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
“____” _____ 2024 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення системи кібербезпеки ускладнення аналізу алгоритмів роботи вихідного коду”

Виконав здобувач вищої освіти
IV курсу, групи КБ-21-3СК
ОПП «Кібербезпека»
спеціальності 125 «Кібербезпека»
_____ Марченко Б.С.
«____» _____ 2024 р.

Керівник проекту
доктор технічних наук, професор
_____ Смірнов О.А.
«____» _____ 2024 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет
Факультет *Механіко-технологічний*
Кафедра *Кібербезпеки та програмного забезпечення*
Освітній ступінь *бакалавр*
Галузь знань ._12 _____*"*Інформаційні технології*"*_____
Спеціальність *125 "*Кібербезпека*"*
Освітньо-професійна (освітньо-наукова) програма _*"*Кібербезпека*"*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 17 » січня 2024 року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ПЕРШИМ (БАКАЛАВРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Марченку Богдану Станіславовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Програмне забезпечення системи кібербезпеки ускладнення аналізу алгоритмів роботи вихідного коду*

2. Керівник роботи *Смірнов Олексій Анатолійович, докт. техн. наук, професор*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 136-02 від 01.04.2024 року

3. Строк подання студентом роботи до захисту *23.05.2024 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою роботи є розробка програмного забезпечення системи кібербезпеки ускладнення аналізу алгоритмів роботи вихідного коду*

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

1. Призначення та область використання.
2.Перегляд аналогічних існуючих систем.
3. Опис і обґрунтування проектних рішень.
4. Етапи програмування системи.
5. Впровадження системи кібербезпеки в промислову експлуатацію.
6. Висновки

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

Структурна схема системи кібербезпеки	*1 аркуш*
Функціональна схема системи кібербезпеки	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*

7. Дата видачі завдання « 17 » січня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.03.2024 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2024 р.	
3.	Розробка моделі компонента	20.03.2024 р.	
4.	Розробка структур даних	25.03.2024 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2024 р.	
6.	Програмування алгоритмів	10.04.2024 р.	
7.	Оформлення ПЗ	17.04.2024 р.	
8.	Попередній захист роботи	23.05.2024 р.	

Дата видачі завдання
« 17 » січня 2024 р.

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

Завдання прийнято до виконання
« 17 » січня 2024 р.

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

АНОТАЦІЯ

Марченко Б.С. Програмне забезпечення системи кібербезпеки ускладнення аналізу алгоритмів роботи вихідного коду. 125 Кібербезпека. Центральноукраїнський національний технічний університет. Кропивницький. 2024.

В даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи кібербезпеки ускладнення аналізу алгоритмів роботи вихідного коду.

Метою розробки є програмне забезпечення системи кібербезпеки ускладнення аналізу алгоритмів роботи вихідного коду.

Результат роботи – програмна реалізація системи кібербезпеки ускладнення аналізу алгоритмів роботи вихідного 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 з ОС Windows 10/11.

Програму розроблено в середовищі RAD Studio Delphi.

Ключові слова: кібербезпека, ускладнення аналізу алгоритмів роботи вихідного коду

ABSTRACT

Marchenko B.S. The software of the cyber security system complicates the analysis of the algorithms of the source code. 125 Cyber security. Central Ukrainian National Technical University. Kropyvnytskyi. 2024.

In this graduation thesis for the first (bachelor) level of higher education, software is developed, which is intended for the cyber security system of the complication of the analysis of algorithms of the source code.

The purpose of the development is the software of the cyber security system to complicate the analysis of the algorithms of the source code.

The result of the work is the software implementation of the cyber security system, which complicates the analysis of the algorithms of the source code.

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 a PC with Windows 10/11 OS.

The program was developed in the RAD Studio Delphi environment.

Keywords: cyber security, complications of analysis of source code algorithms

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ 2
ВСТУП 3
1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ 5
1.1 Призначення системи 5
1.2 Область застосування 5
2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ 7
2.1 Огляд існуючих систем, технологій, архітектур та програмних рішень за профілем теми випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти 7
2.2 Обґрунтування вибору засобів для побудови системи кібербезпеки та мови програмування 15
2.3 Розгорнута постановка завдання 21
3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ 22
3.1 Опис функціонування системи 22
3.2 Розробка структурної схеми 27
3.3 Розробка функціональної схеми 36
3.4 Розробка діаграми процесів 53
4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ 54
4.1 Розробка блок-схем та опис алгоритмів функціонування системи 54
4.2 Захист розробленого програмного забезпечення 61
5 ВПРОВАДЖЕННЯ СИСТЕМИ КІБЕРБЕЗПЕКИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ 64
6 ОСНОВНІ ВИСНОВКИ 66
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 68

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

ПЗ	–	програмне забезпечення
ПП	–	програмний 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 високе уповільнення. 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.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи кібербезпеки ускладнення аналізу алгоритмів роботи вихідного коду.

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

– Огляд існуючих систем ускладнення аналізу алгоритмів роботи вихідного коду.

– Дослідження системи кібербезпеки ускладнення аналізу алгоритмів роботи вихідного коду.

– Програмна реалізація системи кібербезпеки ускладнення аналізу алгоритмів роботи вихідного коду.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі ускладнення аналізу алгоритмів роботи вихідного коду.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи кібербезпеки ускладнення аналізу алгоритмів роботи вихідного коду, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

У цей час найпоширеніша методика автоматичного захисту програм від аналізу на основі технології віртуалізації машинного коду. Дана методика ставить у відповідність кожної машинної інструкції одну або кілька інструкцій автоматично згенерованого віртуального процесора, називаних байт-кодом або псевдокодом. У тіло програми, яка захищається, вбудовується захищений від аналізу інтерпретатор, завданням якого є виконання згенерованого на етапі захисту байт-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 захисти й забезпечення більше 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.

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

1.2 Область застосування

У бакалаврському проекті досліджується захист програм від несанкціонованої модифікації, аналізу й налагодження, які були створені за

допомогою компілюємих мов і на момент захисти являють собою файли певного формату (в основному PE і COFF формати). Код являє собою машинний код цільової апаратної платформи.

Проведемо аналіз поточного стану проблеми автоматичного захисту програмного забезпечення від комп'ютерного піратства. У результаті проведеного аналізу було виявлено, що існуючі методики розраховані або на наявність вихідного коду програми, що надзвичайно утрудняє рішення завдання контролю цілісності програми – або мають надзвичайно високе уповільнення. Було також відзначене, що для більшості методик автоматичного захисту програм, що не вимагають наявності вихідного коду, існують механізми автоматичної деактивації захисту.

Ці обставини визначають особливу актуальність питань створення методик автоматичного захисту програм від аналізу й несанкціонованої модифікації, які з однієї сторони не вимагали б наявності вихідного коду програми й забезпечували б невисоке уповільнення, а з іншої сторони захищали б від автоматичних механізмів деактивації захисту. В основі їх лежить принцип програмного «чорного ящика», реалізація якого припускає використання технологій заплутування коду й даних програми або обфускації.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи кібербезпеки ускладнення аналізу алгоритмів роботи вихідного коду, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

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

Проведемо огляд обфускаторів .NET. Активний розвиток платформи .NET сприяє тому, що усе більше розроблювачів використовують її для своїх проектів. Не є тут виключенням і область комерційної програмного забезпечення. І якщо кілька років назад використання платформи в цій області трохи гальмувалося через те, що розроблювачі побоювалися, того, що кінцевий користувач не побажає встановлювати .NET на свій комп'ютер. Сьогодні ця проблема вже напевно не стоїть так гостро, на більшості комп'ютерів фреймворк передвстановлений і користувачеві як правило немає необхідності турбується із цього приводу.

Однак специфічною проблемою поширення комерційного програмного забезпечення написаного на .NET є той факт, що на відміну від традиційних Windows-додатків, які при компіляції перетворяться в низькорівневий машинний код, NET-додатки компілюються в MSIL це процессоронезалежна проміжна мова, створена Microsoft. Якщо додаток ніяк не захистити, то за допомогою спеціальних інструментів типу .NET Reflector можна за кілька хвилин виконати декомпіляцію й відновити вихідний код додатку в придатному для вивчення виді. Завдання захисту вихідного коду програми від вивчення сторонніми особистостями вирішують спеціальні програми – обфускатори. Обфускатори ускладнюють вихідні коди програми, заплутують його, роблять більше важкими для розуміння, але при цьому зберігають функціональність працездатної.

Також, крім проблеми із захистом від перегляду коду, специфічної для .NET додатків, актуальним є також і питання захисту програми від

несанкціонованого використання. У переважній більшості випадків такий захист будується на основі використання файлів ліцензії (або серійних номерів). Дані файли можуть генеруватися із прив'язкою до "заліза" комп'ютера користувача так, щоб ними не можна було скористатися на інших комп'ютерах. У цьому випадку як правило використовується machine key, що може бути отриманий за допомогою WQL. Однак може використовуватися й інший спосіб захисту додатку, специфічний для додатку.

Як правило, розробка якісної системи захисту самотужки – процес досить складний і трудомісткий. Тому більшість розроблювачів намагаються вибрати більше просте й швидке рішення – скористатися готовою системою захисту.

У даному огляді я постараюся розглянути найпоширеніші рішення захисту .NET додатків і мимохіть пробігтися по їхніх основних особливостях. Свою увагу в огляді я постараюся приділити наступним моментам:

– Вартість інструмента (single license).

– Наявність trial-періоду й безкоштовної редакції.

– Наявність механізму ліцензування (evaluate-версія, генерація серійних номерів, прив'язка до комп'ютера й т.д.).

– Наявність і якість методів обфускації.

– Шифрування чутливих даних і захист від налагодження.

– Додаткові можливості, специфічні для кожного окремого інструмента.

Всі продукти, розглянуті в цьому огляді, я розмістив по зростанню вартості продукту, тобто спочатку йдуть більше дешеві, потім більше дорогі. Якщо ви не випробовуєте фінансових складностей, можете почати вивчення із самого кінця, там точно написано куди можна витратити ваші гроші

Dotfuscator

Один з найпоширеніших обфускаторів для .NET. Його особливістю є наявність безкоштовної версії: Dotfuscator CE, що також іде в складі Microsoft Visual Studio починаючи з версії 2003. Знайти його можна в меню Tool – Dotfucsator CE. На жаль дана версія має багато обмежень, як

наприклад відсутність шифрування рядків. Старший брат Dotfuscator PRO коштує грошей і призначена для корпоративного бізнесу. Dotfuscator PRO пропонує підвищений захист, більше зроблені методи й прийоми обфускації.

Phoenix Protector

Ще один абсолютно безкоштовний інструмент. Список його можливостей складається з обфускації, причому, їсти можливість використовувати убудований список виключень, для уникнення небажаних результатів, і склеювання складань. Незважаючи на настільки невеликий перелік можливостей, механізм обфускації реалізований досить уміло, і його цілком достатньо для базового захисту продуктів .NET.

Eazfuscator.NET

Eazfuscator.NET – це безкоштовний обфускатор для .NET платформи. Основною метою інструмента є захист інтелектуальної власності програмного забезпечення. Серед особливостей можна виділити наявність інтеграції з Visual Studio, підтримку .NET CF і Silverlight. По завіреннях розроблювача обфускація зроблена на дуже гарному рівні й захисті піддається 100% керованого коду. Крім обфускації, інструмент надає також строкове шифрування й автоматичну оптимізацію коду.

C# Source Code Formatter

Даний продукт є представником великого сімейства продуктів для різних мов програмування, основною метою яких є форматування вихідного коду, коментарів, вирівнювання, зміна кодування ASCII, Unicode(UTF-8, UTF-16). На додаток до даної функціональності інструмент дозволяє обфускувати додатки .NET. Я по правді не зміг придумати, чим даний продукт у плані захисту може бути краще інших, навіть безкоштовних аналогів, описаних вище.

.Net Reactor

Мабуть самий розкручений серед комерційних продукт. Принаймні, як тільки я зацікавився питанням захисту, воно мені попався чи ледве не найпершим. У ньому є весь спектр функціональності для розроблювача:

обфускація, захист від декомпіляції, шифрування рядків, механізм склеювання складань в один моноліт, система ліцензування й ще багато чого. Крім того продукт має зручний інтуїтивний користувальницький інтерфейс, що робить роботу із продуктом простою і інтуїтивною.

Однак же, відгуки про продукт досить неоднозначні. Багато говорять про нього поганого, як те примітивні методи обфускації й слабкий непрофесійний захист, тормознутість як самого продукту, так і захищених з його допомогою рішень. Однак досить і позитивних відкликань, у яких як сильні сторони продукту відзначають невисоку вартість і зручність і простоту використання.

Так чи інакше, але дійсно невисока вартість у порівнянні з конкурентами затьмарює якщо не всі, те більшу частину недоліків інструмента для невибагливих користувачів. Професійним рішенням даний продукт, напевно, назвати складно, але як відправна точна для невеликих стартапів – дане рішення буде дуже до речі.

Skater .NET Obfuscator

Даний інструмент крім комерційної версії містить безкоштовний варіант, що має інтерфейс командного рядка й уміють обфускувати тільки лише приватні члени класів.

Для більшого вже потрібно заплатити. Взагалі редакцій в даного інструмента існує багато, Найбільш функціональної й більше підходящої під наші умови які були перераховані на початку статті підходить Ultimate Edition. У дану редакцію входить повна обфускація кода, строкове шифрування й функціональність керування ліцензуванням.

Themida / WinLicence

Themida відома не тільки для .NET розроблювачів, але так само й для багатьох інших напрямків. Це досить гарна, що зарекомендувала себе система, що допомагає вирішити багато питань у тому числі й специфічні для даного огляду обфускацію й ліцензування кінцевого продукту. Для захисту продуктів

феміда використовує технологію захисту SecureEngine® яка, виконується на рівні драйверів, тому надавана їй захист програм дійсно на дуже високому рівні.

Dotnet Protector

PV Logiciels dotNet Protector – це потужний інструмент в арсенал можливостей якого входить захист коду .NET, обфускація, керування ліцензуванням. Специфікою ліцензування є наявність в інструменті засобів для створення evaluate копій продукту, що захищається, і інтегрованого hardware блокування. Розроблювачі даного інструмента французи, тому ціна продукту обчислюється в євро, і базова редакція інструмента починається з 300 євро.

Inquartos Obfuscator

Даний інструмент є позиціонується як перший вітчизняний продукт по захисту .NET продуктів. Серед його можливостей можна виділити механізми обфускації, у тому числі й декларативної, шифрування, об'єднання складань, захист від налагодження, оптимізація коду й керування ліцензуванням. Судячи з відкликань є багато недоліків і проблемних ситуацій, однак у цілому продукт виглядає дуже гідно. Крім того існує безкоштовна утиліта яка називається NetObf-Protector. Утиліта має простим, зручним і інтуїтивно зрозумілий інтерфейс, і забезпечує .NET додаток базовими засобами захисту. Небагато, що насторожує моментом, є що вже більше року на офіційному сайті не ведуться роботи.

Smart Assembly

SmartAssembly призначена для обфускації й захисту вихідного коду. Крім цього система здатна виконувати завдання оптимізації, спрощення розгортання .NET рішень, зменшення їхніх розмірів, збільшення продуктивності. Незважаючи на свою досить високу вартість система користується незмінною популярністю в розроблювачів, як має дуже якісну функціональність і здатна вирішити всі питання, що входять у коло її можливостей на високому професійному рівні.

PC Guard .Net

PC Guard .NET – професійне рішення для захисту й ліцензування продуктів написаних на .NET і Win32. Правда з недавніх пор механізм ліцензування був виділений в окремий продукт: activation suite, так що для того що б скористатися даною функціональністю ціну продукту, що до слова, так само як і ціна на Dotnet Protector, оголошена в євро, потрібно помножити на два. Сам же інструмент має дійсно високий рівень захисту продуктів і самодостатньою системою ліцензування. Як говориться, були б гроші.

