Вінницький національний технічний університет
(повне найменування вищого навчального закладу)
Факультет інформаційних технологій та комп'ютерної інженерії
(повне найменування інституту, назва факультету (відділення))

Кафедра програмного забезпечення
(повна назва кафедри (предметної, циклової комісії)

Пояснювальна записка

до магістерської кваліфікаційної роботи
магістр
(освітньо-кваліфікаційний рівень)

на тему: «Розробка методу і засобів системи захисту даних з використанням технології динамічного завантаження коду в робочий процес»

Виконав: студент 2 курсу
групи 2ПІ-18м
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)

Андреєв А.О.
(прізвище та ініціали)

Керівник Войтко В.В.
(прізвище та ініціали)

Рецензент _____.
(прізвище та ініціали)

Вінниця − 2019 року

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Освітньо-кваліфікаційний рівень – магістр
Спеціальність 121 - «Інженерія програмного забезпечення»

УЗГОДЖЕНО
Керівник ТОВ «Он-лайн»
Варшавський Р.В.
“____”_____20___року

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
“____”_____20___року

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Андреєву Андрію Олександровичу

1. Тема роботи: «Розробка методу і засобів системи захисту даних з використанням технології динамічного завантаження коду в робочий процес» керівник роботи: Войтко Вікторія Володимирівна, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від “___”_____20__ року №___

2. Строк подання студентом роботи_____

3. Вихідні дані до роботи : Операційна система – Windows
Мови програмування – C++
Бібліотеки – cURL, STD

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз стану питання та постановка задач дослідження; розробка методу, моделі та алгоритмів системи динамічного завантаження; розробка програмних засобів для пакування виконавчих файлів; тестування програми; економічна частина; висновки; перелік посилань; додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень): технології; моделі системи; модель взаємодії процесів; схема виконання програми з DllLoadManager; рекомендації до використання; алгоритм роботи системи захисту; файл конфігурації; тестування розробленої програми.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Войтко В.В., к.т.н, доцент кафедри ПЗ		
5	Бальзан М.В., к.е.н., доцент кафедри ЕПВМ		

7. Дата видачі завдання_____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз завдання і вибір методів його вирішення	01.10.19 – 11.10.19	Вик.
2	Аналіз існуючих аналогів та постановка задач дослідження	12.10.19 – 17.10.19	Вик.
3	Розробка методу динамічного завантаження	18.10.19 – 01.11.19	Вик.
4	Розробка структур і алгоритмів програмного продукту	02.11.19 – 10.11.19	Вик.
5	Розробка програмного забезпечення	11.11.19 – 21.11.19	Вик.
6	Тестування розробленого програмного продукту	22.11.19 – 25.11.19	Вик.
7	Економічна частина	26.11.19 – 30.11.19	Вик.
8	Оформлення матеріалів до захисту МКР	01.12.19 – 08.12.19	Вик.

Студент _____ **Андреєв А.О.**
(підпис) (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи _____ **Войтко В.В.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Магістерська кваліфіаційна робота спрямована на дослідження процесу захисту програмного забезпечення від несанкціонованого доступу.

У магістерській кваліфікаційній роботі удосконалено метод захисту від несанкціонованого доступу шляхом динамічного завантаження виконуваного коду в робочий процес. Запропонований метод реалізовано в системі захисту програмних додатків «DllLoadManager». Розроблена система призначена для завантаження динамічно завантажуваних бібліотек у процес, що, в свою чергу, забезпечує захист виконуваних файлів від викрадення.

Створений програмний продукт написаний на мові програмування C++ під операційну систему MS Windows з використанням бібліотеки з відкритим доступом libcurl. Характеризується надійністю та швидкістю роботи, а також широким спектром можливих застосувань.

ABSTRACT

The master's qualification work is aimed at investigating the process of protection of software against unauthorized access.

The master's qualification work has improved the method of protection against unauthorized access by dynamically loading executable code into the workflow. The proposed method is implemented in the system protection software "DllLoadManager". The developed system is intended for loading of dynamically loaded libraries in process, which, in turn, provides protection of executable files from theft.

Created software written in C ++ programming language for MS Windows using libcurl public library. It is characterized by reliability and speed of operation, as well as a wide range of possible applications.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Зі збільшенням розповсюдженості комерційних програмних рішень все гостріше постає проблема захисту даних від несанкціонованого копіювання та викрадення [1]. Несанкціоновані копії програмних продуктів завдають значних збитків компаніям-розробникам.

Захист програмного продукту від несанкціонованого копіювання – актуальна задача у зв'язку зі збереженням комерційних і авторських прав фірм і розробників. За висновками закордонних фахівців, економічний збиток від "піратського" копіювання програмного забезпечення складає мільярди доларів[2].

Жоден з методів не дозволяє повністю захистити програмні продукти від копіювання, але може зробити процес копіювання настільки складним та довготривалим, що копіювання буде недоцільним для зловмисника. Пошуки нових методів захисту програмного забезпечення активно ведуться IT-спеціалістами [1-3]. Тому актуальною є розробка системи захисту програмних додатків від несанкціонованого доступу шляхом динамічного завантаження програмного продукту або його частин з метою забезпечення високого рівня захисту.

Мета та завдання дослідження. Метою роботи є підвищення захищеності програмних додатків від несанкціонованого копіювання шляхом розробки динамічно завантажуваної бібліотеки та модуля для управління процесом користувацького додатку, що підвищить захист програм від несанкціонованого доступу.

У відповідності до поставленої мети потрібно виконати такі **задачі**:

– удосконалити метод динамічного завантаження програмного продукту або його частин та розробити модель і алгоритми системи захисту програмних додатків;

– розробити динамічно завантажувану бібліотеку, що буде завантажуватися у виконуваний процес та структуровано копіювати вхідну користувацьку бібліотеку;

– розробити модуль для завантаження користувацької бібліотеки з сервера;

– розробити модуль для управління процесом для користувацького додатку;

– провести тестування програмного продукту.

Виконання перелічених завдань у повному обсязі зможе продемонструвати можливості методу динамічного завантаження програмного продукту, а також покаже його перспективність використання в системах захисту програмних продуктів.

Розроблена система може бути впроваджена на фірмах з розробки програмного забезпечення. На даний час вона використовується у ТОВ «Он-лайн», акт впровадження наведений у додатку А.

Зв’язок роботи з науковими програмами, планами, темами: робота виконувалася відповідно до плану науково-дослідних робіт кафедри програмного забезпечення.

Об’єктом дослідження є процес захисту від несанкціонованого доступу.

Предметом дослідження є засоби захисту програмних додатків від несанкціонованого копіювання.

Методи дослідження:

– теорія захисту інформації для розробки методу захисту програмного забезпечення;

– методи теорії алгоритмів для розробки алгоритмів і розробки програмного забезпечення.

– комп’ютерне моделювання для аналізу та перевірки достовірності отриманих теоретичних положень.

Наукова новизна одержаних результатів:

1. Подальшого розвитку дістав метод динамічного завантаження програмного продукту або його частин, який, на відміну від існуючих,

передбачає збереження виконуваного коду на сервері, а не на локальній машині користувача, та динамічне завантаження бібліотек у робочий процес, що забезпечує високий рівень захисту виконуваних файлів від несанкціонованого копіювання.

2. Подальшого розвитку дістала модель системи захисту програмних продуктів від несанкціонованого доступу, яка, на відміну від існуючих, орієнтована на кодову ін'єкцію завантажуваних бібліотек у виконуваний процес, що дозволяє динамічно забезпечувати захист програм в робочому режимі.

Практична цінність отриманих результатів. Практична цінність отриманих результатів полягає у забезпеченні захищеності програмних рішень від несанкціонованого доступу до виконуваних файлів, а також їх копіювання.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконувалася відповідно до плану науково-дослідних робіт кафедри програмного забезпечення.

Особистий внесок здобувача. Усі наукові результати, викладені у магістерській кваліфікаційній роботі, отримані автором особисто. У роботах, опублікованих у співавторстві, здобувачу належить розроблений метод і модель системи захисту програмних продуктів від несанкціонованого доступу, алгоритми роботи системи та приклади, що ілюструють її роботу по захисту розроблених програм.

Достовірність теоретичних положень підтверджена результатами тестування розробленого додатку.

Апробація матеріалів магістерської кваліфікаційної роботи. Результати роботи доповідалися на трьох науково-технічних конференціях:

- міжвузівському студентському вебінарі «Інноваційні та інформаційні технології в бізнесі та освіті». – Вінниця: ВТЕІ КНТЕУ. – 21 жовтня 2015 р.
- міжнародній науково-практичній Інтернет-конференції «Електронні інформаційні ресурси: створення, використання, доступ – 2015». – Вінниця: ВНТУ, 2015 р.;

- міжнародній науково-практичній Інтернет-конференції «Електронні інформаційні ресурси: створення, використання, доступ – 2018». – Вінниця: ВНТУ, січень 2018 р.

Публікації. Результати роботи були опубліковані в чотирьох наукових працях :

- стаття в збірнику праць міжнародної науково-практичної Інтернет-конференції «Електронні інформаційні ресурси: створення, використання, доступ - 2018» [4].
- тези доповіді у матеріалах міжвузівського студентського вебінару «Інноваційні та інформаційні технології в бізнесі та освіті». – Вінниця: ВТЕІ КНТЕУ. – 21 жовтня 2015 р. [5].
- стаття в збірнику праць міжнародної науково-практичної інтернет конференції «Електронні інформаційні ресурси: створення, використання, доступ - 2015» [6].
- свідоцтво про реєстрацію авторського права на комп'ютерну програму № 64732 [7].

Результати роботи були подані на Всеукраїнський та Міжнародний студентські конкурси, де отримано призові місця:

- диплом переможця ІІ ступеня у Всеукраїнському конкурсі студентських наукових робіт з напрямку «Інформатика і кібернетика»
- диплом за зайняте ІІ місце у Фіналі Міжнародної студентської олімпіади з інформаційних технологій IT-UNIVERSE в конкурсі «Кращий диплом з кібербезпеки».

Структура та обсяг роботи. Магістерська кваліфікаційна робота складається зі вступу, п'яти розділів, висновків, списку літератури, що містить 20 найменувань, 6 додатків. Робота містить 21 ілюстрацію, 8 таблиць.

1 АНАЛІЗ СТАНУ ПИТАННЯ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1. Аналіз методів захисту програмних додатків

Сьогодні комп’ютерне піратство є однією з головних проблем у світовій програмній індустрії. За деякими підрахунками, загальна сума збитків, яких воно завдає розробникам програмного забезпечення, сягає десятків мільярдів доларів на рік [2]. У зв’язку з цим зусиллями багатьох вчених та працівників галузі створено багато етичних, правових і технічних рішень для захисту програм [3]. До етичних та правових методів належать закони про авторське право, які сприяють усвідомленню користувачами неправомірності використання неліцензованого програмного забезпечення. Серед технічних виділяють апаратні, апаратно-програмні та програмні засоби. В апаратних засобах використовується спеціальне обладнання (наприклад, електронні ключі) або фізичні особливості носіїв інформації (CD, DVD тощо), для ідентифікації оригінальних версій програм і захисту продуктів від нелегального використання. Програмні засоби захисту реалізуються програмно, без використання фізичних характеристик носіїв інформації чи спеціального обладнання [8].

Найбільш поширеними методами несанкціонованого розповсюдження є створення копій виконуваних файлів, а також поширення ключів доступу до них, якщо такі є. Даний метод є суто копіюванням програми на інший носій, наприклад, електронний носій інформації, з метою подальшого поширення. Більшість програмних додатків вже мають захист проти такого методу незаконного розповсюдження. Наприклад, такий, при якому ключі доступу є одноразовими. В такому випадку зловмисники вдаються до модифікації виконуваного файлу шляхом декомпіляції та внесенням змін в механізми

програмного додатку, що відповідають за перевірку ключа, пароля чи інший захист, застосований розробником програми.

Декомпіляція — це перетворення виконуваного файлу з машинного (бінарного) коду або псевдокоду в текст на деякій мові програмування, в який можна вносити зміни. Даний метод також може бути застосованим для вивчення алгоритму захисту програмного продукту для створення власних ключів доступу.

Вдаючись до декомпіляції зловмисники виконують «злом» програмного продукту. Злом програмного забезпечення — це дії, спрямовані на усунення захисту програмного забезпечення, вбудованого розробниками для обмеження функціональних можливостей. Останнє необхідно для стимуляції покупки такого програмного забезпечення, після якої обмеження знімаються.

Більшість розробників програмного забезпечення забороняють в ліцензії декомпіляцію своїх програмних додатків, але згідно з законом України «Про авторське право і суміжні права» в статті 24 дозволяється вільне копіювання, модифікація і декомпіляція комп'ютерних програм у випадку, якщо особа правомірно володіє правомірно виготовленим примірником комп'ютерної програми. Тобто, у випадку, якщо особа стала власником примірника програмного забезпечення згідно механізму продажу і розповсюдження програмного додатку, передбаченим її розробником.

Захист від несанкціонованого копіювання – це створення засобів, що забезпечують можливість захисту від несанкціонованого створення копій виконуваних файлів, ресурсів та даних програмного засобу [3]. Методи захисту можна розділити на такі групи:

- прив’язка до дистрибутивного носія (дискети, CD або DVD-диску);
- прив’язка до комп’ютера;
- прив’язка до ключа;
- опитування довідників;
- обмеження на використання програмного забезпечення.

Методи можна класифікувати за способом розповсюдження захищеного програмного забезпечення та типом носія ліцензії.

1. Локальний програмний захист. Вимога введення серійного номера (ключа) при встановленні/запуску. Історія цього методу почалася тоді, коли програми поширювалися тільки на фізичних носіях (наприклад, компакт-дисках). На коробці з диском був надрукований серійний номер, що підходить тільки до даної копії програми.

З поширенням мереж, очевидним недоліком стала проблема поширення образів дисків і серійних номерів мережею. Тому зараз метод використовується тільки в сукупності з одним або більше інших методів.

2. Локальний мережевий програмний захист. Мережі виключає одночасний запуск двох програм з одним реєстраційним ключем на двох комп'ютерах в межах однієї локальної мережі.

Недолік в тому, що брандмауер можна налаштувати так, щоб він не пропускав пакети, що належать захищеній програмі. Правда, налаштування брандмауера вимагає деяких користувацьких навичок. Крім того, програми можуть взаємодіяти по мережі (наприклад, при організації мережевої гри). У цьому випадку брандмауер повинен пропускати такий трафік[1].

3. Глобальний мережевий програмний захист. Якщо програма працює з будь-яким централізованим сервером і без нього марна (наприклад, сервери онлайн-ігор, сервери оновлень антивірусів). Вона може передавати серверу свій серійний номер; якщо номер неправильний, сервер відмовляє в послузі. Недолік в тому, що, існує можливість створити сервер, який не робить такої перевірки.

4. Захист за допомогою компакт-дисків. Програма може вимагати оригінальний компакт-диск. Зокрема, такий спосіб застосовується в іграх. Стійкість таких захистів невелика, через широкий набір інструментів зняття образів компакт-дисків.

Як правило, цей спосіб захисту застосовується для захисту програм, записаних на цьому компакт-диску, що є одночасно ключовим:

– запис інформації в цих секторах;

– перевірка розташування та вмісту збійних секторів;

– перевірка швидкості читання окремих секторів.

Перші два методи практично марні через можливості зняття повного образу диска з використанням відповідного прикладного програмного забезпечення. Третій метод вважається більш надійним. Але існують програми, які можуть емулювати диски з урахуванням геометрії розташування даних, тим самим обходячи й цей захист. Іноді, в числі інших перевірок, також виконується перевірка можливості запису на вставлений диск. Якщо вона можлива, то диск вважається не ліцензійним. Однак, якщо образ буде записаний на диск CD-R, то зазначена перевірка пройде. Існує можливість приховати тип диска, щоб CD-R або CD-RW було видно як звичайний CD-ROM. Однак, в драйвер захисту може бути вбудована перевірка на наявність емуляції.

В даний час найбільшу популярність у світі мають системи захисту від копіювання SecuROM, StarForce, SafeDisc, CD-RX і Tages.[3]

Для багатьох програм зазначений метод захисту недоступний через відмінність способу розповсюдження (наприклад, shareware-програми).

5. Захист за допомогою електронних ключів. Сучасні електронні ключі

Електронний ключ (донгл), вставлений в один з портів комп'ютера (з інтерфейсом USB, LPT або COM) містить ключові дані, звані також ліцензією, записані в нього розробником:

– інформація для читання/запису (зараз практично не застосовується, оскільки після зчитування ключ може бути земульований)

– ключі апаратних криптографічних алгоритмів (використовується найчастіше)

– алгоритми, створені розробником програми (метод, що став доступним порівняно недавно, у зв'язку з появою електронних ключів з

мікропроцесором, здатним виконувати довільний код; в даний час використовується все частіше).

6. Прив'язка до параметрів комп'ютера й активація. Прив'язка до інформації про користувача/серійних номерів компонентів його комп'ютера і подальша активація програмного забезпечення в даний час використовується досить широко (приклад: ОС Windows).

У процесі встановлення програма підраховує код активації — контрольне значення, яке однозначно відповідає встановленим складовим комп'ютера і параметрам встановленої ОС. Це значення передається розробнику програми. На його основі розробник генерує ключ активації, який підходить для активації програми тільки на зазначеній машині (копіювання встановлених файлів на інший комп'ютер призведе до непрацездатності програми).

Перевага в тому, що не потрібно ніякого специфічного апаратного забезпечення, і програму можна поширювати за допомогою цифрової дистрибуції (через Інтернет).

Основний недолік: якщо користувач здійснює модернізацію комп'ютера (у випадку прив'язки до "заліза"), захист відмовляє. Автори багатьох програм у подібних випадках готові дати новий реєстраційний код. Наприклад, Microsoft в Windows XP дозволяє раз у 120 днів генерувати новий реєстраційний код.

Як прив'язки використовуються, в основному, серійний номер BIOS материнської плати, серійний номер вінчестера. В цілях приховування від користувача, дані про захист можуть розташовуватися в нерозміченій області жорсткого диска.

До недавнього часу такий захист розроблявся і впроваджувався розробниками самого програмного продукту. Проте зараз існують SDK для роботи з програмними ключами, наприклад HASP SL від компанії Аладдін Р. Д. Також все більшого поширення набувають сервіси, що пропонують одночасно функціонал навісного захисту і сервера активації.

7. Захист програм від копіювання шляхом перенесення їх в онлайн. Іншим напрямом захисту програм є використання підходу SaaS, тобто надання функціоналу цих програм (всього або частини), як сервісу. При цьому код програми розміщений і виконується на сервері, доступному в глобальній мережі. Доступ до нього здійснюється за принципом тонкого клієнта. Це один з небагатьох випадків, коли реалізується захист від копіювання.

Код виконується на «довіреній» стороні, звідки не може бути скопійований.

Однак, і тут виникає низка проблем, пов'язаних з безпекою:

– стійкість такого захисту залежить, насамперед, від захищеності серверів, на яких він виконується (йдеться про Інтернет-безпеку)

– важливе забезпечення конфіденційності запитів, аутентифікації користувачів, цілісності ресурсу (можливість «гарячого» резервування), і доступності рішення в цілому.

Виникають також питання довіри до сервісу (зокрема правові), оскільки йому фактично «у відкритому вигляді» передаються як саме програмне забезпечення, так і дані, які воно обробляє (наприклад, персональні дані користувачів).

8. Захист коду від аналізу. Можна виокремити засоби захисту безпосередньо коду програми від аналізу і використання в інших програмах. Зокрема, застосовуються обфускатори — програми потрібні для заплутування коду з метою захисту від його аналізу, модифікації та несанкціонованого використання

Програмні методи захисту є перспективними, оскільки для них характерні гнучкість у проектуванні, адаптованість до алгоритмів програм-об’єктів захисту та, порівняно з апаратними засобами, нижча вартість реалізації. Саме тому було вирішено розробити власний прототип захисту програмного забезпечення від викрадення, що може бути використаний для більшості програмних продуктів, що поширюються виконуваними файлами.

1.2 Порівняльний аналіз аналогів

На даний момент не існує поширених загальнодоступних методів захисту шляхом повного або часткового динамчного завантаження. Така тенденція прослідковується здебільшого тому, що такий підхід вимагає додаткових апаратних і системних ресурсів. Розглянемо найбільш поширені методи захисту програмного забезпечення він несанкціонованого копіювання.

VMProtect [8] (рисунок 1.1) – програма для захисту, що здобула популярність завдяки інноваційному підходу, що використовує. Захищені ділянки коду виконуються на віртуальній машині. Вбудований дизассемблер і підключення MAP файлу дозволять вибрати необхідні ділянки коду для захисту від злому. Програма може по-різному обробляти захищається код в залежності від обраного типу компіляції.

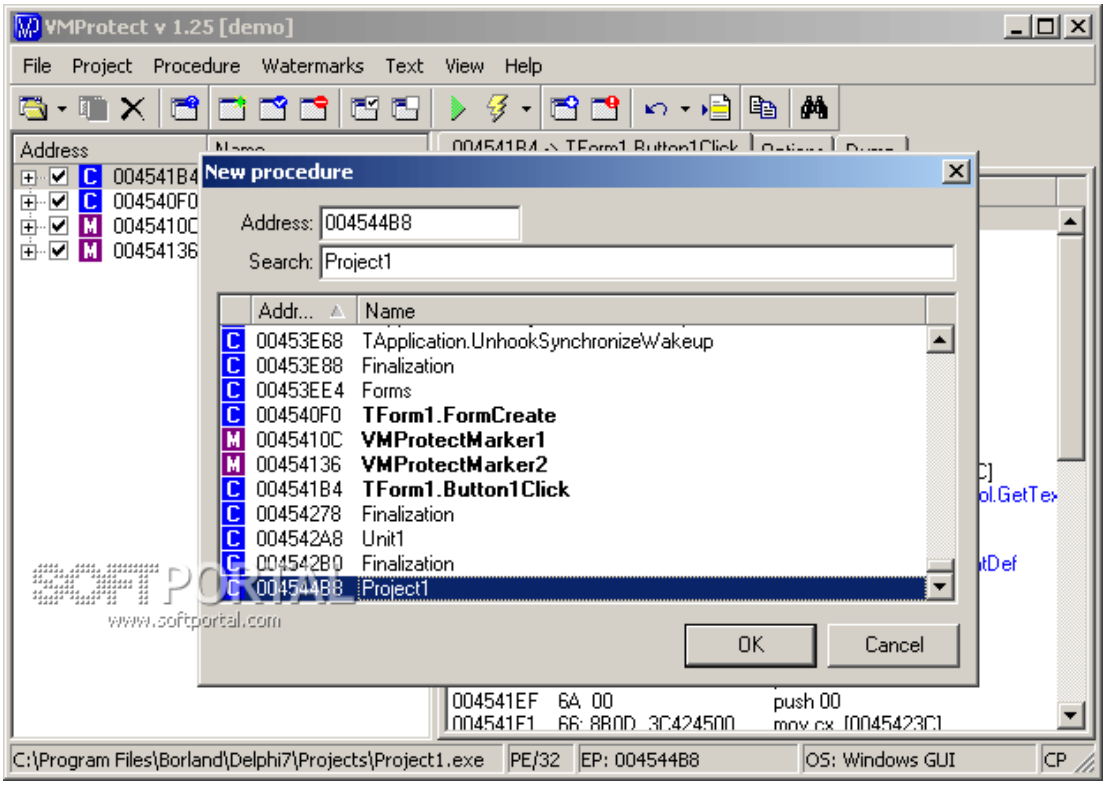


Рисунок 1.1 – Вікно програмного додатку VMProtect

Private exe Protector [9] (рисунок 1.2) – програма для захисту виконуваних файлів додатків від злому і дослідження. В своїй роботі використовує метаморфний завантажувач, алгоритм стиснення lzma, шифрування - aes-256, функції ліцензування, швидку віртуальну машину та загалом дозволяє якісно виконати захист програмних додатків.

