

Міністерство освіти і науки України
Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки

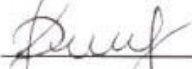
ПОЯСНЮВАЛЬНА ЗАПИСКА

до випускної кваліфікаційної роботи бакалавра
на тему: «Проект підсистеми аналізу обфускованого
коду з використанням інструментів платформи .NET»

Виконав: студент 4-го курсу групи КБ-21-2
спеціальності 125 «Кібербезпека»

 Дмитро ПЕТРЕНКО

Керівник: Старший викладач кафедри КІ та КБ

 Єлизавета ТРОКОЗ

Рецензент: Завідувач кафедри КН, PhD, доцент

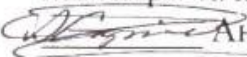
 Марина ГРАФ

Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки
Спеціальність 125 «Кібербезпека»
Освітня програма «Кібербезпека»

ЗАТВЕРДЖУЮ

Завідувач

кафедри комп'ютерної
інженерії та кібербезпеки

 Андрій Єфіменко
3 березня 2025 р.

ЗАВДАННЯ

на випускню кваліфікаційну роботу бакалавра

Студент: Петренко Дмитро Володимирович

Тема роботи: «Проект підсистеми аналізу обфускованого коду з використанням інструментів платформи .NET»

затверджена Наказом університету від 3 березня 2025 р. № 80/с

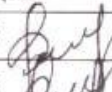
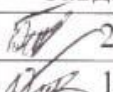
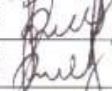
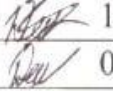
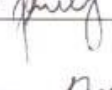
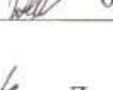
Термін здачі студентом закінченої роботи 20 червня 2025 р.

Вихідні дані роботи (зазначається предмет і об'єкт дослідження):
Об'єктом дослідження є процес обфускації коду у середовищі CLR та процес розробки програмного забезпечення для його аналізу. Предметом дослідження є технології обфускації з приховуванням методів та технічні підходи до аналізу таких механізмів захисту.

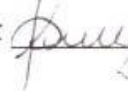
Обладнання та програмне забезпечення:

1. Visual studio
2. dnSpy
3. ExtremeDumper
4. Xenos

Консультанти з випускної кваліфікаційної роботи із зазначенням розділів, що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Трокоз Є.М.	29.03.2025 	29.03.2025 
2	Трокоз Є.М.	14.04.2025 	14.04.2025 
3	Трокоз Є.М.	07.05.2025 	07.05.2025 

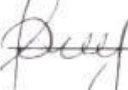
Студент  Дмитро ПЕТРЕНКО

Керівник  Єлизавета ТРОКОЗ

Календарний план

№ з/п	Назви етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка завдання	03.05.2025	Виконано
2	Аналіз механізмів CLR	04.03.2025 - 03.03.2025	Виконано
3	Дослідження .NET обфускації	04.03.2025 - 21.03.2025	Виконано
4	Дослідження структури програми	24.03.2025 - 29.03.2025	Виконано
5	Створення бібліотеки-інжектора	30.03.2025 - 16.04.2025	Виконано
6	Створення бібліотеки-аналізатора	17.04.2025 - 02.05.2025	Виконано
7	Доопрацювання роботи	03.05.2025 - 11.05.2025	Виконано
8	Написання пояснювальної записки	12.05.2025 - 08.06.2025	Виконано
9	Отримання відгуку та рецензії	11.06.2025	Виконано
10	Захист	20.06.2025	Виконано

Студент  Дмитро ПЕТРЕНКО

Керівник  Єлизавета ТРОКОЗ

АНОТАЦІЯ

Кваліфікаційна робота на тему «Проект підсистеми аналізу обфускованого коду з використанням інструментів платформи .NET» включає вступ, три розділи, висновки, список використаних джерел та додатки.

Текстова частина займає 44 сторінках друкованого тексту.

До пояснювальної записки входить список із 19 використаних джерел, що займають 2 сторінки. У роботі представлено 52 рисунків, загальний обсяг – 55 сторінки.

Завданням кваліфікаційної роботи є:

- аналіз механізмів роботи середовища CLR і процесу JIT-компіляції;
- дослідження методів обфускації коду в екосистемі .NET;
- вивчення структури обфускованого додатка, що аналізується;
- створення бібліотеки-інжектора на мові C++;
- розробка C# бібліотеки для перехоплення JIT-компіляції;
- демонстрація роботи системи на прикладі конкретного додатка.

Перший розділ містить теоретичні відомості щодо принципів роботи CLR, вразливостей CLR Injection та основних способів обфускації коду в .NET.

У другому розділі було проаналізовано структуру додатку, який ми аналізуємо та розроблено бібліотеки на C++ та C#.

У третьому розділі наведено результати перехоплення та відслідковування виконання методів, що компілюються під час роботи додатка.

Ключові слова: ОБФУСКАЦІЯ, CLR, JIT, COMPILEMMETHOD, ПЕРЕХОПЛЕННЯ МЕТОДУ.

					ККІК.КР.Б – 125 – 16 – ПЗ			
Зм.	Арк.	№ докум.	Підпис	Дата	Проект підсистеми аналізу обфускованого коду з використанням інструментів платформи .NET	Літ.	Арк.	Аркушів
Розроб.		Петренко Д.В.		11.06.25				
Керівник		Трокоз С.М.		11.06.25			4	44
Реценз.		Граф М.С.		11.06.25		ФІКТ, Гр. КБ-21-2		
Н. контр.		Трокоз С.М.		11.06.25				
Затверд.		Сфіменко А.І.		11.06.25				

ANNOTATION

The qualification work on the topic "Project of a subsystem for analyzing obfuscated code using the .NET platform tools" includes an introduction, three chapters, conclusions, a list of references, and appendices.

The text portion consists of 44 pages of printed text.

The explanatory note includes a list of 19 sources, spanning 2 pages. The work contains 52 figures, with a total volume of 55 pages.

The tasks of the qualification work are:

- Analysis of the CLR environment and the JIT compilation process;
- Research of code obfuscation methods in the .NET ecosystem;
- Study of the structure of the analyzed obfuscated application;
- Creation of an injector library in C++;
- Development of a C# library for intercepting JIT compilation;
- Demonstration of the system's operation using a specific application example.

The first chapter contains theoretical information about the principles of CLR operation, vulnerabilities of CLR Injection, and the main methods of code obfuscation in .NET.

The second chapter analyzes the structure of the application under analysis and describes the development of the C++ and C# libraries.

The third chapter presents the results of intercepting and tracking the execution of methods compiled during the application's execution.

Keywords: OBFUSCATION, CLR, JIT, COMPILEMETHOD, METHOD INTERCEPTION.

		Петренко Д.В.			ККІК.КР.Б – 125 – 16 – ПЗ	Арк.
		Трокоз Є.М.				5
Змн.	Арк.	№ докум.	Підпис	Дата		

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ	7
ВСТУП.....	8
РОЗДІЛ 1 ТЕОРЕТИЧНІ ОСНОВИ РОБОТИ CLR І ОБФУСКАЦІЇ	9
1.1 Огляд Common Language Runtime.....	9
1.2 JIT-компіляція в CLR.....	12
1.3 CLR Injection: Вразливість платформи .NET	15
1.4 Обфускація у контексті CLR.....	16
1.5 NetReactor і підміна compileMethod для приховування методів	18
РОЗДІЛ 2 РОЗРОБКА ІНСТРУМЕНТІВ ДЛЯ АНАЛІЗУ ОБФУСКОВАНОГО КОДУ	20
2.1 Дослідження структури додатку	20
2.2 Проєктування загального алгоритму програми	25
2.3 Розробка CLR Injector та завантаження бібліотеки-аналізатора.....	27
2.4 Розробка аналізатора методів	33
РОЗДІЛ 3 ПІДГОТОВКА СЕРЕДОВИЩА ТА ТЕСТУВАННЯ РОБОТИ ПРОГРАМИ	36
3.1 Підготовка середовища	36
3.2 Демонстрація роботи програми та отримання байтів методів	39
ВИСНОВКИ.....	41
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	42
ДОДАТКИ.....	44

		Петренко Д.В.			ККІК.КР.Б – 125 – 16 – ПЗ	Арк.
		Трокоз Є.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		6

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

CLR – Common Language Runtime

JIT – Just-in-Time

MSIL – Microsoft Intermediate Language

IL – Intermediate Language

CIL – Common Intermediate Language

ОС – Операційна система

CLS – Common Language Specification

CTS – Common Type System

DLL – Dynamic-link library

EXE – Executable

ПЗ – Програмне забезпечення

		Петренко Д.В.			ККІК.КР.Б – 125 – 16 – ПЗ	Арк.
		Трокоз Є.М.				7
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Актуальність теми. З розвитком технологій інформаційної безпеки та захисту даних важливим стає питання захисту програмного коду від аналізу й зловживань. Платформа .NET з її середовищем виконання CLR забезпечує високу продуктивність і гнучкість у розробці ПЗ, але разом із цим створює ризики, пов'язані з легкістю реверс-інжинірингу її програм. Це робить обфускацію коду доречним елементом для запобігання несанкціонованому доступу до програм.

Технологія обфускації використовує унікальні підходи до приховування логіки програмного забезпечення. Це сприяє покращенню безпеки ПЗ та підвищенню стійкості до реверс-інжинірингу. Дослідження цих підходів захисту дозволить оцінити ефективність механізмів обфускації, виявляти вразливості у ПЗ, знаходити небезпечний код, та інші потенційні загрози.

Метою кваліфікаційної роботи є аналіз платформи .NET, механізмів обфускації та проєктування інструментів для дослідження обфускованого коду.

Визначена мета обумовлює наступні завдання:

- дослідити середовище виконання CLR та механізми JIT-компіляції.
- розглянути принципи роботи та алгоритми обфускації коду.
- розробити бібліотеку-інжектор на C++ для взаємодії з середовищем CLR та впровадження власного коду.
- розробити бібліотеку-аналізатор на C#, що дозволяє перехоплювати компіляцію методів та зчитувати байти оригінальних інструкцій.

Об'єктом дослідження є процес обфускації коду у середовищі CLR та процес розробки програмного забезпечення для його аналізу.

Предметом дослідження є технологія обфускації з приховуванням методів та технічні підходи до аналізу таких механізмів захисту.

У процесі виконання кваліфікаційної роботи було розроблено проєкт у вигляді двох бібліотек, які дозволяють виконувати перехоплення байтів оригінальних методів обфускованого додатку.

		Петренко Д.В.			ККІК.КР.Б – 125 – 16 – ПЗ	Арк.
		Трокоз Є.М.				8
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 1 ТЕОРЕТИЧНІ ОСНОВИ РОБОТИ CLR І ОБФУСКАЦІЇ

1.1 Огляд Common Language Runtime

Common Language Runtime – це основна частина платформи .NET, яка забезпечує виконання програмного коду та надає сервіси для спрощення розробки, це особливо корисно, якщо потрібно інтегрувати декілька мов програмування[2]. CLR є віртуальною машиною, яка виконує код, написаний на різних мовах програмування, підтримуваних платформою .NET, таких як C#, VB.NET, F# та інші (рис. 1.1)[1].

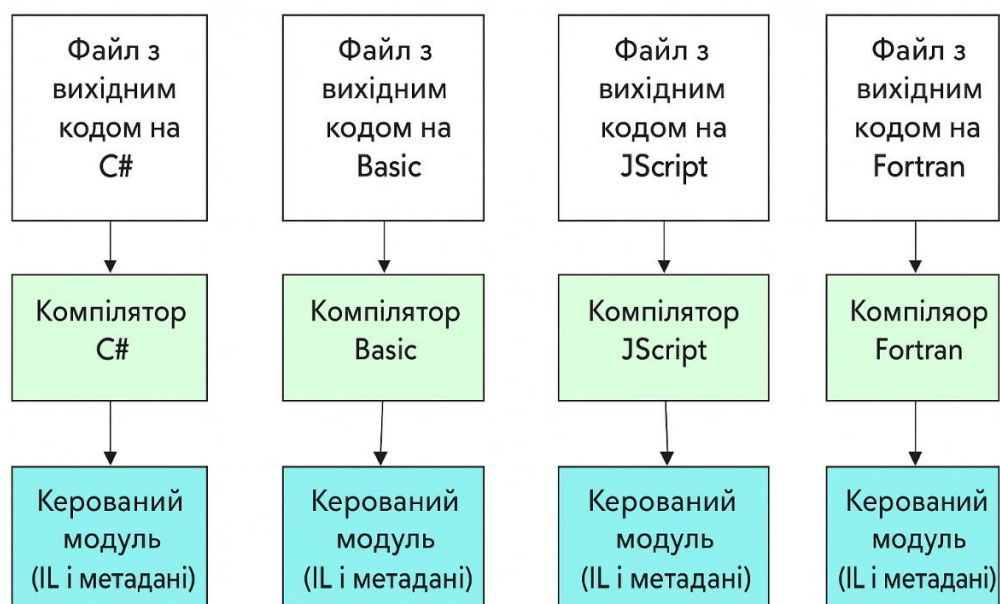


Рисунок 1.1 – Процес компіляції у .NET

Розглянемо основні функції CLR[4]:

Перша функція це *управління пам'яттю*: CLR бере на себе автоматичне управління пам'яттю в програмах .NET, спрощуючи життя розробникам. Основними аспектами є розподіл пам'яті та збір сміття:

- *Розподіл пам'яті*: Коли створюється об'єкт, CLR автоматично виділяє під нього пам'ять у спеціальній області, що називається керованою купою. Не потрібно вручну виділяти або звільняти пам'ять, як у C++.

- *Збір сміття:* CLR має вбудований механізм очищення пам'яті. Збірник сміття аналізує об'єкти в керованій купі та звільняє пам'ять, зайняту об'єктами, які більше не використовуються.

Розглянемо етапи роботи Збірника сміття:

- Пошук недоступних об'єктів;
- Очищення пам'яті, зайнятої такими об'єктами;
- Перерозподіл і дефрагментація пам'яті.

Друга функція CLR - це безпека коду. CLR гарантує безпечне виконання програмного коду, що є важливим для багатокористувацьких або мережових середовищ, та забезпечує, що код дотримується строгої типізації, що знижує ризик помилок на рівні пам'яті. Код виконується у контрольованому середовищі, де неможливий доступ до пам'яті поза межами дозволених ділянок. ІЛ-код перевіряється на наявність помилок або шкідливого функціоналу, перш ніж його компілює JIT-компілятор. Також CLR дозволяє задавати рівень доступу до ресурсів для коду залежно від його походження[5][6].

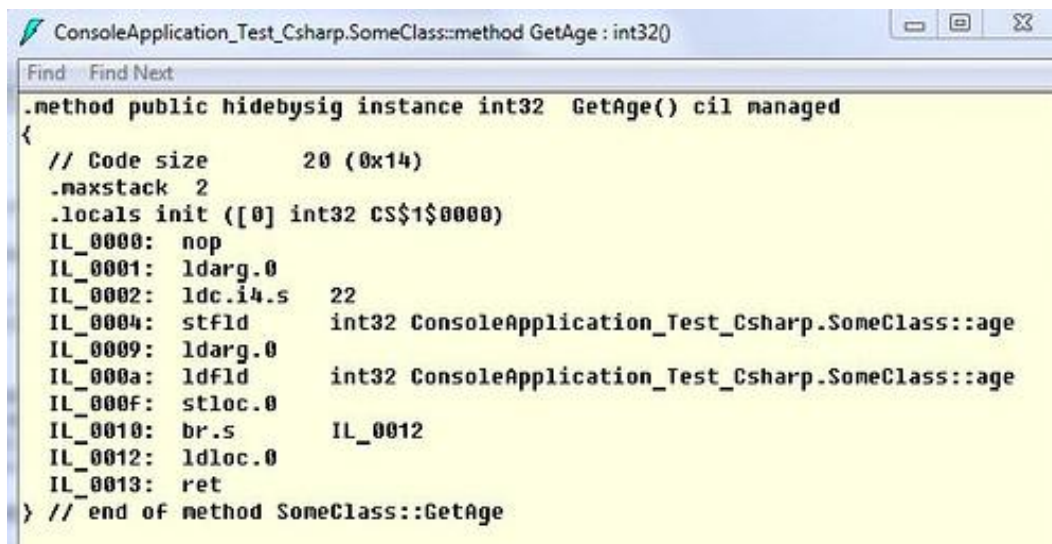
Третя функція CLR - обробка винятків. CLR забезпечує єдиний механізм обробки винятків для всіх мов програмування в .NET, в нього входять[7]:

- *Централізована система обробки:* Незалежно від мови, усі винятки зводяться до об'єкта типу Exception. Це спрощує аналіз і обробку помилок.
- *Вбудовані типи винятків:* CLR надає широкий набір стандартних винятків, наприклад:
 - NullReferenceException (звернення до неіснуючого об'єкта);
 - DivideByZeroException (ділення на нуль);
 - IndexOutOfRangeException (вихід за межі масиву).
- *Ієрархія винятків:* Винятки мають ієрархічну структуру, дозволяючи розробникам створювати власні похідні класи.

