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 для запобігання аналізу та модифікації програмних продуктів”

Виконав здобувач вищої освіти
IV курсу, групи КБ-20
ОПП «Кібербезпека»
спеціальності 125 «Кібербезпека»
_____ Дроздов М.Р.
«____» _____ 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

Дроздову Максиму Романовичу
(прізвище, ім’я, по батькові)

1. Тема роботи *Програмне забезпечення системи кібербезпеки для запобігання аналізу та модифікації програмних продуктів*

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

затверджені наказом вищого навчального закладу № 135-02 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 для запобігання аналізу та модифікації програмних продуктів*

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
_____ Дроздов М.Р.
(прізвище та ініціали)

АНОТАЦІЯ

Дроздов М.Р. Програмне забезпечення системи кібербезпеки для запобігання аналізу та модифікації програмних продуктів. 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 для запобігання аналізу та модифікації програмних продуктів.

Метою розробки є програмне забезпечення системи кібербезпеки для запобігання аналізу та модифікації програмних продуктів.

Результат роботи – програмна реалізація системи кібербезпеки для запобігання аналізу та модифікації програмних продуктів.

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 Delphi 10.4.

Ключові слова: кібербезпека, аналіз та модифікація програмних продуктів

ABSTRACT

Drozdov M.R. Cybersecurity system software to prevent analysis and modification of software products. 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 a cyber security system to prevent the analysis and modification of software products.

The goal of the development is the cyber security system software to prevent the analysis and modification of software products.

The result of the work is the software implementation of the cyber security system to prevent the analysis and modification of software products.

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 Delphi 10.4 environment.

Keywords: cyber security, analysis and modification of software products

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

ПЗ	–	програмне забезпечення
ПП	–	програмний продукт

ВСТУП

Актуальність теми. З розвитком цифрових технологій також зростає загроза неймовірного рівня тиражування та незаконного відтворення програмного забезпечення. Тому рівень піратства зростає пропорційно. Цей сценарій явно створює загрозу для виробників програмного забезпечення та призводить до розробки численних методів захисту програмного забезпечення. Було розроблено численні методи захисту програмного забезпечення, і одним із таких методів захисту є обфускація коду.

Обфускація коду – це механізм для приховування оригінального алгоритму, структур даних або логіки коду, або для посилення або захисту коду (який вважається інтелектуальною власністю автора програмного забезпечення) від несанкціонованого процесу зворотного проектування. Загалом, обфускація коду передбачає приховування деталей реалізації програми від зловмисника, тобто перетворення програми на семантично еквівалентну (з тим самим обчислювальним ефектом) програму, яку набагато важче зрозуміти для зловмисника. Жодна з поточних методик обфускації коду не задовольняє всім критеріям ефективності обфускації для опору атакам зворотного проектування. Тому дослідники, а також індустрії програмного забезпечення намагаються застосувати нові та кращі методи обфускації своєї інтелектуальної власності в звичайному процесі. Але, на жаль, програмний код небезпечний, тобто все одно його можна зламати.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи кібербезпеки для запобігання аналізу та модифікації програмних продуктів.

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

– Огляд існуючих систем для запобігання аналізу та модифікації програмних продуктів.

– Дослідження системи кібербезпеки для запобігання аналізу та модифікації програмних продуктів.

– Програмна реалізація системи кібербезпеки для запобігання аналізу та модифікації програмних продуктів.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі для запобігання аналізу та модифікації програмних продуктів.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи кібербезпеки для запобігання аналізу та модифікації програмних продуктів, 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

Система призначена для запобігання аналізу та модифікації програмних продуктів. Доступність метаданих до вашого коду (звичайно це MSIL, але більше читабельний ніж ассемблерний код) у продуктах .Net стала головним болем для розроблювачів комерційних продуктів і технологій для .Net. Як захистити код від зайво цікавої публіки, як захистити продукт як інтелектуальну власність (якщо можна легко декомпілювати в зрозумілий код, значить можна зняти захист від несанкціонованого використання, так ще зібрати в працюючу збірку). От отут на допомогу приходить обфускація.

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

1.2 Область застосування

Технології запобігання аналізу та модифікації програмних продуктів:

– На рівні машинного коду – Обфускація застосовується в таких частинах програми, як перевірка реєстраційного коду, тобто не критичних до швидкості, але критичних до безпеки. Найпростіший метод заплутати машинний код – вставити в нього недіючі конструкції (or ax, ax).

– На рівні вихідних текстів – Вихідний текст програми на скрипт-мовах (JavaScript, VBScript і т.п.) доступний користувачеві. Менш читаємим його можна зробити шляхом форматування й заміни імен.

– На рівні проміжного коду – Звичайні мови, такі як C++ і Паскаль, компілюють вихідний код у машинний. На відміну від них мови платформи

.NET, а також Net і Java, компілюють вихідний код у проміжний (байт-код), що містить досить інформації для відновлення вихідного коду. Тому для перерахованих мов використовується обфускація проміжного коду.

Нижче перераховані методи захисту, а так само алгоритми інтелектуального аналізу коду й ресурсів, реалізовані в проекті.

– Обфускація (Renaming) класів і їхніх членів, з повною підтримкою Generics, спадкування, перевантаження віртуальних методів, аналізу стандартних атрибутів обфускації.

– Об'єднання збирань (Assembly Merging) – об'єднання коду з декількох вихідних збирань, що захищаються, в одну.

– Декомпозиція структури класів у процедурне подання (Decomposition) – базується на ідеї перекладу програми з об’єктно-орієтованої форми (простої для реверс-інжинірингу) до процедурного стилю, з максимальним знищенням всієї доступної інформації збереженої в метаданих (але збереженням повної працездатності збирача сміття).

– Приховання виклику зовнішніх методів (External Method Call Hiding) – підміна явного виклику методів із зовнішніх збирань (у тому числі звертань до Common Language Runtime), на неявний обіг по некерованому покажчику, витягнутому з метаданих на основі зашифрованого ідентифікатора.

– Шифрування рядків (String Encryption) на основі власного алгоритму. Використовує динамічні змінні й шифровані блоки даних з метою максимального ускладнення потенційної автоматичної де-обфускації.

– Аналіз звертань до механізму відбиття (Reflection Analyzing) – набір алгоритмів, що відслідковують звертання до Reflection методів (як до методів явного обігу по імені, так і до перерахувань членів класу). Автоматично відслідковує зв'язку й коректує імена у вихідному коді.

– Автоматичне виключення з обфускації типів, методів і полів, до яких є обіги з Windows Presentation Foundation (WPF Analyzing) – декомпіляція й аналіз

BAML ресурсів у збираннях. Повністю підтримує розширення синтаксису WPF, включаючи складні конструкції (наприклад, PropertyPath).

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи кібербезпеки для запобігання аналізу та модифікації програмних продуктів, 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

Зробимо огляд, куди включимо найвідоміші обфускатори.

Загальні виводи й міркування:

– Безкоштовні обфускатори досить слабкі, і придатні тільки для простого перейменування. Про control flow знають із них лише деякі.

– Існують досить непогані рішення (control flow, MSIL encryption) вартістю до $500.

– Серйозні рішення коштують близько 5000, але на жаль, для багатьох з них є розпаковники. Деякі з них зламані.

– Зламали обфускатор – значить зрозуміли його систему захисту. На смітник такий обфускатор.

– Є рішення «проти хакерів» – збірка шифрується повністю й розшифровується на лету. Зламати символьним відладником таку збірку простіше простого.

Результати дослідження обфускаторів зібрані в таблиці 2.1.

Таблиця 2.1 – Результати дослідження обфускаторів

Назва й URL	Вартість	Control flow	Шифрування MSIL	Докладно
.NET Reactor	$180	+	+	Шифрує код, зламати його досить важко, але можливо, є розпаковник
{SmartAssembly}	$795	+	-	Використовується RedGate-ом. Зламаний.

Продовження таблиці 2.1

Назва й URL	Вартість	Control flow	Шифрування MSIL	Докладно
Aspose.Obfuscator	(-)	(-)	(-)	Проект більше не підтримується
Assemblur	Free	-	-	Плагін до студії + консоль. Майже нічого не обфускується
Babel	$250	+	?	Щось шифрує, але в рантаймі можна виконати DumlIL, можливо, повна версія працює добре
BitHelmet	$250	?	?	Упав, сказавши, що .NET відсутній
C# Source Code Obfuscator	?	-	-	Схоже, працює з вихідним кодом .NET. Цікавий підхід, але крім опису й приклада я нічого не знайшов
CilSecure	> $1000	?	?	Платний обфускатор, навіть без тріала.
CodeArmor	?	?	?	Ще один платний обфускатор, і теж без тріала. Підтримка не відповіла.

Продовження таблиці 2.1

Назва й URL	Вартість	Control flow	Шифрування MSIL	Докладно
CodeVeil	$900	+	+	Відомий тим, що зламується в антивірусах. У цілому, гарна штука
CodeWall	$400	+	+	Осудний обфускатор
Decompiler.NET	$550	-	-	Занедбаний 3 роки
DeepSea	$200	±	-	Схильний робити багато switch-їй. У цілому, неюзабельно.
DesaWare	$1500	+	?	Немає .NET 4.0
DNGuard HVM	$900	?	?	Проблемний інсталяційний пакет без половини кнопок, перший же додаток упало
Dotfuscator	$1900	+	?	Community-Версія досить убога, а Enterprise дорогою, але можливо, воно того коштує
dotNetProtector	$500	+	+	Додав до проекту більше 4 MB своїх DLL-ек
Eazfuscator.NET	Free	-	-	Простий rename

Продовження таблиці 2.1

Назва й URL	Вартість	Control flow	Шифрування MSIL	Докладно
Goliath.NET	$115	+	±	Збірка рефлектором не відкривається, але в WinDbg видно весь вихідний код
NetOrbiter	Free	-	-	Зробив свій проект, куди повністю скопіював мій exe-шник і щось дописав у добавок.
Obfuscar	Free	-	-	Простий rename, заснований на Mono.Cecil. .NET 4.0 не підтримує
Obfuscator.NET	$200	?	?	Зроблена збірка відразу ж упала. Навіть на .NET 3.5.
PCGuard for .NET	$400	?	?	Більше орієнтований на ліцензування, чим на обфускацію. Тріал не надіслали. Подивитися було б досить цікаво.

Продовження таблиці 2.1

Назва й URL	Вартість	Control flow	Шифрування MSIL	Докладно
Phoenix Protector	Free	(-)	(-)	Навіть .NET 3.5 не тримає
Salamander.NET	$800	-	-	На прикладі, наведеному на сайті, рефлектор, звичайно, лається, але дизассемблює добре
SharpObfuscator	Free	-	-	Як видно, продукт давно занедбаний
Skater.NET	$100	-	-	Дивна штука, перейменувала кілька методів + шукала старий ILDASM. Імовірно, занедбаний.
Spices.NET	$400	-	±	Шифрує всю збірку цілком, що погано
VMWare ThinApp	> $5000	+	+	Отриманий додаток може запускатися навіть без .NET.
Xenocode PostBuild	> $1000	+	+	Сам обфускатор зламаний, що наводить на не дуже гарні думки

.NET Reactor

Шифрує код за допомогою NecroBit (назва їхньої технології), на форумах ходять слухи про те, що NecroBit успішно зламаний. Можливо, я просто цього не знайшов. Код розібрати не вийшло, WinDbg теж нічого не знайшов.

Reflector

Щось витягти за допомогою WinDbg можна, але IL-код методів не віддається.

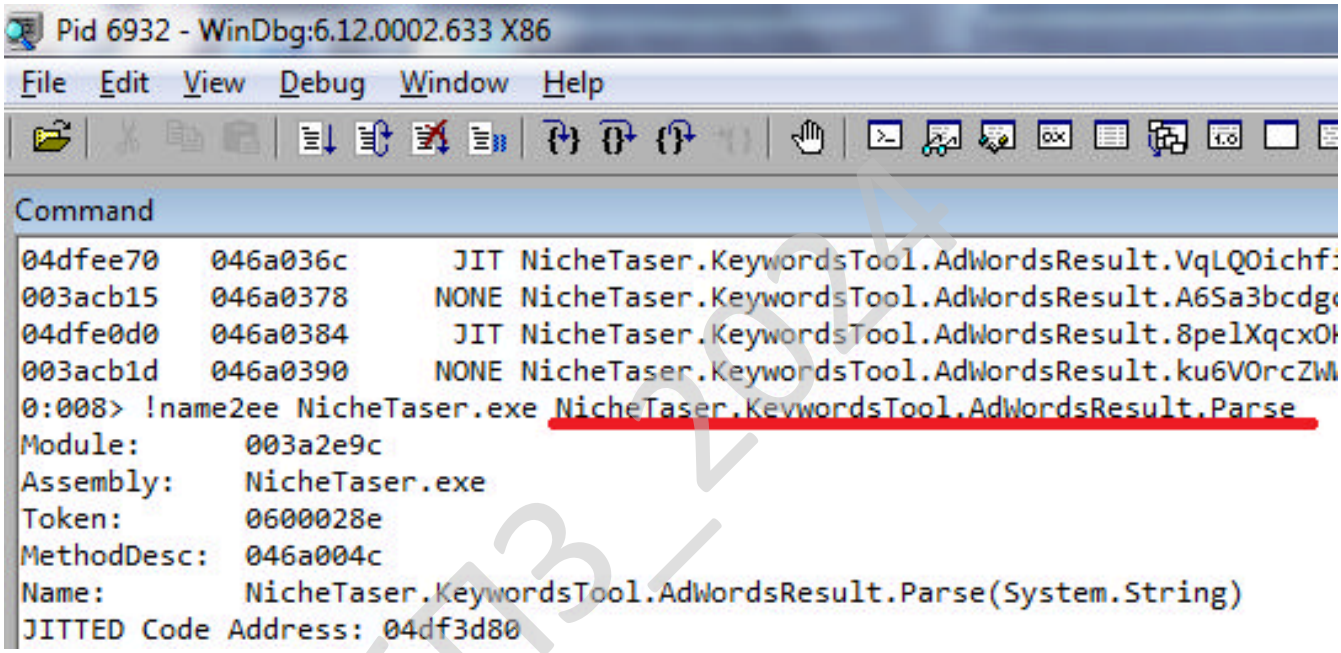

Рисунок 2.1 – Інтерфейс користувача Reflector

{SmartAssembly}

Його не дуже давно купила компанія RedGate. Чесно говорячи, вибору RedGate-А я не зрозумів, {sa} не вміє навіть шифрувати MSIL. Посидівши з відладником, можна розібратися в коді. Я не раджу використовувати цей обфускатор, ціна $750 явно не відповідає якості.

Усе, що робить цей обфускатор з кодом – це обфускація control flow у приблизно такому виді:

```
L_1 br.s L_4
L_2 br.s L_3
   L_3 ret
  L_4 push
L_5 ldc.i4.1
L_6 br.s L_2
```

Reflector в C# код не розбирає (хоча це зробити нескладно), але в IL – відмінно:

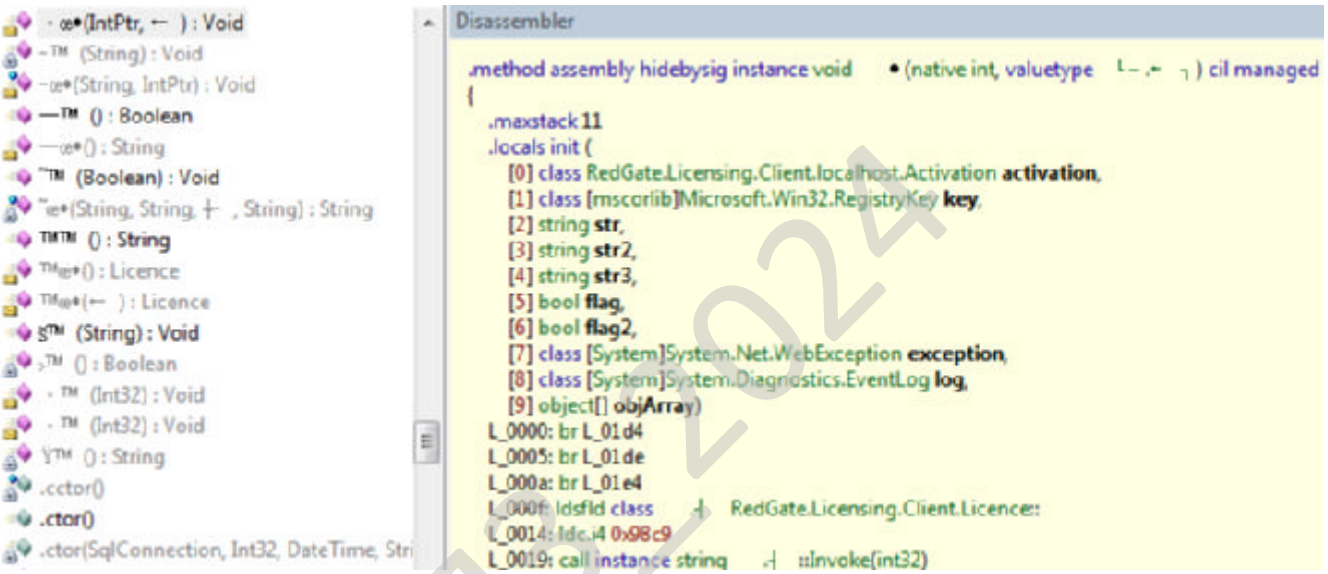

Рисунок 2.2 – Інтерфейс користувача {SmartAssembly}

Salamander.NET

Скачав приклад з їхнього сайту. Обіцяли, що Reflector не відкриє. Reflector, насправді відкрив.

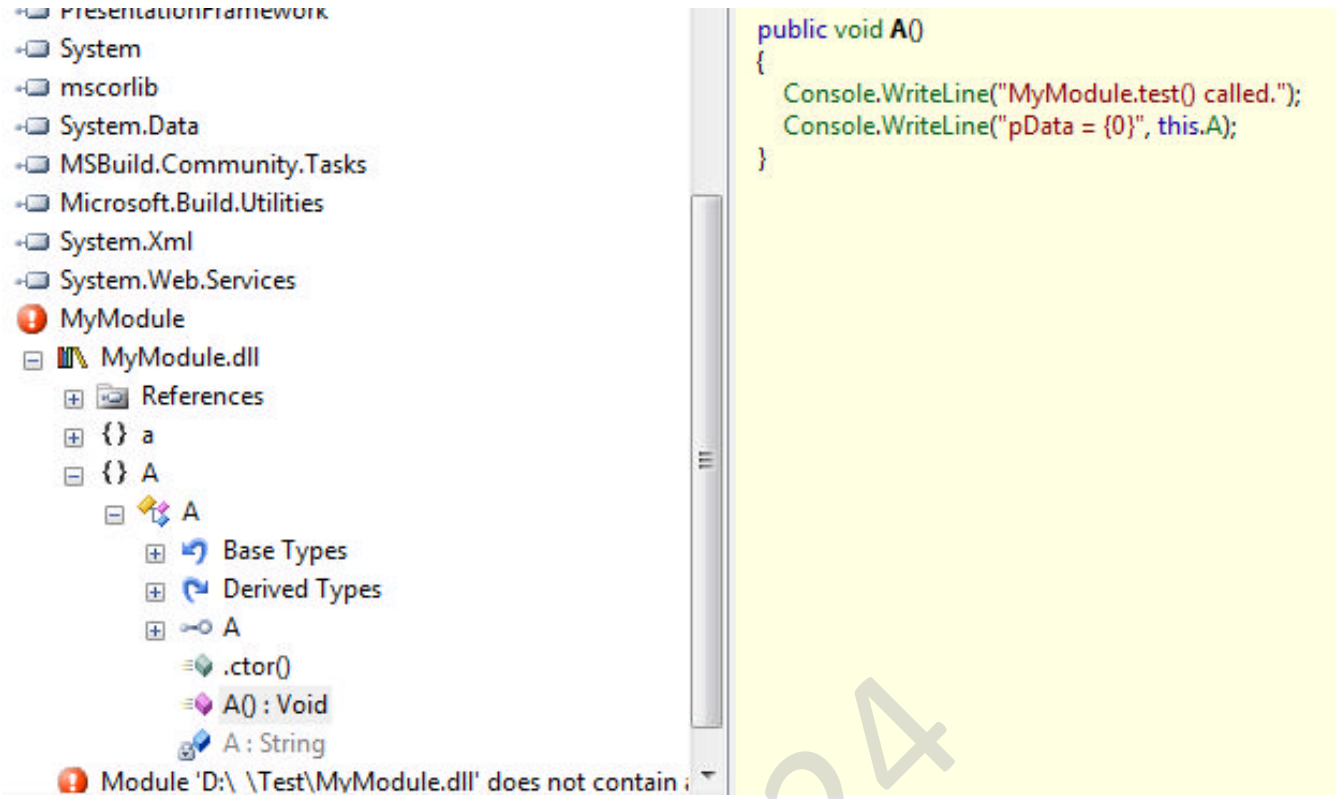

Рисунок 2.3 – Інтерфейс користувача Salamander.NET

Babel, CodeWall, dotNetProtector

Продукти одного класу, я вважаю, що досить підходять, якщо потрібна захист коду від подальшого рефакторингу й зайвої цікавості.

Однак, WinDbg з успіхом показав, що усередині збірки, зробленої Babel (імовірно, тому, що в мене була community версія; повну, на жаль, я не перевіряв).

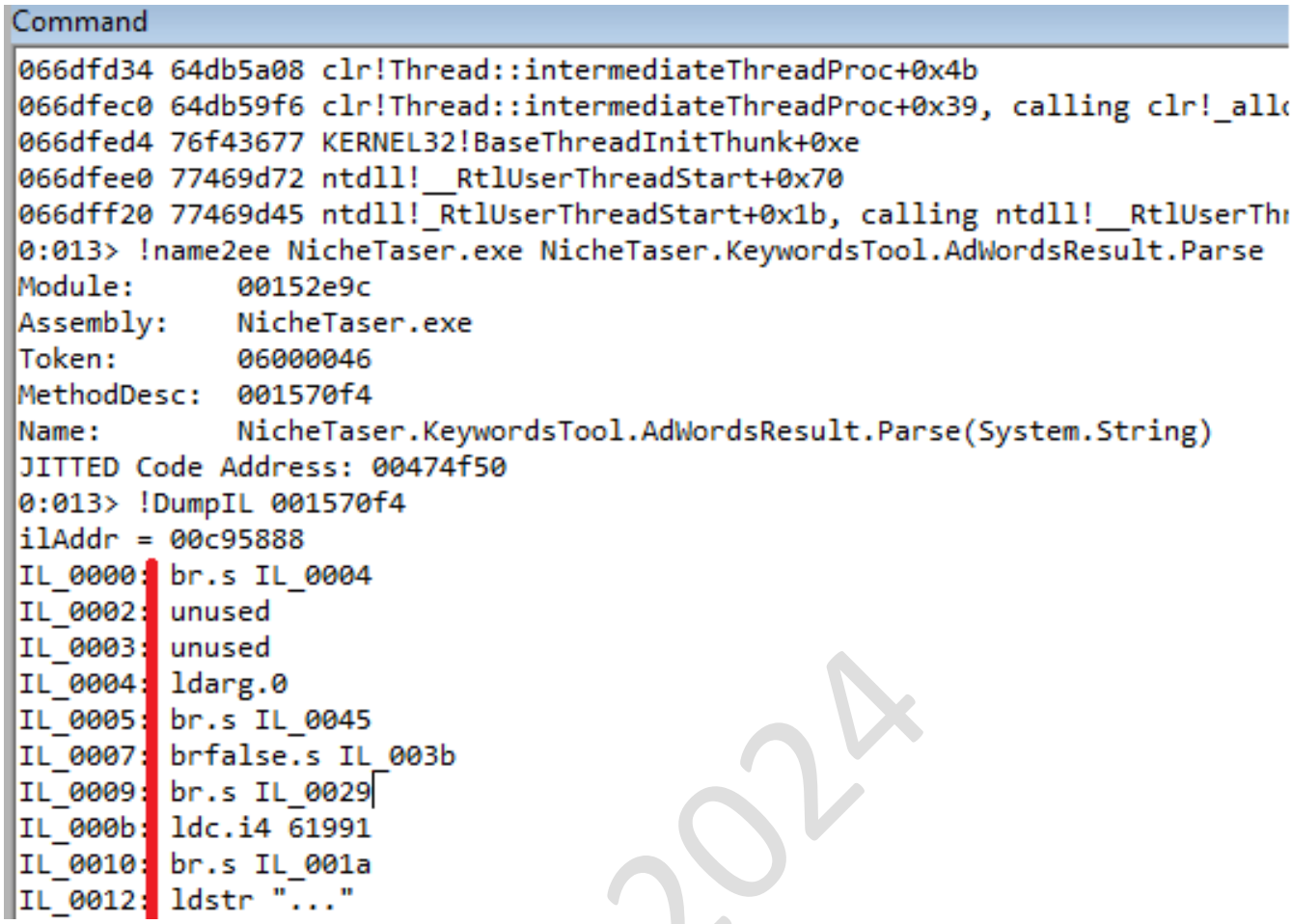

Рисунок 2.4 – Інтерфейс користувача Babel

C# Source Code Obfuscator

На сайті пишуть, що працює з вихідним кодом, при цьому виходить заобфускований вихідний код. Дуже цікавий підхід, але на жаль, скачати сам обфускатор не можна. Мінус – при такому підході неможливе шифрування MSIL і невірні інструкції.

XHEO CodeVeil

На попередню версію цього обфускатора було багато зломщиків, які з'являлися дуже швидко. Що говорить про те, що хакери знають, як працює цей обфускатор. На останню версію зломщика я не бачив, але є негарні проблеми:

– Додатки не подобаються антивірусу, обфускатор зашифровує збірку й записує в себе (що радує, збірка шифрується по частинам)

– Після цього обфускатора, додаток треба дуже добре тестувати, помилки можуть виникнути в самих несподіваних місцях.

Використовувати можна, але чекайте сюрпризів.

dotNetProtector

Збірка вийшло досить гарна, але зі збиранням прийшли 4 МБ DLL-ек від цього обфускатора:

```
public string ò
{
  [MethodImpl(MethodImplOptions.NoInlining)]
  get
  {
    throw new ApplicationException();
  }
}
```

Spices.NET

Досить відомий продукт, я був здивований, що отримані додатки так просто зламати.

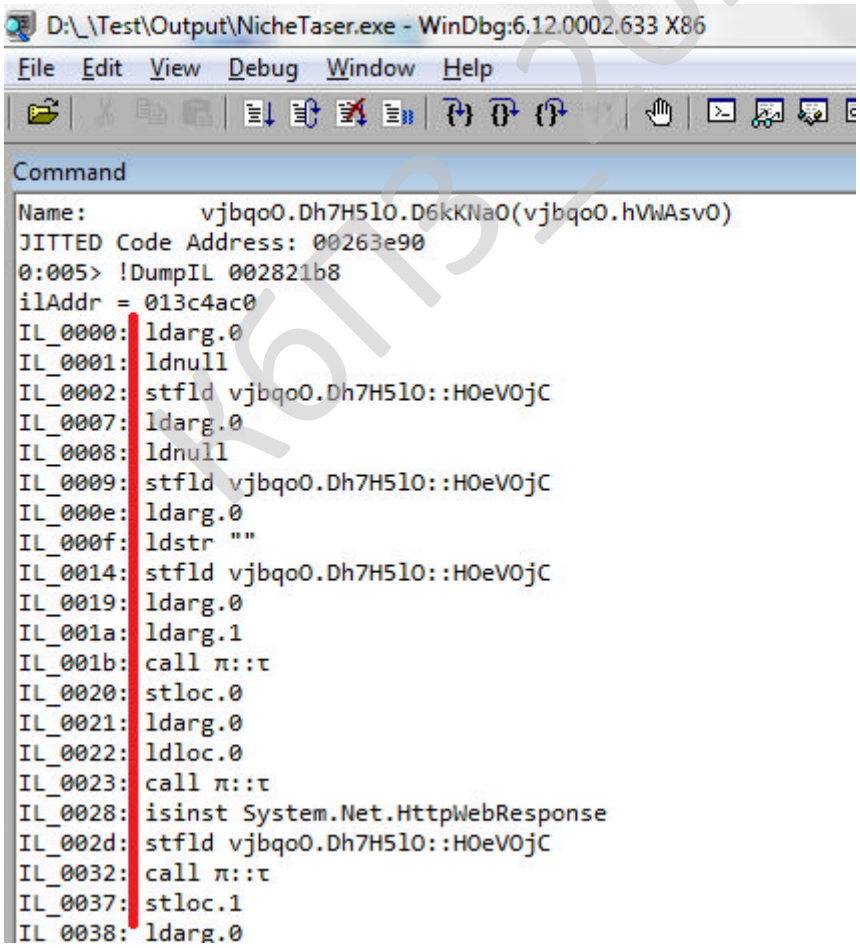

Рисунок 2.5 – Інтерфейс користувача Spices.NET

Goliath.NET

Рефлектором збірку не відкрити, воно зашифровано. Проблема в тому, що збірка зашифрована повністю, і розшифровується воно після того, як запуститься додаток. Після розшифровки ніякого захисту в пам'яті.

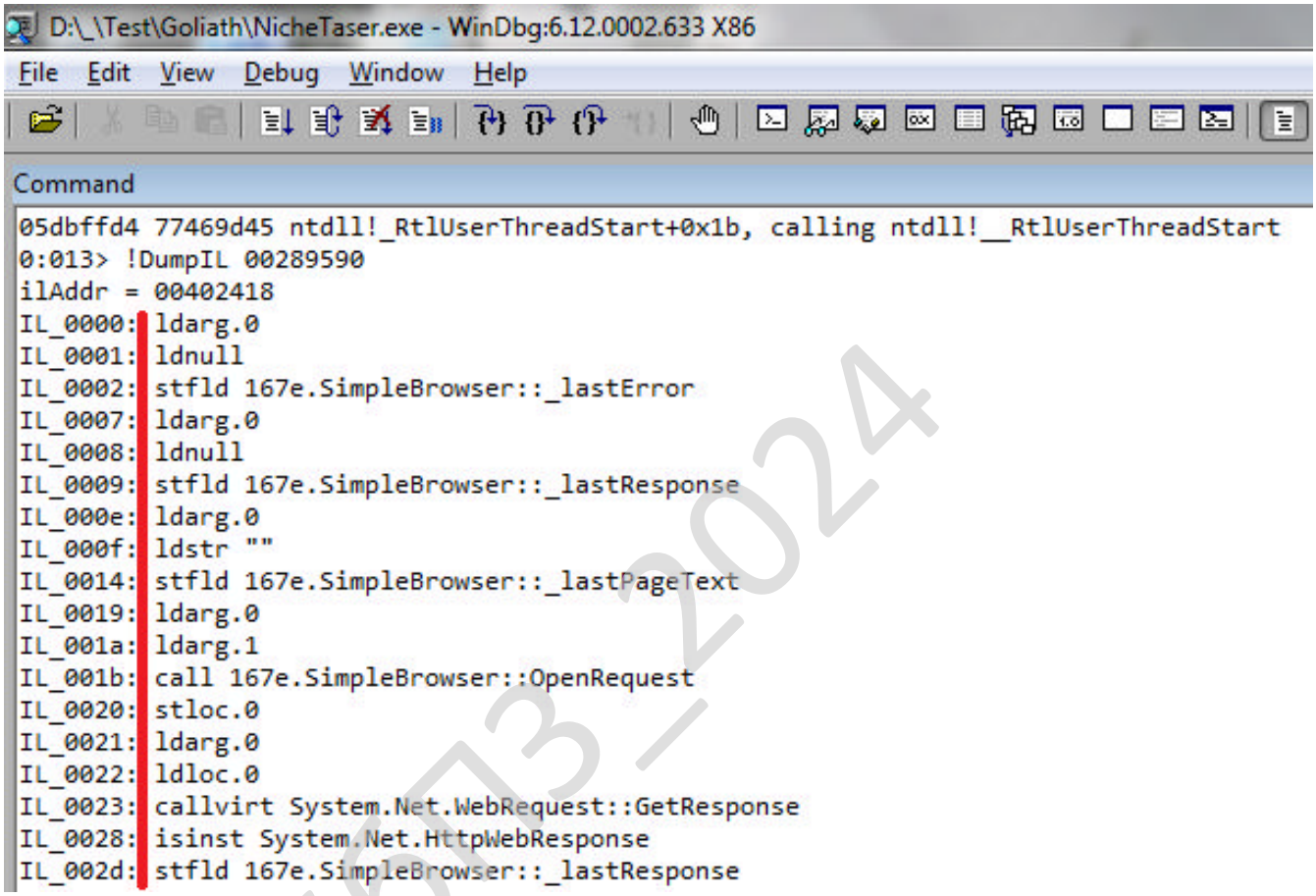

Рисунок 2.6 – Інтерфейс користувача Goliath.NET

Eazfuscator.Net

Обфускатор безкоштовний. Вміє робити тільки простий rename. Що може порадувати (особисто мене більше цікавлять консольні або MSBuild-версії) – досить простий процес обфускації, усе зводиться до перетаскування файлу збірки. От що виходить у підсумку.

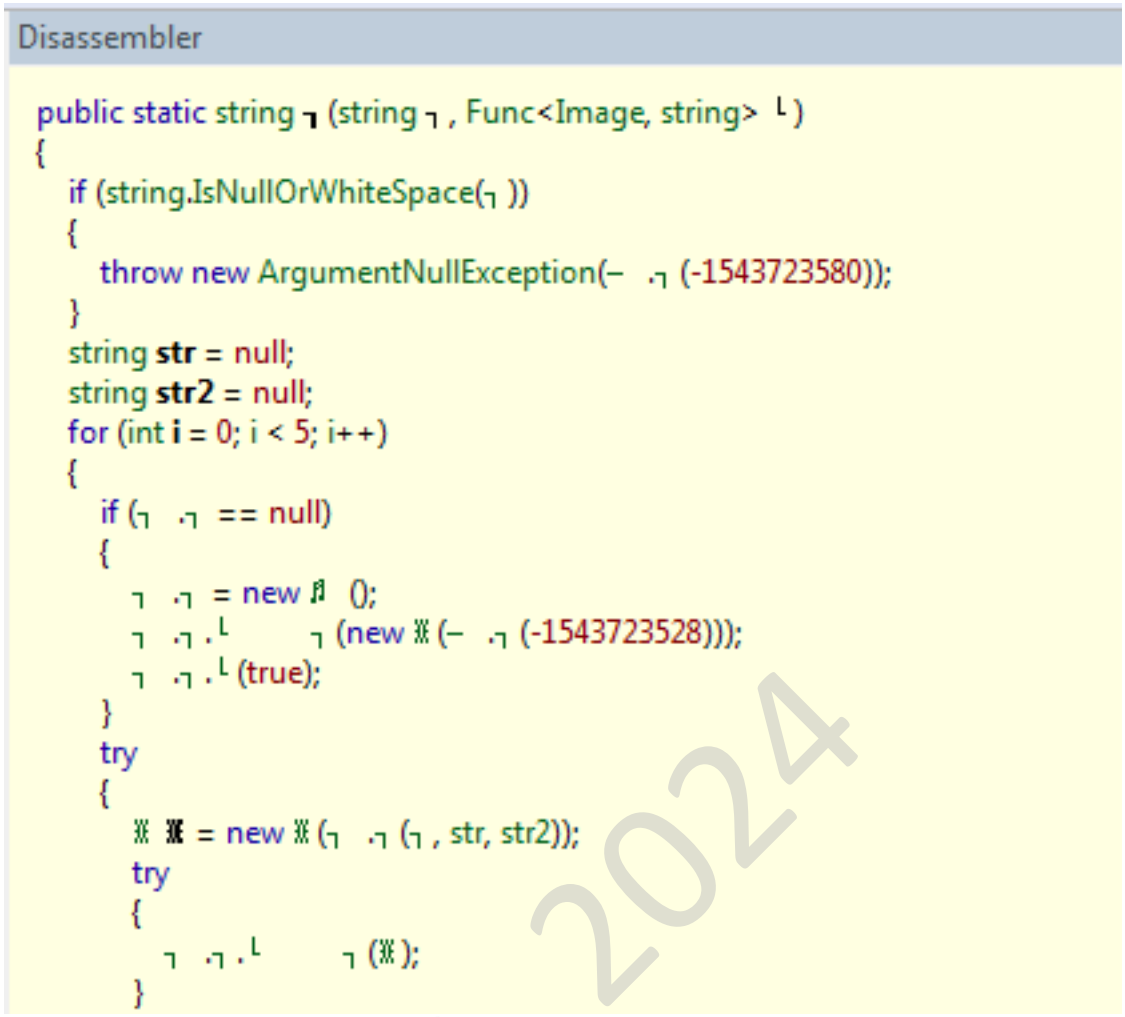

Рисунок 2.7 – Інтерфейс користувача Eazfuscator.Net

VMWare ThinApp, Xenocode PostBuild

Обфускатори побудовані по схожому принципі, уміють виконувати код, впроваджуючи в додаток передкомпьовані збірки .NET, що виключає ймовірність перехоплення викликів JIT-компіляції.

Отримані додатки можуть запускатися навіть без установленого .NET на машині. Розмір додатка, що вийшов – 10..50 МБ, залежно від того, які бібліотеки будете використовувати.

Коштують ці рішення дуже дорого. Але, на жаль, на PostBuild ходять зломщики (навіть на останній). Імовірно, у відомих колах є й готові розпаковники.

Вибір обфускатора залежить від того, що для тебе є цінністю в коді:

1. Весь код у сукупності – важлива не одна «функція» у коді, а код повністю. Зашифруй код яким-небудь простим обфускатором, краще зашифрувати MSIL. Якщо важливо дійсно весь код, розшифрувати його повністю буде складніше написання заново, і ніхто цим не буде займатися.

2. Окрема «функція» – наприклад, перевірка ключа. Я б порадив такий код взагалі в загальний доступ не давати, краще різати функціонал в trial-версії. У повній версії ключ перевірити обов'язково, але ризик злодійства менше. Проте, я б порадив використовувати обфускатор більш серйозний.

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 для запобігання аналізу та модифікації програмних продуктів.

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

Що робить обфускатор – аналізує метадані, тому що не всі члени збірки він може обфусцирувати.

Може бути модифікований ще один розділ (User Strings або #US) – але поки не зустрічав таких обфускаторів. Відкриваємо в ildasm або Reflector або в іншому навігаторі по збираннях і оцінюємо результат. Деякі обфускатори мають додаткові функції: заплутування namespaces (зміна приналежності різних класів певним namespace), шифрування строкових і графічних ресурсів, контроль обфускації на базі спеціальних атрибутів, якими позначаються члени класів у вихідному коді, деякі намагаються варіювати MSIL-код методів збірки.

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

Продукт завантажує з ресурсу потрібні збірки до пам’яті й managed exe файл і передає йому керування, займаючись тільки рішенням проблем зі збираннями, типами, ресурсами (через AssemblyResolve, TypeResolve, ResourceResolve). Але – не всі збірки потрібні відразу, завантаження їх вимагає дешифрування й розпакування – додаткового часу й навантаження на процесор. Не всі роблять exe-файли. А Thinstall буде працювати тільки з exe, тому що dll-збірки вже не мають як раніше процесорного DllMain, з якого це було можливо робити. Але – зламати її проблем також не становить праці. Є така програма ProcDump – вона може продампить запущений процес і відповідно легко одержати розшифрованими й розпакованими захищений exe і referenced збірки. Thinstall буде мати проблеми із завантаженням до пам’яті managed C++ збірки.

Деякі обфускатори створюють замкнуту систему обфусцированих збирань, де необфусцированими залишаються збірки сторонніх виробників і MS-вські. Але й тут все далеко від досконалості – обфускатор не знає як використовуються ресурси збірки, тому може привести вашу програму до неробочого стану. Хоча в цьому випадку якість обфускації стає ідеальним – ні до чого не підкопатися. Знову ж – це можливо тільки для exe-програм. Якщо вам необхідно захищати бібліотеку класів, то вам необхідно буде залишати необфусцированими публічні члени класів і публічні класи – звідси зломщик може почати свою роботу.

Початок роботи зі створення обфускатора

Робота по створенню обфускатора починається із проектування системи (програми або бібліотеки класів), що ви збираєтеся написати. Так, якщо ви хочете написати максимально захищений код, вам доведеться враховувати ряд факторів при проектуванні вашого продукту.

Не варто плутати проектування із програмуванням. Які фактори необхідно враховувати. Обфускатор, усього не може, тому варто піти його можливостям назустріч. Типовою обфускацією є символьна (з огляду на всі плюси й мінуси згадані вище) – коли обфускатор тільки й усього змінює назви типів, полів, методів, властивостей і подій на безглузді. Скажемо тип Obfuscator перейменовується в 0, а його метод Run() – теж в 0, а параметри методів просто перенумеруються – 0,1,2,3,4. Після подібної обфускації губиться логічний зв'язок між класами, дизассембльований код – тяжкочитаемий.

Якщо ви збираєтеся обфусцирувати замкнуту систему – тоді вам підійде повна обфускація, коли змінюються всі назви членів збірки, і в цьому випадку кінці знайти на порядок набагато сутужніше чим у випадку, коли деякі методи, типи у вас залишаються не обфусцированими (таке можливо при обфускації exe-файлів не використовують Reflection).

До речі кажучи, крім символьної обфускації є ще обфускація алгоритмів методів – коли найпростіше множення I*3 може бути представлене більш

складним алгоритмом, наприклад – I*((1+1/2)*2) або заплутані while і for. Але ми будемо говорити тільки про символьний обфускації, стосовно до обфускації .Net збирань.

Рідко можна зустріти повністю замкнуті, автономні системи, тому типовим випадком буде часткова обфускація. Хоча ідеальним завданням для більше якісної обфускація буде саме написання максимальної замкнутої системи збирань. От вам і перший фактор.

Рекомендації з підготовки проекту(продукту) до обфускації:

1. Для ускладнення дизасемблювання ви можете використовувати підміну типів (яку не завжди можливо реалізувати через sealed типи), через такий підхід – якийсь системний тип SomeType успадковується в новому типі AnotherType, що підходить для обфускації (скажемо, він лежить у межах видимості вашої збірки, його можна призначити як internal) і ви його спокійно обфусцируєте. На виході ми одержуємо використання SomeNamespace.0 замість відомого System.SomeType. Звичайно можна встановити що AnotherType – це простий спадкоємець від SomeType, але на це піде час, чи не так? А нам і потрібно витрачати час зловмисника який бажає взломати наш код.

2. Якщо ви плануєте використовувати деякі типи як публічні але хотіли б їх максимально захистити, є сенс їх помістити у "захисну шкарлупу" спадкування. Якийсь тип public SomeType можна перетворити у два класи: internal _SomeType (underground class), що несе всю реалізацію класу, крім публічних властивостей, необхідних для серіалізації й для використання в зовнішньому для збірки коді, і public SomeType, що буде успадковуватися від _ SomeType (front class), але нести зовнішнє навантаження – мати публічні властивості, необхідні для серіалізації, конверсії, використання в зовнішньому для збірки коді. У такий спосіб у вас буде ще можливість скористатися підходом 1. для породження класу SomeType- подібних типів але наслідуваних від SomeType.

3. Уважніше з атрибутами. Не варто забувати про використання атрибутів у вашім коді. Багато які з них досить тісно допомагають взаємодіяти середовищу .Net з вашими класами. Наприклад атрибут TypeConverterAttribute – їм ви прив'язуєте до вашого класу клас конвертера SomeConverter. Не кожний обфускатор «знає» про це – і тому варто вберегти клас конвертера від обфускації або перевірити як обфускатор працює з атрибутами. Інакше зв'язок, установлений між двома класами за допомогою атрибута може бути зруйнована.

4. Якщо ваш клас іде під ніж обфускації, варто також задуматися про механізм його серіалізації. Скажемо клас-спадкоємець Form серіалізує себе в такий спосіб що якщо обфускатор змінив ім'я його типу SomeForm на 0, то виникне проблема при ініціалізації десеріалізації такого класу – він просто не зможе знайти ресурс 0.resources, тому що серіалізувався в в SomeForm.resources.

5. Використовуйте static string оголошення замість const string – це утруднить пошук ініціалізації цього поля (у метаданих обоє оголошення будуть представлені як поля).