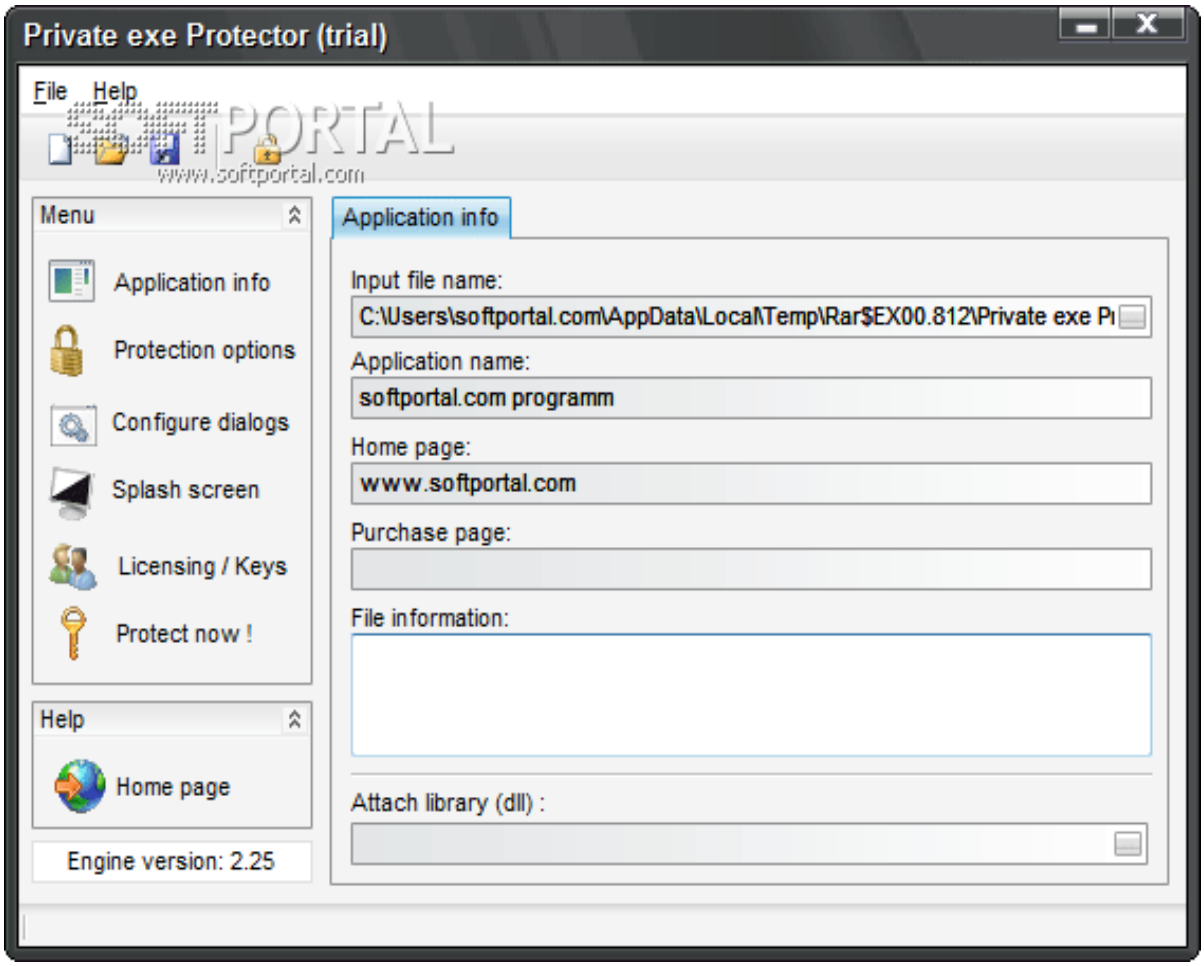


Рисунок 1.2 – Вікно програмного додатку Private exe Protector

ASProtect [10] (рисунок 1.3) – система програмного захисту додатків від несанкціонованого копіювання. Розроблено для швидкого встановлення функцій захисту програми і призначена, в першу чергу, для розробників програмного забезпечення. ASProtect розроблена для таких специфічних завдань, як робота з реєстраційними ключами і створення оціночних і пробних версій додатків. Також дозволяє шифрувати та стискати виконувані файли

програми, що можуть виконуватись без попереднього розпакування, а розпаковуватись за потреби при виконанні.

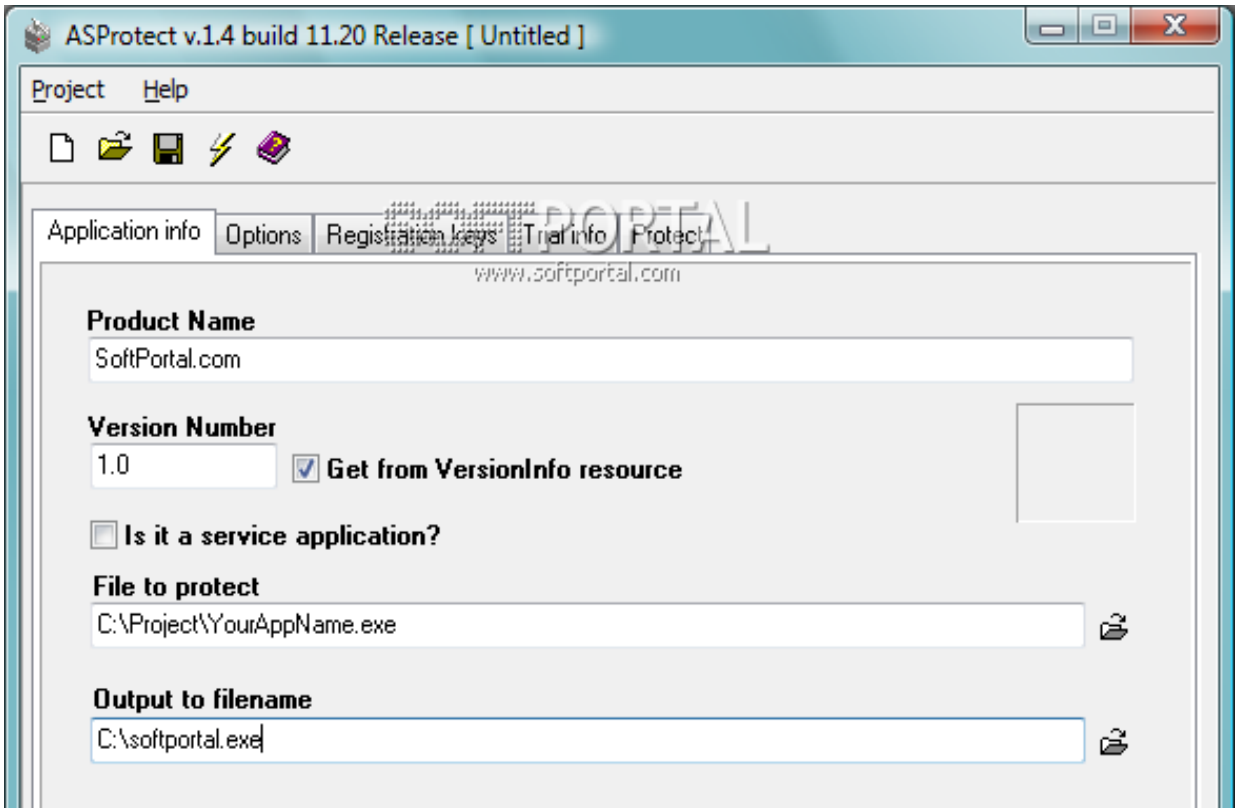


Рисунок 1.3 – Вікно програмного додатку ASProtect

Ripe Exe [11] (рисунок 1.4) – захищає програмний продукт від несанкціонованого копіювання, модифікації, аналізу алгоритму роботи і зняття "дампа" з його образу в пам'яті. Крім того, є «пакером», тобто додатково стискає програмний продукт. Захищені додатки працюють з тією ж швидкістю, що і незахищені і не вимагають для своєї роботи наявності прав адміністратора або відключення UAC, що в свою чергу робить використання продукту зручнішим та простішим, як в умовах особистого користування, так і на підприємствах.

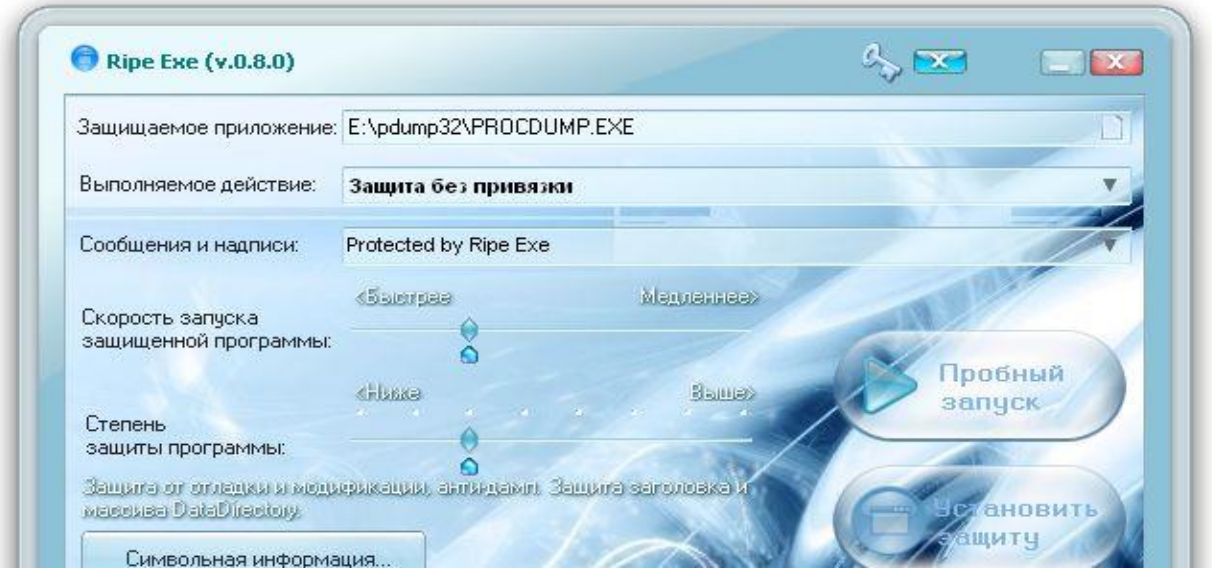


Рисунок 1.4 – Вікно програмного додатку Ripe Exe

Проаналізувавши популярні аналоги, визначимо їх можливості та недоліки, сильні й слабкі сторони, які врахуємо при створенні власного програмного продукту з назвою «DllLoadManager». Результати порівняльного аналізу аналогів зведено в табл. 1.1.

Таблиця 1.1 – Порівняльні характеристики аналогів

Критерій	VMProtect	Private exe Protector	ASProtect	Ripe Exe	DllLoad Manager
Відсутність використання додаткових апаратних ресурсів	+	+	+	+	-
Відсутність використання віртуальної машини	+	-	+	+	+
Доступ до коду захищеної програми	-	-	-	-	+
Можливість розширення	-	-	-	-	+
Небезпека при взломі програми-захисника	-	-	-	-	+
Відсутність необхідності додаткової обробки програмного продукту	-	-	-	-	+

Таблиця порівняльних характеристик (табл.1.1) показала, що розробка власної системи захисту програмного забезпечення від несанкціонованого доступу є доцільною. В результаті отримаємо продукт, що покриває недоліки існуючих рішень і дає користувачам можливість забезпечення захищеності програмного продукту без додаткових чи надлишкових заходів, використовуючи при цьому метод передачі виконуваних файлів захищеним каналом з сервера, що зберігає користувацький виконуваний файл.

1.3 Постановка задач роботи

Після аналізу стану методів захисту програмного забезпечення та порівняння існуючих аналогів було визначено, що розробка власної системи захисту програмного забезпечення від несанкціонованого доступу є доцільною. Врахувавши недоліки існуючих аналогів було визначено загальну структуру та архітектуру програмного додатку, що зможе не тільки забезпечити необхідний рівень захищеності програмного додатку, а й не матиме недоліків, які було визначено в ході аналізу існуючих аналогів.

Основним підходом до забезпечення захисту програмного забезпечення від несанкціонованого копіювання буде метод динамічного завантаження виконуваного коду програми в процес на стадії його запуску. Тобто основною ідеєю є відмова від збереження виконуваного коду програмного продукту безпосередньо на жорсткому диску користувача як на постійній основі, так і тимчасово. Виконуваний код перебуватиме лише в оперативній пам’яті комп’ютера. Ще однією перевагою даного методу є можливість паралельного використання інших методів захисту, наприклад, такого, як заплутування коду.

Систему захисту було вирішено розбити на такі ключові модулі:

1. Програма-менеджер, що буде управляти роботою всієї системи та виконуватиме ключову роль, а саме виконуватиме завантаження, створюватиме

канал міжпроцесної взаємодії, створюватиме процес для запуску коду, що захищається, завантажуватиме в пам’ять інші компоненти системи.

2. Динамічно завантажувана бібліотека, що буде виконувати розпаковку коду безпосередньо в оперативну пам’ять процесу.

3. Сервер для зберігання файлів програмного продукту, що захищається. Для захисту програм, що на ньому будуть зберігатись, може бути захищений любим методом.

4. Сервер авторизації клієнта системи. Даний вузол є опціональним. Його задачею є забезпечення додаткового захисту сервера зберігання шляхом генерації токенів (ключів) доступу та автоматизація процесу доступу до нього.

Для реалізації системи є необхідним забезпечити взаємодію між процесами головної програми-менеджера та бібліотекою. Для забезпечення цієї взаємодії було обрано технологію поширеної пам’яті (Shared Memory). Дана технологія є розробкою корпорації Microsoft та широко використовується на платформі операційної системи Windows. Вона зможе забезпечити достатній захист програм, а також є зручною при розробці.

Для реалізації вищеописаних компонентів та забезпечення їх зв’язку та взаємодії було визначено ряд задач, які необхідно виконати у магістерській кваліфікаційній роботі:

– удосконалити метод динамічного завантаження програмного продукту або його частин та розробити модель і алгоритми системи захисту програмних додатків від несанкціонованого копіювання виконуваних файлів;

– розробити динамічно завантажувану бібліотеку, що буде завантажуватись у виконуваний процес та структуровано копіювати вхідну користувацьку програму в оперативну пам’ять процесу;

– розробити модуль для роботи з поширеною пам’яттю системи, що забезпечить передачу необхідних налаштувань в процес користувацького додатку;

– розробити модуль для завантаження користувацької бібліотеки з сервера;

– розробити модуль для управління процесом користувацького додатку, що дозволить передавати управління процесом програмі, що захищається;

– розробити модуль для завантаження розробленої динамічно завантажуваної бібліотеки в процес;

– провести тестування програмного продукту.

1.4 Висновки

У даному розділі було розглянуто актуальність проблеми захисту програмного продукту від несанкціонованого доступу, а також конкретні методи реалізації такого захисту.

Також було проаналізовано перспективи вирішення даного питання шляхом розгляду аналогів: VMProtect, Private Exe Protector, ASProtect, Ripe Exe. Було визначено їх сильні та слабкі сторони. Проведено оцінку аналогів за критеріями можливості статичного аналізу виконуваного файлу програми, можливості створення несанкціонованих копій, використання додаткових апаратних ресурсів, використання віртуальної машини для захисту та можливість розширення методу захисту його поєднанням з іншим методом. Було проведено порівняння порівняння аналогів між собою та розроблюваною програмою. В результаті порівняння було виокремлено недоліки існуючих методів захисту, на основі яких сформульовано основні задачі магістерської кваліфікаційної роботи та відображено доцільність розробки власної реалізації системи захисту програмних продуктів від несанкціонованого доступу та копіювання.

2. РОЗРОБКА МЕТОДУ, МОДЕЛІ ТА АЛГОРИТМІВ СИСТЕМИ ДИНАМІЧНОГО ЗАВАНТАЖЕННЯ

2.1 Вибір технологій реалізації компонентів системи

Для реалізації системи необхідно обрати технології, які в повній мірі зможуть задовольнити технічні та функціональні вимоги модулів.

В якості основного фреймворка було обрано WinAPI.

Windows API (application programming interfaces) — загальне найменування для цілого набору базових функцій інтерфейсів програмування застосунків операційних систем сімейства Windows корпорації Майкрософт. Є найпрямішим способом взаємодії програмних додатків з Windows. Для створення програм, що використовують Windows API, Майкрософт випускає SDK, який називається Platform SDK і містить документацію, набір бібліотек, утиліт і інших інструментальних засобів.

Для забезпечення міжпроцесної взаємодії, в якій виникає необхідність в ході передачі налаштувань процесу користувацької програми, було обрано технологію SharedMemory.

Спільна пам'ять (англ. Shared memory) — регіон комп'ютерної пам'яті, до якої мають доступ кілька програм водночас. Такий доступ може організовуватись з метою зв'язку або передачі даних між програмами (чи їх потоками виконання), коли зайве копіювання даних небажане. Залежно від контексту, програми можуть запускатись як на одному процесорі, так і на кількох.

Для реалізації модуля завантаження користувацького додатку с сервера постала необхідність у виборі бібліотеки, що змогла б забезпечити передачу даних в мережі. В якості такої бібліотеки було обрано бібліотека cURL для мови C++.

cURL — це набір утиліт та бібліотек для організації вибірки даних з вебу, що надає можливість гнучкого формування запиту із завданням таких параметрів, як cookie, user_agent, referrer і будь-яких інших заголовків. cURL — це додаткова можливість оперувати з файлами на стороні сервера сторінок Інтернету за допомогою параметрів, що можуть бути переданими в рядку URL. За допомогою cURL можна, наприклад, отримати html-сторінку, не використовуючи для цього браузер. За допомогою бібліотеки libcURL можна створювати запити на завантажування файлів використовуючи обраний протокол передачі даних в мережі. libcURL забезпечує велику швидкість передачі даних, гнучкість в налаштуванні та простоту використання.

2.2 Розробка методу та моделі системи

Метод динамічного завантаження програмного продукту або його частин, на відміну від існуючого, передбачає збереження виконуваного коду на сервері, а не на локальній машині користувача, та динамічне завантаження бібліотек у робочий процес, що забезпечує високий рівень захисту виконуваних файлів від несанкціонованого копіювання.

Програмний додаток DllLoadManager отримуватиме вхідні дані безпосередньо від користувача. Завантажуваний виконуваний файл або його частина зберігатимуться на простому сервері користувача, наприклад, FTPS сервері. FTPS сервер є зручним та простим рішенням, що задовольняє всі вимоги щодо зберігання виконуваних файлів та їх захисту. Авторизація на сервері є обов’язкова. Крім власника програмного продукту, ніхто не має відкритий доступ до виконуваного файлу. Доступ до програмного додатку DllLOadManager мають всі користувачі, адже програма не зберігає даних, що можуть бути використані для отримання інформації про виконуваний код та

виконуваного коду власника. Використання сервера реєстрації для додаткового захисту від несанкціонованого використання є важливим кроком до захисту програмного забезпечення, однак не обов'язковим. DllLoadManager гнучко налаштовується як на роботу з сервером реєстрації, так і без нього, в залежності від вибору користувача програмного забезпечення.

Узагальнена модель системи захисту зображена на рисунку 2.1.

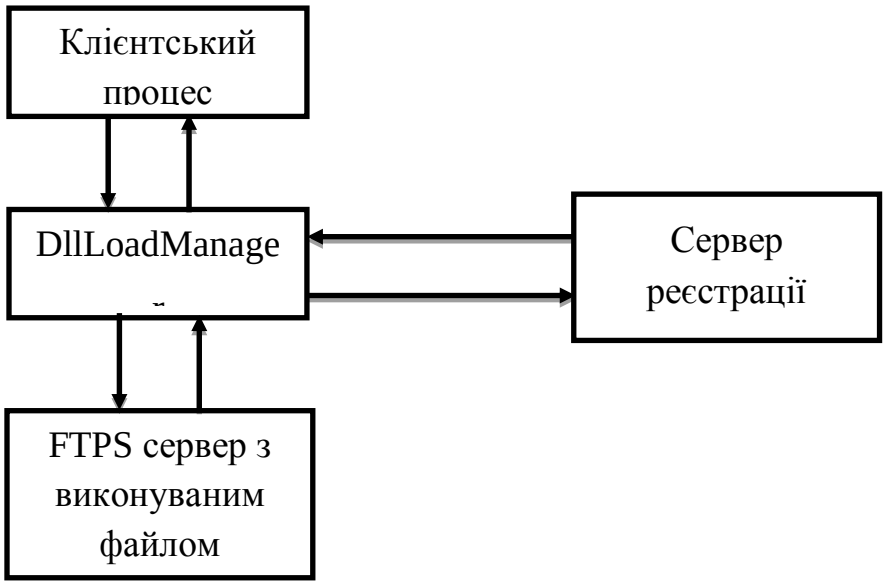


Рисунок 2.1 – Узагальнена модель системи захисту, реалізована в програмному додатку DllLoadManager

Програмний додаток DllLoadManager являє собою програму на стороні клієнта, що управляє процесом авторизації, завантаження виконуваного коду або його частини з сервера, що знаходиться на стороні власника виконуваного коду та власне зберігає цей код, а також запуском клієнтського процесу та завантаженням виконуваного коду безпосередньо в даний процес.

Для виконання вище вказаних функцій DllLoadManager має такі модулі: DllLoadManager.dll, SharedMemoryManager, UnpackModule, ServerCommunicator, ProcessController, ConfigModule. Більш детальний опис кожного з модулів наведений нижче.

Динамічно завантажувана бібліотека DllLoadManager.dll завантажується безпосередньо в клієнтську програму та відповідає за отримання та структуроване завантаження необхідного виконуваного коду, що був отриманий з сервера.

SharedMemoryManager – модуль для створення і управління спільною (поширеною) пам’яттю.

UnpackModule – модуль для розгортування та структурованого завантаження виконуваного коду в пам’ять.

Модуль комунікації з сервером авторизації та сервером зберігання коду ServerCommunicator відповідає за отримання підтвердження сесії від сервера авторизації, а також за завантаження з сервера виконуваного коду.

Модуль ProcessController відповідає за запуск клієнтського процесу, за завантаження в пам’ять процесу DllLoadManager.dll та виконання необхідних початкових налаштувань клієнтського процесу, що є необхідними для подальшої роботи з ним. Такими налаштуваннями є, наприклад, запуск процесу в режимі, що дозволяє «заморозити» (поставити на паузу виконання инструкцій коду програми) процес відразу після завантаження та розміщення коду та ресурсів в відповідні секції пам’яті процесу.

ConfigModule – модуль для створення та взаємодії з файлом конфігурації користувацьких налаштувань програми, в якому зберігаються налаштування між сесіями роботи програми та який забезпечує зручне використання програмного додатку DllLoadManager.

В повній сесії роботи додатку запускаються два різні процеси, перший – для взаємодії з сервером, конфігураційним файлом та запуском дочірнього процесу, дочірній процес, для завантаження і розпакування виконуваного коду та передача йому керування. Для забезпечення міжпроцесної взаємодії, необхідність в якій виникає при передачі налаштувань дочірньому процесу, використано SharedMemory технологію. Модель взаємодії процесів та модулів всередині кожного з них представлена на рисунку 2.2.

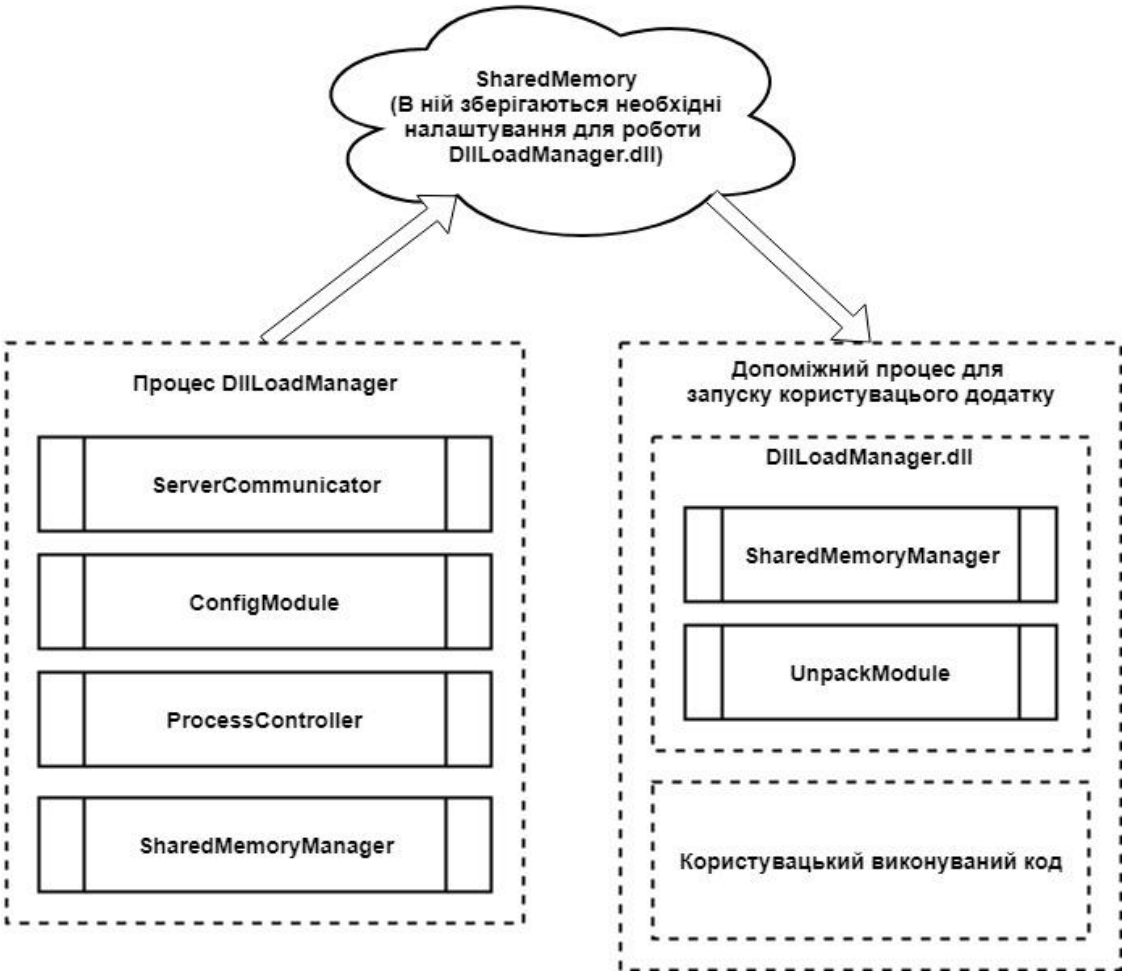


Рисунок 2.2 – Модель взаємодії процесів DllLoadManager та користувацького додатку

Звичайний запуск програми в операційній системі Windows відбувається шляхом завантаження інструкцій бінарного коду в оперативну пам’ять створеного операційною системою процесу напряму з бінарного файлу програми. Такий запуск можна поділити на три основні кроки. Схематично звичайний запуск зображено на рисунку 2.3.