Demeanor for .NET

Demeanor for.NET – це інструмент який одержить у собі функції захисту, обфускації, оптимізації продуктів, зменшення розмірів продукту й збільшення швидкості завантаження. Крім того, присутні засобу інтеграції із процесом складання, зокрема з MS Build. По завіреннях розроблювачів і відгуками користувачів усе реалізовано на вищому рівні.

Salamander .NET Obfuscator

Кілька років назад, даний продукт був на піку популярності, однак з недавніх пор проект перестав розвиватися, і як наслідок втратив інтерес із боку розроблювачів. Незважаючи на це продукт усе ще існує. Можливості інструмента обмежуються на досить якісній обфускації складань і захисту їхніх ресурсів.

DNGuard HVM

Інструмент містить засобу дуже гарні засоби обфускації, захисту коду, ресурсів, шифрування рядків і керування ліцензуванням продукту. Засоби обфускації крім усього іншого як і для CliSecure, що буде описаний пізніше, дозволяють використовувати спеціальні атрибути. Старанні китайці постаралися на славу намагаючись реалізувати дуже серйозний захист. У її основі лежить технологія HVM, що не доступна в trial версії.

Spices.Net Suite

Продукт складається з потужного обфускатора, що дозволяє конфігурувати виключення обфускації, включення, маски імен, атрибути

обфускації, декомпілятора, компонентів засоби, що представляє, для моделювання структури складань, дослідження метаданих проекту, інформації про складові частини проекту а також інгегаційний пакет до Visual Studio підтримуючу роботу з MSBuild і NAnt. Варто відзначити, що продукт не містить ніякої системи ліцензування, для цього він не призначений. Основне його завдання – це допомога розроблювачеві на етапі розробки, і підготовки до публікації продукту. Із цим Spice.NET Suite справляється вище всяких похвал, принаймні вкрай негативних відгуків про даний продукт я не знайшов.

CodeVeil

CodeVeil – це інструмент, що містить у собі функції обфускації й шифрування MSIL, ресурсів і модулів. Крім того, використання даного інструмента дозволяє захистити продукт від налагодження, трейсингу й модифікації коду, що виконується.

Існує більше легка версія CodeVeil яка коштує 99$, але її функціональність зведена до мінімуму. Там немає підтримки 64-бітних складань, відсутня функціональність шифрування, захист від налагодження, модифікації, трейсингу. Взагалі мінімум можливостей за мінімум грошей.

Крім цього в XHEO існує окремий продукт для ліцензування: Licensing, що пропонує зручний інструмент для керування ліцензіями продукту. Існує багато редакцій Licensing, ціна на кожну редакцію корелює з її можливостями й починається з 169$.

CliSecure

Даний інструмент має наступну функціональність: професійна обфускація, що включає тім числі підтримку декларативної обфускації за допомогою атрибутів, якісна захист коду й ресурсів, захист від налагодження, механізм керування ліцензуванням, інтеграція з MSBuild і Nant. Існує безкоштовна версія, у якій доступні тільки лише функціональність обфускації. Керування ліцензіями, шифрування й механізми захисту доступні тільки в платній версії.

Xenocode

XenoCode це потужний, гнучкому й легкий у використанні інструмент призначений для захисту й оптимізації додатків для різних технологій у тому числі й .NET рішень. Серед особливостей інструмента можна виділити обфускацію й шифрування коду й метаданих, оптимізацію продуктивності продуктів, що захищаються, простоту в конфігурації й використанні. Крім того даний інструмент дозволяє конвертувати Windows, .NET, Java, AIR, Flash, Shockwave або інше Windows-сумісний додаток у самостійний віртуальний додаток (один єдиний що виконується *.exe файл), яких можна запускати без його установки на будь-якому користувальницькому комп'ютері.

В огляд не потрапили продукти які перестали підтримуватися. Так серед таких от нерозглянутих, але про які я щось десь чув залишилися інструменти: LSW IL-Obfuscator 2.0, Deploy.NET, Dynu .net protector, Assembly Lockbox, Maxtocode, Aspose.Obfuscator, SharpObfuscator. Багато інструментів у ході своєї еволюції змінили свої основні завдання, і тепер у них методи захисту, такі як обфускація або шифрування, є залишковим явищем, приклад тут може бути Xenocode. Або ж є інший клас інструментів, для яких обфускація ніколи не була головною особливістю, а служила як доповнення до основної функціональності. От до таким можна віднести C# Source CodeFormatter. Напевно багато чого не потрапило в поле мого зору й по яким те причинах було мною пропущено. На жаль, такий момент теж присутній. Однак я думаю, що того що я все таки описав у даному огляді вистачить навіть самому вимогливому розроблювачеві що б почати захищати свою інтелектуальну власність використовуючи якісні методи й інструменти.

2.2 Обґрунтування вибору засобів для побудови системи кібербезпеки 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 першим (бакалаврським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи кібербезпеки ускладнення аналізу алгоритмів роботи вихідного коду.

В процесі розробки випускної кваліфікаційної роботи за першим (бакалаврським) 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 кібербезпеки 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 кібербезпеки 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

Одним з найвідоміших фахівців в області обфускації – професором Б. Бараком, були сформульовані основні вимоги, яким повинен відповідати ідеальний алгоритм заплутування (обфускатор):

1. Властивість функціональності. Заплутаний алгоритм повинен виконувати ту ж функцію, що й вихідний.

2. Поліноміальне уповільнення. Заплутаний алгоритм повинен працювати в поліноміальне число раз повільніший, ніж вихідний. Аналогічне можна сказати й про збільшення розміру коду після заплутування.

3. Властивість віртуального «чорного ящика». Не повинне існувати алгоритму розпізнавання обфускації більше ефективного, ніж звичайне припущення зроблене на основі аналізу входів і виходів заплутаної програми

Бараком було доведено, що властивість віртуального «чорного ящика» у загальному випадку не виконується. Однак при аналізі доказу був виявлений ряд недоліків, які дозволяють засумніватися в його коректності в ряді випадків.

Докладно дослідимо технологію віртуалізації коду, що має на увазі під собою переклад машинного коду в код деякого віртуального процесора, програмний емулятор якого вбудовується в тіло програми, яка захищається, і дозволяє інтерпретувати отриманий псевдокод. Показано, що стійкість інтерпретатора є невисокою, а, отже, необхідно застосувати перетворення, що заплутують, стосовно інтерпретатора, що сильно зменшує швидкість роботи захищеного додатку (звичайно швидкість виконання захищеної ділянки коду менше швидкості виконання вихідного варіанта на 3-4 порядки). У силу обмеження по швидкості перетворення, що заплутують, застосовані до

інтерпретатора, не можуть мати високу стійкість, а, отже, можливе створення декомпілятора отриманого псевдокоду.

У результаті аналізу існуючих підходів, використовуваних для захисту програм від комп'ютерного піратства, був зроблений вивід про необхідність створення методик захисту програм, заснованих на технології обфускації, які з однієї сторони не вимагали б наявності вихідного коду програми й забезпечували з не більше ніж на 2 порядки, а з іншого боку – задовольняли умові стійкості стосовно алгоритмів деобфускації, що працюють за поліноміальний час:

$$\forall M \in П \; \exists O \in O^*, O^* \subset П : \forall A(O(M)) \in П, A(O(M)) \not\subset P \qquad (3.1),$$

де:

– $П$ – множина всіх програм, процес роботи яких має кінцевий результат;

– M – програма з множини $П$;

– O – алгоритм обфускації;

– O^* – сімейство алгоритмів обфускації;

– P – множина алгоритмів поліноміальної складності;

– A – алгоритм деобфускації, що приймає на вхід заплутаний алгоритм $O(M)$.

Розглянемо модифікований теоретичний апарат, що описує процес обфускації, як додавання в програму надлишкового функціонала, що і є основним змістом цього процесу.

У зв'язку із цим під функціональною властивістю підпрограми розуміються описувач логіки, що виконує дана підпрограма. Для функціональних властивостей справедливо наступне:

$$\forall p_1, p_2 \in П(f_{p_1} = f_{p_2} \Rightarrow (p_1 \in \pi \Leftrightarrow p_2 \in \pi)), \qquad (3.2)$$

де:

– π – функціональна властивість підпрограми;

– $П$ – множина всіх підпрограм;

– p_1, p_2 – підпрограми, що належать множині $П$;

– f_{p_1}, f_{p_2} – функції, що обчислюються відповідно підпрограмами p_1, p_2.

Уведемо лівосторонню операцію «*», що означає конкатенацію функціональних властивостей. Запис $\pi_1 * \pi_2$ означає, що підпрограма з функціональною властивістю π_2 виконується відразу ж за підпрограмою з функціональною властивістю π_1. Таким чином, деяку підпрограму, що володіє функціональною властивістю $\pi_{исх}$, можна представити в такий спосіб:

$$\pi_{исх} = \pi_1 * \pi_2 * ... * \pi_n \tag{3.3}$$

Після застосування перетворень, що заплутують, одержимо наступне:

$$\pi_{исх} = \pi_0 * \nu_0 * \pi_1 * \nu_1 * \pi_2 * \nu_2 * ... * \pi_n * \nu_n \tag{3.4}$$

$\nu_0, \nu_1, \nu_2, ..., \nu_n$ – додані функціональні властивості.

$$\pi_0 = e,$$

де e – одинична функціональна властивість, тобто функціональна властивість підпрограми, що не впливає на вхідні й вихідні дані.

Помітимо, що якщо $\nu_0, \nu_1, \nu_2, ..., \nu_n$ з попередньої формули рівняються *e,* та ця рівність буде зводитися до системи рівностей наступного виду:

$$\begin{cases} \pi_0 = \nu_0 = e \\ \pi_1 = \pi_1 * e \\ \pi_2 = \pi_2 * e \\ ..... \\ \pi_n = \pi_n * e \end{cases} \tag{3.5}$$

Припустимо тепер, що:

$$\pi = \pi_0 * \nu_0 * \pi_1 * \nu_1 * \pi_2 * \nu_2 * ... * \pi_n * \nu_n \neq \pi_{исх} \tag{3.6}$$

У цьому випадку незвідність нерівності (3.6) до системи (3.5) досить нескладно забезпечити. Доведемо наступне твердження.

Завдання визначення істотності функціональної властивості $\pi_i(\nu_i)$ в рівності (3.6) *NP*-повні.

Для того, щоб перевірити істотність деякої функціональної властивості (тобто вплив наявності їх на результат роботи програми), необхідно виключити

цю функціональну властивість із послідовності (3.6) і перевірити результати виконання підпрограми на всіх вхідних наборах. Тобто, по суті справи, виконати булевську формулу наступного виду:

$$\bigcup_i (X_i \cdot \overline{Y_i}), \tag{3.7}$$

де X – вихідні дані, отримані до виключення функціональної властивості, Y – вихідні дані, отримані після виключення функціональної властивості. Якщо результат формули (3.7) не буде нульовим, то виключена функціональна властивість буде істотним. Очевидно, що завдання визначення істотності функціональної властивості зводиться до завдання виконуваності булевської формули, а, отже, лежить у класі *NP*.

Щоб перевірити виконуваність булевської формули необхідно перевірити її значення деяка кількість разів. Тобто, по суті справи, необхідно перевірити значення істотних складових булевської формули. Таким чином, можна затверджувати, що завдання виконуваності булевської формули зводиться до завдання визначення істотності її складових (тобто перевірити істотність функціональних властивостей у нашім контексті визначень).

З вищесказаного треба, що завдання визначення істотності функціональних властивостей у рівності (3.6) *NP*-повна.

Додавання глобального (стосовно досліджуваної підпрограми) контексту забезпечить набагато більше високу стійкість заплутаного коду стосовно існуючих алгоритмів деобфускації. Проходження ряду розроблених на основі теорії компіляторів рекомендацій з побудови перетворень, що заплутують, дозволить істотно знизити ймовірність побудови деобфускатора, що працює за поліноміальний час. Ці правила наведені нижче.

1. Маскування графа потоку керування підпрограми (зокрема, адресу, на який передається керування з інструкції розгалуження, розраховується динамічно, виходячи з хеш-суми підпрограми).

2. Граф потоку керування підпрограми повинен бути що неприводиться. Відомо, що аналіз графів потоку, що приводяться, керування набагато простіше,

ніж що не приводяться. Більше того, ряд алгоритмів деобфускації застосуємо тільки стосовно приводяться графам, що, потоку керування.

3. Визначення «фальшивих» змінних не повинні знищуватися наприкінці підпрограми. Під «фальшивими» змінними розуміються змінні, додані в процесі обфускації для ускладнення розуміння логіки роботи заплутаного коду.

4. Визначення «фальшивих» змінних не повинні знищуватися до того, поки ці змінні хоч раз не були використані.

5. Множині комірок пам'яті для вихідного й доданого в процесі обфускації контекстів, з якими працює заплутана програма, повинні відповідати системі (3.8).

$$\begin{cases} V_{REAL}^{R} \bigcap V_{FAKE}^{R} \neq \Theta \\ V_{REAL}^{W} \bigcap V_{FAKE}^{W} \neq \Theta \end{cases}, \tag{3.8}$$

де:

– $V_{REAL}^{R}, V_{FAKE}^{R}$ – множині всіх комірок пам'яті, з яких роблять читання відповідно вихідні й додані в результаті застосування перетворень, що заплутують, інструкції;

– $V_{REAL}^{W}, V_{FAKE}^{W}$ – множині всіх комірок пам'яті, у які роблять запис відповідно вихідні й додані в результаті застосування перетворень, що заплутують, інструкції;

– Θ – порожня множина.

6. Визначення глобальних «фальшивих» змінних повинні знищуватися тоді й тільки тоді, коли дана глобальна змінна використовується в якості одного з параметрів у визначенні, що знищує її попереднє значення.

7. «Мертвий» код (код, на який не передбачена передача керування в процесі нормального функціонування програми) не повинен сильно відрізнятися від реально виконуваного коду. Всі правила, наведені вище, повинні ставиться, у тому числі, і до нього.

3.2 Розробка структурної схеми

Опишемо розроблені методики ускладнення аналізу алгоритмів роботи вихідного коду. Дані методики використовують глобальний, стосовно заплутується блоку, що, контекст для забезпечення стійкості до автоматичних алгоритмів деобфускації. Процес заплутування виглядає в такий спосіб:

1. Аналіз вхідних даних.

2. Додавання «фальшивого» локального контексту.

3. Генерація «сміттєвого» коду (коду, доданого в процесі ускладнення аналізу алгоритмів роботи вихідного коду).

4. Розведення інструкцій розгалуження.

5. Маскування графа потоку керування (динамічне обчислення адрес розгалужень).

6. Розбивка отриманих базових блоків на набір функцій.

7.Генерація машинних інструкцій з використанням генератора поліморфного коду.

Даний процес зображений на структурній схемі, яка наведена на рисунку 3.1. Розглянемо більш докладно кожний з пунктів.

1. Методика аналізу вхідних даних має на увазі під собою розбивку на базові блоки, аналіз псевдонімів, переклад у проміжне подання, розпізнавання неподільних блоків пам'яті (структур, масивів і т.д.), виділення операція пам'ять-пам'ять (3.часткова оптимізація). Алгоритми розроблені для архітектур x86 і AMD64, але можуть бути легко модифіковані для використання на інших апаратних платформах. Розбивка на базові блоки виробляється за допомогою емулюючого дизасемблера. Під час емуляції збирається інформація про можливі значення змінних, котра придасться на наступних етапах. Стік споконвічно представляється якимось пулом пам'яті, з осередками якого працює емулятор. Стековий пул має свою адресацію. Як точка відліку приймається покажчик на адресу повернення. Аналіз псевдонімів теж виробляється на етапі емуляції.