6. Якщо ви маєте список рядків, які у вас представлені як список строкових констант, краще список рядків оголосіть/ опишіть як строковий масив, а в константах зберігаєте індекс до необхідного рядка в строковому масиві.

7. Якщо ви хочете захистити якийсь алгоритм від зайвого перегляду – віддайте його виконання декільком класам, цим ви розподілите завдання, може бути розвантажите пам'ять, але безумовно утрудните завдання дослідника зрозуміти, що у вас отут відбувається.

8. Користуйтеся такими функціями як nested types, це не вплине на продуктивність, але вплине на «тяжковивчаємість» вашого коду.

9. Якщо говорити про захист алгоритмів, то варто перекласти їхнє виконання не одному методу, а частини алгоритму передавати у виконання різним класам, тобто фактично виконання алгоритму буде взаємодією декількох класів.

10. Ініціалізуйте клас форми без використання .resx і .resources файлів – до них краще звертатися по індексі щоб уникнути проблем при обфускації ресурсів.

Після обфускації:

– Обов'язкове тестуйте обфусцировані збірки. Обфускатор усього лише робить свою роботу, а ви – свою. Тому не варто зневажати тестами збірки після обфускації.

– Після обфускації, обов'язково перевірте збірку утилітою peverify, що йде з .Net Framework SDK – ця утиліта перевіряє метадані вашої збірки на коректність. Якщо ваша збірка позначене як CLS-compliant – це тестування обов'язково. Є не відповідальні програмісти, які некоректно обфусцировану збірку прикривають спеціальним дозволом з PermissionSet.SkipVerification для запобігання зустрічі результуючої збірки з її верифікацією. Перевірте – чи не з'явився такий дозвіл у вашім збиранні після обфускації, зрозуміло на що натякаю. Правда managed C++ збірки мають споконвічно такий дозвіл, навіть якщо ви про нього не згадували у вашім коді.

– Небагато про якість обфускації – не полінуєтеся, подивитеся як обфусцирован сам обфускатор. Якщо обфускатор написаний в native code – можливо автори сумніваються в якості своєї обфускації?

– Створіть тестовий проект, що швиденько протестує вашу збірку, але поки обфускатори не мають можливість протестувати вашу збірку на функціональність, лише одиниці взагалі перевіряють отриману збірку хоча б на завантаження.

Так що ж є найкращим для захисту .Net збирань? На сучасний момент сполучення протектора (software protection є через) і обфускатора й повна обфускація проекту (крос-обфускація) дають найкращий захист. Звичайно варто порекомендувати обфусцирувати проект як замкнуту систему, де всі типи, використовувані проекті – обфусцировані, це можливо. Тому що зломом займається усі, а зломом добре захищених програм – одиниці, те поки єдино можливим є спільне використання цих двох видів продуктів.

3.2 Розробка структурної схеми

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

Вона складається з:

1. Блок захисту від аналізу та модифікації, який включає в себе:

– Блок вибору алгоритму та функцій захисту від аналізу та модифікації, який обирає Алгоритм Ченга Вонга або Алгоритм Колберга.

– Блок вибору функції захисту від аналізу та модифікації.

2. Код програми, яку потрібно підвергнути захисту від аналізу та модифікації.

3. Блок визначення рівня захисту від аналізу та модифікації.

4. Блок визначення мови програми.

5. Код програми, підвергнутий захисту від аналізу та модифікації.

ІТ-галузі щорічно витрачають мільярди доларів на запобігання атакам на безпеку, таким як підробка та зловмисне зворотне проектування. Через величезне застосування та розвиток інтернет-технологій і мультимедіа виникла величезна потреба в дослідженнях безпеки та захисту. Кожна організація має власну інтелектуальну власність, і для них є серйозним викликом захистити свої дані, наприклад, піратство програмного забезпечення або впровадження шкідливого коду тощо. Крім того, їхня обробка даних через програму є конфіденційною, тому її розкриття може завдати шкоди покупцеві програмного забезпечення. бізнес безпосередньо. Існує два загальних способи захисту інтелектуальної власності, юридичний або технічний. Юридично означає отримання авторських прав або підписання юридичних контрактів проти створення дублікатів тощо. А технічно означає, що власники програмного забезпечення нададуть рішення для захисту за допомогою цього конкретного програмного забезпечення. Раніше захист даних означав використання брандмауерів і шлюзів у самій операційній системі або в мережі. Але для захисту від сторонніх кращою ідеєю є

використання цих механізмів або методів у прикладному програмному забезпеченні. Одним із таких типів методів є обфускація, яка є новою областю досліджень у галузі захисту програмного забезпечення та набуває все більшого значення в цю цифрову еру.

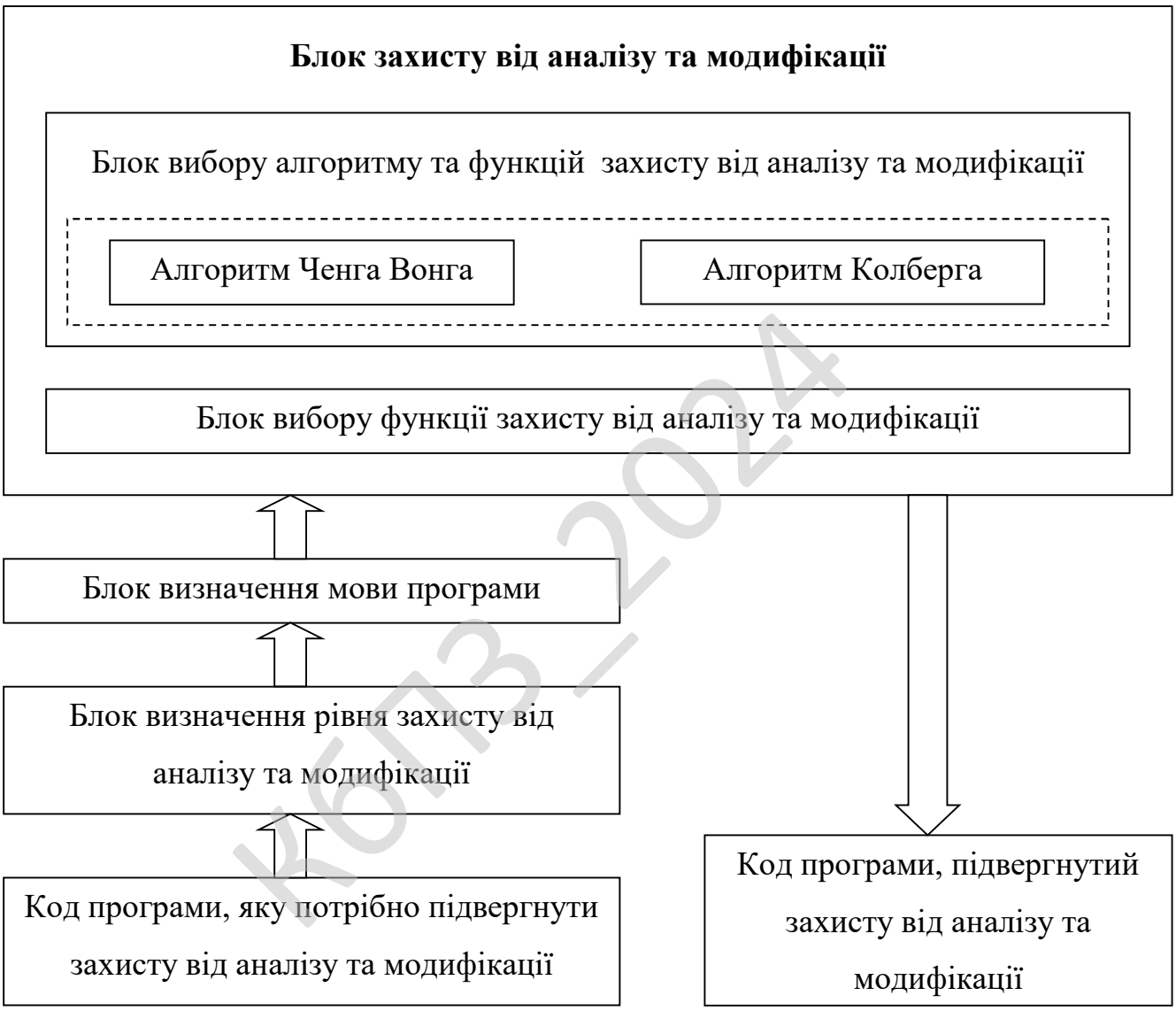

Рисунок 3.1 – Структурна схема системи

Обфускація складається з перетворень коду, які роблять програму більш важкою для розуміння, змінюючи її структуру, зберігаючи оригінальні функції, непридатні також для зворотного проектування. Шифрування та брандмауери є одними з поширених рішень для зменшення загрози зловмисників, які

намагаються зламати програму. Але ці підходи не допомагають захистити програмне забезпечення, якщо зловмисник сам є кінцевим користувачем. Серед різноманітних методів, доступних для захисту коду від різних атак, обфускація коду є однією з найпопулярніших альтернатив для запобігання розуміння коду, підробки коду тощо. Отже, обфускація коду є широко прийнятим рішенням, і було запропоновано багато різних підходів до обфускації. . Це також тип захисту програмного забезпечення від несанкціонованого зворотного проектування.

Однак рішучий зловмисник, витративши достатньо часу на перевірку заплутаного коду, може знайти функціональні можливості для зміни та досягти успіху у своїй зловмисній меті. З цієї причини методи обфускації реалізуються з іншими підходами, такими як заміна/оновлення коду, виявлення фальсифікації коду, оновлення захисту (таким чином зловмисники отримують обмежену кількість часу для виконання своєї мети) тощо. Практично шифрування, захист сервером Рішення безпеки на основі апаратного забезпечення, різні підписані рідні коди, захист від втручання, водяні знаки, застаріле програмне забезпечення, пакування є одними з найпоширеніших методів, щоб уникнути чи кинути виклик механізмам виявлення. Однак постачальник повинен оцінити, як довго обфускація буде протистояти, тобто час, який потрібен зловмиснику, щоб зрозуміти код. Згідно з цим, деякі інші методи обфускації можуть бути реалізовані на оригінальному коді, так що; супротивник не зможе отримати алгоритм чи логіку коду.

Крім того, методи обфускації включають зміну порядку коду, перетворення для заміни значущих імен ідентифікаторів у вихідному коді безглуздими випадковими іменами (перейменування ідентифікаторів), вставки небажаного коду, безумовні переходи, умовні переходи, прозору вставку розгалужень, перепризначення змінних, випадковий мертвий код, злиття локальних цілих чисел, кодування рядків, генерація фальшивого коду середнього рівня, придушення констант, зчеплення потоків керування та багато іншого.

По суті, обфускація багато в чому відрізняється від шифрування. По-перше, він не потребує зворотного перетворення. Крім того, зловмиснику не обов'язково весь час шукати вихідний код, оскільки атака може бути успішною, не маючи вихідного коду програмного забезпечення. І, нарешті, зашифрований текст буде марним без ключа, оскільки обфускована програма може працювати без додаткової інформації.

Якщо ми обговоримо інший аспект, здебільшого програмний код є портативним і розповсюдженим у мережах, яким також можна не довіряти. Отже, механізм захисту має бути включений у програмне забезпечення, але він також має бути незалежним від апаратного забезпечення. Основною функцією будь-яких методів захисту програмного забезпечення є виявлення піратського, пошкодженого чи неправильного використання коду чи програми. Виходячи з цього, передбачається, що «обфускація коду» є простим і головним інструментом захисту вихідного коду в області захисту програмного забезпечення та безпеки. Основна ідея цих методів обфускації полягає в тому, щоб приховати оригінальний код від зловмисника, оскільки код буде трансформовано, але його функціональність буде подібною до оригінального коду; але набагато складніше проаналізувати чи зрозуміти.

Для аналізу коду потрібні будь-які дизассемблери або декомпілятори для використання у виконуваному коді. Але очевидно, що розібраний код не буде схожий на оригінальний код, оскільки неможливо повернутися з усіма тими ж функціями. Оскільки більшість методів аналізу коду були досліджені та експериментовані на Ассамблері код або з кодом низького рівня; тому в цій статті розглядається програмування на рівні складання. У наступному розділі обговорюються кілька методів обфускації.

Відповідно до Collberg та інших [1], обфускація коду відноситься до класу методів, які перетворюють вихідну програму в цільову програму, так що обидві програми мають однакову поведінку, але цільову програму важко змінити будь-яким зловмисником. Наведені вище спостереження спонукали нас розробити

схеми обфускації, які захищають чутливі фрагменти коду, задовольняючи всім критеріям ефективності, згаданим у [1]. Також можна комбінувати більше одного методу обфускації, щоб ми могли досягти кращої міцності. Ідея пішла в бік обфускації через наступні зауваження.

Обфускатоване прикладне програмне забезпечення ніколи не переривається через мережеві обмеження. Крім того, для шифрування чи дешифрування коду не потрібно жодного обладнання. Таким чином, для автентифікації програмного забезпечення не вимагається цифровий підпис, незалежно від того, має програма захищений код із надійного джерела чи ні.

– Для ефективної роботи методів шифрування потрібне певне апаратне забезпечення з підписаним кодом, який накладає певні обмеження, які не повинні залежати від апаратної платформи. Це використання спеціального апаратного забезпечення також призводить до витрат для покупця програмного забезпечення.

– Декомпіляція обфусцованого коду буде важчою, навіть якщо витратити достатньо зусиль і часу. Але це можна повернути алгоритми та структури даних. Таким чином, основний намір полягає в тому, щоб збільшити час і зусилля, завдяки чому для зловмисника буде фактично неможливо виконати зворотне проектування заплутаного коду або програмного забезпечення.

– За допомогою обфускації можна заплутати код і усунути більшість логічних зв’язків, так що трансформований код стає досить складним для аналізу та несанкціонованих змін.

– Існує так багато складних алгоритмів шифрування, які неможливо реалізувати через обмеження пам’яті та пропускної здатності мережі.

– В останній прогрес теорія обфускації все ще потребує оцінки якості на практиці заплутування перетворень швидшим і простішим способом.

Зібравши разом деякі з вищезазначених аспектів у техніках захисту програмного забезпечення, можна вважати, що концепція «заплутування коду»

може бути сильнішим, а також кращим інструментом для захисту програмного забезпечення.

Щоб проаналізувати код, дуже важливо знайти порядок, у якому виконуються всі інструкції. Цей порядок виконання можна виявити за допомогою графа потоку керування. Як правило, якщо граф потоку керування складний, це означає, що код програми також складний. Але важливий момент полягає в тому, що граф потоку керування корисний, якщо в програмі доступні цикли та умовні оператори. В іншому випадку, якщо в коді немає умовних операторів або циклів, то весь код буде розглядатися як один блок. У цьому документі згадані методи обфускації коду будуть корисні для частини коду (якщо немає умовного оператора) або коду в базовому блоці. Отже, вважається, що наші методи обфускації можуть бути реалізовані до тих частин коду, для яких створення графа потоку керування неможливе або немає доступних умов.

Незважаючи на те, що цілі методи, засновані на програмному забезпеченні, не можуть забезпечити ідеальний захист, ми передбачаємо, що згадані вище методи значно ускладнять зворотне проектування, а також виявлення на основі сигнатур і виявлення коду через зіставлення шаблонів, особливо коли ці методи будуть поєднані з існуючі методи обфускації коду.

3.3 Розробка функціональної схеми

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

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

1. Лексична обфускація:

– Видалення всіх коментарів у коді програми.

– Видалення різних пробілів.

– Заміна імен ідентифікаторів.

– Додавання різних зайвих операцій.

– Зміна розташування блоків.

2. Обфускація даних.

а) Обфускація зберігання:

– Зміна інтерпретації даних певного типу.

– Зміна строку використання сховищ даних.

– Перетворення статичних даних у процедурні.

– Поділ змінних.

– Зміна подання (або кодування).

б) Обфускація з'єднання:

– Об'єднання змінних.

– Реструктурування масивів.

– Зміна ієрархій спадкування класів

в) Обфускація переупорядкування.

3. Обфускація керування.

а) Обчислювальна обфускація:

– Розширення умов циклів.

– Додавання недосяжного коду.

– Усунення бібліотечних викликів.

– Додавання надлишкових операцій.

– Розпаралелювання коду.

б) Обфускація з'єднання:

– Вбудовування функцій.

– Добування функцій.

– Чергування, об'єднання фрагментів коду програми.

– Клонування.

– Трансформація циклів.

– Розгорнення циклів.

– Поділ циклів.

в) Обфускація послідовності.

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

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

Обфускація даних

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

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

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

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

– Перетворення статичних (незмінюванних) даних у процедурні. Більшість програм, у процесі роботи, виводять різну інформацію, що найчастіше в коді програми представляється у вигляді статичних даних таких як рядка, які

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

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

– Зміна подання (або кодування).

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

– Об'єднання змінних. Дві або більше змінних "v1,...,vk" можуть бути об'єднані в один змінну "V", якщо їхній загальний розмір ("v1,...,vk") не перевищує розмір змінної "V". Наприклад, розглянемо простий приклад об'єднання двох коротких цілочисельних змінних "X","Y" (розміром 16 біт) в одну цілочисельну змінну "Z" (розміром 32 біта). Для цього скористаємося формулою: Z(X,Y) = 2^16 * Y + X, що дозволить, зневажаючи додаванням, визначати значення Y, тобто нехай X = 12, Y = 4 => Z = 65536 * 4 + 12 = 262156,

тепер знаючи "Z" для знаходження "Y" можна 262156 / 65536 = 4.000183105 або приблизно 4. При здійсненні арифметичних операцій над значеннями змінних "X", "Y" зберігаються в "Z" потрібно враховувати вище наведену формулу.

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

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

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

Лексична обфускація

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

Обфускація такого виду містить у собі:

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

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

Рисунок 3.2 – Функціональна схема системи

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

– Додавання різних зайвих (сміттєвих) операцій.

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

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

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

Обфускація керування

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

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

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

Позначимо непрозорий предикат, що повертає завжди значення TRUE як "P(t)", а повертаючий значення FALSE, як "P(f)", тоді непрозорий предикат, що може повернути кожне із цих двох значень (тобто або TRUE, або FALSE, що нам

невідомо) як "P(t,f)". Ці позначення, будуть використовуватися далі в контексті опису обфускації керування. Непрозорі предикати можуть бути:

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

– Глобальними – обчислення втримуватися усередині однієї процедури (функції.

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

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

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

(указувати) на покажчик q" і таке інше. Ці умови в наслідку дозволяють створювати необхідні непрозорі предикати.

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

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

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

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

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

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

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

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

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

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

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

– Клонування, даний метод дозволяє ускладнити аналіз контексту використання функцій, і об'єктів використовуваних у коді вихідної програми. Процес клонування функцій складається у виділенні певної функції "F", часто використовуваної в коді програми, після чого над кодом цієї функції

здійснюється трансформація, і створюється її клон "F'", що також буде доданий у код вихідної програми, при цьому частина викликів функції "F" у коді вихідної програми, буде заміщена на виклик функції "F'". У результаті цього в зловмисника створиться подання про те, що функції "F", і "F'" різні. Клонування об'єктів здійснюється аналогічним способом.

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

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

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

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

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

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

3.4 Розробка діаграми процесів

Діаграма взаємодії процесів системи, розробленої у результаті виконання бакалаврського проектування, наведена на рисунку 3.3. Першим процесом, який завантажується у системі є процес виведення головного вікна програми.

Він взаємодіє з наступними процесами:

– Процес відкриття коду захищеного від аналізу та модифікації.

– Процес завантаження сирцевого коду програми.

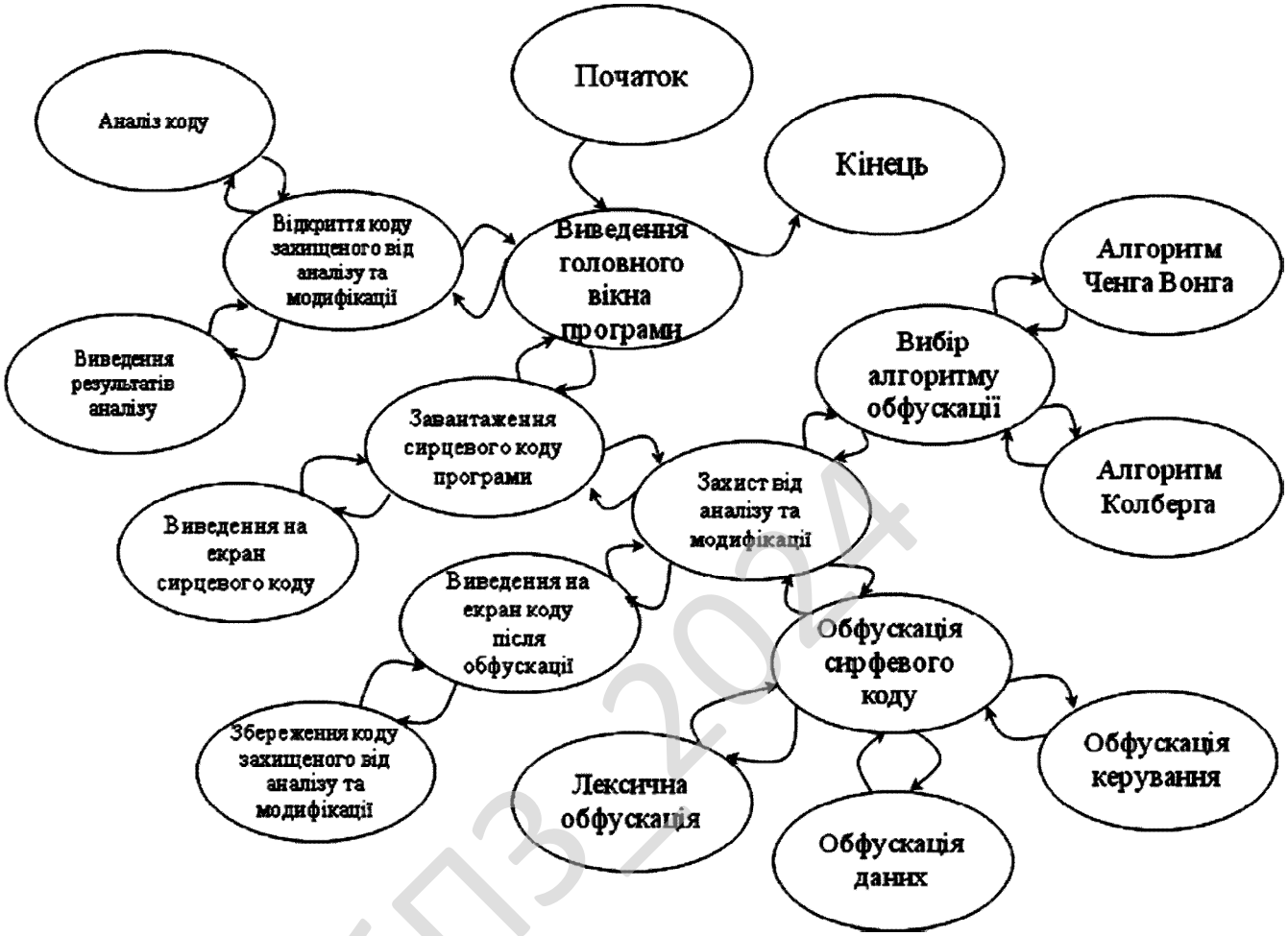

Рисунок 3.3 – Діаграма взаємодії процесів

Процес відкриття коду захищеного від аналізу та модифікації взаємодіє з наступними процесами:

– Процес аналізу коду.

– Процес виведення результатів аналізу.

Процес завантаження сирцевого коду програми взаємодіє з наступними процесами:

– Процес виведення на екран сирцевого коду.

– Процес захисту від аналізу та модифікації.

Процес захисту від аналізу та модифікації взаємодіє з наступними процесами:

– Процес виведення на екран коду після обфускації, який, у свою чергу, взаємодіє з процесом збереження коду захищеного від аналізу та модифікації.

– Процес обфускації сирцевого коду.

– Процес вибору алгоритму обфускації.

Процес обфускації сирцевого коду взаємодіє з наступними процесами:

– Процес лексичної обфускації.

– Процес обфускації даних.

– Процес обфускації керування.

Процес вибору алгоритму обфускації взаємодіє з наступними процесами:

– Процес алгоритму Чена Вонга.

– Процес алгоритму Колберга.

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

4 РЕАЛІЗАЦІЯ 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.

Спершу відбувається виведення основного вікна програми. Після цього відбувається відкриття та виведення сирцевого коду програми, яку потрібно захистити.

Для вибору алгоритму обфускації визначається мова програми сирцевого коду виходячи із притаманних їй лексичних та синтаксичних конструкцій. На основі цього здійснюється вибір алгоритму та функції обфускації, після чого програма приступає до здійснення обфускації коду.

Після завершення обфускації отриманий код виводиться на екран та пропонується його збереження на диск, а також виводиться звіт про виконані зміни.

Програма пропонує перевірити код, захищений від модифікації, на працездатність. Якщо програма, що підлягала обфускації, проходить перевірку, видається повідомлення про успішне проходження перевірки, в протилежному випадку, – видається повідомлення про помилку.

Алгоритми обфускації

Задачі заплутування і аналізу заплутаних програм мають три аспекти: теоретичний, що включає розробку нових алгоритмів перетворення графа потоку управління або трансформації даних програми, а також теоретичну оцінку складності їх аналізу і розкриття. Прикладний аспект включає розробку конкретних методів заплутування (розплутування), тобто якнайкращих

комбінацій алгоритмів, емпіричний порівняльний аналіз різних методів, емпіричний аналіз стійкості методів тощо.

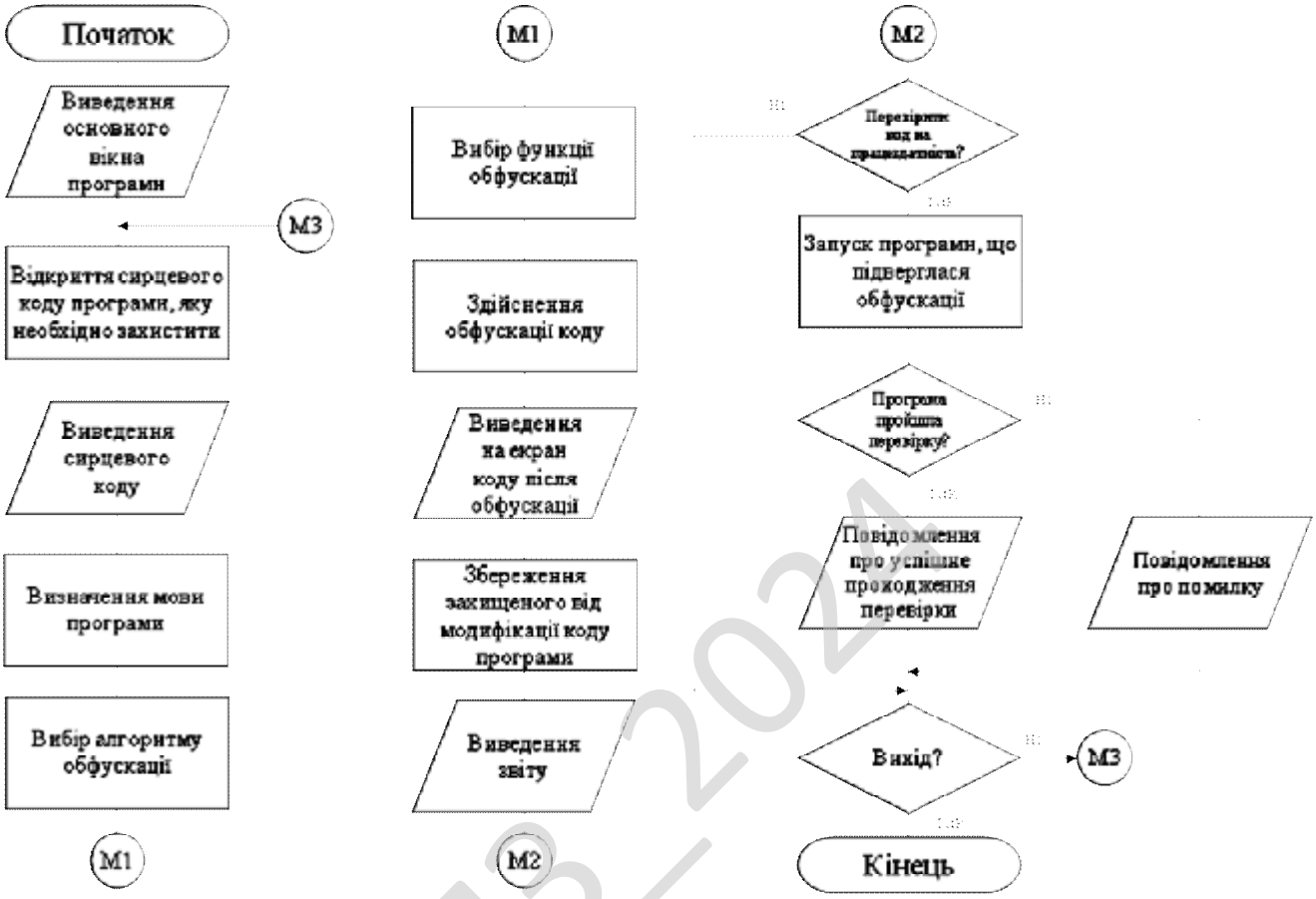

Рисунок 4.1 –Блок-схема основної програми

Третій аспект, психологічний поки не піддається формалізації, але не може ігноруватися. Зворотна інженерія (розуміння) програм – це процес, результатом якого є деяке знання суб'єкта, що вивчає програму, який є невід'ємною часткою процесу розуміння. Методи заплутування повинні максимально використовувати властивості (точніше, слабкості) людської психіки.

Ефективне обчислення – це обчислення, що вимагає поліноміального від довжини входу часу і поліноміальної від довжини входу пам'яті. Ефективна програма (машина Тюрінга) – програма, що працює поліноміальний від довжини

входу час і що вимагає поліноміальну від довжини входу робочу пам'ять на всіх входах, на яких програма завершується.

Нехай П – множина всіх програм (машин Тюрінга), що задовольняють сформульованим вище обмеженням, і нехай програма $p \in П$ обчислює функцію

$$f_p : Input \to Output,$$

підмножина $\pi \subseteq П$ називається функціональною властивістю, якщо

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

1. Нехай π – функціональна властивість, $P \subseteq П$ – клас програм такий, що існує ефективна програма *c* така, що для будь-якої програми $p \in P$

$$c(p) = \begin{cases} 1, & если\ p \in \pi \\ 0, & если\ p \notin \pi \end{cases}$$

Іншими словами, для функціональної властивості π ми визначаємо клас програм *P* таких, що існує ефективна програма-розпізнавач *c* властивості π за програмою *p* із класу *P* .

2. Імовірнісна програма *o* називається *обфускатором* класу *P* щодо властивості π,(*P*,π)-обфускатором, якщо виконуються умови:

а) (еквівалентність перетворення заплутування). Для будь-якої $p \in P$ та

$$p' \in o(p)$$
$$f_p = f_{p'},$$
$$|p'| = poly(|p|),$$
$$\forall x \in Dom_{f_p} \quad time_{p'}(x) = poly(time_p(x))$$

Тут *y=poly(x)* означає, що *y* обмежений поліномом деякого степеня від змінної *x, $time_p(x)$* – час виконання програми *p* на вході *x*, |p| – розмір програми *p*.

b) (складність визначення властивостей за заплутаною програмою). Для будь-якого полінома *q* і для будь-якої програми (ймовірнісної машини Тюрінга) *a* такої, що *a(o(P))={0,1}* , і для будь-якої $p \in P$ виконується *$time_a(o(p))=poly(|o(p)|)$*, існує програма (імовірнісна машина Тюрінга з оракулом) *b*, і при цьому для будь-якої $p \in P$.

$$|\Pr(a(o(p)) = c(p)) - \Pr(b^p(1^{|p|}) = c(p))| \le \frac{1}{q(|p|)}.$$

Іншими словами, ймовірність визначити властивість π за заплутаною програмою дорівнює вірогідності визначення властивості π тільки по входах і виходах функції f_p . Тобто, наявність тексту заплутаної програми нічого не дає для виявлення властивостей цієї програми.

Універсальний обфускатор – це програма O, яка для будь-якого класу програм P і будь-якої властивості π є (P, π)-обфускатором. Як показано в роботі, універсального обфускатора не існує. Доказ полягає в побудові спеціального класу програм P і виборі такої властивості π, що для будь-якого перетворення програми з цього класу властивість π встановлюється легко. Проте питання про те, чи існують обфускатори для окремих класів властивостей програм, і наскільки широкі і практично значущі ці класи властивостей, залишається відкритим. З практичної точки зору заплутування програми можна розглядувати як таке перетворення програми, яке робить її зворотну інженерію економічно невигідною. Не дивлячись на слабке теоретичну розробку, вже створена велика кількість інструментів для заплутування програм.

Заплутування перетворює програму, затрудняючи її зворотну інженерію. В результаті заплутана програма може виявитися більше за розміром і працювати повільніше. Вартість (cost) перетворення – це метрика, яка дозволяє оцінити, наскільки більше потрібно ресурсів (пам'яті, процесорного часу) для виконання заплутаної програми, ніж для виконання початкової програми. Вартість визначається за наступною шкалою: (безкоштовна, дешева, помірна, дорога).

Вартість перетворення дозволяє оцінити, наскільки збільшується розмір функції в результаті заплутування. Безкоштовне перетворення збільшує розмір функції на O(1), дешеве перетворення збільшує розмір на O(m), де m – розмір функції, помірне за вартістю перетворення збільшує розмір функції на O(m^p), де p>1. Нарешті, дороге перетворення експоненціально збільшує розмір заплутаної функції в порівнянні з початковою.

Вартість виконання дозволяє оцінити, наскільки більше потрібний ресурсів при виконанні програми. Вартість оцінюється як функція від характерного розміру вхідних даних n.

Перетворення оцінюється як безкоштовне, якщо виконання перетвореної програми $p`$ вимагає на O(1) більше ресурсів, чим виконання оригінальної програми. Перетворення оцінюється як дешеве, якщо виконання програми $p`$ вимагає на O(n) ресурсів більше, ніж виконання початкової програми, де n – розмір вхідних даних. Перетворення оцінюється як помірне за вартістю, якщо виконання програми $p`$ вимагає на $O(n^p)$ більше ресурсів, де p>1. Перетворення оцінюється як *дороге*, якщо виконання програми $p`$ вимагає експоненціально більше ресурсів, ніж виконання початкової програми. Практично застосовуватися можуть, мабуть, тільки безкоштовні і дешеві методи заплутування.

Розглянемо детальніше процес обфускації. На рисунку 4.2 приведена блок-схема алгоритму роботи підпрограми захисту програмного коду від аналізу та модифікації.

Після відкриття всіх складових модулів програми відбувається ініціалізація лічильника модулів.

Доки не будуть оброблені всі модулі програми в циклі виконуються операції обфускації:

- Видалення всіх коментарів в коді програми.
- Видалення різних пробілів.
- Заміна імен ідентифікаторів.
- Додавання різних зайвих операцій.
- Зміна розташування блоків.
- Зміна інтерпретації даних певного виду.
- Зміна рядків використання сховищ даних.
- Перетворення статичних даних у процедурні.
- Поділ змінних.
- Зміна подавання або кодування.

– Об’єднання змінних.
– Реструктуризація масивів.
– Зміна ієрархій спадкування класів.
– Розширення умов циклів.
– Усунення бібліотечних викликів.
– Додавання надлишкових операцій.
– Розпаралелювання коду.
– Вбудовування та додавання операцій.
– Чергування, об’єднання та клонування фрагментів коду програми.
– Трансформація, розгорнення та поділ циклів.

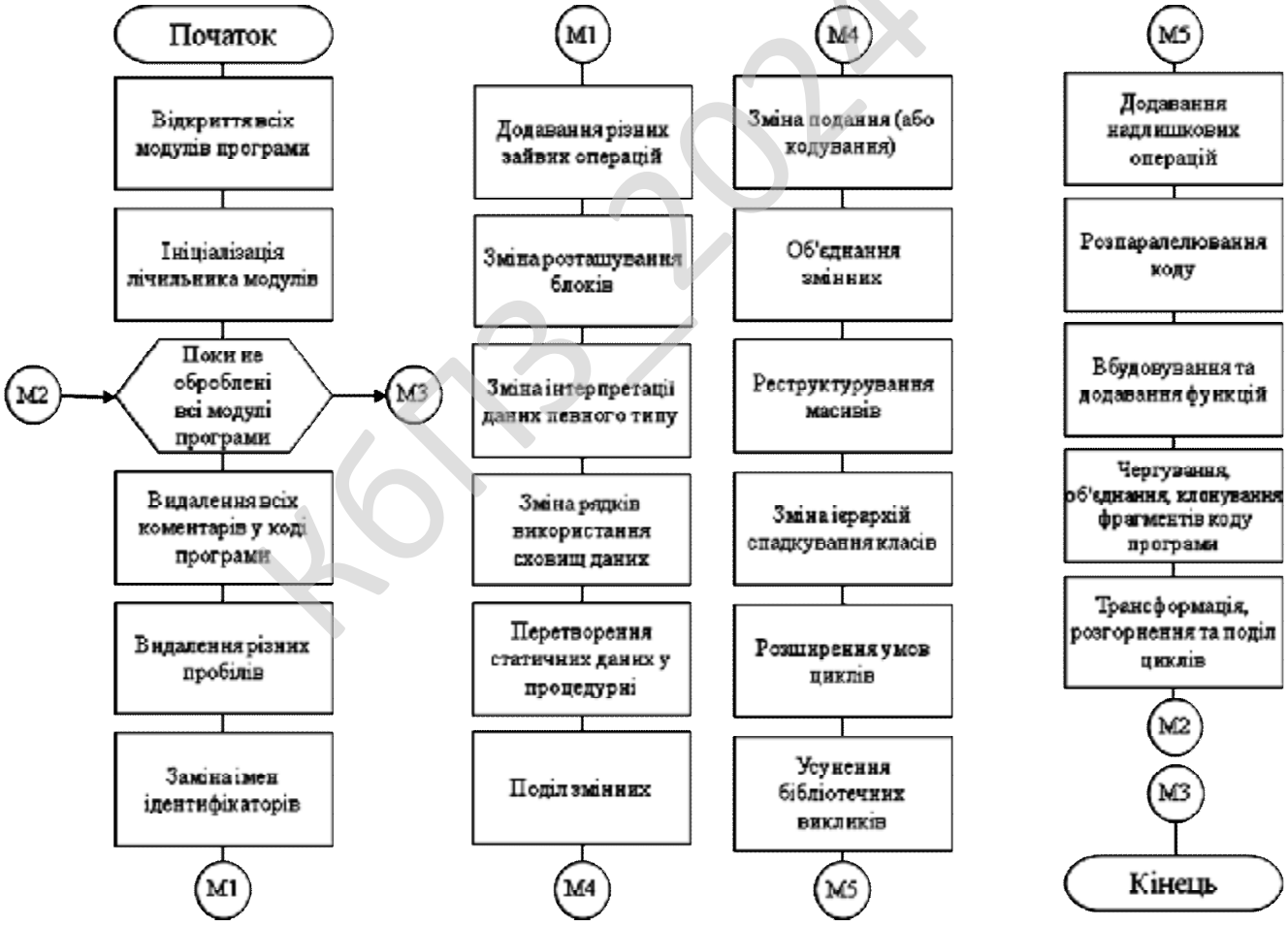

Рисунок 4.2 – Блок-схема алгоритму роботи підпрограми захисту програмного коду від аналізу та модифікації

Заплутуючі перетворення можна розділити на декілька груп залежно від того, на трансформацію якої з компонент програми вони націлені.

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

– Перетворення структур даних, що змінюють структури даних, з якими працює програма. До цієї групи відносяться, наприклад, перетворення, що змінює ієрархію наслідування класів в програмі, або перетворення, об'єднуюче скалярні змінні одного типа в масив.

– Перетворення потоку управління програми, які змінюють структуру її графа потоку управління, такі як розгортка циклів, виділення фрагментів коду в процедури, та інші.

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

Розглянемо перетворення форматування. До них відносяться видалення коментарів, переформатування програми, видалення відлагоджувальна інформації, зміна імен ідентифікаторів.

Видалення коментарів і переформатування програми застосовні, коли заплутування виконується на рівні початкового коду програми. Ці перетворення не вимагають тільки лексичного аналізу програми. Хоча видалення коментарів – однобічне перетворення, їх відсутність не ускладнює сильно зворотну інженерію програми, оскільки при зворотній інженерії наявність хороших коментарів до коду програми є швидше виключенням, чим правилом. При переформатуванні програми початкове форматування втрачається безповоротно, але програма завжди може бути переформатована з використанням якого-небудь інструменту для автоматичного форматування програм (наприклад, indent для програм на Сі).

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

Зміна імен локальних змінних вимагає семантичного аналізу (прив'язки імен) в межах однієї функції. Зміна імен всіх змінних і функцій програми окрім повної прив'язки імен в кожній одиниці компіляції вимагає аналізу міжмодульних зв'язків. Імена, визначені в програмі і не використовувані в зовнішніх бібліотеках, можуть бути змінені довільним, але узгодженим у всіх одиницях компіляції чином, тоді як імена бібліотечних змінних і функцій змінюватися не можуть. Дане перетворення може замінювати імена на короткі імена, що автоматично генеруються (наприклад, всі змінні програми отримають ім'я v<номер> відповідно до їх деяким порядковим номером). З іншого боку, імена змінних можуть бути замінені на довгі, але безглузді (випадкові) ідентифікатори з розрахунку на те, що довгі імена гірше сприймаються людиною.

Перетворення потоку управління змінюють граф потоку управління однієї функції. Вони можуть приводити до створення в програмі нових функцій. Коротка характеристика методів приведена 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 за перетвореною програмою автоматично відновити вставлені функції неможливо.

Винесення групи операторів (function outlining). Дане перетворення є зворотним до попереднього і добре доповнює його. Деяка група операторів початкової програми виділяється в окрему функцію. При необхідності створюються формальні параметри. Перетворення може бути легко обернене компілятором, який (як було сказано вище) може підставляти тіла функцій в точки їх виклику.

Відзначимо, що виділення операторів в окрему функцію є складним для запутуватача перетворенням. Обфускатор повинен провести глибокий аналіз графа потоку управління і потоку даних з врахуванням покажчиків, щоб бути упевненим, що перетворення не порушить роботу програми.

Приклад застосування у програмі приведено у наступній функції:

```
function
VirtAddrToPhysAddr(ANtHeaders:PImageNtHeaders;AVirtAddr:Pointer):Pointer;
var
 LI:Integer;
 LPSection:PImageSectionHeader;
 LAddr:Cardinal;
begin
 Result:=nil;
 LAddr:=Cardinal(AVirtAddr)-ANtHeaders^.OptionalHeader.ImageBase;
LPSection:=Pointer(Cardinal(@ANtHeaders^.OptionalHeader)+ANtHeaders^.FileHeade
r.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;
```

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

Змінна v є непрозорою, якщо існує властивість π щодо цієї змінної, яка апріорі відома у момент заплутування програми, але яку складно встановити

після того, як заплутування завершене. Аналогічно, предикат P називається непрозорим, якщо його значення відоме у момент заплутування програми, але його складно встановити після того, як заплутування завершене.

Непрозорі предикати можуть бути трьох видів: P^F – предикат, який завжди має значення "хиба", P^T – предикат, який завжди має значення "істина", і $P^?$ – предикат, який може набувати обидва значення, і у момент заплутування поточне значення предиката відоме.

Методи побудови непрозорих предикатів і змінних, засновані на "вбудовуванні" в програму обчислювально складних задач. Деякі можливі способи введення непрозорих предикатів і непрозорих виразів коротко перераховані нижче:

– Використання різних способів доступу до елементів масиву. Наприклад, в програмі може бути створений масив (скажімо, a), який ініціалізується заздалегідь відомими значеннями, далі в програму додаються декілька змінних (скажімо, i, j), в яких зберігаються індекси елементів цього масиву. Тепер непрозорі предикати можуть мати вигляд a[i]== a[j]. Якщо до того ж змінні i та j в програмі змінюються, методи статичного аналізу аліасів, що існують зараз, дозволять тільки визначити, що як i, так і j можуть вказувати на будь-який елемент масиву a.