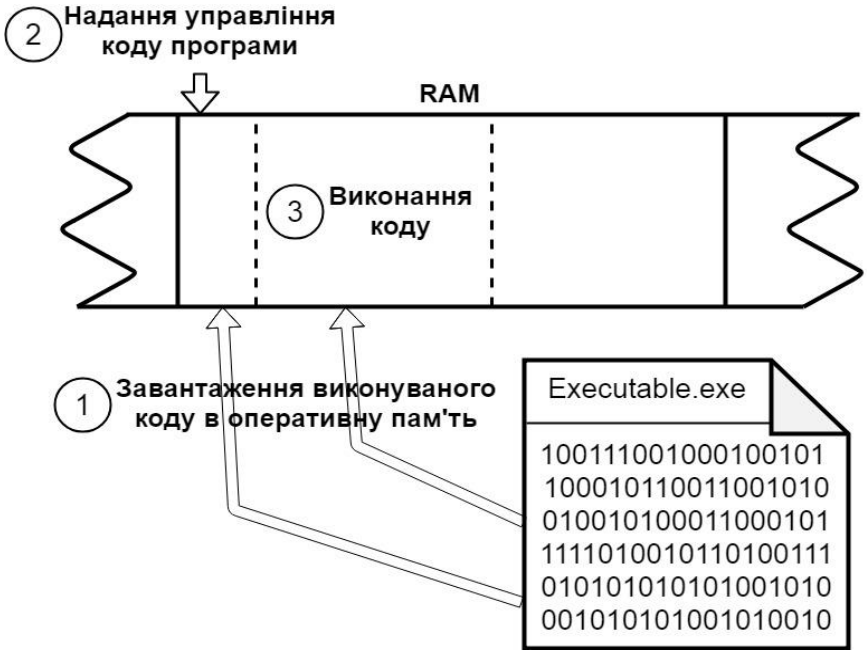


Рисунок 2.3 – Схематичне зображення звичайного запуску програми в операційній системі Windows

Після створення операційною системою процесу запуск виконується наступним чином:

1. Копіювання інструкцій бінарного коду з файлу, що знаходиться на жорсткому диску, в оперативну пам’ять процесу.
2. Передача управління процесом точці входу коду програми
3. Початок виконання коду програми

Система захисту DllLoadManager модифікує процес запуску програми, що і надає можливість захисту програми. Основною ідеєю є завантаження

бінарного коду в оперативу пам’ять не напряму з файлу, що знаходиться на жорсткому диску користувача, а зі спеціального сервера. Покрокова схема роботи системи захисту зображена на рисунку 2.4.

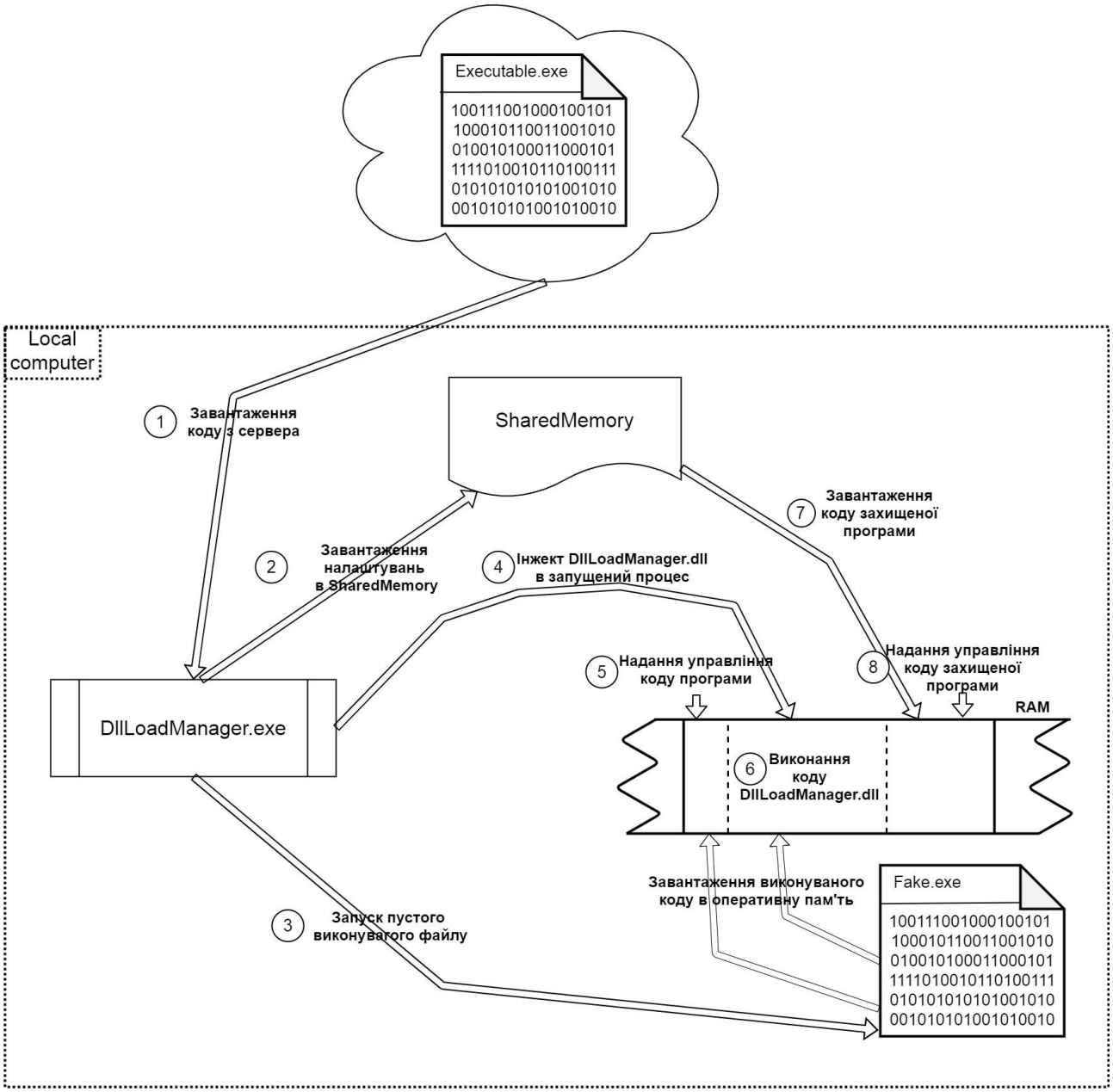


Рисунок 2.4 – Схематичне зображення роботи програми DllLoadManager

Роботу системи захисту можно логічно поділити на 8 основних кроків:

1. Завантаження виконуваного коду користувацького програмного додатку, який необхідно захистити від взалму, з сервера. Під час скачування

код тимчасово зберігається в оперативній пам’яті процесу DllLoadManager у вигляді набору байтів, що представляють бінарний код користувацького додатку або бібліотеки.

2. Завантаження налаштувань системи захисту, а саме: імені програмного додатку, що захищається, адреси комірки оперативної пам’яті DllLoadManager з початком коду додатку та розміру (кількістю байтів) ділянки пам’яті з кодом, з оперативної пам’яті процесу DllLoadManager в попередньо створену ділянку поширеної пам’яті.

3. Запуск звичайного пустого виконуваного файлу в операційній системі Windows (зазвичай файл з розширенням «.exe»), що ініцією створення нового процесу в операційній системі, реєстрацію процеу, виділення ресурсів під цей процес, виділення сегменту в оперативній пам’яті та адресації.

4. На єтапі запуску процесу після виділення всіх необхідних ресурсів для нього, процес «заморожується» і виконання коду даного файлу не відбувається. В цей момент DllLoadManager проводить «ін’єкцію» DllLoadManager.dll (підхід відомий під назвою dll injection) файлу в запущений процес. Бібліотека DllLoadManager.dll має необхідні інструкції, що дозволяють завантажити налаштування з поширеної пам’яті. Відповідно до налаштувань DllLoadManager.dll виконує завантаження коду користувацького додатку та структуровано копіює його в оперативну пам’ять поточного процесу.

5. Передача управління процесом коду програми. В даний момент кодом програми, що починає виконуватись, є безпосередньо код попередньо завантаженої DllLoadManager.dll. Дана поведінка досягається за допомогою правил завантаження .dll файлів, що надаються та створюються системними функціями Microsoft.

6. Виконання коду DllLoadManager.dll, що виконує завантаження налаштувань системи, обробку отриманих даних та проведення попередньої підготовки процесу.

7. Виконання завантаження коду користувацької програми з поширеної пам’яті. Також DllLoadManager.dll виконує структуроване копіювання коду користувацького додатку у власний кодовий сегмент процесу.

8. Останнім кроком є передача управління процесом користувацькому коду, що захищається. Після цього кроку DllLoadManager не впливає на роботу захищеної програми.

Запропонований метод динамічного завантаження програмного продукту або його частин передбачає збереження виконуваного коду на сервері, а не на локальній машині користувача, та динамічне завантаження бібліотек у робочий процес, що забезпечує високий рівень захисту виконуваних файлів від несанкціонованого копіювання. Метод динамічного завантаження програмного продукту передбачає наступні кроки користувача системи:

1. Розміщення файлів виконуваного коду свого додатку на файловому сервері (FTPS), заборонивши вільний доступ до нього, та надання доступу за паролем.

2. Запуск сервера реєстрації, що буде отримувати GET запити на отримання пароля для доступу до FTPS сервера, перевіряючи дані користувача, що прийшли в запиті.

3. Формування конфігураційного файлу, в якому потрібно прописати адреси серверів.

4. Доставку до користувача-замовника програмного забезпечення конфігураційного файлу та додатку DllLoadManager, а також порожнього виконуваного файлу, в який буде виконано завантаження файлу.

5. Реєстрація користувача-замовника програмного забезпечення на сервері реєстрації.

Після виконання вищевказаних дій користувач-замовник програмного забезпечення зможе скористатись додатком DllLoadManager та налаштованим конфігураційним файлом для нього, щоб запустити додаток, який постачає користувач-власник. У користувача є змога змінити налаштування конфігураційного файлу або ж замінити файл зовсім, та продовжити використовувати додаток DllLoadManager за запуску інших додатків від інших постачальників програмного забезпечення. Тобто, в залежності від додатку, який необхідно запустити, слід використовувати той чи інший конфігураційний файл, що є зручним для швидкого поширення програмного забезпечення. Звичайно, для використання програмного додатку DllLoadManager необхідний доступ до мережі Internet, адже він є необхідним для доступу до серверів.

2.3 Розробка алгоритму роботи програми DllLoadManager

Головними функціями програмного додатку DllLoadManager є отримання дозволу на сесію від сервера реєстрації, взаємодія з файлом конфігурації користувацьких налаштувань, отримання виконуваного коду від сервера, що зберігає виконуваний код, який потрібно захистити, (далі «сервер зберігання коду»), створення поширеної пам’яті та передачу в неї необхідних даних, запуск клієнтського процесу та завантаження в нього бібліотеки DllLoadManager.dll.

Було розроблено основний алгоритм програмного додатку, що об’єднує головні модулі програми. Даний алгоритм описує зв’язок класів програмного додатку та їх взаємодію між собою. Алгоритм, що вцілому описує роботу програмного додатку DllLoadManager, наведений на рисунку 2.4.

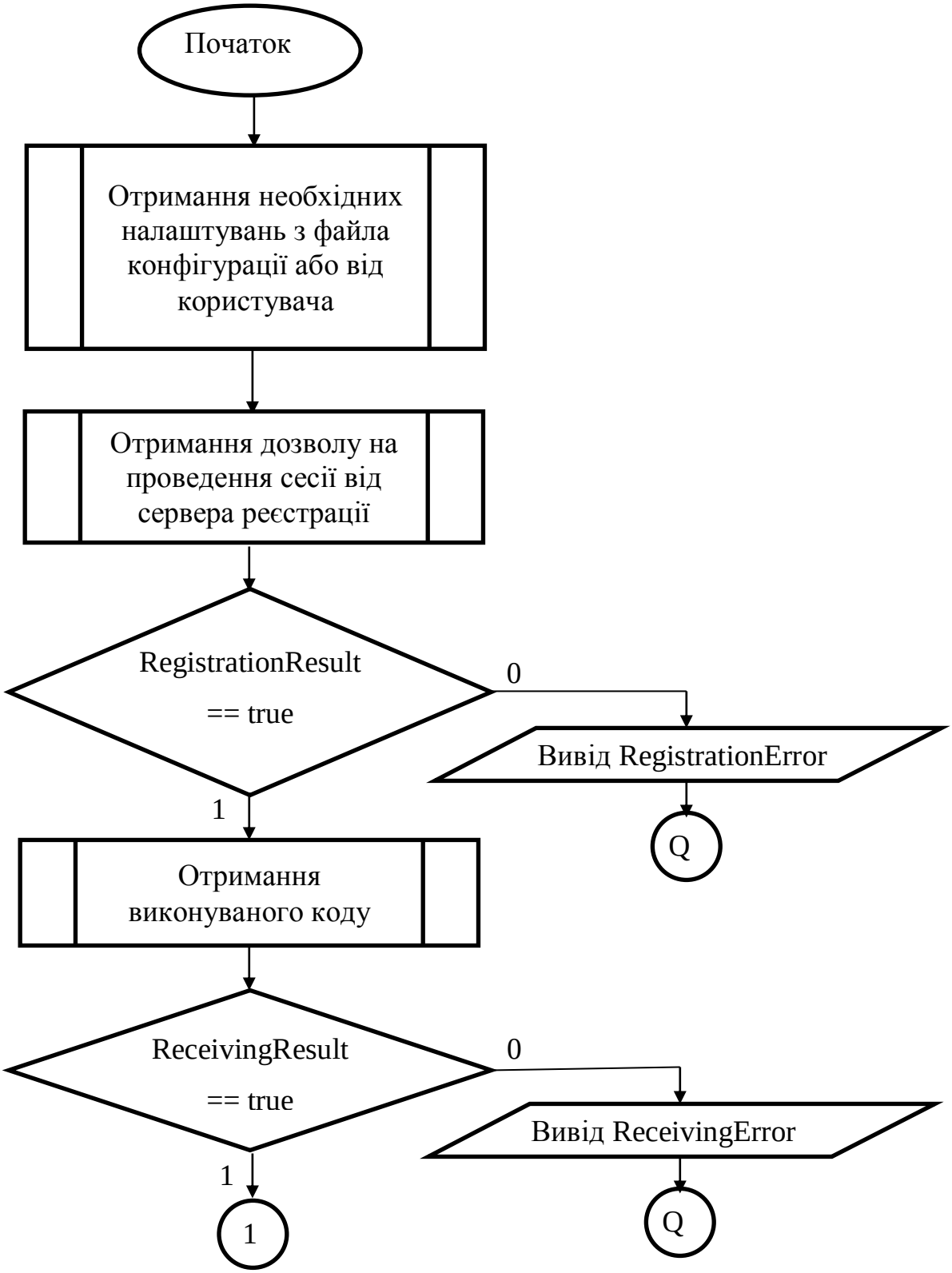


Рисунок 2.5 – Алгоритм виконання програми DllLoadManager

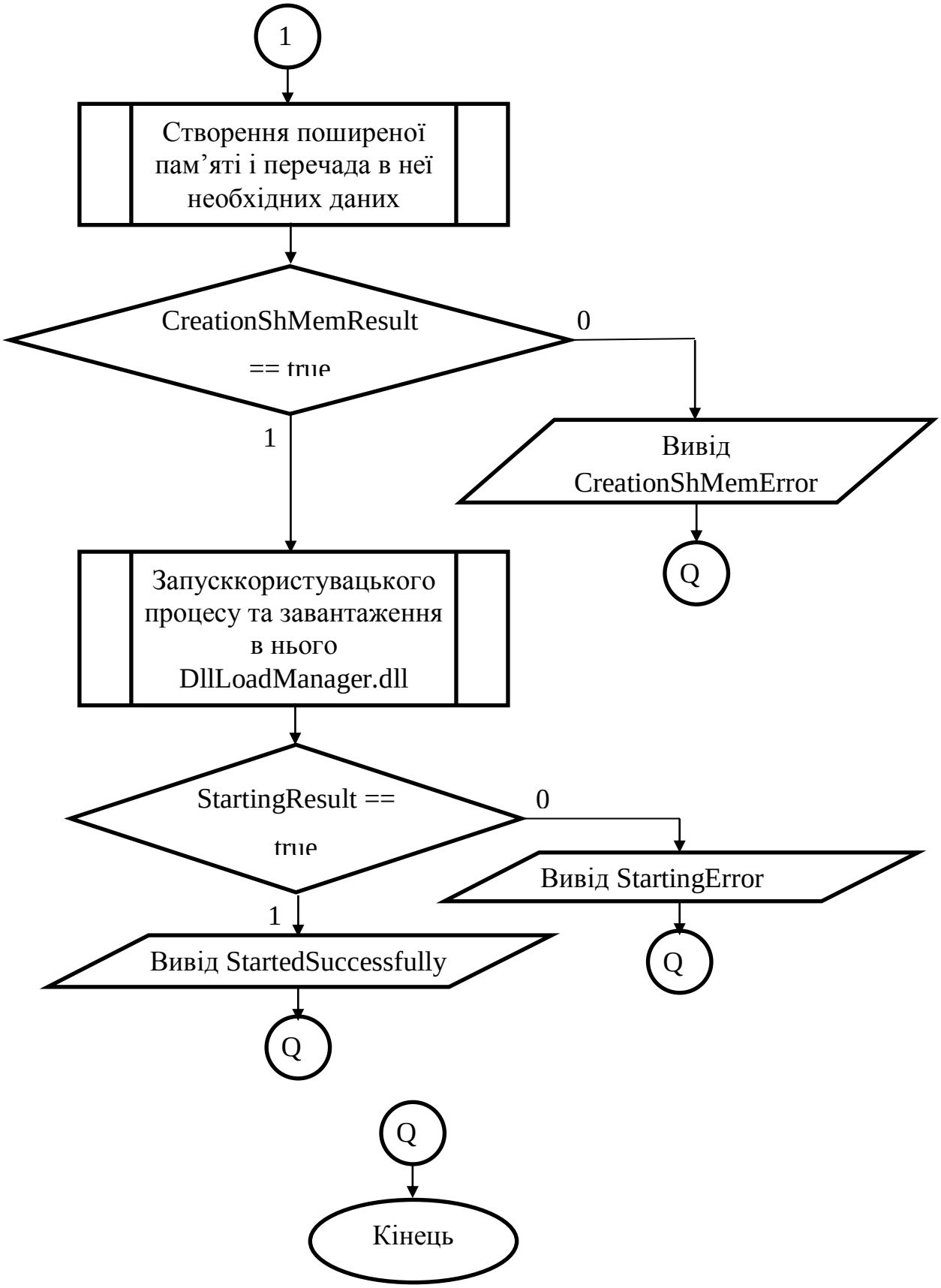


Рисунок 2.5 – Алгоритм виконання програми DllLoadManager (продовження)

У випадку помилки на будь-якій стадії виконання програми відповідне повідомлення про помилку буде виведено в консоль програми з уточнюючими параметрами. Також у користувача є можливість включити детальне звітування. В випадку детального звітування кожен логічний етап програми буде супроводжуватись коментарями виконання в консолі, що допоможе якнайточніше визначити причину помилки у разі її виникнення [12]. При відключеній можливості детального звітування програма буде стисло повідомляти про поточний етап виконання, на якому вона перебуває. Список етапів ілюструє рисунок 2.4, де кожен модуль відповідає певному етапу.

У випадку успішного виконання програма виводить повідомлення про успішний запуск. Успішним виконанням програми вважається успішне виконання всіх етапів без виключень.

2.4 Розробка алгоритму роботи динамічно завантажуваної бібліотеки DllLoadManager.dll та файлу конфігурації

Бібліотека DllLoadManager.dll є важливою складовою роботи додатку. Вона завантажується програмним додатком в процес. Далі з поширеної пам’яті отримує необхідні налаштування. З переданих даних бібліотека отримує адресу та розмір виконуваного файлу, завантаженого в пам’ять з сервера, а також деякі налаштування, що додатково корегують її роботу, наприклад, необхідність збереження звіту про свою роботу в файл, необхідність затримок по часу тощо.

Наступним кроком виконання інструкцій є структуроване розгортання виконуваного коду в поточний процес, в який, власне, була завантажена бібліотека, таким чином, що після виконання роботи бібліотеки процес почне виконувати завантажені інструкції.

Алгоритм роботи динамічно завантажуваної бібліотеки DllLoadManager.dll наведено на рисунку 2.6.

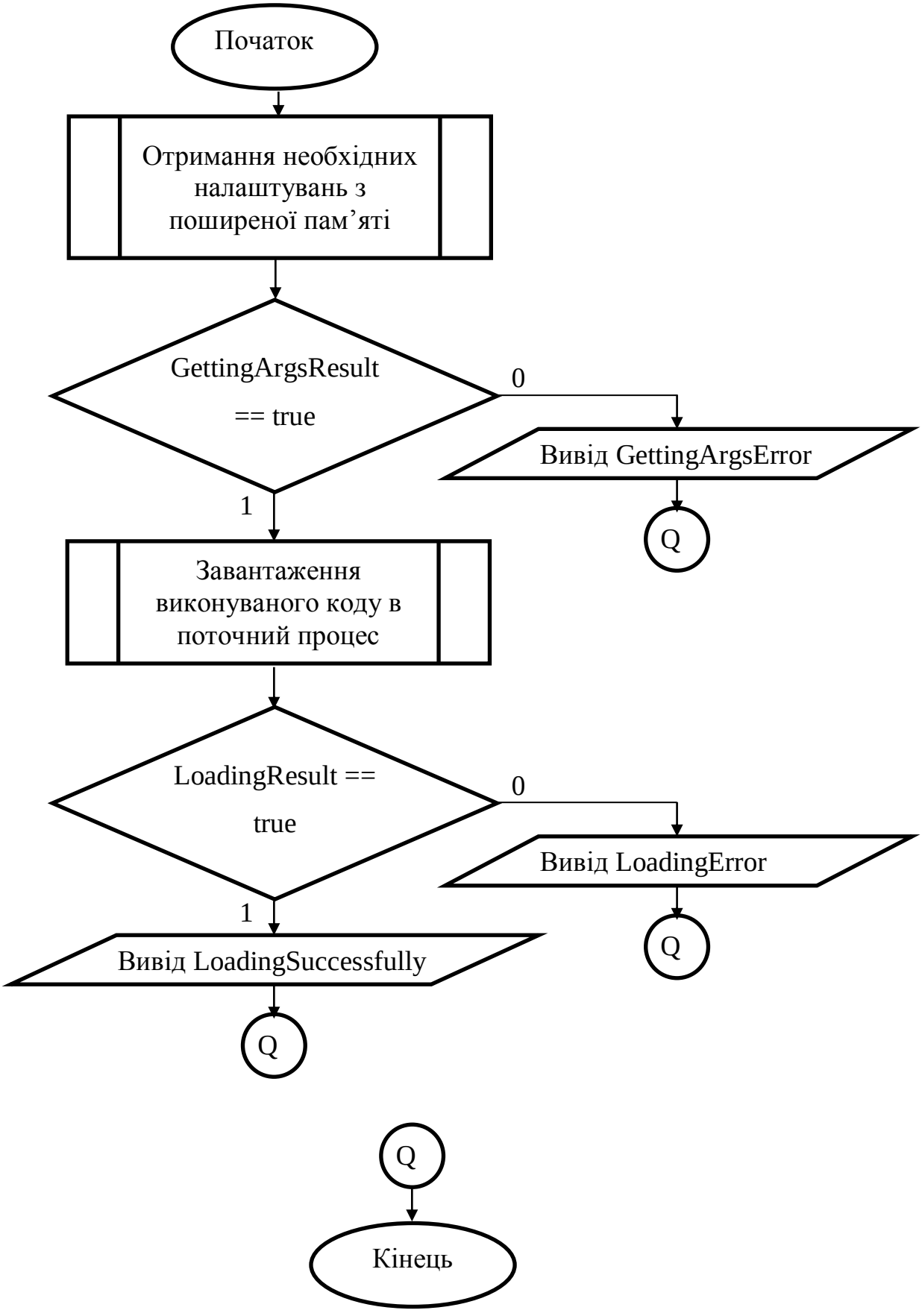


Рисунок 2.6 – Алгоритм виконання бібліотеки DllLoadManager.dll

У випадку виникнення помилок при виконанні інструкцій бібліотеки відповідне повідомлення про помилку буде виведено в файл звіту з вказаними налаштуваннями, що призвели до помилки. В разі успішного виконання в файл звіту буде виведено повідомлення про передані аргументи та успішне завершення роботи бібліотеки. Після завершення роботи бібліотеки управління процесом буде передане завантаженим інструкціями коду користувацього додатку.

Для зручного використання програми було прийнято рішення створити файл конфігурації, що буде відповідати за користувацькі налаштування програми. В файлі конфігурації перераховано всі необхідні налаштування для коректної роботи програми.

Файл конфігурації має назву «Config.ini». Він являє собою звичайний текстовий файл (з розширенням «.txt»), в якому перелічені налаштування зі значеннями за замовчуванням. При першому запуску програмного додатку DllLoadManager файл конфігурації генерується програмою. При кожному наступному запуску програма перевіряє наявність усіх необхідних полів налаштувань. Якщо деякі поля відсутні, програма перегенерує файл конфігурації та виведе відповідне повідомлення про помилку з інструкціями для користувача щодо правильного налаштування програми.

Відкрити файл конфігурації та внести в нього зміни відповідно до користувацьких налаштувань можна за допомогою текстового редактора «Блокнот» чи будь-якого іншого аналогу, що дозволяє порматування текстових файлів з розширенням «.txt».

Файл конфігурації представлений на рисунку 2.7.

