtr;

VirtMainData:=NtHeaders.OptionalHeader.ImageBase+CodeSection.VirtualAddress+Load
erPtr+LoaderSize+Trash2Size+InitSize;

VirtKey:=NtHeaders.OptionalHeader.ImageBase+CodeSection.VirtualAddress+KeyPtr;

           //initiate DynLoader image info
           PB:=Pointer(Cardinal(LoaderData)+LoaderSize);
           while PCardinal(PB)^<>DYN_LOADER_END_MAGIC do Dec(PB);
//DYN_LOADER_END_MAGIC search
           Dec(PB,5);
           PCardinal(PB)^:=MainRealSize;
           Dec(PB,7);
           PCardinal(PB)^:=NtHeaders.OptionalHeader.AddressOfEntryPoint;
           Dec(PB,7);
           PCardinal(PB)^:=NtHeaders.OptionalHeader.SizeOfImage;
           Dec(PB,7);
           case ImageType of
            itExe:PCardinal(PB)^:=IMAGE_TYPE_EXE;
            itDLL:PCardinal(PB)^:=IMAGE_TYPE_DLL;
            itSys:PCardinal(PB)^:=IMAGE_TYPE_SYS;
            else PCardinal(PB)^:=IMAGE_TYPE_UNKNOWN;
           end;

           //фіксуємо точки у DynLoader
           //це 3 інструкції, які запам'ятовуються

LdrPtrCode:=NtHeaders.OptionalHeader.ImageBase+CodeSection.VirtualAddress;

LdrPtrThunk:=NtHeaders.OptionalHeader.ImageBase+Cardinal(InitcodeThunk);

           LdrPtr:=LoaderData;
           Inc(LdrPtr);
           PCardinal(LdrPtr)^:=LdrPtrThunk+4-LdrPtrCode;
```

```
            Inc(LdrPtr,5);
            PCardinal(LdrPtr)^:=LdrPtrThunk-LdrPtrCode;
            Inc(LdrPtr,5);
            PCardinal(LdrPtr)^:=VirtMainData-LdrPtrCode;

            DynCoder(LoaderData,LoaderSize,Key);

            frmMain.AddLog(' Генерація коду ініціалізації ...',0);


//GenerateInitCode(ACodePtr,AKeyPtr,AData1Ptr,ASize1,AData2Ptr,ASize2,ADynLoadAd
dr,AMainPtr,
            //                 AEntryPointAddr,AImpThunk:Cardinal);


GenerateInitCode(LdrPtrCode,KeyPtr,LoaderPtr,LoaderSize,LoaderPtr+LoaderSize+Tra
sh2Size+InitSize,

MainRealSize,NtHeaders.OptionalHeader.ImageBase+CodeSection.VirtualAddress+Loade
rPtr,

VirtMainData,NtHeaders.OptionalHeader.ImageBase+NtHeaders.OptionalHeader.Address
OfEntryPoint,
                            LdrPtrThunk);


FileHandle:=CreateFile(PChar(OutputFileName),GENERIC_WRITE,0,nil,CREATE_ALWAYS,F
ILE_ATTRIBUTE_NORMAL,0);
            if FileHandle<>INVALID_HANDLE_VALUE then
            begin
             SetFilePointer(FileHandle,TotalFileSize,nil,FILE_BEGIN);
             SetEndOfFile(FileHandle);
             if not Quiet then
             begin
              frmMain.AddLog(' Розмір нового блоку: '+IntToStr(TotalFileSize-
MainRealSize)+' bytes',0);
              LogCnt:=0;
              Inc(LogCnt,SizeOf(DosHeader));
              Inc(LogCnt,SizeOf(DosStub));
              Inc(LogCnt,SizeOf(DosStubEnd));
              Inc(LogCnt,SizeOf(NtHeaders));
              Inc(LogCnt,SizeOf(CodeSection));
              Inc(LogCnt,SizeOf(ImportSection));
              if ExportSectionPresent then
              begin
               Inc(LogCnt,SizeOf(ExportSection));
              end;