– Використання покажчиків на спеціально створювані динамічні структури. У цьому підході в програму додаються операції по створенню посилальних структур даних (списків, дерев), і додаються операції над покажчиками на ці структури, підібрані так, щоб зберігалися деякі інваріанти, які і використовуються як непрозорі предикати.

– Побудова складних булевих виразів за допомогою еквівалентних перетворень із формули *true*. У простому випадку ми можемо узяти k довільних булевих змінних $x_1 ... x_k$ і побудувати з них тотожність $\left(x_1 \vee \overline{x_1}\right) \wedge \ldots \wedge \left(x_k \vee \overline{x_k}\right)$. Далі за допомогою еквівалентних алгебраїчних перетворень частина дужок (або всі) розкриваються, і в результаті виходить шуканий непрозорий предикат.

– Використання комбінаторної тотожності, наприклад, $\sum_{i=0}^{n} C_n^i = 2^n$.

Роботу методу покажемо на прикладі процедури DynCoder, за допомогою якої розбиваємо ключ на декілька блоків у пам’яті:

```
procedure DynCoder(AAddr:Pointer;ASize:Cardinal;AKey:Pointer); assembler;
stdcall;
asm
 @Coder_begin:
  push edi
  push esi
 @Coder_main_loop:
  mov edi,[ebp+008h]
  mov ecx,[ebp+00Ch]
  shr ecx,002h
 @Coder_pre_code:
  mov esi,[ebp+010h]
 @Coder_code:
  mov eax,[esi]
  test eax,0FF000000h
  jz @Coder_pre_code
 @Coder_do_code:
  add eax,ecx
  xor eax,[edi]
  stosd
  inc esi
  loop @Coder_code
 @Coder_end:
  pop esi
  pop edi
  leave
  ret 00Ch
end;
```

Внесення недосяжного коду (adding unreachable code). Якщо до програми внесені непрозорі предикати виду P^F або P^T, вітки умови, відповідні умові "істина" в першому випадку і умові "хиба" в другому випадку, ніколи не виконуватимуться. Фрагмент програми, який ніколи не виконується, називається *недосяжним* кодом. Ці вітки можуть бути заповнені довільними обчисленнями,

які можуть бути схожі на дійсно виконуваний код, наприклад, зібрані з фрагментів тієї ж самої функції. Оскільки недосяжний код ніколи не виконується, дане перетворення впливає тільки на розмір заплутаної програми, але не на швидкість її виконання. Спільне завдання виявлення недосяжної коди, як відомо, алгоритмічно нерозв'язна. Це означає, що для виявлення недосяжного коду повинні застосовуватися різні евристичні методи, наприклад, засновані на статистичному аналізі програми.

Приклад генерації недосяжного коду:

```
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;
```

Внесення мертвого коду (adding dead code). На відміну від недосяжного коду, мертвий код в програмі виконується, але його виконання ніяк не впливає на результат роботи програми. При внесенні мертвої коди обфускатор має бути упевнений, що фрагмент, що вставляється, не може впливати на код, який обчислює значення функції. Це практично означає, 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, оскільки обчислюється або константне значення, або значення, вже обчислене раніше. Для внесення надлишкової коди можна використовувати перетворення алгебри виразів початкової програми або введення в програму

математичної тотожності. Наприклад, можна скористатися комбінаторною тотожністю $\sum_{i=0}^{8} C_8^i = 256$ і замінити скрізь в програмі використання константи 256 на цикл, який обчислює суму біноміальних коефіцієнтів за приведеною формулою.

Подібні перетворення алгебри обмежені цілими значеннями, оскільки при виконанні операцій з плаваючою комою виникає проблема накопичення помилки обчислень. Наприклад, вираз $\sin^2 x+\cos^2 x$ при обчисленні на машині практично ніколи не дасть в результаті значення 1. З іншого боку, при операціях з цілими значеннями виникає проблема переповнення. Наприклад, якщо використання 32-бітової цілої змінної x замінено на вираз x * p / q, де p і q гарантовано мають одне і те ж значення, при виконанні множення x * p може статися переповнювання розрядної сітки, і після ділення на q вийде результат не рівний p. Як часткове розв'язання задачі можна виконувати множення 64-бітових цілих числах.

Перетворення звідного графу потоку управління до незвідного (transforming reducible to non-reducible flow graph). Коли цільова мова (байт-код або машинна мова) виразніша, ніж початкова, можна використовувати перетворення, що "суперечать" структурі початкової мови. В результаті таких перетворень виходять послідовності інструкцій цільової мови, не відповідні жодній з конструкцій початкової мови.

Можна запропонувати заплутуюче перетворення, яке трансформує графи потоку управління функцій, що зводяться, в байт-коді, отримуваних в результаті компіляції програм, в незвідні графи. Наприклад, таке перетворення може полягати в трансформації структурного циклу в цикл з множинними заголовками з використанням непрозорих предикатів. З одного боку, декомпілятор може спробувати виконати зворотне перетворення, усуваючи області, що не зводяться, в графі, дублюючи вершини або вводячи нові булеві змінні. З іншого боку, распутыватель може за допомогою статичних або статистичних методів аналізу визначити значення непрозорих предикатів, використаних при заплутуванні, і

усунути переходи, що ніколи не виконуються. Проте, якщо здогадка про значення предиката опиниться невірна, в результаті вийде неправильна програма.

Усунення бібліотечних викликів (eliminating library calls). 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 непереносною.

Для програм на традиційних мовах ця проблема стоїть менш гостро, оскільки стандартні бібліотеки, як правило, можуть бути скомпоновані статично разом з самою програмою. В даному випадку програма не містить ніяких імен функцій із стандартної бібліотеки.

Переплетення функції (function interleaving). Ідея цього заплутуючого перетворення в тому, що дві або більш за функції об'єднуються в одну функцію. Списки параметрів початкових функцій об'єднуються, і до них додається ще один параметр, який дозволяє визначити, яка функція насправді виконується.

Клонування функцій (function cloning). При зворотній інженерії функцій насамперед вивчається сигнатура функції, а також те, як ця функція використовується, в яких місцях програми, з якими параметрами і в якому оточенні викликається. Аналіз контексту використання функції можна утруднити, якщо кожен виклик деякої функції виглядатиме як виклик якоїсь іншої, кожен раз нової функції. Може бути створене декілька клонів функції, і до кожного з клонів буде застосований різний набір заплутуючих перетворень.

Розгортка циклів (loop unrolling). Розгортка циклів застосовується в оптимізуючих компіляторах для прискорення роботи циклів або їх розпаралелювання. Разверка циклів полягає в тому, що тіло циклу розмножується два або більше разів, умова виходу з циклу і оператор приросту лічильника відповідним чином модифікуються. Якщо кількість повторень циклу відома у момент компіляції, цикл може бути розгорнутий повністю.

Розкладання циклів (loop fission). Розкладання циклів полягає в тому, що цикл з складним тілом розбивається на декілька окремих циклів з простими тілами і з тим же простором ітерації.

Реструктуризація графа потоку управління. Структура графа потоку управління, наявність в графі потоку управління характерних шаблонів для циклів, умовних операторів і так далі дає цінну інформацію при аналізі програми. Наприклад, по конструкціях графа потоку управління, що повторюються, можна легко встановити, що над функцією було виконано перетворення розгортки циклів, а далі можна запустити спеціальні інструменти, які проаналізують розгорнені ітерації циклу для виділення індуктивних змінних і згортки циклу. Як міра протидії може бути застосоване таке перетворення графа потоку управління, яке приводить граф до однорідного ("плоского") вигляду. Оператори передачі управління на наступні за ними базові блоки, розташовані на кінцях базових блоків, замінюються на оператори передачі управління на спеціально створений базовий блок диспетчера, який по попередньому базовому блоку і керівникам змінним обчислює наступний блок і передає на нього управління. Технічно це може бути зроблено перенумеруванням всіх базових блоків і введенням нової змінної, наприклад state, яка містить номер поточного виконуваного базового блоку. Заплутана функція замість операторів if, for і так далі міститиме оператор switch, розташований усередині нескінченного циклу.

Локалізація змінних в базовому блоці. Це перетворення локалізує використання змінних одним базовим блоком. Для кожного заплутуваного базового блоку функції створюється свій набір змінних. Всі використання локальних і глобальних змінних в початковому базовому блоці замінюються на використання відповідних нових змінних. Щоб забезпечити правильну роботу програми між базовими блоками вставляються так звані з’єднуючі (connective) базові блоки, завдання яких скопіювати вихідні змінні попереднього базового блоку у вхідні змінні наступного базового блоку.

Використання такого заплутуючого перетворення призводить до появи у функції великого числа нових змінних, які, проте, використовуються тільки в одном-двух базових блоках, що заплутує людину, що аналізує програму.

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

Розширення зони дії змінних. Дане перетворення по смислу обернене до попереднього. 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 зона дії змінної і 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 із статичними змінними потрібні складніші методи міжпроцедурного аналізу.

Для подальшого заплутування можна об'єднати декілька 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 заплутуючих перетворень

Існуючі методи заплутування і інструменти для заплутування програм використовують не єдине заплутуюче перетворення, а деяку їх комбінацію. У даному розділі ми розгледимо деякі використовувані на практиці методи заплутування.

Метод Колберга використовує перетворення введення "диспетчера" в заплутувану функцію. Номер наступного базового блоку обчислюється безпосередньо в базовому блоці, що самому виконується, прямим привласненням змінній, яка зберігає номер поточного базового блоку. Для того, щоб утруднити

статичний аналіз, номери базових блоків поміщаються в масив, кожен елемент якого індексується декількома різними способами. Таким чином, для статичного дослідження ладу виконання базових блоків необхідно провести аналіз показчиків.

Метод Ченг Ванга заплутування заснований на наступних заплутуючих перетвореннях: кожен базовий блок заплутуваної функції розбивається на дрібніші частки (т.з. piece) і клонується один або кілька разів. У кожному фрагменті базового блоку змінні локалізуються, і для скріплення базових блоків створюються спеціальні з'єднувальні базові блоки. Далі в кожен фрагмент вводиться мертвий код. Джерелом мертвого коду може бути, наприклад, фрагмент іншого базового блоку тієї ж самої функції або фрагмент базового блоку іншої функції. Оскільки кожен фрагмент використовує свій набір змінних, об'єднуватися вони можуть безболісно (за умови відсутності в програмі показників і викликів функцій з побічним ефектом). Далі з таких комбінованих фрагментів збирається нова функція, в якій для перемикання між базовими блоками використовується диспетчер. Диспетчер приймає як параметри номер попереднього базового блоку і набір булевих 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 в результаті клонування. Виражаючи функцію переходу у вигляді булевої формули, можна добитися того, що задача статичного аналізу диспетчера буде PSPACE-повною.

Методи аналізу

Коротко розглянемо методи аналізу, які застосовуються при аналізі програм в компіляторах. Мета таких методів – виявлення залежностей між компонентами програми, що дає можливість застосувати певні оптимізаційні перетворення, або накладає обмеження на оптимізаційні перетворення, що проводяться.

Методи аналізу програм можуть бути розділені на 4 групи:

– Синтаксичні. До цієї групи відносяться методи, що грунтуються тільки на результатах лексичного, синтаксичного і семантичного аналізу програми.

– Статичні**.** До цієї групи відносяться методи аналізу потоків управління і даних і методи, що грунтуються на результатах аналізу потоків управління і даних. Статичні методи аналізу працюють з 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. Відмітимо, що самі по собі динамічні методи рідко застосовуються для аналізу програм, оскільки, як правило, необхідна інформація про поведінку програми на різних наборах вхідних даних, яка збирається за допомогою статистичних методів аналізу.

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

4.2 Захист розробленого програмного забезпечення

Для захисту розробленого програмного забезпечення запропоновано використовувати алгоритм Madryga. Алгоритм Madryga складається із двох вкладених циклів. Зовнішній цикл повторюється вісім разів (для гарантії надійності число циклів можна збільшити) і полягає в застосуванні внутрішнього циклу до відкритого тексту. Внутрішній цикл перетворює відкритий текст у шифртекст і виконується однократно над кожним 8-бітовим блоком (байтом) відкритого тексту. Таким чином, весь відкритий текст послідовно вісім разів обробляється алгоритмом.

Ітерація внутрішнього циклу оперує з 3-байтовим вікном даних, називаним робочим кадром (рисунок 4.3). Це вікно зрушується на 1 байт за

ітерацію. (При роботі з останніми 2 байтами дані покладаються циклічно замкнутими). Перші два байти робочого кадру циклічно зрушуються на змінне число позицій, а для останнього байта виконується операція XOR з декількома бітами ключа. У міру переміщення робочого кадру всі байти послідовно циклічно зрушуються й піддаються операції XOR із частинами ключа. Послідовні циклічні зрушення перемішують результати попередніх операцій XOR і циклічного зрушення, причому на циклічне зрушення впливають результати XOR. Завдяки цьому процес у цілому оборотний.

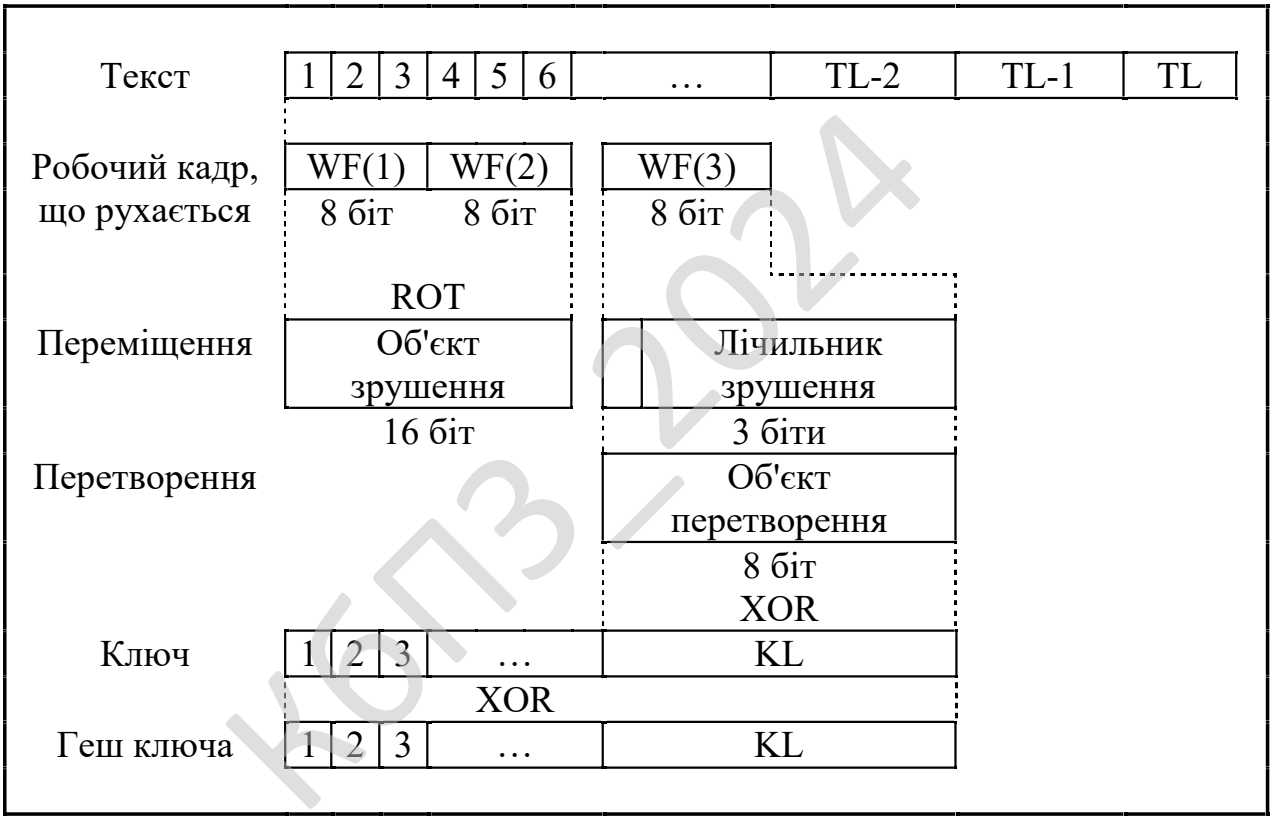

Рисунок 4.3 – Одна ітерація алгоритму Madryga

Оскільки кожний байт даних впливає на два байти ліворуч і на один байт праворуч від себе, після восьми проходів кожний байт шифртексту залежить від 16 байтів ліворуч і 8 байтів праворуч.

При шифруванні кожна ітерація внутрішнього циклу встановлює робочий кадр на передостанній байт відкритого тексту й циклічно переміщає його до

третього з кінця байту відкритого тексту. Спочатку весь ключ піддається операції XOR з випадковою константою й потім циклічно зрушується вправо на 3 біти (ключ і дані рухаються в різних напрямках, щоб мінімізувати надлишкові операції з бітами ключа). Молодші три біти молодшого байта робочого кадру зберігаються, вони визначають циклічне зрушення інших двох байтів. Далі конкатенація двох старших байтом циклічно зрушується вліво на змінне число біт (від 0 до 7). Потім над молодшим байтом робочого кадру виконується операція XOR з молодшим байтом ключа. Нарешті робочий кадр зміщається вправо на один байт і весь процес повторюється.

Випадкова константа призначена для перетворення ключа в псевдовипадкову послідовність. Довжина константи повинна бути рівній довжині ключа. При обміні даними абоненти повинні користуватися однієї й тією же константою. Для 64-бітового ключа Madryga рекомендує константу 0x0fle2d3c4b5a6978.

При розшифруванні процес повторюється у зворотному порядку. У кожній ітерації внутрішнього циклу робочий кадр установлюється на байт, третій ліворуч від останнього байта шифртексту, і циклічно зрушується у зворотному напрямку до байта, розташованого на 2 байти уліво відносно останнього байта шифртексту. 2 байти шифртексту в процесі циклічно зрушуються вправо, а ключ – уліво. Після циклічних зрушень виконується операція XOR.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ КІБЕРБЕЗПЕКИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Програма має простий на інтуїтивно зрозумілий інтерфейс. На рисунку 5.1 приведено головне вікно програми. В лівій половині виводиться сирцевий код програми, в правій половині – код, захищений від модифікації.

За допомогою основного меню можна виконати наступні операції:

– Відкриття сирцевого коду програми та збереження коду, захищеного від модифікації.

– Вибір алгоритму та запуск процедури обфускації коду.

– Вибір алгоритму та запуск процедури деобфускації коду.

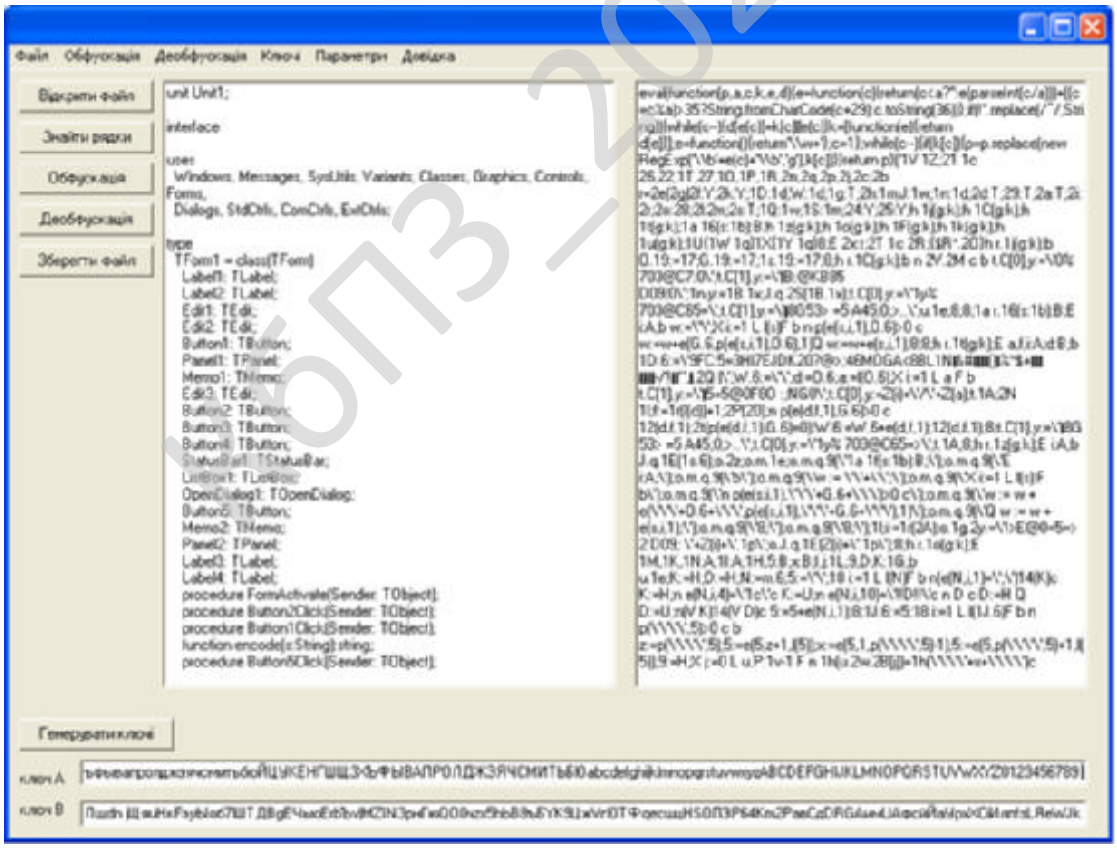

Рисунок 5.1 – Головне вікно програми

– Генерація ключів.

– Встановлення налаштувань програми та параметрів відображення.

– Виведення вікна інформації про програму.

Для більш швидкого доступу до основних операцій з лівого боку виведено блок клавіш із відповідними значеннями.

Значення згенерованих ключів A та B виводяться в нижній частині головного вікна програми.

На рисунку 5.2 приведено вікно довідки про програму, в якому вказуються тема бакалаврського проекту, ПІБ його виконавця на керівника проекту та дані про місце і час розробки.

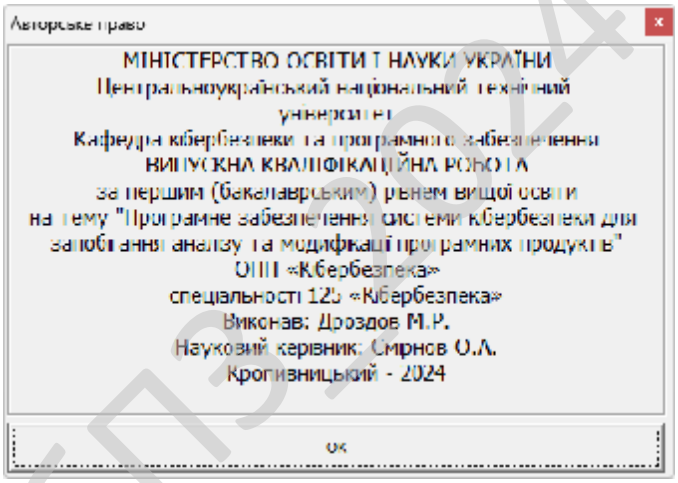

Рисунок 5.2 – Вікно довідки

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, призначено для системи кібербезпеки для запобігання аналізу та модифікації програмних продуктів.

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

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем для запобігання аналізу та модифікації програмних продуктів.

– Досліджена система для запобігання аналізу та модифікації програмних продуктів.

– На основі отриманих результатів досліджень створена програмна реалізація системи кібербезпеки для запобігання аналізу та модифікації програмних продуктів.

Розроблені під час виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання для запобігання аналізу та модифікації програмних продуктів.

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 Delphi 10.4. Дана мова програмування дозволяє найбільш ефективно обробляти дані призначені для системи кібербезпеки для запобігання аналізу та модифікації програмних

продуктів. 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 Madryga.

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. Derek Fisher. Application Security Program Handbook. Manning Publications. 2021. 155 p.

2. Cameron Wyatt PH.D. Kali Linux Tutorial. Independently published. 2021. 60 p.

3. Alex Matrosov, Eugene Rodionov, Sergey Bratus. Rootkits and Bootkits. No Starch Press. 2019. 450 p.

4. Kuznetsov, O., Kuznetsova, Y., Smirnov, O., Kostenko, O., Zvieriev, V. «Evaluating Hashing Algorithms in the Age of ASIC Resistance». *CEUR Workshop Proceedings*, 2023, 3628, pp. 93-105.

5. Kuznetsov O., Frontoni E., Kuznetsova Ye., Smirnov O., Chevardin V. «Achieving Enhanced Security in Biometric Authentication: A Rigorous Analysis of Code-Based Fuzzy Extractor». *CEUR Workshop Proceedings,* Volume 3624, 2023, pp. 330-339.

6. Smirnov, O., Sydorenko, V., Aleksander, M., Zhyharevych, O., Yenchev, S. «Simulation of the cloud IoT-based monitoring system for critical infrastructures». *CEUR Workshop Proceedings,* Volume 3530, 2023, pp. 256-265.

7. Kuznetsov, O., Kandiy, S., Frontoni, E., Smirnov, O. «Trade-offs in Post-Quantum Cryptography: A Comparative Assessment of BIKE, HQC, and Classic McEliece». *CEUR Workshop Proceedings,* Volume 3504, 2023, pp. 1-11.

8. Smirnov, O., Neskorodieva, T., Fedorov, E., Rudakov, K., Neskorodieva, A. «Method Detection Audit Data Anomalies on Basis Restricted Cauchy Machine» *CEUR Workshop Proceedings,* Volume 3187, 2022,

9. Smirnov, O., Lakhno, V., Akhmetov, B., Chubaievskyi, V., Khorolska, K., Bebeshko, B. «Selection of a Rational Composition of İnformation Protection Means Using a Genetic Algorithm». *In: Rajakumar, G., Du, KL., Vuppalapati, C., Beligiannis, G.N. (eds) Intelligent Communication Technologies and Virtual Mobile Networks.*

Lecture Notes on Data Engineering and Communications Technologies, vol 131. 2023. Springer, Singapore. pp. 21-34.

10. Smirnov O.A., Al-Oraiqat A.M., Ulichev O.S., Meleshko Ye.V., Al-Rawashdeh H.S., Polishchuk L.I. «Modeling strategies for information influence dissemination in social networks». *Journal of Ambient Intelligence and Humanized Computing* Volume 13, Issue 5. Springer, Cham. 2022, pp. 2463-2477.

11. Smirnov O., Kuznetsov A., Zhora V., Onikiychuk A., Pieshkova O. «Hiding Messages in Audio Files Using Direct Spread Spectrum». *11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2021,* Cracow, Poland, 22-25 September 2021. P. 414-418

12. Smirnov O., Kuznetsov A., Lokotkova I., Kuznetsova T., Florov S., Lebid O. «Using Orthogonal Signals to Hide Information in Images». *4 IEEE International Conference on Advanced Information and Communication Technologies (AICT) - 2021*, Lviv, Ukraine, September 21-25, 2021. P. 255-260.

13. Smirnov O., Kuznetsov A., Girzheva O., Kiian A., Nakisko O., Kuznetsova T. «Advanced Code-Based Electronic Digital Signature Scheme». *2020 IEEE International Conference on Problems of Infocommunications Science and Technology, PIC S and T 2020,* Kharkiv, 6 October 2020-9 October 2020, P. 358-362.

14. Smirnov O., Kuznetsov A., Kiian A., Kuznetsova K. «Data hiding scheme based on spread sequence addressing». *CEUR Workshop Proceedings* Volume 2805, 2020, Pages *44-58.*

15. 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.

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

17. Smirnov O., Alimseitova Zh., Adranova A., Akhmetov B., Lakhno V., Zhilkishbayeva G. «Models and algorithms for ensuring functional stability and cybersecurity of virtual cloud resources». *Journal of theoretical and applied information technology* Vol.98. No 21, 2020, P. 3334-3346.

18. Smirnov O., Kuznetsov A., Arischenko A., Chepurko I., Onikiychuk A., Kuznetsova T. «Pseudorandom sequences for spread spectrum image steganography». *CEUR Workshop Proceedings* Volume 2654, 2020, Pages 122-131.

19. Smirnov O., Kuznetsov A., Kovalchuk D., Kuznetsova T. «New technique for data hiding in cover images using adaptively generated pseudorandom sequences». *CEUR Workshop Proceedings* Volume 2654, 2020, Pages 1-14.

20. Smirnov O., Lutsenko M., Kuznetsov A., Kiian A., Kuznetsova T., «Biometric cryptosystems: overview, state-of-the-art and perspective directions». *Lecture Notes in Networks and Systems*, vol 152. Springer, Cham. 2021, pp 66-84.

21. Smirnov O., Kuznetsov A., Onikiychuk A., Makushenko T., Anisimova O., Arischenko A. «Adaptive pseudo-random sequence generation for spread spectrum image steganography». *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, Ukraine, Kyiv, May 14-18. 2020. P. 161-165.

22. Smirnov O., Kuznetsov A., Kiian A., Babenko V., Perevozova I.,Chepurko I. «New Approach to the Implementation of Post-Quantum Digital Signature Scheme». *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT),* Ukraine, Kyiv, May 14-18. 2020. P. 166-171.

23. 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.

24. Smirnov O., Kuznetsov A., Pushkar'ov A., Serhiienko R., Babenko V., Kuznetsova T., «Representation of Cascade Codes in the Frequency Domain». In:

Radivilova T., Ageyev D., Kryvinska N. (eds) Data-Centric Business and Applications. Lecture Notes on Data Engineering and Communications Technologies, vol 48. Springer, Cham. 2021. pp 557-587.

25. Smirnov, O., Markovets, O. Vovk, N., Turchyn, Y., «Model of informational support for social network administrators' content creation». *CEUR Workshop Proceedings* Volume 2616, 2020, Pages 125-136.

26. 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.

27. Smirnov, O., Kuznetsov, A., Gorbacheva, L., Babenko, V., «Hiding data in images using a pseudo-random sequence», *CEUR Workshop Proceedings* Volume 2608, 2020, Pages 646-660.

28. 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.

29. 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.

30. Smirnov, O., Ulichev, O., Meleshko, Y., Khokh, V., Goncharenko, I. «Method of Choosing Objects for Informational Influence in Social Networks during Information Campaign Based on the Analytic Hierarchy Process». *CEUR Workshop Proceedings*, Vol 2588, P. 215-227, 2019.

31. 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.

32. Smirnov, O., Kuznetsov, A., Kiian, A., Gorbenko, Y., Cherep, O., Bexhter L. «Code-based Pseudorandom Generator for the Post-Quantum Period», *2019 IEEE International Conference on Advanced Trends in Information Theory (IEEE ATIT 2019).* 18.12.19-20.12.19 Kyiv Ukraine. P. 204 – 209.

33. Smirnov, O., Kuznetsov, A., Nariezhnii, O., Stelnyk, S., Kokhanovska, T., Kuznetsova T., «Side Channel Attack on a Quantum Random Number Generator», *10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2019*; Metz; France; 18 - 21 September 2019. P.713-718.

34. 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.

35. 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.

36. Smirnov, O., Hu, Z., Vasiliu, Y., Sydorenko, V., Polishchuk, Y., «Abstract Model of Eavesdropper and Overview on Attacks in Quantum Cryptography Systems», *10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2019*; Metz; France; 18-21 September 2019. P.399-405.

37. 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.

38. Smirnov, O., Kuznetsov, A., Kiian, A., Babenko, B., Zhosan, H., Prokopovych-Tkachenko, D., «Soft Decoding Method for Turbo-Productive Codes»,

2019 3rd International Conference on Advanced Information and Communications Technologies, AICT 2019, Lviv, Ukraine, 2-6 July, 2019, P. 129-134.

39. Smirnov, O., Kuznetsov, A., Kiian, A., Zamula, A., Rudenko, S., Hryhorenko, V., «Variance Analysis of Networks Traffic for Intrusion Detection in Smart Grids», *2019 IEEE 6th International Conference On Energy Smart Systems (2019 IEEE ESS)*, Kyiv, Ukraine April 17-19, 2019 P. 353-358.

40. Smirnov, O., Kuznetsov, A., Kavun, S., Babenko, B., Nakisko, O., Kuznetsova, K., «Malware Correlation Monitoring in Computer Networks of Promising Smart Grids», *2019 IEEE 6th International Conference On Energy Smart Systems (2019 IEEE ESS)*, Kyiv, Ukraine April 17-19, 2019 P. 347-352.

41. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Pastukhov, M., Kuznetsova, K., Prokopovych-Tkachenko, D., «Discrete Signals with Special Correlation Properties», *CEUR Workshop Proceedings* Volume 2353, *CEUR Workshop Proceedings* 2019, Pages 618-629.

42. Smirnov, O., Kuznetsov, A., Kiian, A., Kuznetsova, K., Ivko, T., Prokopovych-Tkachenko, D., «Soft Decoding Based on Ordered Subsets of Verification Equations of Turbo-Productive Codes», *CEUR Workshop Proceedings* Volume 2353, *CEUR Workshop Proceedings* 2019, Pages 873-884.

43. Smirnov A.A., Kuznetsov A.A., Danilenko D.A., Berezovsky A., «The statistical analysis of a network traffic for the intrusion detection and prevention systems», *Telecommunications and Radio Engineering*. – Volume 74, Issue 1. – Begel House Inc. – 2015. – P. 61-78.

44. Смірнов О.А. Козлов Я.О., Смірнова Т.В. «Дослідження застосування SIEM-систем для забезпечення кібербезпеки та захисту інформації». *II Міжнародна науково-практична Інтернет-конференція «Інновації та перспективні шляхи hозвитку інформаційних технологій (ІПШРІТ-2023)»* м.Черкаси 6 грудня 2023 року – Черкаси: ЧДТУ.– 2023. – С.251-252.

45. Козлов Я.О., Смірнова Т.В., Смірнов О.А. «Дослідження SIEM-систем для забезпечення кібербезпеки». *VII міжнародна науково-практична*

конференція "Інформаційна безпека та комп'ютерні технології" до 30-ти річчя кафедри кібербезпеки та програмного забезпечення, м. Кропивницький. 1 листопада 2023 р. – Кропивницький: ЦНТУ. – 2023. – С. 26.

46. Козлов Я.О., Козірова Н.Л., Смірнов О.А. «Дослідження структури та принципу роботи SIEM-системи». *VII міжнародна науково-практична конференція "Інформаційна безпека та комп'ютерні технології" до 30-ти річчя кафедри кібербезпеки та програмного забезпечення,* м. Кропивницький. 1 листопада 2023 р. – Кропивницький: ЦНТУ. – 2023. – С. 59.

47. 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.

48. Смірнов О.А., Смірнова Т.В., Якименко Н.М., Смірнов С.А., Поліщук Л.І., «Дослідження стійкості до диференціального криптоаналізу запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» *Системи управління, навігації та зв'язку,* 2022, № 3(69). С. 93-98.

49. Смірнов О.А., Смірнова Т.В., Якименко Н.М., Поліщук Л.І., Смірнов С.А. «Дослідження статистичної стійкості та швидкісних характеристик запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» *Вісник Хмельницького національного університету. Серія: «Технічні науки»,* № 2 (307). С. 46-52. 2022.

50. Смірнов О.А., Смірнова Т.В., Константинова Л.В., Смірнов С.А., Якименко Н.М., «Дослідження стійкості до лінійного криптоаналізу запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» *Системи управління, навігації та зв'язку*, 2022, № 1(67). С. 84-89.

Додаток А
(обов’язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи кібербезпеки для запобігання аналізу та модифікації програмних продуктів.

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. № 135-02 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 для запобігання аналізу та модифікації програмних продуктів.

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 для запобігання аналізу та модифікації програмних продуктів;

– 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 Delphi 10.4.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

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

7 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 – 80 аркушів.

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 4.06.2024 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 для запобігання аналізу та модифікації програмних продуктів

Лістинг програми

Код документу 12

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

Загальна кількість аркушів: 101

Літера: РП

Кропивницький – 2024 року

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 для запобігання ');
Memo1.Lines.Add(' аналізу та модифікації програмних продуктів ');
Memo1.Lines.Add('');
Memo1.Lines.Add('');
Memo1.Lines.Add('Керівник: Смірнов О.А.');
Memo1.Lines.Add('');
Memo1.Lines.Add('Розробив: студент Дроздов Максим Романович');
Memo1.Lines.Add('                  гр. КБ-20 ');
Memo1.Lines.Add('');
Memo1.Lines.Add('');
Memo1.Lines.Add('м. Кропивницький 2024');
end;
end.
```

Файл morphine.pas – запобігання на основі обфускації коду

```
   unit morphine;

//Якщо RUBBISH_NOPS визначено, додаємо ложні команди та пусті цикли для
погіршання дизасемблювання

interface
{ $DEFINE RUBBISH_NOPS}
{ $DEFINE STATIC_CONTEXT}
uses Windows, SysUtils;

//
//Блок коду:
//0..$10: jmp GetProcAddress+jmp LoadLibrary+pad
//$10..$10+KeySize:Key
//$10+KeySize..$10+KeySize+sizeof(DynLoader):DynLoader
//$10+KeySize+sizeof(DynLoader): code
//
//Блок даних:
//0..sizeof(host)-1: host
//
//Блок імпорту:
//0..$70-1: imports
//
//[TlsSection:]
//0..sizeof(tls): tls
//


//
//переміщуємо функцію jmps (getprocaddress, loadlibrary) в кінець
ініціалізуючого/поліморфічного редактора коду
//для запобігання AV детектування (блок коду стартує з  ..000000FF2534.. у якому
записана сигнатура ):
//реалізовано декілька варіантів для кожного jmp для коду імпорту
(getprocaddress, loadlibrary) та додане фіксування імпортованого коду

//Це новий заплутаний код:
//
//Блок коду:
//$0..KeySize:Key
//KeySize..KeySize+sizeof(DynLoader):DynLoader
//KeySize+sizeof(DynLoader): code
//
//Блок даних:
//0..sizeof(host)-1: host
//
//Блок імпортування:
//0..$70-1: imports
//
//[TlsSection:]
//0..sizeof(tls): tls


//- деякі довільні дані(CoderRoller1) у програму заплутування (DynCoder and
Decoder)
//- блок видалення даних
//- блок незначних помилок
//
//Це новий заплутаний код:
//
//Блок коду:
//0: Rubbish
//KeyPtr..KeyPtr+KeySize:Key
//KeyPtr+KeySize..KeyPtr+KeySize+sizeof(DynLoader):DynLoader
//KeyPtr+KeySize+sizeof(DynLoader): code
```

```
//code+sizeof(code): host
//
//Блок імпортування:
//0..$70-1: imports
//
//[TlsSection:]
//0..sizeof(tls): tls

//
//- заплутаний код імен інструкцій

//- Підтримує DLL
//- введення цього коду дозволяє реалізувати помилковий для дизасемблювання код
//- введення незначних помилок
//+ .edata блок після .tls

//- заплутаний код покращено
//- відбувається шифрування основних даних

//Якщо ви маєте суму PEB, TEB структур (визначених у  DynLoader)

const
 //реалізація заглушки для DOS
 //Ця програма записує  "Ця програма не працює під  DOS режимом"
 DosStub:array[0..$38-1] of Byte=
 ($BA,$10,$00,$0E,$1F,$B4,$09,$CD,$21,$B8,$01,$4C,$CD,$21,$90,$90,
  $54,$68,$69,$73,$20,$70,$72,$6F,$67,$72,$61,$6D,$20,$6D,$75,$73,
  $74,$20,$62,$65,$20,$72,$75,$6E,$20,$75,$6E,$64,$65,$72,$20,$57,
  $69,$6E,$33,$32,$0D,$0A,$24,$37);

 //константи блоку імпорту
 NumberOfDLL=1;                              //кількість бібліотек
 NumberOfImports=2;                          //кількість функцій
 Kernel32Name='kernel32.dll';                //ім'я бібліотеки
 NtdllName='ntdll.dll';                      //ім'я ntdll.dll

 GetProcAddressName='GetProcAddress';        //ім'я funct1
 LoadLibraryName='LoadLibraryA';             //ім'я func2
 Kernel32Size=12;                            //довжина імені dll
 GetProcAddressSize=14;                      //довжина імені func1
 LoadLibrarySize=12;                         //довжина імені func2

 //індекси інструкції заплутування
 PII_BEGIN                   = 0;

 PII_POLY_BEGIN              = PII_BEGIN;
 PII_POLY_MOV_REG_LOADER_SIZE= PII_POLY_BEGIN;
 PII_POLY_MOV_REG_LOADER_ADDR= PII_POLY_MOV_REG_LOADER_SIZE+1;

 PII_CODER_BEGIN             = PII_POLY_MOV_REG_LOADER_ADDR+1;
 PII_CODER_CALL_GET_EIP      = PII_CODER_BEGIN+1;
 PII_CODER_GET_EIP           = PII_CODER_CALL_GET_EIP+1;
 PII_CODER_FIX_DST_PTR       = PII_CODER_GET_EIP+1;
 PII_CODER_KEY_START         = PII_CODER_FIX_DST_PTR+1;
 PII_CODER_MOV_REG_KEY       = PII_CODER_KEY_START;
 PII_CODER_FIX_SRC_PTR       = PII_CODER_MOV_REG_KEY+1;

 PII_CODER_CODE              = PII_CODER_FIX_SRC_PTR+1;
 PII_CODER_LOAD_KEY_TO_REG   = PII_CODER_CODE;
 PII_CODER_TEST_KEY_END      = PII_CODER_LOAD_KEY_TO_REG+1;
 PII_CODER_JZ_CODER_BEGIN    = PII_CODER_TEST_KEY_END+1;
 PII_CODER_ADD_DATA_IDX      = PII_CODER_JZ_CODER_BEGIN+1;
 PII_CODER_XOR_DATA_REG      = PII_CODER_ADD_DATA_IDX+1;
 PII_CODER_STORE_DATA        = PII_CODER_XOR_DATA_REG+1;
 PII_CODER_INC_SRC_PTR       = PII_CODER_STORE_DATA+1;
 PII_CODER_LOOP_CODER_CODE   = PII_CODER_INC_SRC_PTR+1;
 PII_CODER_END               = PII_CODER_LOOP_CODER_CODE+1;

 PII_POLY_JMP_DYNLOADER      = PII_CODER_END+1;
```

```
 PII_POLY_END                    = PII_POLY_JMP_DYNLOADER;
 PII_END                         = PII_POLY_END;

 //інші константи
 MaxPolyCount=20;                                  //максимальна кількість
варіантів для однієї інструкції
 InitInstrCount=PII_END+1;                         //редактор лічильника інструкцій
 RawDataAlignment=$200;                            //вирівнювання SizeOfRawData
 DosStubEndSize=$88;                               //$100 - SizeOf(DosStub)

 //image type const
 IMAGE_TYPE_EXE=0;
 IMAGE_TYPE_DLL=1;
 IMAGE_TYPE_SYS=2;
 IMAGE_TYPE_UNKNOWN=$FFFFFFFF;

 //це двійне слово закінчення  DYN_LOADER у формі декодування
 DYN_LOADER_END_MAGIC=$C0DEC0DE;
 DYN_LOADER_DEC_MAGIC=$1EE7C0DE;

 //регістри
 REG_EAX=0;
 REG_ECX=1;
 REG_EDX=2;
 REG_EBX=3;
 REG_ESP=4;
 REG_EBP=5;
 REG_ESI=6;
 REG_EDI=7;
 REG_NON=255;

 Reg8Count=8;
 Reg16Count=8;
 Reg32Count=8;

 RT_XP_MANIFEST=24;

type
 //тепер декілька типів неможливо знайти у windows.pas

 PImageImportByName=^TImageImportByName;
 TImageImportByName=packed record
  Hint:Word;
  Name:array of Char;
 end;
 PImageThunkData=^TImageThunkData;
 TImageThunkData=packed record
  case Byte of
   0:(ForwarderString:PByte);
   1:(FunctionPtr:PCardinal);
   2:(Ordinal:Cardinal);
   3:(AddressOfData:PImageImportByName);
 end;
 PImageImportDescriptor=^TImageImportDescriptor;
 TImageImportDescriptor=packed record
  case Byte of

0:(Characteristics,cTimeDateStamp,cForwarderChain,cName:Cardinal;cFirstThunk:PIm
ageThunkData);

1:(OriginalFirstThunk:PImageThunkData;oTimeDateStamp,oForwarderChain,oName:Cardi
nal;oFirstThunk:PImageThunkData);
 end;