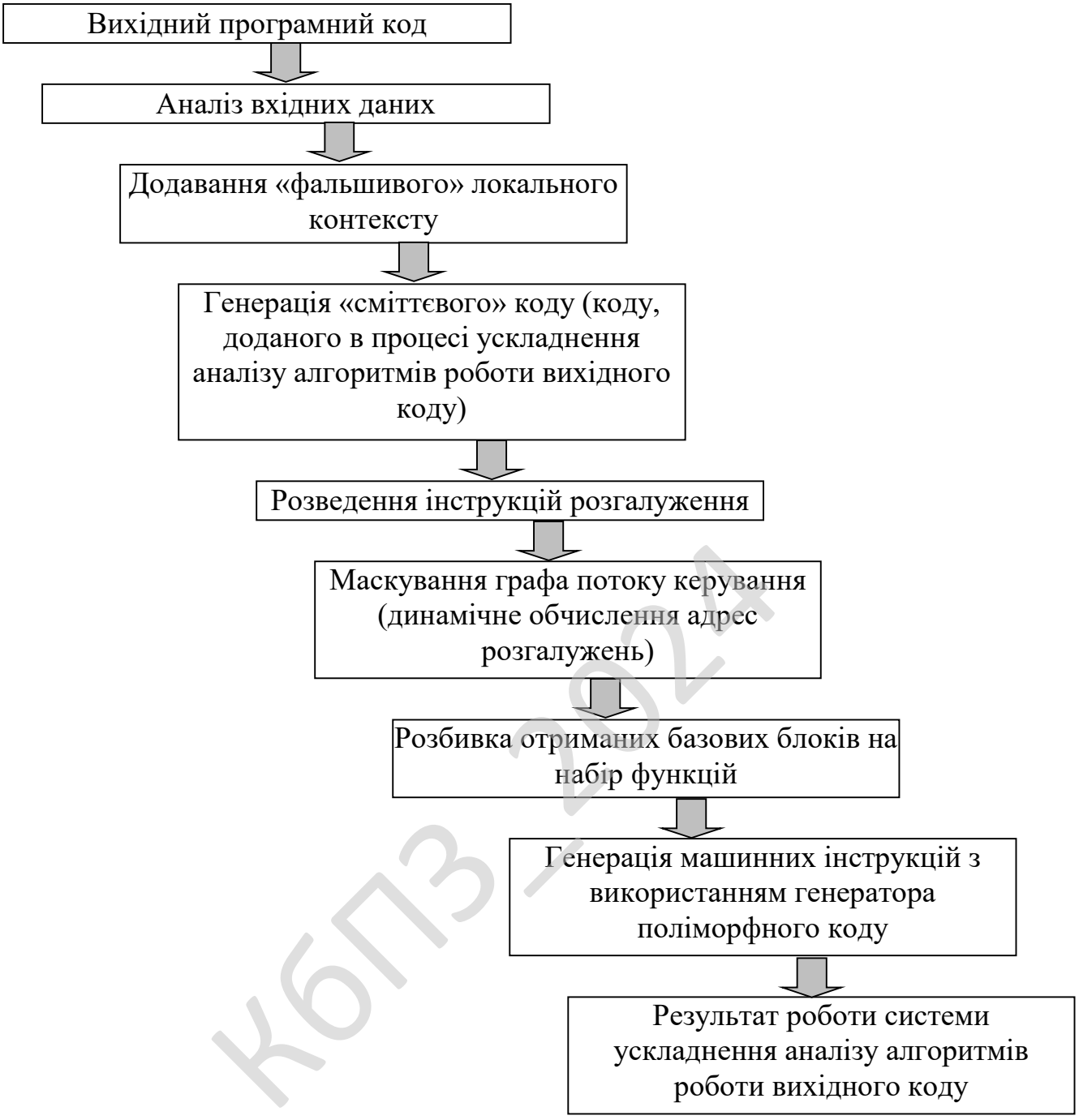

Рисунок 3.1 – Структурна схема системи

Переклад у проміжний код виробляється відразу після аналізу псевдонімів. Основним завданням даного етапу є необхідність рятування від прапорів і стека. Абстрагування від прапорів виробляється або безпосереднім порівнянням операндів (за допомогою інструкції проміжного коду if) – або

введенням додаткової змінної, котра згодом буде використана в інструкції if. Типи операндів можуть бути наступними:

– *UNKNOWN* – тип, про який неможливо одержати інформацію;

– *LOC_PTR* – покажчик на локальну пам'ять функції;

– *GLOB_PTR* – покажчик на пам'ять, глобальну стосовно функції;

– *ARG_PTR* – покажчик на аргумент;

– *UNKNOWN_PTR* – покажчик на невідому область пам'яті;

– *CONST* – константне значення.

Для аналізу типу операнду необхідно передбачити ряд правил, що регламентують пріоритетність типів. Наприклад, якщо якої-небудь точки підпрограми досягають два визначення типу якого-небудь операнду, те потрібно виділити те з визначень, що найбільше правильно описує операнд із дотриманням принципу консервативності. У ході виконання бакалаврського проекту ці правила були розроблені.

У процесі аналізу кожна змінна представляється наступною трійкою значень. (MemId, Offset, Size), де MemId – ідентифікаційний номер області пам'яті, Offset – зсув початку абстрактного осередку щодо точки відліку, а Size – розмір осередку. Зсув у цьому випадку зручно вважати від адреси повернення, що має нульовий зсув. Для зручності аналізу доступу до пам'яті введемо поняття множині значень для кожної змінної (vs). Множині значень зручно записувати у вигляді стислих інтервальних конгруенцій (RIC) кожна з яких являє собою кортеж з 4-х значенні (a, b, c, d) і трактується в такий спосіб: $a \times [b,c] + d$, що описує множина цілих значень $\{aZ + d \mid Z \in [b,c]\}$. Нижче наведений ряд операцій, які можна застосовувати стосовно множин значень:

– $(vs_1 \subseteq vs_2)$: повертає TRUE, якщо множина значень vs_1 є підмножиною vs_2.

– $(vs_1 \cap vs_2)$: повертає перетинання множин значень vs_1 і vs_2.

– $(vs_1 \cup vs_2)$: повертає об'єднання множин значень vs_1 і vs_2.

– $(vs_1 \nabla vs_2)$: повертає множина значень, отримана за допомогою розширення множині значень vs_1 по відношенню к. vs_2 Тобто, якщо vs_1=(10,4[0,1]), а vs_2=(10,4[0,2]), те $(vs_1 \nabla vs_2)$= (10,4[0,∞]).

– $(vs \oplus c)$: повертає множина значень, отримана в результаті «підстроювання» значень vs із константою c. Тобто, якщо vs=(4,4[0,2]+4), а c=12, те $(vs \oplus c)$=(16,4[0,2]+16). Аналогічним образом може бути заданий цілий ряд арифметичних операцій.

– $*(vs,s)$: повертає пари множин (3.*F, P*). *F* являє собою множина «повністю доступних» абстрактних комірок пам'яті (тобто комірок пам'яті розміром *s* і їхня стартова адреса перебуває в *vs*). *P* являє собою множина «частково доступних» комірок пам'яті. Тобто комірок пам'яті в яких стартова адреса лежи в *vs*, а розмір не дорівнює *s*, а також осередків, які лежать в *vs,* але їхня стартова адреса й розмір не задовольняють вимогам до множині *F*.

– *RemoveLowerBounds*(vs): повертає множина значень, отримана за допомогою установки нижньої границі кожного компонента RIC рівного -∞. Наприклад, якщо *vs*=([0, 100],[0, 200]), те *RemoveLowerBounds*(vs)= ([-∞, 100],[-?, 200]).

– *RemoveUpperBounds*(vs): аналогічно *RemoveLowerBounds,* але встановлює верхню границю кожного компонента рівної ∞.

Якщо відбувається виклик зовнішньої функції, що у свою чергу теж аналізується, то можливі дві ситуації:

а) зовнішня функція вже проаналізована з обліком того, що її параметри невідомі;

б) зовнішня функція ще не проаналізована.

У першому випадку необхідно зробити корекцію, що полягає в повторному аналізі. Щоб мінімізувати кількість таких проходів або зовсім уникнути їх, необхідно, насамперед, одержати інформацію про ієрархію виклику таких функцій, а на основі цієї інформації становити порядок аналізу функцій. Міжпроцедурний аналіз можна вдосконалити за допомогою «псевдокопіювання»

функції. Припустимо, що в блоці, який захищається, функція викликається із двох різних місць. У цьому випадку доцільно замінити викликувану функцію на дві однакових і вважати, що кожна з них викликається тільки з одного місця.

Розпізнавання неподільних блоків пам'яті виробляється на основі інформації про множині значень абстрактних комірок пам'яті. Консервативним рішенням буде перебір всіх множин значень для кожного з операндів і обчислення операції $*(vs,s)$ для кожного з *RIC*. Отримані множині *F, P* і будуть шуканими масивами абстрактних комірок пам'яті, які у свою чергу є неподільними блоками пам'яті.

Далі проводиться додатковий аналіз вхідних даних, що містить у собі наступні пункти:

а) виділення множин «мертвих» локальних змінних *D*. Тобто множина змінних, які після проходження певної точки підпрограми більше не використовуються;

б) виділення множин змінних, що мають прості типи (множина *C*);

в) виділення множин змінних, що мають складні типи (*CO*);

г) виділення абстрактних комірок пам'яті, що є елементами неподільних блоків пам'яті, які вже були використані в підпрограмі, як єдине ціле й після проходження певної точки графа потоку керування, як єдине ціле, у підпрограмі не використовуються (множина *CA*);

д) виділення абстрактних комірок пам'яті, що є елементами неподільних блоків пам'яті, які використовуються порізно в межах одного або декількох базових блоків (множина *CB*). Такі змінні можна переносити в «фальшивий» локальний контекст тільки в межах границь цих базових блоків. У наслідку значення цих комірок пам'яті повинні бути відновлені;

е) Виділення множин абстрактних комірок пам'яті зі свідомо відомими значеннями (множина *V*).

2. Додавання «фальшивого» локального контексту. На даному етапі додається «фальшивий» локальний контекст, що буде згодом використовуватися

для створення «сміттєвого» коду й забезпечення поліморфізму. Локальний контекст перекомпоновується з метою ускладнення аналізу даних. Перекомпонування означає перенос частини локальних змінних в область пам'яті «фальшивого» контексту. Область пам'яті, звідки цей перенос відбувся, використовується для роботи «сміттєвого» коду. Переносити можна змінні з множин *C, CO, CA* (після проходження певної точки) і *CB* (у межах базового блоку).

У підпрограму може бути переданий один або кілька покажчиків на «фальшиві» буфери пам'яті. Для алгоритму заплутування «фальшиві» буфери пам'яті складаються з абстрактних комірок пам'яті, з якими згодом ведеться робота. Ці «фальшиві» буфери пам'яті називаються «фальшивим» глобальним контекстом.

3. Генерація «сміттєвого» коду. На основі наведених вище теоретичних викладень генерується «фальшивий» код, що працює як з локальним, так і із глобальним контекстом. Забезпечується поліморфізм коду за рахунок заміни частини вихідних інструкцій на набір інших інструкцій, що виконують у своїй сукупності ті ж дії (у тому числі й із глобальним контекстом). У бакалаврському проекті наведений докладний приклад алгоритму генерації «сміттєвого» коду. Алгоритм має на увазі додавання коду відповідно до розроблених рекомендацій по побудові перетворень, що заплутують. Для генерації «сміттєвих» інструкцій активно використовується «фальшивий» глобальний і «фальшивий» локальний контекст.

4. Розведення інструкцій переходу. Під розведенням інструкцій переходу маються на увазі різні схеми, що дозволяють замінити одну інструкцію переходу декількома.

Автоматичний алгоритм генерації подібних схем розроблений, реалізований і докладно описаний у бакалаврському проекті. Доведено збіжність цього алгоритму й коректність його роботи. Для приховання деяких констант використовується метод, заснований на тотожностях наступного виду:

$$a > C \equiv a\,op\,k > C\,op\,k \tag{3.9}$$

Операція *op* означає будь-яку операцію, що задовольняє тотожності (3.9).

5. Маскування графа потоку керування виробляється при перекладі вже заплутаного коду із проміжного подання в машинне. Всі інструкції розгалуження робляться регістра, що розраховуються в залежності від значення, прапорів, хеш-суми додатку й ще ряду значень.

6. Розбивка базових блоків на функції виробляється як на етапі проміжного подання інструкцій, так і на етапі платформозалежного ускладнення аналізу алгоритмів роботи вихідного коду (ускладнення аналізу алгоритмів роботи вихідного коду на рівні машинного коду цільової платформи). Кожної інструкції в базовому блоці ставиться у відповідність рівень вкладеності. У міру наближення до «центра» базового блоку рівень вкладеності збільшується. Таким чином, виходить, що перша й остання інструкції базового блоку мають рівень вкладеності рівним 0. Максимальний рівень вкладеності є задається параметром, що.

7. Розроблені методики ускладнення аналізу алгоритмів роботи вихідного коду мають на увазі використання для генерації машинного коду генератора поліморфного коду. Генератор машинного коду називається генератором поліморфного коду, якщо для однакових вхідних даних він створює різні машинні подання, що володіють однаковими функціональними властивостями. При цьому жодна з доданих підпрограм не має функціональну властивість *e*.

$$instr \to p\ : p, instr \in P^{<k>},\ P^{<k>} \subset \Pi_{\pi} \tag{3.10}$$

Формула (3.10) дає формальне визначення дій, виконуваних генератором поліморфного коду. *instr* – машинна інструкція, *p* – підпрограма, що володіє тим же функціональною властивістю, що й інструкція *instr*. Вибір підпрограми здійснюється з кінцевої множині *P* потужністю *k*. Множина *P* є підмножиною всіх підпрограм з функціональною властивістю π. Основним завданням генератора поліморфного коду варто вважати завдання приховання сигнатур. Якщо найдеться алгоритм, що зможе встановити взаємооднозначну відповідність

між інструкцією *instr* і підпрограмою *p*, то згодом можна здійснити деобфускацію за допомогою алгоритмів сигнатурного пошуку. Для того щоб перешкодити сигнатурному пошуку, необхідно доповнити умову (3.10) у такий спосіб:

$$instr_i \to p_i : p_i, instr_i \in P_i^{<k>}, P_i^{<k>} \subset \Pi_{\pi_i}$$

$$instr_{i+1} \to p_{i+1} : p_{i+1}, instr_{i+1} \in P_{i+1}{}^{<k>}, P_{i+1}{}^{<k>} \subset \Pi_{\pi_{i+1}}$$

…….....(3.11)

$$instr_{i+m} \to p_{i+m} : p_{i+m}, instr_{i+m} \in P_{i+m}{}^{<k>}, P_{i+m}{}^{<k>} \subset \Pi_{\pi_{i+m}}$$

$$\{instr_i, instr_{i+1}, ..., instr_{i+m}\} \to p : \pi_p \subset \pi_i * \pi_{i+1} * ... * \pi_{i+m}$$

$$p \neq W(p_i, p_{i+1}, ..., p_{i+m})$$

Вихідні інструкції $instr_i, instr_{i+1}, instr_{i+m}$ випливають один за одним. На етапі генерації поліморфного коду вони відображаються відповідно в підпрограми p_i, p_{i+1}, p_{i+m}. Далі ці підпрограми поєднуються в одну підпрограму, але так, щоб отримана підпрограма p не являла собою конкатенацію елементів p_i, p_{i+1}, $p_{i+m} - W(p_i, p_{i+1}, ..., p_{i+m})$. Ця технологія була названа технологією перетинання поліморфних інструкцій. Вона полягає в тім, що частина інструкцій підпрограми p_i переноситься в тіло підпрограми p_{i+1} й навпаки.

Для генерації поліморфних еквівалентів інструкцій побітового додавання, множення, додавання по модулі 2, заперечення й ряду інших використовуються закони булевої алгебри – закон де Моргана й закон поглинання.

У генераторі поліморфного коду використовуються системи повних функцій І-НІ й АБО-НІ. Також використовуються наступні тотожності:

$$\begin{aligned} &\bar{x} = x \oplus -1 \\ &x_1 \vee x_2 = x_1 + x_2 - x_1 \wedge x_2 \\ &x_1 \vee x_2 = x_1 + x_2 - 2 \cdot (x_1 \wedge x_2) \end{aligned} \qquad (3.12)$$

Генератор поліморфного коду сконструйований таким чином, що інструкції, які він створює, можуть або міняти прапори довільним образом – або строго відповідати специфікації.