Четверта функція CLR – це міжмовна взаємодія. CLR забезпечує сумісність між різними мовами програмування завдяки кільком основним принципам:

		Петренко Д.В.			ККІК.КР.Б – 125 – 16 – ПЗ	Арк.
		Трокоз Є.М.				10
Змн.	Арк.	№ докум.	Підпис	Дата		

1. Використання загальної мови проміжного коду. Код, написаний будь-якою мовою .NET, компілюється в загальний проміжний код (рис. 1.2), який має назву Microsoft Intermediate Language. CLR потім перетворює цей код у машинний для виконання[3][8].



```

ConsoleApplication_Test_Csharp.SomeClass::method GetAge : int32()
{
    // Code size      20 (0x14)
    .maxstack 2
    .locals init ([0] int32 CS$1$0000)
    IL_0000: nop
    IL_0001: ldarg.0
    IL_0002: ldc.i4.s    22
    IL_0004: stfld      int32 ConsoleApplication_Test_Csharp.SomeClass::age
    IL_0009: ldarg.0
    IL_000a: ldfl      int32 ConsoleApplication_Test_Csharp.SomeClass::age
    IL_000f: stloc.0
    IL_0010: br.s      IL_0012
    IL_0012: ldloc.0
    IL_0013: ret
} // end of method SomeClass::GetAge

```

Рисунок 1.2 – Приклад MSIL- коду

2. Наявність системи типів Common Language Specification: Це набір правил, яким повинні відповідати всі мови на платформі .NET, щоб бути сумісними між собою (рис. 1.3)[9]. Наприклад, типи int в C# і integer у VB.NET взаємозамінні.

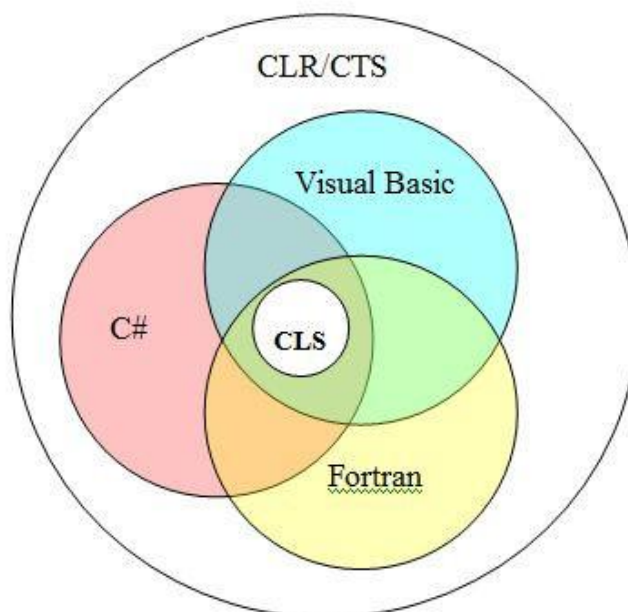


Рисунок 1.3 – Інтеграція мов між собою через CLS

3. Використання метаданих: Кожна збірка .dll або .exe містить метадані, що описують типи, методи та інші елементи. CLR використовує ці метадані для забезпечення взаємодії[2][9].

П'ята функція CLR - управління виконанням коду. CLR виконує MSIL-код через JIT-компіляцію, а також забезпечує керування виконанням:

- *Компіляція в машинний код:* JIT-компілятор перетворює проміжний код у машинний безпосередньо перед виконанням[9]. Це дозволяє програмам адаптуватися до конкретної архітектури.
- *Оптимізація виконання:* JIT-компілятор може вносити оптимізації, які покращують продуктивність програми під час її роботи.

1.2 JIT-компіляція в CLR

JIT-компіляція – це процес перетворення проміжного коду MSIL у машинний код безпосередньо під час виконання програми (рис. 1.4). Цей підхід дозволяє платформі .NET бути кросплатформною та оптимізувати виконання програм для конкретної архітектури процесора. JIT-компіляція є одним із основних компонентів Common Language Runtime[1][10].

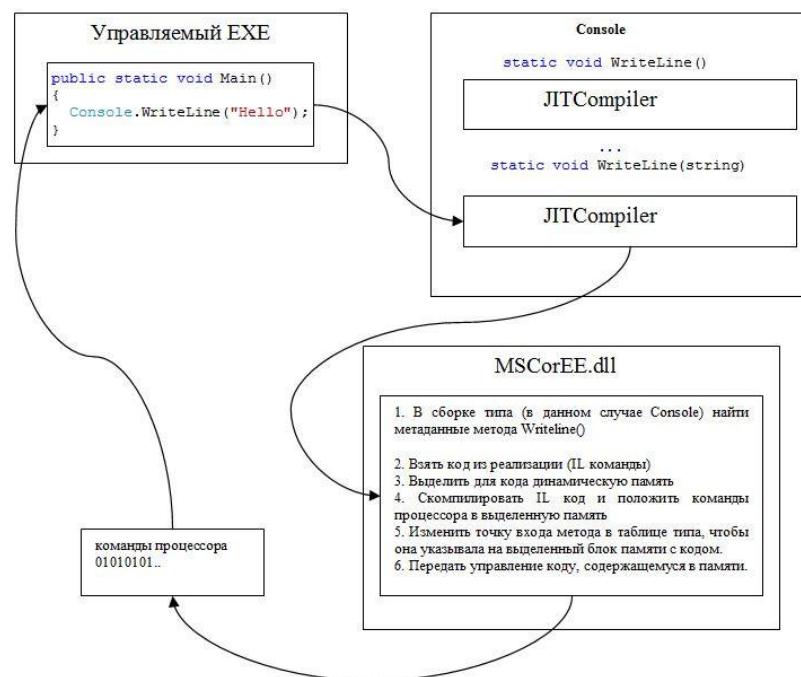


Рисунок 1.4 – Блок-схема роботи JIT-компілятора

		Петренко Д.В.			ККІК.КР.Б – 125 – 16 – ПЗ	Арк.
		Трокоз Є.М.				12
Змн.	Арк.	№ докум.	Підпис	Дата		

Тепер розглянемо JIT-компіляцію детальніше. JIT-компіляція виконується за запитом, коли певний метод або частина коду викликається вперше (рис. 1.5).

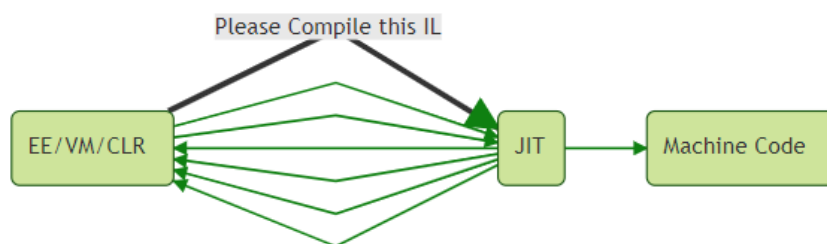


Рисунок 1.5 – Демонстрація першого виклику методу через JIT

Розберемо основні принципи JIT-компіляції:

- *Динамічна компіляція:* Код компілюється під час виконання, а не заздалегідь, що дозволяє програмам адаптуватися до особливостей середовища.
- *Перетворення MSIL у машинний код:* JIT-компілятор перетворює MSIL у машинний код, який може виконуватись безпосередньо на процесорі.
- *Оптимізація під час виконання:* JIT-компілятор аналізує поточні умови, такі як апаратні ресурси та ОС, і виконує оптимізації, які неможливі на етапі компіляції[9][10].

Далі розглянемо типи JIT-компіляторів. CLR підтримує кілька типів JIT-компіляції[10], залежно від вимог програми:

- *Normal JIT:* Методи компілюються лише під час їх першого виклику. Всі скомпільовані методи кешуються, щоб не компілювати їх повторно (рис. 1.6). Недолік цього типу у витратах часу на перший виклик методів.
- *Econo JIT:* Оптимізований для середовищ із обмеженими ресурсами. Компілює лише необхідний код і може відкидати скомпільовані частини після виконання. Недолік: низька продуктивність через повторну компіляцію.
- *Pre-JIT:* Виконується завчасно за допомогою Native Image Generator. Усі методи компілюються під час встановлення програми. Недолік: втрата певних динамічних оптимізацій.

Далі розглянемо етапи роботи JIT-компіляції (рис. 1.6)[10]:

- *Завантаження MSIL:* Коли програма викликає метод, CLR завантажує відповідний код у форматі MSIL разом із метаданими.

- *Верифікація коду:* CLR перевіряє MSIL-код на безпеку та правильність. Якщо код не відповідає вимогам, робота припиняється.
- *Генерація машинного коду:* JIT-компілятор перетворює MSIL-код у машинний код, з зауваженнями про поточну платформу.
- *Кешування:* Скомпільований код методів зберігається у пам'яті, щоб не виконувати повторну компіляцію того ж методу.
- *Виконання:* Скомпільований код виконується.

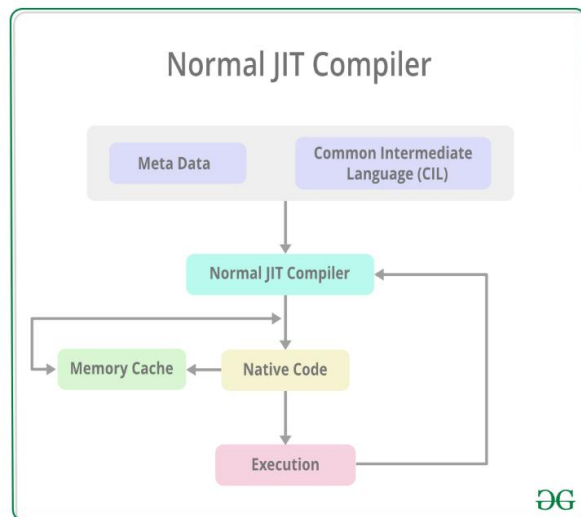


Рисунок 1.6 – Принцип роботи JIT-компілятора

Повний процес компіляції з використанням JIT продемонстровано на рисунку 1.7.

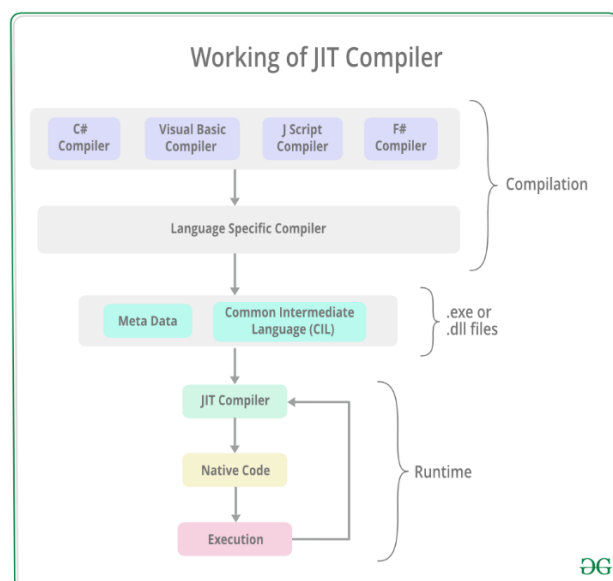


Рисунок 1.7 – Принцип роботи повного процесу компіляції .NET

На основі виконаних нами дослідженнях, розглянемо переваги та недоліки технології JIT-компіляції. Переваги JIT-компіляції:

- *Кросплатформність*: MSIL-код може виконуватись на будь-якому пристрої з підтримкою CLR та .NET.
- *Оптимізація під середовище виконання*: JIT-компіляція враховує особливості машини та середовища.

Недоліки JIT-компіляції:

- *Затримка під час першого виклику*: Перше виконання методу займає більше часу через компіляцію методів під час запуску.
- *Використання пам'яті*: Скомпільований код займає додаткову пам'ять.
- *Потреба в ресурсах*: JIT-компіляція може навантажувати процесор, особливо в середовищах з нехваткою вільних ресурсів.

1.3 CLR Injection: Вразливість платформи .NET

CLR Injection – це метод атаки, який дозволяє впровадити свій код у процеси, що використовують CLR[12]. Це виконується шляхом зміни контексту виконання програми або підміни оригінальних функцій або класів. Мета цієї атаки - це отримання контролю над програмним додатком для виконання певних шкідливих дій, таких як:

- Викрадення даних.
- Зміна поведінки програми.
- Виконання довільного коду.

Розглянемо механізми реалізації CLR Injection[13]:

- *Впровадження через збірки*: Можлива підміна або додавання власних збірок до програми. Під час виконання CLR може завантажити їх замість легітимних.

		Петренко Д.В.			ККІК.КР.Б – 125 – 16 – ПЗ	Арк.
		Трокоз Є.М.				15
Змн.	Арк.	№ докум.	Підпис	Дата		

- *Маніпуляція з метаданими:* У CLR використовується механізм метаданих для опису структур типів, методів та властивостей. Атака може полягати у зміні цих метаданих для підміни реальних методів на шкідливі.
- *Зловживання Рефлексіями:* Використання механізмів рефлексії для виклику приватних або захищених методів, які можуть бути використані для доступу до логіки програми.
- *Підміна AppDomain:* Використання підміненого домену додатку для виконання свого коду в контексті цільового процесу.
- *Впровадження через JIT-компілятор:* Підміна або модифікація JIT-компілятора дозволяє виконувати шкідливий код під час компіляції MSIL у машинний код.

1.4 Обфускація у контексті CLR

Обфускація – це процес зміни структури програмного коду з метою ускладнення його аналізу без впливу на функціональність програми[14][16]. У контексті CLR обфускація розрахована на захист збірок .NET від реверс-інжинірингу, оскільки чистий MSIL-код можна декомпілювати у читабельний людиною код.

NetReactor – це інструмент обфускації, який пропонує розширені методи захисту збірок .NET. Однією з його основних функцій є *приховування тіла методів*, що значно ускладнює декомпіляцію та аналіз коду[15].

Розберемо як NetReactor приховує тіла методів. NetReactor використовує декілька технік для захисту тіл методів у збірках .NET. Розглянемо основні з них[15][16]:

1. *Шифрування тіла методів.* NetReactor шифрує тіло методів, щоб воно не було доступним у вихідному вигляді під час декомпіляції. Цей процес включає:

- *Шифрування MSIL-коду:* тіло методу замінюється зашифрованою версією, яка зберігається в метаданих.

		Петренко Д.В.			ККІК.КР.Б – 125 – 16 – ПЗ	Арк.
		Трокоз Є.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		16

- *Дешифрування під час виконання:* зашифрований MSIL-код розшифровується в момент виконання за допомогою інтегрованого дешифратора.
- *Захист від аналізу:* навіть якщо збірку вдається декомпілювати, зашифрований код залишається нечитабельним.

2. *Ін'єкція завантажувача:* NetReactor замінює оригінальне тіло методу невеликим завантажувачем, який:

- Завантажує зашифроване тіло методу в пам'ять.
- Виконує дешифрування.

3. *Вставка хибних інструкцій:* NetReactor додає до методу хибні інструкції, мертвий код та інші техніки, щоб ускладнити його аналіз. Це призводить до того, що декомпілятори генерують некоректний код, який не відображає реальної логіки роботи методу. Такий підхід значно ускладнює реверс-інжиніринг і аналіз коду.

Тепер детальніше дослідимо методи захисту від декомпіляції[15]:

- *Заплутування MSIL-коду:* NetReactor може модифікувати MSIL-код так, щоб він був працездатним для CLR, але нечитабельним або неспроможним до декомпіляції інструментами аналізу.

- *Динамічне виконання:* захищені методи виконуються динамічно в пам'яті, не залишаючи слідів, доступних для аналізу.

- *Антивідладкові механізми:* NetReactor виконує перевірки на наявність налагоджувачів або інструментів реверс-інжинірингу. Якщо такі інструменти виявляються, виконання програми припиняється або видається хибний результат.

Тепер на основі дослідженої інформації розглянемо переваги використання методу NetReactor:

- *Високий рівень захисту:* шифрування та динамічне виконання ускладнюють аналіз коду.
- *Гнучкість:* NetReactor дозволяє комбінувати методи захисту, такі як шифрування, обфускація імен та вставка хибних інструкцій.

		Петренко Д.В.			ККІК.КР.Б – 125 – 16 – ПЗ	Арк.
		Трокоз Є.М.				17
Змн.	Арк.	№ докум.	Підпис	Дата		

○ *Мінімальний вплив на продуктивність*: незважаючи на використання дешифрування під час виконання, вплив на продуктивність мінімальний, оскільки дешифрування виконується лише один раз під час першого виклику методу.

1.5 NetReactor і підміна compileMethod для приховування методів

Одним із передових методів, що використовуються інструментом NetReactor для захисту збірок .NET, є підміна функції compileMethod під час JIT-компіляції[12]. Цей підхід використовується для приховування тіла методів до моменту їх виконання, що значно ускладнює реверс-інжиніринг і аналіз програми.

CompileMethod – це функція, яка використовується JIT-компілятором CLR для перетворення MSIL-коду в машинний код (рис. 1.8). Вона викликається для кожного методу під час його першого виконання, після чого скомпільований код кешується, і повторна компіляція не потрібна.