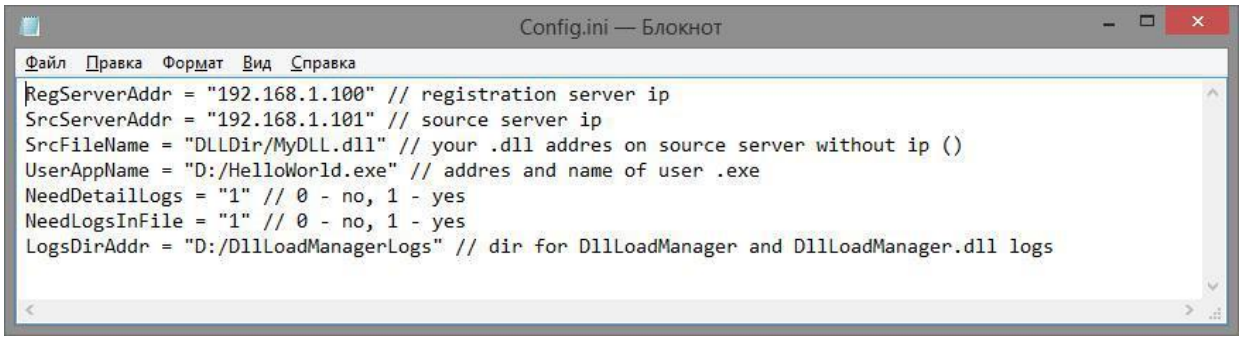


Рисунок 2.7 – Файл конфігурації програмного додатку

Файл конфігурації вміщує такі поля:

– RegServerAddr – поле для IP адреси сервера реєстрації програмного додатка (опціональне);

– SrcServerAddr – поле для IP адреси сервера, що зберігає файл виконуваного коду, що необхідно завантажити;

– SrcFileName – поле для адреси та назви виконуваного файлу на сервері SrcServerAddr;

– UserAppName – поле для адреси в файловій системі та назва файлу .exe, в який необхідно завантажити виконуваний код з сервера

– NeedDetailLogs – поле для мітки, що вказує на необхідність детального звітування;

– NeedLogsInFile – поле для мітки, що вказує на необхідність збереження звіту роботи програмного додатку в файл;

– LogsDirAddr – поле для адреси директорії, в котру повинні зберігатись звіти роботи програми DllLoadManager та бібліотеки.

За виключенням поля RegServerAddr, усі поля є обов’язковими до заповнення. Якщо обов’язкове поле незаповнене, програма виведе відповідне повідомлення про помилку та запропонує ввести значення поля для продовження роботи. Після введення інформації програма продовжить працювати без додаткового перезавантаження.

2.5 Висновки

У даному розділі було розроблено метод та модель системи захисту програмного забезпечення, реалізовані в програмному додатку DllLoadManager, описано алгоритми роботи програмного додатку DllLoadManager та динамічно завантажуваної бібліотеки DllLoadManager.dll. У рамках розробки алгоритму робота програмного додатку, передбачено обробку помилок та виведення повідомлень про відповідні помилки. Розроблено стратегію забезпечення послідовного виконання етапів алгоритму з урахуванням можливості продовження роботи програми за умови неуспішного виконання попереднього етапу алгоритму.

Розроблено файл конфігурації програмного додатку, що регулює налаштування програми та бібліотеки, обрано необхідні поля конфігурації, що забезпечать найбільш зручне та водночас гнучке використання програмного застосунку. При розробці передбачено варіацію користувацьких налаштувань, за бажанням користувача чи за необхідністю. Деякі налаштування є обов’язковими і потребують установлення реального значення, без установки цих налаштувань програма не буде функціонувати, адже не матиме необхідних даних для підключення до сервера, використання користувацької бібліотеки тощо. Частина налаштувань не є обов’язковою і використовується за бажанням користувача, якщо користувач потребує функцій програми, передбачених даними налаштуваннями.

3. РОЗРОБКА ПРОГРАМНИХ ЗАСОБІВ ДЛЯ ПАКУВАННЯ ВИКОНАВЧИХ ФАЙЛІВ

3.1 Варіантний аналіз і обґрунтування вибору засобів реалізації програмного засобу.

C++ – мова програмування загального призначення, що виникла на початку 1980-х років, коли співробітник фірми Bell Labs Б. Страуструп розробив ряд удосконалень для мови C. Вона підтримує такі парадигми програмування як процедурне, об’єктно-орієнтоване та узагальнене програмування[15]. C++ забезпечує модульність, роздільну компіляцію, обробку виключень, абстракцію даних, об’явлення типів(класів) об’єктів та створення віртуальних функцій. Стандартна бібліотека включає загальновживані контейнери і алгоритми. C++ поєднує властивості як низькорівневих, так і високорівневих мов програмування [20].

Java – об'єктно-орієнтована мова програмування, випущена компанією Sun Microsystems у 1995 році як основний компонент платформи Java[19]. Зараз мовою займається компанія Oracle, яка придбала Sun Microsystems у 2009 році. Синтаксис мови багато в чому схожий на C та C++. У офіційній реалізації, Java програми компілюються у байткод, який при виконанні інтерпретується віртуальною машиною для конкретної платформи. Oracle надає компілятор Java та віртуальну машину Java, які задовольняють специфікації Java Community Process, під ліцензією GNU General Public License. Мова значно запозичила синтаксис із C і C++. Зокрема, взято за основу об'єктну модель C++, проте її модифіковано.

C# – об'єктно-орієнтована мова програмування з безпечною системою типізації для платформи .NET[18]. Розроблена Андерсом Гейлсбергом, Скотом Вілтамутом та Пітером Гольде під егідою Microsoft Research. C# є дуже близьким родичем мови програмування Java. Вона успадкувала від Java концепції віртуальної машини (середовище .NET), байт-коду (MSIL) і більшої

безпеки вихідного коду програм, плюс врахувала досвід використання програм на Java. Нововведенням C# стала можливість легшої взаємодії, порівняно з мовами-попередниками, з кодом програм, написаних на інших мовах, що є важливим при створенні великих проектів. Якщо програми на різних мовах виконуються на платформі .NET, .NET бере на себе клопіт щодо сумісності програм (тобто типів даних, за кінцевим рахунком) [20].

У табл. 3.1 зведено результати порівняльного аналізу мов програмування.

Таблиця 3.1 – Порівняння мов програмування

Характеристики	Мова програмування		
	C++	C#	Java
Об'єктно-орієнтована	+	+	+
Створення багатовимірних масивів	+	+	-
Узагальнене програмування	+	+	+
Створення анонімних функцій	+	-	-
Створення динамічних масивів	+	+	+
Розробка програмного інтерфейсу	+	+	+

Мова програмування C++ має перевагу над Java і C#, тому для реалізації програмного додатку було обрано саме її, адже вона задовольняє всі необхідні вимоги для створення модуля завантаження виконуваного коду і має значну

швидкодію та набір інструментів, що грає значну роль в розробці програмних продуктів для малопотужних користувацьких ПК.

Для вибору середовища розробки створення програмного додатку було проаналізовано такі середовища розробки, як Microsoft Visual Studio, Dev-C++ та C++ Builder.

Microsoft Visual Studio — серія продуктів фірми Microsoft, які включають інтегроване середовище розробки програмного забезпечення та ряд інших інструментальних засобів[16]. Ці продукти дозволяють розробляти як консольні програми, так і програми з графічним інтерфейсом, в тому числі з підтримкою технології Windows Forms, а також веб-сайти, веб-застосунки, веб-служби як в рідному, так і в керованому кодах для всіх платформ, що підтримуються Microsoft Windows, Windows Mobile, Windows CE, .NET Framework, .NET Compact Framework та Microsoft Silverlight (рисунок 3.1). Visual Studio включає один або декілька з наступних компонентів: Visual Basic .NET, а до його появи — Visual Basic; Visual C++; Visual C#; Visual J#; Visual F# (входить до складу Visual Studio 2010); Visual Studio Debugger [16].

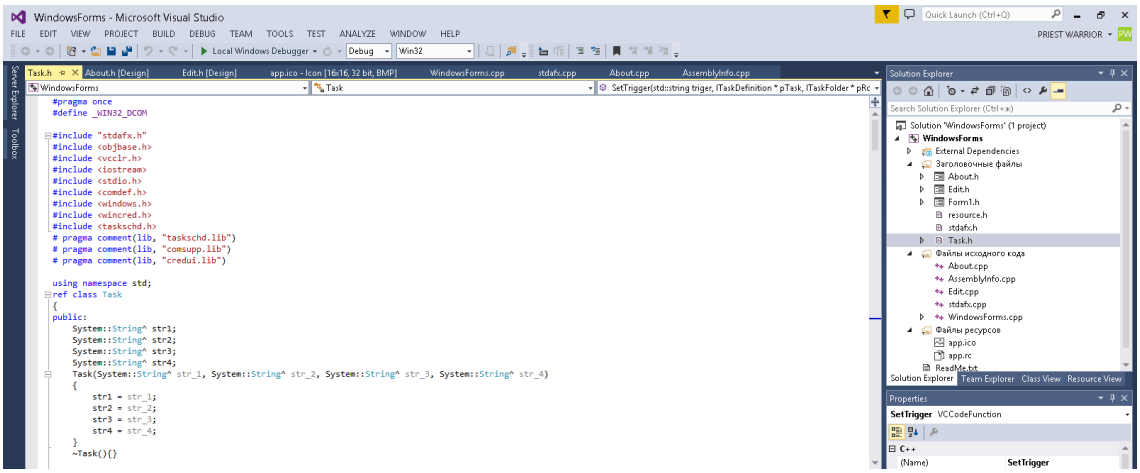


Рисунок 3.1 – Приклад роботи програми Microsoft Visual Studio

Багато варіантів постачання також включають:

- Microsoft SQL Server;

- MSDE Visual Source Safe — файл-серверна система управління версіями.

Dev-C++ — вільне інтегроване середовище розробки для мов програмування C/C++ (рисунок 3.2). У дистрибутив входить компілятор MinGW. Сам Dev-C++ написаний на Delphi. Розповсюджується згідно з GPL. На поточний момент не розробляється, замість нього активно розробляється порт інтерфейсу Dev-C++ на wxWidgets — wxDev-C++. 30 червня 2011 незалежним програмістом було випущено неофіційну версію 4.9.9.3, що включала новіший компілятор GCC 4.5.2, ресурси Windows SDK (Win32 та D3D).

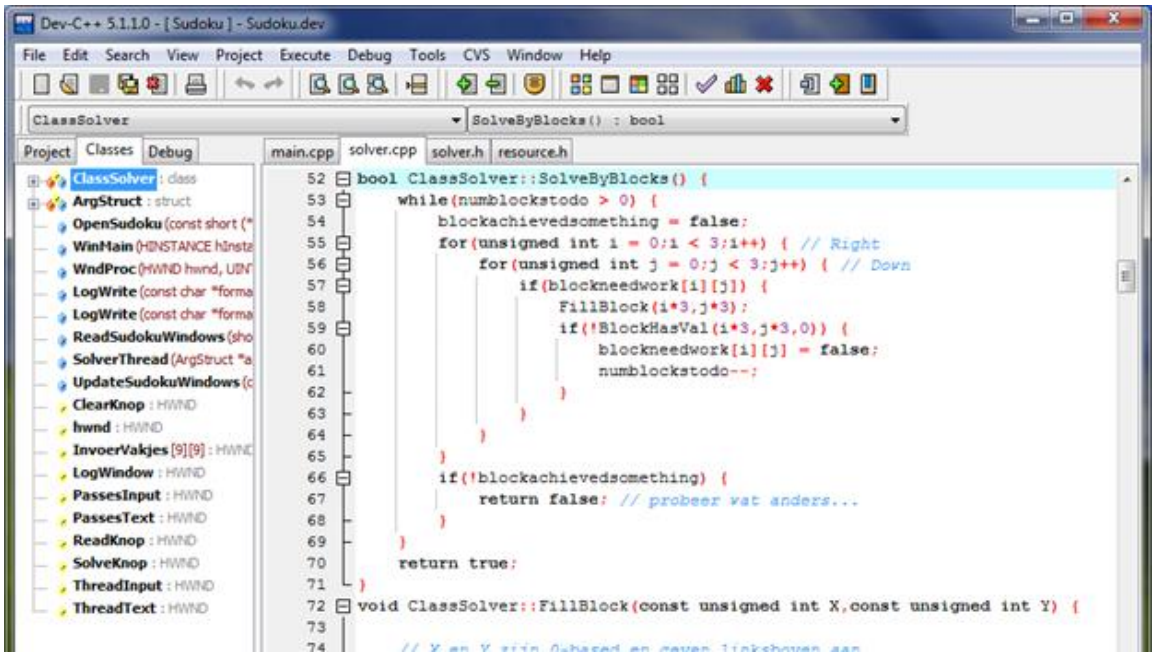


Рисунок 3.2 – Приклад роботи програми Dev-C++

Було виправлено багато помилок та покращено стабільність. 27 серпня, після 5-річного перебування у стані бета-версії, розробником з блогу Орвела було випущено версію 5.0[16]. Цей варіант має власну сторінку на SourceForge, починаючи з версії 5.0.0.5, оскільки попередні розробники не відповідають на запити.

C++ Builder — середовище швидкої розробки (RAD), що випускається компанією Codegear, дочірньою фірмою компанії Embarcadero (раніше Borland) (рисунок 3.3). Призначена для написання програм на мові програмування C++. C++ Builder об'єднує Бібліотеку візуальних компонентів і середовище програмування (IDE), написане на Delphi з компілятором C++. C++ Builder об’єднує в собі комплекс об’єктних бібліотек (STL, VCL, CLX, MFC и др.), компілятор, редактор коду і багато інших компонентів[16].

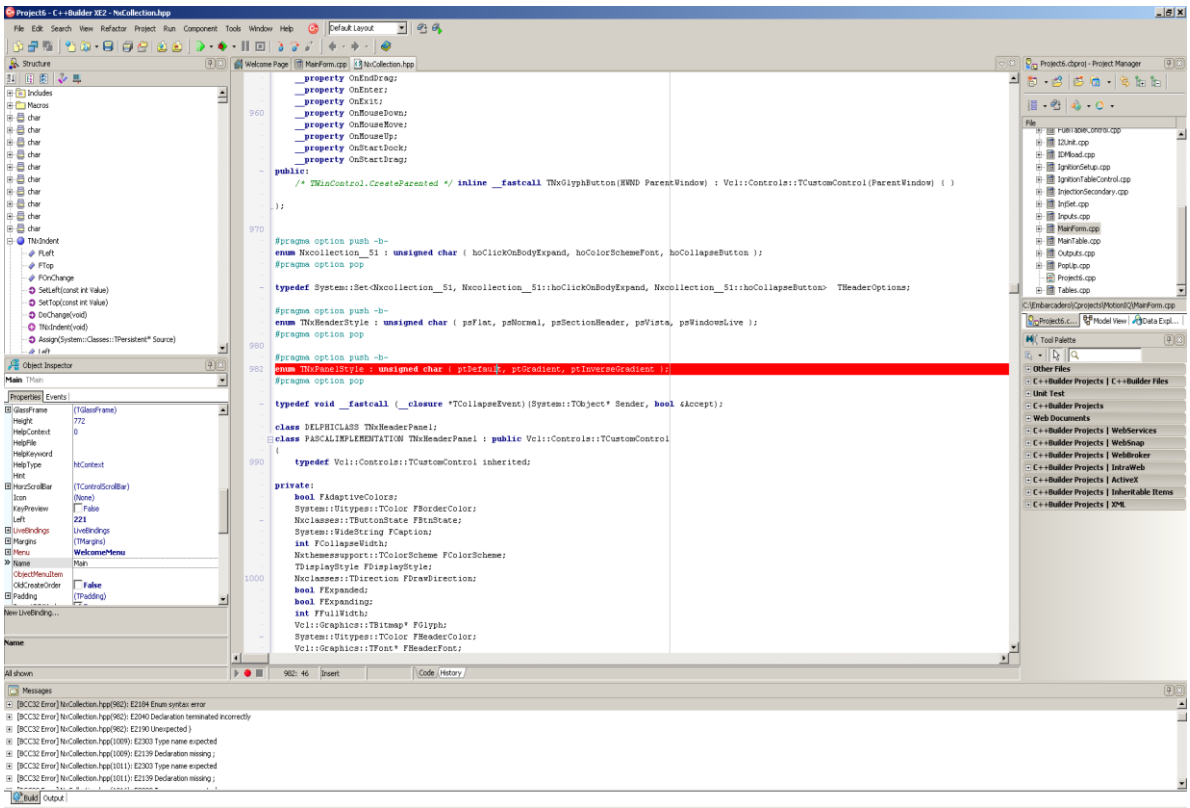


Рисунок 3.3 – Приклад роботи програми C++ Builder

Цикл розробки аналогічний Delphi[15]. Більшість компонентів, розроблених на Delphi, можна використовувати і в C++ Builder без модифікації, але не навпаки.

Builder містить інструменти, які дозволяють здійснювати справжню візуальну розробку Windows-програм методом drag-and-drop та спрощують програмування завдяки вбудованому WYSIWYG — редактору інтерфейсу.

Порівняння даних середовищ програмування наведено в таблиці 3.2.

Таблиця 3.2 – Порівняння середовищ програмування

Функції	Середовище програмування		
	C++ Builder	Dev-C++	Microsoft Visual Studio
Підтримка MFC	-	-	+
Режим відлагодження	+/-	+/-	+
Кросплатформеність	-	-	+
Функція авто заповнення	+/-	-	+

Серед розглянутих середовищ було обрано Microsoft Visual Studio, оскільки це середовище має багато готових шаблонів проектів, підтримує технологію MFC і Windows Forms, що спрощує розробку інтерфейсу додатку.

3.2 Розробка модуля для завантаження виконуваного коду в програму

Модуль завантаження коду в виконуваний процес розміщується в DllLoadManager.dll і використовується для того, щоб завантажити виконуваний код у кодовий сегмент запущеного процесу і передати його на виконання [13].

Основним об’єктом даного модуля є структура UNPACKMODULE, що вміщує в собі опис файлу, що доставлений для виконання. Також важливим функціональними складовими є структури SECTIONFINALIZEDATA та CallList. Опис цих структур наведений на рисунку 3.4.

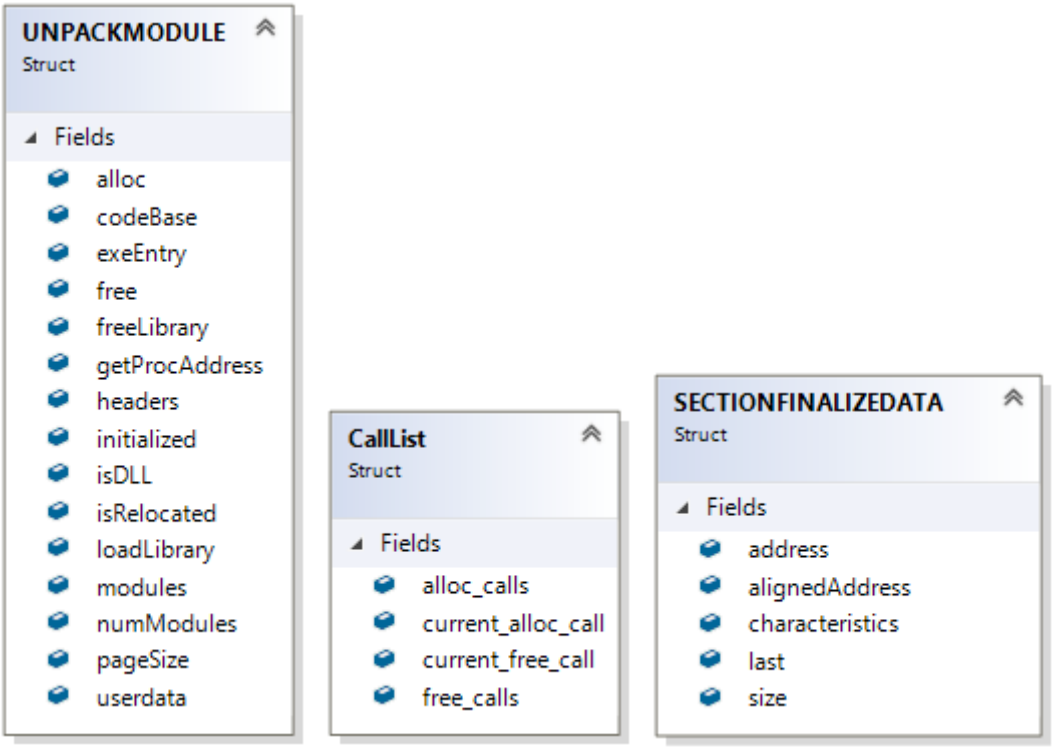


Рисунок 3.4 – Діаграма функціональних структур DllLoadManager.dll

Також важливими складовими DllLoadManager.dll є класи Dll, Log та ShardMemoryManager. Їх діаграми подані на рисунку 3.5.

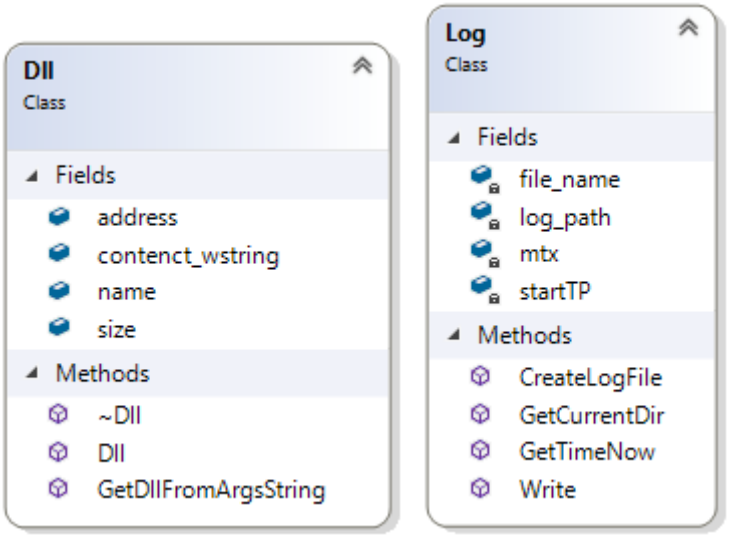


Рисунок 3.5 – Діаграми класів Dll та Log

Дані класи та структури використовується для збереження та використання налаштувань, обробку вхідних налаштувань, структурування бінарного файла та завантаження виконуваного коду в користувацький процес.

3.3 Розробка модуля для роботи з поширеною пам’яттю

Для зв’язку між програмним додатком та динамічною бібліотекою використовується технологія SharedMemory. Shared Memory – є технологією, що дозволяє максимально швидкий обмін даними між процесами в середовищі, тому допоможе досягти максимальної швидкодії програмного продукту і бібліотеки. Створення і управління поширеною пам’яттю здійснюється в класі SharedMemoryManager. Структура класу SharedMemoryManager наведена на рисунку 3.6. Даний модуль використовується як і програмному додатку DllLoalManager, так і в бібліотеці DllLoalManager.dll, адже виконує функції як запису в поширену пам’ять операційної системи, так і зчитування інформації з неї.

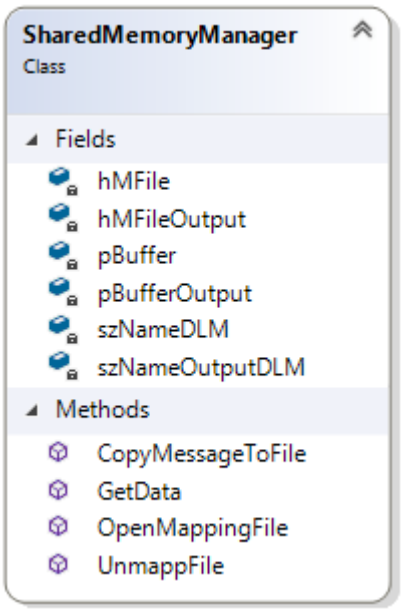


Рисунок 3.6 – Структура класу SharedMemoryManager

Модуль створює та задає значення поширеній пам’яті, а також дозволяє зчитувати значення з пам’яті за іменем.

3.4 Висновки

У даному розділі було проведено аналіз мов програмування та IDE.

У результаті було прийнято рішення використовувати Microsoft Visual Studio в якості IDE, адже дане середовище розробки відповідає всім необхідним вимогам. Має зручний текстовий редактор для написання, редагування та форматування коду. Вбудований препроцесор, що полегшить написання коду завдяки знаходженню синтаксичних помилок під час написання коду програмного додатку. Багатофункціональний інструмент для відлагодження коду в режимі роботи програми. Інструмент аналізу пам’яті та використання ресурсів системи. Також Microsoft Visual Studio дозволяє швидко та зручно обирати цільовий формат побудови виконуваного файлу, а також налаштування побудови, такі як розрядність системи, платформу тощо.

У якості мови програмування було обрано мову C++ для розробки програмного додатку. Мова C++ є об’єктно орієнтованою. Також вона забезпечить високу швидкодію програмного додатку, адже дозволяє напряму взаємодіяти з системними функціями без додаткових обгорток. Це дозволит не втрачати продуктивність роботи програми.

Також були розроблені модулі для завантаження виконуваного коду в робочий процес клієнта, а також модуль взаємодії з поширеною пам'яттю Shared Memory. Shared Memory – є технологією, що дозволяє максимально швидкий обмін даними між процесами в середовищі, тому допоможе досягти максимальної швидкодії програмного продукту і бібліотеки.

4 ТЕСТУВАННЯ ПРОГРАМИ

4.1 Тестування програмного продукту

Для тестування додатку «DllLoadManager» було обрано стратегію «чорної скриньки» [14]. Перевіримо роботу програми за відсутності поля SrcServerAddr у файлі конфігурації (рисунок 4.1).