Розроблені алгоритми й методи дозволяють не тільки робити заплутування, але й ефективно впроваджувати код у тіло додатку.

Опишемо практичну реалізацію розроблених методик, алгоритмів і підходів, а також зроблений аналіз захищеності й швидкості роботи додатку, захищеного за допомогою даних методик.

У процесі розробки виникла необхідність відділення платформозалежних і платформонезалежних (що працюють на рівні проміжного коду) компонентів. Обмін між проміжним компонентом і платформозалежними компонентами стандартизований. Таким чином, ми можемо ввести досить велику кількість платформозалежних компонент, якщо це буде необхідно. Роль платформозалежних компонент – переклад з машинного подання в проміжне, а також генерація машинного подання із проміжного. У платформозалежних компонентах закладені відповідні платформозалежні методи ускладнення аналізу алгоритмів роботи вихідного коду, зокрема, описаний вище генератор поліморфного коду.

Розраховано максимальні коефіцієнти зі швидкості роботи заплутаного коду в порівнянні з вихідним і збільшення обсягу коду в порівнянні з вихідним.

$$K_{speed} = max_i(C_{speed} \cdot 4 \cdot m \cdot FPT \cdot IterNum_i) \quad (3.13)$$

$$K_{value} = C_{value} \cdot 4 \cdot m \cdot k \cdot FPT \quad (3.14)$$

Коефіцієнти C_{speed} й C_{value} – коефіцієнти, що залежать від платформи, на якій буде виконуватися отриманий код і від генератора машинного коду із проміжного. *IterNum* у формулі (3.13) – це число входжень в і-*й* базовий блок або число ітерацій у циклі. *FPT* – число «сміттєвих» інструкцій на одне дійсну, *m* – число констант, згенерованих для порівняння. У результаті проведення 200 вимірів на різних підпрограмах було відзначено, що швидкість роботи заплутаного коду зменшується в порівнянні з вихідною не більше ніж на 2 порядки. У свою чергу розмір заплутаного коду збільшується в порівнянні з вихідним не більше ніж на 2 порядки.

У результаті оцінки якості перетворень, що заплутують, здійснюваними розробленими методиками, по методу, описаному в роботі Гайсаряна, Чернова, Белеванцева й ін., був зроблений вивід, що якість заплутування за допомогою даних методик перевершує якість ускладнення аналізу алгоритмів роботи вихідного коду, забезпечуване технологіями віртуалізації на 30%.

Існуючі методики оцінки якості перетворень, що заплутують, не припускають оцінки потенціалу вихідних даних до заплутування. Виділимо ряд характеристик коду, що заплутується – частка часу знаходження в частині коду, що заплутується стосовно часу роботи всієї програми (*vc*), частка інструкцій переходу в зовнішнє середовище стосовно загальної кількості інструкцій (*tc*) і відсоток рідких інструкцій з відношення до загальної кількості інструкцій (*rc*). Якщо *vc* деякої функції перевищує 20%, то дану функцію захищати не рекомендується. Якщо *tc* перевищує 30%, то дану функцію теж не рекомендується захищати так само, як якщо *rc* перевищує 50%. Якщо функція задовольняє вищеописаною вимогою, то якість перетворень, що заплутують, відповідає високому рівню по Коллбергу.

3.3 Розробка функціональної схеми

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

Перетворення, що заплутують, можна розділити на кілька груп залежно від того, на трансформацію якої з компонентів програми вони націлені:

– Перетворення потоку керування програми, які змінюють структуру її графа потоку керування, такі як розгорнення циклів, виділення фрагментів коду в процедури, і інші. Дана стаття присвячена аналізу саме цього класу перетворень, що заплутують.

– Превентивні перетворення, націлені проти певних методів декомпіляції програм або помилки, що використовують, у певних інструментальних засобах декомпіляції.

– Перетворення форматування, які змінюють тільки зовнішній вигляд програми. До цієї групи відносяться перетворення, що видаляють коментарі, відступи в тексті програми або ідентифікатори, що перейменовують.

– Перетворення структур даних, що змінюють структури даних, з якими працює програма. До цієї групи відносяться, наприклад, перетворення, що змінює ієрархію спадкування класів у програмі, або перетворення, що поєднує скалярні змінні одного типу в масив. У даній роботі ми не будемо розглядати перетворення, що заплутують, цього типу.

Перетворення потоку керування

Перетворення потоку керування змінюють граф потоку керування однієї функції. Вони можуть приводити до створення в програмі нових функцій. Коротка характеристика методів наведена нижче.

Відкрита вставка функцій (function inlining) полягає в тому, що тіло функції підставляється в точку виклику функції. Дане перетворення є стандартним для оптимізуючих компіляторів. Це перетворення однобічне, тобто по перетвореній програмі автоматично відновити вставлені функції неможливо. Ми не будемо розглядати докладно пряму вставку функцій і її ефект на заплутування й розплутування програм.

Винос групи операторів (function outlining). Дане перетворення є зворотним до попереднього й добре доповнює його. Деяка група операторів вихідної програми виділяється в окрему функцію. При необхідності створюються формальні параметри. Перетворення може бути легко звернено компілятором, що (як було сказано вище) може підставляти тіла функцій у точки їхнього виклику.

Відзначимо, що виділення операторів в окрему функцію є складним для заплутувача перетворенням. Заплутувач повинен провести глибокий аналіз графа

потоку керування й потоку даних з урахуванням покажчиків, щоб бути впевненим, що перетворення не порушить роботу програми.

Непрозорі предикати (opaque predicates). Основною проблемою при проектуванні перетворень, що заплутують, графа потоку керування є те, як зробити їх не тільки дешевими, але й стійкими. Для забезпечення стійкості багато перетворень ґрунтуються на введенні непрозорих змінних і предикатів. Сила таких перетворень залежить від складності аналізу непрозорих предикатів і змінних.

Змінна v є непрозорої, якщо існує властивість π щодо цієї змінної, котре апріорі відомо в момент заплутування програми, але трудновстановлюємо після того, як заплутування завершене. Аналогічно, предикат P називається непрозорим, якщо його значення відомо в момент заплутування програми, але трудновстановлюємо після того, як заплутування завершене.

Непрозорі предикати можуть бути трьох видів: P^F – предикат, що завжди має значення "неправда", P^T – предикат, що завжди має значення "істина", і $P^?$ – предикат, що може приймати обоє значення, і в момент заплутування поточне значення предиката відомо.

У роботах [3], [7], [23] розроблені методи побудови непрозорих предикатів і змінних, засновані на "вбудовуванні" у програму розрахунково складних завдань. Деякі можливі способи введення непрозорих предикатів і непрозорих виражень коротенько перераховані нижче.

– Використання різних способів доступу до елементи масиву [23]. Наприклад, у програмі може бути створений масив, що ініціалізується заздалегідь відомими значеннями, далі в програму додаються трохи змінних, у яких зберігаються індекси елементів цього масиву.

– Використання покажчиків на спеціально створювані динамічні структури [7]. У цьому підході в програму додаються операції по створенню посилальних структур даних (списків, дерев), і додаються операції над

покажчиками на ці структури, підібрані таким чином, щоб зберігалися деякі інваріанти, які й використовуються як непрозорі предикати.

– Конструювання булевських виражень спеціального виду [3].

– Побудова складних булевських виражень за допомогою еквівалентних перетворень із формули true. Використання комбінаторних тотожностей.

Внесення недосяжного коду (adding unreachable code). Якщо в програму внесені непрозорі предикати видів P^F або P^T, вітки умови, що відповідають умові "істина" у першому випадку й умові "неправда" у другому випадку, ніколи не будуть виконуватися. Фрагмент програми, що ніколи не виконується, називається недосяжним кодом. Ці вітки можуть бути заповнені довільними обчисленнями, які можуть бути схожі на дійсно виконуваний код, наприклад, зібрані із фрагментів тої ж самої функції. Оскільки недосяжний код ніколи не виконується, дане перетворення впливає тільки на розмір заплутаної програми, але не на швидкість її виконання. Загальне завдання виявлення недосяжного коду, як відомо, алгоритмічно нерозв'язна. Це значить, що для виявлення недосяжного коду повинні застосовуватися різні евристичні методи, наприклад, засновані на статистичному аналізі програми.

Внесення мертвого коду (adding dead code). На відміну від недосяжного коду, мертвий код у програмі виконується, але його виконання ніяк не впливає на результат роботи програми. При внесенні мертвого коду заплутувач повинен бути впевнений, що вставляється фрагмент, що, не може впливати на код, що обчислює значення функції. Це практично виходить, що мертвий код не може мати побічного ефекту, навіть у вигляді модифікації глобальних змінних, не може змінювати оточення працюючої програми, не може виконувати ніяких операцій, які можуть викликати виключення в роботі програми.

Внесення надлишкового коду (adding redundant code). Надлишковий код, на відміну від мертвого коду виконується, і результат його виконання використовується надалі в програмі, але такий код можна спростити або зовсім видалити, тому що обчислюється або константне значення, або значення, уже

обчислене раніше. Для внесення надлишкового коду можна використовувати алгебраїчні перетворення виражень вихідної програми або введення в програму математичних тотожностей.

Подібні алгебраїчні перетворення обмежені цілими значеннями, тому що при виконанні операцій із плаваючою точкою виникає проблема нагромадження помилки обчислень. З іншого боку, при операціях із цілими значеннями виникає проблема переповнення. Як часткове рішення завдання можна виконувати множення в 64-бітних цілих числах.

Перетворення графа потоку, що зводиться, керування до незвідного (transforming reducible to non-reducible flow graph). Коли цільова мова (байт-код або машинна мова) більше виразна, чим вихідний, можна використовувати перетворення, "суперечні" структурі вихідної мови. У результаті таких перетворень виходять послідовності інструкцій цільової мови, не відповідні ні однієї з конструкцій вихідної мови.

Наприклад, байт-код віртуальної машини Java містить інструкцію goto, у той час як у мові Java оператор goto відсутній. Графи потоку керування програм мовою Java виявляються зводи_ завжди, у той час як у байт-коді можуть бути представлені й незвідні графи.

Можна запропонувати перетворення, що заплутує, що трансформує графи, що зводяться, потоку керування функцій у байт-коді, одержуваних у результаті компіляції Java-програм, у незвідні графи. Наприклад, таке перетворення може полягати в трансформації структурного циклу в цикл із множинними заголовками з використанням непрозорих предикатів. З одного боку, декомпілятор може спробувати виконати зворотне перетворення, усуваючи незвідні області в графі, дублюючи вершини або вводячи нові булевські змінні. З іншого боку, розплутувач може за допомогою статичних або статистичних методів аналізу визначити значення непрозорих предикатів, використаних при заплутуванні, і усунути ніколи що не виконуються переходи. Однак, якщо здогад про значення предиката виявиться невірна, у результаті вийде неправильна програма.

Усунення бібліотечних викликів (eliminating library calls). Більшість програм мовою Java істотно використовують стандартні бібліотеки. Оскільки семантика бібліотечних функцій добре відома, такі виклики можуть дати корисну інформацію при зворотній інженерії програм. Проблема збільшується ще й тим, що посилання на класи бібліотеки Java завжди є іменами, і ці імена не можуть бути перекручені.

У багатьох випадках можна обійти цю обставину, просто використовуючи в програмі власні версії стандартних бібліотек. Таке перетворення не змінить істотно час виконання програми, зате значно збільшить її розмір і може зробити її нестерпною.

Для програм на традиційних мовах ця проблема стоїть менш гостро, тому що стандартні бібліотеки, як правило, можуть бути скомпоновані статично разом із самою програмою. У цьому випадку програма не містить ніяких імен функцій зі стандартної бібліотеки.

Переплетення функції **(**function interleaving). Ідея цього перетворення, що заплутує, у тому, що дві або більше функції поєднуються в одну функцію. Списки параметрів вихідних функцій поєднуються, і до них додається ще один параметр, що дозволяє визначити, яка функція в дійсності виконується.

Клонування функцій (function cloning). При зворотній інженерії функцій у першу чергу вивчається сигнатура функції, а також те, як ця функція використовується, у яких місцях програми, з якими параметрами й у якому оточенні викликається. Аналіз контексту використання функції можна утруднити, якщо кожний виклик деякої функції буде виглядати як виклик якийсь інший, щораз нової функції. Може бути створено кілька клонів функції, і до кожного із клонів буде застосований різний набір перетворень, що заплутують.

Розгорнення циклів (loop unrolling). Розгорнення циклів застосовується в оптимізуючих компіляторах для прискорення роботи циклів або їх розпаралелювання. Розвертання циклів полягає в тому, що тіло циклу розмножується два або більше рази, умова виходу із циклу й оператор

збільшення лічильника відповідним чином модифікуються. Якщо кількість повторень циклу відомо в момент компіляції, цикл може бути розгорнутий повністю.

Розкладання циклів (loop fission). Розкладання циклів полягає в тому, що цикл зі складним тілом розбивається на кілька окремих циклів із простими тілами й з тим же простором ітерування.

Реструктуризація графа потоку керування [24]. Структура графа потоку керування, наявність у графі потоку керування характерних шаблонів для циклів, умовних операторів і т.д. подає коштовну інформацію при аналізі програми. Наприклад, по повторюваних конструкціях графа потоку керування можна легко встановити, що над функцією було виконане перетворення розгорнення циклів, а далі можна запустити спеціальні інструменти, які проаналізують розгорнуті ітерації циклу для виділення індуктивних змінних і згортки циклу. Як міра протидії може бути застосоване таке перетворення графа потоку керування, що приводить графа до однорідного ("плоского") виду. Оператори передачі керування на наступні за ними базові блоки, розташовані на кінцях базових блоків, заміняються на оператори передачі керування на спеціально створений базовий блок диспетчера, що по попередньому базовому блоці й керуючим змінним обчислює наступний блок і передає на нього керування. Технічно це може бути зроблено перенумеруванням всіх базових блоків і введенням нової змінної, наприклад state, що містить номер поточного базового блоку, що виконується. Заплутана функція замість операторів if, for і т.д. буде містити оператор switch, розташований усередині нескінченного циклу.

Локалізація змінних у базовому блоці [3]. Це перетворення локалізує використання змінних одним базовим блоком. Для кожного базового блоку, що заплутується, функції створюється свій набір змінних. Всі використання локальних і глобальному змінних у вихідному базовому блоці заміняються на використання відповідних нових змінних. Щоб забезпечити правильну роботу програми між базовими блоками вставляються так звані сполучні (connective)

базові блоки, завдання яких скопіювати вихідні змінні попереднього базового блоку у вхідні змінні наступного базового блоку.

Застосування такого заплутуючого перетворення приводить до появи у функції великого числа нових змінних, які, однак, використовуються тільки в одним-двох базових блоках, що заплутує людину, що аналізує програму.

При реалізації цього перетворення, що заплутує, виникає необхідність точного аналізу покажчиків і контекстно-залежного міжпроцедурного аналізу. У противному випадку не можна гарантувати, що запис по якому-небудь покажчику або виклик функції не модифікують справжню змінну, а не поточну робочу копію.

Розширення області дії змінних. Дане перетворення за змістом обратне попередньому. Це перетворення намагається збільшити час життя змінних настільки, наскільки можна. Наприклад, виносячи блокову змінну на рівень функції або виносячи локальну змінну на статичний рівень, розширюється область дії змінної й ускладнюється аналіз програми. Тут використовується те, що глобальні методи аналізу (тобто, методи, що працюють над однією функцією в цілому) добре обробляють локальні змінні, але для роботи зі статичними змінними потрібні більше складні методи міжпроцедурного аналізу.

Для подальшого заплутування можна об'єднати трохи таких статичних змінних в одну змінну, якщо точно відомо, що змінні не можуть використовуватися одночасно. Очевидно, що перетворення може застосовуватися тільки до функцій, які ніколи не викликають один одного безпосередньо або через ланцюжок інших викликів.