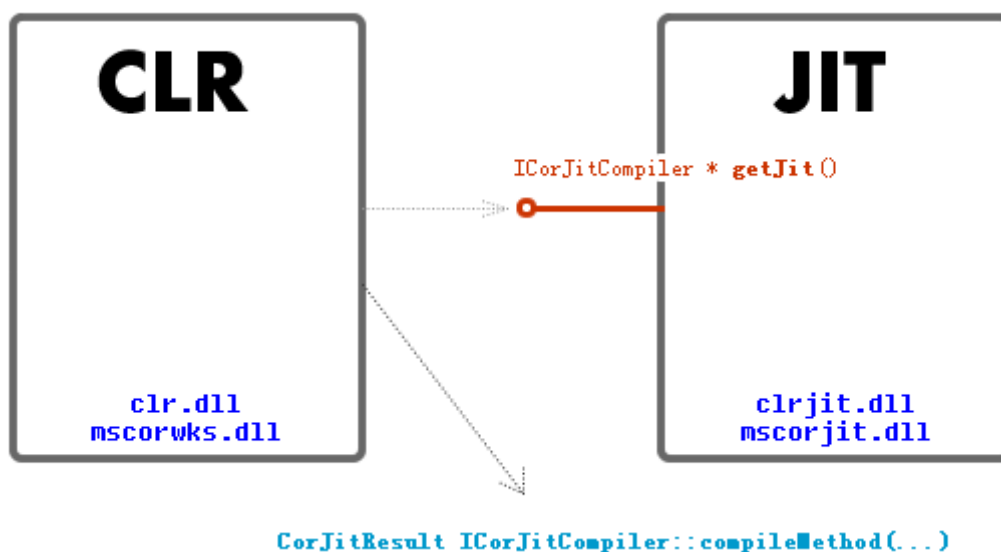


Рисунок 1.8 – Принцип виклику compileMethod

NetReactor використовує техніку перехоплення функції compileMethod, щоб впровадити свій код у процес JIT-компіляції[13].

Головна задача всіх цих методів та технологій це протидія реверс-інжинірингу[15]. Розглянемо деякі функції, що ускладнюють реверс-інжиніринг:

○ *Антиналагодження*: NetReactor перевіряє наявність активних налагоджувачів і може припинити виконання програми або повернути хибні дані у випадку, якщо налагоджувач виявлено.

○ *Шифрування в пам'яті*: Навіть після дешифрування тіло методу може залишатися у зашифрованій формі в пам'яті та дешифруватися лише для виконання поточної інструкції.

○ *Динамічне оновлення коду*: Зашифрований код може змінюватися між сеансами виконання, що робить його декомпіляцію ще складнішою.

Також тепер необхідно розглянути недоліки метода підміни `compileMethod`:

○ *Підвищені вимоги до продуктивності*: Дешифрування та динамічна компіляція можуть збільшувати затримки під час першого виклику методу. Затримки не значні, але вони присутні.

○ *Потенційні конфлікти*: Підміна `compileMethod` може спричинити проблеми сумісності з іншими інструментами, які залежать від роботи JIT-компілятора.

Висновки до першого розділу

У даному розділі було розглянуто основні теоретичні аспекти роботи середовища виконання коду Common Language Runtime, який є основним компонентом .NET програмування. Розглянуто принцип роботи JIT-компіляції, а також механізми, які роблять CLR вразливою до модифікацій, таких як CLR Injection. Було розглянуто обфускацію, як засіб захисту програмного забезпечення від аналізу та реверс-інжинірингу. Було розглянуто як NetReactor використовує обфускацію і техніку підміни методу `compileMethod`, щоб приховати тіла методів і ускладнити їх декомпіляцію.

		Петренко Д.В.			ККІК.КР.Б – 125 – 16 – ПЗ	Арк.
		Трокоз Є.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		19

РОЗДІЛ 2 РОЗРОБКА ІНСТРУМЕНТІВ ДЛЯ АНАЛІЗУ ОБФУСКОВАНОГО КОДУ

2.1 Дослідження структури додатку

Першим кроком нашої роботи буде дослідження структури додатку, який був підвернутий обфускації через NetReactor. Для цього використовуємо програму dnSpy, яка надає можливість реверс інжинірингу практично будь-яких програм, написаних на C#.

Можемо побачити що програма ніяк не ідентифікується як C# додаток, та не містить нічого корисного для нас (рис. 2.1).

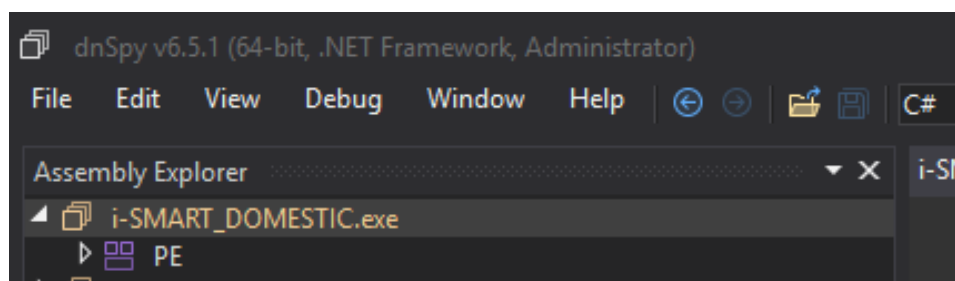


Рисунок 2.1 – Спроба переглянути структуру додатку

Наступним кроком буде спроба запустити програму через налагоджувач на x86 (рис. 2.2).

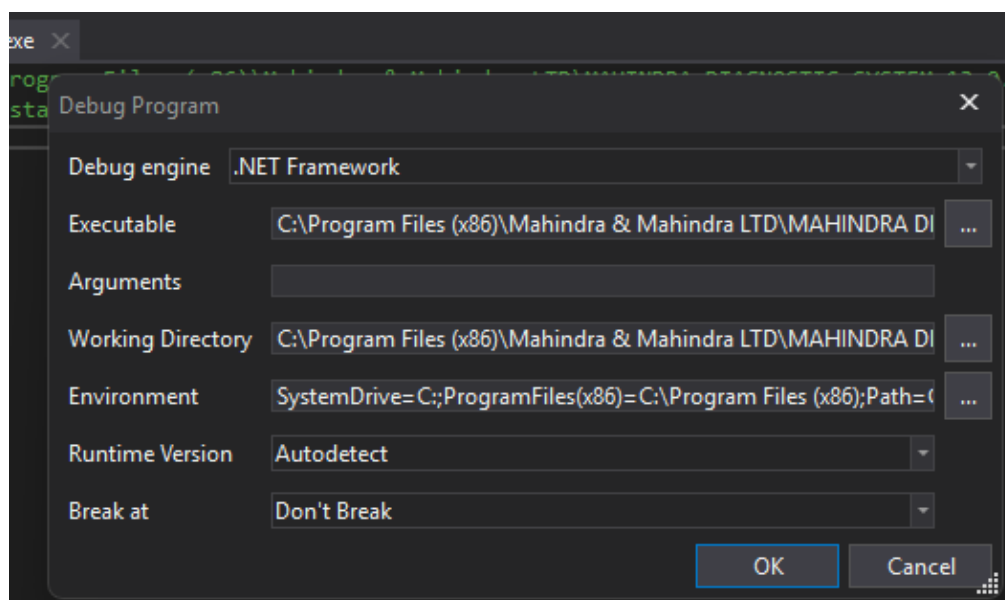


Рисунок 2.2 – Запуск додатку через налагоджувач

Маємо наступний результат – додаток виводить повідомлення, про те що неможливо запустити додаток через те, що на нашій машині активний налагоджувач (рис. 2.3).

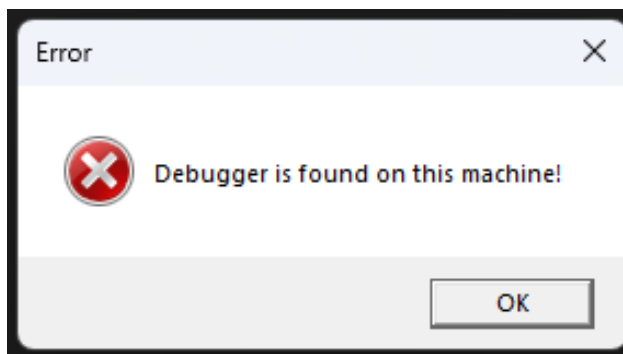


Рисунок 2.3 – Відмова запуску додатку

Якщо напряду додаток дослідити не виходить, спробуємо запустити програму ExtremeDumper та зробити його дамп (рис. 2.4). Дампер – це інструмент, який дозволяє зробити знімок процесу додатка. Дамп представляє собою збережений стан оперативної пам'яті процесу у відведений період часу і включає у себе всі дані, код, бібліотеки та інші структури додатку, що надає нам можливість їх дослідити.

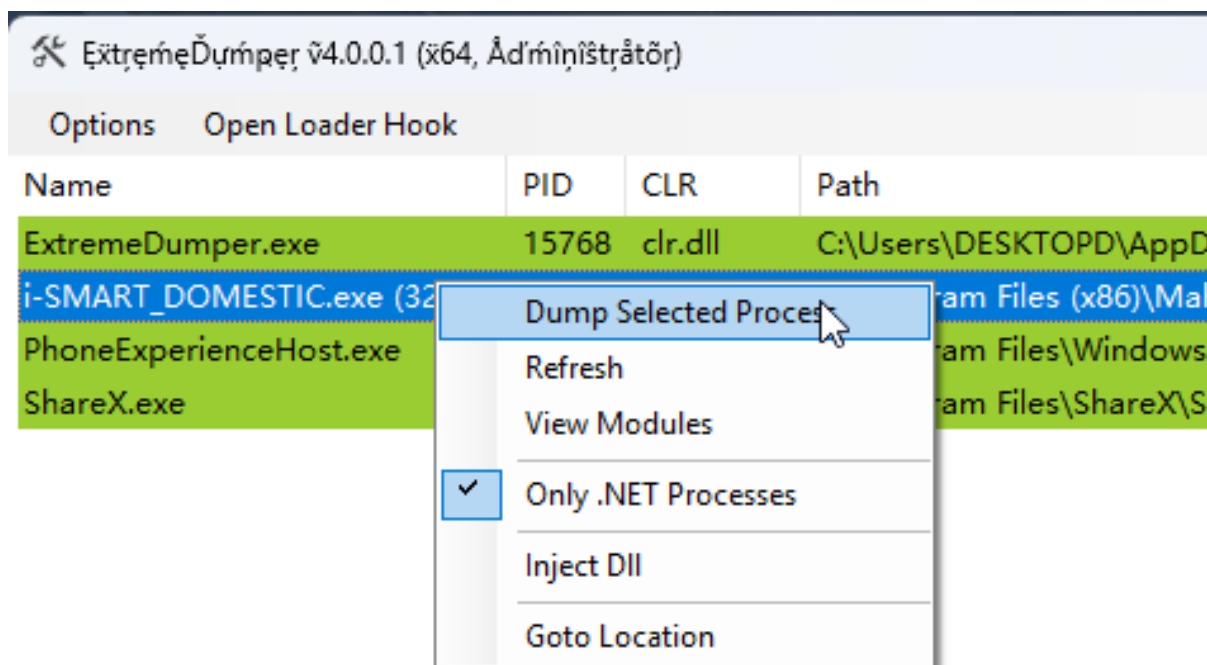


Рисунок 2.4 – Дамп програми через ExtremeDumper

Результатом виконання є новостворена папка з програмою та бібліотеками які представляють собою знімок оперативної пам'яті (рис. 2.5).

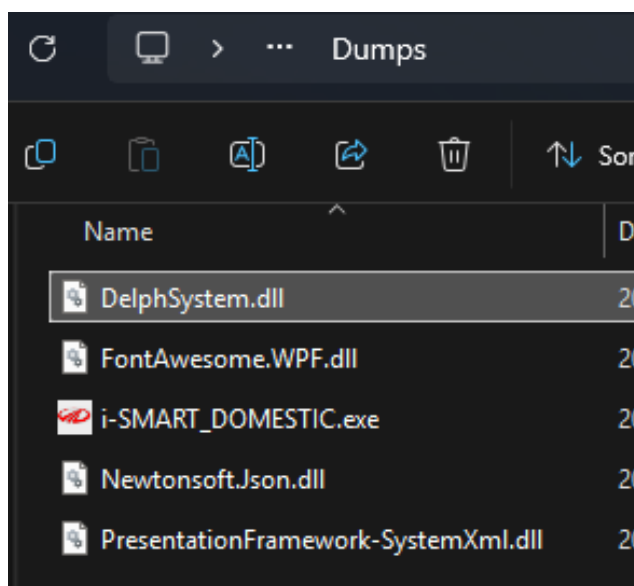


Рисунок 2.5 – Результат виконання дампу

Наступним кроком буде відкриття дампу нашого додатку у dnSpy. Як ми бачимо, нам тепер відображаються простори імен. Якщо ми розгорнемо їх далі, то побачимо класи, методи, поля і таке інше. Але додаток був підданий обфускації, і тепер все, що ми бачимо – це заплутані назви методів та відсутність тіл методів (рис. 2.6).

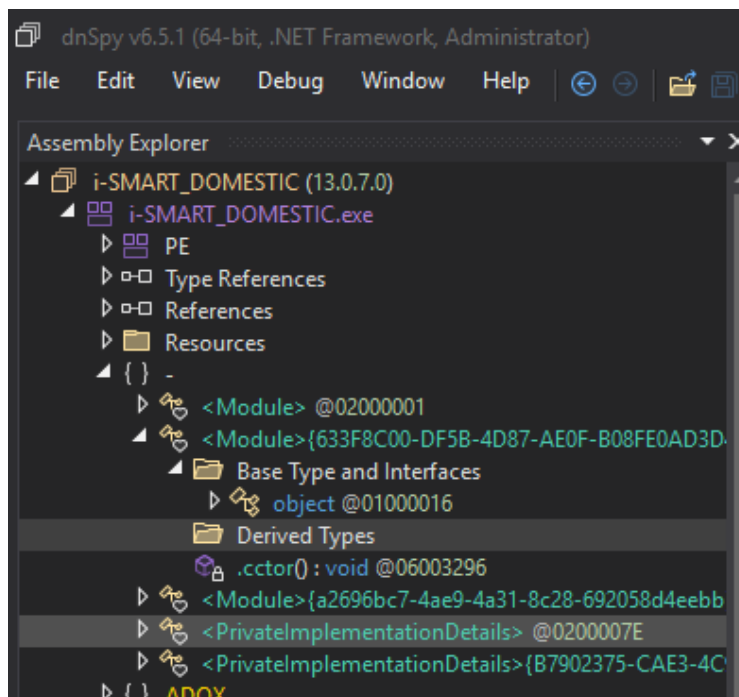


Рисунок 2.6 – Структура додатку

		Петренко Д.В.			ККІК.КР.Б – 125 – 16 – ПЗ	Арк.
		Трокоз Є.М.				22
Змн.	Арк.	№ докум.	Підпис	Дата		

Спроба переглянути IL-код теж не має сенсу, оскільки тепер там лише набір несуттєвих інструкцій (рис. 2.7).

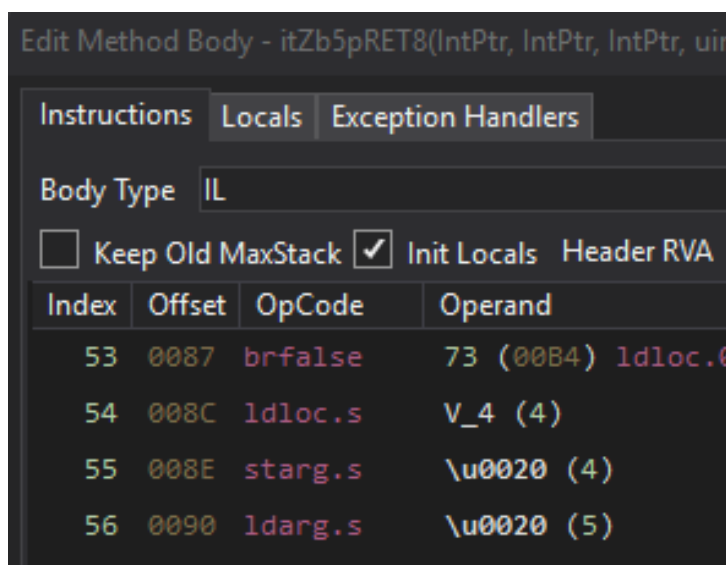


Рисунок 2.7 – Результат переглядання IL-коду

Наша задача знайти та перехопити `compileMethod`, який був підмінений через `NetReactor`. Залишається питання, як саме дізнатися, що потрібно перехоплювати.

Для виконання певних дій програми необхідно виділити пам'ять. У випадку, якщо пам'ять не може бути виділена автоматично, як це робить C#, використовується метод `Marshal.AllocCoTaskMem`. Розглянемо де він використовується (рис. 2.8).