 PExportDirectoryTable=^TExportDirectoryTable;
 TExportDirectoryTable=packed record
  Flags,TimeStamp:Cardinal;
  MajorVersion,MinorVersion:Word;
```

```
NameRVA,OrdinalBase,AddressTableEntries,NumberOfNamePointers,ExportAddressTableR
VA,
  NamePointerRVA,OrdinalTableRVA:Cardinal;
 end;

 //ось так подібно блоку .tls
 PTlsSectionData=^TTlsSectionData;
 TTlsSectionData=packed record

RawDataStart,RawDataEnd,AddressOfIndex,AddressOfCallbacks,SizeOfZeroFill,Charact
eristics:Cardinal;
 end;

 //мій тип для блоку  tls
 TTlsCopy=record
  Directory:PImageDataDirectory;
  блокData:PTlsSectionData;
  RawData:Pointer;
  RawDataLen,Index:Cardinal;
  Callbacks:Pointer;
  CallbacksLen:Cardinal;
 end;

 //одна псевдо інструкція (p-i) для движка перетворень коду (може містити більш
ніж одну x86 інструкцію)
 TInstruction=packed record
  Len:Byte;                                  //довжина заплутаного коду
  Fix1,Fix2,Fix3,Fix4:Byte;                  //байти індексування для
фіксації
  Code:array[0..30] of Char;                 //заплутаний код
 end;

 //список  p-i, який обирається кожний раз при операції заплутування коду
 TVarInstruction=packed record
  Count,Index:Byte;                          //число p-i та число можливостей
вибору
  VirtualAddress:Cardinal;                   //адреса інструкції у блоці
CODE
  Vars:array[0..MaxPolyCount-1] of TInstruction;//список
 end;

 PResourceDirectoryTable=^TResourceDirectoryTable;
 TResourceDirectoryTable=packed record
  Characteristics:Cardinal;
  TimeDateStamp:Cardinal;
  MajorVersion:Word;
  MinorVersion:Word;
  NumberOfNameEntries:Word;
  NumberOfIDEntries:Word;
 end;

 PResourceDirectoryEntry=^TResourceDirectoryEntry;
 TResourceDirectoryEntry=packed record
  NameID:Cardinal;
  SubdirDataRVA:Cardinal;
 end;

 PResourceDataEntry=^TResourceDataEntry;
 TResourceDataEntry=packed record
  DataRVA:Cardinal;
  Size:Cardinal;
  Codepage:Cardinal;
  Reserved:Cardinal;
 end;

 PResourceTableDirectoryEntry=^TResourceTableDirectoryEntry;
 TResourceTableDirectoryEntry=packed record
  Table:TResourceDirectoryTable;
```

```
  Directory:TResourceDirectoryEntry;
 end;

 PIconDirectoryEntry=^TIconDirectoryEntry;
 TIconDirectoryEntry=packed record
  Width:Byte;
  Height:Byte;
  ColorCount:Byte;
  Reserved:Byte;
  Planes:Word;
  BitCount:Word;
  BytesInRes:Cardinal;
  ID:Word;
 end;

 PIconDirectory=^TIconDirectory;
 TIconDirectory=packed record
  Reserved:Word;
  ResType:Word;
  Count:Word;
  Entries:array[0..31] of TIconDirectoryEntry;
 end;

 TImageType=(itExe,itDLL,itSys);

 TEncoderProc=function(AAddr:Pointer):Cardinal; stdcall;
 procedure Protect(InputFileName: string);


var
 DosHeader:TImageDosHeader;
 DosStubEnd:array[0..DosStubEndSize-1] of Char;
 NtHeaders:TImageNtHeaders;
 FileHandle,MainFile:THandle;
 InputFileName,OutputFileName,Options:string;

NumBytes,TotalFileSize,MainSize,LoaderSize,VirtLoaderData,VirtMainData,VirtKey,I
nitSize,KeyPtr,

AnyDWORD,LoaderPtr,TlsSectionSize,Delta,HostImageBase,HostSizeOfImage,HostCharac
teristics,

ReqImageBase,RandomValue,ExportSectionSize,CurVirtAddr,CurRawData,ExportRVADelta
,
 HostExportSectionVirtualAddress,ExportNamePointerRVAOrg,ExportAddressRVAOrg,
 ImportSectionDataSize,HostImportSectionSize,ImportSectionDLLCount,

HostImportSectionVirtualAddress,InitcodeThunk,CodeSectionVirtualSize,LoaderRealS
ize,

MainRealSize,MainRealSize4,LogCnt,MainDataDecoderLen,DynLoaderDecoderOffset,LdrP
trCode,LdrPtrThunk,

ResourceSectionSize,HostResourceSectionSize,ResourceIconGroupDataSize,HostResour
ceSectionVirtualAddress,
 ResourceXPMDirSize,AfterImageOverlaysSize:Cardinal;

CodeSection,ExportSection,TlsSection,ImportSection,ResourceSection:TImageSection
Header;
 ImportDesc,NullDesc:TImageImportDescriptor;
 PImportDesc:PImageImportDescriptor;
 ThunkGetProcAddress,ThunkLoadLibrary:TImageThunkData;
 NullWord,KeySize,TrashSize,Trash2Size,HostSubsystem:Word;

MainData,MainDataCyp,LoaderData,Key,InitData,Trash,Trash2,Ptr,ExportData,ImportS
ectionData,ResourceData,
 MainDataEncoder,MainDataDecoder,AfterImageOverlays:Pointer;
 PB,PB2,PB3,PB4,DynLoaderSub,LdrPtr,MainDataDecPtr:PByte;
```

```
TlsSectionPresent,ExportSectionPresent,Quiet,DynamicDLL,ResourceSectionPresent,S
aveIcon,
 SaveOverlay,OverlayPresent:Boolean;
 TlsCopy:TTlsCopy;
 TlsSectionData:TTlsSectionData;
 ImageType:TImageType;
 I:Integer;
 DynLoaderJmp:PCardinal;
 ResourceRoot,ResourceIconGroup,ResourceXPManifest:PResourceDirectoryTable;
 ResourceDirEntry:PResourceDirectoryEntry;
 EncoderProc:TEncoderProc;

implementation

uses maincode;

procedure DynLoader; assembler; stdcall;
//Завантажувач
//він завантажує ре файли до пам'яті з MainData
//фіксуються переміщення
//фіксуються імпортування
//фіксуються експортування
//
asm
  push 012345678h               //LoadLibrary
  push 012345678h               //GetProcAddress
  push 012345678h               //Addr of MainData
  //операція заплутування
  //використовується rva для основниного коду, не знаючи базового образу
  //який отримується  eip та починає робити з 0FFFFF000h
  //по 000401XXXh приблизно 000401000h , саме тому
  //код використовується до 2000h
  call @get_eip
  @get_eip:
  pop eax
  and eax,0FFFFF000h
  add [esp],eax
  add [esp+004h],eax
  add [esp+008h],eax

  call @DynLoader_begin

  //ще один метод заплутування
  //код у LoadLibrary який викликає DllMain зберігає esp у esi
  //якщо змінюється esi то ми додаємо зліва його від esp, та правдиве його
значення
  //додаємо суму 010h для параметрів DllMain + повертаємо адресу заплутаного
коду
  mov esi,esp
  mov [esi+004h],ecx            //змінюємо DllMain.hinstDLL
  add esi,010h
  jmp eax                       //переходимо до наступної точки зміни

 @DynLoader_begin:
  //маємо базовий образ коду у  eax (except ax), зберігаємо його у ebp-050h
  push ebp
  mov ebp,esp
  sub esp,00000200h
  {
   -01F8..-0100 -       NtHeaders:TImageNtHeaders
   -09C         -       MemoryBasicInformation.BaseAddress
   -098         -       MemoryBasicInformation.AllocationBase
   -094         -       MemoryBasicInformation.AllocationProtect
   -090         -       MemoryBasicInformation.RegionSize
   -08C         -       MemoryBasicInformation.State
   -088         -       MemoryBasicInformation.Protect
   -084         -       MemoryBasicInformation.Type
```

```
 -07C         -       IsBadReadPtr:Pointer
 -078         -       VirtualQuery:Pointer
 -074         -       VirtualProtect:Pointer
 -070         -       FirstModule:Cardinal

 -054         -       OrgImageSize:Cardinal
 -050         -       ImageBase:Cardinal
 -04C         -       ImageEntryPoint:Cardinal
 -048         -       ImageSize:Cardinal
 -044         -       ImageType:Cardinal
 -040         -       HintName:Cardinal
 -03C         -       Thunk:Cardinal
 -038..-010   -       блок:TImageSectionHeader
 -00C         -       FileData:Pointer
 -008         -       ImageSizeOrg:Cardinal
 -004         -       ImageBaseOrg:Cardinal
 +008         -       AddrOfMainData:Pointer
 +00C         -       GetProcAddress:Pointer
 +010         -       LoadLibrary:Pointer
}
push ebx                                //зберігаємо ebx, edi, esi
push edi
push esi

and eax,0FFFF0000h

mov [ebp-050h],eax                      //зберігаємо ImageBase

mov ecx,00008000h
@DynLoader_fake_loop:
add eax,0AF631837h
xor ebx,eax
add bx,ax
rol ebx,007h
loop @DynLoader_fake_loop
//Включаємо криптo програму заплутування
//esp та ebp не повинні змінюватися
push dword ptr [ebp+008h]               //AAddr
dd DYN_LOADER_DEC_MAGIC
//\кінець криптоперетворень

call @DynLoader_fill_image_info

push 000h
push 06C6C642Eh
push 032336C65h
push 06E72656Bh                         //kernel32.dll до стеку
push esp                                //lpLibFileName
mov eax,[ebp+010h]                      //ImportThunk.LoadLibrary
call [eax]                              //LoadLibrary
add esp,010h
mov edi,eax

push 000h
push 0636F6C6Ch
push 0416C6175h
push 074726956h                         //VirtualAlloc до стеку
push esp                                //lpProcName
push eax                                //hModule
mov eax,[ebp+00Ch]                      // реалізація ImportThunk.GetProcAddress
call [eax]                              //GetProcAddress
add esp,010h
mov ebx,eax
test eax,eax
jz @DynLoader_end

push 000007463h
push 065746f72h
push 0506C6175h
```

```
push 074726956h                         //VirtualProtect до стеку
push esp                                //lpProcName
push edi                                //hModule
mov eax,[ebp+00Ch]                      // реалізація ImportThunk.GetProcAddress
call [eax]                              //GetProcAddress
add esp,010h
mov [ebp-074h],eax                      //VirtualProtect
test eax,eax
jz @DynLoader_end

push 000h
push 079726575h
push 0516C6175h
push 074726956h                         //VirtualQuery до стеку
push esp                                //lpProcName
push edi                                //hModule
mov eax,[ebp+00Ch]                      // реалізація ImportThunk.GetProcAddress
call [eax]                              //GetProcAddress
add esp,010h
mov [ebp-078h],eax                      //VirtualQuery
test eax,eax
jz @DynLoader_end

push 000h
push 072745064h
push 061655264h
push 061427349h                         //IsBadReadPtr до стеку
push esp                                //lpProcName
push edi                                //hModule
mov eax,[ebp+00Ch]                      // реалізація ImportThunk.GetProcAddress
call [eax]                              //GetProcAddress
add esp,010h
mov [ebp-07Ch],eax                      //IsBadReadPtr
test eax,eax
jz @DynLoader_end


lea edi,[ebp-01F8h]                     //NtHeaders
push edi
mov esi,[ebp+008h]                      //TImageDosHeader
add esi,[esi+03Ch]                      //TImageDosHeader._lfanew
push 03Eh                               //SizeOf(NtHeaders) div 4
pop ecx
rep movsd
pop edi
mov eax,[edi+034h]                      //NtHeaders.OptionalHeader.ImageBase
mov [ebp-004h],eax                      //ImageBaseOrg
mov ecx,[edi+050h]                      //NtHeaders.OptionalHeader.SizeOfImage
mov [ebp-008h],ecx                      //ImageSizeOrg

push ecx
push PAGE_EXECUTE_READWRITE             //flProtect
push MEM_COMMIT or MEM_RESERVE          //flAllocationType
push ecx                                //dwSize
push eax                                //lpAddress
call ebx                                //VirtualAlloc
pop ecx
test eax,eax
jnz @DynLoader_alloc_done

push PAGE_EXECUTE_READWRITE             //flProtect
push MEM_COMMIT                         //flAllocationType
push ecx                                //dwSize
push eax                                //lpAddress
call ebx                                //VirtualAlloc
test eax,eax
jz @DynLoader_end

@DynLoader_alloc_done:
```

```
  mov [ebp-00Ch],eax                    //FileData
  mov edi,eax
  mov esi,[ebp+008h]                    //TImageDosHeader
  push esi
  mov ecx,esi                           //TImageDosHeader
  add ecx,[esi+03Ch]                    //+TImageDosHeader._lfanew = NtHeaders
  mov ecx,[ecx+054h]                    //NtHeaders.SizeOfHeaders
  rep movsb
  pop esi
  add esi,[esi+03Ch]                    //TImageNtHeaders
  add esi,0F8h                          //+SizeOf(TImageNtHeaders) = блок
headers

 @DynLoader_LoadSections:
  mov eax,[ebp+008h]                    //TImageDosHeader
  add eax,[eax+03Ch]                    //TImageDosHeader._lfanew
  movzx eax,[eax+006h]                  //NtHeaders.FileHeader.NumberOfSections

 @DynLoader_LoadSections_do_section:
  lea edi,[ebp-038h]                    //Section
  push edi
  push 00Ah                             //SizeOf(TImageSectionHeader) div 4
  pop ecx
  rep movsd
  pop edi

 @DynLoader_LoadSections_copy_data:
  mov edx,[edi+014h]                    //Section.PointerToRawData
  test edx,edx
  jz @DynLoader_LoadSections_next_section
  push esi
  mov esi,[ebp+008h]                    //AHostAddr
  add esi,edx                           //AHostAddr + блок.PointerToRawData
  mov ecx,[edi+010h]                    //Section.SizeOfRawData
  mov edx,[edi+00Ch]                    //Section.VirtualAddress
  mov edi,[ebp-00Ch]                    //FileData
  add edi,edx                           //FileData + блок.VirtualAddress
  rep movsb
  pop esi
 @DynLoader_LoadSections_next_section:
  dec eax
  jnz @DynLoader_LoadSections_do_section

  mov edx,[ebp-00Ch]                    //FileData
  sub edx,[ebp-004h]                    //Delta = FileData - ImageBaseOrg
  je @DynLoader_PEBTEBFixup

 @DynLoader_RelocFixup:
  mov eax,[ebp-00Ch]                    //FileData
  mov ebx,eax
  add ebx,[ebx+03Ch]                    //TImageDosHeader._lfanew
  mov ebx,[ebx+0A0h]
//IMAGE_DIRECTORY_ENTRY_BASERELOC.VirtualAddress
  test ebx,ebx
  jz @DynLoader_PEBTEBFixup
  add ebx,eax
 @DynLoader_RelocFixup_block:
  mov eax,[ebx+004h]                    //ImageBaseRelocation.SizeOfBlock
  test eax,eax
  jz @DynLoader_PEBTEBFixup
  lea ecx,[eax-008h]                    //ImageBaseRelocation.SizeOfBlock -
SizeOf(TImageBaseRelocation)
  shr ecx,001h                          //(ImageBaseRelocation.SizeOfBlock -
SizeOf(TImageBaseRelocation)) div SizeOf(Word)
  lea edi,[ebx+008h]                    //PImageBaseRelocation +
SizeOf(TImageBaseRelocation)
 @DynLoader_RelocFixup_do_entry:
  movzx eax,word ptr [edi]              //Entry
  push edx
```

```
  mov edx,eax
  shr eax,00Ch                          //Type = Entry shr 12

  mov esi,[ebp-00Ch]                    //FileData
  and dx,00FFFh
  add esi,[ebx]                         //FileData +
ImageBaseRelocation.VirtualAddress
  add esi,edx                           //FileData +
ImageBaseRelocation.VirtualAddress+Entry and $0FFF
  pop edx

 @DynLoader_RelocFixup_HIGH:
  dec eax
  jnz @DynLoader_RelocFixup_LOW
  mov eax,edx
  shr eax,010h                          //HiWord(Delta)
  jmp @DynLoader_RelocFixup_LOW_fixup
 @DynLoader_RelocFixup_LOW:
  dec eax
  jnz @DynLoader_RelocFixup_HIGHLOW
  movzx eax,dx                          //LoWord(Delta)
 @DynLoader_RelocFixup_LOW_fixup:
  add word ptr [esi],ax
  jmp @DynLoader_RelocFixup_next_entry
 @DynLoader_RelocFixup_HIGHLOW:
  dec eax
  jnz @DynLoader_RelocFixup_next_entry
  add [esi],edx

 @DynLoader_RelocFixup_next_entry:
  inc edi
  inc edi                               //Inc(Entry)
  loop @DynLoader_RelocFixup_do_entry

 @DynLoader_RelocFixup_next_base:
  add ebx,[ebx+004h]                    //ImageBaseRelocation +
ImageBaseRelocation.SizeOfBlock
  jmp @DynLoader_RelocFixup_block

 @DynLoader_PEBTEBFixup:
  //існують погані вказівники у InLoadOrderModuleList, ми змінюємо базу нашого
модуля
  //і якщо ми - програма (не dll), ми повинні змінювати базову адресу у PEB
також
  //Для програм написаних на VB, це потрібно робити тут. так як бібліотеки
читають звідси дані
  //у ImportFixup блоку
//  int 3
  mov ecx,[ebp-00Ch]                    //FileData
  mov edx,[ebp-050h]                    //ImageBase
  add [ebp-04Ch],edx                    //ImageEntryPoint

  mov eax,fs:[000000030h]               //TEB.PPEB
  cmp dword ptr [ebp-044h],IMAGE_TYPE_EXE // тип змінених даних= IMAGE_TYPE_EXE
  jnz @DynLoader_in_module_list
  mov [eax+008h],ecx                    //PEB.ImageBaseAddr -> rewrite old
imagebase
 @DynLoader_in_module_list:
  mov eax,[eax+00Ch]                    //PEB.LoaderData
  mov eax,[eax+00Ch]                    //LoaderData.InLoadOrderModuleList

  //тепер неможливо знайти модуль у списку (але та же база, той же розмір та та
жа точка входу)
  mov esi,eax                           //перший запис

 @DynLoader_in_module_list_one:
  mov edx,[eax+018h]                    //InLoadOrderModuleList.BaseAddress
  cmp edx,[ebp-050h]                    //ImageBase
  jnz @DynLoader_in_module_list_next
```

```
  mov edx,[eax+01Ch]                    //InLoaderOrderModuleList.EntryPoint
  cmp edx,[ebp-04Ch]                    //ImageEntryPoint
  jnz @DynLoader_in_module_list_next
  mov edx,[eax+020h]                    //InLoaderOrderModuleList.SizeOfImage
  cmp edx,[ebp-048h]                    //ImageSize
  jnz @DynLoader_in_module_list_next
  mov [eax+018h],ecx                    //InLoadOrderModuleList.BaseAddress ->
перезапис старого запису
  add ecx,[ebp-01D0h]
//+NtHeaders.OptionalHeader.AddressOfEntryPoint
  mov [eax+01Ch],ecx                    //InLoadOrderModuleList.EntryPoint ->
перезапис старої точки входу
  mov ecx,[ebp-01A8h]                   //NtHeaders.OptionalHeader.SizeOfImage
  mov [eax+020h],ecx                    //InLoaderOrderModuleList.SizeOfImage ->
перезапис строго розміру блоку
  jmp @DynLoader_ImportFixup

 @DynLoader_in_module_list_next:
  cmp [eax],esi                         //InLoadOrderModuleList.Flink ?= перший
запис
  jz @DynLoader_ImportFixup
  mov eax,[eax]                         //запис = InLoadOrderModuleList.Flink
  jmp @DynLoader_in_module_list_one


 @DynLoader_ImportFixup:
  mov ebx,[ebp-0178h]
//NtHeaders.OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_IMPORT].VirtualAd
dress
  test ebx,ebx
  jz @DynLoader_export_fixup
  mov esi,[ebp-00Ch]                    //FileData
  add ebx,esi                           //FileData +
NtHeaders.OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_IMPORT].VirtualAddr
ess
 @DynLoader_ImportFixup_module:
  mov eax,[ebx+00Ch]                    //TImageImportDescriptor.Name
  test eax,eax
  jz @DynLoader_export_fixup

  mov ecx,[ebx+010h]                    //TImageImportDescriptor.FirstThunk
  add ecx,esi
  mov [ebp-03Ch],ecx                    // Заглушка
  mov ecx,[ebx]                         //TImageImportDescriptor.Characteristics
  test ecx,ecx
  jnz @DynLoader_ImportFixup_table
  mov ecx,[ebx+010h]
 @DynLoader_ImportFixup_table:
  add ecx,esi
  mov [ebp-040h],ecx                    //HintName
  add eax,esi                           //TImageImportDescriptor.Name + FileData
= ModuleName
  push eax                              //lpLibFileName
  mov eax,[ebp+010h]                    //ImportThunk.LoadLibrary
  call [eax]                            //LoadLibrary
  test eax,eax
  jz @DynLoader_end
  mov edi,eax
 @DynLoader_ImportFixup_loop:
  mov ecx,[ebp-040h]                    //HintName
  mov edx,[ecx]                         //TImageThunkData.Ordinal
  test edx,edx
  jz @DynLoader_ImportFixup_next_module
  test edx,080000000h                   //імпорт порядку?
  jz @DynLoader_ImportFixup_by_name
  and edx,07FFFFFFFh                    //беремо порядок
  jmp @DynLoader_ImportFixup_get_addr
 @DynLoader_ImportFixup_by_name:
```

```
  add edx,esi                           //TImageThunkData.Ordinal + FileData =
OrdinalName
  inc edx
  inc edx                               //OrdinalName.Name
 @DynLoader_ImportFixup_get_addr:
  push edx                              //lpProcName
  push edi                              //hModule
  mov eax,[ebp+00Ch]                    // реалізація ImportThunk.GetProcAddress
  call [eax]                            //GetProcAddress
  mov ecx,[ebp-03Ch]                    //HintName
  mov [ecx],eax
  add dword ptr [ebp-03Ch],004h         // Заглушка -> next Thunk
  add dword ptr [ebp-040h],004h         //HintName -> next HintName
  jmp @DynLoader_ImportFixup_loop
 @DynLoader_ImportFixup_next_module:
  add ebx,014h                          //SizeOf(TImageImportDescriptor)
  jmp @DynLoader_ImportFixup_module

 @DynLoader_export_fixup:
  // перетворюємо усі модулі та шукаємо блок IAT для нашого модуля, потім
змінюємо базу образу  у усіх імпортуємих модулях
//  int 3
  mov eax,fs:[000000030h]               //TEB.PPEB
  mov eax,[eax+00Ch]                    //PEB.LoaderData
  mov ebx,[eax+00Ch]                    //LoaderData.InLoadOrderModuleList
  mov [ebp-070h],ebx                    //FirstModule

 @DynLoader_export_fixup_process_module:
  mov edx,[ebx+018h]                    //InLoadOrderModuleList.BaseAddress
  cmp edx,[ebp-050h]                    //ImageBase
  jz @DynLoader_export_fixup_next

  push edx
  push 004h                             //ucb
  push edx                              //lp
  call [ebp-07Ch]                       //IsBadReadPtr
  pop edx
  test eax,eax
  jnz @DynLoader_export_fixup_next

  mov edi,edx
  add edi,[edi+03Ch]                    //TImageDosHeader._lfanew
  mov edi,[edi+080h]
//TImageNtHeaders.OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_IMPORT].Vir
tualAddress
  test edi,edi
  jz @DynLoader_export_fixup_next
  add edi,edx                           //+ module.ImageBase
 @DynLoader_export_fixup_check_idt:
  xor eax,eax
  push edi
  push 005h                             //sizeof(ImportDirectoryTable)/4
  pop ecx
  rep scasd                             //тест для неіснуючої директорії
  pop edi
  jz @DynLoader_export_fixup_next

  mov esi,[edi+010h]                    //Блок
імпортування.ImportAddressTableRVA
  add esi,[ebx+018h]                    //+ module.ImageBase
  mov eax,[esi]                         //first IAT func address
  sub eax,[ebp-050h]                    //- ImageBase
  jb @DynLoader_export_fixup_next_idir  // це не перетворюється
  cmp eax,[ebp-048h]                    //ImageSize
  jbe @DynLoader_export_fixup_prefixaddr // це перетворюється

 @DynLoader_export_fixup_next_idir:
  add edi,014h                          //+ sizeof(IDT) = next IDT
  jmp @DynLoader_export_fixup_check_idt
```

```
 @DynLoader_export_fixup_prefixaddr:
  push 01Ch                             //dwLength =
sizeof(MemoryBasicInformation)
  lea eax,[ebp-09Ch]                    //MemoryBasicInformation
  push eax                              //lpBuffer
  push esi                              //lpAddress
  call [ebp-078h]                       //VirtualQuery

  lea eax,[ebp-088h]                    //MemoryBasicInformation.Protect
  push eax                              //lpflOldProtect
  push PAGE_READWRITE                   //flNewProtect
  push dword ptr [ebp-090h]             //dwSize =
MemoryBasicInformation.RegionSize
  push dword ptr [ebp-09Ch]             //lpAddress =
MemoryBasicInformation.BaseAddress
  call [ebp-074h]                       //VirtualProtect
  test eax,eax
  jz @DynLoader_export_fixup_next

  push edi
  mov edi,esi
 @DynLoader_export_fixup_fixaddr:
  lodsd
  test eax,eax
  jz @DynLoader_export_fixup_protect_back
  sub eax,[ebp-050h]                    //- ImageBase
  add eax,[ebp-00Ch]                    //+ FileData
  stosd
  jmp @DynLoader_export_fixup_fixaddr

 @DynLoader_export_fixup_protect_back:
  lea eax,[ebp-084h]                    //MemoryBasicInformation.Type (just need
some pointer)
  push eax                              //lpflOldProtect
  push dword ptr [ebp-088h]             //flNewProtect =
MemoryBasicInformation.Protect
  push dword ptr [ebp-090h]             //dwSize =
MemoryBasicInformation.RegionSize
  push dword ptr [ebp-09Ch]             //lpAddress =
MemoryBasicInformation.BaseAddress
  call [ebp-074h]                       //VirtualProtect
  pop edi
  jmp @DynLoader_export_fixup_next_idir

 @DynLoader_export_fixup_next:
  mov ebx,[ebx]
  cmp ebx,[ebp-070h]                    //InLoadOrderModuleList.Flink ?=
FirstModule
  jnz @DynLoader_export_fixup_process_module

 @DynLoader_run:
//  int 3
  mov eax,[ebp-01D0h]
//NtHeaders.OptionalHeader.AddressOfEntryPoint
  add eax,[ebp-00Ch]
//NtHeaders.OptionalHeader.AddressOfEntryPoint + FileData = EntryPoint

 @DynLoader_end:
  mov ecx,[ebp-00Ch]                    // нам небхідно FileData
  pop esi
  pop edi
  pop ebx
  leave
  ret 00Ch

 @DynLoader_fill_image_info:
```

```
  //ці величини дають інформацію про наш образ, інформація заповнена раніше, ніж
DynLoader буде поміщений у кінцеву прграму, ми знаходимо їх компенсацію, яка
виходить з DynLoader_end, та шукає ознаку DYN_LOADER_END_MAGIC
  mov [ebp-044h],012345678h             //ImageType
  mov [ebp-048h],012345678h             //ImageSize
  mov [ebp-04Ch],012345678h             //ImageEntryPoint
  mov [ebp-054h],012345678h             //OrgImageSize
  ret
  dd DYN_LOADER_END_MAGIC
end;
procedure DynLoader_end; assembler; asm end;


procedure DynCoder(AAddr:Pointer;ASize:Cardinal;AKey:Pointer); assembler;
stdcall;
//розбиваємо ключ на декілький блоків у пам'яті
asm
 @Coder_begin:
  push edi
  push esi

 @Coder_main_loop:
  mov edi,[ebp+008h]            //AAddr
  mov ecx,[ebp+00Ch]            //ASize
  shr ecx,002h
 @Coder_pre_code:
  mov esi,[ebp+010h]            //AKey
 @Coder_code:
  mov eax,[esi]
  test eax,0FF000000h
  jz @Coder_pre_code
 @Coder_do_code:
  add eax,ecx
  xor eax,[edi]                 // розбиваємо це
  stosd                         // запам'ятовуємо це
  inc esi
  loop @Coder_code

 @Coder_end:
  pop esi
  pop edi
  leave
  ret 00Ch
end;

function
VirtAddrToPhysAddr(ANtHeaders:PImageNtHeaders;AVirtAddr:Pointer):Pointer;
//це повинно підтримувати tls, завантажуючи механізм повернення вказівника у
вихідні дані у старі PE данні о VA визначених AVirtAddr . або нулем, якщо ніякий
блок не містить ці данні
var
 LI:Integer;
 LPSection:PImageSectionHeader;
 LAddr:Cardinal;
begin
 Result:=nil;
 LAddr:=Cardinal(AVirtAddr)-ANtHeaders^.OptionalHeader.ImageBase;

LPSection:=Pointer(Cardinal(@ANtHeaders^.OptionalHeader)+ANtHeaders^.FileHeader.
SizeOfOptionalHeader);
 for LI:=0 to ANtHeaders^.FileHeader.NumberOfSections-1 do
 begin
  if (LPSection^.VirtualAddress<=Cardinal(LAddr)) and
(LPSection^.VirtualAddress+LPSection^.SizeOfRawData>Cardinal(LAddr)) and
(LPSection^.SizeOfRawData<>0) then
  begin
   Result:=Pointer(Cardinal(LPSection^.PointerToRawData)+LAddr-
LPSection^.VirtualAddress);
   Break;
```

```
  end;
  Inc(LPSection);
 end;
end;

function RVA2RAW(ANtHeader,AVirtImage:Pointer;ARVA:Cardinal):Pointer;
//Конвертуємо точку  RVA до RAW
var
 LPB:PByte;
begin
 Result:=nil;

LPB:=VirtAddrToPhysAddr(ANtHeader,Pointer(ARVA+PImageNtHeaders(ANtHeader)^.Optio
nalHeader.ImageBase));
 if LPB=nil then Exit;
 Inc(LPB,Cardinal(AVirtImage));
 Result:=LPB;
end;

function GetTlsCallbacksLen(ACallbacks:Pointer):Cardinal;
//підраховуємо розмір масиву tls який повертається
var
 LPC:PCardinal;
begin
 Result:=4;
 LPC:=ACallbacks;
 while LPC^<>0 do
 begin
  Inc(Result,4);
  Inc(LPC);
 end;
end;

function RoundSize(ASize,AAlignment:Cardinal):Cardinal;
// округляємо у більшу сторону
begin
 Result:=(ASize+AAlignment-1) div AAlignment*AAlignment;
end;

procedure GenerateRandomBuffer(ABuf:PByte;ASize:Cardinal);
//генеруємо буфер псевдовипадкових значень від 1 до 255
var
 LI:Integer;
begin
 for LI:=0 to ASize-1 do
 begin
  ABuf^:=Random($FE)+1;
  Inc(ABuf);
 end;
end;

procedure GenerateKey(AKey:PByte;ASize:Word);
//// генеруємо ключ для кодування даних
//ключ є псевдовипадковим буфером, який закінчується нулем0
begin
 GenerateRandomBuffer(AKey,ASize);
 PByte(Cardinal(AKey)+Cardinal(ASize)-1)^:=0;
end;

procedure ThrowTheDice(var ADice:Cardinal;ASides:Cardinal=6); overload;
// Закінчення нарізання ний блоки
begin
 ADice:=Random(ASides)+1;
end;

procedure ThrowTheDice(var ADice:Word;ASides:Word=6); overload;
// Закінчення нарізання ний блоки
begin
 ADice:=Random(ASides)+1;
```

```
end;

procedure ThrowTheDice(var ADice:Byte;ASides:Byte=6); overload;
// Закінчення нарізання ний блоки
begin
 ADice:=Random(ASides)+1;
end;

function RandomReg32All:Byte;
// Вибір одного з eax,ecx,edx,ebx,esp,ebp,esi,edi
begin
 Result:=Random(Reg32Count);
end;

function RandomReg16All:Byte;
// Вибір одного з ax,cx,dx,bx,sp,bp,si,di
begin
 Result:=Random(Reg16Count);
end;

function RandomReg8ABCD:Byte;
// Вибір одного з al,cl,dl,bl,ah,ch,dh,bh
begin
 Result:=Random(Reg8Count);
end;

function RandomReg32Esp:Byte;
// Вибір одного з eax,ecx,edx,ebx,-,ebp,esi,edi
begin
 Result:=Random(Reg32Count-1);
 if Result=REG_ESP then Result:=7;
end;

function RandomReg32EspEbp:Byte;
// Вибір одного з eax,ecx,edx,ebx,-,-,esi,edi
begin
 Result:=Random(Reg32Count-2);
 if Result=REG_ESP then Result:=6
 else if Result=REG_EBP then Result:=7;
end;

procedure PutRandomBuffer(var AMem:PByte;ASize:Cardinal);
begin
 GenerateRandomBuffer(AMem,ASize);
 Inc(AMem,ASize);
end;

function Bswap(var AMem:PByte;AReg:Byte):Byte;
begin
 Result:=2;
 AMem^:=$0F;                          //bswap
 Inc(AMem);
 AMem^:=$C8+AReg;                     //reg32
 Inc(AMem);
end;

function Stosd(var AMem:PByte):Byte;
begin
 Result:=1;
 AMem^:=$AB;                          //stosd
 Inc(AMem);
end;

function Movsd(var AMem:PByte):Byte;
begin
 Result:=1;
 AMem^:=$A5;                          //movsd
 Inc(AMem);
end;
```

```
function Ret(var AMem:PByte):Byte;
begin
 Result:=1;
 AMem^:=$C3;                              //повернення
 Inc(AMem);
end;

procedure Ret16(var AMem:PByte;AVal:Word);
begin
 AMem^:=$C2;                              //повернення
 Inc(AMem);
 PWord(AMem)^:=AVal;                      //поверненняval
 Inc(AMem,2);
end;

procedure RelJmpAddr32(var AMem:PByte;AAddr:Cardinal);
begin
 AMem^:=$E9;                              //jmp
 Inc(AMem);
 PCardinal(AMem)^:=AAddr;
 Inc(AMem,4);
end;

procedure RelJmpAddr8(var AMem:PByte;AAddr:Byte);
begin
 AMem^:=$EB;                              //jmp
 Inc(AMem);
 AMem^:=AAddr;                            //Addr8
 Inc(AMem);
end;


procedure RelJzAddr32(var AMem:PByte;AAddr:Cardinal);
begin
 AMem^:=$0F;                              // умовний перехід
 Inc(AMem);
 AMem^:=$84;                              // якщо дорівнює нулю
 Inc(AMem);
 PCardinal(AMem)^:=AAddr;
 Inc(AMem,4);
end;

procedure RelJnzAddr32(var AMem:PByte;AAddr:Cardinal);
begin
 AMem^:=$0F;                              // умовний перехід
 Inc(AMem);
 AMem^:=$85;                              // якщо не дорівнює нулю
 Inc(AMem);
 PCardinal(AMem)^:=AAddr;
 Inc(AMem,4);
end;

procedure RelJbAddr32(var AMem:PByte;AAddr:Cardinal);
begin
 AMem^:=$0F;                              // умовний перехід
 Inc(AMem);
 AMem^:=$82;                              // якщо нижче
 Inc(AMem);
 PCardinal(AMem)^:=AAddr;
 Inc(AMem,4);
end;

procedure RelJzAddr8(var AMem:PByte;AAddr:Byte);
begin
 AMem^:=$74;                              //jz
 Inc(AMem);
 AMem^:=AAddr;                            //addr8
 Inc(AMem);
```

```
end;

procedure RelJnzAddr8(var AMem:PByte;AAddr:Byte);
begin
 AMem^:=$75;                            //jnz
 Inc(AMem);
 AMem^:=AAddr;                          //addr8
 Inc(AMem);
end;

function JmpRegMemIdx8(var AMem:PByte;AReg,AIdx:Byte):Byte;
begin
 Result:=3;
 AMem^:=$FF;                            //jmp
 Inc(AMem);
 AMem^:=$60+AReg;                       //regmem
 InC(AMem);
 if AReg=REG_ESP then
 begin
  Inc(Result);
  AMem^:=$24;                           //esp
  Inc(AMem);
 end;
 AMem^:=AIdx;                           //idx8
 Inc(AMem);
end;

function PushRegMem(var AMem:PByte;AReg:Byte):Byte;
begin
 Result:=2;
 AMem^:=$FF;                            //push
 Inc(AMem);
 if AReg=REG_EBP then
 begin
  Inc(Result);
  AMem^:=$75;                           //ebp
  Inc(AMem);
  AMem^:=$00;                           //+0
 end else AMem^:=$30+AReg;              //regmem
 Inc(AMem);
 if AReg=REG_ESP then
 begin
  Inc(Result);
  AMem^:=$24;                           //esp
  Inc(AMem);
 end;
end;

procedure PushReg32(var AMem:PByte;AReg:Byte);
begin
 AMem^:=$50+AReg;                       //push reg
 Inc(AMem);
end;

function PushReg32Rand(var AMem:PByte):Byte;
begin
 Result:=RandomReg32Esp;
 PushReg32(AMem,Result);
end;

procedure PopReg32(var AMem:PByte;AReg:Byte);
begin
 AMem^:=$58+AReg;                       //pop reg
 Inc(AMem);
end;

function PopReg32Idx(var AMem:PByte;AReg:Byte;AIdx:Cardinal):Byte;
begin
 Result:=6;
```

```
 AMem^:=$8F;                            //pop
 Inc(AMem);
 AMem^:=$80+AReg;                       //reg32
 Inc(AMem);
 if AReg=REG_ESP then
 begin
  AMem^:=$24;                           //esp
  Inc(AMem);
  Inc(Result);
 end;
 PCardinal(AMem)^:=AIdx;                //+ idx
 InC(AMem,4);
end;

procedure RelCallAddr(var AMem:PByte;AAddr:Cardinal);
begin
 AMem^:=$E8;                            //call
 Inc(AMem);
 PCardinal(AMem)^:=AAddr;               //Addr
 Inc(AMem,4);
end;

procedure MovReg32Reg32(var AMem:PByte;AReg1,AReg2:Byte);
begin
 AMem^:=$89;                            //mov
 Inc(AMem);
 AMem^:=AReg2*8+AReg1+$C0;              //reg32,reg32
 Inc(AMem);
end;

procedure AddReg32Reg32(var AMem:PByte;AReg1,AReg2:Byte);
begin
 AMem^:=$01;                            //add
 Inc(AMem);
 AMem^:=AReg2*8+AReg1+$C0;              //reg32,reg32
 Inc(AMem);
end;

function AddReg32RegMem(var AMem:PByte;AReg1,AReg2:Byte):Byte;
begin
 Result:=2;
 AMem^:=$03;                            //add
 Inc(AMem);
 if AReg2=REG_EBP then
 begin
  Inc(Result);
  AMem^:=AReg1*8+$45;                   //reg32,ebp
  Inc(AMem);
  AMem^:=$00;                           //+0
 end else AMem^:=AReg1*8+AReg2;         //reg32,regmem
 Inc(AMem);
 if AReg2=REG_ESP then
 begin
  Inc(Result);
  AMem^:=$24;                           //esp
  Inc(AMem);
 end;
end;

function AddRegMemReg32(var AMem:PByte;AReg1,AReg2:Byte):Byte;
begin
 Result:=2;
 AMem^:=$01;                            //add
 Inc(AMem);
 if AReg1=REG_EBP then
 begin
  Inc(Result);
  AMem^:=AReg2*8+$45;                   //regmem,ebp
  Inc(AMem);
```

```
  AMem^:=$00;                           //+0
 end else AMem^:=AReg2*8+AReg1;         //regmem,reg
 Inc(AMem);
 if AReg1=REG_ESP then
 begin
  Inc(Result);
  AMem^:=$24;                           //esp
  Inc(AMem);
 end;
end;

procedure AddReg32Num8(var AMem:PByte;AReg,ANum:Byte);
begin
 AMem^:=$83;                            //add
 Inc(AMem);
 AMem^:=$C0+AReg;                       //reg32
 Inc(AMem);
 AMem^:=ANum;                           //num8
 Inc(AMem);
end;

procedure MovReg32Num32(var AMem:PByte;AReg:Byte;ANum:Cardinal);
begin
 AMem^:=$B8+AReg;                       //mov reg32
 Inc(AMem);
 PCardinal(AMem)^:=ANum;                //num32
 Inc(AMem,4);
end;

function MovReg32IdxNum32(var AMem:PByte;AReg:Byte;AIdx,ANum:Cardinal):Byte;
begin
 Result:=10;
 AMem^:=$C7;                            //mov
 Inc(AMem);
 AMem^:=$80+AReg;                       //reg32
 Inc(AMem);
 if AReg=REG_ESP then
 begin
  Inc(Result);
  AMem^:=$24;                           //esp
  Inc(AMem);
 end;
 PCardinal(AMem)^:=AIdx;                //+ idx
 Inc(AMem,4);
 PCardinal(AMem)^:=ANum;                //Num32
 Inc(AMem,4);
end;

procedure MovReg32Reg32IdxNum32(var AMem:PByte;AReg1,AReg2:Byte;ANum:Cardinal);
//обидва AReg не повинні бути REG_ESP або REG_EBP
begin
 if AReg1=REG_ESP then begin AReg1:=AReg2; AReg2:=REG_ESP; end;
 if AReg2=REG_EBP then begin AReg2:=AReg1; AReg1:=REG_EBP; end;
 AMem^:=$C7;                            //mov
 Inc(AMem);
 AMem^:=$04;
 Inc(AMem);
 AMem^:=AReg1*8+AReg2;
 Inc(AMem);
 PCardinal(AMem)^:=ANum;                //Num32
 Inc(AMem,4);
end;

function MovReg32RegMem(var AMem:PByte;AReg1,AReg2:Byte):Byte;
begin
 Result:=2;
 AMem^:=$8B;                            //mov
 Inc(AMem);
 if AReg2=REG_EBP then
```

```
 begin
  Inc(Result);
  AMem^:=AReg1*8+$45;                  //reg32,ebp
  Inc(AMem);
  AMem^:=$00;                          //+0
 end else AMem^:=AReg1*8+AReg2;        //reg32,regmem
 Inc(AMem);
 if AReg2=REG_ESP then
 begin
  Inc(Result);
  AMem^:=$24;                          //esp
  Inc(AMem);
 end;
end;

function MovRegMemReg32(var AMem:PByte;AReg1,AReg2:Byte):Byte;
begin
 Result:=2;
 AMem^:=$89;                          //mov
 Inc(AMem);
 if AReg1=REG_EBP then
 begin
  Inc(Result);
  AMem^:=AReg2*8+$45;                  //reg32,ebp
  Inc(AMem);
  AMem^:=$00;                          //+0
 end else AMem^:=AReg2*8+AReg1;        //reg32,regmem
 Inc(AMem);
 if AReg1=REG_ESP then
 begin
  Inc(Result);
  AMem^:=$24;                          //esp
  Inc(AMem);
 end;
end;

function MovReg32RegMemIdx8(var AMem:PByte;AReg1,AReg2,AIdx:Byte):Byte;
begin
 Result:=3;
 AMem^:=$8B;                          //mov
 Inc(AMem);
 AMem^:=AReg1*8+AReg2+$40;            //AReg1,AReg2
 Inc(AMem);
 if AReg2=REG_ESP then
 begin
  Inc(Result);
  AMem^:=$24;                          //esp
  Inc(AMem);
 end;
 AMem^:=AIdx;                         //AIdx
 Inc(AMem);
end;

procedure PushNum32(var AMem:PByte;ANum:Cardinal);
begin
 AMem^:=$68;                          //push
 Inc(AMem);
 PCardinal(AMem)^:=ANum;
 Inc(AMem,4);
end;

procedure JmpReg32(var AMem:PByte;AReg:Byte);
begin
 AMem^:=$FF;                          //jmp | call
 Inc(AMem);
 AMem^:=$E0+AReg;                     //reg32
 Inc(AMem);
end;
```