Застосування перетворень, що заплутують

Існуючі методи заплутування й інструменти для заплутування програм використовують не єдине перетворення, що заплутує, а деяку їхню комбінацію. У даному розділі ми розглянемо деякі використовувані на практиці методи заплутування.

У роботах Ч. Ванг [23][24] пропонується метод заплутування, і описується його реалізація в інструменті для заплутування програм мовою Delphi або Ci. Запропонований метод заплутування використовує перетворення введення "диспетчера" у заплутується функцію, що. Номер наступного базового блоку обчислюється безпосередньо в самому базовому блоці, що виконується, прямим присвоюванням змінної, котра зберігає номер поточного базового блоку. Для того щоб утруднити статичний аналіз, номери базових блоків містяться в масив, кожний елемент якого індексується декількома різними способами. Таким чином, для статичного простежування порядку виконання базових блоків необхідно провести аналіз покажчиків.

У роботі [3] пропонується метод заплутування, заснований на наступних перетвореннях, що заплутують: кожний базовий блок заплутується функції, що, розбивається на більше дрібні частини (т.зв. piece) і клонується один або кілька разів. У кожному фрагменті базового блоку змінні локалізуються, і для зв'язування базових блоків створюються спеціальні сполучні базові блоки. Далі в кожний фрагмент уводиться мертвий код. Джерелом мертвого коду може бути, наприклад, фрагмент іншого базового блоку тої ж самої функції або фрагмент базового блоку іншої функції. Оскільки кожний фрагмент використовує свій набір змінних, поєднуватися вони можуть безболісно (за умови відсутності в програмі покажчиків і викликів функцій з побічним ефектом). Далі з таких комбінованих фрагментів збирається нова функція, у якій для перемикання між базовими блоками використовується диспетчер. Диспетчер приймає як параметри номер попереднього базового блоку й набір булевських змінних, які використовуються в базових блоках для обчислення умов переходу, і обчислює номер наступного блоку. При цьому наступний блок може вибиратися з декількох еквівалентних блоків, отриманих у результаті клонування. Виражаючи функцію переходу у вигляді булевської формули, можна домогтися того, що завдання статичного аналізу диспетчера буде PSPACE-повна. Робота [3] описує

алгоритм заплутування, але не вказує, чи реалізований цей алгоритм у якій-небудь системі.

Заплутувачі для мови Java, наприклад Zelix KlassMaster [25], як правило, використовують наступне сполучення перетворень: з .class-файлів видаляється вся відладочна інформація, включаючи імена локальних змінних; класи й методи перейменовуються в короткі й семантично неосмислені імена; граф потоку керування заплутується функції, що, перетворюється до незвідного графа, щоб утруднити декомпіляцію назад у мову Java.

Побічним ефектом такого заплутування є істотне прискорення роботи програми. По-перше, видалення відладочної інформації робить class-файли істотно менше, і відповідно їхнє завантаження (особливо по повільних каналах зв'язку) прискорюється. По-друге, різке зменшення довжини імен методів приводить до того, що прискорюється пошук методу по імені, виконуваний щораз при виклику. Як наслідок, заплутування Java-програм часто розглядається як один зі способів їхньої оптимізації.

Перетворення форматування

До перетворень форматування відносяться видалення коментарів, переформатування програми, видалення відладочної інформації, зміна імен ідентифікаторів**.**

Видалення коментарів і переформатування програми застосовні, коли заплутування виконується на рівні вихідного коду програми. Ці перетворення не вимагають тільки лексичного аналізу програми. Хоча видалення коментарів – однобічне перетворення, їхня відсутність не утрудняє сильно зворотну інженерію програми, тому що при зворотній інженерії наявність гарних коментарів до коду програми є скоріше виключенням, чим правилом. При переформатуванні програми вихідне форматування губиться безповоротно, але програма завжди може бути переформатована з використанням якого-небудь інструмента для автоматичного форматування програм (наприклад, indent для програм на Delphi або Ci).

Видалення відладочної інформації застосовно, коли заплутування виконується на рівні об'єктної програми. Видалення відладочної інформації приводить до того, що імена локальних змінних стають невідновні.

Зміна імен локальних змінних вимагає семантичного аналізу (прив'язки імен) у межах однієї функції. Зміна імен всіх змінних і функцій програми крім повної прив'язки імен у кожній одиниці компіляції вимагає аналізу міжмодульних зв'язків. Імена, певні в програмі й не використовувані в зовнішніх бібліотеках, можуть бути змінені довільними, але погодженим у всіх одиницях компіляції образом, у той час як імена бібліотечних змінних і функцій мінятися не можуть. Дане перетворення може заміняти імена на короткі автоматично генеруємі імена (наприклад, всі змінні програми одержать ім'я v<номер> відповідно до їх деякого порядкового номера). З іншого боку, імена змінних можуть бути замінені на довгі, але безглузді (випадкові) ідентифікатори розраховуючи на те, що довгі імена гірше сприймаються людиною.

Методи аналізу програм

У даному розділі ми розглянемо методи, які застосовуються при аналізі програм у компіляторах. Ціль таких методів – виявлення залежностей між компонентами програми, що дає можливість застосувати певні оптимізаційні перетворення, або накладає обмеження на проведені оптимізаційні перетворення.

Методи аналізу програм можуть бути розділені на 4 групи:

– Синтаксичні. До цієї групи відносяться методи, засновані тільки на результатах лексичного, синтаксичного й семантичного аналізу програми.

– Статичні. До цієї групи відносяться методи аналізу потоків керування й даних і методи, засновані на результатах аналізу потоків керування й даних. Статичні методи аналізу працюють із програмою, не використовуючи інформацію про роботу програми на конкретних початкових даних.

– Динамічні. Динамічні методи аналізу програм використовують інформацію, отриману в результаті "спостереження" за роботою програми на конкретних вхідних даних. Помітимо, що самі по собі динамічні методи рідко

застосовуються для аналізу програм, оскільки, як правило, необхідна інформація про поводження програми на різних наборах вхідних даних, що збирається за допомогою статистичних методів аналізу.

– Статистичні. Статистичні методи використовують інформацію, зібрану в результаті значної кількості запусків програми на великій кількості наборів вхідних даних.

Коротка характеристика найважливіших для нас методів аналізу програм наведена нижче.

Методи статичного аналізу

Статичний аналіз аліасів (alias analysis) [11] необхідний у мовах, у яких кілька імен можуть бути використані для доступу до однієї й тій же області пам'яті. У результаті аналізу аліасів кожному операторові, що виконує непрямий запис на згадку або непряме читання з пам'яті, ставиться у відповідність безліч імен змінних, які можуть зачіпатися даною операцією.

Якщо мова допускає аліаси, проведення тією чи іншою мірою аналізу покажчиків необхідно для коректного аналізу потоків даних і для перетворення програм. У випадку доступу до елементів масивів і полям структур ми можемо в найпростішому випадку припускати, що зчитується або модифікується відразу весь масив або вся структура. Для покажчиків або посилань у найпростішому випадку ("консервативний" аналіз) ми можемо виходити із припущення про те, що непряме читання з пам'яті зачіпає весь локальний і глобальний змінні, а непрямий запис на згадку може всі їх модифікувати. Така схема занадто груба й у дійсності блокує глибоку трансформацію практично будь-якої програми.

Відомо, що загальне завдання точного аналізу покажчиків як мінімум NP-важка. У цей час існують методи, що працюють за поліноміальний час, для покажчиків на локальні змінні у випадку нерекурсивних функцій.

Аналіз покажчиків не може бути безпосередньо використаний для заплутування або розплутування програми, але він є ключовим для точного аналізу властивостей програми.

Статичне усунення мертвого коду (dead-code elimination) [19] має на меті виявити в програмі код, що виконується, але не впливає на результат роботи програми.

Статична мінімізація кількості змінних (variable minimization) [19] має на меті зменшити кількість використовуваних у функції локальних змінних за рахунок об'єднання змінних, часи життя значень у які не перетинаються, в одну змінну. Стандартна техніка, що використовується для мінімізації кількості змінних, складається в побудові графа перекриття змінних за допомогою ітераційного рішення рівняння потоку даних і наступному розфарбуванню вершин цього графа в мінімальне або близьке до мінімального кількість кольорів.

Статичне просування констант і копій (constant and copy propagation) [19] полягає в просуванні константних виражень якнайдалі по тексту функції. Якщо вираження використовує тільки значення змінних, які в даній точці програми свідомо містять одне відоме при аналізі програми значення, таке вираження може бути обчислене на етапі аналізу програми. Якщо у вираженні використовується змінна, котра в даній точці програми свідомо є копією якийсь інший змінної, у вираження може бути підставлена вихідна змінна.

Статичний аналіз доменів (domain analysis) є розширенням алгоритму просування констант. Він дозволяє визначити безліч значень, які може приймати дана змінна в даній точці програми, якщо ця безліч не велика.

Статичний слайсинг (slicing) [21] – це побудова "скороченої" програми, з якої вилучений весь код, що не впливає на обчислення заданої змінної в заданій точці (зворотний слайс), але при цьому програма залишається синтаксично й семантично коректної й може бути виконана. Крім описаного вище зворотного слайсинга розроблені алгоритми прямого слайсинга. Прямій слайсинг залишає в програмі тільки ті оператори, які залежать від значення змінної, обчисленої в даній точці програми. Методи слайсинга можуть бути корисні при поділі "переплетених" обчислень, коли одночасно обчислюються два незалежні друзі від друга величини. Наприклад, в одному циклі може обчислюватися скалярний

добуток двох векторів, а також мінімальний і максимальний елемент кожного вектора, і такі цикли можуть бути розщеплені за допомогою побудови слайсів.

Методи динамічного й статистичного аналізу

Статистичний аналіз покриття базових блоків програми дозволяє встановити, чи виконувався коли-або при виконанні програми на заданій безлічі наборів вхідних даних заданий базовий блок.

Статистичне порівняння трас дозволяє виявити, чи однакові траси програми, отримані при різних запусках на тому самому наборі вхідних даних.

Статистична побудова графа потоку керування будує граф потоку керування на підставі інформації про порядок проходження базових блоків на одному наборі або на безлічі наборів вхідних даних.

Динамічне просування копій уздовж трас необхідно для точного міжпроцедурного аналізу залежностей за даними на основі траси виконання програми. Оскільки траса виконання програми, по суті, є одним більшим базовим блоком, просування копій – нескладне завдання.

Динамічне виділення мертвого коду дозволяє виявити інструкції програми, які виконувалися при даному запуску програми, але не зробили ніякого впливу на результат роботи програми. Якщо аналізується сукупність запусків програми на безлічі наборів вхідних даних, можна говорити про статистичне виділення мертвого коду.

Динамічний слайсинг залишає в трасі програми тільки ті інструкції, які вплинули на обчислення даного значення в даній точці програми (прямій динамічний слайсинг), або тільки ті інструкції, на які вплинуло присвоювання значення даної змінної в даній точці програми.

Помітимо, що про точність динамічних методів аналізу можна говорити, тільки якщо відомо повне тестове покриття програми (побудова повного тестового покриття – алгоритмічно нерозв'язне завдання). У противному випадку статистичне виявлення властивостей програми не дозволяє нам затверджувати, що дана властивість справедливо на всіх припустимих наборах вхідних даних.

Тому описані вище динамічні методи не можуть застосовуватися в автоматичному інструменті аналізу програм. Роль цих методів у тому, щоб привернути увагу користувача інструмента аналізу програм до особливостей роботи програми. Надалі користувач може вивчити "підозрілий" фрагмент коду більш детально із застосуванням інших інструментів, щоб підтвердити або спростувати висунуту гіпотезу.

Якщо непрозорі предикати й недосяжний код усуваються тільки на підставі статистичного аналізу, завжди залишається можливість, що предикат був істотним (як у прикладі вище). Щоб все-таки спростити програму, можна, наприклад, винести приблизно недосяжний код із загального графа потоку керування функції в оброблювач спеціального виключення, що збуджується щораз, коли предикат прийме значення, відмінне від звичайного. З одного боку, граф потоку керування й потоку даних основної програми в результаті спроститься, а з іншого боку, програма збереже свою функціональність.

З рисунку 3.2 ми бачимо, що програма ускладнення аналізу алгоритмів роботи вихідного коду виконує наступні дії над кодом, які визначаються заданими видами ускладнення аналізу алгоритмів роботи вихідного коду:

1. Лексичне ускладнення аналізу алгоритмів роботи вихідного коду:

– Видалення всіх коментарів у коді програми.

– Видалення різних пробілів.

– Заміна імен ідентифікаторів.

– Додавання різних зайвих операцій.

– Зміна розташування блоків.

2. Ускладнення аналізу алгоритмів роботи вихідного коду даних.

а) Ускладнення аналізу алгоритмів роботи вихідного коду зберігання:

– Зміна інтерпретації даних певного типу.

– Зміна строку використання сховищ даних.

– Перетворення статичних даних у процедурні.

– Поділ змінних.

– Зміна подання (або кодування).

б) Ускладнення аналізу алгоритмів роботи вихідного коду з'єднання:

– Об'єднання змінних.

– Реструктурування масивів.

– Зміна ієрархій спадкування класів

в) Ускладнення аналізу алгоритмів роботи вихідного коду переупорядкування.

3. Ускладнення аналізу алгоритмів роботи вихідного коду керування.

а) Обчислювальна ускладнення аналізу алгоритмів роботи вихідного коду:

– Розширення умов циклів.

– Додавання недосяжного коду.

– Усунення бібліотечних викликів.

– Додавання надлишкових операцій.

– Розпаралелювання коду.

б) Ускладнення аналізу алгоритмів роботи вихідного коду з'єднання:

– Вбудовування функцій.

– Добування функцій.

– Чергування, об'єднання фрагментів коду програми.

– Клонування.

– Трансформація циклів.

– Розгорнення циклів.

– Поділ циклів.

в) Ускладнення аналізу алгоритмів роботи вихідного коду послідовності.

Рисунок 3.2 – Функціональна схема системи

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 бакалаврського проектування, наведена на рисунку 3.3.

Після початку роботи розробленого ПЗ ми потрапляємо до головного вікна ПЗ звідки можемо перейти до відкриття вихідного коду ПЗ що розробляється далі проходить аналіз мови вихідного коду, створення та формування пакету обфускації, аналіз та обрання типу обфускації з можливістю застосування алгоритму Колберга чи алгоритму Ченга Вонга. Далі проводиться обфускація, лексична обфускація, обфускація даних та керування та в кінцевому результаті збереження видозміненого коду.