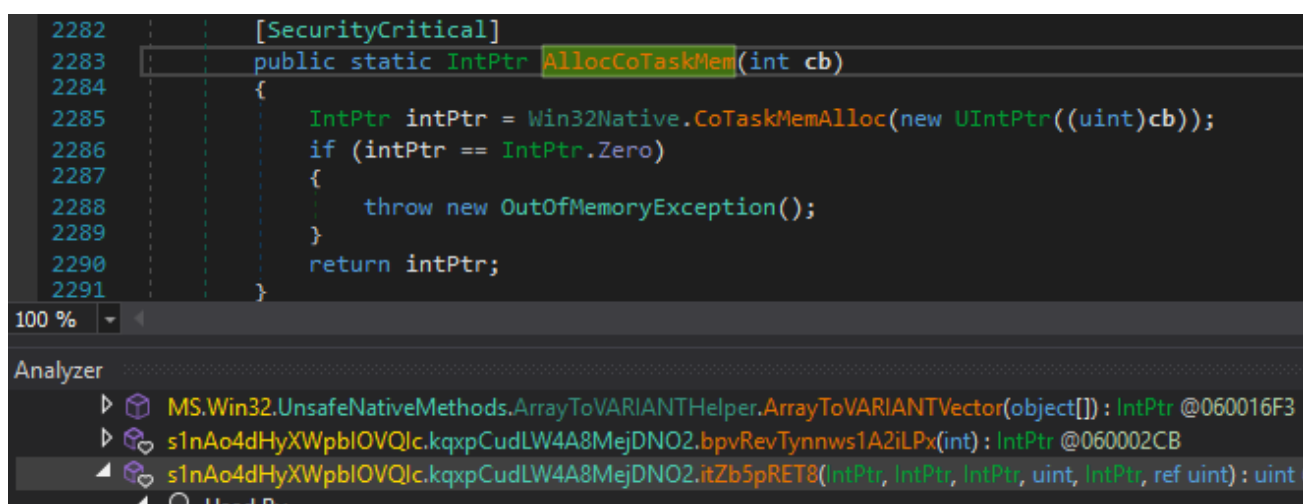


Рисунок 2.8 – Список використання `Marshal.AllocCoTaskMem`

Серед переліку бачимо наступний метод – itZb5pRET8, розглянемо його детальніше (рис. 2.9). Звертаємо увагу на те, що кількість аргументів цього методу майже збігається з оригінальною кількістю аргументів у compileMethod.

```
[MethodImpl(MethodImplOptions.NoInlining)]
internal static uint itZb5pRET8(intPtr \u0020, IntPtr \u0020, IntPtr \u0020, [MarshalAs(UnmanagedType.U4)] uint \u0020, IntPtr \u0020, ref uint \u0020)
{
    IntPtr intPtr = \u0020;
    if (kqxpCudLW4A8MejDNO2.jYebR4HPHV)
    {
        intPtr = \u0020;
    }
    long num;
    if (IntPtr.Size == 4)
    {
        num = (long)Marshal.ReadInt32(intPtr, IntPtr.Size * 2);
    }
    else
    {
        num = Marshal.ReadInt64(intPtr, IntPtr.Size * 2);
    }
    object obj = kqxpCudLW4A8MejDNO2.DdSKEXryOx[num];
    if (obj == null)
    {
        return kqxpCudLW4A8MejDNO2.L4UK1ddRQh(\u0020, \u0020, \u0020, \u0020, \u0020, ref \u0020);
    }
    kqxpCudLW4A8MejDNO2.TrpEb1KBfSYOhvKQAgk trpEb1KBfSYOhvKQAgk = (kqxpCudLW4A8MejDNO2.TrpEb1KBfSYOhvKQAgk)obj;
    IntPtr intPtr2 = Marshal.AllocCoTaskMem(trpEb1KBfSYOhvKQAgk.aQAK3DsNdu.Length);
```

Рисунок 2.9 – Знайдений compileMethod

Це і буде compileMethod цього додатку. Якщо почати аналізувати код, ми з'ясуємо, що в object obj = kqxpCudLW4A8MejDNO2.DdSKEXryOx[num]; зберігаються оригінальні тіла методів, а сам об'єкт представлений певним Hashtable (рис. 2.10).

```
// Token: 0x0400010F RID: 271
internal static object DdSKEXryOx = new Hashtable();
```

Рисунок 2.10 – Дослідження об'єкту DdSKEXryOx

Також ми з'ясуємо, що DdSKEXryOx є певним словником, ключем якого є тип long, а значенням – деяка структура, через яку проходять байти скомпільованих методів (рис. 2.11).

```
// Token: 0x0400011D RID: 285
internal struct TrpEb1KBfSYOhvKQAgk
{
    // Token: 0x0400011E RID: 286
    internal bool QlvKaoglnL;
    internal byte[] aQAK3DsNdu;
```

Рисунок 2.11 – Структура об'єкту

Тепер ми маємо можливість за допомогою бібліотеки Harmony перехопити методи і, використовуючи різні методи рефлексії, отримати доступ до цього поля та дістати оригінальні байти методів.

		Петренко Д.В.			ККІК.КР.Б – 125 – 16 – ПЗ	Арк.
		Трокоз Є.М.				24
Змн.	Арк.	№ докум.	Підпис	Дата		

2.2 Проєктування загального алгоритму програми

Після дослідження структури програми приступимо до етапу проєктування. Нам необхідні будуть 2 бібліотеки: бібліотека-інжектор на C++ та бібліотека-аналізатор на C#. Спочатку складемо загальний алгоритм роботи бібліотеки-інжектора (рис. 2.12). Ми реалізуємо точку входу у бібліотеку, створюємо консоль та ініціалізуємо CLR середовище, після чого шукаємо домен додатку, який ми аналізуємо, та перевіряємо чи знаходиться у ньому exe-збірка. Після чого ми завантажуюмо бібліотеку на C#, знаходимо головний клас та метод Main, після чого виконуємо виклик цього методу, що приведе до виконання коду бібліотеки-аналізатора.

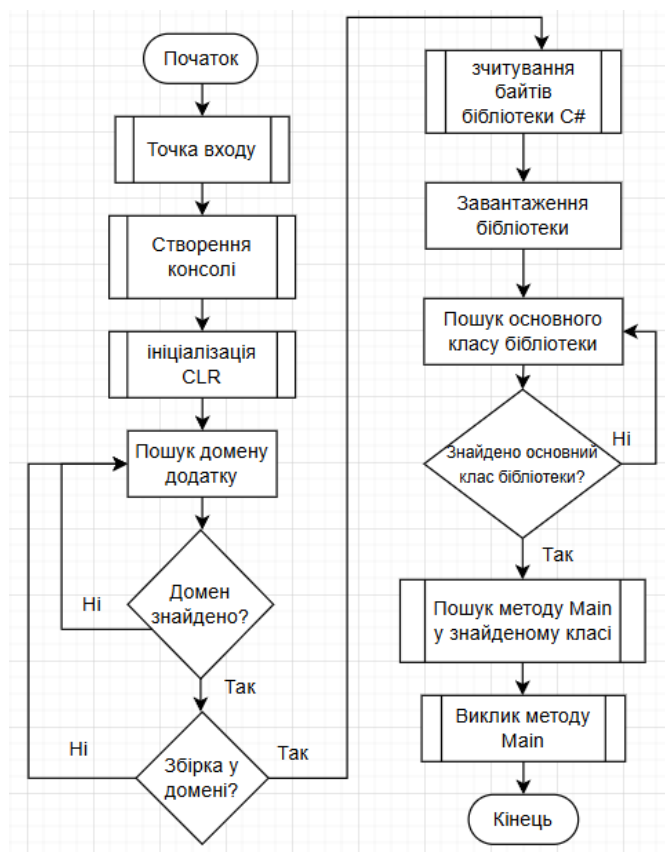


Рисунок 2.12 – Блок схема бібліотеки-інжектора

Тепер розглянемо алгоритм роботи бібліотеки-аналізатора (рис. 2.13). Ми починаємо з отримання поточного домену, після чого ми шукаємо збірку з `compileMethod`, знаходимо клас, сам `compileMethod` та поле з байтами методів. Також встановлюємо патч на `compileMethod` за заданим нами методом, який ми

розглянемо далі. Це надасть нам можливість виконання методу до компіляції методів додатком.

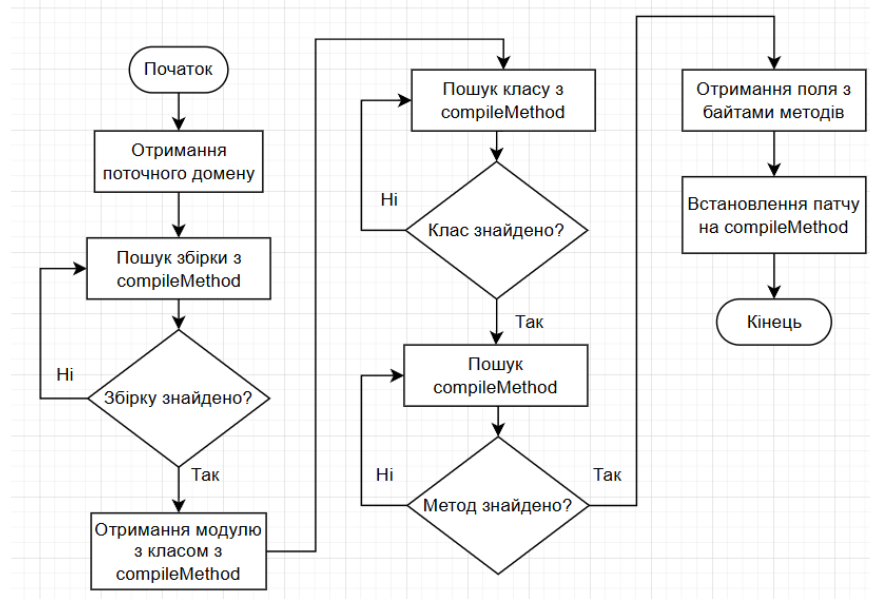


Рисунок 2.13 – Блок схема головного методу бібліотеки-інжектора

Тепер оглянемо патч-метод (рис. 2.14). Ми отримуємо ключ словника із вхідних аргументів, після чого звертаємося до отриманого раніше поля з використанням знайденого ключа, перевіряємо чи об'єкт не пустий, зчитуємо байти перехоплених методів та виводимо їх.

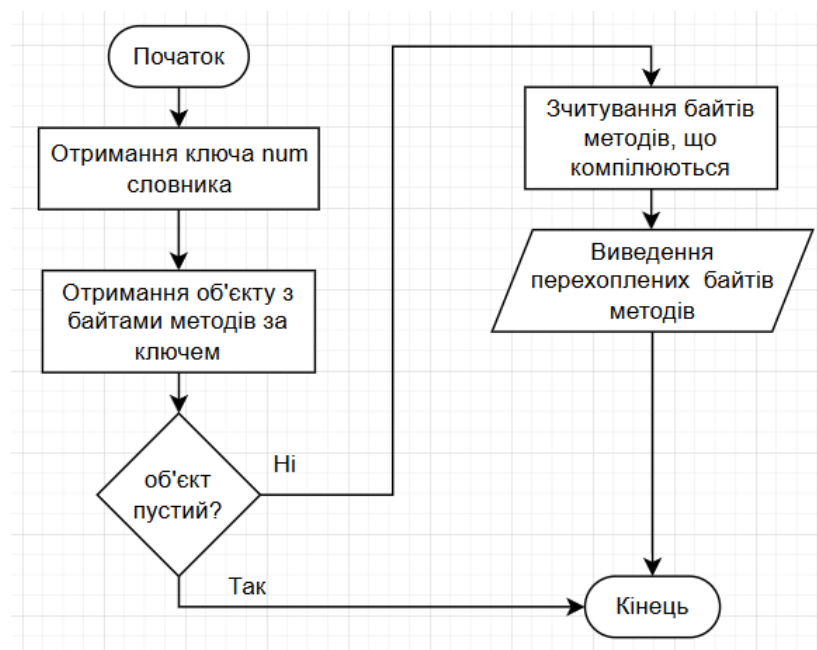


Рисунок 2.14 – Блок схема патч-методу бібліотеки-інжектора

2.3 Розробка CLR Injector та завантаження бібліотеки-аналізатора

На цьому етапі ми проводимо розробку бібліотеки DLL на мові C++ з ініціалізацією CLR середовища. Реалізуємо точку входу в DLL, якщо причина виклику точки входу – під'єднання до процесу, то починаємо запуск середовища (рис. 2.15)[17].

```
BOOL APIENTRY DllMain( HMODULE hModule,
                      DWORD ul_reason_for_call,
                      LPVOID lpReserved
                      )
{
    if (ul_reason_for_call == DLL_PROCESS_ATTACH)
    {
        ConsoleSetup();
        rundotnet();
    }
    else if (ul_reason_for_call == DLL_PROCESS_DETACH)
    {
    }

    return TRUE;
}
```

Рисунок 2.15 – Точка входу в DLL

Далі створюємо відлагодочну консоль з можливостями виводу тексту, помилок та введення тексту (рис. 2.16).

```
void ConsoleSetup()
{
    AllocConsole();
    freopen("CONOUT$", "w", stdout);
    freopen("CONOUT$", "w", stderr);
    freopen("CONIN$", "r", stdin);
}
```

Рисунок 2.16 – Відлагодочна консоль

Наступним кроком є створення функції ініціалізації CLR середовища[18]. CLRCreateInstance створює об'єкт метаклассу CLR, який дає змогу знаходити і керувати середовищем. Метод GetRuntime повертає об'єкт, що містить інформацію про версію CLR. GetInterface повертає ICorRuntimeHost, який

відповідає за ініціалізацію та запуск CLR у поточному процесі, після чого ми запускаємо основний метод програми (рис. 2.17)[11].

```
void InstallCorBindHook() {
    HRESULT hr;

    CoInitializeEx(nullptr, COINIT_MULTITHREADED);
    ICLRMetaHost* pMetaHost = nullptr;
    ICLRRuntimeInfo* pRuntimeInfo = nullptr;
    pCorHost = nullptr;

    hr = CLRCreateInstance( // отримання ICLRMetaHost
        CLSID_CLRMetaHost,
        IID_ICLRMetaHost,
        reinterpret_cast<void*>(&pMetaHost)
    );

    hr = pMetaHost->GetRuntime( // Отримуємо ICLRRuntimeInfo
        L"v4.0.30319",
        IID_ICLRRuntimeInfo,
        reinterpret_cast<void*>(&pRuntimeInfo)
    );

    hr = pRuntimeInfo->GetInterface( // ICorRuntimeHost
        CLSID_CorRuntimeHost,
        IID_ICorRuntimeHost,
        reinterpret_cast<void*>(&pCorHost)
    );

    std::thread th(startFuncLoop, pCorHost);
    th.detach();
}
```

Рисунок 2.17 – Ініціалізація CLR

Далі реалізуємо метод startFuncLoop для реалізації основної логіки роботи нашої бібліотеки. Спочатку нам потрібно перебрати всі Домени додатку і вибрати той, в якому знаходяться бібліотеки. Для цього скористаємося EnumDomain і, за допомогою циклу та NextDomain, зможемо перебрати всі AppDomain (рис. 2.18).

```
void startFuncLoop(ICorRuntimeHost* pCorRuntimeHost) {
    Sleep(1000);
    HRESULT hr;
    HCORENUM hEnum = nullptr;
    const wchar_t* targetName = L"i-SMART_DOMESTIC.exe";

    while (true) {
        if (hEnum) { // скидаємо перелічник, якщо існує
            pCorRuntimeHost->CloseEnum(hEnum);
            hEnum = nullptr;
        }

        hr = pCorRuntimeHost->EnumDomains(&hEnum);
        if (FAILED(hr)) {
            continue;
        }

        bool found = false;
        while (true) {
            IUnknown* pUnkDomain = nullptr;
            hr = pCorRuntimeHost->NextDomain(hEnum, &pUnkDomain); // Перебір доменів
            if (hr == S_FALSE) break;
            if (FAILED(hr) || !pUnkDomain) continue;
        }
    }
}
```

Рисунок 2.18 – startFuncLoop та пошук домена

		Петренко Д.В.			ККІК.КР.Б – 125 – 16 – ПЗ	Арк.
		Трокоз Є.М.				28
Змн.	Арк.	№ докум.	Підпис	Дата		

Далі отриманий домен необхідно перетворити з IUnknown на _AppDomain, для цього використаємо спеціальний метод QueryInterface (рис. 2.19).

```
CComPtr<_AppDomain> pAppDomain;
hr = pUnkDomain->QueryInterface(__uuidof(_AppDomain), (void**)&pAppDomain);
```

Рисунок 2.19 – Перетворення формату даних з IUnknown на _AppDomain

Тепер нам потрібно викликати get_FriendlyName і порівняти імена з потрібними через wcscmp, це потрібно, тому що не всі бібліотеки можуть знаходитися в DefaultAppDomain (рис. 2.20).

```
BSTR bstrName = nullptr;
pAppDomain->get_FriendlyName(&bstrName);
wcout << bstrName << endl;
if (bstrName && wcscmp(bstrName, targetName) == 0) {
    found = true;
}
```

Рисунок 2.20 – Пошук необхідної бібліотеки

Все, що нам тепер потрібно зробити, – це завантажити байти бібліотеки (рис. 2.21) і викликати функцію load_3 (рис. 2.22).

```
std::vector<BYTE> assemblyData = ReadFileToBytes(
    L"D:\\Main\\Project\\ClassLibrary1\\ClassLibrary1\\bin\\x86\\Release\\"
    L"ClassLibrary1.dll");
SAFEARRAYBOUND bounds[1];
bounds[0].lLbound = 0;
bounds[0].cElements = static_cast<ULONG>(assemblyData.size());

SAFEARRAY* pSafeArray = SafeArrayCreate(VT_UI1, 1, bounds);
if (pSafeArray) {
    for (LONG i = 0; i < assemblyData.size(); i++) {
        SafeArrayPutElement(pSafeArray, &i, &assemblyData[i]);
    }
}
```

Рисунок 2.21 – Завантаження бібліотеки

```
_Assembly* pUnkAssembly;

hr = pAppDomain->Load_3(pSafeArray, &pUnkAssembly);
```

Рисунок 2.22 – Загрузка бібліотеки

Після завантаження нашої бібліотеки в пам'ять додатку нам необхідно викликати метод з точки входу для запуску коду, який потрібно буде виконати.