```
procedure CallReg32(var AMem:PByte;AReg:Byte);
begin
 AMem^:=$FF;                            //jmp | call
 Inc(AMem);
 AMem^:=$D0+AReg;                       //reg32
 Inc(AMem);
end;

procedure Cld(var AMem:PByte);
begin
 AMem^:=$FC;                            //cld
 Inc(AMem);
end;

procedure Std(var AMem:PByte);
begin
 AMem^:=$FD;                            //std
 Inc(AMem);
end;

procedure Nop(var AMem:PByte);
begin
 AMem^:=$90;                            //nop
 Inc(AMem);
end;

procedure Stc(var AMem:PByte);
begin
 AMem^:=$F9;                            //stc
 Inc(AMem);
end;

procedure Clc(var AMem:PByte);
begin
 AMem^:=$F8;                            //clc
 Inc(AMem);
end;

procedure Cmc(var AMem:PByte);
begin
 AMem^:=$F5;                            //cmc
 Inc(AMem);
end;

procedure XchgReg32Rand(var AMem:PByte);
begin
 AMem^:=$87;                            //xchg
 Inc(AMem);
 AMem^:=$C0+RandomReg32All*9;           //reg32
 Inc(AMem);
end;

function XchgReg32Reg32(var AMem:PByte;AReg1,AReg2:Byte):Byte;
begin
 if AReg2=REG_EAX then begin AReg2:=AReg1; AReg1:=REG_EAX end;
 if AReg1=REG_EAX then ThrowTheDice(Result,2)
 else Result:=2;
 if Result=2 then
 begin
  AMem^:=$87;                           //xchg
  Inc(AMem);
  AMem^:=$C0+AReg2*8+AReg1;             //reg32
 end else AMem^:=$90+AReg2;             //xchg eax,reg32
 Inc(AMem);
end;

procedure MovReg32Rand(var AMem:PByte);
begin
 AMem^:=$8B;                            //mov
```

```
 Inc(AMem);
 AMem^:=$C0+RandomReg32All*9;          //reg32
 Inc(AMem);
end;

procedure IncReg32(var AMem:PByte;AReg:Byte);
begin
 AMem^:=$40+AReg;                      //inc reg32
 Inc(AMem);
end;

procedure DecReg32(var AMem:PByte;AReg:Byte);
begin
 AMem^:=$48+AReg;                      //dec reg32
 Inc(AMem);
end;

function IncReg32Rand(var AMem:PByte):Byte;
begin
 Result:=RandomReg32All;
 IncReg32(AMem,Result);
end;

function DecReg32Rand(var AMem:PByte):Byte;
begin
 Result:=RandomReg32All;
 DecReg32(AMem,Result);
end;

function LeaReg32Reg32(var AMem:PByte;AReg1,AReg2:Byte):Byte;
begin
 Result:=2;
 AMem^:=$8D;                           //mov
 Inc(AMem);
 if AReg2=REG_EBP then
 begin
  Inc(Result);
  AMem^:=AReg1*8+$45;                  //reg32,ebp
  Inc(AMem);
  AMem^:=$00;                          //+0
 end else AMem^:=AReg1*8+AReg2;        //reg32,regmem
 Inc(AMem);
 if AReg2=REG_ESP then
 begin
  Inc(Result);
  AMem^:=$24;                          //esp
  Inc(AMem);
 end;
end;

procedure LeaReg32Rand(var AMem:PByte);
begin
 AMem^:=$8D;                           //lea
 Inc(AMem);
 AMem^:=$00+RandomReg32EspEbp*9;       //reg32
 Inc(AMem);
end;

procedure LeaReg32Addr32(var AMem:PByte;AReg,AAddr:Cardinal);
begin
 AMem^:=$8D;                           //lea
 Inc(AMem);
 AMem^:=$05+AReg*8;                    //reg32
 Inc(AMem);
 PCardinal(AMem)^:=AAddr;              //addr32
 Inc(AMem,4);
end;
```

```
procedure TestReg32Rand(var AMem:PByte);
begin
 AMem^:=$85;                            //test
 Inc(AMem);
 AMem^:=$C0+RandomReg32All*9;           //reg32
 Inc(AMem);
end;

procedure OrReg32Rand(var AMem:PByte);
begin
 AMem^:=$0B;                            //or
 Inc(AMem);
 AMem^:=$C0+RandomReg32All*9;           //reg32
 Inc(AMem);
end;

procedure AndReg32Rand(var AMem:PByte);
begin
 AMem^:=$23;                            //and
 Inc(AMem);
 AMem^:=$C0+RandomReg32All*9;           //reg32
 Inc(AMem);
end;

procedure TestReg8Rand(var AMem:PByte);
var
 LReg8:Byte;
begin
 LReg8:=RandomReg8ABCD;
 AMem^:=$84;                            //test
 Inc(AMem);
 AMem^:=$C0+LReg8*9;                    //reg8
 Inc(AMem);
end;

procedure OrReg8Rand(var AMem:PByte);
var
 LReg8:Byte;
begin
 LReg8:=RandomReg8ABCD;
 AMem^:=$0A;                            //or
 Inc(AMem);
 AMem^:=$C0+LReg8*9;                    //reg8
 Inc(AMem);
end;

procedure AndReg8Rand(var AMem:PByte);
var
 LReg8:Byte;
begin
 LReg8:=RandomReg8ABCD;
 AMem^:=$22;                            //and
 Inc(AMem);
 AMem^:=$C0+LReg8*9;                    //reg8
 Inc(AMem);
end;

procedure CmpRegRegNum8Rand(var AMem:PByte);
var
 LRnd:Byte;
begin
 LRnd:=Random(3);
 AMem^:=$3A+LRnd;                       //cmp
 Inc(AMem);
 if LRnd<2 then LRnd:=Random($40)+$C0
 else LRnd:=Random($100);
 AMem^:=LRnd;                           //reg16 | reg32 | num16
 Inc(AMem);
end;
```

```
function CmpReg32Reg32(var AMem:Pbyte;AReg1,AReg2:Byte):Byte;
begin
 Result:=2;
 AMem^:=$39;                              //cmp
 Inc(AMem);
 AMem^:=$C0+AReg1+AReg2*8;                //reg1,reg2
 Inc(AMem);
end;

procedure CmpReg32Num8(var AMem:PByte;AReg,ANum:Byte);
begin
 AMem^:=$83;                              //cmp
 Inc(AMem);
 AMem^:=$F8+AReg;                         //reg32
 Inc(AMem);
 AMem^:=ANum;                             //num8
 Inc(AMem);
end;

procedure CmpReg32RandNum8(var AMem:PByte;AReg:Byte);
begin
 CmpReg32Num8(AMem,AReg,Random($100));
end;

procedure CmpRandReg32RandNum8(var AMem:PByte);
begin
 CmpReg32RandNum8(AMem,RandomReg32All);
end;

procedure JmpNum8(var AMem:PByte;ANum:Byte);
var
 LRnd:Byte;
begin
 LRnd:=Random(16);
 if LRnd=16 then AMem^:=$EB               //jmp
 else AMem^:=$70+LRnd;                    //cond jmp
 Inc(AMem);
 AMem^:=ANum;                             //num8
 Inc(AMem);
end;

procedure SubReg32Reg32(var AMem:PByte;AReg1,AReg2:Byte);
begin
 AMem^:=$29;                              //sub
 Inc(AMem);
 AMem^:=AReg2*8+AReg1+$C0;                //reg32,reg32
 Inc(AMem);
end;


procedure SubReg32Num8(var AMem:PByte;AReg,ANum:Byte);
begin
 AMem^:=$83;                              //sub
 Inc(AMem);
 AMem^:=$E8+AReg;                         //reg32
 Inc(AMem);
 AMem^:=ANum;                             //num8
 Inc(AMem);
end;

function SubReg32Num8Rand(var AMem:PByte;ANum:Byte):Byte;
begin
 Result:=RandomReg32All;
 SubReg32Num8(AMem,Result,ANum);
end;

function AddReg32Num8Rand(var AMem:PByte;ANum:Byte):Byte;
begin
```

```
 Result:=RandomReg32All;
 AddReg32Num8(AMem,Result,ANum);
end;

procedure SubAlNum8(var AMem:PByte;ANum:Byte);
begin
 AMem^:=$2C;                          //sub al
 Inc(AMem);
 AMem^:=ANum;                         //num8
 Inc(AMem);
end;

procedure TestAlNum8(var AMem:PByte;ANum:Byte);
begin
 AMem^:=$A8;                          //test al
 Inc(AMem);
 AMem^:=ANum;                         //num8
 Inc(AMem);
end;

procedure TestAlNum8Rand(var AMem:PByte);
begin
 TestAlNum8(AMem,Random($100));
end;

procedure SubReg8Num8(var AMem:PByte;AReg,ANum:Byte);
begin
 AMem^:=$80;                          //sub
 Inc(AMem);
 AMem^:=$E8+AReg;                     //reg8
 Inc(AMem);
 AMem^:=ANum;                         //num8
 Inc(AMem);
end;

procedure SubReg8Num8Rand(var AMem:PByte;ANum:Byte);
var
 LReg8:Byte;
begin
 LReg8:=RandomReg8ABCD;
 SubReg8Num8(AMem,LReg8,ANum);
end;

procedure TestReg8Num8(var AMem:PByte;AReg,ANum:Byte);
begin
 AMem^:=$F6;                          //test
 Inc(AMem);
 AMem^:=$C0+AReg;                     //reg8
 Inc(AMem);
 AMem^:=ANum;                         //num8
 Inc(AMem);
end;

procedure TestReg8Num8Rand(var AMem:PByte);
begin
 TestReg8Num8(AMem,RandomReg8ABCD,Random($100));
end;

procedure AddReg8Num8(var AMem:PByte;AReg,ANum:Byte);
begin
 AMem^:=$80;                          //add
 Inc(AMem);
 AMem^:=$C0+AReg;                     //reg8
 Inc(AMem);
 AMem^:=ANum;                         //num8
 Inc(AMem);
end;

procedure AddReg8Num8Rand(var AMem:PByte;ANum:Byte);
```

```
var
 LReg8:Byte;
begin
 LReg8:=RandomReg8ABCD;
 AddReg8Num8(AMem,LReg8,ANum);
end;

procedure AddAlNum8(var AMem:PByte;ANum:Byte);
begin
 AMem^:=$04;                          //add al
 Inc(AMem);
 AMem^:=ANum;                         //num8
 Inc(AMem);
end;

procedure FNop(var AMem:PByte);
begin
 AMem^:=$D9;                          //fnop
 Inc(AMem);
 AMem^:=$D0;
 Inc(AMem);
end;

procedure OrReg16Rand(var AMem:PByte);
var
 LReg16:Byte;
begin
 LReg16:=RandomReg16All;
 AMem^:=$66;                          //or | test | and
 Inc(AMem);
 AMem^:=$0B;
 Inc(AMem);
 AMem^:=$C0+LReg16*9;                 //reg16
 Inc(AMem);
end;

procedure TestReg16Rand(var AMem:PByte);
var
 LReg16:Byte;
begin
 LReg16:=RandomReg16All;
 AMem^:=$66;                          //or | test | and
 Inc(AMem);
 AMem^:=$85;
 Inc(AMem);
 AMem^:=$C0+LReg16*9;                 //reg16
 Inc(AMem);
end;

procedure AndReg16Rand(var AMem:PByte);
var
 LReg16:Byte;
begin
 LReg16:=RandomReg16All;
 AMem^:=$66;                          //or | test | and
 Inc(AMem);
 AMem^:=$23;
 Inc(AMem);
 AMem^:=$C0+LReg16*9;                 //reg16
 Inc(AMem);
end;

procedure Cdq(var AMem:PByte);
begin
 AMem^:=$99;
 Inc(AMem);
end;

procedure ShlReg32Num8(var AMem:PByte;AReg,ANum:Byte);
```

```
begin
 AMem^:=$C1;                            //shl | shr | sal | sar
 Inc(AMem);
 AMem^:=$E0+AReg;                       //reg32
 Inc(AMem);
 AMem^:=ANum;                           //num8
 Inc(AMem);
end;

procedure ShlReg32RandNum8FullRand(var AMem:PByte);
begin
 ShlReg32Num8(AMem,RandomReg32All,Random(8)*$20);
end;

procedure ShrReg32Num8(var AMem:PByte;AReg,ANum:Byte);
begin
 AMem^:=$C1;                            //shl | shr | sal | sar
 Inc(AMem);
 AMem^:=$E8+AReg;                       //reg32
 Inc(AMem);
 AMem^:=ANum;                           //num8
 Inc(AMem);
end;

procedure ShrReg32RandNum8FullRand(var AMem:PByte);
begin
 ShrReg32Num8(AMem,RandomReg32All,Random(8)*$20);
end;

procedure SalReg32Num8(var AMem:PByte;AReg,ANum:Byte);
begin
 AMem^:=$C1;                            //shl | shr | sal | sar
 Inc(AMem);
 AMem^:=$F0+AReg;                       //reg32
 Inc(AMem);
 AMem^:=ANum;                           //num8
 Inc(AMem);
end;

procedure SalReg32RandNum8FullRand(var AMem:PByte);
begin
 SalReg32Num8(AMem,RandomReg32All,Random(8)*$20);
end;

procedure SarReg32Num8(var AMem:PByte;AReg,ANum:Byte);
begin
 AMem^:=$C1;                            //shl | shr | sal | sar
 Inc(AMem);
 AMem^:=$F8+AReg;                       //reg32
 Inc(AMem);
 AMem^:=ANum;                           //num8
 Inc(AMem);
end;

procedure SarReg32RandNum8FullRand(var AMem:PByte);
begin
 SarReg32Num8(AMem,RandomReg32All,Random(8)*$20);
end;

procedure RolReg8Num8(var AMem:PByte;AReg,ANum:Byte);
begin
 AMem^:=$C0;                            //rol | ror
 Inc(AMem);
 AMem^:=$C0+AReg;                       //reg8
 Inc(AMem);
 AMem^:=ANum;                           //num8
 Inc(AMem);
end;
```

```
procedure RolReg8RandNum8FullRand(var AMem:PByte);
begin
 RolReg8Num8(AMem,RandomReg8ABCD,Random($20)*8);
end;

procedure RorReg8Num8(var AMem:PByte;AReg,ANum:Byte);
begin
 AMem^:=$C0;                              //rol | ror
 Inc(AMem);
 AMem^:=$C8+AReg;                         //reg8
 Inc(AMem);
 AMem^:=ANum;                             //num8
 Inc(AMem);
end;

procedure RorReg8RandNum8FullRand(var AMem:PByte);
begin
 RorReg8Num8(AMem,RandomReg8ABCD,Random($20)*8);
end;

procedure RolReg32Num8(var AMem:PByte;AReg,ANum:Byte);
begin
 AMem^:=$C1;                              //rol | ror
 Inc(AMem);
 AMem^:=$C0+AReg;                         //reg32
 Inc(AMem);
 AMem^:=ANum;                             //num8
 Inc(AMem);
end;

procedure RolReg32RandNum8FullRand(var AMem:PByte);
begin
 RolReg32Num8(AMem,RandomReg32All,Random(8)*$20);
end;

procedure RorReg32Num8(var AMem:PByte;AReg,ANum:Byte);
begin
 AMem^:=$C1;                              //rol | ror
 Inc(AMem);
 AMem^:=$C8+AReg;                         //reg32
 Inc(AMem);
 AMem^:=ANum;                             //num8
 Inc(AMem);
end;

procedure RorReg32RandNum8FullRand(var AMem:PByte);
begin
 RorReg32Num8(AMem,RandomReg32All,Random(8)*$20);
end;

procedure TestAxNum16(var AMem:PByte;ANum:Word);
begin
 AMem^:=$66;                              //test ax
 Inc(AMem);
 AMem^:=$A9;
 Inc(AMem);
 PWord(AMem)^:=ANum;                      //num16
 Inc(AMem,2);
end;

procedure TestAxNum16Rand(var AMem:PByte);
begin
 TestAxNum16(AMem,Random($10000));
end;

procedure CmpAxNum16(var AMem:PByte;ANum:Word);
begin
 AMem^:=$66;                              //cmp ax
 Inc(AMem);
```

```
 AMem^:=$3D;
 Inc(AMem);
 PWord(AMem)^:=ANum;                    //num16
 Inc(AMem,2);
end;

procedure CmpAxNum16Rand(var AMem:PByte);
begin
 TestAxNum16(AMem,Random($10000));
end;

procedure PushNum8(var AMem:PByte;ANum:Byte);
begin
 AMem^:=$6A;                            //push
 Inc(AMem);
 AMem^:=ANum;                           //num8
 Inc(AMem);
end;

procedure PushNum8Rand(var AMem:PByte);
begin
 PushNum8(AMem,Random($100));
end;

function XorRand(var AMem:PByte):Word;
var
 LRnd:Byte;
 LRes:PWord;
begin
 LRes:=Pointer(AMem);
 LRnd:=Random(5);
 AMem^:=$30+LRnd;                       //xor
 Inc(AMem);
 if LRnd=4 then AMem^:=Random($100)     //num8
 else AMem^:=Random(7)*9+Random(8)+1+$C0;      //reg8 | reg32 but never the same
reg
 Inc(AMem);
 Result:=LRes^;
end;

procedure InvertXor(var AMem:PByte;AXor:Word);
begin
 PWord(AMem)^:=AXor;
 Inc(AMem,2);
end;

procedure DoubleXorRand(var AMem:PByte);
begin
 InvertXor(AMem,XorRand(AMem));
end;

function NotReg32(var AMem:PByte;AReg:Byte):Byte;
begin
 Result:=2;
 AMem^:=$F7;                            //not
 Inc(AMem);
 AMem^:=$D0+AReg;                       //reg32
 Inc(AMem);
end;

function NegReg32(var AMem:PByte;AReg:Byte):Byte;
begin
 Result:=2;
 AMem^:=$F7;                            //not
 Inc(AMem);
 AMem^:=$D8+AReg;                       //reg32
 Inc(AMem);
end;
```

```
function NotRand(var AMem:PByte):Word;
var
 LRes:PWord;
begin
 LRes:=Pointer(AMem);
 AMem^:=$F6+Random(1);                  //not
 Inc(AMem);
 AMem^:=$D0+Random(8);                  //reg8 | reg32
 Inc(AMem);
 Result:=LRes^;
end;

procedure InvertNot(var AMem:PByte;ANot:Word);
begin
 PWord(AMem)^:=ANot;
 Inc(AMem,2);
end;

procedure DoubleNotRand(var AMem:PByte);
begin
 InvertNot(AMem,NotRand(AMem));
end;

function NegRand(var AMem:PByte):Word;
var
 LRes:PWord;
begin
 LRes:=Pointer(AMem);
 AMem^:=$F6+Random(1);                  //neg
 Inc(AMem);
 AMem^:=$D8+Random(8);                  //reg8 | reg32
 Inc(AMem);
 Result:=LRes^;
end;

procedure InvertNeg(var AMem:PByte;ANeg:Word);
begin
 PWord(AMem)^:=ANeg;
 Inc(AMem,2);
end;

procedure DoubleNegRand(var AMem:PByte);
begin
 InvertNeg(AMem,NegRand(AMem));
end;

procedure AddReg16Num8(var AMem:PByte;AReg,ANum:Byte);
begin
 AMem^:=$66;                            //add | or | and | sub | xor | cmp
 Inc(AMem);
 AMem^:=$83;
 Inc(AMem);
 AMem^:=$C0+AReg;                       //reg16
 Inc(AMem);
 AMem^:=ANum;                           //num;
 Inc(AMem);
end;

procedure AddReg16Num8Rand(var AMem:PByte;ANum:Byte);
begin
 AddReg16Num8(AMem,RandomReg16All,ANum);
end;

procedure OrReg16Num8(var AMem:PByte;AReg,ANum:Byte);
begin
 AMem^:=$66;                            //add | or | and | sub | xor | cmp
 Inc(AMem);
 AMem^:=$83;
 Inc(AMem);
```

```
 AMem^:=$C8+AReg;                        //reg16
 Inc(AMem);
 AMem^:=ANum;                            //num
 Inc(AMem);
end;

procedure OrReg16Num8Rand(var AMem:PByte;ANum:Byte);
begin
 OrReg16Num8(AMem,RandomReg16All,ANum);
end;

procedure AndReg16Num8(var AMem:PByte;AReg,ANum:Byte);
begin
 AMem^:=$66;                             //add | or | and | sub | xor | cmp
 Inc(AMem);
 AMem^:=$83;
 Inc(AMem);
 AMem^:=$E0+AReg;                        //reg16
 Inc(AMem);
 AMem^:=ANum;                            //num
 Inc(AMem);
end;

procedure AndReg16Num8Rand(var AMem:PByte;ANum:Byte);
begin
 AndReg16Num8(AMem,RandomReg16All,ANum);
end;

procedure SubReg16Num8(var AMem:PByte;AReg,ANum:Byte);
begin
 AMem^:=$66;                             //add | or | and | sub | xor | cmp
 Inc(AMem);
 AMem^:=$83;
 Inc(AMem);
 AMem^:=$E8+AReg;                        //reg16
 Inc(AMem);
 AMem^:=ANum;                            //num
 Inc(AMem);
end;

procedure SubReg16Num8Rand(var AMem:PByte;ANum:Byte);
begin
 SubReg16Num8(AMem,RandomReg16All,ANum);
end;

procedure XorReg16Num8(var AMem:PByte;AReg,ANum:Byte);
begin
 AMem^:=$66;                             //add | or | and | sub | xor | cmp
 Inc(AMem);
 AMem^:=$83;
 Inc(AMem);
 AMem^:=$F0+AReg;                        //reg16
 Inc(AMem);
 AMem^:=ANum;                            //num
 Inc(AMem);
end;

procedure XorReg16Num8Rand(var AMem:PByte;ANum:Byte);
begin
 XorReg16Num8(AMem,RandomReg16All,ANum);
end;

procedure CmpReg16Num8(var AMem:PByte;AReg,ANum:Byte);
begin
 AMem^:=$66;                             //add | or | and | sub | xor | cmp
 Inc(AMem);
 AMem^:=$83;
 Inc(AMem);
 AMem^:=$F8+AReg;                        //reg16
```

```
 Inc(AMem);
 AMem^:=ANum;                          //num
 Inc(AMem);
end;

procedure CmpReg16Num8RandRand(var AMem:PByte);
begin
 CmpReg16Num8(AMem,RandomReg16All,Random($100));
end;

procedure RolReg16Num8(var AMem:PByte;AReg,ANum:Byte);
begin
 AMem^:=$66;                           //rol | ror
 Inc(AMem);
 AMem^:=$C1;
 Inc(AMem);
 AMem^:=$C0+AReg;                      //reg16
 Inc(AMem);
 AMem^:=ANum;                          //num8
 Inc(AMem);
end;

procedure RolReg16RandNum8FullRand(var AMem:PByte);
begin
 RolReg16Num8(AMem,RandomReg16All,Random($10)*$10);
end;

procedure RorReg16Num8(var AMem:PByte;AReg,ANum:Byte);
begin
 AMem^:=$66;                           //rol | ror
 Inc(AMem);
 AMem^:=$C1;
 Inc(AMem);
 AMem^:=$C1+AReg;                      //reg16
 Inc(AMem);
 AMem^:=ANum;                          //num8
 Inc(AMem);
end;

procedure RorReg16RandNum8FullRand(var AMem:PByte);
begin
 RorReg16Num8(AMem,RandomReg16All,Random($10)*$10);
end;

function XchgRand(var AMem:PByte):Word;
var
 LRes:PWord;
 LRnd:Byte;
begin
 LRes:=Pointer(AMem);
 LRnd:=Random(4);
 case LRnd of
  0,1:AMem^:=$66+LRnd;                 //xchg
  2,3:AMem^:=$86+LRnd-2;               //xchg
 end;
 Inc(AMem);
 case LRnd of
  0,1:AMem^:=$90+Random(8);            //reg16 | reg32
  2,3:AMem^:=$C0+Random($10);          //reg8 | reg32
 end;
 Inc(AMem);
 Result:=LRes^;
end;

procedure InvertXchg(var AMem:PByte;AXchg:Word);
begin
 PWord(AMem)^:=AXchg;
 Inc(AMem,2);
end;
```

```
procedure DoubleXchgRand(var AMem:PByte);
begin
 InvertXchg(AMem,XchgRand(AMem));
end;

procedure LoopNum8(var AMem:PByte;ANum:Byte);
begin
 AMem^:=$E2;                              //loop
 Inc(AMem);
 AMem^:=ANum;                             //ANum
 Inc(AMem);
end;

procedure JecxzNum8(var AMem:PByte;ANum:Byte);
begin
 AMem^:=$E3;                              //jecxz
 Inc(AMem);
 AMem^:=ANum;                             //ANum
 Inc(AMem);
end;

procedure MovzxEcxCl(var AMem:PByte);
begin
 AMem^:=$0F;                              //movzx
 Inc(AMem);
 AMem^:=$B6;
 Inc(AMem);
 AMem^:=$C9;                              //ecx:cx
 Inc(AMem);
end;

procedure MovReg32Reg32Rand(var AMem:PByte;AReg:Byte);
begin
 AMem^:=$8B;                              //mov
 Inc(AMem);
 AMem^:=$C0+8*AReg+RandomReg32All;        //reg32
 Inc(AMem);
end;

procedure CmpEaxNum32(var AMem:PByte;ANum:Cardinal);
begin
 AMem^:=$3D;                              //cmp eax
 Inc(AMem);
 PCardinal(AMem)^:=ANum;                  //num32
 Inc(AMem,4);
end;

procedure CmpEaxNum32Rand(var AMem:PByte);
begin
 CmpEaxNum32(AMem,Random($FFFFFFFF));
end;

procedure TestEaxNum32(var AMem:PByte;ANum:Cardinal);
begin
 AMem^:=$A9;                              //test eax
 Inc(AMem);
 PCardinal(AMem)^:=ANum;                  //num32
 Inc(AMem,4);
end;

procedure TestEaxNum32Rand(var AMem:PByte);
begin
 TestEaxNum32(AMem,Random($FFFFFFFF));
end;

procedure SubEaxNum32(var AMem:PByte;ANum:Cardinal);
begin
 AMem^:=$2D;                              //sub eax
```

```
 Inc(AMem);
 PCardinal(AMem)^:=ANum;                //num32
 Inc(AMem,4);
end;

procedure AddEaxNum32(var AMem:PByte;ANum:Cardinal);
begin
 AMem^:=$05;                            //add eax
 Inc(AMem);
 PCardinal(AMem)^:=ANum;                //num32
 Inc(AMem,4);
end;

procedure AndEaxNum32(var AMem:PByte;ANum:Cardinal);
begin
 AMem^:=$25;                            //and eax
 Inc(AMem);
 PCardinal(AMem)^:=ANum;                //num32
 Inc(AMem,4);
end;

procedure OrEaxNum32(var AMem:PByte;ANum:Cardinal);
begin
 AMem^:=$0D;                            //or eax
 Inc(AMem);
 PCardinal(AMem)^:=ANum;                //num32
 Inc(AMem,4);
end;

procedure XorEaxNum32(var AMem:PByte;ANum:Cardinal);
begin
 AMem^:=$35;                            //xor eax
 Inc(AMem);
 PCardinal(AMem)^:=ANum;                //num32
 Inc(AMem,4);
end;

procedure AddReg32Num32(var AMem:PByte;AReg:Byte;ANum:Cardinal);
begin
 AMem^:=$81;                            //add | or | and | sub | xor | cmp
 Inc(AMem);
 AMem^:=$C0+AReg;                       //reg32
 Inc(AMem);
 PCardinal(AMem)^:=ANum;                //num32
 Inc(AMem,4);
end;

procedure OrReg32Num32(var AMem:PByte;AReg:Byte;ANum:Cardinal);
begin
 AMem^:=$81;                            //add | or | and | sub | xor | cmp
 Inc(AMem);
 AMem^:=$C8+AReg;                       //reg32
 Inc(AMem);
 PCardinal(AMem)^:=ANum;                //num32
 Inc(AMem,4);
end;

procedure AndReg32Num32(var AMem:PByte;AReg:Byte;ANum:Cardinal);
begin
 AMem^:=$81;                            //add | or | and | sub | xor | cmp
 Inc(AMem);
 AMem^:=$E0+AReg;                       //reg32
 Inc(AMem);
 PCardinal(AMem)^:=ANum;                //num32
 Inc(AMem,4);
end;

procedure SubReg32Num32(var AMem:PByte;AReg:Byte;ANum:Cardinal);
begin
```

```
 AMem^:=$81;                            //add | or | and | sub | xor | cmp
 Inc(AMem);
 AMem^:=$E8+AReg;                       //reg32
 Inc(AMem);
 PCardinal(AMem)^:=ANum;                //num32
 Inc(AMem,4);
end;

procedure XorReg32Num32(var AMem:PByte;AReg:Byte;ANum:Cardinal);
begin
 AMem^:=$81;                            //add | or | and | sub | xor | cmp
 Inc(AMem);
 AMem^:=$F0+AReg;                       //reg32
 Inc(AMem);
 PCardinal(AMem)^:=ANum;                //num32
 Inc(AMem,4);
end;

procedure XorReg32Reg32(var AMem:PByte;AReg1,AReg2:Byte);
begin
 AMem^:=$31;                            //xor
 Inc(AMem);
 AMem^:=$C0+AReg2*8+AReg1;              //reg32,reg32
 Inc(AMem);
end;

function XorReg32RegMem(var AMem:PByte;AReg1,AReg2:Byte):Byte;
begin
 Result:=2;
 AMem^:=$33;                            //xor
 Inc(AMem);
 if AReg2=REG_EBP then
 begin
  Inc(Result);
  AMem^:=AReg1*8+$45;                   //reg32,ebp
  Inc(AMem);
  AMem^:=$00;                           //+0
 end else AMem^:=AReg1*8+AReg2;         //reg32,regmem
 Inc(AMem);
 if AReg2=REG_ESP then
 begin
  Inc(Result);
  AMem^:=$24;                           //esp
  Inc(AMem);
 end;
end;

function XorRegMemReg32(var AMem:PByte;AReg1,AReg2:Byte):Byte;
begin
 Result:=2;
 AMem^:=$31;                            //xor
 Inc(AMem);
 if AReg1=REG_EBP then
 begin
  Inc(Result);
  AMem^:=AReg2*8+$45;                   //reg32,ebp
  Inc(AMem);
  AMem^:=$00;                           //+0
 end else AMem^:=AReg2*8+AReg1;         //reg32,regmem
 Inc(AMem);
 if AReg1=REG_ESP then
 begin
  Inc(Result);
  AMem^:=$24;                           //esp
  Inc(AMem);
 end;
end;

procedure CmpReg32Num32(var AMem:PByte;AReg:Byte;ANum:Cardinal);
```

```
begin
 AMem^:=$81;                          //add | or | and | sub | xor | cmp
 Inc(AMem);
 AMem^:=$F8+AReg;                     //reg32
 Inc(AMem);
 PCardinal(AMem)^:=ANum;              //num32
 Inc(AMem,4);
end;

function TestReg32Num32(var AMem:PByte;AReg:Byte;ANum:Cardinal):Byte;
begin
 if AReg=REG_EAX then ThrowTheDice(Result,2)
 else Result:=2;
 Inc(Result,4);
 if Result=6 then
 begin
  AMem^:=$F7;                          //test
  Inc(AMem);
  AMem^:=$C0+AReg;                     //reg32
  Inc(AMem);
  PCardinal(AMem)^:=ANum;              //num32
  Inc(AMem,4);
 end else TestEaxNum32(AMem,ANum);
end;

procedure TestReg32Reg32(var AMem:PByte;AReg1,AReg2:Byte);
begin
 AMem^:=$85;                          //test
 Inc(AMem);
 AMem^:=AReg2*8+AReg1+$C0;            //reg32,reg32
 Inc(AMem);
end;

function TestRegMemNum32(var AMem:PByte;AReg:Byte;ANum:Cardinal):Byte;
begin
 Result:=6;
 AMem^:=$F7;                          //test
 Inc(AMem);
 if AReg=REG_EBP then
 begin
  Inc(Result);
  AMem^:=$45;                         //ebp
  Inc(AMem);
  AMem^:=$00;                         //+0
 end else AMem^:=AReg;                //reg32
 Inc(AMem);
 if AReg=REG_ESP then
 begin
  Inc(Result);
  AMem^:=$24;                         //esp
  Inc(AMem);
 end;
 PCardinal(AMem)^:=ANum;              //num32
 Inc(AMem,4);
end;


procedure AddReg32RandNum32(var AMem:PByte;ANum:Cardinal);
begin
 AddReg32Num32(AMem,RandomReg32All,ANum);
end;

procedure OrReg32RandNum32(var AMem:PByte;ANum:Cardinal);
begin
 OrReg32Num32(AMem,RandomReg32All,ANum);
end;

procedure AndReg32RandNum32(var AMem:PByte;ANum:Cardinal);
begin
```

```
 AndReg32Num32(AMem,RandomReg32All,ANum);
end;

procedure SubReg32RandNum32(var AMem:PByte;ANum:Cardinal);
begin
 SubReg32Num32(AMem,RandomReg32All,ANum);
end;

procedure XorReg32RandNum32(var AMem:PByte;ANum:Cardinal);
begin
 XorReg32Num32(AMem,RandomReg32All,ANum);
end;

procedure CmpReg32RandNum32Rand(var AMem:PByte);
begin
 CmpReg32Num32(AMem,RandomReg32All,Random($FFFFFFFF));
end;

procedure TestReg32RandNum32Rand6(var AMem:PByte);
var
 LLen:Byte;
begin
 LLen:=TestReg32Num32(AMem,RandomReg32All,Random($FFFFFFFF));
 if LLen=5 then
 begin
  AMem^:=$90;
  Inc(AMem);
 end;
end;

procedure MovReg32Num32Rand(var AMem:PByte;AReg:Byte);
begin
 MovReg32Num32(AMem,AReg,Random($FFFFFFFF));
end;

procedure MovReg16Num16(var AMem:PByte;AReg:Byte;ANum:Word);
begin
 AMem^:=$66;                          //mov
 Inc(AMem);
 AMem^:=$B8+AReg;                     //reg16
 Inc(AMem);
 PWord(AMem)^:=ANum;                  //num16
 Inc(AMem,2);
end;

procedure MovReg16Num16Rand(var AMem:PByte;AReg:Byte);
begin
 MovReg16Num16(AMem,AReg,Random($10000));
end;



procedure GenerateRubbishCode(AMem:Pointer;ASize,AVirtAddr:Cardinal); stdcall;
//генерує буфер інструкцій, нічого не роблять, та не потрібно забувати, що флаги
зазвичай змінюються тут й не використані nops

 procedure InsertRandomInstruction(var AMem:PByte;ALength:Byte;var
ARemaining:Cardinal);
 var
  LRegAny:Byte;
  LMaxDice,LXRem:Cardinal;
 begin
  case ALength of
   1:begin
    ThrowTheDice(LMaxDice,50);
{$IFDEF RUBBISH_NOPS}
    LMaxDice:=11;
{$ENDIF}
    case LMaxDice of
```

```
     001..010:Cld(AMem);
     011..020:Nop(AMem);
     021..030:Stc(AMem);
     031..040:Clc(AMem);
     041..050:Cmc(AMem);
    end;
   end;
   2:begin
    ThrowTheDice(LMaxDice,145);
    case LMaxDice of
     001..010:XchgReg32Rand(AMem);
     011..020:MovReg32Rand(AMem);
     021..030:begin LRegAny:=IncReg32Rand(AMem); DecReg32(AMem,LRegAny); end;
     031..040:begin LRegAny:=DecReg32Rand(AMem); IncReg32(AMem,LRegAny); end;
     041..050:begin LRegAny:=PushReg32Rand(AMem); PopReg32(AMem,LRegAny); end;
     051..060:LeaReg32Rand(AMem);
     061..070:TestReg32Rand(AMem);
     071..080:OrReg32Rand(AMem);
     081..090:AndReg32Rand(AMem);
     091..100:TestReg8Rand(AMem);
     101..110:OrReg8Rand(AMem);
     111..120:AndReg8Rand(AMem);
     121..130:CmpRegRegNum8Rand(AMem);
     131..132:begin Std(AMem); Cld(AMem); end;
     133..134:JmpNum8(AMem,0);
     135..138:SubAlNum8(AMem,0);
     139..140:TestAlNum8Rand(AMem);
     141..142:AddAlNum8(AMem,0);
     143..145:FNop(AMem);
    end;
   end;
   3:begin
    ThrowTheDice(LMaxDice,205);
    case LMaxDice of
     001..010:begin JmpNum8(AMem,1); InsertRandomInstruction(AMem,1,LXRem); end;
     011..020:SubReg32Num8Rand(AMem,0);
     021..030:AddReg32Num8Rand(AMem,0);
     031..040:begin LRegAny:=PushReg32Rand(AMem); IncReg32(AMem,LRegAny);
PopReg32(AMem,LRegAny); end;
     041..050:begin LRegAny:=PushReg32Rand(AMem); DecReg32(AMem,LRegAny);
PopReg32(AMem,LRegAny); end;
     051..060:CmpRandReg32RandNum8(AMem);
     061..070:TestReg8Num8Rand(AMem);
     071..080:SubReg8Num8Rand(AMem,0);
     081..090:AddReg8Num8Rand(AMem,0);
     091..100:AndReg16Rand(AMem);
     101..110:TestReg16Rand(AMem);
     111..120:OrReg16Rand(AMem);
     121..130:ShlReg32RandNum8FullRand(AMem);
     131..140:ShrReg32RandNum8FullRand(AMem);
     141..150:SalReg32RandNum8FullRand(AMem);
     151..160:SarReg32RandNum8FullRand(AMem);
     161..170:RolReg8RandNum8FullRand(AMem);
     171..180:RorReg8RandNum8FullRand(AMem);
     181..190:RolReg32RandNum8FullRand(AMem);
     191..200:RorReg32RandNum8FullRand(AMem);
     201..203:begin PushReg32(AMem,REG_EDX); Cdq(AMem); PopReg32(AMem,REG_EDX);
end;
     204..205:begin LRegAny:=PushReg32Rand(AMem);
InsertRandomInstruction(AMem,1,LXRem); PopReg32(AMem,LRegAny); end;
    end;
   end;
   4:begin
    ThrowTheDice(LMaxDice,170);
    case LMaxDice of
     001..020:begin JmpNum8(AMem,2); InsertRandomInstruction(AMem,2,LXRem); end;
     021..040:begin LRegAny:=PushReg32Rand(AMem);
InsertRandomInstruction(AMem,2,LXRem); PopReg32(AMem,LRegAny); end;
     041..050:TestAxNum16Rand(AMem);
```

```
     051..060:CmpAxNum16Rand(AMem);
     061..063:DoubleXorRand(AMem);
     064..066:DoubleNegRand(AMem);
     067..070:DoubleNotRand(AMem);
     071..080:AddReg16Num8Rand(AMem,0);
     081..090:OrReg16Num8Rand(AMem,0);
     091..100:AndReg16Num8Rand(AMem,$FF);
     101..110:SubReg16Num8Rand(AMem,0);
     111..120:XorReg16Num8Rand(AMem,0);
     121..130:CmpReg16Num8RandRand(AMem);
     131..140:RolReg16RandNum8FullRand(AMem);
     141..150:RorReg16RandNum8FullRand(AMem);
     151..155:DoubleXchgRand(AMem);
     156..160:begin LRegAny:=PushReg32Rand(AMem);
MovReg32Reg32Rand(AMem,LRegAny); PopReg32(AMem,LRegAny); end;
     161..170:begin PushReg32Rand(AMem); AddReg32Num8(AMem,REG_ESP,4); end;
    end;
   end;
   5:begin
    ThrowTheDice(LMaxDice,150);
    case LMaxDice of
     001..030:begin JmpNum8(AMem,3); InsertRandomInstruction(AMem,3,LXRem); end;
     031..060:begin LRegAny:=PushReg32Rand(AMem);
InsertRandomInstruction(AMem,3,LXRem); PopReg32(AMem,LRegAny); end;
     061..070:begin LRegAny:=PushReg32Rand(AMem); PushNum8Rand(AMem);
PopReg32(AMem,LRegAny); PopReg32(AMem,LRegAny); end;
     071..080:begin PushNum8Rand(AMem); AddReg32Num8(AMem,REG_ESP,4); end;
     081..090:AddEaxNum32(AMem,0);
     091..100:OrEaxNum32(AMem,0);
     101..110:AndEaxNum32(AMem,$FFFFFFFF);
     111..120:SubEaxNum32(AMem,0);
     121..130:XorEaxNum32(AMem,0);
     131..140:CmpEaxNum32Rand(AMem);
     141..150:TestEaxNum32Rand(AMem);
    end;
   end;
   6:begin
    ThrowTheDice(LMaxDice,161);
    case LMaxDice of
     001..040:begin JmpNum8(AMem,4); InsertRandomInstruction(AMem,4,LXRem); end;
     041..080:begin LRegAny:=PushReg32Rand(AMem);
InsertRandomInstruction(AMem,4,LXRem); PopReg32(AMem,LRegAny); end;
     081..090:AddReg32RandNum32(AMem,0);
     091..100:OrReg32RandNum32(AMem,0);
     101..110:AndReg32RandNum32(AMem,$FFFFFFFF);
     111..120:SubReg32RandNum32(AMem,0);
     121..130:XorReg32RandNum32(AMem,0);
     131..140:CmpReg32RandNum32Rand(AMem);
     141..150:TestReg32RandNum32Rand6(AMem);
     151..161:begin LRegAny:=PushReg32Rand(AMem);
MovReg16Num16Rand(AMem,LRegAny); PopReg32(AMem,LRegAny); end;
    end;
   end;
   7:begin
    ThrowTheDice(LMaxDice,110);
    case LMaxDice of
     001..050:begin JmpNum8(AMem,5); InsertRandomInstruction(AMem,5,LXRem); end;
     051..100:begin LRegAny:=PushReg32Rand(AMem);
InsertRandomInstruction(AMem,5,LXRem); PopReg32(AMem,LRegAny); end;
     101..110:begin LRegAny:=PushReg32Rand(AMem);
MovReg32Num32Rand(AMem,LRegAny); PopReg32(AMem,LRegAny); end;
    end;
   end;
   8:begin
    ThrowTheDice(LMaxDice,120);
    case LMaxDice of
     001..060:begin JmpNum8(AMem,6); InsertRandomInstruction(AMem,6,LXRem); end;
     061..120:begin LRegAny:=PushReg32Rand(AMem);
InsertRandomInstruction(AMem,6,LXRem); PopReg32(AMem,LRegAny); end;
```

```
    end;
   end;
   9..10:begin
    ThrowTheDice(LMaxDice,200);
    case LMaxDice of
     001..100:begin JmpNum8(AMem,ALength-2);
InsertRandomInstruction(AMem,ALength-2,LXRem); end;
     101..200:begin LRegAny:=PushReg32Rand(AMem);
InsertRandomInstruction(AMem,ALength-2,LXRem); PopReg32(AMem,LRegAny); end;
    end;
   end;
  end;
  if ALength<11 then Dec(ARemaining,ALength);
 end;

var
 LPB:PByte;
 LReg:Byte;
 LDice,LDecSize,LSize,LAddr:Cardinal;

begin
 LPB:=AMem;
 LSize:=ASize;

 while LSize>0 do
 begin
  ThrowTheDice(LDice,6);                //1-5 генерує одну маленьку інструкцію
                                        //6 генерує повно розмірні інструкції

  if LSize<32 then LDice:=1;            //для навеликого використання невеликих
інструкцій буферів
  if AVirtAddr=0 then LDice:=1;         //декілька додаткових інструкцій це
використовують

{$IFDEF RUBBISH_NOPS}
  LDice:=1;
{$ENDIF}
  if LDice<6 then                       //генерує повнорозмірні інструкції
  begin                                 //генерує малі інструкції
   ThrowTheDice(LDice,LSize*100);       //001..100 для однобайтних інструкцій
                                        //011..200 для двобайтних інструкцій