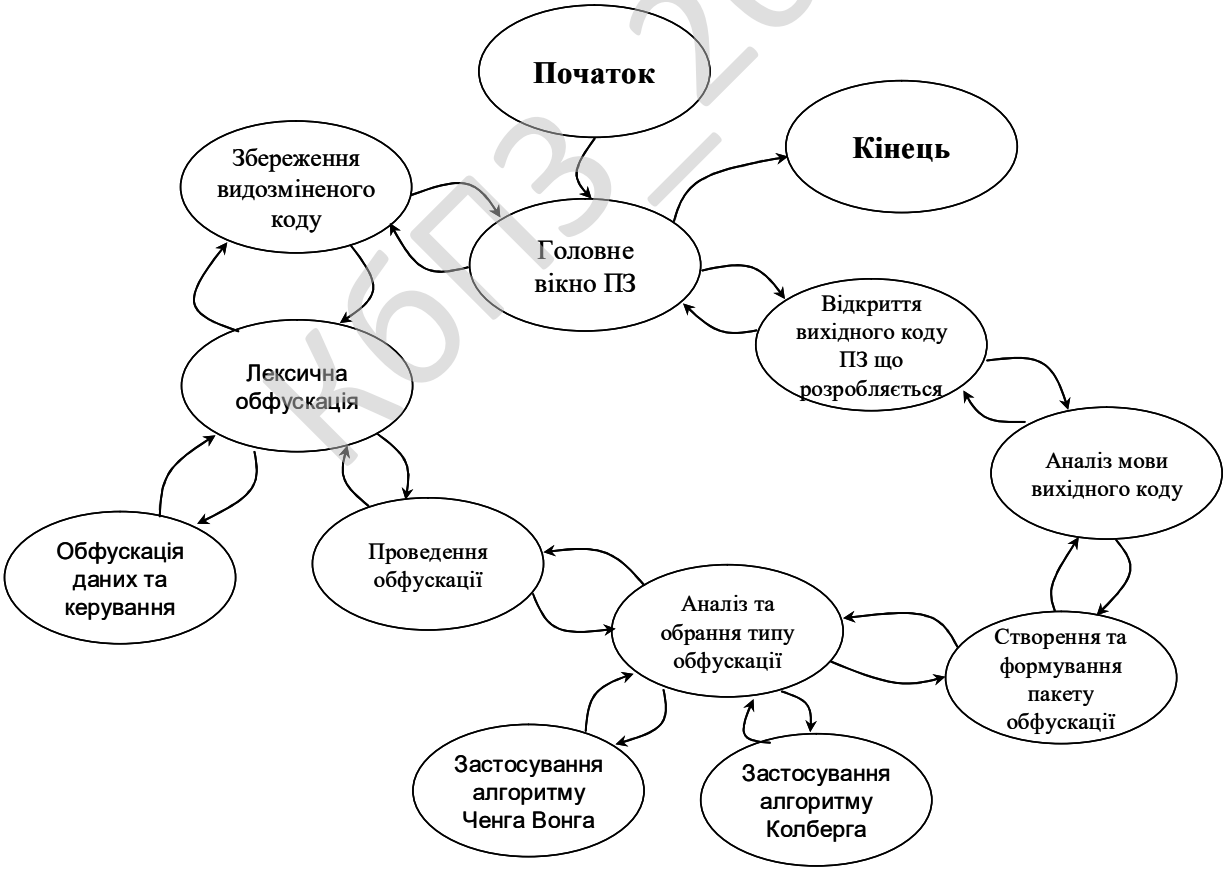

Рисунок 3.3 – Діаграма взаємодії процесів

4 РЕАЛІЗАЦІЯ ПРОЕКТУ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ПРАВИЛЬНІСТЬ ПРОЕКТНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

На рисунку 4.1 наведено блок-схему основної програми. Її робота складається з виконання наступних кроків:

– Виведення головного вікна ПЗ.

– Завантаження вихідного коду ПЗ що розробляється.

– Сканування та визначення мови програми та тип ресурсів.

– Створення та формування пакету обфускації.

– Аналіз та обрання типу обфускації.

– Збереження налаштувань.

– Підвергнення коду програми обфускації.

– Збереження видозміненого коду програми.

– Виведення звіту обфускації.

– Запустити на виконання ПЗ (запит).

– Запуск на виконання ПЗ після обфускації.

– ПЗ завантажено (запит).

– Повідомлення про успішне проходження обфускації.

– Запит WM_CLOSE?

На рисунку 4.2 наведено блок-схему підпрограми проведення обфускації. Її робота складається з виконання наступних кроків:

– Запит проведення лексичної обфускації?

– Крок 1. Видалення коментарів у вихідному коді ПЗ.

– Крок 2. Видалення системних символів.

– Крок 3. Пошук та заміна типів даних на системні.

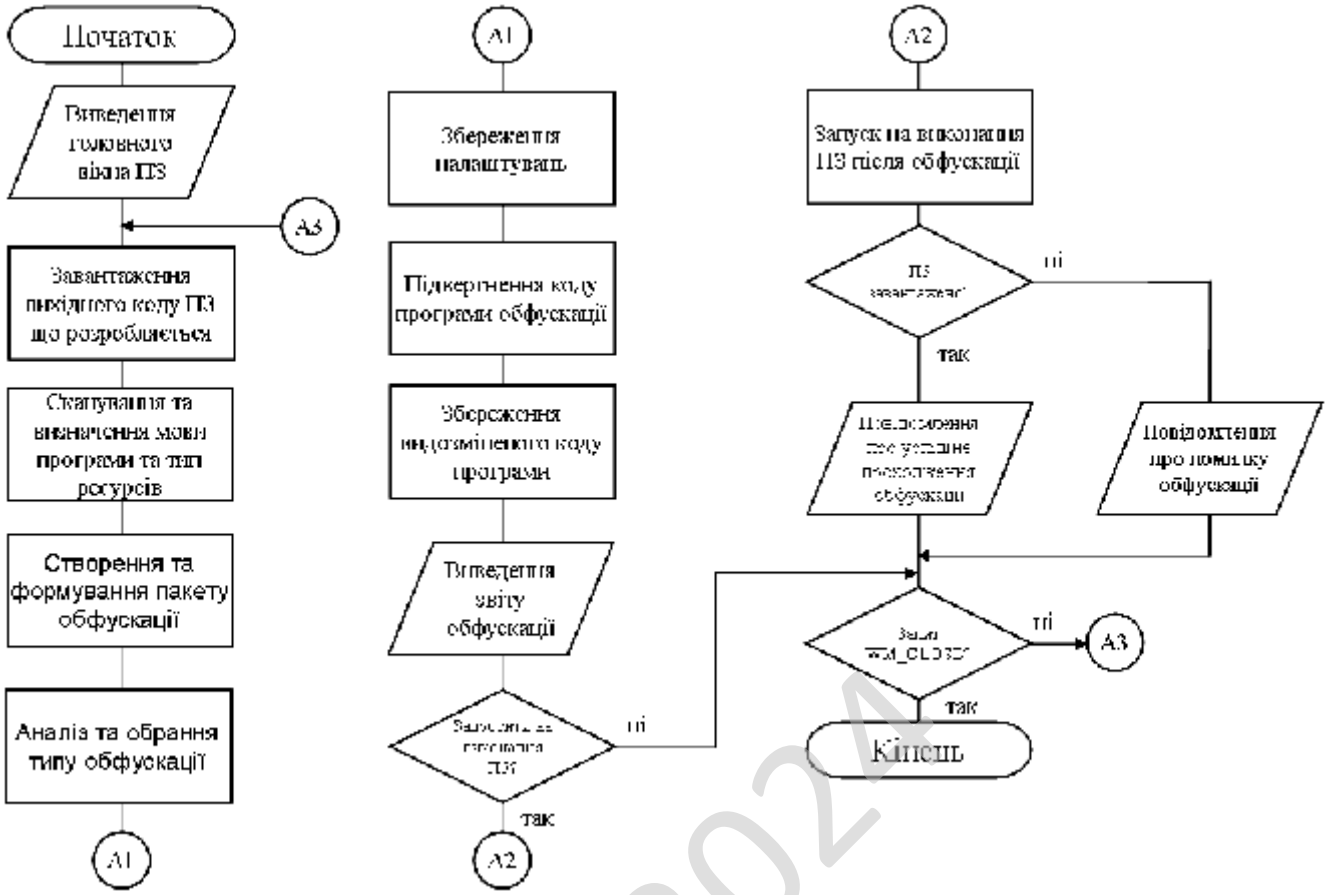

Рисунок 4.1 – Блок-схема основної програми

– Крок 4. Генерування додаткових операцій (зайвих).

– Крок 5. Випадкова зміна розташування блоків.

– Запит проведення обфускації даних?

– Поділ змінних.

– Зміна рядок використання сховищ даних.

– Перетворення статичних даних у процедурні.

– Зміна інтерпретації даних певного типу.

– Кодування даних.

– Зміна ієрархій спадкування класів.

– Реструктурування масивів.

– Об'єднання змінних.

– Запит проведення обфускація керування?

– Додавання надлишкових операцій.

– Усунення бібліотечних викликів.

– Розширення умов циклів.

– Ділення коду на частини.

– Вбудовування та додавання функцій.

– Чергування, об'єднання фрагментів коду ПЗ.

– Трансформація, розгорнення та поділ циклів.

– Клонування коду.

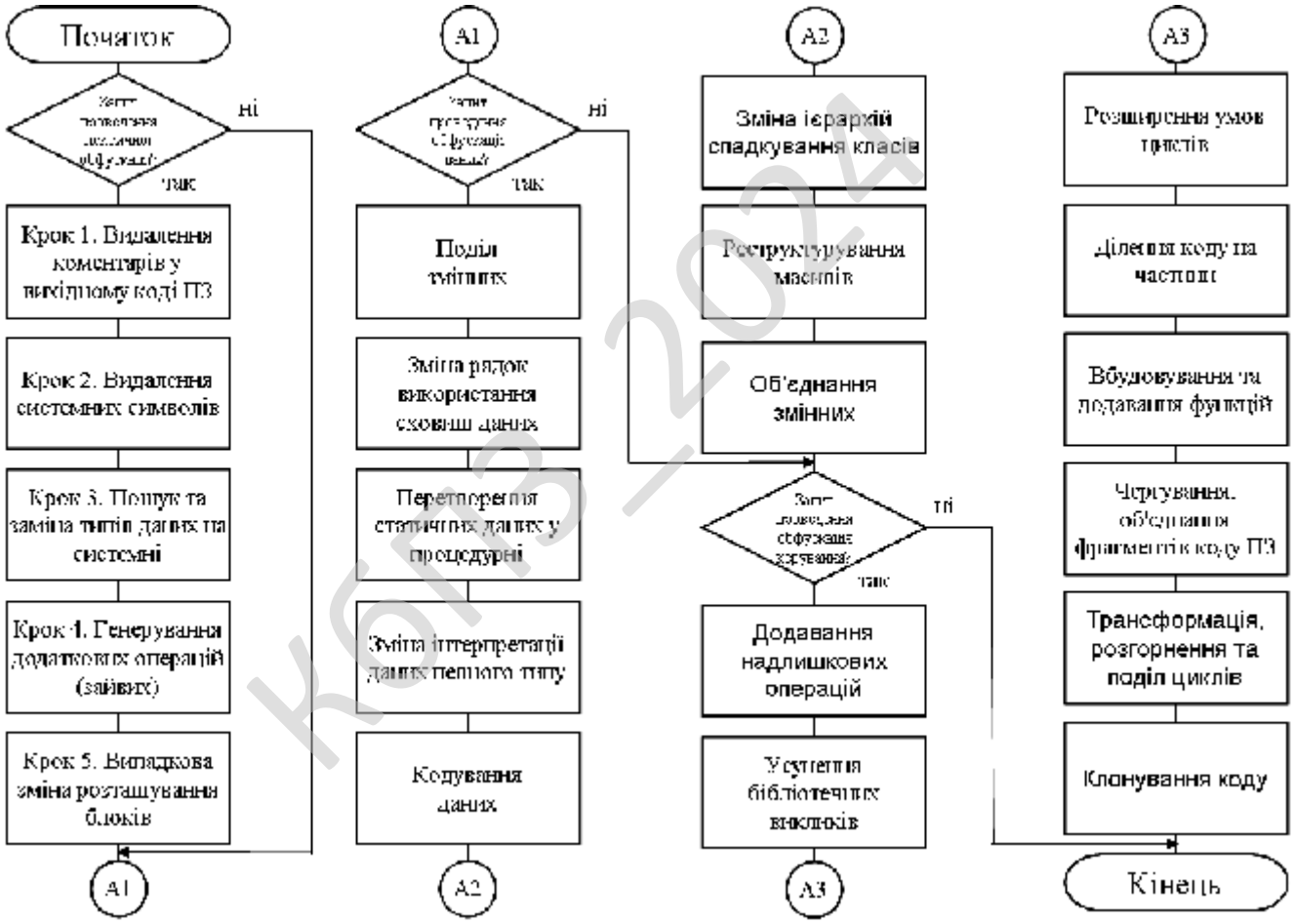

Рисунок 4.2 – Блок-схема підпрограми проведення обфускації

Опис алгоритмів функціонування системи

Розглянемо приклад роботи обфускатора:

Спочатку був наступний вихідний код:

```
for i:=0 to Pred(Count) do
   DoSome;
```

Після роботи розробленого ПЗ отримали наступний вихідний код:

```
asm
  push eax
  pop eax
  jmp @@1
@@1:
  jne @@2
  je  @@3
@@3:
  push eax
  jmp @@4
@@2:
  jmp @@5
@@4:
  pop eax
@@5:
  nop
 end;
 for i:=0 to Pred(Count) do
    begin
      asm
        jmp @@6
        db ....
        db .... різні опкоди, які нічого не роблять
       @@6:
        nop
      end;
      DoSome;
    end;
```

Так що незважаючи на всі можливості відладника ми в ньому одержимо досить цікаву мішанину сміттєвого коду, що може утруднити налаштування та взлом програми.

До програмування обфускація має опосередковане відношення, тому що важливо тільки на стадії відправлення продукту в продакш без надання вихідних кодів.

Захист від Restorator, ResHacker, Exescope і інших програм дизасемблювання коду програм: досить вставити кілька разів до подій на натискання кнопок або на подію форми OnCreate, такий код.

```
var str:string;
    i:integer;
str:='ььь|хььххалдорвдолартожкду6г4щуьфхэккхщгмхфхйхт';
for i:=1 to length(str) do
str:=Chr(Ord(str) xor $AABDEE66);
form1.memo1.lines.text:=str;
```

У прикладі, рядок str-заксорений нормальний рядок з текстом. Щоб одержувати заксорені рядки з нормальних, рекомендую створити просту програму-X0RER: Киньте на форму 2 memo, 1 edit і кнопку. Для кнопки:

```
procedure TForm1.Button1Click(Sender: TObject);
var i:integer;
     s:string;
begin
S:=memo1.text;
for i:=1 to length(S) do
s:=Chr(Ord(S) xor strtoint(edit1.text));
memo2.Text:=s;
end;
```

Все, пишете в memo1 нормальний рядок, в Edit1 число, на яке буде вироблятися XOR. Натискаємо кнопку й в memo2 заксорений текст. А в програмі робите взаємооборотну процедуру, зазначену вище. Головне, що всякі

Ресторатори не розпізнають у формах нечитабельні символи, що не відносяться до алфавітних й циферних знаків. Тому, при відкритті в Ресторатрі форми із заксорениими рядками він не зможе цього зробити.

Що саме дивне, компілятор не видає повідомлення про помилку, що ми вставляємо в string несимвольні дані.

Захист від відладників

Пропонується використовувати виявлення в зовнішніх криптерах, типу ASProtect, TElock, SVKP, BopCrypt і ін. Але якщо програма взлому знімає "навісний захист", то Ви втрачаєте анти-відладочні прийоми. Як же цього уникнути – просто вставити ці прийоми в тих місцях, наприклад, де використовуються функціональні обмеження. Причому, дуже важливо, ніколи не вставляйте посилання на процедуру перевірки відладника, а вставляйте цілий код перевірки знову й знову. Розглянемо приклад.