Для цього нам потрібно отримати метод, використовуючи рефлексію, перебравши всі існуючі класи та їх методи.

Створюємо змінні для зберігання нашого методу класу та простору імен (рис. 2.23).

```
CComPtr < _MethodInfo> method;
CComPtr<_Type> pType;
BSTR asmName;
pAssembly->get_FullName(&asmName);
wcout << asmName << endl;
```

Рисунок 2.23 – Змінні для зберігання методу класу та простору імен

Зберігаємо всі можливі типи з нашої збірки в SafeArray і отримуємо розміри масиву (рис. 2.24).

```
SAFEARRAY* pSafeArray = nullptr;
hr = pAssembly->GetTypes(&pSafeArray); // пошук класів бібліотеки
if (FAILED(hr) || pSafeArray == nullptr) {
    std::cout << "Failed to get types. HRESULT: " << hr << std::endl;
}

long lowerBound, upperBound; // Розмір масива
SafeArrayGetLBound(pSafeArray, 1, &lowerBound);
SafeArrayGetUBound(pSafeArray, 1, &upperBound);
```

Рисунок 2.24 – Збереження можливих типів зі збірки в SafeArray

Перебираємо всі наші типи всередині збірки і, керуючись нашим фільтром, де вказаний простір імен та клас, у випадку збігу зберігаємо цей тип (рис. 2.25).

```
for (long i = lowerBound; i <= upperBound; ++i) {
    hr = SafeArrayGetElement(pSafeArray, &i, &pType);
    if (SUCCEEDED(hr) && pType) {
        BSTR typeName;
        hr = pType->get_FullName(&typeName); // знаходження типів
        std::wcout << L" Global Type: " << typeName << std::endl;
        if (SUCCEEDED(hr)) {
            if (wcscmp(typeName, L"Patcher.MyClass") == 0) {
                SysFreeString(typeName);
                break;
            }
            SysFreeString(typeName);
        }
    }
}
```

Рисунок 2.25 – Збереження необхідного типу збірки

		Петренко Д.В.			ККІК.КР.Б – 125 – 16 – ПЗ	Арк.
		Трокоз Є.М.				30
Змн.	Арк.	№ докум.	Підпис	Дата		

Перевіряємо чи зберігся тип, після чого отримуємо метод за його ім'ям та типом (рис. 2.26).

```
cout << pType << endl;
if (!SUCCEEDED(hr)) {
    wprintf(L"Failed to get type: 0x%x\n", hr);
    return;
}
GetMethodByName(L"Main", pType, method);
```

Рисунок 2.26 – Отримання методу за іменем

Так як нам не надано функцію з подібним функціоналом, ми реалізуємо її самостійно. У цьому фрагменті коду представлена частина реалізації GetMethodByName. Ми приймаємо на вхід фільтри імен, тип та вказівник, куди буде збережений метод. Створюємо змінну для зберігання масиву методів, далі оголошуємо змінну bindingFlags для вказання прапорців збірки, вказуємо public, static та instance, щоб охопити максимально можливий діапазон функцій. Потім викликаємо з вхідного типу метод GetMethods і передаємо йому аргументи: bindingFlags та посилання на наш масив методів. Далі отримуємо розміри нашого масиву (рис. 2.27).

```
HRESULT GetMethodByName(BSTR methodNameFilter, CComPtr<_Type> type, CComPtr<_MethodInfo>& ppMethod) {
    SAFEARRAY* pMethods;
    BSTR name;
    BindingFlags bindingFlags = static_cast<BindingFlags>(BindingFlags_Public | BindingFlags_Static
        | BindingFlags_Instance);
    type->GetMethods(bindingFlags, &pMethods);
    type->get_FullName(&name);
    wcout << name << endl;
    long lowerBound, upperBound;
    SafeArrayGetLBound(pMethods, 1, &lowerBound);
    SafeArrayGetUBound(pMethods, 1, &upperBound);
```

Рисунок 2.27 – Частина реалізації GetMethodByName

Далі нам необхідно за допомогою перебору елементів масиву отримати потрібний нам метод за заданим фільтром. Для цього створюємо цикл, у якому перебираємо всі елементи масиву, що містить методи, отримуємо ім'я кожного

методу та порівнюємо його з нашим фільтром. У випадку, якщо ім'я методу співпадає з фільтром, зберігаємо його та очищуємо пам'ять (рис. 2.28).

```
std::cout << "Number of methods in types: " << (upperBound - lowerBound + 1)
for (long i = lowerBound; i <= upperBound; ++i) {
    CComPtr<_MethodInfo> pMethod;
    HRESULT hr = SafeArrayGetElement(pMethods, &i, &pMethod);
    if (SUCCEEDED(hr) && pMethod) {
        BSTR methodName;
        hr = pMethod->get_name(&methodName);
        std::wcout << L"Global Method Found: " << methodName << std::endl;
        if (SUCCEEDED(hr)) {
            if (_wcsicmp(methodName, methodNameFilter) == 0) {
                std::wcout << L"Method Found: " << methodName << std::endl;
                ppMethod = pMethod;
                SysFreeString(methodName);
                break;
            }
            SysFreeString(methodName);
        }
    }
}
SafeArrayDestroy(pMethods);
```

Рисунок 2.28 – Продовження GetMethodByName

Після отримання методу нам потрібно його викликати, для цього ми реалізуємо InvokeMethod, куди ми передаємо аргументи, які отримали раніше, а також задаємо bindingFlags та інші пусті параметри. Далі для переданого нами методу викликаємо внутрішній метод invoke2. Після цього буде виконуватись код нашої бібліотеки C# (рис. 2.29).

```
HRESULT InvokeMethod(CComPtr<_MethodInfo> pMethod) {
    VARIANT obj;
    VariantInit(&obj);

    obj.vt = VT_NULL;

    BindingFlags bindingFlags = static_cast<BindingFlags>(BindingFlags_Public |
        BindingFlags_Instance | BindingFlags_Static);
    _Binder* binder = nullptr;
    SAFEARRAY* pParams = nullptr;
    _CultureInfo* culture = nullptr;

    VARIANT result;
    VariantInit(&result);

    HRESULT hr = pMethod->Invoke_2(obj, bindingFlags, binder, pParams, culture, &result);
    if (SUCCEEDED(hr)) {
        if (result.vt == VT_BSTR) {
            std::wcout << L"Method result: " << result.bstrVal << std::endl;
        }
        else {
            std::cout << "Method executed successfully, but the result type is not BSTR."
                << std::endl;
        }
    }
    else {
        wprintf(L"Failed to invoke method: 0x%x\n", hr);
    }

    VariantClear(&obj);
    VariantClear(&result);
    return hr;
}
```

Рисунок 2.29 – Реалізація InvokeMethod

		Петренко Д.В.			ККІК.КР.Б – 125 – 16 – ПЗ	Арк.
		Трокоз Є.М.				32
Змн.	Арк.	№ докум.	Підпис	Дата		

2.4 Розробка аналізатора методів

Ми розробляємо динамічну бібліотеку DLL мовою C#, яка призначена для аналізу програми під час її виконання. Основне завдання бібліотеки – надати можливість перехоплення методів із використанням бібліотеки Harmony за допомогою механізмів рефлексії. Для початку створюємо нашу точку входу, яку викликатиме інжектор. Це обов’язково має бути статичний клас зі статичним методом (рис. 2.30).

```
namespace Library {  
    0 references  
    public static class EntryPointClass {  
        0 references  
        public static void EntryPoint() {  
            Patcher.MyClass.Main();  
        }  
    }  
}
```

Рисунок 2.30 – Створення точки входу

У точці входу, яку ми вказали для виклику в нашій бібліотеці на C++, ми викликаємо наш статичний метод Main, в якому реалізовано отримання всіх збірок (рис. 2.31). За допомогою методів рефлексії ми отримуємо збірку, яка використовує обфускацію та містить у собі compileMethod.

```
public static void Main()  
{  
    Assembly[] asms = null;  
    deps = null;  
  
    AppDomain currentDomain = AppDomain.CurrentDomain;  
  
    asms = currentDomain.GetAssemblies();  
    foreach (Assembly a in asms) // пошук збірки з compileMethod  
    {  
        Console.WriteLine(a.FullName);  
        if (a.GetName().Name == "DelphSystem")  
        {  
            deps = a;  
        }  
    }  
}
```

Рисунок 2.31 – Пошук необхідної збірки

Після цього ми шукаємо перший модуль, оскільки в ньому будуть знаходитись всі типи, які нас стосуються. Далі ми перебираємо типи і знаходимо

клас у якому знаходиться compileMethod, після чого за аналогічною логікою знаходимо наш метод і зберігаємо його (рис. 2.32).

```
MethodInfo method = null;
Module module = deps.GetModules()[0]; // 1й модуль
Console.WriteLine(module.Name);
Type type = null;
var types = module.GetTypes();
foreach (Type t in types) {
    if (t.Name == "kqxpCudLW4A8MejDNO2") { // клас з методом
        type = t;
        Console.WriteLine("Found");
        break;
    }
}
var methods = type.GetMethods(BindingFlags.NonPublic
    | BindingFlags.Static | BindingFlags.Instance);

foreach (MethodInfo mi in methods) {
    if (mi.Name == "itZb5pRET8") { // compileMethod
        Console.WriteLine($"{mi.Name} - Found");
        method = mi;
    }
};
```

Рисунок 2.32 – Знаходження compileMethod

Після цього ми за токеном об'єкта hashtable (див. рис. 2.10) отримуємо дані поля об'єкту-словника. Після чого ми реалізуємо патч compileMethod з посиланням на метод який ми розглянемо далі. (рис. 2.33).

```
int token = 0x0400010F;
// поле s1nAo4dHyXWpbIOVQIc.kqxpCudLW4A8MejDNO2.DdSKEXry0x за токеном;
FieldInfo fi;
fi = deps.GetModules()[0].ResolveField(token);
fieldValue = (Hashtable)fi.GetValue(null);
Console.WriteLine($"Found Field: {fi.Name}, Value = {fieldValue}");

try // патч метода
{
    var harmony = new Harmony("com.example.patch");
    var patch = new HarmonyMethod(typeof(MyClass).GetMethod("MethodPatch"));
    var result = harmony.Patch(method, prefix: patch);
    Console.WriteLine(result.Name);
}
catch (Exception e) {
    Console.WriteLine(e.Message);
}
```

Рисунок 2.33 – Пошук поля з даними та створення патчу

Далі використовуючи бібліотеку Harmony для перехоплення виклику функції, ми встановлюємо хук на compileMethod, який був реалізований обфускатором, і реалізуємо свій метод, який завдяки бібліотеці Harmony вклинюється у додаток і буде виконуватися раніше за оригінальну реалізацію compileMethod, та буде зчитувати байти методів що компілюються у даний

момент часу із отриманого раніше fieldValue, за отриманим ключем num з поля aQAK3DsNdu (див рис. 2.11) структури яка містить ці байти. (рис. 2.34).

```
public static bool MethodPatch(object[] __args)
{
    IntPtr ptr = (IntPtr)__args[2]; // копія оригінальної структури програми
    long num;
    if (IntPtr.Size == 4)
    {
        num = (long)Marshal.ReadInt32(ptr, IntPtr.Size * 2);
    }
    else
    {
        num = Marshal.ReadInt64(ptr, IntPtr.Size * 2);
    }
    Console.WriteLine("Method compile");
    object obj = fieldValue[num]; // об'єкт з fieldValue по ключу

    if (obj != null)
    {
        var structType = obj.GetType();

        byte[] data = (byte[])structType // оригінальні байти
            .GetField("aQAK3DsNdu", BindingFlags.Instance | BindingFlags.NonPublic)
            .GetValue(obj);
        Console.WriteLine(BitConverter.ToString(data).Replace("-", " "));
    }
}
```

Рисунок 2.34 – Хук compileMethod

Ми виконали дії, які необхідні, виходячи зі структури коду оригінального compileMethod (рис. 2.35). Спочатку ми перевіряємо розрядність системи, після чого отримуємо obj, але з власною реалізацією.

```
internal static uint itZb5pRET8(IntPtr \u0020, IntPtr \u0020, IntPtr \u0020, [MarshalAs(UnmanagedType.U4)]
    \u0020)
{
    IntPtr intPtr = \u0020;
    if (kqxpCudLW4A8MejDN02.jYebR4HPHV)
    {
        intPtr = \u0020;
    }
    long num;
    if (IntPtr.Size == 4)
    {
        num = (long)Marshal.ReadInt32(intPtr, IntPtr.Size * 2);
    }
    else
    {
        num = Marshal.ReadInt64(intPtr, IntPtr.Size * 2);
    }
    object obj = kqxpCudLW4A8MejDN02.DdSKEXryOx[num];
    if (obj == null)
    {
    }
}
```

Key I

Object -> Dictionary<long, struct>

struct Data{
byte[] bytes
bool unknown
}

Рисунок 2.35 – Оригінальний compileMethod від NetReactor

Висновки до другого розділу

У цьому розділі було проаналізовано логіку роботи додатку, який ми досліджуємо, був розроблений інжектор CLR на мові C++, який завантажує бібліотеки у середовище додатку, та було реалізовано аналізатор методів у вигляді бібліотеки на мові C#, який перехоплює оригінальні байти методів, які компілюються додатком.

		Петренко Д.В.			ККІК.КР.Б – 125 – 16 – ПЗ	Арк.
		Трокоз Є.М.				35
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 3 ПІДГОТОВКА СЕРЕДОВИЩА ТА ТЕСТУВАННЯ РОБОТИ ПРОГРАМИ

3.1 Підготовка середовища

Першим кроком підготовки є повне вимкнення будь-якого антивірусного програмного забезпечення, що працює у системі (рис. 3.1). Це необхідно, оскільки розроблені бібліотеки та DLL-інжектор, який ми розглянемо далі, втручаються у процеси операційної системи та виконують дії, що можуть вважатися потенційно шкідливими. Вони впливають на цілісність ОС, впроваджуючи себе або інші збірки туди, де вони не передбачені системною безпекою. Через це антивірусні засоби або захисні механізми Windows одразу реагують і блокують спроби запуску або завантаження наших компонентів, вважаючи їх небезпечними.



Рисунок 3.1 – Вимкнення Windows Defender

Якщо цього не зробити, отримаємо повідомлення у списку історії захисту Windows (рис. 3.2), про те що програма виконує потенційно шкідливі дії та відправляє все в карантин, після чого виконувати відновлення буває

		Петренко Д.В.			ККІК.КР.Б – 125 – 16 – ПЗ	Арк.
		Трокоз Є.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		36

проблематично, через те що починаються проблеми зі зв’язками. Також іноді деякі файли можуть пошкодитись.

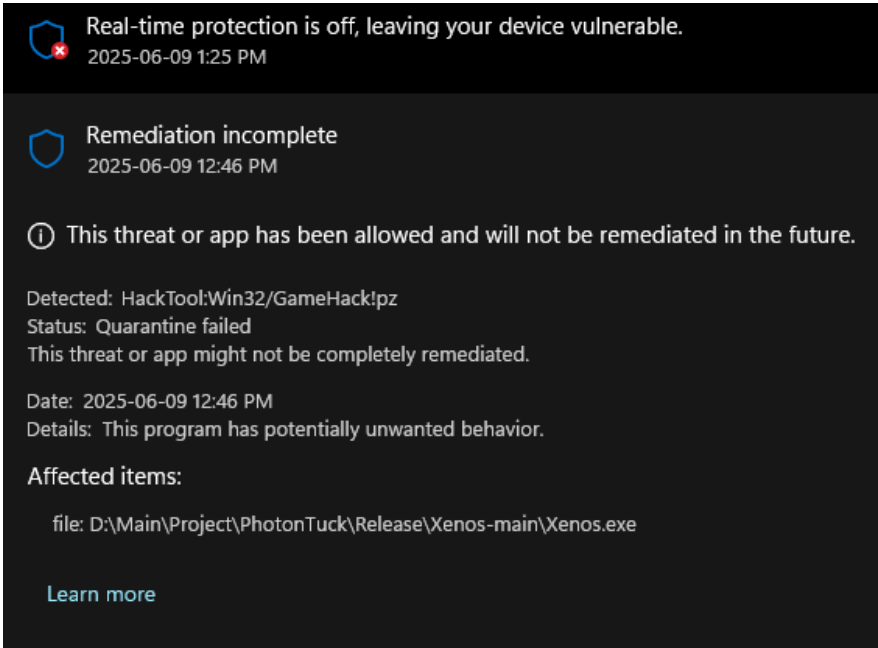


Рисунок 3.2 – Повідомлення від Windows Defender

Далі, для того щоб впровадити нашу бібліотеку у процес, ми використовуємо інжектор Xenos, який знаходиться у вільному доступі та має відкритий програмний код. Сторінка гітхабу з інформацією та авторами наведено на рисунку 3.3.