              LogCnt:=CodeSection.PointerToRawData;
              Inc(LogCnt,TrashSize);
              Inc(LogCnt,KeySize);
              Inc(LogCnt,LoaderSize);
              Inc(LogCnt,Trash2Size);
              Inc(LogCnt,InitSize);

              LogCnt:=ImportSection.PointerToRawData;
              if TlsSectionPresent then
              begin
               LogCnt:=TlsSection.PointerToRawData;
              end;
              if ExportSectionPresent then
              begin
               LogCnt:=ExportSection.PointerToRawData;
              end;
              if ResourceSectionPresent then
              begin
               LogCnt:=ResourceSection.PointerToRawData;
              end;
```

```
             if OverlayPresent then
             begin
              LogCnt:=TotalFileSize-AfterImageOverlaysSize;
             end;
            end;

            // заглушка
            SetFilePointer(FileHandle,0,nil,FILE_BEGIN);
            WriteFile(FileHandle,DosHeader,SizeOf(DosHeader),NumBytes,nil);
            WriteFile(FileHandle,DosStub,SizeOf(DosStub),NumBytes,nil);
            WriteFile(FileHandle,DosStubEnd,SizeOf(DosStubEnd),NumBytes,nil);
            WriteFile(FileHandle,NtHeaders,SizeOf(NtHeaders),NumBytes,nil);
            WriteFile(FileHandle,CodeSection,SizeOf(CodeSection),NumBytes,nil);

WriteFile(FileHandle,ImportSection,SizeOf(ImportSection),NumBytes,nil);
            if TlsSectionPresent then
WriteFile(FileHandle,TlsSection,SizeOf(TlsSection),NumBytes,nil);
            if ExportSectionPresent then
WriteFile(FileHandle,ExportSection,SizeOf(ExportSection),NumBytes,nil);
            if ResourceSectionPresent then
WriteFile(FileHandle,ResourceSection,SizeOf(ResourceSection),NumBytes,nil);


SetFilePointer(FileHandle,ImportSection.PointerToRawData,nil,FILE_BEGIN);

WriteFile(FileHandle,ImportSectionData^,ImportSection.SizeOfRawData,NumBytes,nil
);

            // Блок коду

SetFilePointer(FileHandle,CodeSection.PointerToRawData,nil,FILE_BEGIN);

            // Переходний блок імпорту, який переміщується у кінець коду
ініціалізації
            WriteFile(FileHandle,Trash^,TrashSize,NumBytes,nil);
            WriteFile(FileHandle,Key^,KeySize,NumBytes,nil);
            WriteFile(FileHandle,LoaderData^,LoaderSize,NumBytes,nil);
            WriteFile(FileHandle,Trash2^,Trash2Size,NumBytes,nil);
            WriteFile(FileHandle,InitData^,InitSize,NumBytes,nil);

            // Блок даних
            WriteFile(FileHandle,MainDataCyp^,MainSize,NumBytes,nil);

            // Tls блок
            if TlsSectionPresent then
            begin

SetFilePointer(FileHandle,TlsSection.PointerToRawData,nil,FILE_BEGIN);

WriteFile(FileHandle,TlsSectionData,SizeOf(TlsSectionData),NumBytes,nil);

             Delta:=TlsSectionData.RawDataStart-TlsSection.VirtualAddress-
NtHeaders.OptionalHeader.ImageBase;

SetFilePointer(FileHandle,TlsSection.PointerToRawData+Delta,nil,FILE_BEGIN);

WriteFile(FileHandle,TlsCopy.RawData^,TlsCopy.RawDataLen,NumBytes,nil);

             Delta:=TlsSectionData.AddressOfCallbacks-TlsSection.VirtualAddress-
NtHeaders.OptionalHeader.ImageBase;

SetFilePointer(FileHandle,TlsSection.PointerToRawData+Delta,nil,FILE_BEGIN);