Так не треба робити:

```
Procedure SoftICEActive:boolean;
begin
...
код перевірки
...
End;
procedure TForm1.OnlyForReggedUserButtonClick(Sender: TObject);
begin
if SoftIceActive=true then begin
ShowMessage('SoftIce Active');
Halt;
end else begin
...
нормальний код
...
end;
```

Так треба робити:

```
procedure TForm1.OnlyForReggedUserButtonClick(Sender: TObject);
begin
//код перевірки активності SoftICE
if SoftIceActive=true then begin
ShowMessage('SoftIce Active');
Halt;
end else begin
...
нормальний код
...
end;
```

Програма взламу не зможе патчити програму, що захищається. А от сам код виявлення SoftICE:

1 Спосіб.

```
function SoftIce95: boolean;
var hfile: Thandle;
begin
  result:=false;
    hFile:=CreateFile('\\.\SICE',
    GENERIC_READ or GENERIC_WRITE,
    FILE_SHARE_READ or FILE_SHARE_WRITE,
    nil, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0);
  if(hfile<>INVALID_HANDLE_VALUE) then  begin
  CloseHandle(hfile);
  result:=true;
end;
end;
```

2 Спосіб (універсальний). Для цього використовується точна копія функції NmSymIsSoftICELoaded з NMTRANS.DLL, якою користується "рідний" Symbol Loader (loader32.exe), хоча, при бажанні, можна користуватися безпосередньо функціями з NMTRANS.DLL (остання входить у сам SoftICE). Користуватися цим кодом, можна як окремим юнітом.

```
unit Security;
interface
uses Windows;
const nmtrans = 'NMTRANS.DLL';
function IsSoftICELoaded: BOOL; stdcall;
function NmSymIsSoftICELoaded: BOOL; stdcall;
{$EXTERNALSYM NmSymIsSoftICELoaded}
function DevIO_ConnectToSoftICE: THANDLE; stdcall;
{$EXTERNALSYM DevIO_ConnectToSoftICE}
implementation
{$IFDEF _USE_NMTRANS_DLL}
function NmSymIsSoftICELoaded; external nmtrans name
'NmSymIsSoftICELoaded';
function DevIO_ConnectToSoftICE; external nmtrans name
'DevIO_ConnectToSoftICE';
{$ELSE}
function NmSymIsSoftICELoaded: BOOL;
var hf: THandle;
```

```
begin
  Result := TRUE;
  hf := DevIO_ConnectToSoftICE();
  if hf<>INVALID_HANDLE_VALUE
    then CloseHandle(hf)
    else Result := FALSE;
end;
function DevIO_ConnectToSoftICE: THANDLE;
const
  si_9x : PChar = '\\.\SICE'#0#0;
  si_nt : PChar = '\\.\NTICE'#0#0;
begin
  Result := CreateFile(si_9x, $80000000, $3, nil, $3, $80, $0);
  if Result<>INVALID_HANDLE_VALUE then Exit;
  Result := CreateFile(si_nt, $80000000, $3, nil, $3, $80, $0);
  if Result<>INVALID_HANDLE_VALUE then Exit;
  SetLastError($0A6580001);
end;
{$ENDIF}
function IsSoftICELoaded: BOOL;
begin
  Result := NmSymIsSoftICELoaded;
end;
end.
```

3 Спосіб. Асемблерний код:

```
00000000 : B4 43                    MOV   AH, 43h
00000002 : CD 68                    INT   68h
00000004 : 66 3D 86 F3              CMP   AX,0F386h
00000008 : 75 06                    JNE   00000010
0000000A :                          ; Активний
0000000E : EB 04                    JMP 00000012
00000010 :                          ; Не Активний
```

Цей спосіб варто використовувати тільки в операційних системах Win9x.

4.2 Захист розробленого програмного забезпечення

Захист розробленого програмного забезпечення буде відбуватися за допомогою RC6 – симетричний блоковий криптографічний алгоритм, похідний від алгоритму RC5. Був створений Роном Рівестом, Меттом Робшау і Реєм

Сіднеєм для задоволення вимог конкурсу Advanced Encryption Standard (AES). Алгоритм був одним з п'яти фіналістів конкурсу, був також представлений NESSIE і CRYPTREC. Є власницьким (пропрієтарним) алгоритмом, і запатентований RSA Security, одначе дія патентів сплила, і зараз алгоритм знаходиться у відкритому доступі. В той же час, "RC6" залишається зареєстрованою торговою маркою RSA.

Варіант шифру RC6, заявлений на конкурс AES, підтримує блоки довжиною 128 біт і ключі довжиною 128, 192 і 256 біт, але сам алгоритм, як і RC5, може бути налаштований для підтримки більш широкого діапазону довжин як блоків, так і ключів (від 0 до 2040 біт)[1]. RC6 дуже схожий на RC5 за своєю структурою і також досить простий у реалізації.

Є фіналістом AES, проте одна з примітивних операцій – операція множення, повільно виконується на певному обладнанні і ускладнює реалізацію шифру на ряді апаратних платформ і, що виявилося сюрпризом для авторів, на системах з архітектурою Intel IA-64 також реалізована досить погано. В даному випадку алгоритм втрачає одну зі своїх ключових переваг – високу швидкість виконання, що стало причиною для критики і однією з перепон для обрання як нового стандарту.

Деталі RC6

Так само, як і RC5, RC6 – повністю параметризована сім'я алгоритмів шифрування. Для специфікації алгоритму з конкретними параметрами, прийнято позначення RC6-w/r/b, де

– W – довжина машинного слова в бітах.

– R – число раундів.

– B – довжина ключа в байтах. Можливі значення 0 .. 255 байт.

Для того щоб відповідати вимогам AES, блочний шифр повинен працювати з 128-бітовими блоками. Так як RC5 – виключно швидкий блочний шифр, розширення його, щоб працювати з 128-бітовими блоками, привело б до використання двох 64-бітових робочих регістрів. Але архітектура і мови

програмування ще не підтримують 64-бітні операції, тому довелося змінити проект так, щоб використовувати чотири 32-бітних регістри замість двох 64-бітних.

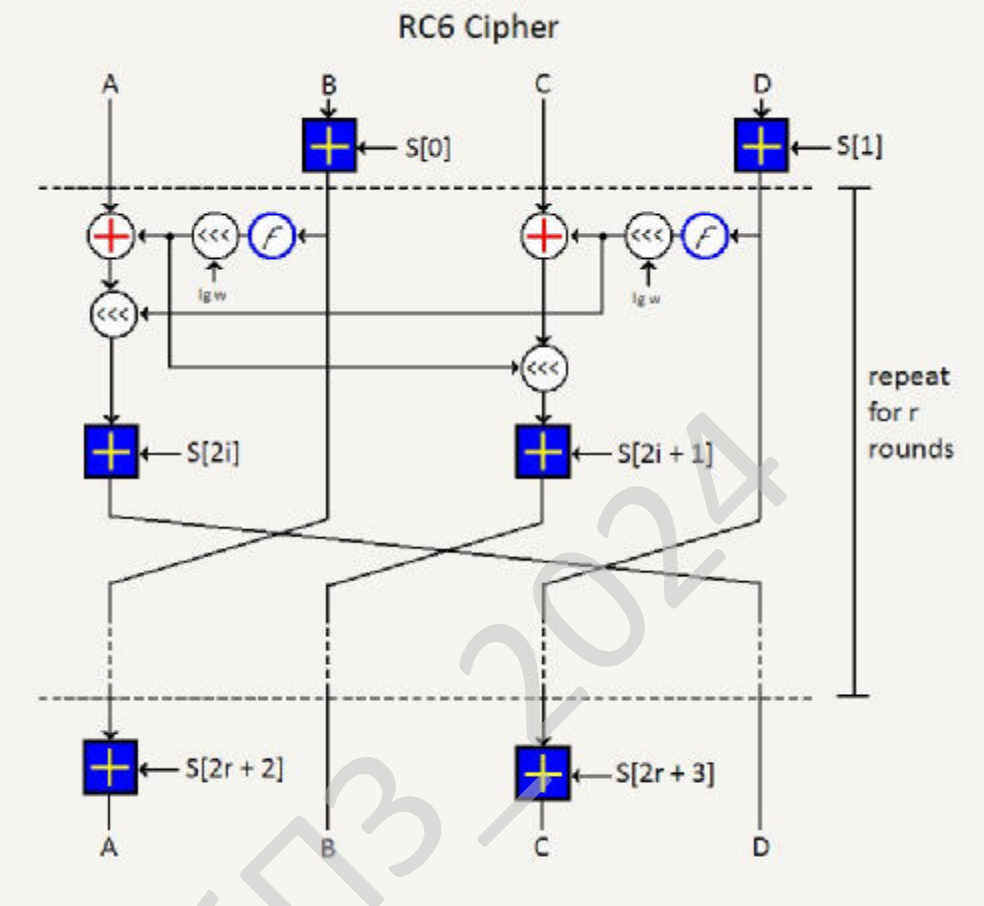

Рисунок 4.3 – Структура RC6

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ КІБЕРБЕЗПЕКИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

На рисунку 5.1 зображено головне вікно програми. З нього видно, що інтерфейс користувача програми складається з таких логічних блоків:

– Меню: Обфускація. Ресурси. Типи даних. Налаштування. Довідка..

– Вихідний код(Вхідні дані).

– Лексична обфускація.

– Обфускація коду.

– Отриманий код (Вихідні дані).

– Кнопка відкрити файл, відкрити проект.

– Кнопка зачинити все.

– Кнопка обфускація.

– Кнопка налаштування обфускації, налаштування ПЗ.

– Кнопка журнал дій.

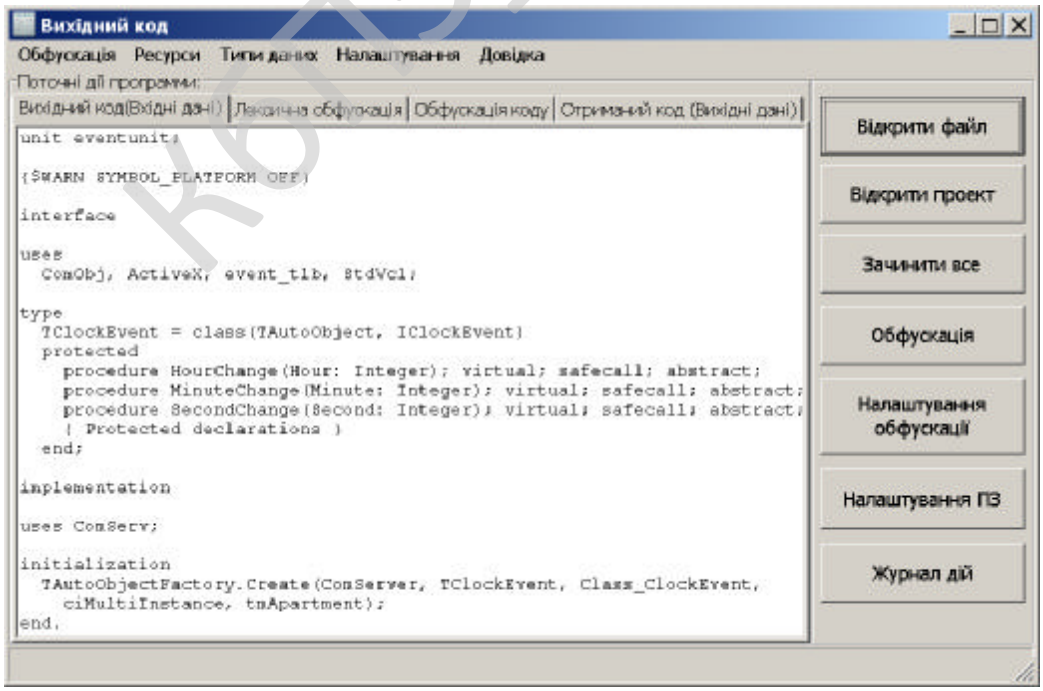

Рисунок 5.1 – Головне вікно програми

На рисунку 5.2 зображено форму авторського права, де відображені дані розробника.

Було обрано Shareware умову розповсюдження. Під умовно-безплатним програмним забезпеченням можна розуміти спосіб або метод розповсюдження комерційного ПЗ на ринку (тобто на шляху до кінцевого користувача), при якому випробувачеві пропонується обмежена за можливостями (неповнофункціональна або демонстраційна версія), терміном дії (тріал версія) або версія з вбудованим набридливим нагадуванням про необхідність оплати використання програми.

В угоді про використання (ліцензії для кінцевого користувача, EULA) також може бути обумовлена заборона на комерційне або професійне (не тестове) її використання. Основний принцип умовно-безплатного ПЗ – «спробуй, перш ніж купити» (try before you buy). ПЗ що поширюється як умовно-безплатний, надається користувачам безоплатно.

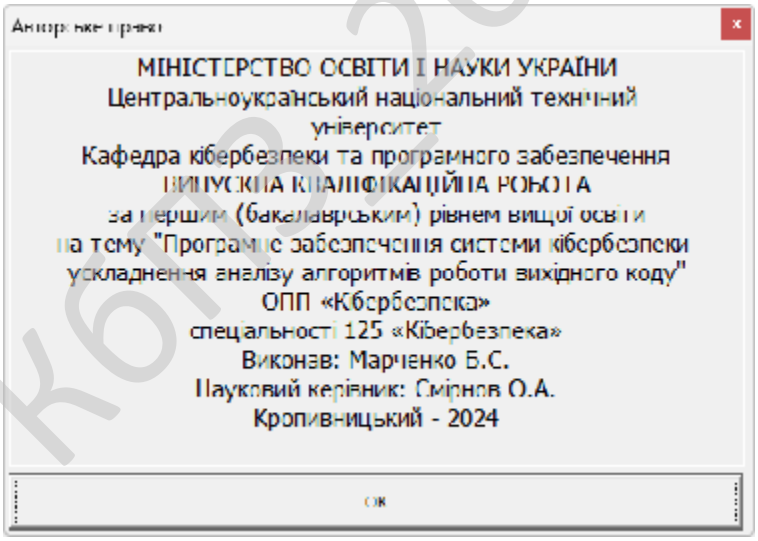

Рисунок 5.2 – Довідка автора

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, призначено для системи кібербезпеки ускладнення аналізу алгоритмів роботи вихідного коду.

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

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем ускладнення аналізу алгоритмів роботи вихідного коду.

– Досліджена система ускладнення аналізу алгоритмів роботи вихідного коду.

– На основі отриманих результатів досліджень створена програмна реалізація системи кібербезпеки ускладнення аналізу алгоритмів роботи вихідного коду.

Розроблені під час виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання ускладнення аналізу алгоритмів роботи вихідного коду.

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. Дана мова програмування дозволяє найбільш ефективно обробляти дані призначені для системи кібербезпеки ускладнення аналізу алгоритмів роботи вихідного коду. 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 кібербезпеки й включає програми, що реалізують її функції.

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

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм RC6.

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.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Knuth D. The Art of Computer Programming, Vol. 1: Fundamental Algorithms, 3rd Edition 3rd Edition. – Addison-Wesley Professional, 2019. – 672 p.

2. Knuth D. Art of Computer Programming, Vol. 2: Seminumerical Algorithms 3rd Edition, Kindle Edition. – Addison-Wesley Professional, 2019. – 672 p.

3. Knuth D. The Art of Computer Programming: Vol. 3: Sorting and Searching 2nd Edition, Kindle Edition. – Addison-Wesley Professional, 2019. – 800 p.

4. Cormen T.H., Leiserson C.E., Rivest R.L., Stein C. Introduction to Algorithms, 3rd Edition (The MIT Press) 3rd Edition – The MIT Press, 2019. – 1292 p.

5. Smirnov, O., Odarchenko, R., Smirnova, T., Bondar, S., Volosheniuk, D. «Optimal Structure Construction of Private 5G Network for the Needs of Enterprises». Lecture Notes on Data Engineering and Communications Technologies, 2023, 178, pp. 208–223.

6. Smirnov, O., Karapetyan, A., Fedorov, E., «Creating Neural Network and Single Solution Human-Based Metaheuristic Methods of Solving the Traveling Salesman Problem». CEUR Workshop Proceedings, Volume 3312, 2022, pp. 47-58.

7. Smirnov O., Kuznetsov A., Kryvinska N., Kiian A., Kuznetsova K. «Full Non-Binary Constant-Weight Codes». SN Computer Science, Vol 2, 337, 2021. https://doi.org/10.1007/s42979-021-00739-w.

8. Smirnov O., Kovalenko O., Kovalenko A., Kavun S. «Quantitative Risk Assessment Method Development in the Context of the SDLC-model». 2021 IEEE 8th International Conference on Problems of Infocommunications, Science and Technology (PIC S&T), 2021, pp. 203-208, doi: 10.1109/PICST54195.2021.9772143

9. Smirnova T., Gnatyuk S., Berdibayev R., Avkurova Zh., Iavich M. «Cloud-Based Cyber Incidents Response System and Software Tools». Communications in Computer and Information Science, 2021, vol 1486. Springer, Cham. pp 169-184.

10. Smirnov, O., Kuznetsov, A., Potii, O., Poluyanenko, N., Stelnyk, I., Mialkovsky, D. «Combining and filtering functions in the framework of nonlinear-feedback shift register». International Journal of Computing; 2020, Volume 19, Issue 2 – Research Institute for Intelligent Computer Systems – 2020. – P. 247-256.