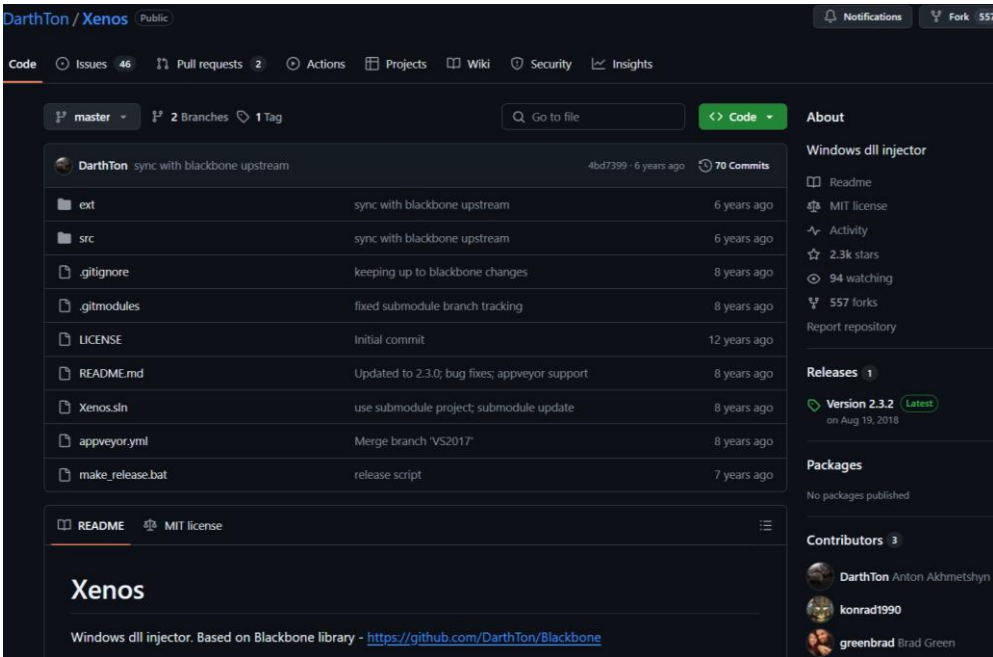


Рисунок 3.3 – Github сторінка інжектору Xenos

Далі завантажуюємо та запускаємо його, після чого в інжекторі ми вибираємо нашу зібрану DLL на C++ та обираємо x86 архітектуру (рис. 3.4).

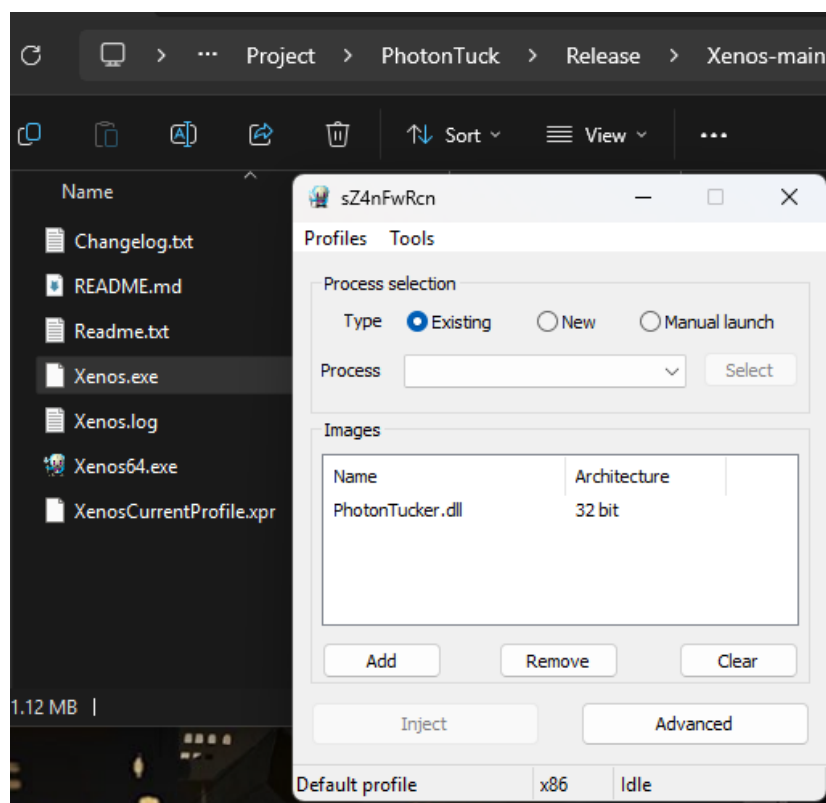


Рисунок 3.4 – Інжектор Xenos

Після чого нам треба задати затримку у 1 секунду (рис. 3.5), бо якщо цього не зробити, успішно виконуватись код буде нестабільно, через те що він не буде встигати виконуватись.

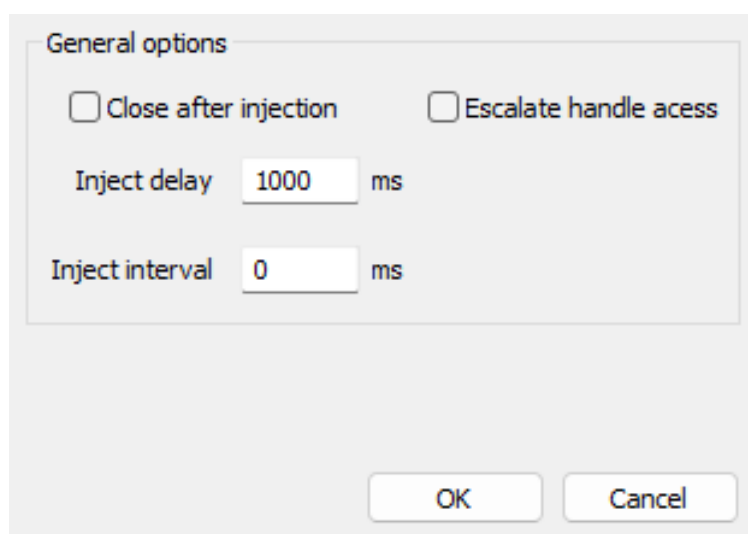


Рисунок 3.5 – Встановлення затримки інжектора

3.2 Демонстрація роботи програми та отримання байтів методів

Тепер ми запускаємо додаток який ми досліджуємо, вибираємо в інжекторі існуючий процес, та шукаємо необхідний процес по назві додатку, після чого натискаємо Inject. В результаті відкривається консольне вікно з результатами виконання перехоплення тіл методів (рис. 3.6).

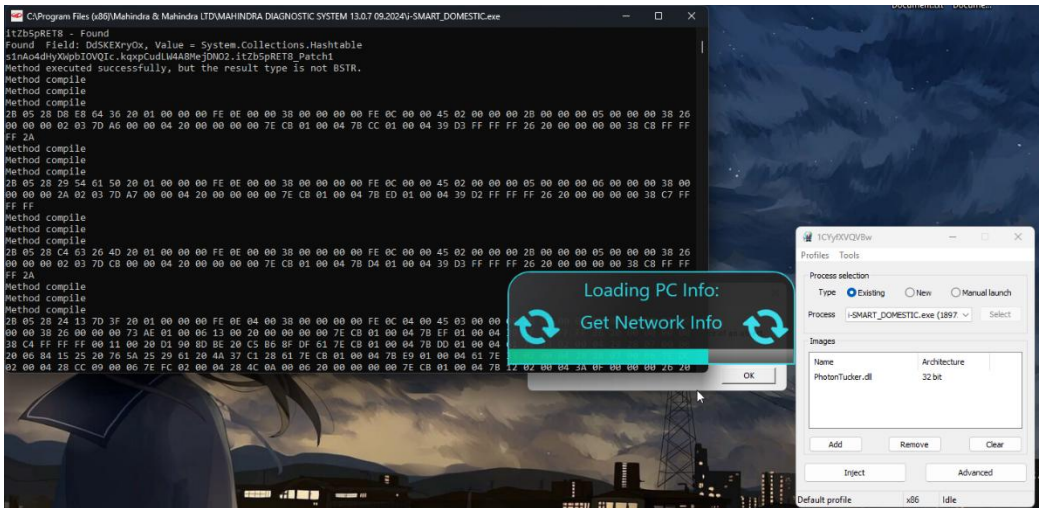


Рисунок 3.6 – Виконання інжекції бібліотеки

Тепер розглянемо отримані результати, першим ми бачимо повідомлення, які ми створювали для дебагінгу програми під час її розробки (рис. 3.7). Серед них: назви та номери збірок, результати виконання методів, результати пошуків методів, класів, збірок та інформація про них тощо.

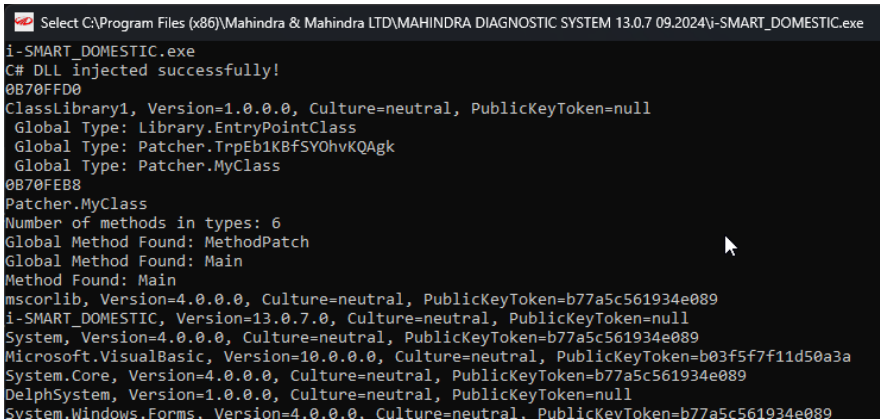


Рисунок 3.7 – Результат дебагінгу під час виконання програми

Після чого ми бачимо результати виконання хука, повідомлення про компіляцію, та оригінальні байти методів які компілювалися (рис. 3.8). Повний результат з більшою кількість методів наведено у додатку В.

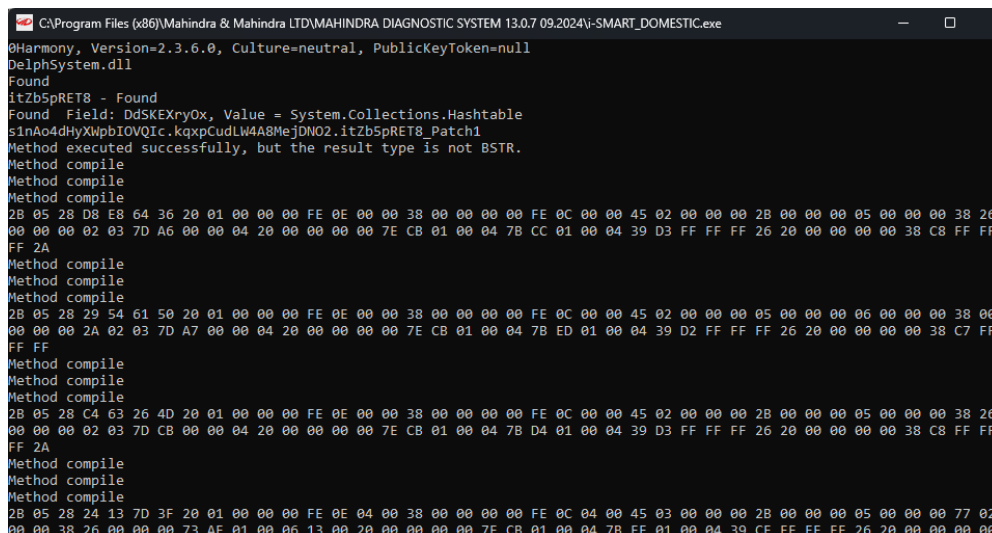


Рисунок 3.8 – Оригінальні байти скомпільованих методів

Ці байти представляють собою послідовність байт-коду, який виконується в середовищі CLR. З ними можна зробити декодування у вигляді IL-інструкцій, використовуючи таблицю Opcode (рис. 3.9)[19]. Однак для цього потрібні глибокі знання у сфері реверс-інжинірингу, структури .NET CLR та внутрішньої організації байт-коду, які вже виходять за рамки дослідження обфускації.

Opcode ↕	Instruction ↕	Description ↕	Type of instruction ↕
0x58	add	Add two values, returning a new value.	Base instruction
0xD6	add.ovf	Add signed integer values with overflow check.	Base instruction
0xD7	add.ovf.un	Add unsigned integer values with overflow check.	Base instruction
0x5F	and	Bitwise AND of two integral values, returns an integral value.	Base instruction
0xFE 0x00	arglist	Return argument list handle for the current method.	Base instruction
0x3B	beq <int32 (target)>	Branch to target if equal .	Base instruction
0x2E	beq.s <int8 (target)>	Branch to target if equal , short form.	Base instruction
0x3C	bge <int32 (target)>	Branch to target if greater than or equal to.	Base instruction
0x2F	bge.s <int8 (target)>	Branch to target if greater than or equal to, short form.	Base instruction
0x41	bge.un <int32 (target)>	Branch to target if greater than or equal to (unsigned or unordered).	Base instruction
0x34	bge.un.s <int8 (target)>	Branch to target if greater than or equal to (unsigned or unordered), short form.	Base instruction

Рисунок 3.9 – Список частини CIL інструкцій

Висновки до третього розділу

У цьому розділі було виконано підготовку системи, встановлено та використано інжектор CLR для впровадження нашої збірки у додаток. У результаті маємо байти оригінальних методів, які компілювались додатком.

ВИСНОВКИ

У кваліфікаційній роботі було проведено дослідження технологій обфускації у середовищі Common Language Runtime з акцентом на вивчення механізмів захисту програмного коду. Основна увага приділялася аналізу роботи платформи .NET, її основних компонентів, а також потенційних загроз, що виникають у результаті вразливостей, пов'язаних із реверс-інжинірингом. Розглянуто теоретичні аспекти JIT-компіляції та вразливості CLR Injection, які створюють додаткові ризики для безпеки програмного забезпечення.

Було виконано дослідження механізмів обфускації, що дозволяють приховувати методи та ускладнюють доступ до логіки програм. Особливу увагу було приділено аналізу роботи NetReactor, зокрема, його здатності змінювати процес компіляції для приховування методів. У процесі роботи було проаналізовано структуру програми, розроблено інструменти для дослідження цих механізмів, включаючи CLR Injector, який забезпечує інтеграцію себе у додаток та завантаження бібліотеки C#, яка виконує перехоплення оригінальних байтів методів, які компілюються додатком.

У результаті виконання роботи ми отримали програмний інструментарій, який дозволяє використати вразливість CLR Injection для отримання оригінальних байтів методів в обхід механізмів захисту програмного забезпечення.

Досягнуті результати сприяють розширенню знань та навичок у сфері захисту та аналізу програмного забезпечення, що є актуальним у сучасних умовах розвитку інформаційних технологій.