{$IFDEF RUBBISH_NOPS}
   LDice:=1;
{$ENDIF}
   if LSize=1 then LDice:=1;

   case LDice of
    001..002:InsertRandomInstruction(LPB,1,LSize);      //однобайтні інструкції
    101..104:InsertRandomInstruction(LPB,2,LSize);      //двобайтні інструкції
    201..208:InsertRandomInstruction(LPB,3,LSize);      //трибайтні інструкції
    301..316:InsertRandomInstruction(LPB,4,LSize);      //чотирьохбайтні
інструкції
    401..432:InsertRandomInstruction(LPB,5,LSize);      //п'ятибайтні інструкції
    501..564:InsertRandomInstruction(LPB,6,LSize);      //шостибайтні інструкції
    else InsertRandomInstruction(LPB,(LDice+99) div 100,LSize); //інструкції
більшої довжини
   end;
  end else
  begin
//   ThrowTheDice(LDice,100);
   ThrowTheDice(LDice,63);
//   if LDice<76 then LDecSize:=LSize
   if LDice<57 then LDecSize:=LSize
   else LDecSize:=0;
   case LDice of
    1..18:begin                         // використовує rel jump
     RelJmpAddr32(LPB,LSize-5);           //5 jump
     PutRandomBuffer(LPB,LSize-5);
```

```
    end;
    19..37:begin                          //використовує rel call
     LReg:=PushReg32Rand(LPB);
     ThrowTheDice(LDice);
     if LDice>3 then LAddr:=LSize-8       //1 push, 5 call, 1 pop, 1 pop
     else LAddr:=LSize-10;                //1 push, 5 call, 3 add, 1 pop

     RelCallAddr(LPB,LAddr);
     PutRandomBuffer(LPB,LAddr);
     if LDice>3 then PopReg32(LPB,LReg)
     else AddReg32Num8(LPB,REG_ESP,4);
     PopReg32(LPB,LReg);
    end;
(*
 цей код не може бути використаний для dll, так як нам потрібно переходи для
цього
    38..56:begin                          // використовує reg jmp
     LReg:=PushReg32Rand(LPB);
     ThrowTheDice(LDice);
     LAddr:=AVirtAddr+ASize-1;            //1 pop
                                          // використовує ASize cuz of not rel jmp
     if LDice>3 then
     begin
      MovReg32Num32(LPB,LReg,LAddr);
      LAddr:=LSize-9;                     //1 push, 5 mov, 2 jmp, 1 pop
     end else
     begin
      PushNum32(LPB,LAddr);
      PopReg32(LPB,LReg);
      LAddr:=LSize-10;                    //1 push, 5 push, 1 pop, 2 jmp, 1 pop
     end;
     JmpReg32(LPB,LReg);
     PutRandomBuffer(LPB,LAddr);
     PopReg32(LPB,LReg);
    end;

    57..75:begin                          // використовує reg call
     LReg:=PushReg32Rand(LPB);
     ThrowTheDice(LDice,8);               //1,2 - push,mov,call,pop,pop
                                          //3,4 - push,mov,call,add,pop
                                          //5,6 - push,push,pop,call,pop,pop
                                          //7,8 - push,push,pop,call,add,pop

     case LDice of
      1,2,5,6:LAddr:=AVirtAddr+ASize-2; //1 pop, 1 pop
      else LAddr:=AVirtAddr+ASize-4;    //1 pop, 3 add
     end;

     if LDice<5 then
     begin
      MovReg32Num32(LPB,LReg,LAddr);
      if LDice<3 then LAddr:=LSize-10   //1 push, 5 mov, 2 call, 1 pop, 1 pop
      else LAddr:=LSize-12;             //1 push, 5 mov, 2 call, 3 add, 1 pop
     end else
     begin
      PushNum32(LPB,LAddr);
      PopReg32(LPB,LReg);
      if LDice<7 then LAddr:=LSize-11   //1 push, 5 push, 1 pop, 2 call, 1 pop,
1 pop
      else LAddr:=LSize-13;             //1 push, 5 push, 1 pop, 2 call, 3 add,
1 pop
     end;
     CallReg32(LPB,LReg);
     PutRandomBuffer(LPB,LAddr);
     case LDice of
      1,2,5,6:PopReg32(LPB,LReg);
      else AddReg32Num8(LPB,REG_ESP,4);
     end;
     PopReg32(LPB,LReg);
```

```
    end;
    *)

//    76..94:begin                          // використовує loop + jeczx
     38..56:begin                          // використовує loop + jeczx
     if LSize-3<$7D then LAddr:=LSize-4
     else LAddr:=$7C;
     LAddr:=Random(LAddr)+2;
     LoopNum8(LPB,LAddr);
     JecxzNum8(LPB,LAddr-2);
     PutRandomBuffer(LPB,LAddr-2);
     IncReg32(LPB,REG_ECX);
     LDecSize:=LAddr+3;
    end;
    //95..100:begin                         // використовує back loop
    57..63:begin                          // використовує back loop
     if LSize-7<$7D then LAddr:=LSize-7
     else LAddr:=$75;
     LAddr:=Random(LAddr)+3;
     PushReg32(LPB,REG_ECX);
     MovzxEcxCl(LPB);                                    //не є чеканням, якщо
ecx = 0
     GenerateRubbishCode(LPB,LAddr-3,0);
     Inc(LPB,LAddr-3);
     LoopNum8(LPB,$FE-LAddr);
     PopReg32(LPB,REG_ECX);

     LDecSize:=LAddr+4;
    end;
   end;
   Dec(LSize,LDecSize);
  end;
 end;
end;

procedure
GenerateInitCode(ACodePtr,AKeyPtr,AData1Ptr,ASize1,AData2Ptr,ASize2,ADynLoadAddr
,
                                 AMainPtr,AEntryPointAddr,AImpThunk:Cardinal);
//Це - полідешифратор. Завантажувач бачучи кінець цієї функції, не забуває
додавати фіксуючі вказівники для деяких інструкцій. що дозволяє додавати більше
варіантів для кожної інструкції

type
 TPolyContext=record
  DataSizeRegister:Byte;
  DataAddrRegister:Byte;
  EipRegister:Byte;
  KeyAddrRegister:Byte;
  KeyBytesRegister:Byte;
  FreeRegisters:array[0..1] of Byte;
 end;

var
 LInitInstr:array[0..InitInstrCount-1] of TVarInstruction;
 LI:Integer;
 LVirtAddr,LRubbishSize,LDelta,LDelta2,LRemaining,LCodeStart,LEIPSub:Cardinal;
 LPB:PByte;
 PolyContext:TPolyContext;
{$IFNDEF STATIC_CONTEXT}
 LRegUsed:array[0..Reg32Count-1] of Boolean;
 LNotUsed:Integer;
 LReg:Byte;
{$ENDIF}

 function InstructionAddress(AInstruction:Cardinal):PByte;
 //повертає точку виклику інструкції
 begin
```

```
  Result:=Pointer(Cardinal(InitData)+LInitInstr[AInstruction].VirtualAddress-
LCodeStart);
 end;

 function CallAddress(AFromInstruction,AToInstruction:Cardinal):Cardinal;
 //повертає відносну дельту між двоиа інструкціями для виклику
 begin
  Result:=LInitInstr[AToInstruction].VirtualAddress-
(LInitInstr[AFromInstruction].VirtualAddress+5);
 end;

 function JcxAddress(AFromInstruction,AToInstruction:Cardinal):Cardinal;
 //повертає відносну дельту між двоиа інструкціями для умовного переходу
 begin
  Result:=LInitInstr[AToInstruction].VirtualAddress-
(LInitInstr[AFromInstruction].VirtualAddress+6);
 end;

 function InsVAddr(AInstr:Cardinal):Cardinal;
 begin
  Result:=LInitInstr[AInstr].VirtualAddress;
 end;

 function InsFix1(AInstr:Cardinal):Byte;
 begin
  Result:=LInitInstr[AInstr].Vars[LInitInstr[AInstr].Index].Fix1;
 end;

 function InsFix2(AInstr:Cardinal):Byte;
 begin
  Result:=LInitInstr[AInstr].Vars[LInitInstr[AInstr].Index].Fix2;
 end;

 function InsFix3(AInstr:Cardinal):Byte;
 begin
  Result:=LInitInstr[AInstr].Vars[LInitInstr[AInstr].Index].Fix3;
 end;

 function InsFix4(AInstr:Cardinal):Byte;
 begin
  Result:=LInitInstr[AInstr].Vars[LInitInstr[AInstr].Index].Fix4;
 end;

 procedure FixInstr(AInstr:Cardinal;AFix1:Cardinal;AFix2:Cardinal=Cardinal(-1));
 begin
  if InsFix1(AInstr)<>Byte(-1) then
  begin
   LPB:=InstructionAddress(AInstr);
   Inc(LPB,InsFix1(AInstr));
   PCardinal(LPB)^:=AFix1;
  end;
  if (InsFix2(AInstr)<>Byte(-1)) and (AFix2<>Cardinal(-1)) then
  begin
   LPB:=InstructionAddress(AInstr);
   Inc(LPB,InsFix2(AInstr));
   PCardinal(LPB)^:=AFix2;
  end;
 end;

 procedure GeneratePolyInstruction(AInstruction,ARegister:Byte);
 var
  LPB:PByte;
  LReg,LFreeReg,LFreeRegOther,LAnyReg:Byte;

   function CtxFreeReg:Byte;
   var
    LIdx:Byte;
   begin
    LIdx:=Random(10) mod 2;
```

```
   LFreeReg:=PolyContext.FreeRegisters[LIdx];
   LFreeRegOther:=PolyContext.FreeRegisters[(LIdx+1) mod 2];
   Result:=LFreeReg;
  end;

  function CtxAnyRegEsp:Byte;
  begin
   LAnyReg:=RandomReg32Esp;
   Result:=LAnyReg;
  end;

begin
 case AInstruction of
  PII_POLY_MOV_REG_LOADER_SIZE,PII_POLY_MOV_REG_LOADER_ADDR,
  PII_CODER_MOV_REG_KEY:
   with LInitInstr[AInstruction] do
   begin
    Count:=4;
    Vars[0].Len:=5;
    Vars[0].Fix1:=1;
    LPB:=@Vars[0].Code;
    MovReg32Num32(LPB,ARegister,$12345678);

    Vars[1].Len:=6;
    Vars[1].Fix1:=1;
    LPB:=@Vars[1].Code;
    PushNum32(LPB,$12345678);
    PopReg32(LPB,ARegister);

    Vars[2].Len:=5;
    Vars[2].Fix1:=1;
    LPB:=@Vars[2].Code;
    MovReg32Num32(LPB,CtxFreeReg,$12345678);
    Inc(Vars[2].Len,XchgReg32Reg32(LPB,LFreeReg,ARegister));

    Vars[3].Len:=6;
    Vars[3].Fix1:=2;
    LPB:=@Vars[3].Code[0];
    LeaReg32Addr32(LPB,ARegister,$12345678);
   end;

  PII_POLY_JMP_DYNLOADER:
   with LInitInstr[AInstruction] do
   begin
    Count:=3;
    Vars[0].Len:=5;
    Vars[0].Fix1:=1;
    Vars[0].Fix2:=0;
    LPB:=@Vars[0].Code;
    RelJmpAddr32(LPB,$12345678);

    Vars[1].Len:=8;
    Vars[1].Fix1:=4;
    Vars[1].Fix2:=3;
    LPB:=@Vars[1].Code;
    LReg:=RandomReg32Esp;
    XorReg32Reg32(LPB,LReg,LReg);
    RelJzAddr32(LPB,$12345678);

    Vars[2].Len:=7;
    Vars[2].Fix1:=3;
    Vars[2].Fix2:=2;
    LPB:=@Vars[2].Code;
    DecReg32(LPB,PolyContext.EipRegister);
    RelJnzAddr32(LPB,$12345678);
   end;

  PII_CODER_CALL_GET_EIP:
   with LInitInstr[AInstruction] do
```

```
 begin
  Count:=4;
  Vars[0].Len:=5;
  Vars[0].Fix1:=1;
  Vars[0].Fix2:=0;
  Vars[0].Fix3:=5;
  LPB:=@Vars[0].Code;
  RelCallAddr(LPB,$12345678);

  Vars[1].Len:=12;
  Vars[1].Fix1:=3;
  Vars[1].Fix2:=2;
  Vars[1].Fix3:=12;
  LPB:=@Vars[1].Code;
  RelJmpAddr8(LPB,5);
  RelJmpAddr32(LPB,$12345678);
  RelCallAddr(LPB,Cardinal(-10));

  Vars[2].Len:=5;
  Vars[2].Fix1:=Byte(-1);
  Vars[2].Fix2:=Byte(-1);
  Vars[2].Fix3:=5;
  LPB:=@Vars[2].Code;
  RelCallAddr(LPB,0);

  Vars[3].Len:=9;
  Vars[3].Fix1:=Byte(-1);
  Vars[3].Fix2:=Byte(-1);
  Vars[3].Fix3:=9;
  LPB:=@Vars[3].Code;
  RelJmpAddr8(LPB,2);
  RelJmpAddr8(LPB,5);
  RelCallAddr(LPB,Cardinal(-7));
 end;

PII_CODER_GET_EIP:
 with LInitInstr[AInstruction] do
 begin
  Count:=4;
  Vars[0].Len:=1;
  LPB:=@Vars[0].Code;
  PopReg32(LPB,ARegister);

  Vars[1].Len:=3;
  LPB:=@Vars[1].Code;
  Inc(Vars[1].Len,MovReg32RegMem(LPB,ARegister,REG_ESP));
  AddReg32Num8(LPB,REG_ESP,4);

  Vars[2].Len:=3;
  LPB:=@Vars[2].Code;
  AddReg32Num8(LPB,REG_ESP,4);
  Inc(Vars[2].Len,MovReg32RegMemIdx8(LPB,ARegister,REG_ESP,Byte(-4)));

  Vars[3].Len:=4;
  LPB:=@Vars[3].Code;
  Inc(Vars[3].Len,MovReg32RegMem(LPB,ARegister,REG_ESP));
  IncReg32(LPB,REG_ESP);
  IncReg32(LPB,REG_ESP);
  IncReg32(LPB,REG_ESP);
  IncReg32(LPB,REG_ESP);
 end;

PII_CODER_FIX_DST_PTR,PII_CODER_FIX_SRC_PTR:
 with LInitInstr[AInstruction] do
 begin
  Count:=4;
  Vars[0].Len:=2;
  LPB:=@Vars[0].Code;
  AddReg32Reg32(LPB,ARegister,PolyContext.EipRegister);
```

```
  Vars[1].Len:=6;
  LPB:=@Vars[1].Code;
  PushReg32(LPB,PolyContext.EipRegister);
  AddReg32Reg32(LPB,PolyContext.EipRegister,ARegister);
  MovReg32Reg32(LPB,ARegister,PolyContext.EipRegister);
  PopReg32(LPB,PolyContext.EipRegister);

  Vars[2].Len:=6;
  LPB:=@Vars[2].Code;
  PushReg32(LPB,PolyContext.EipRegister);
  AddReg32Reg32(LPB,PolyContext.EipRegister,ARegister);
  PushReg32(LPB,PolyContext.EipRegister);
  PopReg32(LPB,ARegister);
  PopReg32(LPB,PolyContext.EipRegister);

  Vars[3].Len:=2;
  LPB:=@Vars[3].Code;
  PushReg32(LPB,PolyContext.EipRegister);
  Inc(Vars[3].Len,AddReg32RegMem(LPB,ARegister,REG_ESP));
  PopReg32(LPB,PolyContext.EipRegister);
 end;

PII_CODER_LOAD_KEY_TO_REG:
 with LInitInstr[AInstruction] do
 begin
  Count:=4;
  LPB:=@Vars[0].Code;
  Vars[0].Len:=MovReg32RegMem(LPB,ARegister,PolyContext.KeyAddrRegister);

  Vars[1].Len:=1;
  LPB:=@Vars[1].Code;
  Inc(Vars[1].Len,PushRegMem(LPB,PolyContext.KeyAddrRegister));
  PopReg32(LPB,ARegister);

  LPB:=@Vars[2].Code;
  Vars[2].Len:=LeaReg32Reg32(LPB,ARegister,PolyContext.KeyAddrRegister);
  Inc(Vars[2].Len,MovReg32RegMem(LPB,ARegister,ARegister));

  Vars[3].Len:=2;
  LPB:=@Vars[3].Code;
  XorReg32Reg32(LPB,ARegister,ARegister);
  Inc(Vars[3].Len,AddReg32RegMem(LPB,ARegister,PolyContext.KeyAddrRegister));
 end;

PII_CODER_TEST_KEY_END:
 with LInitInstr[AInstruction] do
 begin
  Count:=4;
  LPB:=@Vars[0].Code;
  Vars[0].Len:=TestReg32Num32(LPB,ARegister,$FF000000);

  LPB:=@Vars[1].Code;
  Vars[1].Len:=TestRegMemNum32(LPB,PolyContext.KeyAddrRegister,$FF000000);

  LPB:=@Vars[2].Code;
  Vars[2].Len:=7;
  MovReg32Reg32(LPB,CtxFreeReg,ARegister);
  ShrReg32Num8(LPB,LFreeReg,$18);
  TestReg32Reg32(LPB,LFreeReg,LFreeReg);

  LPB:=@Vars[3].Code;
  Vars[3].Len:=11;
  PushReg32(LPB,ARegister);
  PopReg32(LPB,CtxFreeReg);
  AndReg32Num32(LPB,LFreeReg,$FF000000);
  CmpReg32Num8(LPB,LFreeReg,0);
 end;
```

```
PII_CODER_JZ_CODER_BEGIN:
 with LInitInstr[AInstruction] do
 begin
  Count:=2;
  LPB:=@Vars[0].Code;
  Vars[0].Len:=6;
  Vars[0].Fix1:=2;
  Vars[0].Fix2:=0;
  RelJzAddr32(LPB,$12345678);

  LPB:=@Vars[1].Code;
  Vars[1].Len:=7;
  Vars[1].Fix1:=3;
  Vars[1].Fix2:=1;
  RelJnzAddr8(LPB,5);
  RelJmpAddr32(LPB,$12345678);
 end;

PII_CODER_ADD_DATA_IDX:
 with LInitInstr[AInstruction] do
 begin
  Count:=4;
  LPB:=@Vars[0].Code;
  Vars[0].Len:=2;
  AddReg32Reg32(LPB,ARegister,PolyContext.DataSizeRegister);

  LPB:=@Vars[1].Code;
  Vars[1].Len:=2;
  PushReg32(LPB,PolyContext.DataSizeRegister);
  Inc(Vars[1].Len,AddReg32RegMem(LPB,ARegister,REG_ESP));
  PopReg32(LPB,PolyContext.DataSizeRegister);

  LPB:=@Vars[2].Code;
  Vars[2].Len:=6;
  PushReg32(LPB,CtxFreeReg);
  MovReg32Reg32(LPB,LFreeReg,PolyContext.DataSizeRegister);
  AddReg32Reg32(LPB,Aregister,LFreeReg);
  PopReg32(LPB,LFreeReg);

  LPB:=@Vars[3].Code;
  Vars[3].Len:=2;
  PushReg32(LPB,ARegister);
  Inc(Vars[3].Len,AddRegMemReg32(LPB,REG_ESP,PolyContext.DataSizeRegister));
  PopReg32(LPB,ARegister);
 end;

PII_CODER_XOR_DATA_REG:
 with LInitInstr[AInstruction] do
 begin
  Count:=4;
  LPB:=@Vars[0].Code;
  Vars[0].Len:=XorReg32RegMem(LPB,ARegister,PolyContext.DataAddrRegister);

  LPB:=@Vars[1].Code;
  Vars[1].Len:=3;
  PushReg32(LPB,CtxFreeReg);
  Inc(Vars[1].Len,PushRegMem(LPB,PolyContext.DataAddrRegister));
  Inc(Vars[1].Len,XorReg32RegMem(LPB,ARegister,REG_ESP));
  PopReg32(LPB,LFreeReg);
  PopReg32(LPB,LFreeReg);

  LPB:=@Vars[2].Code;
  Vars[2].Len:=4;
  PushReg32(LPB,CtxFreeReg);
  Inc(Vars[2].Len,MovReg32RegMem(LPB,LFreeReg,PolyContext.DataAddrRegister));
  XorReg32Reg32(LPB,ARegister,LFreeReg);
  PopReg32(LPB,LFreeReg);

  LPB:=@Vars[3].Code;
```

```
     Vars[3].Len:=4;
     PushReg32(LPB,CtxFreeReg);
     Inc(Vars[3].Len,MovReg32RegMem(LPB,LFreeReg,PolyContext.DataAddrRegister));
     PushReg32(LPB,LFreeReg);
     Inc(Vars[3].Len,XorRegMemReg32(LPB,REG_ESP,ARegister));
     PopReg32(LPB,ARegister);
     PopReg32(LPB,LFreeReg);
    end;

   PII_CODER_STORE_DATA:
    with LInitInstr[AInstruction] do
    begin
     Count:=4;
     LPB:=@Vars[0].Code;
     Vars[0].Len:=1;

     if (PolyContext.DataAddrRegister<>REG_EDI) or
(PolyContext.KeyBytesRegister<>REG_EAX) then
     begin
      if (PolyContext.DataAddrRegister<>REG_EDI) then Inc(Vars[0].Len,6);
      if (PolyContext.DataAddrRegister<>REG_EDI) then PushReg32(LPB,REG_EDI);

      if (PolyContext.DataAddrRegister<>REG_EAX) then Inc(Vars[0].Len,2);
      if (PolyContext.DataAddrRegister<>REG_EAX) then PushReg32(LPB,REG_EAX);

      if (PolyContext.DataAddrRegister<>REG_EDI) then
PushReg32(LPB,PolyContext.DataAddrRegister);
      PushReg32(LPB,PolyContext.KeyBytesRegister);
      PopReg32(LPB,REG_EAX);
      if (PolyContext.DataAddrRegister<>REG_EDI) then PopReg32(LPB,REG_EDI);
      Inc(Vars[0].Len,4);
     end;
     Stosd(LPB);
     if (PolyContext.DataAddrRegister<>REG_EDI) or
(PolyContext.KeyBytesRegister<>REG_EAX) then
     begin
      PushReg32(LPB,REG_EAX);
      if (PolyContext.DataAddrRegister<>REG_EDI) then PushReg32(LPB,REG_EDI);
      if (PolyContext.DataAddrRegister<>REG_EDI) then
PopReg32(LPB,PolyContext.DataAddrRegister);
      PopReg32(LPB,PolyContext.KeyBytesRegister);
      if (PolyContext.DataAddrRegister<>REG_EAX) then PopReg32(LPB,REG_EAX);
      if (PolyContext.DataAddrRegister<>REG_EDI) then PopReg32(LPB,REG_EDI);
     end;

     LPB:=@Vars[1].Code;
     Vars[1].Len:=4;

Inc(Vars[1].Len,MovRegMemReg32(LPB,PolyContext.DataAddrRegister,ARegister));
     IncReg32(LPB,PolyContext.DataAddrRegister);
     IncReg32(LPB,PolyContext.DataAddrRegister);
     IncReg32(LPB,PolyContext.DataAddrRegister);
     IncReg32(LPB,PolyContext.DataAddrRegister);

     LPB:=@Vars[2].Code;
     Vars[2].Len:=5;
     PushReg32(LPB,CtxFreeReg);
     Inc(Vars[2].Len,XchgReg32Reg32(LPB,REG_ESP,PolyContext.DataAddrRegister));
     PopReg32(LPB,LFreeReg);
     PushReg32(LPB,ARegister);
     PopReg32(LPB,LFreeReg);
     Inc(Vars[2].Len,XchgReg32Reg32(LPB,PolyContext.DataAddrRegister,REG_ESP));
     PopReg32(LPB,LFreeReg);

     LPB:=@Vars[3].Code;
     Vars[3].Len:=2;

     if ARegister=REG_EDI then
     begin
```

```
      MovReg32Reg32(LPB,CtxFreeReg,REG_EDI);
      Inc(Vars[3].Len,2);
     end;

     if PolyContext.DataAddrRegister<>REG_EDI then
     begin
      PushReg32(LPB,REG_EDI);
      MovReg32Reg32(LPB,REG_EDI,PolyContext.DataAddrRegister);
      Inc(Vars[3].Len,6);
     end;
     if ARegister=REG_EDI then PushReg32(LPB,LFreeReg)
     else PushReg32(LPB,ARegister);
     Inc(Vars[3].Len,XchgReg32Reg32(LPB,REG_ESI,REG_ESP));
     Movsd(LPB);
     Inc(Vars[3].Len,XchgReg32Reg32(LPB,REG_ESP,REG_ESI));
     if PolyContext.DataAddrRegister<>REG_EDI then
     begin
      MovReg32Reg32(LPB,PolyContext.DataAddrRegister,REG_EDI);
      PopReg32(LPB,REG_EDI);
     end;
    end;

   PII_CODER_INC_SRC_PTR:
    with LInitInstr[AInstruction] do
    begin
     Count:=4;
     LPB:=@Vars[0].Code;
     Vars[0].Len:=1;
     IncReg32(LPB,ARegister);

     LPB:=@Vars[1].Code;
     Vars[1].Len:=3;
     AddReg32Num8(LPB,ARegister,1);

     LPB:=@Vars[2].Code;
     Vars[2].Len:=3;
     SubReg32Num8(LPB,ARegister,Byte(-1));

     LPB:=@Vars[3].Code;
     Vars[3].Len:=7;
     PushReg32(LPB,CtxFreeReg);
     PushNum8(LPB,1);
     PopReg32(LPB,LFreeReg);
     AddReg32Reg32(LPB,ARegister,LFreeReg);
     PopReg32(LPB,LFreeReg);
    end;

   PII_CODER_LOOP_CODER_CODE:
    with LInitInstr[AInstruction] do
    begin
     Count:=1;
     LPB:=@Vars[0].Code;
     Vars[0].Len:=7;
     Vars[0].Fix1:=3;
     Vars[0].Fix2:=1;
     DecReg32(LPB,ARegister);
     RelJnzAddr32(LPB,$12345678);
    end;

  end;
 end;

begin
 ASize1:=ASize1 shr 2;
// ASize2:=ASize2 shr 2;

 ZeroMemory(@LInitInstr,SizeOf(LInitInstr));

 //генеруємо випадковий контекст
```

```
 with PolyContext do
 begin
{$IFNDEF STATIC_CONTEXT}
  DataSizeRegister:=REG_NON;
  DataAddrRegister:=REG_NON;
  EipRegister:=REG_NON;
  KeyAddrRegister:=REG_NON;
  KeyBytesRegister:=REG_NON;
  FreeRegisters[0]:=REG_NON;
  FreeRegisters[1]:=REG_NON;
{$ELSE}
//  DataSizeRegister:=REG_ESI;
//  DataAddrRegister:=REG_EBP;
//  EipRegister:=REG_ECX;
//  KeyAddrRegister:=REG_EAX;
//  KeyBytesRegister:=REG_EBX;
//  FreeRegisters[0]:=REG_EDI;
//  FreeRegisters[1]:=REG_EDX;
  DataSizeRegister:=REG_EAX;
  DataAddrRegister:=REG_EBX;
  EipRegister:=REG_ECX;
  KeyAddrRegister:=REG_EDX;
  KeyBytesRegister:=REG_ESI;
  FreeRegisters[0]:=REG_EDI;
  FreeRegisters[1]:=REG_EBP;
{$ENDIF}
 end;

{$IFNDEF STATIC_CONTEXT}
 for LI:=0 to Reg32Count-1 do LRegUsed[LI]:=False;
 LNotUsed:=Reg32Count-1;
 while LNotUsed>0 do
 begin
  LReg:=Random(Reg32Count);
  while LRegUsed[LReg] or (LReg=REG_ESP) do LReg:=(LReg+1) mod Reg32Count;
  LRegUsed[LReg]:=True;

  with PolyContext do
  case LNotUsed of
   1:DataSizeRegister:=LReg;
   2:DataAddrRegister:=LReg;
   3:EipRegister:=LReg;
   4:KeyAddrRegister:=LReg;
   5:KeyBytesRegister:=LReg;
   6:FreeRegisters[0]:=LReg;
   7:FreeRegisters[1]:=LReg;
  end;
  Dec(LNotUsed);
 end;
{$ENDIF}

// ці рядки добрі для відлагодження
// PolyContext.DataSizeRegister:=REG_ESI;
// PolyContext.DataAddrRegister:=REG_EBX;
// PolyContext.EipRegister:=REG_EDX;
// PolyContext.KeyAddrRegister:=REG_EBP;
// PolyContext.KeyBytesRegister:=REG_EAX;
// PolyContext.FreeRegisters[0]:=REG_ECX;
// PolyContext.FreeRegisters[1]:=REG_EDI;


GeneratePolyInstruction(PII_POLY_MOV_REG_LOADER_SIZE,PolyContext.DataSizeRegiste
r);

GeneratePolyInstruction(PII_POLY_MOV_REG_LOADER_ADDR,PolyContext.DataAddrRegiste
r);

 GeneratePolyInstruction(PII_CODER_CALL_GET_EIP,REG_NON);
 GeneratePolyInstruction(PII_CODER_GET_EIP,PolyContext.EipRegister);
```

```
 GeneratePolyInstruction(PII_CODER_FIX_DST_PTR,PolyContext.DataAddrRegister);
 GeneratePolyInstruction(PII_CODER_MOV_REG_KEY,PolyContext.KeyAddrRegister);
 GeneratePolyInstruction(PII_CODER_FIX_SRC_PTR,PolyContext.KeyAddrRegister);

GeneratePolyInstruction(PII_CODER_LOAD_KEY_TO_REG,PolyContext.KeyBytesRegister);
 GeneratePolyInstruction(PII_CODER_TEST_KEY_END,PolyContext.KeyBytesRegister);
 GeneratePolyInstruction(PII_CODER_JZ_CODER_BEGIN,REG_NON);
 GeneratePolyInstruction(PII_CODER_ADD_DATA_IDX,PolyContext.KeyBytesRegister);
 GeneratePolyInstruction(PII_CODER_XOR_DATA_REG,PolyContext.KeyBytesRegister);
 GeneratePolyInstruction(PII_CODER_STORE_DATA,PolyContext.KeyBytesRegister);
 GeneratePolyInstruction(PII_CODER_INC_SRC_PTR,PolyContext.KeyAddrRegister);

GeneratePolyInstruction(PII_CODER_LOOP_CODER_CODE,PolyContext.DataSizeRegister);
 GeneratePolyInstruction(PII_POLY_JMP_DYNLOADER,REG_NON);

 //
 //вписуємо деякий текст який нічого не означає, вибираємо інструкцію, й до неї
його дописуємо, після цього вибираємо наступну інструкцію. й до неї дописуємо
також деяку нічого не значущу інформацію, й так далі…
 //
 //але повинні ураховувати, що такі інструкції, які нічого не виконують
неможливо вписувати між PII_CODER_TEST_KEY_END  та PII_CODER_JZ_CODER_BEGIN
 //

 ZeroMemory(InitData,InitSize);
 LRemaining:=InitSize;

 LPB:=InitData;

LCodeStart:=NtHeaders.OptionalHeader.ImageBase+NtHeaders.OptionalHeader.AddressO
fEntryPoint;
 LVirtAddr:=LCodeStart;

 for LI:=0 to InitInstrCount-1 do
 with LInitInstr[LI] do
 begin
  LDelta:=InitInstrCount-LI;
  LDelta2:=LRemaining-LDelta*10;
  LRubbishSize:=Random(LDelta2 div LDelta);
  if (LI<>PII_CODER_JZ_CODER_BEGIN) and (LRubbishSize>0) then           //не
змінювати флаги після тестування
  begin
   GenerateRubbishCode(LPB,LRubbishSize,LVirtAddr);
   Inc(LPB,LRubbishSize);
   Inc(LVirtAddr,LRubbishSize);
   Dec(LRemaining,LRubbishSize);
  end;

  VirtualAddress:=LVirtAddr;
  Index:=Random(LInitInstr[LI].Count);
  with Vars[Index] do
  begin
   CopyMemory(LPB,@Code,Len);
   Inc(LPB,Len);
   Inc(LVirtAddr,Len);
   Dec(LRemaining,Len);
  end;
 end;

 LRubbishSize:=Random(LRemaining);
 GenerateRubbishCode(LPB,LRubbishSize,LVirtAddr);
 Dec(LRemaining,LRubbishSize);
 Inc(LPB,LRubbishSize);
 LRubbishSize:=LRemaining;
 GenerateRandomBuffer(LPB,LRubbishSize);


 //
 //тепер скоректуємо вказівники
```

```
 //

 //викликаємо та відновлюємо для отримання eip
 //але потрібно тільки базовий образ, так як потрібно вирахувати rva цього
виклику
 LEIPSub:=InsVAddr(PII_CODER_CALL_GET_EIP)-
ACodePtr+InsFix3(PII_CODER_CALL_GET_EIP);

 FixInstr(PII_POLY_MOV_REG_LOADER_SIZE,ASize1);
 FixInstr(PII_POLY_MOV_REG_LOADER_Addr,AData1Ptr-LEIPSub);

 FixInstr(PII_CODER_MOV_REG_KEY,AKeyPtr-LEIPSub);

FixInstr(PII_CODER_CALL_GET_EIP,CallAddress(PII_CODER_CALL_GET_EIP,PII_CODER_GET
_EIP)-InsFix2(PII_CODER_CALL_GET_EIP));

FixInstr(PII_CODER_JZ_CODER_BEGIN,JcxAddress(PII_CODER_JZ_CODER_BEGIN,PII_CODER_
KEY_START)-InsFix2(PII_CODER_JZ_CODER_BEGIN));

FixInstr(PII_CODER_LOOP_CODER_CODE,JcxAddress(PII_CODER_LOOP_CODER_CODE,PII_CODE
R_CODE)-InsFix2(PII_CODER_LOOP_CODER_CODE));

 FixInstr(PII_POLY_JMP_DYNLOADER,ADynLoadAddr-
(InsVAddr(PII_POLY_JMP_DYNLOADER)+5)-InsFix2(PII_POLY_JMP_DYNLOADER));


//
//повідомлення про кінець роботи
//
//

//                                                  // PII_BEGIN

//                                                  // PII_POLY_BEGIN
// mov ecx,0WWXXYYZZh              //редагування розміру        //
PII_POLY_MOV_REG_LOADER_SIZE
// mov edi,0WWXXYYZZh              //редагування адреси         //
PII_POLY_MOV_REG_LOADER_ADDR

//                                                  // PII_CODER_BEGIN
// виклик PII_CODER_GET_EIP                           //
PII_CODER_CALL_GET_EIP
// pop edx                                          // PII_CODER_GET_EIP
// add edi,edx                                      //
PII_CODER_FIX_DST_PTR
//                                                  // PII_CODER_KEY_START
// mov esi,0WWXXYYZZh              //адреса ключа           //
PII_CODER_MOV_REG_KEY
// add esi,edx                                      //
PII_CODER_FIX_SRC_PTR
//                                                  // PII_CODER_CODE
// mov eax,[esi]                   //редагування байт ключа     //
PII_CODER_LOAD_KEY_TO_REG
// test eax,0FF000000h             //тестування кінця ключа    //
PII_CODER_TEST_KEY_END
// jz PII_CODER_KEY_START          //перезавантаження  ключа        //
PII_CODER_JZ_CODER_BEGIN
// add eax,ecx                     //додавання деякого непотрібного коду
// PII_CODER_ADD_DATA_IDX
// xor [edi],eax                   //декодування             //
PII_CODER_XOR_DATA_REG
// stosd                           //зберігання даних         //
PII_CODER_STORE_DATA
// inc esi                         //зміна ключа         //
PII_CODER_INC_SRC_PTR
// loop PII_CODER_CODE                              //
PII_CODER_LOOP_CODER_CODE
//                                                  // PII_CODER_END
```

```
// jmp @DynLoader_begin                                    //
PII_POLY_JMP_DYNLOADER
//                                                         // PII_POLY_END

//                                                         // PII_END

end;

function ExtractFileName(APath:string):string;
//повертаємо ім'я файлу з повним шляхом
var
 LI,LJ:Integer;
begin
 if Length(APath)<>0 then
 begin
  LJ:=0;
  for LI:=Length(APath) downto 1 do
   if APath[LI]='\' then
   begin
    LJ:=LI;
    Break;
   end;
  Result:=Copy(APath,LJ+1,MaxInt);
 end else Result:='';
end;


procedure Usage;
var
 LStr:string;
begin
end;

procedure ErrorMsg(AErrorMsg:string);
begin
 frmMain.AddLog(AErrorMsg,2);
end;

function UpperCase(AStr:string):string;
//вибір для рядка
var
 LI:Integer;
begin
 SetLength(Result,Length(AStr));
 for LI:=1 to Length(AStr) do Result[LI]:=UpCase(AStr[LI]);
end;

{$R-}
function IntToHex(ACard:Cardinal;ADigits:Byte):string;
//перетворення числа у рядок hex
const
 HexArray:array[0..15] of
Char=('0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F');
var
 LHex:string;
 LInt,LHInt:Cardinal;
begin
 LHex:=StringOfChar('0',ADigits);
 LInt:=ADigits;
 LHInt:=16;
 while ACard>0 do
 begin
  LHex[LInt]:=HexArray[(ACard mod LHint) mod 16];
  ACard:=ACard div 16;
  LHInt:=LHInt*16;
  if LHInt=0 then LHInt:=$FFFFFFFF;
  Dec(LInt);
 end;
 Result:=LHex;
```

```
end;

function HexToInt(AHex:string):Cardinal;
//перетворення hex рядку у число
var
 LI,LO:Byte;
 LM:Cardinal;
begin
 LM:=1;
 Result:=0;
 AHex:=UpperCase(AHex);
 if (Length(AHex)>2) and (AHex[2]='X') then AHex:=Copy(AHex,3,MaxInt);

 for LI:=Length(AHex) downto 1 do
 begin
  if not ((AHex[LI] in ['0'..'9']) or (AHex[LI] in ['A'..'F'])) then
  begin
   Result:=0;
   Exit;
  end;
  if AHex[LI] in ['0'..'9'] then LO:=48 else LO:=55;
  LO:=Ord(AHex[LI])-LO;
  Result:=Result+LO*LM;
  LM:=LM shl 4;
 end;
end;
{$R+}


function CheckPEFile(AData:PByte):Boolean;
//повертає True якщо крапка AData валідного файлу образу
var
 LPNtHdr:PImageNtHeaders;
begin
 Result:=False;
 try
  if PImageDosHeader(AData)^.e_magic<>PWord(PChar('MZ'))^ then Exit;

  LPNtHdr:=Pointer(Cardinal(AData)+Cardinal(PImageDosHeader(AData)^._lfanew));

  if LPNtHdr^.Signature<>PCardinal(PChar('PE'))^ then Exit;
  if LPNtHdr^.FileHeader.Machine<>IMAGE_FILE_MACHINE_I386 then Exit;
  if LPNtHdr^.OptionalHeader.Magic<>IMAGE_NT_OPTIONAL_HDR_MAGIC then Exit;
  Result:=True;
 except
 end;
end;

function ProcessCmdLine:Boolean;
//командний рядок процесу, повертає True якщо аргументи відповідають дійсності
var
 LI:Integer;
 LPar,LUpArg:string;
begin
 Result:=False;

 Options:='';
 Quiet:=False;
 DynamicDLL:=False;
 SaveIcon:=True;
 SaveOverlay:=False;
 ReqImageBase:=0;
 InputFileName:='';
 OutputFileName:='';

 if (ParamCount<1) or (ParamCount>5) then Exit;
 LI:=1;
 while LI<=ParamCount do
 begin
```

```
  LPar:=ParamStr(LI);
  LUpArg:=UpperCase(LPar);
  if LUpArg[1]='-' then
  begin
   if Length(LUpArg)=1 then Break;
   case LUpArg[2] of
    'Q':Quiet:=True;
    'D':DynamicDLL:=True;
    'I':SaveIcon:=False;
    'A':SaveOverlay:=True;
    'B','O':begin
     if Length(LUpArg)<4 then Break;
     if LUpArg[3]<>':' then Break;
     if LUpArg[2]='B' then
     begin
      ReqImageBase:=HexToInt(Copy(LUpArg,4,MaxInt));
      if ReqImageBase=0 then Break;
     end else OutputFileName:=Copy(LPar,4,MaxInt);
    end;
    else Break;
   end;
  end else
  begin
   InputFileName:=LPar;
   Inc(LI);
   Break;
  end;
  Inc(LI);
 end;
 if Length(OutputFileName)=0 then OutputFileName:=InputFileName;
 Result:=(LI-1=ParamCount) and (Length(InputFileName)>0);
end;

function MyAlloc(ASize:Cardinal):Pointer;
//виділяє пам'ять для VirtualAlloc
begin
 Result:=VirtualAlloc(nil,ASize,MEM_COMMIT,PAGE_EXECUTE_READWRITE);
end;

function MyFree(APtr:Pointer):Boolean;
// визволяє пам'ять для VirtualAlloc
begin
 if APtr<>nil then Result:=VirtualFree(APtr,0,MEM_RELEASE)
 else Result:=False;
end;

procedure PrepareResourceSectionData;
//розпаковує та заповнює блок ресурсу
var
 LTypeTable:record
  Directory:TResourceDirectoryTable;
  IconsEntry,IconGroupEntry,XPEntry:TResourceDirectoryEntry;
 end;

 LXPManifest:record
  NameDir:TResourceTableDirectoryEntry;
  LangDir:TResourceTableDirectoryEntry;
  DataEntry:TResourceDataEntry;
 end;

 LIconGroup:record
  GroupNameDir:TResourceTableDirectoryEntry;
  GroupLangDir:TResourceTableDirectoryEntry;
  GroupData:TResourceDataEntry;

  IconCount:Integer;

  NameDir:TResourceDirectoryTable;
  NameEntries:array[0..31] of TResourceDirectoryEntry;
```

```
  LangDirs:array[0..31] of TResourceTableDirectoryEntry;

  DataEntries:array[0..31] of TResourceDataEntry;
 end;

 LResourceStrings:array[0..1023] of Char;
 LResourceData:array[0..65535] of Char;
 LResourceStringsPtr,LResourceDataPtr:Cardinal;

 LIconDirectory:PIconDirectory;

 LNameEntry:PResourceDirectoryEntry;
 LLangEntry:PResourceTableDirectoryEntry;
 LDataEntry:PResourceDataEntry;

LNameRVA,LSubEntryRVA,LSize,LResStringsRVA,LResDataRVA,LManifestSize,LResRawRVA:
Cardinal;
 LNameLen:Word;
 LPB,LPBManifest:PByte;
 LI:Integer;
 LImage:HMODULE;
 LIcoRes:HRSRC;

begin
 LImage:=LoadLibraryEx(PChar(InputFileName),0,LOAD_LIBRARY_AS_DATAFILE);
 ResourceSectionSize:=0;
 ZeroMemory(@LTypeTable,SizeOf(LTypeTable));
 if ResourceIconGroup<>nil then Inc(LTypeTable.Directory.NumberOfIDEntries,2);
 if ResourceXPManifest<>nil then Inc(LTypeTable.Directory.NumberOfIDEntries);
 LTypeTable.IconsEntry.NameID:=Cardinal(RT_ICON);
 LTypeTable.IconsEntry.SubdirDataRVA:=$80000000;
 LTypeTable.IconGroupEntry.NameID:=Cardinal(RT_GROUP_ICON);
 LTypeTable.IconGroupEntry.SubdirDataRVA:=$80000000;
 LTypeTable.XPEntry.NameID:=RT_XP_MANIFEST;
 LTypeTable.XPEntry.SubdirDataRVA:=$80000000;

 LResourceStringsPtr:=0;
 LResourceDataPtr:=0;

 if ResourceIconGroup<>nil then
 begin
  ZeroMemory(@LIconGroup,SizeOf(LIconGroup));
  LPB:=Pointer(ResourceIconGroup);
  Inc(LPB,SizeOf(TResourceDirectoryTable));
  LNameEntry:=Pointer(LPB);

  LNameRVA:=LNameEntry^.NameID and $7FFFFFFF;
  LSubEntryRVA:=LNameEntry^.SubdirDataRVA and $7FFFFFFF;

  if LNameEntry^.NameID and $80000000<>0 then
  begin
   LIconGroup.GroupNameDir.Table.NumberOfNameEntries:=1;
   LPB:=RVA2RAW(Ptr,MainData,HostResourceSectionVirtualAddress+LNameRVA);
   LNameLen:=2*PWord(LPB)^;
   LIconGroup.GroupNameDir.Directory.NameID:=LResourceStringsPtr+$80000000;
   CopyMemory(@LResourceStrings[LResourceStringsPtr],LPB,LNameLen+2);
   Inc(LResourceStringsPtr,LNameLen+2);
  end else
  begin
   LIconGroup.GroupNameDir.Directory.NameID:=LNameEntry^.NameID;
   LIconGroup.GroupNameDir.Table.NumberOfIDEntries:=1;
  end;
  LIconGroup.GroupNameDir.Directory.SubdirDataRVA:=0;