11. Smirnov O., Kuznetsov A., Kiian A., Kuznetsova T. «Non-binary constant weight coding technique». CEUR Workshop Proceedings. Volume 2740, 2020, Pages 102-114.

12. Smirnov O., Kuznetsov A., Kiian A., Cherep A., Kanabekova M., Chepurko I. «Testing of code-based pseudorandom number generators for post-quantum application». 2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT), Ukraine, Kyiv, May 14-18. 2020. P. 172-177.

13. Smirnov, O., Shekhanin, K., Kuznetsov, A., Krasnobayev, V. «Detecting Hidden Information in FAT». International Journal of Computer Network and Information Security (IJCNIS). Vol. 12, No. 3, 2020. PP.33-43.

14. Smirnov, O., Drieieva, H., Drieiev, O., Simakhin, V., Bondar, S., Odarchenko, R. «Managing multifractal properties of the binary sequence generated with the Markov chains», CEUR Workshop Proceedings Volume 2608, 2020, Pages 633-645.

15. Smirnov O. Kuznetsov A., Zaicenko Yu., Pastukhov M., Oleshko O., Kuznetsova K., «Formation of Discrete Signals with Special Correlation Properties». International Conference on Information and Telecommunication Technologies and Radio Electronics, UkrMiCo 2019; Odessa; Ukraine; 9-13 September 2019. P.22-28.

16. Smirnov, O., Kuznetsov, A., Kolovanova, I., Kuznetsova, T., «Noise immunity of the algebraic geometric codes». International Journal of Computing; 2019, Volume 18, Issue 4 – Research Institute for Intelligent Computer Systems – 2019. – P. 393-407.

17. Smirnov, O., Kuznetsov, A., Reshetniak, O., Ivko, N., Katkova, T., Kuznetsova, T., «Generators of Pseudorandom Sequence with Multilevel Function of

Correlation». 2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T), Kyiv, Ukraine, 8 – 11 October 2019 . P.517-522.

18. Smirnov, O., Krasnobayev, V., Yanko, A., Kuznetsova, T. «Methods of nulling numbers in the system of residual classes». CEUR Workshop Proceedings, Vol 2588, P. 90-106, 2019.

19. Kuznetsova, T., «Code-Based Schemes for Post-Quantum Digital Signatures», 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2019; Metz; France; 18-21 September 2019. P. 707-712.

20. Smirnov, O., Kuznetsov, A., Stefanovych, O., Gorbenko, Y., Krasnobaev, V., Kuznetsova K. «Information Hiding Using 3D-Printing Technology», 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2019; Metz; France; 18-21 September 2019. P.701-706.

21. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Averchev, A., Pastukhov, M., Kuznetsova, K., «Formation of Pseudorandom Sequences with Special Correlation Properties», 2019 3rd International Conference on Advanced Information and Communications Technologies, AICT -2019/ Lviv, Ukraine, 2-6 July, 2019, P. 395-399.

22. Вінтенко Б.Ю., Смірнов О.А., Коваленко А.С., Смірнов С.А., Буравченко К.О. «Дослідження вимог міжнародних стандартів IEC60880 та IEC62138 з розробки програмного забезпечення інформаційно-керуючих систем АЕС, важливих для безпеки». *Системи управління, навігації та зв'язку*, 2023, вип. 3(73), С. 155-166.

23. Вінтенко Б.Ю., Смірнов О.А., Коваленко О.В., Смірнов С.А., Коваленко А.С. «Дослідження нормативних документів та галузевих стандартів розробки програмного забезпечення комп'ютерних систем управління АЕС,

важливих для безпеки». Системи управління, навігації та зв'язку, 2023, вип. 2(72), С. 170-178.

24. Аль-Мудхафар Акіл Абдулхуссейн М., Смірнова Т.В., Буравченко К.О., Смірнов О.А. «Метод оцінки та підвищення користувальницького досвіду абонентів в програмно-конфігурованих мережах на основі використання машинного навчання». Сучасні інформаційні системи, 2023, том 7, № 2, С. 49-56.

25. Вінтенко Б.Ю., Смірнов О.А., Коваленко О.В., Смірнов С.А. «Дослідження нормативної документації та стандартів розробки програмного забезпечення комп'ютерних систем управління АЕС, важливих для безпеки». VI міжнародна науково-практична конференція "Інформаційна безпека та комп'ютерні технології", м. Кропивницький. 20-21 квітня 2023 р. – Кропивницький: ЦНТУ. – 2023. – С. 35-36.

26. Смірнов, О.А., Усік П.С., Полігенько О.О., Одарченко Р.С., Терещенко Л.Ю. «Інформаційна технологія та програмне забезпечення для підвищення ефективності планування підсистеми базових станцій стільникового зв'язку». Проблеми телекомунікацій. № 1(26). С. 83-96. 2020.

27. Смірнов О.А., Усік П.С., Миронець І.В., Буравченко К.О., Якименко Н.М. «Метод підвищення ефективності розподіленої обробки даних у комп'ютерних системах операторів стільникового зв'язку» Вісник Черкаського державного технологічного університету. Технічні науки. №4. С. 103-110. 2020.

28. О.А.Смірнов, Т.В.Смірнова, Л.І. Поліщук, К.О. Буравченко, А.О.Макевнін, «Дослідження хмарних технологій як сервісів», Кібербезпека: освіта, наука, техніка. № 3(7). С. 43-62. 2020.

29. Смірнов О.А., Дрєєва Г.М., Дрєєв О.М., Смірнова Т.В. «Фрактальний аналіз генератора самоподібного трафіку на основі ланцюга Маркова». Центральноукраїнський науковий вісник. Технічні науки. № 2(33). с. 161-172, 2019.

30. Smirnov, O., Kuznetsov, A., Kuznetsova., K. Synthesis of Discrete Signals with Improved Correlation Properties. Монографія: In.: ISCI'2019:

Information Security in Critical Infrastructures. Collective monograph. Edited by Ivan D. Gorbenko and Alexandr A. Kuznetsov, ASC Academic Publishing, USA, 2019, pp. 281-299. – ISBN: 978-0-9989826-8-7 (Hardback), ISBN: 978-0-9989826-9-4 (Ebook).

31. Смірнова Т.В., Солових Є.К., Смірнов О.А., Дрєєв О.М. Побудова хмарних інформаційних технологій оптимізації технологічного процесу відновлення та зміцнення поверхонь деталей. Центральноукраїнський науковий вісник. Технічні науки. № 1(32). с. 184-194, 2019.

32. Смірнов О.А., Котелянець В.В. Стійкі до колізій стохастичні моделі функціонування безпроводових сенсорних мереж. Вісник інженерної академії України, №3, с. 145-152, 2018

33. O. Smirnov, O. Kovalenko, A. Kovalenko, S. Smirnov, V. Vialkova. The mathematical model of the testing technology for DOM XSS vulnerabilities. Scientific & practical cyber security journal (SPCSJ) Vol 2 Issue 1, 22-28 pp. [Электронный Журнал]. Georgia. Tbilisi: SCSA – 2018.

34. Oleksii Smirnov, Oleksandr Kovalenko, Jamil Al-Azzeh, Anna Kovalenko, Serhii Smirnov. Qualitative risk analysis of software development. Asian Journal of Information Technology. – Volume 17(3). – Medwell Journals. – 2018. – P. 218-230.

35. Смірнов О.А., Коваленко О.В., Коваленко А.С., Смірнов С.А. Розробка методу передтестової компіляції й розподілу доступу. Збірник наукових праць III міжнародної науково-практичної конференції “Інформаційна безпека та комп’ютерні технології”, м. Кропивницький. 19-20 квітня 2018р. – Кропивницький: ЦНТУ. – 2018. – С. 214-215

36. Smirnov Oleksii, Kovalenko Oleksandr, Kovalenko Anna, Smirnov Serhii. Method of testing the DOM XSS vulnerability. International Conference «Information technologies, systems and networks ITSN-2017». Chisinau, Republic of Moldova. 17 – 18 October 2017. – Chisinau: Academy of Sciences of Moldova, Military Academy of Armed Forces “Alexandru cel Bun”. 2017. P7.

37. Смірнов О.А., Смірнов С.А., Коваленко О.В., Коваленко А.С. Технологія тестування DOM XSS уразливості. Науково-практичний журнал кібер безпеки (SPCSJ) № 1. [Електронний журнал]. Грузія. Тбілісі: SCSA - 2017.

38. Смірнов О.А., Лисенко І.А. Інформаційна технологія проектування тестових наборів з урахуванням вимог до програмного забезпечення. Системи управління, навігації та зв'язку. – Випуск 4 (44). - Полтава: ПолтНТУ. - 2017. - С. 112-115.

39. Смірнов О.А., Смірнов С.А., Рябой Д.К., Рябая О.В. Модель вузла комутації з відносними пріоритетами, резервуванням ресурсів і обліком реальної надійності обслуговуючих приладів .Збірник тез всеукраїнської науково-практичної інтернет-конференції «Автоматика та комп'ютерно-інтегровані технології у промисловості, телекомунікаціях, енергетиці та транспорті». м. Кропивницький. 16-17 листопада 2017 р. – Кропивницький: ЦНТУ. – 2017. – С. 198-199.

40. Смірнов О.А., Коваленко О.В. Використання псевдобулевих методів бівалентного програмування для управління ризиками розробки програмного забезпечення. Системи управління, навігації та зв'язку. – Випуск 1 (37). - Полтава: ПолтНТУ. - 2016. - С. 98-103.

41. Смірнов О.А., Лисенко І.А. Формалізація процесу проектування тестових наборів. Збірник наукових праць Харківського університету Повітряних Сил. Випуск 3 (48). - Харків: ХУПС. - 2016. - С.96-100.

42. Смірнов О.А., Лисенко І.А. Удосконалення методу перевірки коректності таблиць рішень для подання тестових наборів. Збірник наукових праць "Системи обробки інформації". - Випуск 8 (145). - Х.: ХУПС - 2016. - С. 77-80.

43. Смірнов О.А., Лисенко І.А. Розробка впорядкованих каскадних таблиць рішень із використанням матриць слідування. Збірник наукових праць "Системи обробки інформації". - Випуск 6 (143). - Х.: ХУПС - 2016. - С. 216-220.

44. Смірнов О.А., Коваленко О.В., Якименко Н.М., Доренський О.П. Метод кількісної оцінки ризиків розроблення програмного забезпечення. Збірник наукових праць Харківського університету Повітряних Сил. Випуск 2 (47). - Харків: ХУПС. - 2016. - С. 128-133.

45. Смірнов О.А., Коваленко О.В., Якименко Н.М., Доренський О.П. Метод якісного аналізу ризиків розроблення програмного забезпечення. Наука і техніка Збройних Сил України. – Випуск 2(23). - Харків: ХУПС. - 2016. - С. 150-158.

46. Смірнов О.А., Коваленко О.В., Якименко Н.М., Доренський О.П. Проблеми аналізу та оцінки ризиків інформаційної діяльності. Збірник наукових праць "Системи обробки інформації". - Випуск 3 (140). - Х.: ХУПС - 2016. - С. 40-42.

47. Смірнов О.А., Коваленко А.С., Коваленко О.В., Доренський О.П. Удосконалення методу технічного обслуговування об’єктів інтегрованої інформаційної системи. Системи озброєння і військова техніка. – Випуск 2(46) – Х.: ХУПС – 2016. – С. 103-107.

48. Smirnov A.A., Kovalenko A.V. Kovalenko A.S. Dorensky A.P. Information model and its element for displaying information on technical condition of objects of integrated information system. International Journal of Computational Engineering Research (IJCER). – Volume 6, Issue 1. – India. Delhi. – 2016. – P. 21-27.

49. Смірнов О.А., Євсеєв С.П., Король О.Г., Коваленко О.В., Коваленко А.С., Смірнов С.А. Архітектура мікропроцесорів та компонентів ЕОМ. Навчальний посібник – Кіровоград: Вид. Лисенко В.Ф., 2015. – 550 с.

50. Смірнов О.А., Коваленко О.В., Мелешко Є.В., Константинова Л.В., Кожанова А.С. Інженерія програмного забезпечення. Навчальний посібник. За ред. О.А. Смірнова. – Кіровоград: КНТУ 2013. – 409с.

51. Смірнов О.А., Коваленко О.В., Кожанова А.С., Левошко О.Л., Константинова Л.В. Основи системного програмування. Навчальний посібник. – Кіровоград: КНТУ 2013. – 257с.

Додаток А
(обов’язковий)

Технічне завдання

Зміст

1 Найменування та область застосування.....2
2 Підстава для розробки.....2
3 Мета та призначення розробки.....2
4 Джерела розробки.....2
5 Технічні вимоги.....2
5.1 Вміст проекту.....2
5.2 Показники призначення.....3
5.3 Вимоги до функціональних характеристик.....3
5.4 Вимоги до архітектури.....3
5.5 Вимоги до надійності.....3
5.6 Умови експлуатації.....4
5.7 Вимоги до складу та параметрів технічних засобів.....4
5.8 Вимоги до інформаційної і програмної сумісності.....4
5.8.1 Обладнання.....4
5.8.2 Мова програмування.....4
5.8.3 Вхідні дані.....5
5.8.4 Вихідні дані.....5
6 Вимоги до програмної документації.....5
7 Перелік документів, що розробляються.....5
8 Етапи розробки.....6
9 Порядок контролю та приймання.....6

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи кібербезпеки ускладнення аналізу алгоритмів роботи вихідного коду.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 136-02 від 01.04.2024 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є розробка програмного забезпечення системи кібербезпеки ускладнення аналізу алгоритмів роботи вихідного коду.

4 Джерела розробки

Джерелом цієї випускної кваліфікаційної роботи за першим (бакалаврським) 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 кібербезпеки з ОС та з користувачем;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– системи кібербезпеки ускладнення аналізу алгоритмів роботи вихідного 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, працювати в ОС Windows 10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпеченням.

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.

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 Перелік документів, що розробляються

- Структурна схема системи – 1 аркуш.
- Функціональна схема системи – 1 аркуш.
- Діаграма процесів – 1 аркуш.
- Блок-схема алгоритму роботи програми – 2 аркуша.
- Пояснювальна записка – 74 аркуші.

8 Етапи розробки

8.1 Збір і обробка інформації по темі випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти. Постановка задачі на виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти.

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

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

9 Порядок контролю та приймання

9.1 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на попередній захист 23.05.2024 р.

9.2 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на захист 8.06.2024 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти

_____ Смірнов О.А.

Програмне забезпечення системи кібербезпеки ускладнення аналізу алгоритмів роботи вихідного коду

Лістинг програми

Код документу 12

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

Загальна кількість аркушів: 101

Літера: РП

Кропивницький – 2024 року

Файл main_uscladnennya_analizu.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,
uscladnennya_analizu,
  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 F8 F6 05 00  иц▯ .........шц▯ .
0045F6B4  E8 F6 05 00 E0 F6 05 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  ▯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 uscladnennya_analizu.pas – захист на основі ускладнення аналізу алгоритмів роботи вихідного коду коду

```
   unit uscladnennya_analizu;

//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('БАКАЛАВРСЬКИЙ ПРОЕКТ');
Memo1.Lines.Add('');
Memo1.Lines.Add('на  тему:');
Memo1.Lines.Add('');
Memo1.Lines.Add('Програмне забезпечення системи кібербезпеки ускладнення аналізу
алгоритмів роботи вихідного коду');
Memo1.Lines.Add('');
Memo1.Lines.Add('');
Memo1.Lines.Add('Керівник: Смірнов О.А.');
Memo1.Lines.Add('');
Memo1.Lines.Add('Розробив: студент Марченко Богдан Станіславович');
Memo1.Lines.Add('                  гр. КБ-21-3СК');
Memo1.Lines.Add('');
Memo1.Lines.Add('');
Memo1.Lines.Add('м. Кропивницький 2024');
end;
end.
```