		Петренко Д.В.			ККІК.КР.Б – 125 – 16 – ПЗ	Арк.
		Трокоз Є.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		41

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Richter J. R. CLR via C# : підручник. 4th ed. Redmond, Washington : Microsoft Press, 2012. 896 p.
2. Common Language Runtime (CLR) overview [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/dotnet/standard/clr?redirectedfrom=MSDN>
3. Введення у MSIL. Що таке MSIL/IL. Іldasm і декомпіляція IL-коду [Електронний ресурс] – Режим доступу до ресурсу: <https://metanit.com/sharp/msil/1.1.php>
4. Common Language Runtime (CLR) in C# [Електронний ресурс] – Режим доступу до ресурсу: <https://www.geeksforgeeks.org/common-language-runtime-clr-in-c-sharp/>
5. Code Access Security in CLR [Електронний ресурс] – Режим доступу до ресурсу: https://positiwise.com/blog/code-access-security-in-clr#Code_Group
6. Code Access Security Basics [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/previous-versions/dotnet/framework/code-access-security/code-access-security-basics>
7. C# | Exception [Електронний ресурс] – Режим доступу до ресурсу: <https://www.geeksforgeeks.org/c-sharp-exception/?ref=shm>
8. CIL or MSIL | Microsoft Intermediate Language or Common Intermediate Language [Електронний ресурс] – Режим доступу до ресурсу: <https://www.geeksforgeeks.org/cil-or-msil-microsoft-intermediate-language-or-common-intermediate-language/>
9. Особливості роботи CLR в .NET framework [Електронний ресурс] – Режим доступу до ресурсу: <https://habr.com/ru/articles/90426/>
10. What is Just-In-Time(JIT) Compiler in .NET [Електронний ресурс] – Режим доступу до ресурсу: <https://www.geeksforgeeks.org/what-is-just-in-time-jit-compiler-in-dot-net/>

		Петренко Д.В.			ККІК.КР.Б – 125 – 16 – ПЗ	Арк.
		Трокоз Є.М.				42
Змн.	Арк.	№ докум.	Підпис	Дата		

11. ICLRRuntimeHost Interface [Електронний ресурс] – Режим доступу до ресурсу: <https://learn.microsoft.com/en-us/dotnet/framework/unmanaged-api/hosting/iclrruntimehost-interface>
12. Ін'єкція коду у .NET CLR: зміна IL-коду під час виконання програми [Електронний ресурс] – Режим доступу до ресурсу: <https://habr.com/en/articles/154419/>
13. .NET CLR Injection: Modify IL Code during Run-time [Електронний ресурс] – Режим доступу до ресурсу: <https://www.codeproject.com/Articles/463508/NET-CLR-Injection-Modify-IL-Code-during-Run-time>
14. Обфускація як метод захисту програмного забезпечення [Електронний ресурс] – Режим доступу до ресурсу: <https://habr.com/en/articles/533954/>
15. .Net Reactor. The Ultimate .NET Obfuscator features [Електронний ресурс] – Режим доступу до ресурсу: https://www.eziriz.com/reactor_features.htm
16. Обфускація коду – що, як і навіщо [Електронний ресурс] – Режим доступу до ресурсу: <https://habr.com/en/articles/735812/>
17. How to debug DLLMain function [Електронний ресурс] – Режим доступу до ресурсу: <https://stackoverflow.com/questions/47968776/how-to-debug-dllmain-function>
18. How to inject .NET DLL using CLR Hosting [Електронний ресурс] – Режим доступу до ресурсу: <https://www.unknowncheats.me/forum/general-programming-and-reversing/332825-inject-net-dll-using-clr-hosting.html>
19. List of CIL instructions [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/List_of_CIL_instructions

ДОДАТКИ

		Петренко Д.В.			ККІК.КР.Б – 125 – 16 – ПЗ	Арк.
		Трокоз Є.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		44

Лістинг dllmain.cpp бібліотеки-інжектора на C++

```

#include <cor.h>
#include <iostream>
#include <vector>
#include <algorithm>
#include <thread>
#include <string>
#include <metahost.h>
#include <atlbase.h>
#include <fstream>

#pragma comment(lib, "mscoree.lib")
#import "mscorlib.tlb" raw_interfaces_only\
    high_property_prefixes("_get", "_put", "_putref")\
    rename("ReportEvent", "InteropServices_ReportEvent")
using namespace mscorlib;
using namespace std;

extern "C" __declspec(dllexport) void PhotonTuck() {}
extern "C" __declspec(dllexport) void dummy() {};

void ConsoleSetup()
{
    AllocConsole();
    freopen("CONOUT$", "w", stdout);
    freopen("CONOUT$", "w", stderr);
    freopen("CONIN$", "r", stdin);
}

std::vector<BYTE> ReadFileToBytes(const std::wstring& filePath) {
    std::ifstream file(filePath, std::ios::binary);
    return std::vector<BYTE>(std::istreambuf_iterator<char>(file), {});
}

HRESULT InvokeMethod(CComPtr<_MethodInfo> pMethod) {
    VARIANT obj;
    VariantInit(&obj);

    obj.vt = VT_NULL;

    BindingFlags bindingFlags = static_cast<BindingFlags>(BindingFlags_Public |
        BindingFlags_Instance | BindingFlags_Static);
    _Binder* binder = nullptr;
    SAFEARRAY* pParams = nullptr;
    _CultureInfo* culture = nullptr;

    VARIANT result;
    VariantInit(&result);

    HRESULT hr = pMethod->Invoke_2(obj, bindingFlags, binder, pParams, culture, &result);
    // виклик методу main
    if (SUCCEEDED(hr)) {
        if (result.vt == VT_BSTR) {
            std::wcout << L"Method result: " << result.bstrVal << std::endl;
        }
        else {
            std::cout << "Method executed successfully, but the result type is not BSTR."
            << std::endl;
        }
    }
}

```

		Петренко Д.В.			ККІК.КР.Б – 125 – 16 – ПЗ	Арк.
		Трокоз Є.М.				45
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    }
    else {
        wprintf(L"Failed to invoke method: 0x%x\n", hr);
    }

    VariantClear(&obj);
    VariantClear(&result);
    return hr;
}

HRESULT GetMethodByName(BSTR methodNameFilter, CComPtr<_Type> type, CComPtr<_MethodInfo>&
ppMethod) {
    SAFEARRAY* pMethods;
    BSTR name;
    BindingFlags bindingFlags = static_cast<BindingFlags>(BindingFlags_Public |
BindingFlags_Static // флаги мметоду
    | BindingFlags_Instance);
    type->GetMethods(bindingFlags, &pMethods); // отримання методів
    type->get_FullName(&name);
    wcout << name << endl;
    long lowerBound, upperBound;
    SafeArrayGetLBound(pMethods, 1, &lowerBound);
    SafeArrayGetUBound(pMethods, 1, &upperBound);
    std::cout << "Number of methods in types: " << (upperBound - lowerBound + 1) <<
std::endl;

    for (long i = lowerBound; i <= upperBound; ++i) { // пошук методу main
        CComPtr<_MethodInfo> pMethod;
        HRESULT hr = SafeArrayGetElement(pMethods, &i, &pMethod);
        if (SUCCEEDED(hr) && pMethod) {
            BSTR methodName;
            hr = pMethod->get_name(&methodName);
            std::wcout << L"Global Method Found: " << methodName << std::endl;
            if (SUCCEEDED(hr)) {
                if (_wcsicmp(methodName, methodNameFilter) == 0) {
                    std::wcout << L"Method Found: " << methodName << std::endl;
                    ppMethod = pMethod;
                    SysFreeString(methodName);
                    break;
                }
                SysFreeString(methodName);
            }
        }
    }
    SafeArrayDestroy(pMethods);

    return (S_OK);
}

ICorRuntimeHost* pCorHost = nullptr;

void startFuncLoop(ICorRuntimeHost* pCorRuntimeHost) {
    Sleep(1000);
    HRESULT hr;
    HCORENUM hEnum = nullptr;
    const wchar_t* targetName = L"i-SMART_DOMESTIC.exe";

    while (true) {
        if (hEnum) { // скидаємо перелічник, якщо існує
            pCorRuntimeHost->CloseEnum(hEnum);
            hEnum = nullptr;
        }
    }
}

```

		Петренко Д.В.			ККІК.КР.Б – 125 – 16 – ПЗ	Арк.
		Трокоз Є.М.				46
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    }

    hr = pCorRuntimeHost->EnumDomains(&hEnum);
    if (FAILED(hr)) {
        continue;
    }

    bool found = false;
    while (true) {
        IUnknown* pUnkDomain = nullptr;
        hr = pCorRuntimeHost->NextDomain(hEnum, &pUnkDomain); // Перебір доменів
        if (hr == S_FALSE) break;
        if (FAILED(hr) || !pUnkDomain) continue;

        CComPtr<_AppDomain> pAppDomain;
        hr = pUnkDomain->QueryInterface(__uuidof(_AppDomain), (void**)&pAppDomain);
        pUnkDomain->Release();
        if (FAILED(hr) || !pAppDomain) continue;

        BSTR bstrName = nullptr;
        pAppDomain->get_FriendlyName(&bstrName); // отримання імені збірки
        wcout << bstrName << endl;

        if (bstrName && wcscmp(bstrName, targetName) == 0) { // якщо збірка знайдена
            found = true;

            std::vector<BYTE> assemblyData = ReadFileToBytes(
                L"D:\\Main\\Project\\ClassLibrary1\\ClassLibrary1\\bin\\x86\\Release\\" // зчитування
                бібліотеки
                L"ClassLibrary1.dll");
            SAFEARRAYBOUND bounds[1];
            bounds[0].lLbound = 0;
            bounds[0].cElements = static_cast<ULONG>(assemblyData.size());

            SAFEARRAY* pSafeArray = SafeArrayCreate(VT_UI1, 1, bounds);
            if (pSafeArray) {
                for (LONG i = 0; i < assemblyData.size(); i++) {
                    SafeArrayPutElement(pSafeArray, &i, &assemblyData[i]);
                }

                _Assembly* pUnkAssembly;

                hr = pAppDomain->Load_3(pSafeArray, &pUnkAssembly); // завантаження
                бібліотеки

                if (SUCCEEDED(hr)) {
                    CComPtr<_Assembly> pAssembly;
                    hr = pUnkAssembly->QueryInterface(__uuidof(_Assembly),
                    (void**)&pAssembly);

                    if (!SUCCEEDED(hr)) {
                        wprintf(L"Failed to get asm: 0x%x\n", hr);
                        return;
                    }

                    std::cout << "C# DLL injected successfully!" << std::endl;
                    std::cout << pUnkAssembly << std::endl;
                    CComPtr<_MethodInfo> method;
                    CComPtr<_Type> pType;
                    BSTR asmName;
                    pAssembly->get_FullName(&asmName);
                    wcout << asmName << endl;
                    SAFEARRAY* pSafeArray = nullptr;
                    hr = pAssembly->GetTypes(&pSafeArray); // пошук класів бібліотеки
                    if (FAILED(hr) || pSafeArray == nullptr) {

```

		Петренко Д.В.			ККІК.КР.Б – 125 – 16 – ПЗ	Арк.
		Трокоз Є.М.				47
Змн.	Арк.	№ докум.	Підпис	Дата		

```

std::endl;

std::cout << "Failed to get types. HRESULT: " << hr <<

}

long lowerBound, upperBound; // Розмір масива
SafeArrayGetLBound(pSafeArray, 1, &lowerBound);
SafeArrayGetUBound(pSafeArray, 1, &upperBound);

for (long i = lowerBound; i <= upperBound; ++i) {

    hr = SafeArrayGetElement(pSafeArray, &i, &pType);
    if (SUCCEEDED(hr) && pType) {
        BSTR typeName;
        hr = pType->get_FullName(&typeName); // знаходження типів
        std::wcout << L" Global Type: " << typeName << std::endl;
        if (SUCCEEDED(hr)) {

            if (wcscmp(typeName, L"Patcher.MyClass") == 0) {
                SysFreeString(typeName);
                break;
            }
            SysFreeString(typeName);

        }
    }
}
SafeArrayDestroy(pSafeArray);

cout << pType << endl;
if (!SUCCEEDED(hr)) {
    wprintf(L"Failed to get type: 0x%x\n", hr);
    return;
}
GetMethodByName(L"Main", pType, method); // пошук методу
InvokeMethod(method); // виклик методу
}
else {
    wprintf(L"Failed to load assembly: 0x%x\n", hr);
}
}

break;
}
SysFreeString(bstrName);
}

if (!found) { // не знайдено збірку
    continue;
}

break;
}

if (hEnum) {
    pCorRuntimeHost->CloseEnum(hEnum);
}
pCorRuntimeHost->Release();
}

void InstallCorBindHook() {
    HRESULT hr;

    CoInitializeEx(nullptr, COINIT_MULTITHREADED);

```

		Петренко Д.В.			ККІК.КР.Б – 125 – 16 – ПЗ	Арк.
		Трокоз Є.М.				48
Змн.	Арк.	№ докум.	Підпис	Дата		

```

ICLRMetaHost* pMetaHost = nullptr;
ICLRRuntimeInfo* pRuntimeInfo = nullptr;
pCorHost = nullptr;

hr = CLRCreatInstance( // отримання ICLRMetaHost
    CLSID_CLRMetaHost,
    IID_ICLRMetaHost,
    reinterpret_cast<void**>(&pMetaHost)
);

hr = pMetaHost->GetRuntime( // Отримуємо ICLRRuntimeInfo
    L"v4.0.30319",
    IID_ICLRRuntimeInfo,
    reinterpret_cast<void**>(&pRuntimeInfo)
);

hr = pRuntimeInfo->GetInterface( // ICorRuntimeHost
    CLSID_CorRuntimeHost,
    IID_ICorRuntimeHost,
    reinterpret_cast<void**>(&pCorHost)
);

std::thread th(startFuncLoop, pCorHost);
th.detach();

}

void rundotnet() {
    InstallCorBindHook();
    return;
}

void main() {
    cout << "TEST" << endl;
}

BOOL APIENTRY DllMain( HMODULE hModule,
    DWORD ul_reason_for_call,
    LPVOID lpReserved
)
{
    if (ul_reason_for_call == DLL_PROCESS_ATTACH)
    {
        ConsoleSetup();
        rundotnet();
    }
    else if (ul_reason_for_call == DLL_PROCESS_DETACH)
    {
    }

    return TRUE;
}

```

		Петренко Д.В.			ККІК.КР.Б – 125 – 16 – ПЗ	Арк.
		Трокоз Є.М.				49
Змн.	Арк.	№ докум.	Підпис	Дата		

Лістинг MainClass.cs бібліотеки-аналізатора на C#

```

using System;
using System.Collections;
using System.Reflection;
using System.Runtime.InteropServices;
using HarmonyLib;

namespace Patcher
{
    //internal struct TrpEb1KBfSYOhvKQAgk
    //{
    //    // Token: 0x0400011D RID: 285
    //    internal bool QlvKaoglnL;
    //    // Token: 0x0400011E RID: 286
    //    internal byte[] aQAK3DsNdu;
    //}

    public static class MyClass
    {
        public static Assembly deps = null;
        public static Hashtable fieldValue = null;
        public static bool MethodPatch(object[] __args)
        {
            IntPtr ptr = (IntPtr)__args[2]; // копія оригінальної структури програми
            long num;
            if (IntPtr.Size == 4)
            {
                num = (long)Marshal.ReadInt32(ptr, IntPtr.Size * 2);
            }
            else
            {
                num = Marshal.ReadInt64(ptr, IntPtr.Size * 2);
            }
            Console.WriteLine("Method compile");
            object obj = fieldValue[num]; // об'єкт з fieldValue по ключу

            if (obj != null)
            {
                var structType = obj.GetType();

                byte[] data = (byte[])structType // оригінальні байти
                    .GetField("aQAK3DsNdu", BindingFlags.Instance | BindingFlags.NonPublic)
                    .GetValue(obj);
                Console.WriteLine(BitConverter.ToString(data).Replace("-", " "));
            }
            return true;
        }

        public static void Main()
        {
            Assembly[] asms = null;
            deps = null;

            AppDomain currentDomain = AppDomain.CurrentDomain;

            asms = currentDomain.GetAssemblies();
            foreach (Assembly a in asms) // пошук збірки з compileMethod
            {
                Console.WriteLine(a.FullName);
                if (a.GetName().Name == "DelphSystem")
                {

```

		Петренко Д.В.			ККІК.КР.Б – 125 – 16 – ПЗ	Арк.
		Трокоз Є.М.				50
Змн.	Арк.	№ докум.	Підпис	Дата		

```

        deps = a;
    }
}
MethodInfo method = null;
Module module = deps.GetModules()[0]; // 1й модуль
Console.WriteLine(module.Name);
Type type = null;
var types = module.GetTypes();
foreach (Type t in types) {
    if (t.Name == "kqxpCudLW4A8MejDN02") { // класс з методом
        type = t;
        Console.WriteLine("Found");
        break;
    }
}
var methods = type.GetMethods(BindingFlags.NonPublic
    | BindingFlags.Static | BindingFlags.Instance);

foreach (MethodInfo mi in methods) {

    if (mi.Name == "itZb5pRET8") { // compileMethod
        Console.WriteLine($"{mi.Name} - Found");
        method = mi;
    }
}
int token = 0x0400010F;
// поле s1nAo4dHyXWpbIOVQIc.kqxpCudLW4A8MejDN02.DdSKEXryOx за токеном;
FieldInfo fi;
fi = deps.GetModules()[0].ResolveField(token);
fieldValue = (Hashtable)fi.GetValue(null);
Console.WriteLine($"Found Field: {fi.Name}, Value = {fieldValue}");

try // патч метода
{
    var harmony = new Harmony("com.example.patch");
    var patch = new HarmonyMethod(typeof(MyClass).GetMethod("MethodPatch"));
    var result = harmony.Patch(method, prefix: patch);
    Console.WriteLine(result.Name);
}
catch (Exception e) {
    Console.WriteLine(e.Message);
}
}
}

namespace Library {
    public static class EntryPointClass {
        public static void EntryPoint() {
            Patcher.MyClass.Main();
        }
    }
}

```

		Петренко Д.В.			ККІК.КР.Б – 125 – 16 – ПЗ	Арк.
		Трокоз Є.М.				51
Змн.	Арк.	№ докум.	Підпис	Дата		

Повний результат виконання програми:

C# DLL injected successfully!
 0B54FFD0
 ClassLibrary1, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null
 Global Type: Library.EntryPointClass
 Global Type: Patcher.TrpEb1KBfSYOhvKQAgk
 Global Type: Patcher.MyClass
 0B54FEB8
 Patcher.MyClass
 Number of methods in types: 6
 Global Method Found: MethodPatch
 Global Method Found: Main
 Method Found: Main
 mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
 DomainName, Version=13.0.7.0, Culture=neutral, PublicKeyToken=null
 System, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
 Microsoft.VisualBasic, Version=10.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a
 System.Core, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
 DelphSystem, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null
 System.Windows.Forms, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
 System.Drawing, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a
 Newtonsoft.Json, Version=13.0.0.0, Culture=neutral, PublicKeyToken=30ad4fe6b2a6aeed
 PresentationFramework, Version=4.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35
 WindowsBase, Version=4.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35
 PresentationCore, Version=4.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35
 System.Xaml, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
 System.Configuration, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a
 System.Xml, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
 PresentationFramework.Aero2, Version=4.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35
 FontAwesome.WPF, Version=4.7.0.37774, Culture=neutral, PublicKeyToken=0758b07a11a4f466
 PresentationFramework-SystemXml, Version=4.0.0.0, Culture=neutral,
 PublicKeyToken=b77a5c561934e089
 UIAutomationTypes, Version=4.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35
 UIAutomationProvider, Version=4.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35
 System.Management, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a
 PhotonTucker, Version=0.0.0.0, Culture=neutral, PublicKeyToken=null
 ClassLibrary1, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null
 0Harmony, Version=2.3.6.0, Culture=neutral, PublicKeyToken=null
 DelphSystem.dll
 Found
 itZb5pRET8 - Found
 Found Field: DdSKEXryOx, Value = System.Collections.Hashtable
 s1nAo4dHyXWpbIOVQIc.kqxpCudLW4A8MejDNO2.itZb5pRET8_Patch1
 Method executed successfully, but the result type is not BSTR.
 Method compile
 Method compile
 Method compile
 2B 05 28 87 CC 43 3B 02 03 7D D0 00 00 04 2A
 Method compile