LLangEntry:=RVA2RAW(Ptr,MainData,HostResourceSectionVirtualAddress+LSubEntryRVA)
;
  LSubEntryRVA:=LLangEntry^.Directory.SubdirDataRVA and $7FFFFFFF;
```

```
  LIconGroup.GroupLangDir.Table.NumberOfIDEntries:=1;
  LIconGroup.GroupLangDir.Directory.NameID:=LLangEntry^.Directory.NameID;
  LIconGroup.GroupLangDir.Directory.SubdirDataRVA:=0;


LDataEntry:=RVA2RAW(Ptr,MainData,HostResourceSectionVirtualAddress+LSubEntryRVA)
;
  LPB:=RVA2RAW(Ptr,MainData,LDataEntry^.DataRVA);
  LIconGroup.GroupData.Size:=LDataEntry^.Size;
  LIconGroup.GroupData.DataRVA:=LResourceDataPtr;
  LIconGroup.GroupData.Codepage:=LDataEntry^.Codepage;

  CopyMemory(@LResourceData[LResourceDataPtr],LPB,LDataEntry^.Size);
  Inc(LResourceDataPtr,LDataEntry^.Size);

  LIconDirectory:=Pointer(LPB);
  LIconGroup.IconCount:=LIconDirectory^.Count;
  LIconGroup.NameDir.NumberOfIDEntries:=LIconGroup.IconCount;
  for LI:=0 to LIconDirectory^.Count-1 do
  begin
   LIconGroup.NameEntries[LI].NameID:=LIconDirectory^.Entries[LI].ID;
   LIconGroup.NameEntries[LI].SubdirDataRVA:=$80000000;
   LIconGroup.LangDirs[LI].Table.NumberOfIDEntries:=1;
   LIconGroup.LangDirs[LI].Directory.SubdirDataRVA:=$80000000;


LIcoRes:=FindResource(LImage,MakeIntResource(LIconDirectory^.Entries[LI].ID),RT_
ICON);
   LPB:=LockResource(LoadResource(LImage,LIcoRes));
   LSize:=SizeofResource(LImage,LIcoRes);
   LIconGroup.DataEntries[LI].Size:=LSize;
   LIconGroup.DataEntries[LI].DataRVA:=LResourceDataPtr;

   CopyMemory(@LResourceData[LResourceDataPtr],LPB,LSize);
   Inc(LResourceDataPtr,LSize);
  end;

  LSize:=6+LIconDirectory^.Count*SizeOf(TIconDirectoryEntry);
  CopyMemory(@LResourceData[LResourceDataPtr],LIconDirectory,LSize);
  Inc(LResourceDataPtr,LSize);
 end;

 if ResourceXPManifest<>nil then
 begin
  LPB:=Pointer(ResourceXPManifest);
  Inc(LPB,SizeOf(TResourceDirectoryTable));
  LNameEntry:=Pointer(LPB);
  LNameRVA:=LNameEntry^.NameID and $7FFFFFFF;
  LSubEntryRVA:=LNameEntry^.SubdirDataRVA and $7FFFFFFF;

  LXPManifest.NameDir.Table.NumberOfIDEntries:=1;
  LXPManifest.NameDir.Directory.NameID:=LNameRVA;
  LXPManifest.NameDir.Directory.SubdirDataRVA:=$80000000;


LLangEntry:=RVA2RAW(Ptr,MainData,HostResourceSectionVirtualAddress+LSubEntryRVA)
;
  LNameRVA:=LLangEntry^.Directory.NameID and $7FFFFFFF;
  LSubEntryRVA:=LLangEntry^.Directory.SubdirDataRVA and $7FFFFFFF;
  LXPManifest.LangDir.Table.NumberOfIDEntries:=1;
  LXPManifest.LangDir.Directory.NameID:=LNameRVA;
  LXPManifest.LangDir.Directory.SubdirDataRVA:=$80000000;


LDataEntry:=RVA2RAW(Ptr,MainData,HostResourceSectionVirtualAddress+LSubEntryRVA)
;
  LPB:=RVA2RAW(Ptr,MainData,LDataEntry^.DataRVA);
  LXPManifest.DataEntry.DataRVA:=LResourceDataPtr;
  LXPManifest.DataEntry.Size:=LDataEntry^.Size;
```

```
  LXPManifest.DataEntry.Codepage:=LDataEntry^.Codepage;

  CopyMemory(@LResourceData[LResourceDataPtr],LPB,LDataEntry^.Size);
  Inc(LResourceDataPtr,LDataEntry^.Size);
 end;

 LPB:=ResourceData;
 LManifestSize:=0;
 if ResourceXPManifest<>nil then
LManifestSize:=2*SizeOf(TResourceTableDirectoryEntry);

 LSubEntryRVA:=SizeOf(LTypeTable.Directory) or $80000000;
 if ResourceIconGroup<>nil then
Inc(LSubEntryRVA,SizeOf(LTypeTable.IconsEntry)+SizeOf(LTypeTable.IconGroupEntry)
);
 if ResourceXPManifest<>nil then Inc(LSubEntryRVA,SizeOf(LTypeTable.XPEntry));
 if ResourceIconGroup=nil then LTypeTable.XPEntry.SubdirDataRVA:=LSubEntryRVA
 else LTypeTable.IconsEntry.SubdirDataRVA:=LSubEntryRVA;
 LSize:=LSubEntryRVA and $7FFFFFFF;
 Inc(LPB,LSize);

 if ResourceIconGroup=nil then
 begin
  LResDataRVA:=LSubEntryRVA and $7FFFFFFF;
  Inc(LResDataRVA,SizeOf(LXPManifest.NameDir));
  Inc(LResDataRVA,SizeOf(LXPManifest.LangDir));
 end else
 begin
  LResStringsRVA:=LSubEntryRVA;
  Inc(LResStringsRVA,SizeOf(LIconGroup.NameDir));
  Inc(LResStringsRVA,LIconGroup.IconCount*SizeOf(TResourceDirectoryEntry));
  Inc(LResStringsRVA,LIconGroup.IconCount*SizeOf(TResourceTableDirectoryEntry));
  Inc(LResStringsRVA,SizeOf(LIconGroup.GroupNameDir));
  Inc(LResStringsRVA,SizeOf(LIconGroup.GroupLangDir));
  Inc(LResStringsRVA,LManifestSize);
  LResDataRVA:=LResStringsRVA and $7FFFFFFF+LResourceStringsPtr;

  //icons - name directory
  LSize:=SizeOf(LIconGroup.NameDir);
  Inc(LSubEntryRVA,LSize);
  CopyMemory(LPB,@LIconGroup.NameDir,LSize);
  Inc(LPB,LSize);

  //іконки – ім’я введення
  LSize:=LIconGroup.IconCount*SizeOf(TResourceDirectoryEntry);
  Inc(LSubEntryRVA,LSize);
  for LI:=0 to LIconGroup.IconCount-1 do
  begin
   LIconGroup.NameEntries[LI].SubdirDataRVA:=LSubEntryRVA;
   Inc(LSubEntryRVA,SizeOf(TResourceTableDirectoryEntry));
  end;
  CopyMemory(LPB,@LIconGroup.NameEntries,LSize);
  Inc(LPB,LSize);

  //іконки – ім’я директорії + введення
  LSize:=LIconGroup.IconCount*SizeOf(TResourceTableDirectoryEntry);
  for LI:=0 to LIconGroup.IconCount-1 do
  begin
   LIconGroup.LangDirs[LI].Directory.SubdirDataRVA:=LResDataRVA;
   Inc(LResDataRVA,SizeOf(TResourceDataEntry));
  end;
  CopyMemory(LPB,@LIconGroup.LangDirs,LSize);
  Inc(LPB,LSize);

  //іконка групи - ім’я директорії
  LTypeTable.IconGroupEntry.SubdirDataRVA:=LSubEntryRVA;
  LSize:=SizeOf(LIconGroup.GroupNameDir.Table);
  Inc(LSubEntryRVA,LSize);
  CopyMemory(LPB,@LIconGroup.GroupNameDir.Table,LSize);
```

```
  Inc(LPB,LSize);

  //іконка групи - ім'я введення
  if LIconGroup.GroupNameDir.Directory.NameID and $80000000<>0 then
   LIconGroup.GroupNameDir.Directory.NameID:=LResStringsRVA or $80000000;

  LSize:=SizeOf(LIconGroup.GroupNameDir.Directory);
  Inc(LSubEntryRVA,LSize);
  LIconGroup.GroupNameDir.Directory.SubdirDataRVA:=LSubEntryRVA;
  CopyMemory(LPB,@LIconGroup.GroupNameDir.Directory,LSize);
  Inc(LPB,LSize);

  /іконка групи – мова директорії + введення
  LIconGroup.GroupLangDir.Directory.SubdirDataRVA:=LResDataRVA;
  LSize:=SizeOf(LIconGroup.GroupLangDir);
  Inc(LSubEntryRVA,LSize);
  Inc(LResDataRVA,SizeOf(TResourceDataEntry));
  CopyMemory(LPB,@LIconGroup.GroupLangDir,LSize);
  Inc(LPB,LSize);
 end;

 if ResourceXPManifest<>nil then
 begin
  LTypeTable.XPEntry.SubdirDataRVA:=LSubEntryRVA;

  //визначення- ім'я директорії + введення
  LSize:=SizeOf(LXPManifest.NameDir);
  Inc(LSubEntryRVA,LSize);
  LXPManifest.NameDir.Directory.SubdirDataRVA:=LSubEntryRVA;
  CopyMemory(LPB,@LXPManifest.NameDir,LSize);
  Inc(LPB,LSize);

  //визначення- мова директорії + введення
  LSize:=SizeOf(LXPManifest.LangDir);
  LXPManifest.LangDir.Directory.SubdirDataRVA:=LResDataRVA;
  Inc(LResDataRVA,SizeOf(TResourceDataEntry));
  CopyMemory(LPB,@LXPManifest.LangDir,LSize);
  Inc(LPB,LSize);
 end;

 //рядки
 CopyMemory(LPB,@LResourceStrings,LResourceStringsPtr);
 Inc(LPB,LResourceStringsPtr);

 LResRawRVA:=LResDataRVA and $7FFFFFFF;

 if ResourceIconGroup<>nil then
 begin
  //іконки - дані
  LSize:=SizeOf(TResourceDataEntry)*LIconGroup.IconCount;
  for LI:=0 to LIconGroup.IconCount-1 do

Inc(LIconGroup.DataEntries[LI].DataRVA,LResRawRVA+ResourceSection.VirtualAddress
);
  CopyMemory(LPB,@LIconGroup.DataEntries,LSize);
  Inc(LPB,LSize);

  /іконка групи - дані
  LSize:=SizeOf(LIconGroup.GroupData);
  Inc(LIconGroup.GroupData.DataRVA,LResRawRVA+ResourceSection.VirtualAddress);
  CopyMemory(LPB,@LIconGroup.GroupData,LSize);
  Inc(LPB,LSize);
 end;

 if ResourceXPManifest<>nil then
 begin
  //визначення - дані
  LSize:=SizeOf(LXPManifest.DataEntry);
  Inc(LXPManifest.DataEntry.DataRVA,LResRawRVA+ResourceSection.VirtualAddress);
```

```
  CopyMemory(LPB,@LXPManifest.DataEntry,LSize);
  Inc(LPB,LSize);
 end;

 CopyMemory(LPB,@LResourceData,LResourceDataPtr);
 Inc(LPB,LResourceDataPtr);
 ResourceSectionSize:=Cardinal(LPB)-Cardinal(ResourceData);

 LPB:=ResourceData;
 CopyMemory(LPB,@LTypeTable,SizeOf(LTypeTable.Directory));
 Inc(LPB,SizeOf(LTypeTable.Directory));
 if ResourceIconGroup<>nil then
 begin
  CopyMemory(LPB,@LTypeTable.IconsEntry,SizeOf(LTypeTable.IconsEntry));
  Inc(LPB,SizeOf(LTypeTable.IconsEntry));
  CopyMemory(LPB,@LTypeTable.IconGroupEntry,SizeOf(LTypeTable.IconGroupEntry));
  Inc(LPB,SizeOf(LTypeTable.IconGroupEntry));
 end;
 if ResourceXPManifest<>nil then
  CopyMemory(LPB,@LTypeTable.XPEntry,SizeOf(LTypeTable.XPEntry));

 FreeLibrary(LImage);
end;

function GenerateEncoderDecoder(AHostSize:Cardinal;out
OEncoder,ODecoder:Pointer):Cardinal;
//генератор шифратора та дешифратора для головного файлу, повертає розмір
декодера
const
 CI_XOR         = 00;
 CI_ADD         = 01;
 CI_SUB         = 02;
 CI_ROR         = 03;
 CI_ROL         = 04;
 CI_NOT         = 05;
 CI_NEG         = 06;
 CI_BSWAP       = 07;
 CI_XOR_OFS     = 08;
 CI_ADD_OFS     = 09;
 CI_SUB_OFS     = 10;
 CI_XOR_SMH     = 11;
 CI_ADD_SMH     = 12;
 CI_SUB_SMH     = 13;
 CI_SMH_ADD     = 14;
 CI_SMH_SUB     = 15;
 CI_MAX         = 16;

type
 TCoderInstruction=packed record
  IType,ILen:Byte;
  IArg1,IArg2,IArg3:Cardinal;
 end;
 TCoderContext=record
  DataSizeRegister:Byte;
  DataAddrRegister:Byte;
  DataRegister:Byte;
  OffsetRegister:Byte;
  SmashRegister:Byte;
  FreeRegister:Byte;
 end;

 TCoder=array[0..255] of TCoderInstruction;

var
 LCoderContext:TCoderContext;
 LEncoder,LDecoder:TCoder;
 LEncoderData,LDecoderData:array[0..511] of Char;
 LInstrCount,LReg:Byte;
 LI,LNotUsed:Integer;
```

```
 LRegUsed:array[0..Reg32Count-1] of Boolean;
 LPB,LPB2:PByte;
 LEncSize,LDecSize,LSmashNum:Cardinal;

 procedure GenerateCoderInstruction(ACoder:TCoder;AInstr:Integer);
 begin
  case ACoder[LI].IType of
   CI_XOR:XorReg32Num32(LPB,LCoderContext.DataRegister,ACoder[LI].IArg1);
   CI_ADD:AddReg32Num32(LPB,LCoderContext.DataRegister,ACoder[LI].IArg1);
   CI_SUB:SubReg32Num32(LPB,LCoderContext.DataRegister,ACoder[LI].IArg1);
   CI_ROR:RorReg32Num8(LPB,LCoderContext.DataRegister,ACoder[LI].IArg1);
   CI_ROL:RolReg32Num8(LPB,LCoderContext.DataRegister,ACoder[LI].IArg1);
   CI_NOT:NotReg32(LPB,LCoderContext.DataRegister);
   CI_NEG:NegReg32(LPB,LCoderContext.DataRegister);
   CI_BSWAP:Bswap(LPB,LCoderContext.DataRegister);

CI_XOR_OFS:XorReg32Reg32(LPB,LCoderContext.DataRegister,LCoderContext.OffsetRegi
ster);

CI_ADD_OFS:AddReg32Reg32(LPB,LCoderContext.DataRegister,LCoderContext.OffsetRegi
ster);

CI_SUB_OFS:SubReg32Reg32(LPB,LCoderContext.DataRegister,LCoderContext.OffsetRegi
ster);

CI_XOR_SMH:XorReg32Reg32(LPB,LCoderContext.DataRegister,LCoderContext.SmashRegis
ter);

CI_ADD_SMH:AddReg32Reg32(LPB,LCoderContext.DataRegister,LCoderContext.SmashRegis
ter);

CI_SUB_SMH:SubReg32Reg32(LPB,LCoderContext.DataRegister,LCoderContext.SmashRegis
ter);
   CI_SMH_ADD:AddReg32Num32(LPB,LCoderContext.SmashRegister,ACoder[LI].IArg1);
   CI_SMH_SUB:SubReg32Num32(LPB,LCoderContext.SmashRegister,ACoder[LI].IArg1);
  end;
 end;

begin
 LInstrCount:=Random(32)+16;                    //число інструкцій у
кодуванні/декодуванні
 LSmashNum:=Cardinal(Random($FFFFFFFF));

 //спершу генеруємо контекст кодувальника
 for LI:=0 to Reg32Count-1 do LRegUsed[LI]:=False;
 LRegUsed[REG_ESP]:=True;
 LRegUsed[REG_EBP]:=True;
 LNotUsed:=Reg32Count-2;
 while LNotUsed>0 do
 begin
  LReg:=Random(Reg32Count);
  while LRegUsed[LReg] do LReg:=(LReg+1) mod Reg32Count;
  LRegUsed[LReg]:=True;

  with LCoderContext do
  case LNotUsed of
   1:DataSizeRegister:=LReg;
   2:DataAddrRegister:=LReg;
   3:DataRegister:=LReg;
   4:OffsetRegister:=LReg;
   5:SmashRegister:=LReg;
   6:FreeRegister:=LReg;
  end;
  Dec(LNotUsed);
 end;

 // генеруємо кодер/декодер
 for LI:=0 to LInstrCount-1 do
 begin
```

```
LEncoder[LI].IType:=Random(CI_MAX);
case LEncoder[LI].IType of
 CI_XOR:begin
  //DataRegister = DataRegister xor IArg1
  LEncoder[LI].IArg1:=Cardinal(Random($FFFFFFFF));
  LDecoder[LI].IType:=CI_XOR;
  LDecoder[LI].IArg1:=LEncoder[LI].IArg1;
 end;

 CI_ADD:begin
  //DataRegister = DataRegister + IArg1
  LEncoder[LI].IArg1:=Cardinal(Random($FFFFFFFF));
  //DataRegister = DataRegister - IArg1
  LDecoder[LI].IType:=CI_SUB;
  LDecoder[LI].IArg1:=LEncoder[LI].IArg1;
 end;

 CI_SUB:begin
  //DataRegister = DataRegister - IArg1
  LEncoder[LI].IArg1:=Cardinal(Random($FFFFFFFF));
  //DataRegister = DataRegister + IArg1
  LDecoder[LI].IType:=CI_ADD;
  LDecoder[LI].IArg1:=LEncoder[LI].IArg1;
 end;

 CI_ROR:begin
  //DataRegister = DataRegister ror IArg1
  LEncoder[LI].IArg1:=Random($100);
  //DataRegister = DataRegister rol IArg1
  LDecoder[LI].IType:=CI_ROL;
  LDecoder[LI].IArg1:=LEncoder[LI].IArg1;
 end;

 CI_ROL:begin
  //DataRegister = DataRegister rol IArg1
  LEncoder[LI].IArg1:=Random($100);
  //DataRegister = DataRegister ror IArg1
  LDecoder[LI].IType:=CI_ROR;
  LDecoder[LI].IArg1:=LEncoder[LI].IArg1;
 end;

 CI_NOT:begin
  //DataRegister = not DataRegister
  LDecoder[LI].IType:=CI_NOT;
 end;

 CI_NEG:begin
  //DataRegister = -DataRegister
  LDecoder[LI].IType:=CI_NEG;
 end;

 CI_BSWAP:begin
  //DataRegister = swaped DataRegister
  LDecoder[LI].IType:=CI_BSWAP;
 end;

 CI_XOR_OFS:begin
  //DataRegister = DataRegister xor OffsetRegister
  LDecoder[LI].IType:=CI_XOR_OFS;
 end;

 CI_ADD_OFS:begin
  //DataRegister = DataRegister + OffsetRegister
  //DataRegister = DataRegister - OffsetRegister
  LDecoder[LI].IType:=CI_SUB_OFS;
 end;

 CI_SUB_OFS:begin
  //DataRegister = DataRegister + OffsetRegister
```

```
   //DataRegister = DataRegister - OffsetRegister
   LDecoder[LI].IType:=CI_ADD_OFS;
  end;

  CI_XOR_SMH:begin
   //DataRegister = DataRegister xor SmashRegister
   LDecoder[LI].IType:=CI_XOR_SMH;
  end;

  CI_ADD_SMH:begin
   //DataRegister = DataRegister + SmashRegister
   //DataRegister = DataRegister - SmashRegister
   LDecoder[LI].IType:=CI_SUB_SMH;
  end;

  CI_SUB_SMH:begin
   //DataRegister = DataRegister + SmashRegister
   //DataRegister = DataRegister - SmashRegister
   LDecoder[LI].IType:=CI_ADD_SMH;
  end;

  CI_SMH_ADD:begin
   //SmashRegister = SmashRegister + IArg1
   LEncoder[LI].IArg1:=Cardinal(Random($FFFFFFFF));
   //SmashRegister = SmashRegister - IArg1
   LDecoder[LI].IType:=CI_SMH_SUB;
   LDecoder[LI].IArg1:=LEncoder[LI].IArg1;
  end;

  CI_SMH_SUB:begin
   //SmashRegister = SmashRegister - IArg1
   LEncoder[LI].IArg1:=Cardinal(Random($FFFFFFFF));
   //SmashRegister = SmashRegister + IArg1
   LDecoder[LI].IType:=CI_SMH_ADD;
   LDecoder[LI].IArg1:=LEncoder[LI].IArg1;
  end;
 end;
end;


LPB:=@LEncoderData;
// заглушка
PushReg32(LPB,REG_EBX);
PushReg32(LPB,REG_ESI);
PushReg32(LPB,REG_EDI);
MovReg32RegMemIdx8(LPB,LCoderContext.DataAddrRegister,REG_ESP,$10);
MovReg32Num32(LPB,LCoderContext.DataSizeRegister,AHostSize);
XorReg32Reg32(LPB,LCoderContext.OffsetRegister,LCoderContext.OffsetRegister);
MovReg32Num32(LPB,LCoderContext.SmashRegister,LSmashNum);
//головний цикл
LPB2:=LPB;
//редагування даних
MovReg32RegMem(LPB,LCoderContext.DataRegister,LCoderContext.DataAddrRegister);
// генеруємо інструкції кодування
for LI:=0 to LInstrCount-1 do
 GenerateCoderInstruction(LEncoder,LI);
//запам'ятовуємо дані
MovRegMemReg32(LPB,LCoderContext.DataAddrRegister,LCoderContext.DataRegister);
//збільшуємо вказівник на дані
AddReg32Num8(LPB,LCoderContext.DataAddrRegister,4);
//збільшуємо зсув
AddReg32Num8(LPB,LCoderContext.OffsetRegister,4);
//кінець даних?
CmpReg32Reg32(LPB,LCoderContext.OffsetRegister,LCoderContext.DataSizeRegister);
RelJnzAddr32(LPB,-((Cardinal(LPB)+6)-Cardinal(LPB2)));
//повернення
MovReg32Reg32(LPB,REG_EAX,LCoderContext.SmashRegister);
PopReg32(LPB,REG_EDI);
PopReg32(LPB,REG_ESI);
```

```
 PopReg32(LPB,REG_EBX);
 Ret16(LPB,4);

 LEncSize:=Cardinal(LPB)-Cardinal(@LEncoderData);
 OEncoder:=MyAlloc(LEncSize);
 if OEncoder=nil then
 begin
  Result:=0;
  Exit;
 end;
 CopyMemory(OEncoder,@LEncoderData,LEncSize);

 EncoderProc:=OEncoder;
 LSmashNum:=EncoderProc(MainDataCyp);

 LPB:=@LDecoderData;
 // заглушка
 PopReg32(LPB,LCoderContext.DataAddrRegister);
 AddReg32Num32(LPB,LCoderContext.DataAddrRegister,AHostSize);
 MovReg32Num32(LPB,LCoderContext.DataSizeRegister,AHostSize);
 MovReg32Num32(LPB,LCoderContext.OffsetRegister,AHostSize);
 MovReg32Num32(LPB,LCoderContext.SmashRegister,LSmashNum);
 //головний цикл
 LPB2:=LPB;
 //dec offset
 SubReg32Num8(LPB,LCoderContext.OffsetRegister,4);
 //dec data ptr
 SubReg32Num8(LPB,LCoderContext.DataAddrRegister,4);
 //редагування даних
 MovReg32RegMem(LPB,LCoderContext.DataRegister,LCoderContext.DataAddrRegister);
 // генеруємо інструкції декодування
 for LI:=LInstrCount-1 downto 0 do
  GenerateCoderInstruction(LDecoder,LI);
 //запам'ятовуємо дані
 MovRegMemReg32(LPB,LCoderContext.DataAddrRegister,LCoderContext.DataRegister);
 //кінець даних?
 TestReg32Reg32(LPB,LCoderContext.OffsetRegister,LCoderContext.OffsetRegister);
 RelJnzAddr32(LPB,-((Cardinal(LPB)+6)-Cardinal(LPB2)));

 LDecSize:=Cardinal(LPB)-Cardinal(@LDecoderData);

 ODecoder:=MyAlloc(LDecSize);
 if ODecoder=nil then
 begin
  MyFree(OEncoder);
  OEncoder:=nil;
  Result:=0;
 end else
 begin
  CopyMemory(ODecoder,@LDecoderData,LDecSize);
  Result:=LDecSize;
 end;
end;

procedure FindAfterImageOverlays;
//переглядаємо оверлейні дані у MainData записані у кінці заповненого файлу
AfterImageOverlays, точка визначення його розміру - AfterImageOverlaysSize
var
 LI:Integer;
 LPSection:PImageSectionHeader;
 LMaxAddr,LDataSize:Cardinal;
 LHdr:PImageNtHeaders;

begin
 AfterImageOverlays:=nil;
 AfterImageOverlaysSize:=0;
 LMaxAddr:=0;
 LHdr:=Pointer(Cardinal(MainData)+Cardinal(PImageDosHeader(MainData)^._lfanew));
```

```
LPSection:=Pointer(Cardinal(@LHdr^.OptionalHeader)+LHdr^.FileHeader.SizeOfOption
alHeader);

 for LI:=0 to LHdr^.FileHeader.NumberOfSections-1 do
 begin
  LDataSize:=RoundSize(LPSection^.SizeOfRawData,RawDataAlignment);
  if LPSection^.PointerToRawData+LDataSize>LMaxAddr then
LMaxAddr:=LPSection^.PointerToRawData+LDataSize;
  Inc(LPSection);
 end;
 if (LMaxAddr>0) and (LMaxAddr<MainRealSize) then
 begin
  AfterImageOverlays:=Pointer(Cardinal(MainData)+LMaxAddr);
  AfterImageOverlaysSize:=MainRealSize-LMaxAddr;
 end;
end;


procedure Protect(InputFileName: string);
begin
 Randomize;
 SaveIcon:=frmMain.cbIcon.Checked;
 DynamicDLL:=False;
 SaveOverlay:=frmMain.cbOverlay.Checked;
 ReqImageBase:=HexToInt(frmMain.txtImageBase.Text);

 OutputFileName:=InputFileName;

MainFile:=CreateFile(PChar(InputFileName),GENERIC_READ,FILE_SHARE_READ,nil,OPEN_
EXISTING,0,0);
 if MainFile<>INVALID_HANDLE_VALUE then
 begin
  MainRealSize:=GetFileSize(MainFile,nil);

  MainRealSize4:=MainRealSize;
  frmMain.AddLog('Захищає...',0);
  if MainRealSize4 mod 4<>0 then Inc(MainRealSize4,4-MainRealSize4 mod 4);

  MainSize:=MainRealSize+Cardinal(Random(100)+10);
  MainData:=MyAlloc(MainSize);
  MainDataCyp:=MyAlloc(MainSize);
  if (MainData<>nil) and (MainDataCyp<>nil) then
  begin
   GenerateRandomBuffer(MainData,MainSize);
   ZeroMemory(MainData,MainRealSize4);
   if ReadFile(MainFile,MainData^,MainRealSize,NumBytes,nil) then
   begin
    CloseHandle(MainFile);
    MainFile:=INVALID_HANDLE_VALUE;
    CopyMemory(MainDataCyp,MainData,MainSize);
    if CheckPEFile(MainData) then
    begin

Ptr:=Pointer(Cardinal(MainData)+Cardinal(PImageDosHeader(MainData)^._lfanew));

     ImageType:=itExe;
     HostCharacteristics:=PImageNtHeaders(Ptr)^.FileHeader.Characteristics;
     if HostCharacteristics and IMAGE_FILE_DLL<>0 then ImageType:=itDLL;

HostExportSectionVirtualAddress:=PImageNtHeaders(Ptr)^.OptionalHeader.DataDirect
ory[IMAGE_DIRECTORY_ENTRY_EXPORT].VirtualAddress;

     HostImageBase:=PImageNtHeaders(Ptr)^.OptionalHeader.ImageBase;
     HostSubsystem:=PImageNtHeaders(Ptr)^.OptionalHeader.Subsystem;
     HostSizeOfImage:=PImageNtHeaders(Ptr)^.OptionalHeader.SizeOfImage;

HostImportSectionSize:=PImageNtHeaders(Ptr)^.OptionalHeader.DataDirectory[IMAGE_
DIRECTORY_ENTRY_IMPORT].Size;
```

```
HostImportSectionVirtualAddress:=PImageNtHeaders(Ptr)^.OptionalHeader.DataDirect
ory[IMAGE_DIRECTORY_ENTRY_IMPORT].VirtualAddress;


HostResourceSectionVirtualAddress:=PImageNtHeaders(Ptr)^.OptionalHeader.DataDire
ctory[IMAGE_DIRECTORY_ENTRY_RESOURCE].VirtualAddress;

     if (HostSubsystem=IMAGE_SUBSYSTEM_WINDOWS_GUI) or
(HostSubsystem=IMAGE_SUBSYSTEM_WINDOWS_CUI) then
     begin
      FindAfterImageOverlays;


TlsSectionSize:=PImageNtHeaders(Ptr)^.OptionalHeader.DataDirectory[IMAGE_DIRECTO
RY_ENTRY_TLS].Size;
      TlsSectionPresent:=TlsSectionSize<>0;

      ExportSectionPresent:=False;
      ExportSectionSize:=0;
      ExportData:=nil;
      if ImageType=itDLL then
      begin

ExportSectionSize:=PImageNtHeaders(Ptr)^.OptionalHeader.DataDirectory[IMAGE_DIRE
CTORY_ENTRY_EXPORT].Size;
       ExportSectionPresent:=ExportSectionSize<>0;
      end;

      ResourceSectionPresent:=False;
      HostResourceSectionSize:=0;
      ResourceData:=nil;
      if (ImageType=itExe) or (ImageType=itDLL) then
      begin

HostResourceSectionSize:=PImageNtHeaders(Ptr)^.OptionalHeader.DataDirectory[IMAG
E_DIRECTORY_ENTRY_RESOURCE].Size;
       ResourceSectionPresent:=HostResourceSectionSize<>0;
      end;

      OverlayPresent:=(AfterImageOverlays<>nil) and (AfterImageOverlaysSize>0);

       if TlsSectionPresent then
       begin
        frmMain.AddLog('.tls блок присутній',0);
        frmMain.AddLog('Початковий  .tls розмір блоку:
'+IntToStr(TlsSectionSize),0);
       end else frmMain.AddLog('.tls блок не присутній',1);

       if ExportSectionPresent then
       begin
        if not DynamicDLL then
        begin
         frmMain.AddLog('Експортуємий блок присутній',0);
         frmMain.AddLog('Початковий  експортуємий розмір блоку:
'+IntToStr(ExportSectionSize),0);
        end else frmMain.AddLog('Динамічна DLL – експортуємий блок не
використовується',1);
       end else frmMain.AddLog('Експортуємий блок не присутній',1);

       if ResourceSectionPresent then
       begin
        if SaveIcon then frmMain.AddLog('Ресурсний блок присутній',0)
        else frmMain.AddLog('Ресурсний блок присутній але не
використовується',1);
       end else frmMain.AddLog('Ресурсний блок не присутній',1);

       if OverlayPresent then
       begin
```

```
        if SaveOverlay then frmMain.AddLog('Оверлейні дані в наявності',0)
        else frmMain.AddLog('Оверлейні дані присутні але не
використовуються',1);
       end else frmMain.AddLog('Оверлейні дані не присутні ',1);

      if DynamicDLL then ExportSectionPresent:=False;
      if not SaveIcon then ResourceSectionPresent:=False;
      if not SaveOverlay then OverlayPresent:=False;

      if ResourceSectionPresent then
      begin
       ResourceRoot:=RVA2RAW(Ptr,MainData,HostResourceSectionVirtualAddress);

ResourceDirEntry:=RVA2RAW(Ptr,MainData,HostResourceSectionVirtualAddress+SizeOf(
TResourceDirectoryTable));
       ResourceIconGroup:=nil;
       ResourceXPManifest:=nil;
       for I:=0 to
ResourceRoot^.NumberOfIDEntries+ResourceRoot^.NumberOfNameEntries-1 do
       begin
        if (ResourceIconGroup=nil) and
(ResourceDirEntry^.NameID=Cardinal(RT_GROUP_ICON)) then

ResourceIconGroup:=RVA2RAW(Ptr,MainData,HostResourceSectionVirtualAddress+Resour
ceDirEntry^.SubdirDataRVA and $7FFFFFFF);

        if (ResourceXPManifest=nil) and
(ResourceDirEntry^.NameID=Cardinal(RT_XP_MANIFEST)) then

ResourceXPManifest:=RVA2RAW(Ptr,MainData,HostResourceSectionVirtualAddress+Resou
rceDirEntry^.SubdirDataRVA and $7FFFFFFF);

        if not ((ResourceIconGroup=nil) or (ResourceXPManifest=nil)) then Break;
        Inc(ResourceDirEntry);
       end;

       if not ((ResourceIconGroup=nil) and (ResourceXPManifest=nil)) then
       begin
        ResourceData:=MyAlloc(HostResourceSectionSize);
        if ResourceData=nil then
        begin
         ErrorMsg('Unable to allocate memore for resource data');
         ResourceSectionPresent:=False;
        end;
       end else ResourceSectionPresent:=False;
       if not ResourceSectionPresent then
         frmMain.AddLog('Ресурсний блок не має іконки або XP виявлення',1);
      end;


MainDataDecoderLen:=GenerateEncoderDecoder(MainRealSize4,MainDataEncoder,MainDat
aDecoder);
      if MainDataDecoderLen<>0 then
      begin
       LoaderRealSize:=Cardinal(@DynLoader_end)-Cardinal(@DynLoader);
       LoaderSize:=LoaderRealSize+MainDataDecoderLen+Cardinal(Random(100))+4;
       if LoaderSize mod 4>0 then Inc(LoaderSize,4-LoaderSize mod 4);
       frmMain.AddLog('Редактуємо розмір: '+IntToStr(LoaderSize),0);

       LoaderData:=MyAlloc(LoaderSize);
       if LoaderData<>nil then
       begin
        GenerateRandomBuffer(LoaderData,LoaderSize);
        CopyMemory(LoaderData,@DynLoader,LoaderRealSize);

        MainDataDecPtr:=LoaderData;
        while PCardinal(MainDataDecPtr)^<>DYN_LOADER_DEC_MAGIC do
Inc(MainDataDecPtr);
        DynLoaderDecoderOffset:=Cardinal(MainDataDecPtr)-Cardinal(LoaderData);
```

```
CopyMemory(Pointer(Cardinal(MainDataDecPtr)+MainDataDecoderLen),Pointer(Cardinal
(@DynLoader)+DynLoaderDecoderOffset+4),LoaderRealSize-DynLoaderDecoderOffset);
        CopyMemory(MainDataDecPtr,MainDataDecoder,MainDataDecoderLen);

        KeySize:=Random(200)+50;
        KeyPtr:=Random(200);
        LoaderPtr:=KeyPtr+KeySize;
        Trash2Size:=Random(256)+20;

        frmMain.AddLog(' Розмір ключа кодування: '+IntToStr(KeySize),0);
        Key:=MyAlloc(KeySize);
        if Key<>nil then
        begin
         GenerateKey(Key,KeySize);

         ZeroMemory(@DosHeader,SizeOf(DosHeader));
         ZeroMemory(@NtHeaders,SizeOf(NtHeaders));
         ZeroMemory(@DosStubEnd,SizeOf(DosStubEnd));
         DosHeader.e_magic:=PWord(PChar('MZ'))^;
         DosHeader.e_cblp:=$0050;
         DosHeader.e_cp:=$0002;
         DosHeader.e_cparhdr:=$0004;
         DosHeader.e_minalloc:=$000F;
         DosHeader.e_maxalloc:=$FFFF;
         DosHeader.e_sp:=$00B8;
         DosHeader.e_lfarlc:=$0040;
         DosHeader.e_ovno:=$001A;
         DosHeader._lfanew:=$0100;

         NtHeaders.Signature:=PCardinal(PChar('PE'))^;
         NtHeaders.FileHeader.Machine:=IMAGE_FILE_MACHINE_I386;
         NtHeaders.FileHeader.NumberOfSections:=2;
         if TlsSectionPresent then Inc(NtHeaders.FileHeader.NumberOfSections);
         if ExportSectionPresent then
Inc(NtHeaders.FileHeader.NumberOfSections);
        if ResourceSectionPresent then
Inc(NtHeaders.FileHeader.NumberOfSections);
        NtHeaders.FileHeader.TimeDateStamp:=Random($20000000)+$20000000;

NtHeaders.FileHeader.SizeOfOptionalHeader:=IMAGE_SIZEOF_NT_OPTIONAL_HEADER;
        NtHeaders.FileHeader.Characteristics:=IMAGE_FILE_EXECUTABLE_IMAGE or
IMAGE_FILE_LINE_NUMS_STRIPPED
                                             or IMAGE_FILE_LOCAL_SYMS_STRIPPED or
IMAGE_FILE_32BIT_MACHINE;
        case ImageType of

itExe:NtHeaders.FileHeader.Characteristics:=NtHeaders.FileHeader.Characteristics
or IMAGE_FILE_RELOCS_STRIPPED;

itDLL:NtHeaders.FileHeader.Characteristics:=NtHeaders.FileHeader.Characteristics
or IMAGE_FILE_DLL;
        end;
        RandomValue:=Random(10);
        if RandomValue>5 then
NtHeaders.FileHeader.Characteristics:=NtHeaders.FileHeader.Characteristics or
IMAGE_FILE_BYTES_REVERSED_LO or IMAGE_FILE_BYTES_REVERSED_HI;

        NtHeaders.OptionalHeader.Magic:=IMAGE_NT_OPTIONAL_HDR_MAGIC;
        NtHeaders.OptionalHeader.MajorLinkerVersion:=Random(9)+1;
        NtHeaders.OptionalHeader.MinorLinkerVersion:=Random(99)+1;
        NtHeaders.OptionalHeader.BaseOfCode:=$00001000;                //може
змінюватися
        if ReqImageBase<>0 then
NtHeaders.OptionalHeader.ImageBase:=RoundSize(ReqImageBase,$00010000)
        else if HostImageBase=$00400000 then
NtHeaders.OptionalHeader.ImageBase:=RoundSize(HostImageBase+HostSizeOfImage+$001
00000,$00010000)
        else NtHeaders.OptionalHeader.ImageBase:=$00400000;
```

```
        NtHeaders.OptionalHeader.SectionAlignment:=$00001000;          //1000h
= 4096
        NtHeaders.OptionalHeader.FileAlignment:=$00000200;             //може
змінюватися 200h = 512
        NtHeaders.OptionalHeader.MajorOperatingSystemVersion:=$0004;
        NtHeaders.OptionalHeader.MajorSubsystemVersion:=$0004;
        NtHeaders.OptionalHeader.SizeOfHeaders:=$00000400;             //може
змінюватися
        NtHeaders.OptionalHeader.Subsystem:=HostSubsystem;
        NtHeaders.OptionalHeader.SizeOfStackReserve:=$00100000;
        NtHeaders.OptionalHeader.SizeOfStackCommit:=$00010000;         //може
змінюватися
        NtHeaders.OptionalHeader.SizeOfHeapReserve:=$00100000;
        NtHeaders.OptionalHeader.SizeOfHeapCommit:=$00010000;
        NtHeaders.OptionalHeader.NumberOfRvaAndSizes:=$00000010;

        frmMain.AddLog('Побудова  .text блоку',0);

        ZeroMemory(@CodeSection,SizeOf(CodeSection));
        CopyMemory(@CodeSection.Name,PChar('.text'),5);        //може
змінюватися -> CODE
        CodeSection.VirtualAddress:=NtHeaders.OptionalHeader.BaseOfCode;
        CodeSection.PointerToRawData:=NtHeaders.OptionalHeader.SizeOfHeaders;

        InitSize:=Random($280)+$280;

CodeSection.SizeOfRawData:=RoundSize(LoaderPtr+LoaderSize+Trash2Size+InitSize+Ma
inSize,RawDataAlignment);

CodeSectionVirtualSize:=RoundSize(CodeSection.SizeOfRawData,NtHeaders.OptionalHe
ader.SectionAlignment);
        if CodeSectionVirtualSize<HostSizeOfImage then
CodeSectionVirtualSize:=RoundSize(HostSizeOfImage,NtHeaders.OptionalHeader.Secti
onAlignment);
        CodeSection.Misc.VirtualSize:=CodeSectionVirtualSize;

        NtHeaders.OptionalHeader.SizeOfCode:=CodeSection.SizeOfRawData;

        CodeSection.Characteristics:=IMAGE_SCN_CNT_CODE or
IMAGE_SCN_MEM_EXECUTE or IMAGE_SCN_MEM_WRITE or IMAGE_SCN_MEM_READ;

        frmMain.AddLog('.text блок, віртуальна адреса:
'+IntToHex(CodeSection.VirtualAddress,8),0);
        frmMain.AddLog('.text блок, віртуальний розмір:
'+IntToHex(CodeSection.Misc.VirtualSize,8),0);

        frmMain.AddLog('Будуємо .idata блок,0);


NtHeaders.OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_IMPORT].VirtualAddr
ess:=CodeSection.VirtualAddress+CodeSection.Misc.VirtualSize;       //може
змінюватися

        ZeroMemory(@ImportSection,SizeOf(ImportSection));

ImportSection.VirtualAddress:=NtHeaders.OptionalHeader.DataDirectory[IMAGE_DIREC
TORY_ENTRY_IMPORT].VirtualAddress;
        ImportSectionData:=MyAlloc(HostImportSectionSize+$70);
        ZeroMemory(ImportSectionData,HostImportSectionSize+$70);
        ImportSectionDLLCount:=1;

        if (HostImportSectionSize<>0) and (ImageType=itDLL) and not DynamicDLL
then
        begin

PB:=VirtAddrToPhysAddr(Ptr,Pointer(HostImportSectionVirtualAddress+PImageNtHeade
rs(Ptr)^.OptionalHeader.ImageBase));
         Inc(PB,Cardinal(MainData));
         PImportDesc:=Pointer(PB);
```

```
          while not ((PImportDesc^.Characteristics=0) and
(PImportDesc^.cTimeDateStamp=0)
            and (PImportDesc^.cForwarderChain=0) and (PImportDesc^.cName=0)
            and (PImportDesc^.cFirstThunk=nil)) do
          begin

PB2:=VirtAddrToPhysAddr(Ptr,Pointer(PImportDesc^.cName+PImageNtHeaders(Ptr)^.Opt
ionalHeader.ImageBase));
           Inc(PB2,Cardinal(MainData));
           if (UpperCase(PChar(PB2))<>UpperCase(Kernel32Name))
            and (UpperCase(PChar(PB2))<>UpperCase(NtdllName)) then
Inc(ImportSectionDLLCount);
           Inc(PImportDesc);
          end;
         end;


PB:=VirtAddrToPhysAddr(Ptr,Pointer(HostImportSectionVirtualAddress+PImageNtHeade
rs(Ptr)^.OptionalHeader.ImageBase));
         Inc(PB,Cardinal(MainData));
         PImportDesc:=Pointer(PB);
         PB2:=ImportSectionData;
         ZeroMemory(@ImportDesc,SizeOf(ImportDesc));

ImportDesc.Characteristics:=ImportSection.VirtualAddress+(ImportSectionDLLCount+
1)*SizeOf(ImportDesc);