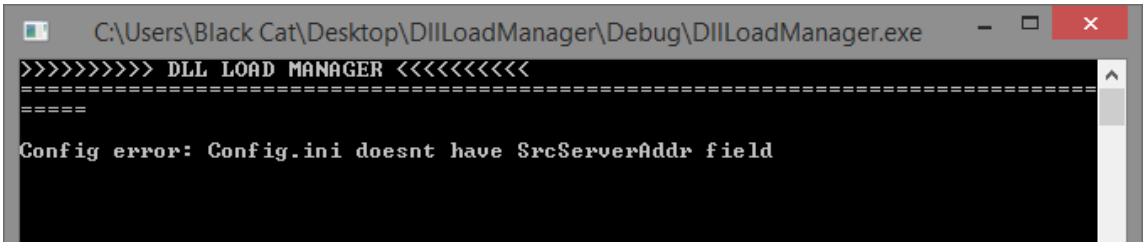


Рисунок 4.1 – Виведення повідомлення про помилку зчитування поля

Помилка на рисунку 4.2 говорить про те, що вказаний сервер є недоступним.

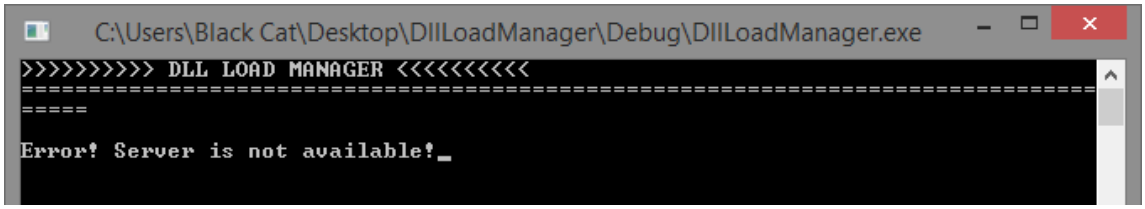


Рисунок 4.2 –Виведення повідомлення про недоступність сервера

Введемо реальний сервер і перезапустимо програму. Результат бачимо на рисунку 4.3.

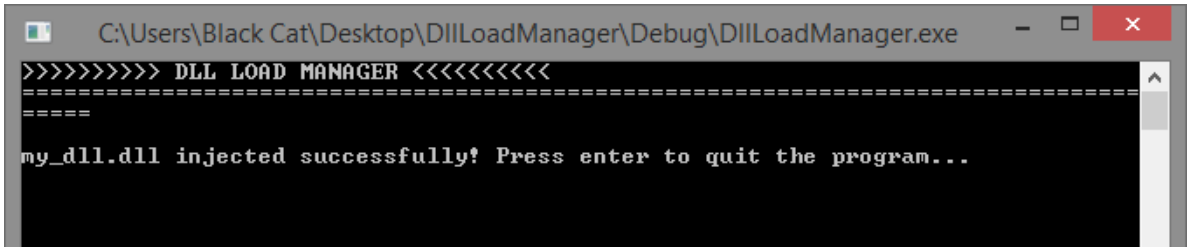


Рисунок 4.3 – Успішне виконання програми

4.2 Розробка інструкції користувача

DllLoadManager вимагає розуміння роботи програми для коректної роботи з нею, тому інструкція користувача є необхідною для даного програмного додатку.

При використанні DllLoadManager необхідно виконати наступні дії:

- заповнити файл Config.ini вірними значеннями полів;
- переконатись, що є підключення до Інтернет;
- переконатись, що вказані сервери доступні;
- переконатись, що користувацька бібліотека знаходиться за вказаною адресою на сервері;
- переконатись, що користувацька програма знаходиться за вказаною локальною адресою;
- переконатись, що всі компоненти програмного додатку розміщені поряд з .exe файлом;
- запустити програмний додаток;
- при виникненні помилок програма виведе повідомлення про помилку та інструкції для виправлення помилки;
- якщо повідомлення про помилку присутнє, виправити помилку згідно наданої інструкції;
- перезапустити програмний додаток;
- дочекатись поки програма завершить виконання.

Отже, якщо слідувати зазначеній інструкції, програма відпрацює коректно. При виникненні помилок програма сповістить про це відповідним повідомленням.

4.3 Висновки

При проведенні тестування отримана повна відповідність вхідних даних і вихідних результатів. Помилок при роботі програми не виявлено, і тому підтверджено її працездатність.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Оцінювання комерційного потенціалу розробки

Метою проведення технологічного аудиту є оцінювання комерційного потенціалу розробки. Для проведення технологічного аудиту було залучено 2-х незалежних експертів. Такими експертами будуть Войтко Вікторія Володимирівна (к.т.н., доц. кафедри ПЗ ВНТУ), Черноволик Галина Олександрівна (к.т.н., доц. кафедри ПЗ ВНТУ).

Здійснюємо оцінювання комерційного потенціалу розробки за 12-ма критеріями за 5-ти бальною шкалою.

Проведено оцінювання комерційного потенціалу за критеріями, наведеними в таблиці 5.1.

Таблиця 5.1 - Критерії оцінювання комерційного потенціалу розробки бальна оцінка

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Кри-терій	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку

Продовження таблиці 5.1

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Кри-тер.	0	1	2	3	4
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна Конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне значне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї

Продовження таблиці 5.1

9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансуванн я ідеї відсутні	Потрібні Незначні фінансові ресурси. Джерела фінансуван ня відсутні	Потрібні значні фінансові ресурси. Джерела фінансува ння є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використов уються у військово-промислово му комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовую ться у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентн их документів та отримання великої кількості дозвільних документів на виробництв о та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництв о та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництв а та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлен ня відповідним органам про виробництв о та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання комерційного потенціалу розробки наведено в таблиці 5.2.

Таблиця 5.2 – Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище, ініціали, посада експерта	
	1. Войтко В. В.	2. Черноволик Г.О.
	Бали, виставлені експертами:	
1	3	2
2	4	3
3	2	3
4	3	3
5	3	2
6	4	3
7	3	3
8	3	4
9	3	3
10	4	4
11	4	4
12	3	3
Сума балів	$СБ_1 = 39$	$СБ_2 = 37$
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_{1}^{3} СБ_i}{2} = 38$	

Отже, з отриманих даних таблиці 5.2 видно, що нова розробка має високий рівень комерційного потенціалу.

5.2 Прогнозування витрат на виконання науково-дослідної роботи та конструкторсько–технологічної роботи.

Для розробки нового програмного продукту необхідні такі витрати.

Основна заробітна плата для розробників визначається за формулою (5.1):

$$З_о = \frac{М}{Т_р} \cdot t, \quad (5.1)$$

де М- місячний посадовий оклад конкретного розробника;

$Т_р$ - кількість робочих днів у місяці, $Т_р$ = 22 дні;

t - число днів роботи розробника, t = 45 днів.

Розрахунки заробітних плат для керівника і програміста наведені в таблиці 5.3.

Таблиця 5.3 – Розрахунки основної заробітної плати

Посада	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату, грн.
Керівник	25000	1136	4	4544
Інженер-програміст	20000	909	20	18180
Всього:				22724

Розрахуємо додаткову заробітну плату:

$$Здод = 0{,}1 \cdot 22724 = 2272{,}4 \text{ (грн.)}$$

Нарахування на заробітну плату операторів НЗП розраховується як 22% від суми їхньої основної та додаткової заробітної плати:

$$Нзп = (Зо + Зр) \cdot \frac{\beta}{100}, \quad (5.2)$$

$$Нзп = (22724 + 2272{,}4) \cdot \frac{22}{100} = 5499{,}2 \text{ (грн.)}.$$

Розрахунок амортизаційних витрат для програмного забезпечення виконується за такою формулою:

$$А = \frac{Ц \cdot Н_а}{100} \cdot \frac{Т}{12}, \quad (5.3)$$

де Ц – балансова вартість обладнання, грн;

$Н_а$ – річна норма амортизаційних відрахувань % (для програмного забезпечення 25%);

Т – Термін використання (Т=3 міс.).

Таблиця 5.4 – Розрахунок амортизаційних відрахувань

Найменування	Ціна, грн.	Норма амортизації, %	Термін використання, м.	Сума амортизації
ПК + прилади керування	9000	20	2	300
Прилади маніпуляції ПК	1000	20	2	33
Всього	333			

Розрахуємо витрати на комплектуючі. Витрати на комплектуючі розрахуємо за формулою:

$$K = \sum_1^n Н_і \cdot Ц_і \cdot К_і, \quad (5.4)$$

де n – кількість комплектуючих;

Ні - кількість комплектуючих і-го виду;

Ці – покупна ціна комплектуючих і-го виду, грн;

Кі – коефіцієнт транспортних витрат (приймемо Кі = 1,1).

Таблиця 5.5 - Витрати на комплектуючі, що були використані для розробки ПЗ.

Найменування матеріалу	Одиниці виміру	Ціна, грн.	Витрачено	Вартість витрачених матеріалів, грн.
Флешка	шт.	180	1	180
Пачка паперу	уп.	115	1	115
Ручка	шт.	5	1	5
Всього з урахуванням транспортних витрат				330

Витрати на силову електроенергію розраховуються за формулою:

$$В_е = В \cdot П \cdot Ф \cdot К_п ; \quad (5.5)$$

де В – вартість 1кВт-години електроенергії (В=8,4 грн/кВт);

П – установлена потужність комп'ютера (П=0,6кВт);

Ф – фактична кількість годин роботи комп'ютера (Ф=200 год.);

$К_п$ – коефіцієнт використання потужності ($К_п$< 1, $К_п = 0,7$).

$$Ве = 8,4 \cdot 0,6 \cdot 200 \cdot 0,7 = 705,6 \text{ (грн.)}$$

Розрахуємо інші витрати $В_{ін}$.

Інші витрати $І_в$ можна прийняти як (100…300)% від суми основної заробітної плати розробників та робітників, які були виконували дану роботу, тобто:

$$В_{ін} = (1..3) \cdot (З_о + З_р). \quad (5.6)$$

Отже, розрахуємо інші витрати:

$$В_{ін} = 1 * (22724 + 2272,4) = 24996,4 \text{ (грн)}.$$

Сума всіх попередніх статей витрат дає витрати на виконання даної частини роботи:

$$В = З_о + З_д + Н_{зп} + А + К + В_е + І_в$$

В = 22724+2272,4+5499,2+333+330+705,6+24996,4= 56860,6 (грн.)

Розрахуємо загальну вартість наукової роботи $B_{заг}$ за формулою:

$$В_{заг} = \frac{В_{ін}}{\alpha} \quad (5.7)$$

де α – частка витрат, які безпосередньо здійснює виконавець даного етапу роботи, у відн. одиницях = 1.

$$В_{заг} = \frac{56860,6}{1} = 56860,6$$

Прогнозування загальних витрат ЗВ на виконання та впровадження результатів виконаної наукової роботи здійснюється за формулою:

$$ЗВ = \frac{В_{заг}}{\beta} \quad (5.8)$$

де β – коефіцієнт, який характеризує етап (стадію) виконання даної роботи.

Отже, розрахуємо загальні витрати:

$$ЗВ = \frac{56860,6}{0,9} = 63178 \text{ (грн.)}$$

5.3 Прогнозування комерційних ефектів від реалізації результатів розробки

Спрогнозуємо отримання прибутку від реалізації результатів нашої розробки. Зростання чистого прибутку можна оцінити у теперішній вартості грошей. Це забезпечить підприємству (організації) надходження додаткових коштів, які дозволять покращити фінансові результати діяльності .

Оцінка зростання чистого прибутку підприємства від впровадження результатів наукової розробки. У цьому випадку збільшення чистого прибутку підприємства $\Delta П_і$ для кожного із років, протягом яких очікується отримання позитивних результатів від впровадження розробки, розраховується за формулою:

$$\Delta П_i = \sum_1^n (\Delta П_я \cdot N + П_я \Delta N)_i \qquad (5.9)$$

де $\Delta П_я$ – покращення основного якісного показника від впровадження результатів розробки у даному році;

N – основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки;

ΔN – покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки;

$П_я$ – основний якісний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

n – кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки.

В результаті впровадження результатів наукової розробки витрати на виготовлення програмного продукту зменшаться на 50 грн (що автоматично спричинить збільшення чистого прибутку підприємства на 50 грн), а кількість користувачів, які будуть користуватись збільшиться: протягом першого року – на 150 користувачів, протягом другого року – на 125 користувачів, протягом третього року – 100 користувачів. Реалізація програмного продукту до впровадження результатів наукової розробки складала 400 користувачів, а прибуток, що отримував розробник до впровадження результатів наукової розробки – 1000 грн.

Спрогнозуємо збільшення чистого прибутку від впровадження результатів наукової розробки у кожному році відносно базового.

Отже, збільшення чистого продукту $\Delta П_1$ протягом першого року складатиме:

$$\Delta П_1 = 50 \cdot 400 + (100 + 50) \cdot 150 = 42500 \text{ грн.}$$

Протягом другого року:

$$\Delta П_2 = 50 \cdot 400 + (100 + 50) \cdot (150 + 125) = 61250 \text{ грн.}$$

Протягом третього року:

$$\Delta П_3 = 50 \cdot 400 + (100 + 50) \cdot (150 + 125 + 100) = 76250 \text{ грн.}$$

5.4 Розрахунок ефективності вкладених інвестицій та період їх окупності

Визначимо абсолютну і відносну ефективність вкладених інвестором інвестицій та розрахуємо термін окупності.

Абсолютна ефективність $Е_{абс}$вкладених інвестицій розраховується за формулою:

$$Е_{абс} = (ПП - PV), \quad (5.10)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДДКР, грн;

т – період часу, протягом якого виявляються результати впровадженої НДДКР, 3 роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,1;

t – період часу (в роках) від моменту отримання чистого прибутку до точки 2, 3,4.

Рисунок, що характеризує рух платежів (інвестицій та додаткових прибутків) буде мати вигляд, рисунок 5.1.

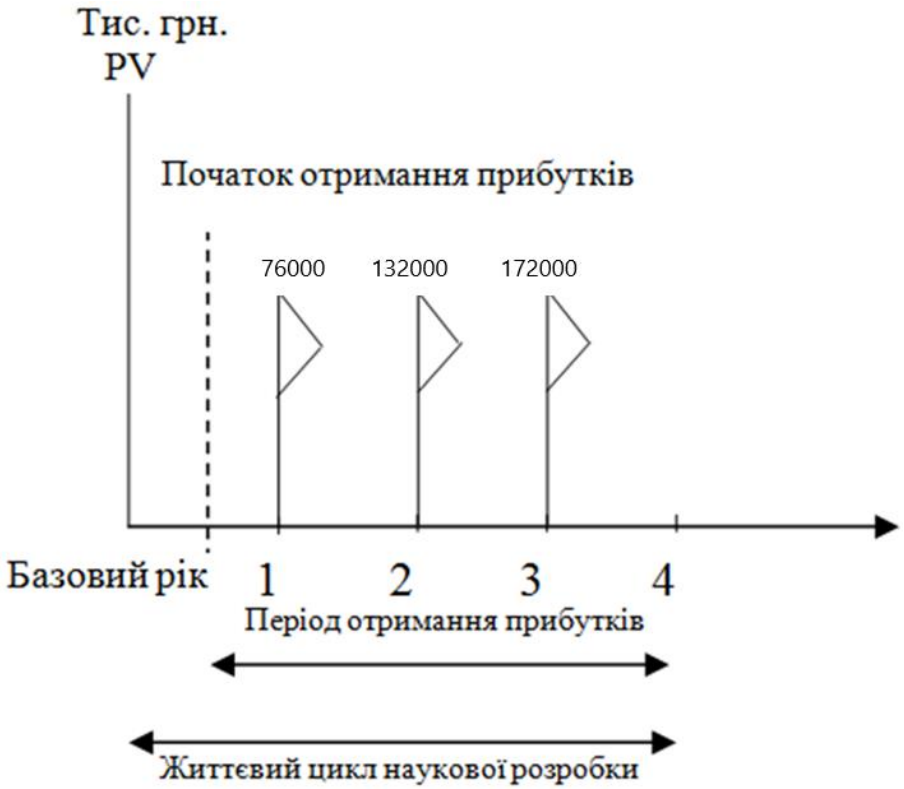


Рисунок 5.1 – Вісь часу з фіксацією платежів, що мають місце під час розробки та впровадження результатів НДДКР

Розрахуємо вартість чистих прибутків за формулою:

$$ПП = \sum_1^m \frac{\Delta П_i}{(1+\tau)^t} \quad (5.11)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДДКР, грн;

т – період часу, протягом якого виявляються результати впровадженої НДДКР, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,1;

t – період часу (в роках) від моменту отримання чистого прибутку до точки.

Отже, розрахуємо вартість чистого прибутку:

$$ПП = \frac{3000}{(1+0,1)^0} + \frac{42500}{(1+0,1)^2} + \frac{61250}{(1+0,1)^3} + \frac{76250}{(1+0,1)^4} = 136221 \text{ (грн.)}$$

Тоді розрахуємо $E_{абс}$:

$$Е_{абс} = 136221 - 63178 = 73043 \text{ грн.}$$

Оскільки $E_{абс} > 0$, то вкладання коштів на виконання та впровадження результатів НДДКР буде доцільним.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $Е_в$ за формулою:

$$Е_в = \sqrt[Т]{1 + \frac{Е_{абс}}{PV}} - 1 \quad (5.12)$$

де $Е_{абс}$ – абсолютна ефективність вкладених інвестицій, грн;

PV – теперішня вартість інвестицій PV = ЗВ, грн;

$Т_ж$ – життєвий цикл наукової розробки, роки.

Тоді будемо мати:

$$E_в = \sqrt[3]{1 + \frac{73043}{63178}} - 1 = 1{,}29 \text{ або } 129\ \%$$

Далі, розраховану величина $E_в$ порівнюємо з мінімальною (бар'єрною) ставкою дисконтування $\tau_{мін}$, яка визначає ту мінімальну дохідність, нижче за яку інвестиції вкладатися не будуть. У загальному вигляді мінімальна (бар'єрна) ставка дисконтування $\tau_{мін}$ визначається за формулою:

$$\tau = d + f,$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2019 році в Україні d = 0,2;

f – показник, що характеризує ризикованість вкладень, величина f = 0,1.

$$\tau = 0{,}2 + 0{,}1 = 0{,}3$$

Оскільки $E_в = 129\% > \tau_{мін} = 0{,}3 = 30\%$, то у інвестор буде зацікавлений вкладати гроші в дану наукову розробку.

Термін окупності вкладених у реалізацію наукового проекту інвестицій. Термін окупності вкладених у реалізацію наукового проекту інвестицій $T_{ок}$ розраховується за формулою:

$$T_{ок} = \frac{1}{1{,}29} = 0{,}78 \text{ року}$$

Обрахувавши термін окупності даної наукової розробки, можна зробити висновок, що фінансування даної наукової розробки буде доцільним.

ВИСНОВКИ

У магістерській кваліфікаційній роботі було розроблено систему захисту програмного забезпечення з динамічним завантаженням виконуваного коду в користувацький процес.

Виконано аналіз основних аналогів, виявлено їхні переваги та недоліки. На основі отриманих результатів було прийнято рішення про створення власного додатку, який би вирішував проблеми, що присутні в аналогах.

Були вирішенні такі задачі:

– удосконалено метод динамічного завантаження програмного продукту або його частин, який передбачає збереження виконуваного коду на сервері, а не на локальній машині користувача, та динамічне завантаження бібліотек у робочий процес, що забезпечує високий рівень захисту виконуваних файлів від несанкціонованого копіювання;

– розроблено динамічно завантажувану бібліотеку, що буде завантажуватись в виконуваний процес та структуровано копіювати вхідну користувацьку бібліотеку;

– розроблено модуль для завантаження користувацької бібліотеки з сервера;

– розроблено модуль для управління процесом для користувацького додатку;

– проведено тестування програмного продукту.

Результати роботи доповідалися на трьох науково-технічних конференціях та опубліковані в чотирьох наукових публікаціях.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Казарин О.В. Теория и практика защиты программ / Олег Казарин – М.: МГУЛ, 2004. – 450 с.- ISBN 5-93517-178-6.

2. Щеглов А.Ю. Защита компьютерной информации от несанкционированного доступа: учебное пособие / Александр Щеглов – Санкт-Петербург: Наука и Техника, 2004. – 384 с. – ISBN 5-318-00244-7.

3. Дудатьєв А.В. Захист програмного забезпечення. Ч.1 : навчальний посібник / Андрій Дудатьєв, Валентина Каплун, Василь Семеренко – Вінниця: ВНТУ, 2005. – 140 с.

4. Войтко В.В. Система захисту програмних додатків з використанням динамічного завантаження виконуваного коду в робочий процес / В.В. Войтко, С.В. Бевз, С.М. Бурбело, А.О. Андреєв // Електронні інформаційні ресурси: створення, використання, доступ // Збірник наукових праць міжнародної науково-практичної Інтернет-конференції. – Вінниця: ВНТУ, січень-2018. – С. 248-253.

5. Войтко В.В. Розробка та провадження на ринок програмного додатку з вивчення хімії / В.В. Войтко, А.О. Андреєв, О.В. Дажура, С.В. Козловський, В.В.Туйчев // Інноваційні та інформаційні технології в бізнесі та освіті // Матеріали міжвузівського студентського вебінару / відп. ред. Л.Б. Ліщинська. – Вінниця: ВТЕІ КНТЕУ. – 21 жовтня 2015р. – С. 6-7.

6. Войтко В.В. Автоматизація процесів формування новин у різних соціальних мережах / В.В. Войтко, С.В. Бевз, А.О. Андреєв, О.В. Дажура, В.В.Туйчев, О.В. Дикий // Електронні інформаційні ресурси: створення, використання, доступ // Збірник наукових праць міжнародної науково-практичної Інтернет-конференції. – Вінниця: ВНТУ, 2015. – С. 91-92

7. Войтко В.В. Комп’ютерна програма «Програмний продукт для автоматизації роботи з різними соціальними мережами» / В.В. Войтко, А.О.Андреєв, О.В. Дикий, О.В. Дажура, В.В. Туйчев // Свідоцтво про реєстрацію авторського права на твір № 64732 від 01.04.2016.

8. VMProtect [Електронний ресурс] – Режим доступу: http://vmpsoft.com/.

9. Private exe Protector [Електронний ресурс] – Режим доступу: http://www.softportal.com/software-5292-private-exe-protector.html.

10. ASProtect [Електронний ресурс] – Режим доступу: http://www.asprotect.ru/.

11. Ripe Exe [Електронний ресурс] – Режим доступу: http://www.softportal.com/software-17273-ripe-exe.html.

12. Кормен Т. Алгоритмы: построение и анализ, 2-е издание.: Пер. с англ. / Т. Кормен – М.: Издательский дом «Вильямс», 2005. – 1296 с.

13. Страуструп Б. Язык программирования C++. Специальное издание: учебное пособие / Б. Страуструп. – Москва : Бином, 2011. – 1136 с.

14. Степанченко И.В. Методы тестирования программного обеспечения: учебное пособие / И.В. Степанченко. – Волгоград : ВолгГТУ, 2006. – 74с.

15. Шилдт Г. C++. Руководство для начинающих / Г. Шилдт – М.: «Вильямс», 2005. – 653с.

16. Інтегроване середовище розробки Microsoft Visual Studio [Електронний ресурс]: режим доступу до матеріалу: http://uk.wikipedia.org/wiki/Microsoft_Visual_Studio

17. Седжвик Р. Алгоритмы на C++. Фундаментальные алгоритмы и структуры данных. / Р. Седжвик — М.: «Вильямс», 2011. — 1056 с.

18. Румянцев П. В. Азбука программирования в Win32 API – «Радио», 1999 – 272 с.

19. Прата С. Язык программирования C++. Лекции и упражнения, 6-е изд.: Пер. с англ. – М.: ООО “И.Д. Вильямс”, 2012. – 1248 с.: ил. – Парал. тит. англ.

20. Хортон А. Visual C++ 2010: полный курс.: Пер. с англ. – М.: ООО“И.Д. Вильямс”, 2011. – 1216 с.: ил. – Парал. тит. англ.

ДОДАТКИ

ДОДАТОК А Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп’ютерної інженерії

УЗГОДЖЕНО
Керівник ТОВ «Он-лайн»
Варшавський Р.В.
“____”_____20___року

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
“____”_____20___року

Технічне завдання
на магістерську кваліфікаційну роботу «Розробка методу і засобів системи захисту даних з використанням технології динамічного завантаження коду в робочий процес» за спеціальністю
121 – Інженерія програмного забезпечення

Керівник магістерської кваліфікаційної роботи:
_____ к.т.н., доцент Войтко В.В.
"____"_____2019 р.
Виконав:
_____студент гр.2ПІ-18м Андреєв А.О.
"____"_____2019 р.

Вінниця – 2019 року

1. Найменування та галузь застосування

Магістерська кваліфікаційна робота: «Розробка методу і засобів системи захисту даних з використанням технології динамічного завантаження коду в робочий процес».

Галузь застосування - системи захисту програмного забезпечення.

2. Підстава для розробки.

Підставою для виконання магістерської кваліфікаційної роботи (МКР) є індивідуальне завдання на МКР та наказ № ректора по ВНТУ про закріплення тем МКР.

3. Мета та призначення розробки.

Метою роботи є підвищення захищеності програмних додатків від несанкціонованого копіювання шляхом розробки динамічно завантажуваної бібліотеки та модуля для управління процесом користувацького додатку, що підвищить захист програм від несанкціонованого доступу.

Призначення роботи – розробка методів і засобів системи захисту даних з використанням технології динамічного завантаження коду в робочий процес.

3 Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись МКР.

1. Казарин О.В. Теория и практика защиты программ / Олег Казарин – М.: МГУЛ, 2004. – 450 с.- ISBN 5-93517-178-6.

2. Щеглов А.Ю. Защита компьютерной информации от несанкционированного доступа: учебное пособие / Александр Щеглов – Санкт-Петербург: Наука и Техника, 2004. – 384 с. – ISBN 5-318-00244-7.