		Петренко Д.В.			ККІК.КР.Б – 125 – 16 – ПЗ	Арк.
		Трокоз Є.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		52

Method compile

Method compile

2B 05 28 86 3D 63 2F 73 40 00 00 0A 0A 20 AB 72 9B 60 20 05 00 00 00 62 20 F0 0D 84 70 61 7E
CB 01 00 04 7B 14 02 00 04 61 7E 33 02 00 04 28 28 07 00 06 73 3D 00 00 0A 0B 07 7E BD 02 00 04
28 50 09 00 06 39 F7 01 00 00 07 7E BD 02 00 04 28 50 09 00 06 6F 32 00 00 0A 0C 38 AA 01 00 00
08 7E 65 02 00 04 28 F0 07 00 06 0D 06 73 DC 01 00 06 25 09 20 86 12 78 0F 20 35 5C 53 35 61 7E
CB 01 00 04 7B FE 01 00 04 61 7E 33 02 00 04 28 28 07 00 06 7E 67 02 00 04 28 F8 07 00 06 7E 68
02 00 04 28 FC 07 00 06 7E E4 02 00 04 28 EC 09 00 06 25 09 20 BC EC 50 5C 20 CA AD 4E 88 61
20 6F 6D CB F6 61 7E CB 01 00 04 7B EC 01 00 04 61 7E 33 02 00 04 28 28 07 00 06 7E 67 02 00 04
28 F8 07 00 06 7E 68 02 00 04 28 FC 07 00 06 7E E5 02 00 04 28 F0 09 00 06 25 09 20 CB FF 1D 95
20 03 00 00 00 62 20 D3 B1 7A C9 61 7E CB 01 00 04 7B F6 01 00 04 61 7E 33 02 00 04 28 28 07 00
06 7E 67 02 00 04 28 F8 07 00 06 7E E6 02 00 04 28 F4 09 00 06 7E E7 02 00 04 28 F8 09 00 06 25 09
20 E3 7D DE 6E 20 B9 C2 A9 04 61 7E CB 01 00 04 7B FF 01 00 04 61 7E 33 02 00 04 28 28 07 00
06 7E 67 02 00 04 28 F8 07 00 06 7E 68 02 00 04 28 FC 07 00 06 7E E8 02 00 04 28 FC 09 00 06 25
09 20 22 A8 12 78 65 20 7B 23 3D F3 61 7E CB 01 00 04 7B FC 01 00 04 61 7E 33 02 00 04 28 28 07
00 06 7E 67 02 00 04 28 F8 07 00 06 7E 68 02 00 04 28 FC 07 00 06 7E E9 02 00 04 28 00 0A 00 06
25 09 20 78 97 76 CE 20 02 00 00 00 62 20 97 E3 D4 2D 61 7E CB 01 00 04 7B 09 02 00 04 61 7E 33
02 00 04 28 28 07 00 06 7E 67 02 00 04 28 F8 07 00 06 7E 68 02 00 04 28 FC 07 00 06 7E EA 02 00
04 28 04 0A 00 06 6F 42 00 00 0A 08 7E 69 02 00 04 28 00 08 00 06 3A 46 FE FF FF DD 12 00 00 00
08 39 0B 00 00 00 08 7E 3C 02 00 04 28 4C 07 00 06 DC 07 7E C3 02 00 04 28 68 09 00 06 38 5F 00
00 00 7E D7 02 00 04 28 B8 09 00 06 20 8C AC B0 DB 20 06 00 00 00 62 20 82 DD 2B D9 61 7E CB
01 00 04 7B C5 01 00 04 61 7E 33 02 00 04 28 28 07 00 06 20 79 4F 3C 93 20 F0 4A 5E B8 61 7E CB
01 00 04 7B E8 01 00 04 61 7E 33 02 00 04 28 28 07 00 06 16 1F 10 16 7E DA 02 00 04 28 C4 09 00
06 26 DD 52 00 00 00 13 04 7E D7 02 00 04 28 B8 09 00 06 11 04 7E D9 02 00 04 28 C0 09 00 06 20
68 27 D2 23 20 75 E5 C3 0D 59 20 42 88 63 06 61 7E CB 01 00 04 7B 13 02 00 04 61 7E 33 02 00 04
28 28 07 00 06 16 1F 10 16 7E DA 02 00 04 28 C4 09 00 06 26 DD 00 00 00 00 06 2A

Method compile

Method compile

2B 05 28 F3 BC 34 69 7E 2D 02 00 04 28 10 07 00 06 38 00 00 00 00 7E 2E 02 00 04 28 14 07 00 06
38 00 00 00 00 02 28 12 00 00 0A 20 00 00 00 00 7E CB 01 00 04 7B 1F 02 00 04 39 14 00 00 00 26
20 00 00 00 00 38 09 00 00 00 38 B8 FF FF FF FE 0C 00 00 45 01 00 00 00 05 00 00 00 38 00 00 00 00
2A

Method compile

Method compile

Method compile

2B 05 28 6B 34 30 36 20 01 00 00 00 FE 0E 00 00 38 00 00 00 00 FE 0C 00 00 45 02 00 00 00 05 00
00 00 06 00 00 00 38 00 00 00 00 2A 02 03 7D C0 00 00 04 20 00 00 00 00 7E CB 01 00 04 7B 15 02
00 04 3A D2 FF FF FF 26 20 00 00 00 00 38 C7 FF FF FF

Method compile

Method compile

Method compile

2B 05 28 7F 88 36 41 20 01 00 00 00 FE 0E 00 00 38 00 00 00 00 FE 0C 00 00 45 02 00 00 00 2B 00
00 00 05 00 00 00 38 26 00 00 00 02 03 7D C1 00 00 04 20 00 00 00 00 7E CB 01 00 04 7B CD 01 00
04 3A D3 FF FF FF 26 20 00 00 00 00 38 C8 FF FF FF 2A

Method compile

Method compile

Method compile

2B 05 28 1A A4 61 41 20 01 00 00 00 FE 0E 01 00 38 00 00 00 00 FE 0C 01 00 45 02 00 00 00 05 00
00 00 12 00 00 00 38 00 00 00 00 12 00 7E 63 02 00 04 28 E8 07 00 06 2A 02 7E 68 02 00 04 28 FC 07
00 06 7E EB 02 00 04 28 08 0A 00 06 20 00 04 00 00 6A 5C 20 00 04 00 00 6A 5C 69 13 00 20 00 00
00 00 7E CB 01 00 04 7B 14 02 00 04 39 A7 FF FF FF 26 20 00 00 00 00 38 9C FF FF FF

Method compile

		Петренко Д.В.			ККІК.КР.Б – 125 – 16 – ПЗ	Арк.
		Трокоз Є.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		53

Method compile

Method compile

Method compile

Method compile

2B 05 28 70 DC 73 2E 20 01 00 00 00 FE 0E 00 00 38 00 00 00 00 FE 0C 00 00 45 02 00 00 00 05 00
00 00 06 00 00 00 38 00 00 00 00 2A 02 03 7D C2 00 00 04 20 00 00 00 00 7E CB 01 00 04 7B 19 02
00 04 3A D2 FF FF FF 26 20 00 00 00 00 38 C7 FF FF FF

Method compile

Method compile

Method compile

2B 05 28 62 D7 34 40 20 01 00 00 00 FE 0E 00 00 38 00 00 00 00 FE 0C 00 00 45 02 00 00 00 2B 00
00 00 05 00 00 00 38 26 00 00 00 02 03 7D C3 00 00 04 20 00 00 00 00 7E CB 01 00 04 7B 11 02 00
04 3A D3 FF FF FF 26 20 00 00 00 00 38 C8 FF FF FF 2A

Method compile

Method compile

Method compile

2B 05 28 F6 8C 17 3A 20 01 00 00 00 FE 0E 00 00 38 00 00 00 00 FE 0C 00 00 45 02 00 00 00 2B 00
00 00 05 00 00 00 38 26 00 00 00 02 03 7D C4 00 00 04 20 00 00 00 00 7E CB 01 00 04 7B F6 01 00
04 39 D3 FF FF FF 26 20 00 00 00 00 38 C8 FF FF FF 2A

Method compile

Method compile

Method compile

2B 05 28 73 DC 64 38 20 01 00 00 00 FE 0E 00 00 38 00 00 00 00 FE 0C 00 00 45 02 00 00 00 2B 00
00 00 05 00 00 00 38 26 00 00 00 02 03 7D C5 00 00 04 20 00 00 00 00 7E CB 01 00 04 7B 1A 02 00
04 3A D3 FF FF FF 26 20 00 00 00 00 38 C8 FF FF FF 2A

Method compile

Method compile

Method compile

2B 05 28 8B 71 61 51 02 03 7D D1 00 00 04 2A

Method compile

Method compile

Method compile

2B 05 28 DF 6F 38 45 20 01 00 00 00 FE 0E 02 00 38 00 00 00 00 FE 0C 02 00 45 03 00 00 00 08 00
00 00 F1 03 00 00 05 00 00 00 38 03 00 00 00 11 03 2A 00 11 03 20 6E 47 CA 72 20 A5 6C 08 7D 61
7E CB 01 00 04 7B F2 01 00 04 61 7E 33 02 00 04 28 28 07 00 06 20 92 27 A0 C4 65 20 30 29 5F 0E
61 7E CB 01 00 04 7B C5 01 00 04 61 7E 33 02 00 04 28 28 07 00 06 7E DC 02 00 04 28 CC 09 00 06
7E DD 02 00 04 28 D0 09 00 06 20 04 00 00 00 7E CB 01 00 04 7B D2 01 00 04 3A 0F 00 00 00 26 20
02 00 00 00 38 04 00 00 00 FE 0C 00 00 45 07 00 00 00 C2 01 00 00 61 01 00 00 E6 00 00 00 89 02 00
00 6B 00 00 00 29 02 00 00 05 00 00 00 38 BD 01 00 00 11 03 20 5E 44 CF 91 20 B2 64 CD F0 61 7E
CB 01 00 04 7B DD 01 00 04 61 7E 33 02 00 04 28 28 07 00 06 20 A3 9D 09 D5 20 13 99 05 6C 61 20
9C 02 6D D7 61 7E CB 01 00 04 7B 1A 02 00 04 61 7E 33 02 00 04 28 28 07 00 06 7E DC 02 00 04 28
CC 09 00 06 7E E2 02 00 04 28 E4 09 00 06 20 05 00 00 00 38 74 FF FF FF 11 03 20 E3 CE 2F B6 20
ED 67 5C 88 61 7E CB 01 00 04 7B D8 01 00 04 61 7E 33 02 00 04 28 28 07 00 06 20 AB 7E DF 14
20 09 42 1E 40 58 20 59 0D 9B 1E 61 7E CB 01 00 04 7B C7 01 00 04 61 7E 33 02 00 04 28 28 07 00
06 7E DC 02 00 04 28 CC 09 00 06 7E DE 02 00 04 28 D4 09 00 06 20 00 00 00 00 7E CB 01 00 04 7B
EC 01 00 04 39 04 FF FF FF 26 20 01 00 00 00 38 F9 FE FF FF 11 03 20 CB DB C9 BC 20 54 93 E2
86 61 7E CB 01 00 04 7B FE 01 00 04 61 7E 33 02 00 04 28 28 07 00 06 20 F4 61 F8 0C 20 02 00 00
00 62 20 8A 6F 2F 49 61 7E CB 01 00 04 7B 05 02 00 04 61 7E 33 02 00 04 28 28 07 00 06 7E DC 02
00 04 28 CC 09 00 06 7E E0 02 00 04 28 DC 09 00 06 20 00 00 00 00 7E CB 01 00 04 7B CC 01 00 04
39 89 FE FF FF 26 20 00 00 00 00 38 7E FE FF FF 11 03 20 D6 5C 15 20 20 FD ED 1B 34 61 7E CB
01 00 04 7B 09 02 00 04 61 7E 33 02 00 04 28 28 07 00 06 20 8F A5 5E DD 65 20 52 8A 85 22 61 7E
CB 01 00 04 7B C0 01 00 04 61 7E 33 02 00 04 28 28 07 00 06 7E DC 02 00 04 28 CC 09 00 06 7E DF

		Петренко Д.В.			ККІК.КР.Б – 125 – 16 – ПЗ	Арк.
		Трокоз Є.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		54

02 00 04 28 D8 09 00 06 20 02 00 00 00 38 1D FE FF FF 11 03 20 8C B3 60 14 65 20 22 BA F4 B8 61
7E CB 01 00 04 7B 06 02 00 04 61 7E 33 02 00 04 28 28 07 00 06 20 DE 55 9A 2A 20 02 00 00 00 63
20 38 EE 76 7E 61 7E CB 01 00 04 7B FC 01 00 04 61 7E 33 02 00 04 28 28 07 00 06 7E DC 02 00 04
28 CC 09 00 06 7E E1 02 00 04 28 E0 09 00 06 20 06 00 00 00 38 B6 FD FF FF 11 03 20 E4 9A AE 13
20 F5 7D 75 61 61 7E CB 01 00 04 7B D4 01 00 04 61 7E 33 02 00 04 28 28 07 00 06 20 5D D7 4C 76
20 C7 C4 76 03 61 7E CB 01 00 04 7B F8 01 00 04 61 7E 33 02 00 04 28 28 07 00 06 7E DC 02 00 04
28 CC 09 00 06 7E E3 02 00 04 28 E8 09 00 06 20 03 00 00 00 38 56 FD FF FF DD D3 FC FF FF 13
01 20 01 00 00 00 7E CB 01 00 04 7B D5 01 00 04 3A 0F 00 00 00 26 20 01 00 00 00 38 04 00 00 00
FE 0C 04 00 45 02 00 00 00 69 00 00 00 05 00 00 00 38 64 00 00 00 7E D7 02 00 04 28 B8 09 00 06 11
01 7E D9 02 00 04 28 C0 09 00 06 20 52 3E B6 A6 20 89 B3 56 A7 61 7E CB 01 00 04 7B 18 02 00 04
61 7E 33 02 00 04 28 28 07 00 06 16 1F 10 16 7E DA 02 00 04 28 C4 09 00 06 26 20 00 00 00 00 7E
CB 01 00 04 7B E5 01 00 04 39 95 FF FF FF 26 20 00 00 00 00 38 8A FF FF FF DD 33 FC FF FF 20
01 00 00 00 7E CB 01 00 04 7B D9 01 00 04 3A 09 FC FF FF 26 20 02 00 00 00 38 FE FB FF FF 73
CC 01 00 06 13 03 20 00 00 00 00 7E CB 01 00 04 7B BD 01 00 04 3A E3 FB FF FF 26 20 00 00 00 00
38 D8 FB FF FF

Method compile

Method compile

2B 05 28 74 08 2D 5D 7E 2D 02 00 04 28 10 07 00 06 38 00 00 00 00 7E 2E 02 00 04 28 14 07 00 06
38 00 00 00 00 02 28 12 00 00 0A 20 00 00 00 00 7E CB 01 00 04 7B ED 01 00 04 3A 14 00 00 00 26
20 00 00 00 00 38 09 00 00 00 38 B8 FF FF FF FE 0C 00 00 45 01 00 00 00 05 00 00 00 38 00 00 00 00
2A

Method compile

Method compile

Method compile

2B 05 28 9F F2 2C 51 20 01 00 00 00 FE 0E 00 00 38 00 00 00 00 FE 0C 00 00 45 02 00 00 00 2B 00
00 00 05 00 00 00 38 26 00 00 00 02 03 7D B8 00 00 04 20 00 00 00 00 7E CB 01 00 04 7B 0C 02 00
04 3A D3 FF FF FF 26 20 00 00 00 00 38 C8 FF FF FF 2A

Method compile

Method compile

Method compile

2B 05 28 EA 3D 65 5C 20 01 00 00 00 FE 0E 00 00 38 00 00 00 00 FE 0C 00 00 45 02 00 00 00 2B 00
00 00 05 00 00 00 38 26 00 00 00 02 03 7D B9 00 00 04 20 00 00 00 00 7E CB 01 00 04 7B 0E 02 00
04 3A D3 FF FF FF 26 20 00 00 00 00 38 C8 FF FF FF 2A

		Петренко Д.В.			ККІК.КР.Б – 125 – 16 – ПЗ	Арк.
		Трокоз Є.М.				
Змн.	Арк.	№ докум.	Підпис	Дата		55