WriteFile(FileHandle,TlsCopy.Callbacks^,TlsCopy.CallbacksLen,NumBytes,nil);
            end;

            // Блок експорту
            if ExportSectionPresent then
            begin
```

```
SetFilePointer(FileHandle,ExportSection.PointerToRawData,nil,FILE_BEGIN);

WriteFile(FileHandle,ExportData^,ExportSection.SizeOfRawData,NumBytes,nil);
             if ExportData<>nil then MyFree(ExportData);
            end;

            //  блок ресурсу
            if ResourceSectionPresent then
            begin

SetFilePointer(FileHandle,ResourceSection.PointerToRawData,nil,FILE_BEGIN);

WriteFile(FileHandle,ResourceData^,ResourceSection.SizeOfRawData,NumBytes,nil);
             if ResourceData<>nil then MyFree(ResourceData);
            end;

            // Оверлейні дані
            if OverlayPresent then
            begin
             SetFilePointer(FileHandle,TotalFileSize-
AfterImageOverlaysSize,nil,FILE_BEGIN);

WriteFile(FileHandle,AfterImageOverlays^,AfterImageOverlaysSize,NumBytes,nil);
            end;

            frmMain.AddLog('Файл успішно захищено',0);
            frmMain.StatusBar1.Panels.Items[0].Text:='Файл успішно захищено';
            CloseHandle(FileHandle);
           end else ErrorMsg('Неможливо створити вихідний файл.');
           MyFree(InitData);
          end else ErrorMsg('Неможливо виділити пам’ять під ініціалізаційні
дані');
          MyFree(Trash);
          MyFree(Trash2);
         end else ErrorMsg('Неможливо виділити пам’ять під дані байтів
засмітчення.');
         if TlsSectionPresent then
         begin
          MyFree(TlsCopy.RawData);
          MyFree(TlsCopy.Callbacks);
         end;
         MyFree(Key);

         if ImportSectionData<>nil then MyFree(ImportSectionData);
        end else ErrorMsg('Неможливо виділити пам’ять  під ключ кодування.');
        MyFree(LoaderData);
       end else ErrorMsg('Неможливо виділити пам’ять  для редагування.');
      end else ErrorMsg('Неможливо генерувати кодер/декодер');
     end else ErrorMsg('Підсистема не підтримується .');
    end else ErrorMsg(' Вхідний файл є не валідним PE файлом.');
   end else ErrorMsg('Неможливо прочитати файл.');
   MyFree(MainData);
  end else ErrorMsg('Неможливо виділити пам’ять  для даних вхідного файлу.');
  if MainFile<>INVALID_HANDLE_VALUE then CloseHandle(MainFile);
 end else ErrorMsg('Неможливо відкрити файл.');
end;
end.
```

Файл about.pas - довідка про програму

```
unit about;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls, ExtCtrls;

type
  Tfrm_about = class(TForm)
    Image1: TImage;
    Memo1: TMemo;
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  frm_about: Tfrm_about;

implementation

{$R *.dfm}

procedure Tfrm_about.Button1Click(Sender: TObject);
begin
frm_about.Close;
end;

procedure Tfrm_about.FormCreate(Sender: TObject);
begin
Memo1.Clear;
Memo1.Lines.Add('МАГІСТЕРСЬКА РОБОТА');
Memo1.Lines.Add('');
Memo1.Lines.Add('на  тему:');
Memo1.Lines.Add('');
Memo1.Lines.Add('Дослідження та програмна реалізація системи для протидії
декомпіляції коду');
Memo1.Lines.Add('');
Memo1.Lines.Add('');
Memo1.Lines.Add('');
Memo1.Lines.Add('Керівник: Смірнов С.А.');
Memo1.Lines.Add('');
Memo1.Lines.Add('Розробив: студент Федоров Богдан Сергійович ');
Memo1.Lines.Add('                  гр. КІ-21М-1,4');
Memo1.Lines.Add('');
Memo1.Lines.Add('');
Memo1.Lines.Add('м. Кропивницький 2022');
end;
end.
```