3. Дудатьєв А.В. Захист програмного забезпечення. Ч.1 : навчальний посібник / Андрій Дудатьєв, Валентина Каплун, Василь Семеренко – Вінниця: ВНТУ, 2005. – 140 с.

4. Войтко В.В. Система захисту програмних додатків з використанням динамічного завантаження виконуваного коду в робочий процес / В.В. Войтко, С.В. Бевз, С.М. Бурбело, А.О. Андреєв // Електронні інформаційні ресурси: створення, використання, доступ // Збірник наукових праць міжнародної науково-практичної Інтернет-конференції. – Вінниця: ВНТУ, січень-2018. – С. 248-253.

4. Технічні вимоги

Базові методи захисту програмного забезпечення – метод динамічного завантаження; метод керування дочірнім процесом – Dll Injection; операційна система – Windows 7, 8, 10; об’єм пам’яті на жорсткому диску – 512 МБ; об’єм оперативної пам’яті – 1 ГБ.

5. Конструктивні вимоги.

Система повинна бути зручною у використанні та обслуговуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

6. Перелік технічної документації, що пред’являється по закінченню робіт:

– пояснювальна записка до МКР;

– технічне завдання;

– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів магістерської кваліфікаційної Роботи	Строк виконання етапів роботи
1	Аналіз завдання і вибір методів його вирішення	01.10.19 – 11.10.19
2	Аналіз існуючих аналогів та постановка задач дослідження	12.10.19 – 17.10.19
3	Розробка методу динамічного завантаження	18.10.19 – 01.11.19
4	Розробка структур і алгоритмів програмного продукту	02.11.19 – 10.11.19
5	Розробка програмного забезпечення	11.11.19 – 21.11.19
6	Тестування розробленого програмного продукту	22.11.19 – 25.11.19
7	Економічна частина	26.11.19 – 30.11.19
8	Оформлення матеріалів до захисту МКР	01.12.19 – 08.12.19

10. Порядок контролю та прийняття.

Виконання етапів магістерської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи.

Прийняття магістерської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

ДОДАТОК Б Акт впровадження (ТОВ «Он-лайн»)

АКТ

про впровадження результатів наукової роботи

«Система захисту від несанкціонованого доступу з використанням динамічного завантаження виконуваного коду в робочий процес»

студента ВНТУ Андреєва Андрія Олександровича

Результати магістерської кваліфікаційної роботи «Розробка методу і засобів системи захисту даних з використанням технології динамічного завантаження коду в робочий процес» студента ВНТУ Андреєва А. О., що полягають в розробці:

- методу динамічного завантаження програмного продукту або його частин, який передбачає збереження виконуваного коду на сервері, а не на локальній машині користувача, та динамічне завантаження бібліотек у робочий процес, що забезпечує високий рівень захисту виконуваних файлів від несанкціонованого копіювання;
- моделі та алгоритмів системи захисту програмних додатків від несанкціонованого доступу;
- програмного додатку DllLoadManager, який призначений для завантаження динамічно завантажуваних бібліотек у робочий процес, що, в свою чергу, забезпечує захист виконуваних файлів від викрадення

впроваджені в ТОВ «Он-лайн».

Варшавський Роман Вікторович

15.11.2019

ДОДАТОК В Лістинг головного модуля додатку DllLoadManager

```
include "stdafx.h"
#include "common.h"
#include <assert.h>
#include <windows.h>
#include <tchar.h>
#include <stdio.h>
#include <malloc.h>
#include <string>
#include <vector>

#include "MemoryModule.h"
#include "SharedMemoryManager.h"
#include "Log.h"

typedef int(*addNumberProc)(int, int);

//#define DLL_FILE TEXT("..\\SampleDLL\\SampleDLL.dll")//("A:\\ProtectRegistry.dll")
#define DLL_FILE TEXT("A:\\MemoryModule\\MemoryModule\\example\\Debug\\SampleDLL.dll")//("..\\SampleDLL\\SampleDLL.dll")//("A:\\ProtectRegistry.dll")

void LoadFromFile(void)
{
	addNumberProc addNumber;
	HRSRC resourceInfo;
	DWORD resourceSize;
	LPVOID resourceData;
	TCHAR buffer[100];

	HINSTANCE handle = LoadLibrary(DLL_FILE);
	if (handle == NULL)
		return;

	addNumber = (addNumberProc)GetProcAddress(handle, "addNumbers");
	_tprintf(_T("From file: %d\n"), addNumber(1, 2));

	resourceInfo = FindResource(handle, MAKEINTRESOURCE(VS_VERSION_INFO), RT_VERSION);
	_tprintf(_T("FindResource returned 0x%p\n"), resourceInfo);

	resourceSize = SizeofResource(handle, resourceInfo);
```

```
		resourceData = LoadResource(handle, resourceInfo);
		_tprintf(_T("Resource data: %ld bytes at 0x%p\n"), resourceSize, resourceData);
		LoadString(handle, 1, buffer, sizeof(buffer));
		_tprintf(_T("String1: %s\n"), buffer);
		LoadString(handle, 20, buffer, sizeof(buffer));
		_tprintf(_T("String2: %s\n"), buffer);
		FreeLibrary(handle);
	}

	void* ReadLibrary(size_t* pSize) //считывание бинарного кода с .dll файла в память//
	{
		size_t read;
		void* result;
		FILE* fp;

		//открываем файл для бинарного чтения
		fp = _tfopen(DLL_FILE, _T("rb"));
		if (fp == NULL)
		{
			_tprintf(_T("Can't open DLL file \"%s\"."), DLL_FILE);
			return NULL;
		}

		//подсчитываем количество байт в файле
		fseek(fp, 0, SEEK_END);//перемещение указателя в конец файла
		*pSize = static_cast<size_t>(ftell(fp));//получаем текущие значение "счетчика байт"
- количество байт в файле
		if (*pSize == 0)
		{
			fclose(fp);
			return NULL;
		}
		//выделяем оперативную память под размер .dll файла
		result = (unsigned char *)malloc(*pSize);
		if (result == NULL)
		{
			return NULL;
		}
		//считываем с файла в память
		fseek(fp, 0, SEEK_SET);//перемещаем указатель на начало со смещением 0
		read = fread(result, 1, *pSize, fp);
		fclose(fp);
		if (read != *pSize)
		{
```

```
                free(result);
                return NULL;
        }

        return result;
}

void LoadFromMemory(void)
{
        void *data;
        size_t size;
        HMEMORYMODULE handle;
        addNumberProc addNumber;
        HMEMORYRSRC resourceInfo;
        DWORD resourceSize;
        LPVOID resourceData;
        TCHAR buffer[100];

        //считываем с файла в память бинарный код .dll файла
        data = ReadLibrary(&size);
        if (data == NULL)
        {
                return;
        }

        handle = MemoryLoadLibrary(data, size);
        if (handle == NULL)
        {
                _tprintf(_T("Can't load library from memory.\n"));
                goto exit;
        }

        addNumber = (addNumberProc)MemoryGetProcAddress(handle, "addNumbers");
        _tprintf(_T("From memory: %d\n"), addNumber(1, 2));

        resourceInfo    =    MemoryFindResource(handle,    MAKEINTRESOURCE(VS_VERSION_INFO),
RT_VERSION);
        _tprintf(_T("MemoryFindResource returned 0x%p\n"), resourceInfo);

        resourceSize = MemorySizeofResource(handle, resourceInfo);
        resourceData = MemoryLoadResource(handle, resourceInfo);
        _tprintf(_T("Memory   resource   data:   %ld   bytes   at   0x%p\n"),   resourceSize,
resourceData);
```

```
		MemoryLoadString(handle, 1, buffer, sizeof(buffer));
		_tprintf(_T("String1: %s\n"), buffer);
		MemoryLoadString(handle, 20, buffer, sizeof(buffer));
		_tprintf(_T("String2: %s\n"), buffer);
		MemoryFreeLibrary(handle);

	exit:
		free(data);
	}

	struct CallList {
		int current_alloc_call, current_free_call;
		CustomAllocFunc alloc_calls[MAX_CALLS];
		CustomFreeFunc free_calls[MAX_CALLS];
	};

	LPVOID MemoryFailingAlloc(LPVOID address, SIZE_T size, DWORD allocationType, DWORD
protect, void* userdata)
	{
		UNREFERENCED_PARAMETER(address);
		UNREFERENCED_PARAMETER(size);
		UNREFERENCED_PARAMETER(allocationType);
		UNREFERENCED_PARAMETER(protect);
		UNREFERENCED_PARAMETER(userdata);
		return NULL;
	}

	LPVOID MemoryMockAlloc(LPVOID address, SIZE_T size, DWORD allocationType, DWORD protect,
void* userdata)
	{
		CallList* calls = (CallList*)userdata;
		CustomAllocFunc current_func = calls->alloc_calls[calls->current_alloc_call++];
		assert(current_func != NULL);
		return current_func(address, size, allocationType, protect, NULL);
	}

	BOOL MemoryMockFree(LPVOID lpAddress, SIZE_T dwSize, DWORD dwFreeType, void* userdata)
	{
		CallList* calls = (CallList*)userdata;
		CustomFreeFunc current_func = calls->free_calls[calls->current_free_call++];
		assert(current_func != NULL);
		return current_func(lpAddress, dwSize, dwFreeType, NULL);
	}
```

```
void InitFuncs(void** funcs, va_list args) {
	for (int i = 0; ; i++) {
		assert(i < MAX_CALLS);
		funcs[i] = va_arg(args, void*);
		if (funcs[i] == NULL) break;
	}
}

void InitAllocFuncs(CallList* calls, ...) {
	va_list args;
	va_start(args, calls);
	InitFuncs((void**)calls->alloc_calls, args);
	va_end(args);
	calls->current_alloc_call = 0;
}

void InitFreeFuncs(CallList* calls, ...) {
	va_list args;
	va_start(args, calls);
	InitFuncs((void**)calls->free_calls, args);
	va_end(args);
	calls->current_free_call = 0;
}

void InitFreeFunc(CallList* calls, CustomFreeFunc freeFunc) {
	for (int i = 0; i < MAX_CALLS; i++) {
		calls->free_calls[i] = freeFunc;
	}
	calls->current_free_call = 0;
}

void TestFailingAllocation(void *data, size_t size) {
	CallList expected_calls;
	HMEMORYMODULE handle;

	InitAllocFuncs(&expected_calls, MemoryFailingAlloc, MemoryFailingAlloc, NULL);
	InitFreeFuncs(&expected_calls, NULL);

	handle = MemoryLoadLibraryEx(
		data, size, MemoryMockAlloc, MemoryMockFree, MemoryDefaultLoadLibrary,
		MemoryDefaultGetProcAddress, MemoryDefaultFreeLibrary, &expected_calls);

	assert(handle == NULL);
	assert(GetLastError() == ERROR_OUTOFMEMORY);
```

```
	assert(expected_calls.current_free_call == 0);

	MemoryFreeLibrary(handle);
	assert(expected_calls.current_free_call == 0);
}

void TestCleanupAfterFailingAllocation(void *data, size_t size) {
	CallList expected_calls;
	HMEMORYMODULE handle;
	int free_calls_after_loading;
	InitAllocFuncs(&expected_calls,
		MemoryDefaultAlloc,
		MemoryDefaultAlloc,
		MemoryDefaultAlloc,
		MemoryDefaultAlloc,
		MemoryFailingAlloc,
		NULL);
	InitFreeFuncs(&expected_calls, MemoryDefaultFree, NULL);
	handle = MemoryLoadLibraryEx(
		data, size, MemoryMockAlloc, MemoryMockFree, MemoryDefaultLoadLibrary,
		MemoryDefaultGetProcAddress, MemoryDefaultFreeLibrary, &expected_calls);
	free_calls_after_loading = expected_calls.current_free_call;
	MemoryFreeLibrary(handle);
	assert(expected_calls.current_free_call == free_calls_after_loading);
}

void TestFreeAfterDefaultAlloc(void *data, size_t size) {
	CallList expected_calls;
	HMEMORYMODULE handle;
	int free_calls_after_loading;
	// Note: free might get called internally multiple times
	InitFreeFunc(&expected_calls, MemoryDefaultFree);
	handle = MemoryLoadLibraryEx(
		data, size, MemoryDefaultAlloc, MemoryMockFree, MemoryDefaultLoadLibrary,
		MemoryDefaultGetProcAddress, MemoryDefaultFreeLibrary, &expected_calls);
	assert(handle != NULL);
	free_calls_after_loading = expected_calls.current_free_call;
	MemoryFreeLibrary(handle);
	assert(expected_calls.current_free_call == free_calls_after_loading + 1);
}

#ifdef _WIN64
```

```
	LPVOID MemoryAllocHigh(LPVOID address, SIZE_T size, DWORD allocationType, DWORD protect,
void* userdata)
	{
		int* counter = static_cast<int*>(userdata);
		if (*counter == 0) {
			// Make sure the image gets loaded to an address above 32bit.
			uintptr_t offset = 0x10000000000;
			address = (LPVOID)((uintptr_t)address + offset);
		}
		(*counter)++;
		return MemoryDefaultAlloc(address, size, allocationType, protect, NULL);
	}

	void TestAllocHighMemory(void *data, size_t size) {
		HMEMORYMODULE handle;
		int counter = 0;
		addNumberProc addNumber;
		HMEMORYRSRC resourceInfo;
		DWORD resourceSize;
		LPVOID resourceData;
		TCHAR buffer[100];

		handle = MemoryLoadLibraryEx(
			data, size, MemoryAllocHigh, MemoryDefaultFree, MemoryDefaultLoadLibrary,
			MemoryDefaultGetProcAddress, MemoryDefaultFreeLibrary, &counter);

		assert(handle != NULL);

		addNumber = (addNumberProc)MemoryGetProcAddress(handle, "addNumbers");
		_tprintf(_T("From memory: %d\n"), addNumber(1, 2));

		resourceInfo = MemoryFindResource(handle, MAKEINTRESOURCE(VS_VERSION_INFO),
RT_VERSION);
		_tprintf(_T("MemoryFindResource returned 0x%p\n"), resourceInfo);

		resourceSize = MemorySizeofResource(handle, resourceInfo);
		resourceData = MemoryLoadResource(handle, resourceInfo);
		_tprintf(_T("Memory resource data: %ld bytes at 0x%p\n"), resourceSize,
resourceData);

		MemoryLoadString(handle, 1, buffer, sizeof(buffer));
		_tprintf(_T("String1: %s\n"), buffer);

		MemoryLoadString(handle, 20, buffer, sizeof(buffer));
```

```
		_tprintf(_T("String2: %s\n"), buffer);

		MemoryFreeLibrary(handle);
	}
	#endif  // _WIN64

	void TestCustomAllocAndFree(void)
	{
		void *data;
		size_t size;

		data = ReadLibrary(&size);
		if (data == NULL)
		{
			return;
		}

		_tprintf(_T("Test MemoryLoadLibraryEx after initially failing allocation function\n"));
		TestFailingAllocation(data, size);
		_tprintf(_T("Test cleanup after MemoryLoadLibraryEx with failing allocation function\n"));
		TestCleanupAfterFailingAllocation(data, size);
		_tprintf(_T("Test custom free function after MemoryLoadLibraryEx\n"));
		TestFreeAfterDefaultAlloc(data, size);
	#ifdef _WIN64
		_tprintf(_T("Test allocating in high memory\n"));
		TestAllocHighMemory(data, size);
	#endif

		free(data);
	}
	std::vector<std::wstring> Split(std::wstring str, std::wstring delimeter)
	{
	std::vector<std::wstring> splittedArgs;
		try
		{
			size_t start = 0;
			int index = str.find(delimeter);

			if (index == std::wstring::npos)
			{
				splittedArgs.push_back(str);
			}
```

```
		else
		{
			while (index != std::wstring::npos)
			{
				splittedArgs.push_back(str.substr(start, index - start));
				start = index + delimeter.length();
				if (start < str.length())
				{
					index = str.find(delimeter, start);
					if (index == std::wstring::npos)
					{
						splittedArgs.push_back(str.substr(start));
					}
				}
				else
				{
					index = -1;
				}
			}
		}
	}
	catch (...)
	{
	}
	return splittedArgs;
}
class Dll {
public:
	LPVOID address;
	DWORD size;
	std::wstring name;
	std::wstring contenct_wstring;
	Dll() {
		this->address = NULL;
		this->size = 0;
		this->name = L"Unknow";
	}
	~Dll() {}
	static Dll* GetDllFromArgsString(std::wstring args) {
	std::vector<std::wstring> splitted_args;
		splitted_args = Split(args, L" ");
		if (splitted_args.size() != 3)
		{
```

```
                        Log::Write(L"ERROR", L"GetDllFromArgsString: Got invalid string \""
+ args + L"\"");
                        return NULL;
                }

                Dll* ret = new Dll;
                ret->name = splitted_args[0];
                ret->size = _wtoi(splitted_args[1].c_str());
                //ret->contenct_wstring = splitted_args[2];
                ret->address = (LPVOID)_wtoi(splitted_args[2].c_str());
                return ret;
        };
};

BOOL APIENTRY DllMain(HMODULE hModule, DWORD  ul_reason_for_call, LPVOID lpReserved)
{
        std::wstring args; /*= L"qwe 123145 123|--|asd 567567 456|--|zxc 7567546 6575";*/
        std::vector<std::wstring> splitted_args;
        switch (ul_reason_for_call)
        {
        case DLL_PROCESS_ATTACH: {
                Log::Write(L"FLAG", L"DLL_PROCESS_ATTACH");
                SharedMemoryManager::OpenMappingFile(GetCurrentProcessId());
                args = std::wstring(SharedMemoryManager::GetData());
                SharedMemoryManager::UnmappFile();
                Log::Write(L"FLAG", L"Loading process got args \"" + args + L"\"");
                splitted_args = Split(args, L"|--|");
                Dll** dlls = new Dll*[splitted_args.size()];
                //создали список длок, которые нужно инжектить
                for (int i = 0; i < splitted_args.size(); ++i)
                {
                        dlls[i] = Dll::GetDllFromArgsString(splitted_args[i]);
                }
                //загружаем длки в память соответственно списку
                for (int i = 0; i < splitted_args.size(); ++i)
                {
                if (dlls[i] != NULL) {
                                Log::Write(L"FLAG",  L"MemoryLoadLibrary:     Loading  "  +
dlls[i]->name + L" library. Addres: " + std::to_wstring((int)dlls[i]->address) + L", size: " +
std::to_wstring((int)dlls[i]->size));
                                if  (MemoryLoadLibrary(dlls[i]->address,  dlls[i]->size)  !=
NULL)
                                {
```

```
                                        Log::Write(L"FLAG", L"MemoryLoadLibrary:   Loaded " +
dlls[i]->name + L" library. Addres: " + std::to_wstring((int)dlls[i]->address) + L", size: " +
std::to_wstring((int)dlls[i]->size));
                                }
                                else
                                {
                                        Log::Write(L"ERROR", L"MemoryLoadLibrary: Loading " +
dlls[i]->name + L" failed.");
                                }
                        }
                }
        }
                                                        break;
        case DLL_THREAD_ATTACH:
                break;
        case DLL_THREAD_DETACH:
                break;
        case DLL_PROCESS_DETACH:
                break;
        }
        return TRUE;
}

int main()
{
        bool res = DllMain(0, DLL_PROCESS_ATTACH, 0);
        return 0;
}
```

ДОДАТОК Г Лістинг модуля управління поширеною пам’яттю SharedMemoryManager

```
#pragma unmanaged
#include "stdafx.h"
#include "common.h"
#include "SharedMemoryManager.h"
#include <stdexcept>
#include <sstream>
#include <iostream>
#include "Log.h"

HANDLE SharedMemoryManager::hMFile;
LPCWSTR SharedMemoryManager::pBuffer;
LPWSTR SharedMemoryManager::szNameDLM = L"Global\\DllLoadManagerMappingObject";
LPWSTR SharedMemoryManager::szNameOutputDLM = L"Global\\DllLoadManagerMappingObjectInput";
HANDLE SharedMemoryManager::hMFileOutput;
LPCWSTR SharedMemoryManager::pBufferOutput;

void SharedMemoryManager::OpenMappingFile(int pid)
{
	std::wstring name = szNameDLM;
	name += std::to_wstring(pid);
	Log::Write(L"FLAG", L"SharedMemoryManager::OpenMappingFile: name = " + name);
	hMFile = OpenFileMapping(
		FILE_MAP_ALL_ACCESS,
		FALSE,
		name.c_str());
	if (hMFile == NULL) {
		Log::Write(L"FLAG", L"SharedMemoryManager::OpenMappingFile: hMFile == NULL");
	}
	std::wstring nameOutput = szNameOutputDLM;
	nameOutput += std::to_wstring(pid);
	Log::Write(L"FLAG", L"SharedMemoryManager::OpenMappingFile: nameOutput = " + nameOutput);
	hMFileOutput = OpenFileMapping(
		FILE_MAP_ALL_ACCESS,
		FALSE,
		nameOutput.c_str());
	if (hMFileOutput == NULL) {
		Log::Write(L"FLAG", L"SharedMemoryManager::OpenMappingFile: hMFileOutput == NULL");
```

```
		}
		if (hMFile == NULL || hMFileOutput == NULL)
		{
			std::wstringstream ss;
			ss << "Could not open file mapping object " << GetLastError();
			Log::Write(L"FLAG", L"SharedMemoryManager::OpenMappingFile: hMFile == NULL
|| hMFileOutput == NULL");
		}
	}
	LPCWSTR SharedMemoryManager::GetData()
	{
		pBuffer = (LPCWSTR)MapViewOfFile(hMFile,
			FILE_MAP_ALL_ACCESS,
			0,
			0,
			BUF_SIZE);
		if (pBuffer == NULL)
		{
			std::wstringstream ss;
			ss << "Could not map view of file " << GetLastError();
			Log::Write(L"FLAG", L"SharedMemoryManager::GetData: pBuffer == NULL");
			CloseHandle(hMFile);
		}
		return pBuffer;
	}
	void SharedMemoryManager::CopyMessageToFile(LPWSTR message)
	{
		pBufferOutput = (LPCWSTR)MapViewOfFile(hMFileOutput,
			FILE_MAP_ALL_ACCESS,
			0,
			0,
			BUF_SIZE);
		CopyMemory((PVOID)pBufferOutput, message, (_tcslen(message) * sizeof(TCHAR)));
		UnmappFile();
	}
	void SharedMemoryManager::UnmappFile()
	{
		UnmapViewOfFile(pBuffer);
		UnmapViewOfFile(pBufferOutput);
		CloseHandle(hMFile);
		CloseHandle(hMFileOutput);
	}
	#pragma managed
```