ImportDesc.cName:=ImportSection.VirtualAddress+(ImportSectionDLLCount+1)*SizeOf(
ImportDesc)+(NumberOfImports+1+2*(ImportSectionDLLCount-
1))*SizeOf(TImageThunkData)*2;

ImportDesc.cFirstThunk:=Pointer(ImportDesc.Characteristics+(NumberOfImports+1+2*
(ImportSectionDLLCount-1))*SizeOf(TImageThunkData));
         InitcodeThunk:=Cardinal(ImportDesc.cFirstThunk);

         CopyMemory(PB2,@ImportDesc,SizeOf(ImportDesc));
         Inc(PB2,SizeOf(ImportDesc));
         PB3:=ImportSectionData;
         Inc(PB3,(ImportSectionDLLCount+1)*SizeOf(ImportDesc));
         PB4:=ImportSectionData;
         Inc(PB4,ImportDesc.cName-ImportSection.VirtualAddress);
         CopyMemory(PB4,PChar(Kernel32Name),Kernel32Size);
         Inc(PB4,RoundSize(Kernel32Size+1,2));

         PCardinal(PB3)^:=Cardinal(PB4)-
Cardinal(ImportSectionData)+ImportSection.VirtualAddress-2;
         CopyMemory(PB4,PChar(GetProcAddressName),GetProcAddressSize);
         Inc(PB4,RoundSize(GetProcAddressSize+1,2));
         Inc(PB3,SizeOf(DWORD));
         PCardinal(PB3)^:=Cardinal(PB4)-
Cardinal(ImportSectionData)+ImportSection.VirtualAddress-2;
         CopyMemory(PB4,PChar(LoadLibraryName),LoadLibrarySize);
         Inc(PB4,RoundSize(LoadLibrarySize+1,2));
         Inc(PB3,SizeOf(DWORD));
         Inc(PB3,SizeOf(DWORD));


         if (HostImportSectionSize<>0) and (ImageType=itDLL) and not DynamicDLL
then
         for I:=2 to ImportSectionDLLCount do
         begin
          ZeroMemory(@ImportDesc,SizeOf(ImportDesc));
          ImportDesc.Characteristics:=Cardinal(PB3)-
Cardinal(ImportSectionData)+ImportSection.VirtualAddress;
          ImportDesc.cName:=Cardinal(PB4)-
Cardinal(ImportSectionData)+ImportSection.VirtualAddress;

ImportDesc.cFirstThunk:=Pointer(ImportDesc.Characteristics+(NumberOfImports+1+2*
(ImportSectionDLLCount-1))*SizeOf(TImageThunkData));
```

```
          CopyMemory(PB2,@ImportDesc,SizeOf(ImportDesc));
          Inc(PB2,SizeOf(ImportDesc));

          while True do
          begin

PB:=VirtAddrToPhysAddr(Ptr,Pointer(PImportDesc^.cName+PImageNtHeaders(Ptr)^.Opti
onalHeader.ImageBase));
           Inc(PB,Cardinal(MainData));
           if (UpperCase(PChar(PB))<>UpperCase(Kernel32Name))
            and (UpperCase(PChar(PB))<>UpperCase(NtdllName)) then Break;
           Inc(PImportDesc);
          end;
          AnyDWORD:=Length(PChar(PB));
          CopyMemory(PB4,PB,AnyDWORD);
          Inc(PB4,RoundSize(AnyDWORD+1,2));

PB:=VirtAddrToPhysAddr(Ptr,Pointer(Cardinal(PImportDesc^.cFirstThunk)+PImageNtHe
aders(Ptr)^.OptionalHeader.ImageBase));
          Inc(PB,Cardinal(MainData));
          if PCardinal(PB)^ and $80000000=0 then
          begin

PB:=VirtAddrToPhysAddr(Ptr,Pointer(PCardinal(PB)^+PImageNtHeaders(Ptr)^.Optional
Header.ImageBase));
           Inc(PB,Cardinal(MainData));
           Inc(PB,2);
           AnyDWORD:=Length(PChar(PB));
           PCardinal(PB3)^:=Cardinal(PB4)-
Cardinal(ImportSectionData)+ImportSection.VirtualAddress;
           Inc(PB4,2);
           CopyMemory(PB4,PB,AnyDWORD);
           Inc(PB4,RoundSize(AnyDWORD+1,2));
          end else PCardinal(PB3)^:=PCardinal(PB)^;
          Inc(PImportDesc);
          Inc(PB3,SizeOf(DWORD));
          Inc(PB3,SizeOf(DWORD));
         end;

         PB3:=ImportSectionData;
         Inc(PB3,(ImportSectionDLLCount+1)*SizeOf(ImportDesc));
         PB:=PB3;
         AnyDWORD:=(NumberOfImports+1+2*(ImportSectionDLLCount-
1))*SizeOf(TImageThunkData);
         Inc(PB,AnyDWORD);
         CopyMemory(PB,PB3,AnyDWORD);
         ImportSectionDataSize:=Cardinal(PB4)-Cardinal(ImportSectionData);

         CopyMemory(@ImportSection.Name,PChar('.idata'),6);

ImportSection.Misc.VirtualSize:=RoundSize(ImportSectionDataSize,NtHeaders.Option
alHeader.SectionAlignment);

ImportSection.SizeOfRawData:=RoundSize(ImportSectionDataSize,RawDataAlignment);

ImportSection.PointerToRawData:=CodeSection.PointerToRawData+CodeSection.SizeOfR
awData;

NtHeaders.OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_IMPORT].Size:=Impor
tSection.SizeOfRawData;
         ImportSection.Characteristics:=IMAGE_SCN_CNT_CODE or
IMAGE_SCN_CNT_INITIALIZED_DATA or IMAGE_SCN_MEM_WRITE or IMAGE_SCN_MEM_READ;


CurVirtAddr:=ImportSection.VirtualAddress+ImportSection.Misc.VirtualSize;
         CurRawData:=ImportSection.PointerToRawData+ImportSection.SizeOfRawData;

         frmMain.AddLog('.idata віртуальна адреса блоку:
'+IntToHex(ImportSection.VirtualAddress,8),0);
```

```
         frmMain.AddLog('.idata віртуальний розмір блоку:
'+IntToHex(ImportSection.Misc.VirtualSize,8),0);

         // .tls блок
         if TlsSectionPresent then
         begin
          frmMain.AddLog('Побудова  .tls блоку',0);

TlsCopy.Directory:=@PImageNtHeaders(Ptr)^.OptionalHeader.DataDirectory[IMAGE_DIR
ECTORY_ENTRY_TLS];

TlsCopy.SectionData:=RVA2RAW(Ptr,MainData,TlsCopy.Directory.VirtualAddress);
          if TlsCopy.SectionData<>nil then
          begin
           TlsCopy.RawDataLen:=TlsCopy.SectionData^.RawDataEnd-
TlsCopy.SectionData^.RawDataStart;
           TlsCopy.RawData:=MyAlloc(TlsCopy.RawDataLen);

           PB:=RVA2RAW(Ptr,MainData,TlsCopy.SectionData^.RawDataStart-
HostImageBase);
           if PB<>nil then CopyMemory(TlsCopy.RawData,PB,TlsCopy.RawDataLen)
           else ZeroMemory(TlsCopy.RawData,TlsCopy.RawDataLen);

           PB:=RVA2RAW(Ptr,MainData,TlsCopy.SectionData^.AddressOfCallbacks-
HostImageBase);
           if PB=nil then
           begin
            TlsCopy.CallbacksLen:=4;
            TlsCopy.Callbacks:=MyAlloc(TlsCopy.CallbacksLen);
            ZeroMemory(TlsCopy.Callbacks,TlsCopy.CallbacksLen);
           end else
           begin
            TlsCopy.CallbacksLen:=GetTlsCallbacksLen(PB);
            TlsCopy.Callbacks:=MyAlloc(TlsCopy.CallbacksLen);
            CopyMemory(TlsCopy.Callbacks,PB,TlsCopy.CallbacksLen);
           end;

           ZeroMemory(@TlsSection,SizeOf(TlsSection));
           CopyMemory(@TlsSection.Name,PChar('.tls'),4);
           TlsSection.VirtualAddress:=CurVirtAddr;
           TlsSection.PointerToRawData:=CurRawData;
           TlsSection.Characteristics:=IMAGE_SCN_MEM_WRITE or
IMAGE_SCN_MEM_READ;

           ZeroMemory(@TlsSectionData,SizeOf(TlsSectionData));

TlsSectionData.RawDataStart:=NtHeaders.OptionalHeader.ImageBase+TlsSection.Virtu
alAddress+RoundSize(SizeOf(TlsSectionData),$10);

TlsSectionData.RawDataEnd:=TlsSectionData.RawDataStart+TlsCopy.RawDataLen;

TlsSectionData.AddressOfCallbacks:=RoundSize(TlsSectionData.RawDataEnd,$10);

TlsSectionData.AddressOfIndex:=RoundSize(TlsSectionData.AddressOfCallbacks+TlsCo
py.CallbacksLen,$08);

           TlsSection.SizeOfRawData:=RoundSize(TlsSectionData.AddressOfIndex-
TlsSection.VirtualAddress-
NtHeaders.OptionalHeader.ImageBase+$10,RawDataAlignment);

TlsSection.Misc.VirtualSize:=RoundSize(TlsSection.SizeOfRawData,NtHeaders.Option
alHeader.SectionAlignment);


NtHeaders.OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_TLS].VirtualAddress
:=CurVirtAddr;       //може змінюватися

NtHeaders.OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_TLS].Size:=TlsSecti
on.SizeOfRawData;
```

```
           CurVirtAddr:=TlsSection.VirtualAddress+TlsSection.Misc.VirtualSize;
           CurRawData:=TlsSection.PointerToRawData+TlsSection.SizeOfRawData;
          end else TlsSectionPresent:=False;
          frmMain.AddLog('.tls віртуальна адреса блоку:
'+IntToHex(TlsSection.VirtualAddress,8),0);
          frmMain.AddLog('.tls віртуальний розмір блоку:
'+IntToHex(TlsSection.Misc.VirtualSize,8),0);
         end;

         if ExportSectionPresent then
         begin
          frmMain.AddLog('Побудова  .edata блоку',0);
          ZeroMemory(@ExportSection,SizeOf(ExportSection));
          CopyMemory(@ExportSection.Name,PChar('.edata'),6);

          ExportSection.VirtualAddress:=CurVirtAddr;
          ExportSection.PointerToRawData:=CurRawData;
          ExportSection.Characteristics:=IMAGE_SCN_MEM_WRITE or
IMAGE_SCN_MEM_READ;

ExportSection.SizeOfRawData:=RoundSize(ExportSectionSize,RawDataAlignment);

ExportSection.Misc.VirtualSize:=RoundSize(ExportSection.SizeOfRawData,NtHeaders.
OptionalHeader.SectionAlignment);


NtHeaders.OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT].VirtualAddr
ess:=CurVirtAddr;       //може змінюватися

NtHeaders.OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT].Size:=Expor
tSection.SizeOfRawData;


CurVirtAddr:=ExportSection.VirtualAddress+ExportSection.Misc.VirtualSize;

CurRawData:=ExportSection.PointerToRawData+ExportSection.SizeOfRawData;

          ExportData:=MyAlloc(ExportSection.Misc.VirtualSize);
          ZeroMemory(ExportData,ExportSection.Misc.VirtualSize);


PB:=VirtAddrToPhysAddr(Ptr,Pointer(PImageNtHeaders(Ptr)^.OptionalHeader.DataDire
ctory[IMAGE_DIRECTORY_ENTRY_EXPORT].VirtualAddress+PImageNtHeaders(Ptr)^.Optiona
lHeader.ImageBase));
          if PB<>nil then Inc(PB,Cardinal(MainData));
          CopyMemory(ExportData,PB,ExportSectionSize);

          //фіксуємо RVAs у блоку експорту у Export Directory Table

ExportNamePointerRVAOrg:=PExportDirectoryTable(ExportData)^.NamePointerRVA;

ExportAddressRVAOrg:=PExportDirectoryTable(ExportData)^.ExportAddressTableRVA;
          ExportRVADelta:=ExportSection.VirtualAddress-
HostExportSectionVirtualAddress;

PExportDirectoryTable(ExportData)^.NameRVA:=PExportDirectoryTable(ExportData)^.N
ameRVA+ExportRVADelta;

PExportDirectoryTable(ExportData)^.ExportAddressTableRVA:=PExportDirectoryTable(
ExportData)^.ExportAddressTableRVA+ExportRVADelta;

PExportDirectoryTable(ExportData)^.NamePointerRVA:=PExportDirectoryTable(ExportD
ata)^.NamePointerRVA+ExportRVADelta;

PExportDirectoryTable(ExportData)^.OrdinalTableRVA:=PExportDirectoryTable(Export
Data)^.OrdinalTableRVA+ExportRVADelta;

          //+ фіксуємо RVAs у Export Name Pointer Table
```

```
PB2:=VirtAddrToPhysAddr(Ptr,Pointer(ExportNamePointerRVAOrg+PImageNtHeaders(Ptr)
^.OptionalHeader.ImageBase));
          Dec(PB2,Cardinal(PB)-Cardinal(MainData));
          Inc(PB2,Cardinal(ExportData));
          for I:=0 to PExportDirectoryTable(ExportData)^.NumberOfNamePointers-1
do
          begin
           PCardinal(PB2)^:=PCardinal(PB2)^+ExportRVADelta;
           Inc(PB2,SizeOf(DWORD));
          end;
          frmMain.AddLog('Експортуєма віртуальна адреса блоку:
'+IntToHex(ExportSection.VirtualAddress,8),0);
          frmMain.AddLog('Експортуємий віртуальний розмір блоку:
'+IntToHex(ExportSection.Misc.VirtualSize,8),0);
         end;

         if ResourceSectionPresent then
         begin
          frmMain.AddLog('Побудова  .rsrc блоку',0);

          ZeroMemory(@ResourceSection,SizeOf(ResourceSection));
          CopyMemory(@ResourceSection.Name,PChar('.rsrc'),5);

          ResourceSection.VirtualAddress:=CurVirtAddr;
          PrepareResourceSectionData;
          ResourceSection.PointerToRawData:=CurRawData;
          ResourceSection.Characteristics:=IMAGE_SCN_MEM_READ;

ResourceSection.SizeOfRawData:=RoundSize(ResourceSectionSize,RawDataAlignment);

ResourceSection.Misc.VirtualSize:=RoundSize(ResourceSection.SizeOfRawData,NtHead
ers.OptionalHeader.SectionAlignment);


NtHeaders.OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_RESOURCE].VirtualAd
dress:=CurVirtAddr;

NtHeaders.OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_RESOURCE].Size:=Res
ourceSection.SizeOfRawData;


CurVirtAddr:=ResourceSection.VirtualAddress+ResourceSection.Misc.VirtualSize;

CurRawData:=ResourceSection.PointerToRawData+ResourceSection.SizeOfRawData;

          frmMain.AddLog('Ресурсна віртуальна адреса блоку:
'+IntToHex(ResourceSection.VirtualAddress,8),0);
          frmMain.AddLog('Ресурсний віртуальний розмір блоку:
'+IntToHex(ResourceSection.Misc.VirtualSize,8),0);
         end;

         NtHeaders.OptionalHeader.SizeOfImage:=CurVirtAddr;

         frmMain.AddLog('Побудова  дескріптора імпорту ...',0);

         ZeroMemory(@ImportDesc,SizeOf(ImportDesc));

ImportDesc.Characteristics:=ImportSection.VirtualAddress+(NumberOfDLL+1)*SizeOf(
ImportDesc);

ImportDesc.cName:=ImportSection.VirtualAddress+(NumberOfDLL+1)*SizeOf(ImportDesc
)+(NumberOfImports+1)*SizeOf(TImageThunkData)*2;

ImportDesc.cFirstThunk:=Pointer(ImportDesc.Characteristics+(NumberOfImports+1)*S
izeOf(TImageThunkData));

ThunkGetProcAddress.Ordinal:=ImportSection.VirtualAddress+(NumberOfDLL+1)*SizeOf
(ImportDesc)+(NumberOfImports+1)*SizeOf(TImageThunkData)*2+Kernel32Size+2;
```

```
ThunkLoadLibrary.Ordinal:=ThunkGetProcAddress.Ordinal+GetProcAddressSize+2+2;

         ZeroMemory(@NullDesc,SizeOf(NullDesc));

TotalFileSize:=RoundSize(CurRawData,NtHeaders.OptionalHeader.FileAlignment);
         if OverlayPresent then Inc(TotalFileSize,AfterImageOverlaysSize);

         frmMain.AddLog('Побудова  поліморфичної частини ...',0);
         TrashSize:=KeyPtr;

         frmMain.AddLog('Адреса ключа: '+IntToHex(KeyPtr,8),0);
         frmMain.AddLog('Редактуємо адресу: '+IntToHex(LoaderPtr,8),0);
         frmMain.AddLog('Розмір байтів доданого сміття:
'+IntToStr(TrashSize),0);
         frmMain.AddLog('Розмір байтів доданого сміття2:
'+IntToStr(Trash2Size),0);
         Trash:=MyAlloc(TrashSize);
         Trash2:=MyAlloc(Trash2Size);
         if (Trash<>nil) and (Trash2<>nil) then
         begin
          GenerateRandomBuffer(Trash,TrashSize);
          GenerateRandomBuffer(Trash2,Trash2Size);


NtHeaders.OptionalHeader.AddressOfEntryPoint:=CodeSection.VirtualAddress+LoaderP
tr+LoaderSize+Trash2Size;
          frmMain.AddLog(' Виконуєма точка входу:
'+IntToHex(NtHeaders.OptionalHeader.AddressOfEntryPoint,8),0);
          InitData:=MyAlloc(InitSize);
          if InitData<>nil then
          begin

VirtLoaderData:=NtHeaders.OptionalHeader.ImageBase+CodeSection.VirtualAddress+Lo
aderPtr;

VirtMainData:=NtHeaders.OptionalHeader.ImageBase+CodeSection.VirtualAddress+Load
erPtr+LoaderSize+Trash2Size+InitSize;

VirtKey:=NtHeaders.OptionalHeader.ImageBase+CodeSection.VirtualAddress+KeyPtr;

           //initiate DynLoader image info
           PB:=Pointer(Cardinal(LoaderData)+LoaderSize);
           while PCardinal(PB)^<>DYN_LOADER_END_MAGIC do Dec(PB);
//DYN_LOADER_END_MAGIC search
           Dec(PB,5);
           PCardinal(PB)^:=MainRealSize;
           Dec(PB,7);
           PCardinal(PB)^:=NtHeaders.OptionalHeader.AddressOfEntryPoint;
           Dec(PB,7);
           PCardinal(PB)^:=NtHeaders.OptionalHeader.SizeOfImage;
           Dec(PB,7);
           case ImageType of
            itExe:PCardinal(PB)^:=IMAGE_TYPE_EXE;
            itDLL:PCardinal(PB)^:=IMAGE_TYPE_DLL;
            itSys:PCardinal(PB)^:=IMAGE_TYPE_SYS;
            else PCardinal(PB)^:=IMAGE_TYPE_UNKNOWN;
           end;

           //фіксуємо точки у DynLoader
           //це 3 інструкції, які запам'ятовуються

LdrPtrCode:=NtHeaders.OptionalHeader.ImageBase+CodeSection.VirtualAddress;

LdrPtrThunk:=NtHeaders.OptionalHeader.ImageBase+Cardinal(InitcodeThunk);

           LdrPtr:=LoaderData;
           Inc(LdrPtr);
           PCardinal(LdrPtr)^:=LdrPtrThunk+4-LdrPtrCode;
```

```
            Inc(LdrPtr,5);
            PCardinal(LdrPtr)^:=LdrPtrThunk-LdrPtrCode;
            Inc(LdrPtr,5);
            PCardinal(LdrPtr)^:=VirtMainData-LdrPtrCode;

            DynCoder(LoaderData,LoaderSize,Key);

            frmMain.AddLog(' Генерація коду ініціалізації ...',0);


//GenerateInitCode(ACodePtr,AKeyPtr,AData1Ptr,ASize1,AData2Ptr,ASize2,ADynLoadAd
dr,AMainPtr,
            //                 AEntryPointAddr,AImpThunk:Cardinal);


GenerateInitCode(LdrPtrCode,KeyPtr,LoaderPtr,LoaderSize,LoaderPtr+LoaderSize+Tra
sh2Size+InitSize,

MainRealSize,NtHeaders.OptionalHeader.ImageBase+CodeSection.VirtualAddress+Loade
rPtr,

VirtMainData,NtHeaders.OptionalHeader.ImageBase+NtHeaders.OptionalHeader.Address
OfEntryPoint,
                            LdrPtrThunk);


FileHandle:=CreateFile(PChar(OutputFileName),GENERIC_WRITE,0,nil,CREATE_ALWAYS,F
ILE_ATTRIBUTE_NORMAL,0);
            if FileHandle<>INVALID_HANDLE_VALUE then
            begin
             SetFilePointer(FileHandle,TotalFileSize,nil,FILE_BEGIN);
             SetEndOfFile(FileHandle);
             if not Quiet then
             begin
              frmMain.AddLog(' Розмір нового блоку: '+IntToStr(TotalFileSize-
MainRealSize)+' bytes',0);
              LogCnt:=0;
              Inc(LogCnt,SizeOf(DosHeader));
              Inc(LogCnt,SizeOf(DosStub));
              Inc(LogCnt,SizeOf(DosStubEnd));
              Inc(LogCnt,SizeOf(NtHeaders));
              Inc(LogCnt,SizeOf(CodeSection));
              Inc(LogCnt,SizeOf(ImportSection));
              if ExportSectionPresent then
              begin
               Inc(LogCnt,SizeOf(ExportSection));
              end;

              LogCnt:=CodeSection.PointerToRawData;
              Inc(LogCnt,TrashSize);
              Inc(LogCnt,KeySize);
              Inc(LogCnt,LoaderSize);
              Inc(LogCnt,Trash2Size);
              Inc(LogCnt,InitSize);

              LogCnt:=ImportSection.PointerToRawData;
              if TlsSectionPresent then
              begin
               LogCnt:=TlsSection.PointerToRawData;
              end;
              if ExportSectionPresent then
              begin
               LogCnt:=ExportSection.PointerToRawData;
              end;
              if ResourceSectionPresent then
              begin
               LogCnt:=ResourceSection.PointerToRawData;
              end;
```

```
             if OverlayPresent then
             begin
              LogCnt:=TotalFileSize-AfterImageOverlaysSize;
             end;
            end;

            // заглушка
            SetFilePointer(FileHandle,0,nil,FILE_BEGIN);
            WriteFile(FileHandle,DosHeader,SizeOf(DosHeader),NumBytes,nil);
            WriteFile(FileHandle,DosStub,SizeOf(DosStub),NumBytes,nil);
            WriteFile(FileHandle,DosStubEnd,SizeOf(DosStubEnd),NumBytes,nil);
            WriteFile(FileHandle,NtHeaders,SizeOf(NtHeaders),NumBytes,nil);
            WriteFile(FileHandle,CodeSection,SizeOf(CodeSection),NumBytes,nil);

WriteFile(FileHandle,ImportSection,SizeOf(ImportSection),NumBytes,nil);
            if TlsSectionPresent then
WriteFile(FileHandle,TlsSection,SizeOf(TlsSection),NumBytes,nil);
            if ExportSectionPresent then
WriteFile(FileHandle,ExportSection,SizeOf(ExportSection),NumBytes,nil);
            if ResourceSectionPresent then
WriteFile(FileHandle,ResourceSection,SizeOf(ResourceSection),NumBytes,nil);


SetFilePointer(FileHandle,ImportSection.PointerToRawData,nil,FILE_BEGIN);

WriteFile(FileHandle,ImportSectionData^,ImportSection.SizeOfRawData,NumBytes,nil
);

            // Блок коду

SetFilePointer(FileHandle,CodeSection.PointerToRawData,nil,FILE_BEGIN);

            // Переходний блок імпорту, який переміщується у кінець коду
ініціалізації
            WriteFile(FileHandle,Trash^,TrashSize,NumBytes,nil);
            WriteFile(FileHandle,Key^,KeySize,NumBytes,nil);
            WriteFile(FileHandle,LoaderData^,LoaderSize,NumBytes,nil);
            WriteFile(FileHandle,Trash2^,Trash2Size,NumBytes,nil);
            WriteFile(FileHandle,InitData^,InitSize,NumBytes,nil);

            // Блок даних
            WriteFile(FileHandle,MainDataCyp^,MainSize,NumBytes,nil);

            // Tls блок
            if TlsSectionPresent then
            begin

SetFilePointer(FileHandle,TlsSection.PointerToRawData,nil,FILE_BEGIN);

WriteFile(FileHandle,TlsSectionData,SizeOf(TlsSectionData),NumBytes,nil);

             Delta:=TlsSectionData.RawDataStart-TlsSection.VirtualAddress-
NtHeaders.OptionalHeader.ImageBase;

SetFilePointer(FileHandle,TlsSection.PointerToRawData+Delta,nil,FILE_BEGIN);

WriteFile(FileHandle,TlsCopy.RawData^,TlsCopy.RawDataLen,NumBytes,nil);

             Delta:=TlsSectionData.AddressOfCallbacks-TlsSection.VirtualAddress-
NtHeaders.OptionalHeader.ImageBase;

SetFilePointer(FileHandle,TlsSection.PointerToRawData+Delta,nil,FILE_BEGIN);

WriteFile(FileHandle,TlsCopy.Callbacks^,TlsCopy.CallbacksLen,NumBytes,nil);
            end;

            // Блок експорту
            if ExportSectionPresent then
            begin
```

```
SetFilePointer(FileHandle,ExportSection.PointerToRawData,nil,FILE_BEGIN);

WriteFile(FileHandle,ExportData^,ExportSection.SizeOfRawData,NumBytes,nil);
             if ExportData<>nil then MyFree(ExportData);
            end;

            //  блок ресурсу
            if ResourceSectionPresent then
            begin

SetFilePointer(FileHandle,ResourceSection.PointerToRawData,nil,FILE_BEGIN);

WriteFile(FileHandle,ResourceData^,ResourceSection.SizeOfRawData,NumBytes,nil);
             if ResourceData<>nil then MyFree(ResourceData);
            end;

            // Оверлейні дані
            if OverlayPresent then
            begin
             SetFilePointer(FileHandle,TotalFileSize-
AfterImageOverlaysSize,nil,FILE_BEGIN);

WriteFile(FileHandle,AfterImageOverlays^,AfterImageOverlaysSize,NumBytes,nil);
            end;

            frmMain.AddLog('Файл успішно захищено',0);
            frmMain.StatusBar1.Panels.Items[0].Text:='Файл успішно захищено';
            CloseHandle(FileHandle);
           end else ErrorMsg('Неможливо створити вихідний файл.');
           MyFree(InitData);
          end else ErrorMsg('Неможливо виділити пам’ять під ініціалізаційні
дані');
          MyFree(Trash);
          MyFree(Trash2);
         end else ErrorMsg('Неможливо виділити пам’ять під дані байтів
засмітчення.');
         if TlsSectionPresent then
         begin
          MyFree(TlsCopy.RawData);
          MyFree(TlsCopy.Callbacks);
         end;
         MyFree(Key);

         if ImportSectionData<>nil then MyFree(ImportSectionData);
        end else ErrorMsg('Неможливо виділити пам’ять  під ключ кодування.');
        MyFree(LoaderData);
       end else ErrorMsg('Неможливо виділити пам’ять  для редагування.');
      end else ErrorMsg('Неможливо генерувати кодер/декодер');
     end else ErrorMsg('Підсистема не підтримується .');
    end else ErrorMsg(' Вхідний файл є не валідним PE файлом.');
   end else ErrorMsg('Неможливо прочитати файл.');
   MyFree(MainData);
  end else ErrorMsg('Неможливо виділити пам’ять  для даних вхідного файлу.');
  if MainFile<>INVALID_HANDLE_VALUE then CloseHandle(MainFile);
 end else ErrorMsg('Неможливо відкрити файл.');
end;
end.
```

Файл maincode.pas – основна програма

```
unit maincode;

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

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

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

var
  frmMain: TfrmMain;

const
  packer_ver:string='';

implementation

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

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


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

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

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

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

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

CONST
MAX_SECTION_NUMBER= $10;

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


    var
    hFile:DWORD;
    e_lfanew:DWORD;
    EXE:WORD;
    i:integer;
    bread:dword;
    EPreal, EP, imagebase,nv,ns,fa,sa:cardinal;
```

```
    num,epsec:integer;
    pe:pe_file;
    PACKEDSECTION:dword;
    PACKEDPOS:pointer;
    temp1:pointer;
    temp2:pointer;

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

Function RVA2Offset(RVA:DWORD):DWORD;
 var i:integer;
     VirtAddr,VA2,szRawData,ptrRawData:DWORD;
 begin
  for i:=1 to PE_HEADER.FILE_HEADER.NumberOfSections do
   begin
    VirtAddr:=SECTION_HEADER[i].VirtualAddress;
    szRawData:=SECTION_HEADER[i].SizeOfRawData;
    ptrRawData:=SECTION_HEADER[i].PointerToRawData;
    if RVA>=VirtAddr then
     begin
      VA2:=VirtAddr+szRawData;
      if RVA<VA2 then
       begin
        RVA:=RVA-VirtAddr;
        RVA:=RVA+ptrRawData;
       end;
     end;
   end;
  RVA2Offset:=RVA;
 end;
//Функція визначення розміру блоку заплутування
function GetLoaderSize(Func:dword):dword;
begin
asm
pushad
mov eax, func
mov esi, 1
@find:
mov dword ptr ebx, [eax]
mov dword ptr ecx, [eax+4]
cmp ebx, $41504544
jnz @notf
cmp ecx, $4E454B43
jnz @notf
mov result, esi
jmp @exit
@notf:
inc esi
inc eax
jmp @find
@exit:
popad
end;
end;
// процедура таблиць підстановки при заплутуванні коду
procedure ImportTable;
begin
{
0045F6A4  E8 F6 05 00 00 00 00 00 00 00 00 00 F8 F6 05 00  иц▯ .........шц▯ .
0045F6B4  E8 F6 05 00 E0 F6 05 00 00 00 00 00 00 00 00 00  иц▯ .ац▯ .........
0045F6C4  05 F7 05 00 E0 F6 05 00 00 00 00 00 00 00 00 00  ▯ч▯.ац▯ .........
0045F6D4  00 00 00 00 00 00 00 00 00 00 00 00 58 12 DA 77  ............X▯Ъw
0045F6E4  00 00 00 00 79 BB E6 77 1F A0 E6 77 C4 EF ED 77  ....y»жw жwДпнw
```

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

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

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

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

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

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

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

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

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

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

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

iat[85]:=byte('K');
iat[86]:=byte('E');
iat[87]:=byte('R');
iat[88]:=byte('N');
iat[89]:=byte('E');
iat[90]:=byte('L');
iat[91]:=byte('3');
iat[92]:=byte('2');
iat[93]:=byte('.');
iat[94]:=byte('D');
```

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

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

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

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

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

iat[162]:=byte('M');
iat[163]:=byte('e');
iat[164]:=byte('s');
iat[165]:=byte('s');
iat[166]:=byte('a');
iat[167]:=byte('g');
iat[168]:=byte('e');
iat[169]:=byte('B');
iat[170]:=byte('o');
iat[171]:=byte('x');
```

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

end;

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

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

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

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

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

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

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

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

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


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

    cld
    mov    dl, 80h
    xor    ebx, ebx

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

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

@shortmatch:
    lodsb
    shr    eax, 1
    jz     @donedepacking
```

```
    adc    ecx, ecx
    jmp    @domatch_with_2inc

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

@domatch_with_2inc:
    inc    ecx

@domatch_with_inc:
    inc    ecx

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

    mov    bl, 1

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

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

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

```
// Імпортуємо відновлення

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

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

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

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

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

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

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

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

  if frmMain.aPLib.Length = 0 then Exit;

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

result:=frmMain.aPLib.Destination;
end;

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

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

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

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

/// Упаковник коду


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

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


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

   num:=PE_HEADER.FILE_HEADER.NumberOfSections;
```

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

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

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


  for i:=1 to PE_HEADER.FILE_HEADER.NumberOfSections do
   begin
   if (ep>=SECTION_HEADER[i].PointerToRawData)and
   (ep<(SECTION_HEADER[i].SizeOfRawData+SECTION_HEADER[i].PointerToRawData))
then begin
   epsec:=i; break;
   end;
   end;

   DEPBEGIN:=nv+imagebase;
   sizeofsec:=SECTION_HEADER[1].SizeOfRawData;


iatrva:=PE_HEADER.OPTIONAL_HEADER.IMAGE_DIRECTORY_ENTRIES.IMPORT.VirtualAddress;

// Записуємо редактуємі дані
stra:='1234';

InsertString(@PE_Loader,'GlobalAlloc',179);
InsertString(@PE_Loader,'GlobalFree',191);
// Записуємо число секторів
// Push Kernel32
asm
mov eax, stra
mov ebx, DEPBEGIN
add ebx, $54
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$101);
// mov eax, GetModuleHandleA
asm
mov eax, stra
mov ebx, DEPBEGIN
add ebx, $48
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$106);

asm
mov eax, stra
mov ebx, DEPBEGIN
add ebx, 179
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$10D);
```

```
asm
mov eax, stra
mov ebx, DEPBEGIN
add ebx, $44
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$113);

// завантажуємо розмір  секторів
asm
mov eax, stra
mov ebx, sizeofsec
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$11A);
// зберігаємо адресу  globalallocess
asm
mov eax, stra
mov ebx, DEPBEGIN
add ebx, 202
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$124);
// mov edi, блок_HEADER[1].PointerToRawData
addrsec:=SECTION_HEADER[1].VirtualAddress+imagebase;
asm
mov eax, stra
mov ebx, addrsec
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$12B);
// mov ecx, sizeofsec
asm
mov eax, stra
mov ebx, sizeofsec
sub ebx, 4
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$1C8);

// loop addr
asm
mov eax, stra
mov ebx, addrsec
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$1D1);

// записуємо imagebase
asm
mov eax, stra
mov ebx, imagebase
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$1DA);
// заданий  iat rva
asm
mov eax, stra
mov ebx, iatrva
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$1DF);
// mov eax, GetModuleHandle
asm
mov eax, stra
mov ebx, DEPBEGIN
add ebx, $48
mov [eax], ebx
```

```
end;
InsertBytes(@PE_Loader,stra,4,$1F6);
// LoadLibraryA
asm
mov eax, stra
mov ebx, DEPBEGIN
add ebx, $4C
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$202);

// mov робота  з буфером 1
asm
mov eax, stra
mov ebx, DEPBEGIN
add ebx, 206
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$20A);

// mov робота  з буфером 2
asm
mov eax, stra
mov ebx, DEPBEGIN
add ebx, 210
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$210);
// записуємо imagebase
asm
mov eax, stra
mov ebx, imagebase
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$219);
InsertBytes(@PE_Loader,stra,4,$26E);
// робота  з буфером 2
asm
mov eax, stra
mov ebx, DEPBEGIN
add ebx, 210
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$22A);
InsertBytes(@PE_Loader,stra,4,$237);
InsertBytes(@PE_Loader,stra,4,$263);

// mov робота  з буфером 1
asm
mov eax, stra
mov ebx, DEPBEGIN
add ebx, 206
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$254);
// GetProcAddress
asm
mov eax, stra
mov ebx, DEPBEGIN
add ebx, $44
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$259);
//======================

// Push Kernel32
asm
mov eax, stra
mov ebx, DEPBEGIN
```

```
add ebx, $54
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$278);
// mov eax, GetModuleHandleA
asm
mov eax, stra
mov ebx, DEPBEGIN
add ebx, $48
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$27d);

asm
mov eax, stra
mov ebx, DEPBEGIN
add ebx, 191
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$284);

asm
mov eax, stra
mov ebx, DEPBEGIN
add ebx, $44
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$28A);

asm
mov eax, stra
mov ebx, DEPBEGIN
add ebx, 202
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$292);

// OEP
asm
mov eax, stra
mov ebx, epreal
mov [eax], ebx
end;
InsertBytes(@PE_Loader,stra,4,$29B);


ImportTable;
//Записуємо  IAT до депакувальника
WriteProcessMemory(GetCurrentProcess(),@PE_Loader,@iat,sizeof(iat),wr);


// Депакування
   temp1:=pointer(GlobalAlloc(GMEM_ZEROINIT,SECTION_HEADER[1].SizeOfRawData));
   temp2:=pointer(GlobalAlloc(GMEM_ZEROINIT,SECTION_HEADER[1].SizeOfRawData));

   SetFilePointer(hFile,SECTION_HEADER[1].PointerToRawData,NIL,FILE_BEGIN);
   ReadFile(hFile,temp1^,SECTION_HEADER[1].SizeOfRawData,bread,NIL);

   PACKEDPOS:= PackSection(pointer(temp1),SECTION_HEADER[1].SizeOfRawData);

// Очищуємо блок
   SetFilePointer(hFile,SECTION_HEADER[1].PointerToRawData,NIL,FILE_BEGIN);
   WriteFile(hFile,temp2^,SECTION_HEADER[1].SizeOfRawData,bread,NIL);
   GlobalFree(cardinal(temp2));

   SetFilePointer(hFile,SECTION_HEADER[1].PointerToRawData,NIL,FILE_BEGIN);
   WriteFile(hFile,PACKEDPOS^,PACKEDSECTION,bread,NIL);
    GlobalFree(cardinal(temp1));
```

```
    // закінчуємо запис упакованих блоків
PE_HEADER.OPTIONAL_HEADER.AddressOfEntryPoint:=nv+$0FF;
PE_HEADER.OPTIONAL_HEADER.IMAGE_DIRECTORY_ENTRIES.IMPORT.VirtualAddress:=nv;
PE_HEADER.OPTIONAL_HEADER.IMAGE_DIRECTORY_ENTRIES.IMPORT.Size:=$b1;

   SetFilePointer(hFile,ns,NIL,FILE_BEGIN);
   temp2:=@PE_Loader;
   WriteFile(hFile,temp2^,GetLoaderSize(dword(@PE_Loader)),bread,NIL);

   for i:=1 to PE_HEADER.FILE_HEADER.NumberOfSections do
   блок_HEADER[i].Characteristics:=$E00000E0;

   SetFilePointer(hFile,e_lfanew,NIL,FILE_BEGIN);
   WriteFile(hFile,PE_HEADER,SizeOF(PE_HEADER),bread,NIL);
   for i:=1 to PE_HEADER.FILE_HEADER.NumberOfSections do
   WriteFile(hFile,SECTION_HEADER[i],SizeOF(Section),bread,NIL);

   CloseHandle(hFile);
   frmMain.StatusBar1.Panels.Items[0].Text:='Оптимізація...';
   AddLog('Оптимізація...',0);

   pe:=pe_file.Create;
   pe.LoadFromFile(InputFileName);
   pe.OptimizeHeader(true);
   pe.OptimizeFileAlignment;
   pe.FlushFileCheckSum;
   pe.OptimizeFile(true,true,true,false);
   pe.SaveToFile(InputFileName);
   pe.Free;
   AddLog('Файл успішно упаковано',0);

   frmMain.StatusBar1.Panels.Items[0].Text:='Файл успішно упаковано';
end;

procedure TfrmMain.cmdProtectClick(Sender: TObject);
begin
  ClearLog;
  try

CopyFile(pchar(txtFileName.Text),pchar(copy(txtFileName.Text,1,Length(txtFileNam
e.Text)-4)+'.bak'),false)
  except
  end;
  frmMain.StatusBar1.Panels.Items[0].Text:='Робота';
  if (rbPacking.Checked or rbFullProtect.Checked) then Pack(txtFileName.Text);
  if (rbProtect.Checked or rbFullProtect.Checked) then begin
    frmMain.StatusBar1.Panels.Items[0].Text:='Захищено....';
    Protect(txtFileName.Text);
  end;
  frmMain.StatusBar1.Panels.Items[0].Text:='Готово';
  cmdTest.Enabled:=True;
end;

/////// Формування програмного інтерфейсу користувача //////////

procedure TfrmMain.AddLog(sText: string; sImage: integer);
begin
  with lstStatus.Items.Add do begin
    Caption:=sText;
    ImageIndex:=sImage;
  end;
end;

procedure TfrmMain.ClearLog;
begin
  lstStatus.Items.Clear;
end;

procedure TfrmMain.cmdOpenClick(Sender: TObject);
```

```
begin
  frmMain.dlgOpen.Title:='Відкрити EXE файл';
  frmMain.dlgOpen.Filter:='EXE файли (*.exe)|*.exe';
  if frmMain.dlgOpen.Execute then begin
    frmMain.txtFileName.Text:=frmMain.dlgOpen.FileName;
    cmdTest.Enabled:=False;
  end;
end;

procedure TfrmMain.FormCreate(Sender: TObject);
begin
  frmMain.lstStatus.Columns[0].Width:=frmMain.lstStatus.Width-25;
end;

procedure TfrmMain.mnuNewClick(Sender: TObject);
begin
  if (MessageDlg('Ви бажаєте встановити весь пакет програмного
забезпечення?',mtConfirmation,[mbYes, mbNo],0)=mrYes) then begin
    txtFileName.Text:='';
    rbFullProtect.Checked:=True;
    cbIcon.Checked:=True;
    cbOverlay.Checked:=True;
    txtImageBase.Text:='0';
  end;
end;

procedure TfrmMain.mnuOpenClick(Sender: TObject);
var
  strProject: ProtectorProject;
  f: file of ProtectorProject;
begin
  frmMain.dlgOpen.Title:='Open GHF проект';
  frmMain.dlgOpen.Filter:='GHF проекты (*.ghf)|*.ghf';
  if frmMain.dlgOpen.Execute then begin
    AssignFile(f,frmMain.dlgOpen.FileName);
    Reset(f);
    Read(f,strProject);
    CloseFile(f);
    txtFileName.Text:=strProject.sFileName;
    if strProject.sPacking then rbPacking.Checked:=True;
    if strProject.sProtection then rbProtect.Checked:=True;
    if (strProject.sPacking and strProject.sProtection) then
rbFullProtect.Checked:=True;
    cbIcon.Checked:=strProject.sSaveIcon;
    cbOverlay.Checked:=strProject.sSaveOverlay;
    txtImageBase.Text:=strProject.sImageBase;
    tabSheet.ActivePageIndex:=strProject.sActivePageIndex;
  end;
end;

procedure TfrmMain.mnuSaveClick(Sender: TObject);
var
  strProject: ProtectorProject;
  f: file of ProtectorProject;
begin
  frmMain.dlgSave.Title:='Зберегти GHF проект';
  frmMain.dlgSave.Filter:='GHF проекти (*.ghf)|*.ghf';
  if frmMain.dlgSave.Execute then begin
    if (ExtractFileExt(frmMain.dlgSave.FileName)='') then
frmMain.dlgSave.FileName:=frmMain.dlgSave.FileName+'.ghf';
    strProject.sFileName:=txtFileName.Text;
    if rbPacking.Checked then strProject.sPacking:=True;
    if rbProtect.Checked then strProject.sProtection:=True;
    if rbFullProtect.Checked then begin
       strProject.sProtection:=True;
       strProject.sPacking:=True;
    end;
    strProject.sSaveIcon:=cbIcon.Checked;
    strProject.sSaveOverlay:=cbOverlay.Checked;
```

```
    strProject.sImageBase:=txtImageBase.Text;
    strProject.sActivePageIndex:=tabSheet.ActivePageIndex;

    AssignFile(f,frmMain.dlgSave.FileName);
    Rewrite(f);
    Write(f,strProject);
    CloseFile(f);
  end;
end;

procedure TfrmMain.mnuExitClick(Sender: TObject);
begin
  halt;
end;

procedure TfrmMain.cmdTestClick(Sender: TObject);
begin

ShellExecute(frmMain.Handle,'open',pchar(txtFileName.Text),'',pchar(ExtractFileD
ir(txtFileName.Text)),0);
end;

procedure TfrmMain.mnuAboutClick(Sender: TObject);
begin
  frmAbout.ShowModal;
end;

procedure TfrmMain.mnuHelpOpenClick(Sender: TObject);
begin
  if not (ShellExecute(frmMain.Handle,'відкрито',pchar(Application.HelpFile),
  pchar(''),pchar(ExtractFilePath(ParamStr(0))),1)=42) then
  ShowMessage('File "'+Application.HelpFile+'" не знайдено у програмній
директорії');
end;

end.
```