ДОДАТОК Д Лістинг модуля розпаковки виконуваного коду UnpackModule

```
#include "common.h"
#include <windows.h>
#include <winnt.h>
#include <stddef.h>
#include <tchar.h>

#ifdef DEBUG_OUTPUT
#include <stdio.h>
#endif

#if _MSC_VER
#pragma warning(disable:4055)
#pragma warning(error: 4244)
#pragma warning(error: 4267)

#define inline __inline
#endif

#ifndef IMAGE_SIZEOF_BASE_RELOCATION
#define IMAGE_SIZEOF_BASE_RELOCATION (sizeof(IMAGE_BASE_RELOCATION))
#endif

#ifdef _WIN64
#define HOST_MACHINE IMAGE_FILE_MACHINE_AMD64
#else
#define HOST_MACHINE IMAGE_FILE_MACHINE_I386
#endif

#include "UnpackModule.h"

typedef BOOL (WINAPI *DllEntryProc)(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID
lpReserved);
typedef int (WINAPI *ExeEntryProc)(void);

struct UNPACKMODULE {
    PIMAGE_NT_HEADERS headers;
    unsigned char *codeBase;
    HCUSTOMMODULE *modules;
    int numModules;
    BOOL initialized;
```

```
    BOOL isDLL;
    BOOL isRelocated;
    CustomAllocFunc alloc;
    CustomFreeFunc free;
    CustomLoadLibraryFunc loadLibrary;
    CustomGetProcAddressFunc getProcAddress;
    CustomFreeLibraryFunc freeLibrary;
    void *userdata;
    ExeEntryProc exeEntry;
    DWORD pageSize;
};

struct SECTIONFINALIZEDATA {
    LPVOID address;
    LPVOID alignedAddress;
    SIZE_T size;
    DWORD characteristics;
    BOOL last;
};

#define GET_HEADER_DICTIONARY(module, idx) &(module)->headers->OptionalHeader.DataDirectory[idx]

static inline uintptr_t
AlignValueDown(uintptr_t value, uintptr_t alignment) {
    return value & ~(alignment - 1);
}

static inline LPVOID
AlignAddressDown(LPVOID address, uintptr_t alignment) {
    return (LPVOID) AlignValueDown((uintptr_t) address, alignment);
}

static inline size_t
AlignValueUp(size_t value, size_t alignment) {
    return (value + alignment - 1) & ~(alignment - 1);
}

static inline void*
OffsetPointer(void* data, ptrdiff_t offset) {
    return (void*) ((uintptr_t) data + offset);
}

static inline void
```

```
        OutputLastError(const char *msg)
        {
        #ifndef DEBUG_OUTPUT
            UNREFERENCED_PARAMETER(msg);
        #else
            LPVOID tmp;
            char *tmpmsg;
            FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER | FORMAT_MESSAGE_FROM_SYSTEM |
FORMAT_MESSAGE_IGNORE_INSERTS,
                NULL, GetLastError(), MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), (LPTSTR)&tmp, 0,
NULL);
            tmpmsg = (char *)LocalAlloc(LPTR, strlen(msg) + strlen(tmp) + 3);
            sprintf(tmpmsg, "%s: %s", msg, tmp);
            OutputDebugString(tmpmsg);
            LocalFree(tmpmsg);
            LocalFree(tmp);
        #endif
        }

        static BOOL
        CheckSize(size_t size, size_t expected) {
            if (size < expected) {
                SetLastError(ERROR_INVALID_DATA);
                return FALSE;
            }

            return TRUE;
        }

        static BOOL
        CopySections(const unsigned char *data, size_t size, PIMAGE_NT_HEADERS old_headers,
PUNPACKMODULE module)
        {
            int i, section_size;
            unsigned char *codeBase = module->codeBase;
            unsigned char *dest;
            PIMAGE_SECTION_HEADER section = IMAGE_FIRST_SECTION(module->headers);
            for (i=0; i<module->headers->FileHeader.NumberOfSections; i++, section++) {
                if (section->SizeOfRawData == 0) {
                    section_size = old_headers->OptionalHeader.SectionAlignment;
                    if (section_size > 0) {
                        dest = (unsigned char *)module->alloc(codeBase + section->VirtualAddress,
                            section_size,
                            MEM_COMMIT,
```

```
                    PAGE_READWRITE,
                    module->userdata);
                if (dest == NULL) {
                    return FALSE;
                }

                dest = codeBase + section->VirtualAddress;
                section->Misc.PhysicalAddress = (DWORD) ((uintptr_t) dest & 0xffffffff);
                memset(dest, 0, section_size);
            }

            continue;
        }

        if (!CheckSize(size, section->PointerToRawData + section->SizeOfRawData)) {
            return FALSE;
        }

        dest = (unsigned char *)module->alloc(codeBase + section->VirtualAddress,
                        section->SizeOfRawData,
                        MEM_COMMIT,
                        PAGE_READWRITE,
                        module->userdata);
        if (dest == NULL) {
            return FALSE;
        }

        dest = codeBase + section->VirtualAddress;
        memcpy(dest, data + section->PointerToRawData, section->SizeOfRawData);
        section->Misc.PhysicalAddress = (DWORD) ((uintptr_t) dest & 0xffffffff);
    }

    return TRUE;
}

static int ProtectionFlags[2][2][2] = {
    {
        {PAGE_NOACCESS, PAGE_WRITECOPY},
        {PAGE_READONLY, PAGE_READWRITE},
    }, {
        {PAGE_EXECUTE, PAGE_EXECUTE_WRITECOPY},
        {PAGE_EXECUTE_READ, PAGE_EXECUTE_READWRITE},
    },
};
```

```
        static SIZE_T
        GetRealSectionSize(PUNPACKMODULE module, PIMAGE_SECTION_HEADER section) {
            DWORD size = section->SizeOfRawData;
            if (size == 0) {
                if (section->Characteristics & IMAGE_SCN_CNT_INITIALIZED_DATA) {
                    size = module->headers->OptionalHeader.SizeOfInitializedData;
                } else if (section->Characteristics & IMAGE_SCN_CNT_UNINITIALIZED_DATA) {
                    size = module->headers->OptionalHeader.SizeOfUninitializedData;
                }
            }
            return (SIZE_T) size;
        }

        static BOOL
        FinalizeSection(PUNPACKMODULE module, PSECTIONFINALIZEDATA sectionData) {
            DWORD protect, oldProtect;
            BOOL executable;
            BOOL readable;
            BOOL writeable;

            if (sectionData->size == 0) {
                return TRUE;
            }

            if (sectionData->characteristics & IMAGE_SCN_MEM_DISCARDABLE) {
                if (sectionData->address == sectionData->alignedAddress &&
                    (sectionData->last ||
                     module->headers->OptionalHeader.SectionAlignment == module->pageSize ||
                     (sectionData->size % module->pageSize) == 0)
                   ) {
                    module->free(sectionData->address,  sectionData->size,  MEM_DECOMMIT,  module-
>userdata);
                }
                return TRUE;
            }

            executable = (sectionData->characteristics & IMAGE_SCN_MEM_EXECUTE) != 0;
            readable =   (sectionData->characteristics & IMAGE_SCN_MEM_READ) != 0;
            writeable =  (sectionData->characteristics & IMAGE_SCN_MEM_WRITE) != 0;
            protect = ProtectionFlags[executable][readable][writeable];
            if (sectionData->characteristics & IMAGE_SCN_MEM_NOT_CACHED) {
                protect |= PAGE_NOCACHE;
            }
```

```
            if (VirtualProtect(sectionData->address, sectionData->size, protect, &oldProtect) ==
0) {
                OutputLastError("Error protecting memory page");
                return FALSE;
            }

            return TRUE;
        }

        static BOOL
        FinalizeSections(PUNPACKMODULE module)
        {
            int i;
            PIMAGE_SECTION_HEADER section = IMAGE_FIRST_SECTION(module->headers);
        #ifdef _WIN64
            uintptr_t  imageOffset  =  ((uintptr_t)  module->headers->OptionalHeader.ImageBase  &
0xffffffff00000000);
        #else
            static const uintptr_t imageOffset = 0;
        #endif
            SECTIONFINALIZEDATA sectionData;
            sectionData.address     =     (LPVOID)((uintptr_t)section->Misc.PhysicalAddress     |
imageOffset);
            sectionData.alignedAddress = AlignAddressDown(sectionData.address, module->pageSize);
            sectionData.size = GetRealSectionSize(module, section);
            sectionData.characteristics = section->Characteristics;
            sectionData.last = FALSE;
            section++;

            for (i=1; i<module->headers->FileHeader.NumberOfSections; i++, section++) {
                LPVOID   sectionAddress   =   (LPVOID)((uintptr_t)section->Misc.PhysicalAddress   |
imageOffset);
                LPVOID alignedAddress = AlignAddressDown(sectionAddress, module->pageSize);
                SIZE_T sectionSize = GetRealSectionSize(module, section);
                if     (sectionData.alignedAddress     ==     alignedAddress     ||     (uintptr_t)
sectionData.address + sectionData.size > (uintptr_t) alignedAddress) {
                    if   ((section->Characteristics   &   IMAGE_SCN_MEM_DISCARDABLE)   ==   0   ||
(sectionData.characteristics & IMAGE_SCN_MEM_DISCARDABLE) == 0) {
                        sectionData.characteristics  =  (sectionData.characteristics  |  section-
>Characteristics) & ~IMAGE_SCN_MEM_DISCARDABLE;
                    } else {
                        sectionData.characteristics |= section->Characteristics;
                    }
```

```
                sectionData.size = (((uintptr_t)sectionAddress) + ((uintptr_t) sectionSize)) -
(uintptr_t) sectionData.address;
                continue;
            }

            if (!FinalizeSection(module, &sectionData)) {
                return FALSE;
            }
            sectionData.address = sectionAddress;
            sectionData.alignedAddress = alignedAddress;
            sectionData.size = sectionSize;
            sectionData.characteristics = section->Characteristics;
        }
        sectionData.last = TRUE;
        if (!FinalizeSection(module, &sectionData)) {
            return FALSE;
        }
        return TRUE;
    }

    static BOOL
    ExecuteTLS(PUNPACKMODULE module)
    {
        unsigned char *codeBase = module->codeBase;
        PIMAGE_TLS_DIRECTORY tls;
        PIMAGE_TLS_CALLBACK* callback;

        PIMAGE_DATA_DIRECTORY        directory        =        GET_HEADER_DICTIONARY(module,
IMAGE_DIRECTORY_ENTRY_TLS);
        if (directory->VirtualAddress == 0) {
            return TRUE;
        }

        tls = (PIMAGE_TLS_DIRECTORY) (codeBase + directory->VirtualAddress);
        callback = (PIMAGE_TLS_CALLBACK *) tls->AddressOfCallBacks;
        if (callback) {
            while (*callback) {
                (*callback)((LPVOID) codeBase, DLL_PROCESS_ATTACH, NULL);
                callback++;
            }
        }
        return TRUE;
    }
```

```
static BOOL
PerformBaseRelocation(PUNPACKMODULE module, ptrdiff_t delta)
{
    unsigned char *codeBase = module->codeBase;
    PIMAGE_BASE_RELOCATION relocation;

    PIMAGE_DATA_DIRECTORY directory = GET_HEADER_DICTIONARY(module, IMAGE_DIRECTORY_ENTRY_BASERELOC);
    if (directory->Size == 0) {
        return (delta == 0);
    }

    relocation = (PIMAGE_BASE_RELOCATION) (codeBase + directory->VirtualAddress);
    for (; relocation->VirtualAddress > 0; ) {
        DWORD i;
        unsigned char *dest = codeBase + relocation->VirtualAddress;
        unsigned short *relInfo = (unsigned short*) OffsetPointer(relocation, IMAGE_SIZEOF_BASE_RELOCATION);
        for (i=0; i<((relocation->SizeOfBlock-IMAGE_SIZEOF_BASE_RELOCATION) / 2); i++, relInfo++) {
            int type = *relInfo >> 12;
            int offset = *relInfo & 0xfff;

            switch (type)
            {
            case IMAGE_REL_BASED_ABSOLUTE:
                break;

            case IMAGE_REL_BASED_HIGHLOW:
                {
                    DWORD *patchAddrHL = (DWORD *) (dest + offset);
                    *patchAddrHL += (DWORD) delta;
                }
                break;

#ifdef _WIN64
            case IMAGE_REL_BASED_DIR64:
                {
                    ULONGLONG *patchAddr64 = (ULONGLONG *) (dest + offset);
                    *patchAddr64 += (ULONGLONG) delta;
                }
                break;
#endif
```

```
                    default:
                        break;
                    }
                }

                relocation = (PIMAGE_BASE_RELOCATION) OffsetPointer(relocation, relocation->SizeOfBlock);
            }
            return TRUE;
        }

        static BOOL
        BuildImportTable(PUNPACKMODULE module)
        {
            unsigned char *codeBase = module->codeBase;
            PIMAGE_IMPORT_DESCRIPTOR importDesc;
            BOOL result = TRUE;

            PIMAGE_DATA_DIRECTORY directory = GET_HEADER_DICTIONARY(module, IMAGE_DIRECTORY_ENTRY_IMPORT);
            if (directory->Size == 0) {
                return TRUE;
            }

            importDesc = (PIMAGE_IMPORT_DESCRIPTOR) (codeBase + directory->VirtualAddress);
            for (; !IsBadReadPtr(importDesc, sizeof(IMAGE_IMPORT_DESCRIPTOR)) && importDesc->Name; importDesc++) {
                uintptr_t *thunkRef;
                FARPROC *funcRef;
                HCUSTOMMODULE *tmp;
                HCUSTOMMODULE handle = module->loadLibrary((LPCSTR) (codeBase + importDesc->Name), module->userdata);
                if (handle == NULL) {
                    SetLastError(ERROR_MOD_NOT_FOUND);
                    result = FALSE;
                    break;
                }

                tmp = (HCUSTOMMODULE *) realloc(module->modules, (module->numModules+1)*(sizeof(HCUSTOMMODULE)));
                if (tmp == NULL) {
                    module->freeLibrary(handle, module->userdata);
                    SetLastError(ERROR_OUTOFMEMORY);
                    result = FALSE;
```

```
                break;
            }
            module->modules = tmp;

            module->modules[module->numModules++] = handle;
            if (importDesc->OriginalFirstThunk) {
                thunkRef = (uintptr_t *) (codeBase + importDesc->OriginalFirstThunk);
                funcRef = (FARPROC *) (codeBase + importDesc->FirstThunk);
            } else {
                thunkRef = (uintptr_t *) (codeBase + importDesc->FirstThunk);
                funcRef = (FARPROC *) (codeBase + importDesc->FirstThunk);
            }
            for (; *thunkRef; thunkRef++, funcRef++) {
                if (IMAGE_SNAP_BY_ORDINAL(*thunkRef)) {
                    *funcRef = module->getProcAddress(handle,
(LPCSTR)IMAGE_ORDINAL(*thunkRef), module->userdata);
                } else {
                    PIMAGE_IMPORT_BY_NAME thunkData = (PIMAGE_IMPORT_BY_NAME) (codeBase +
(*thunkRef));
                    *funcRef = module->getProcAddress(handle, (LPCSTR)&thunkData->Name,
module->userdata);
                }
                if (*funcRef == 0) {
                    result = FALSE;
                    break;
                }
            }

            if (!result) {
                module->freeLibrary(handle, module->userdata);
                SetLastError(ERROR_PROC_NOT_FOUND);
                break;
            }
        }

        return result;
    }

    LPVOID MemoryDefaultAlloc(LPVOID address, SIZE_T size, DWORD allocationType, DWORD
protect, void* userdata)
    {
        UNREFERENCED_PARAMETER(userdata);
        return VirtualAlloc(address, size, allocationType, protect);
    }
```

```
BOOL MemoryDefaultFree(LPVOID lpAddress, SIZE_T dwSize, DWORD dwFreeType, void* userdata)
{
	UNREFERENCED_PARAMETER(userdata);
	return VirtualFree(lpAddress, dwSize, dwFreeType);
}

HCUSTOMMODULE MemoryDefaultLoadLibrary(LPCSTR filename, void *userdata)
{
    HMODULE result;
    UNREFERENCED_PARAMETER(userdata);
    result = LoadLibraryA(filename);
    if (result == NULL) {
        return NULL;
    }

    return (HCUSTOMMODULE) result;
}

FARPROC MemoryDefaultGetProcAddress(HCUSTOMMODULE module, LPCSTR name, void *userdata)
{
    UNREFERENCED_PARAMETER(userdata);
    return (FARPROC) GetProcAddress((HMODULE) module, name);
}

void MemoryDefaultFreeLibrary(HCUSTOMMODULE module, void *userdata)
{
    UNREFERENCED_PARAMETER(userdata);
    FreeLibrary((HMODULE) module);
}

HUNPACKMODULE MemoryLoadLibrary(const void *data, size_t size)
{

    return MemoryLoadLibraryEx(data, size, MemoryDefaultAlloc, MemoryDefaultFree,
MemoryDefaultLoadLibrary, MemoryDefaultGetProcAddress, MemoryDefaultFreeLibrary, NULL);
}



HUNPACKMODULE MemoryLoadLibraryEx(const void *data, size_t size,
    CustomAllocFunc allocMemory,
    CustomFreeFunc freeMemory,
    CustomLoadLibraryFunc loadLibrary,
```

```
            CustomGetProcAddressFunc getProcAddress,
            CustomFreeLibraryFunc freeLibrary,
            void *userdata)
        {
            PUNPACKMODULE result = NULL;
            PIMAGE_DOS_HEADER dos_header;
            PIMAGE_NT_HEADERS old_header;
            unsigned char *code, *headers;
            ptrdiff_t locationDelta;
            SYSTEM_INFO sysInfo;
            PIMAGE_SECTION_HEADER section;
            DWORD i;
            size_t optionalSectionSize;
            size_t lastSectionEnd = 0;
            size_t alignedImageSize;

            if (!CheckSize(size, sizeof(IMAGE_DOS_HEADER))) {
                return NULL;
            }
            dos_header = (PIMAGE_DOS_HEADER)data;

            if (dos_header->e_magic != IMAGE_DOS_SIGNATURE) {
                SetLastError(ERROR_BAD_EXE_FORMAT);
                return NULL;
            }

            if (!CheckSize(size, dos_header->e_lfanew + sizeof(IMAGE_NT_HEADERS))) {
                      return NULL;
            }

            old_header   =   (PIMAGE_NT_HEADERS)&((const   unsigned   char   *)(data))[dos_header-
>e_lfanew];
            if (old_header->Signature != IMAGE_NT_SIGNATURE) {
                SetLastError(ERROR_BAD_EXE_FORMAT);
                      return NULL;
            }

            if (old_header->FileHeader.Machine != HOST_MACHINE) {
                SetLastError(ERROR_BAD_EXE_FORMAT);
                      return NULL;
            }

            if (old_header->OptionalHeader.SectionAlignment & 1) {
                SetLastError(ERROR_BAD_EXE_FORMAT);
```

```
            return NULL;
        }


        section = IMAGE_FIRST_SECTION(old_header);
        optionalSectionSize = old_header->OptionalHeader.SectionAlignment;
        for (i=0; i<old_header->FileHeader.NumberOfSections; i++, section++) {
            size_t endOfSection;
            if (section->SizeOfRawData == 0) {
                endOfSection = section->VirtualAddress + optionalSectionSize;
            } else {
                endOfSection = section->VirtualAddress + section->SizeOfRawData;
            }

            if (endOfSection > lastSectionEnd) {
                lastSectionEnd = endOfSection;
            }
        }

        GetNativeSystemInfo(&sysInfo);
        alignedImageSize        =        AlignValueUp(old_header->OptionalHeader.SizeOfImage,
sysInfo.dwPageSize);
        if (alignedImageSize != AlignValueUp(lastSectionEnd, sysInfo.dwPageSize)) {
            SetLastError(ERROR_BAD_EXE_FORMAT);
                return NULL;
        }

        code = (unsigned char *)allocMemory((LPVOID)(old_header->OptionalHeader.ImageBase),
            alignedImageSize,
            MEM_RESERVE | MEM_COMMIT,
            PAGE_READWRITE,
            userdata);

        if (code == NULL) {
            code = (unsigned char *)allocMemory(NULL,
                alignedImageSize,
                MEM_RESERVE | MEM_COMMIT,
                PAGE_READWRITE,
                userdata);
            if (code == NULL) {
                SetLastError(ERROR_OUTOFMEMORY);
                        return NULL;
            }
        }
```

```
            result       =       (PUNPACKMODULE)HeapAlloc(GetProcessHeap(),       HEAP_ZERO_MEMORY,
sizeof(UNPACKMODULE));
            if (result == NULL) {
                freeMemory(code, 0, MEM_RELEASE, userdata);
                SetLastError(ERROR_OUTOFMEMORY);
                      return NULL;
            }


               result->codeBase = code;
            result->isDLL = (old_header->FileHeader.Characteristics & IMAGE_FILE_DLL) != 0;
            result->alloc = allocMemory;
            result->free = freeMemory;
            result->loadLibrary = loadLibrary;
            result->getProcAddress = getProcAddress;
            result->freeLibrary = freeLibrary;
            result->userdata = userdata;
            result->pageSize = sysInfo.dwPageSize;

            if (!CheckSize(size, old_header->OptionalHeader.SizeOfHeaders)) {
                goto error;
            }

            headers = (unsigned char *)allocMemory(code,
                old_header->OptionalHeader.SizeOfHeaders,
                MEM_COMMIT,
                PAGE_READWRITE,
                userdata);

            memcpy(headers, dos_header, old_header->OptionalHeader.SizeOfHeaders);
            result->headers = (PIMAGE_NT_HEADERS)&((const unsigned char *)(headers))[dos_header-
>e_lfanew];

            result->headers->OptionalHeader.ImageBase = (uintptr_t)code;

            if (!CopySections((const unsigned char *) data, size, old_header, result)) {
                goto error;
            }

            locationDelta = (ptrdiff_t)(result->headers->OptionalHeader.ImageBase - old_header-
>OptionalHeader.ImageBase);
            if (locationDelta != 0) {
                result->isRelocated = PerformBaseRelocation(result, locationDelta);
```

```
            } else {
                result->isRelocated = TRUE;
            }

            if (!BuildImportTable(result)) {
                goto error;
            }

            if (!FinalizeSections(result)) {
                goto error;
            }

            if (!ExecuteTLS(result)) {
                goto error;
            }

            if (result->headers->OptionalHeader.AddressOfEntryPoint != 0) {
                if (result->isDLL) {

                    DllEntryProc   DllEntry   =   (DllEntryProc)(LPVOID)(code   +   result->headers-
>OptionalHeader.AddressOfEntryPoint);

                    BOOL successfull = (*DllEntry)((HINSTANCE)code, DLL_PROCESS_ATTACH, 0);//ERROR

                    if (!successfull) {
                        SetLastError(ERROR_DLL_INIT_FAILED);
                                    goto error;
                    }
                    result->initialized = TRUE;
                } else {
                    result->exeEntry    =    (ExeEntryProc)(LPVOID)(code    +    result->headers-
>OptionalHeader.AddressOfEntryPoint);
                }
            } else {
                result->exeEntry = NULL;

            }

               return (HUNPACKMODULE)result;

        error:
            MemoryFreeLibrary(result);
            return NULL;
        }
```

```
FARPROC MemoryGetProcAddress(HUNPACKMODULE module, LPCSTR name)
{
    unsigned char *codeBase = ((PUNPACKMODULE)module)->codeBase;
    DWORD idx = 0;
    PIMAGE_EXPORT_DIRECTORY exports;
    PIMAGE_DATA_DIRECTORY    directory    =    GET_HEADER_DICTIONARY((PUNPACKMODULE)module,
IMAGE_DIRECTORY_ENTRY_EXPORT);
    if (directory->Size == 0) {
        SetLastError(ERROR_PROC_NOT_FOUND);
        return NULL;
    }

    exports = (PIMAGE_EXPORT_DIRECTORY) (codeBase + directory->VirtualAddress);
    if (exports->NumberOfNames == 0 || exports->NumberOfFunctions == 0) {
        SetLastError(ERROR_PROC_NOT_FOUND);
        return NULL;
    }

    if (HIWORD(name) == 0) {
        if (LOWORD(name) < exports->Base) {
            SetLastError(ERROR_PROC_NOT_FOUND);
            return NULL;
        }

        idx = LOWORD(name) - exports->Base;
    } else {
        DWORD i;
        DWORD *nameRef = (DWORD *) (codeBase + exports->AddressOfNames);
        WORD *ordinal = (WORD *) (codeBase + exports->AddressOfNameOrdinals);
        BOOL found = FALSE;
        for (i=0; i<exports->NumberOfNames; i++, nameRef++, ordinal++) {
            if (_stricmp(name, (const char *) (codeBase + (*nameRef))) == 0) {
                idx = *ordinal;
                found = TRUE;
                break;
            }
        }

        if (!found) {
            SetLastError(ERROR_PROC_NOT_FOUND);
            return NULL;
        }
    }
```

```
            if (idx > exports->NumberOfFunctions) {
                SetLastError(ERROR_PROC_NOT_FOUND);
                return NULL;
            }

            return    (FARPROC)(LPVOID)(codeBase    +    (*(DWORD    *)    (codeBase    +    exports-
>AddressOfFunctions + (idx*4))));
        }

        void MemoryFreeLibrary(HUNPACKMODULE mod)
        {
            PUNPACKMODULE module = (PUNPACKMODULE)mod;

            if (module == NULL) {
                return;
            }
            if (module->initialized) {
                DllEntryProc DllEntry = (DllEntryProc)(LPVOID)(module->codeBase + module->headers-
>OptionalHeader.AddressOfEntryPoint);
                (*DllEntry)((HINSTANCE)module->codeBase, DLL_PROCESS_DETACH, 0);
            }

            if (module->modules != NULL) {
                int i;
                for (i=0; i<module->numModules; i++) {
                    if (module->modules[i] != NULL) {
                        module->freeLibrary(module->modules[i], module->userdata);
                    }
                }

                free(module->modules);
            }

            if (module->codeBase != NULL) {
                module->free(module->codeBase, 0, MEM_RELEASE, module->userdata);
            }

            HeapFree(GetProcessHeap(), 0, module);
        }

        int MemoryCallEntryPoint(HUNPACKMODULE mod)
        {
            PUNPACKMODULE module = (PUNPACKMODULE)mod;
```

```
            if (module == NULL || module->isDLL || module->exeEntry == NULL || !module->isRelocated) {
                return -1;
            }

            return module->exeEntry();
        }

        #define DEFAULT_LANGUAGE        MAKELANGID(LANG_NEUTRAL, SUBLANG_NEUTRAL)

        HMEMORYRSRC MemoryFindResource(HUNPACKMODULE module, LPCTSTR name, LPCTSTR type)
        {
            return MemoryFindResourceEx(module, name, type, DEFAULT_LANGUAGE);
        }

        static PIMAGE_RESOURCE_DIRECTORY_ENTRY _MemorySearchResourceEntry(
            void *root,
            PIMAGE_RESOURCE_DIRECTORY resources,
            LPCTSTR key)
        {
            PIMAGE_RESOURCE_DIRECTORY_ENTRY entries = (PIMAGE_RESOURCE_DIRECTORY_ENTRY) (resources + 1);
            PIMAGE_RESOURCE_DIRECTORY_ENTRY result = NULL;
            DWORD start;
            DWORD end;
            DWORD middle;

            if (!IS_INTRESOURCE(key) && key[0] == TEXT('#')) {
                TCHAR *endpos = NULL;
                long int tmpkey = (WORD) _tcstol((TCHAR *) &key[1], &endpos, 10);
                if (tmpkey <= 0xffff && lstrlen(endpos) == 0) {
                    key = MAKEINTRESOURCE(tmpkey);
                }
            }

            if (IS_INTRESOURCE(key)) {
                WORD check = (WORD) (uintptr_t) key;
                start = resources->NumberOfNamedEntries;
                end = start + resources->NumberOfIdEntries;

                while (end > start) {
                    WORD entryName;
                    middle = (start + end) >> 1;
```

```
                entryName = (WORD) entries[middle].Name;
                if (check < entryName) {
                    end = (end != middle ? middle : middle-1);
                } else if (check > entryName) {
                    start = (start != middle ? middle : middle+1);
                } else {
                    result = &entries[middle];
                    break;
                }
            }
        } else {
            LPCWSTR searchKey;
            size_t searchKeyLen = _tcslen(key);
    #if defined(UNICODE)
            searchKey = key;
    #else
    #define MAX_LOCAL_KEY_LENGTH 2048
            wchar_t _searchKeySpace[MAX_LOCAL_KEY_LENGTH+1];
            LPWSTR _searchKey;
            if (searchKeyLen > MAX_LOCAL_KEY_LENGTH) {
                size_t _searchKeySize = (searchKeyLen + 1) * sizeof(wchar_t);
                _searchKey = (LPWSTR) malloc(_searchKeySize);
                if (_searchKey == NULL) {
                    SetLastError(ERROR_OUTOFMEMORY);
                    return NULL;
                }
            } else {
                _searchKey = &_searchKeySpace[0];
            }

            mbstowcs(_searchKey, key, searchKeyLen);
            _searchKey[searchKeyLen] = 0;
            searchKey = _searchKey;
    #endif
            start = 0;
            end = resources->NumberOfNamedEntries;
            while (end > start) {
                int cmp;
                PIMAGE_RESOURCE_DIR_STRING_U resourceString;
                middle = (start + end) >> 1;
                resourceString     =     (PIMAGE_RESOURCE_DIR_STRING_U)     OffsetPointer(root,
entries[middle].Name & 0x7FFFFFFF);
                cmp    =    _wcsnicmp(searchKey,    resourceString->NameString,    resourceString-
>Length);
```

```
                if (cmp == 0) {
                    if (searchKeyLen > resourceString->Length) {
                        cmp = 1;
                    } else if (searchKeyLen < resourceString->Length) {
                        cmp = -1;
                    }
                }
                if (cmp < 0) {
                    end = (middle != end ? middle : middle-1);
                } else if (cmp > 0) {
                    start = (middle != start ? middle : middle+1);
                } else {
                    result = &entries[middle];
                    break;
                }
            }
        #if !defined(UNICODE)
            if (searchKeyLen > MAX_LOCAL_KEY_LENGTH) {
                free(_searchKey);
            }
        #undef MAX_LOCAL_KEY_LENGTH
        #endif
        }

        return result;
    }

    HMEMORYRSRC MemoryFindResourceEx(HUNPACKMODULE module, LPCTSTR name, LPCTSTR type, WORD
language)
    {
        unsigned char *codeBase = ((PUNPACKMODULE) module)->codeBase;
        PIMAGE_DATA_DIRECTORY directory = GET_HEADER_DICTIONARY((PUNPACKMODULE) module,
IMAGE_DIRECTORY_ENTRY_RESOURCE);
        PIMAGE_RESOURCE_DIRECTORY rootResources;
        PIMAGE_RESOURCE_DIRECTORY nameResources;
        PIMAGE_RESOURCE_DIRECTORY typeResources;
        PIMAGE_RESOURCE_DIRECTORY_ENTRY foundType;
        PIMAGE_RESOURCE_DIRECTORY_ENTRY foundName;
        PIMAGE_RESOURCE_DIRECTORY_ENTRY foundLanguage;
        if (directory->Size == 0) {
            SetLastError(ERROR_RESOURCE_DATA_NOT_FOUND);
            return NULL;
        }
```

```
            if (language == DEFAULT_LANGUAGE) {
                language = LANGIDFROMLCID(GetThreadLocale());
            }

            rootResources = (PIMAGE_RESOURCE_DIRECTORY) (codeBase + directory->VirtualAddress);
            foundType = _MemorySearchResourceEntry(rootResources, rootResources, type);
            if (foundType == NULL) {
                SetLastError(ERROR_RESOURCE_TYPE_NOT_FOUND);
                return NULL;
            }

            typeResources = (PIMAGE_RESOURCE_DIRECTORY) (codeBase + directory->VirtualAddress +
(foundType->OffsetToData & 0x7fffffff));
            foundName = _MemorySearchResourceEntry(rootResources, typeResources, name);
            if (foundName == NULL) {
                SetLastError(ERROR_RESOURCE_NAME_NOT_FOUND);
                return NULL;
            }

            nameResources = (PIMAGE_RESOURCE_DIRECTORY) (codeBase + directory->VirtualAddress +
(foundName->OffsetToData & 0x7fffffff));
            foundLanguage = _MemorySearchResourceEntry(rootResources, nameResources, (LPCTSTR)
(uintptr_t) language);
            if (foundLanguage == NULL) {
                if (nameResources->NumberOfIdEntries == 0) {
                    SetLastError(ERROR_RESOURCE_LANG_NOT_FOUND);
                    return NULL;
                }

                foundLanguage = (PIMAGE_RESOURCE_DIRECTORY_ENTRY) (nameResources + 1);
            }

            return (codeBase + directory->VirtualAddress + (foundLanguage->OffsetToData &
0x7fffffff));
        }

        DWORD MemorySizeofResource(HUNPACKMODULE module, HMEMORYRSRC resource)
        {
            PIMAGE_RESOURCE_DATA_ENTRY entry;
            UNREFERENCED_PARAMETER(module);
            entry = (PIMAGE_RESOURCE_DATA_ENTRY) resource;
            if (entry == NULL) {
                return 0;
            }
```

```
            return entry->Size;
        }

        LPVOID MemoryLoadResource(HUNPACKMODULE module, HMEMORYRSRC resource)
        {
            unsigned char *codeBase = ((PUNPACKMODULE) module)->codeBase;
            PIMAGE_RESOURCE_DATA_ENTRY entry = (PIMAGE_RESOURCE_DATA_ENTRY) resource;
            if (entry == NULL) {
                return NULL;
            }

            return codeBase + entry->OffsetToData;
        }

        int
        MemoryLoadString(HUNPACKMODULE module, UINT id, LPTSTR buffer, int maxsize)
        {
            return MemoryLoadStringEx(module, id, buffer, maxsize, DEFAULT_LANGUAGE);
        }

        int
        MemoryLoadStringEx(HUNPACKMODULE module, UINT id, LPTSTR buffer, int maxsize, WORD
language)
        {
            HMEMORYRSRC resource;
            PIMAGE_RESOURCE_DIR_STRING_U data;
            DWORD size;
            if (maxsize == 0) {
                return 0;
            }

            resource = MemoryFindResourceEx(module, MAKEINTRESOURCE((id >> 4) + 1), RT_STRING,
language);
            if (resource == NULL) {
                buffer[0] = 0;
                return 0;
            }

            data = (PIMAGE_RESOURCE_DIR_STRING_U) MemoryLoadResource(module, resource);
            id = id & 0x0f;
            while (id--) {
                data = (PIMAGE_RESOURCE_DIR_STRING_U) OffsetPointer(data, (data->Length + 1) *
sizeof(WCHAR));
```

```
    }
    if (data->Length == 0) {
        SetLastError(ERROR_RESOURCE_NAME_NOT_FOUND);
        buffer[0] = 0;
        return 0;
    }

    size = data->Length;
    if (size >= (DWORD) maxsize) {
        size = maxsize;
    } else {
        buffer[size] = 0;
    }
#if defined(UNICODE)
    wcsncpy(buffer, data->NameString, size);
#else
    wcstombs(buffer, data->NameString, size);
#endif
    return size;
}

#ifdef TESTSUITE
#include <stdio.h>

#ifndef PRIxPTR
#ifdef _WIN64
#define PRIxPTR "I64x"
#else
#define PRIxPTR "x"
#endif
#endif

static const uintptr_t AlignValueDownTests[][3] = {
    {16, 16, 16},
    {17, 16, 16},
    {32, 16, 32},
    {33, 16, 32},
#ifdef _WIN64
    {0x12345678abcd1000, 0x1000, 0x12345678abcd1000},
    {0x12345678abcd101f, 0x1000, 0x12345678abcd1000},
#endif
    {0, 0, 0},
};
```

```
        static const uintptr_t AlignValueUpTests[][3] = {
            {16, 16, 16},
            {17, 16, 32},
            {32, 16, 32},
            {33, 16, 48},
        #ifdef _WIN64
            {0x12345678abcd1000, 0x1000, 0x12345678abcd1000},
            {0x12345678abcd101f, 0x1000, 0x12345678abcd2000},
        #endif
            {0, 0, 0},
        };

        BOOL UnpackModuleTestsuite() {
            BOOL success = TRUE;
            size_t idx;
            for (idx = 0; AlignValueDownTests[idx][0]; ++idx) {
                const uintptr_t* tests = AlignValueDownTests[idx];
                uintptr_t value = AlignValueDown(tests[0], tests[1]);
                if (value != tests[2]) {
                    printf("AlignValueDown failed for 0x%" PRIxPTR "/0x%" PRIxPTR ": expected 0x%"
PRIxPTR ", got 0x%" PRIxPTR "\n",
                        tests[0], tests[1], tests[2], value);
                    success = FALSE;
                }
            }
            for (idx = 0; AlignValueDownTests[idx][0]; ++idx) {
                const uintptr_t* tests = AlignValueUpTests[idx];
                uintptr_t value = AlignValueUp(tests[0], tests[1]);
                if (value != tests[2]) {
                    printf("AlignValueUp failed for 0x%" PRIxPTR "/0x%" PRIxPTR ": expected 0x%"
PRIxPTR ", got 0x%" PRIxPTR "\n",
                        tests[0], tests[1], tests[2], value);
                    success = FALSE;
                }
            }
            if (success) {
                printf("OK\n");
            }
            return success;
        }
        #endif
```

ДОДАТОК Е Ілюстративний матеріал до захисту

ІЛЮСТРАТИВНИЙ МАТЕРІАЛ ДО ЗАХИСТУ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Завідувач кафедри ПЗ, д. т. н., професор _____ О. Н. Романюк

Науковий керівник, к. т. н., доцент кафедри ПЗ_____ В. В. Войтко

Рецензент, д.т.н, професор, зав. кафедри КН_____ А.А. Яровий

Нормоконтроль, к. т. н., доцент кафедри ПЗ _____В. В. Войтко

Виконавець, студент групи 2ПІ-18м _____ А.О. Андреєв

Розробка методу і засобів системи захисту даних з використанням технології динамічного завантаження коду в робочий процес

Автор:
ст. гр. 2ПІ-18м Андреєв А.О.

Науковий керівник:
к.т.н., доцент Войтко В.В.

Слайд 1 – Тема, автор, науковий керівник

Актуальність

Захист програмного продукту від несанкціонованого копіювання є актуальною задачею у зв'язку зі збереженням комерційних і авторських прав фірм і розробників:

- За висновками закордонних фахівців економічний збиток від "піратського" копіювання програмного забезпечення складає 53 млрд доларів станом на 2018 рік.
- На даний момент 41% програмного забезпечення є «піратським».
- Несанкціоноване розповсюдження ПЗ спричиняє банкротство та збитковість компаній-розробників, що, в свою чергу, загальмовує розвиток ІТ.

Слайд 2 – Актуальність

Мета, об’єкт та предмет дослідження

- **Мета** – підвищення захищеності програмних додатків від несанкціонованого копіювання шляхом розробки динамічно завантажуваної бібліотеки та модуля для управління процесом користувацького додатку, що підвищить захист програм від несанкціонованого доступу.
- **Об’єкт** – процес захисту від несанкціонованого доступу.
- **Предмет** – засоби захисту програмних додатків від несанкціонованого копіювання.

Слайд 3 – Мета, об’єкт та предмет

Задачі

- удосконалити метод динамічного завантаження програмного продукту або його частин та розробити модель і алгоритми системи захисту програмних додатків;
- розробити динамічно завантажувану бібліотеку, що буде завантажуватися у виконуваний процес та структуровано копіювати вхідну користувацьку бібліотеку;
- розробити модуль для завантаження користувацької бібліотеки з сервера;
- розробити модуль для управління процесом для користувацького додатку;
- провести тестування програмного продукту.

Слайд 4 – Задачі

Наукова новизна та практичне значення

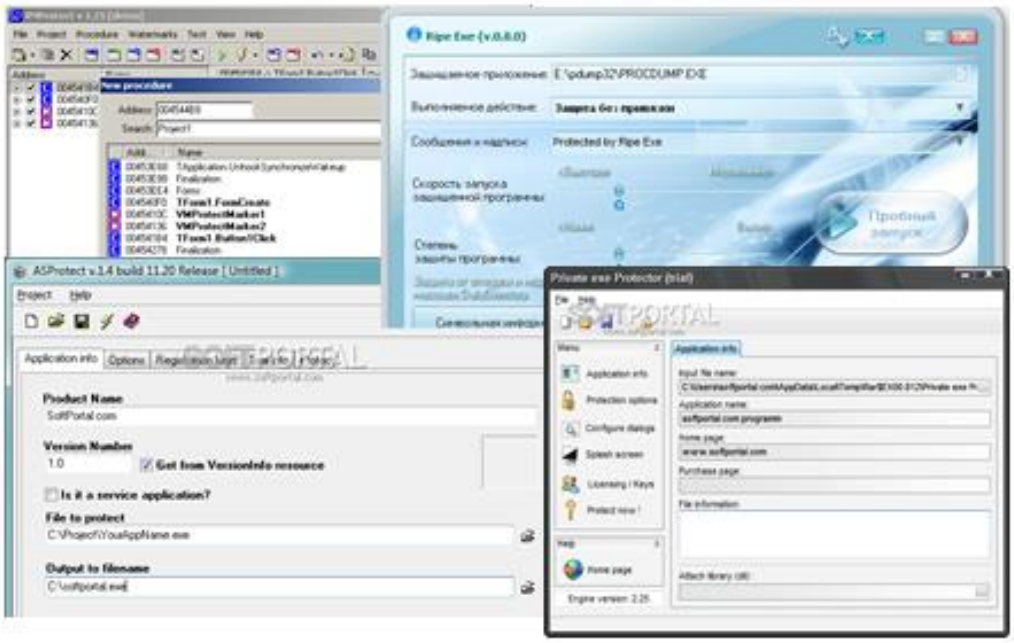
Наукова новизна одержаних результатів:

- Подальшого розвитку дістав метод динамічного завантаження програмного продукту або його частин, який, на відміну від існуючих, передбачає збереження виконуваного коду на сервері, а не на локальній машині користувача, та динамічне завантаження бібліотек у робочий процес, що забезпечує високий рівень захисту виконуваних файлів від несанкціонованого копіювання.
- Подальшого розвитку дістала модель системи захисту програмних продуктів від несанкціонованого доступу, яка, на відміну від існуючих, орієнтована на кодову ін'єкцію завантажуваних бібліотек у виконуваний процес, що дозволяє динамічно забезпечувати захист програм в робочому режимі.

Практична цінність отриманих результатів полягає у забезпеченні захищеності програмних рішень від несанкціонованого доступу до виконуваних файлів, а також їх копіювання.

Слайд 5 – Наукова новизна та практичне значення

Існуючі аналоги



Слайд 6 – Існуючі аналоги

Порівняння з аналогами

Загальне порівняння аналогів та додатку «DllLoadManager» наведено у таблиці

Критерій	VMProtect	Private exe Protector	ASProtect	Ripe Exe	DllLoadManager
Відсутність використання додаткових апаратних ресурсів	+	+	+	+	+
Відсутність необхідності використання віртуальної машини	+	-	+	+	+
Доступ до коду захищеної програми в будь якому вигляді	-	-	-	-	+
Можливість розширення	-	-	-	-	+
Небезпека при взломі програми-захисника	-	-	-	-	+
Відсутність необхідності додаткової обробки програмного продукту	-	-	-	-	+

Слайд 7 – Порівняння з аналогами

Засоби розробки

Характеристики	Мова програмування		
	C++	C#	Java
Об'єктно-орієнтована	+	+	+
Створення багатовимірних масивів	+	+	-
Узагальнене програмування	+	+	+
Створення анонімних функцій	+	-	-
Створення динамічних масивів	+	+	+
Розробка програмного інтерфейсу	+	+	+

Функції	Середовище програмування		
	C++ Builder	Dev-C++	Microsoft Visual Studio
Підтримка MFC	-	-	+
Режим відлагодження	+/-	+/-	+
Кросплатформеність	-	-	+
Функція авто заповнення	+/-	-	+

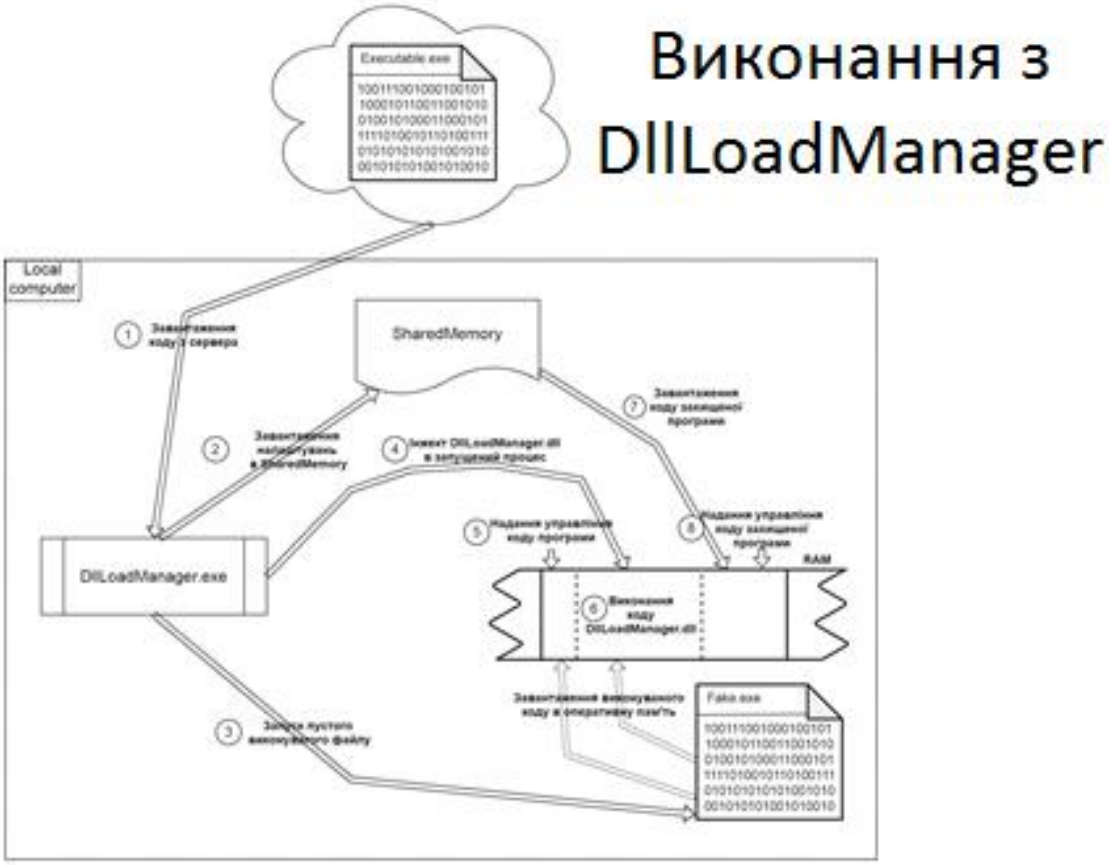
Слайд 8 – Засоби розробки

Слайд 9 – Використані технології

Слайд 10 – Модульна схема DllLoadManager

Слайд 11 – Модель взаємодії процесів

Слайд 12 – Схема звичайного виконання програми



Слайд 13 – Схема виконання програми з DllLoadManager

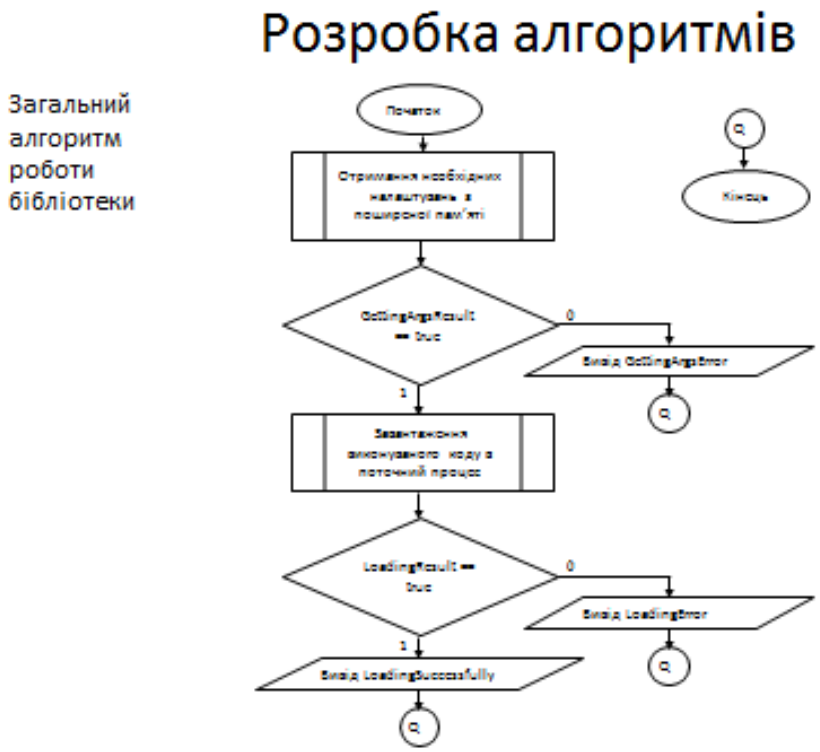
Рекомендації до використання

- Розміщення файлу виконуваного коду свого додатку на файловому сервері (FTP), заборонивши вільний доступ до нього та реалізувавши надання доступу за паролем.
- Запуск серверу реєстрації, що буде отримувати GET запити на отримання пароля для доступу до FTP сервера, перевіряючи дані користувача, що прийшли в запиті.
- Формування конфігураційного файлу, в якому потрібно прописати адреси серверів.
- Доставка користувачу-замовнику програмного забезпечення конфігураційного файлу та додатку DllLoadManager, а також порожнього виконуваного файлу, в який пізніше буде здійснено завантаження файлу.
- Реєстрація користувача-замовника ПЗ на сервері реєстрації.

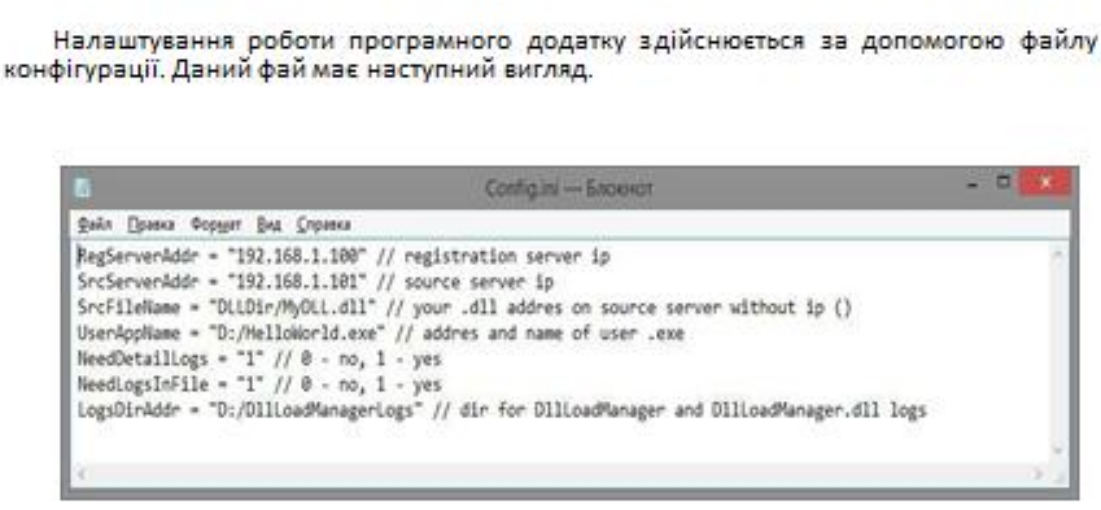
Слайд 14 – Рекомендації до використання

Слайд 15 – Алгоритм роботи програмного додатку DllLoadManager

Слайд 16 – Продовження алгоритму роботи програмного додатку DllLoadManager

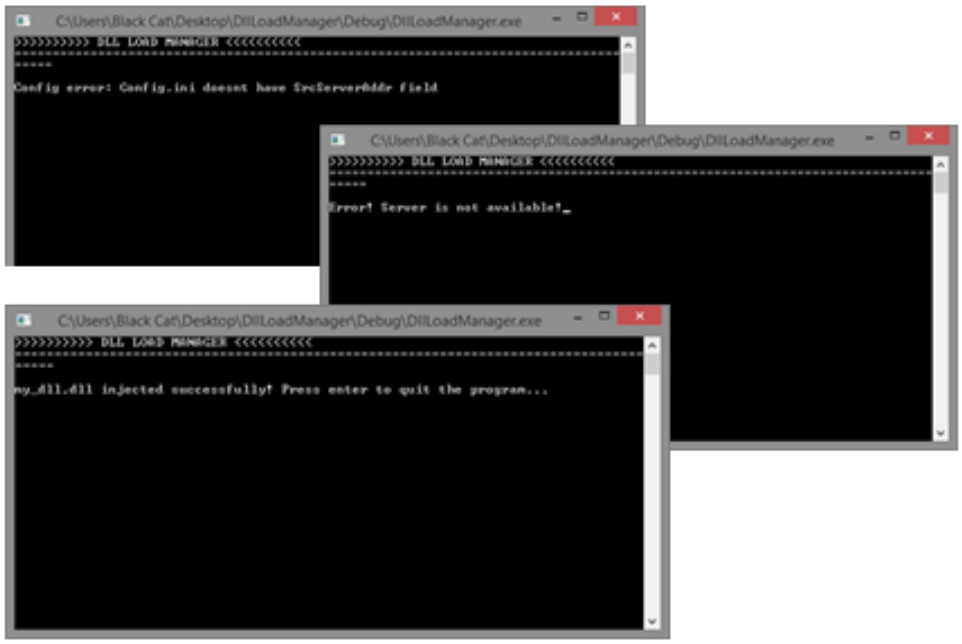


Слайд 17 – Алгоритм роботи бібліотеки DllLoadManager



Слайд 18 – Файл конфігурації DllLoadManager

Тестування сайту



Слайд 19 – Скріншоти тестових запусків додатку

Висновки

- удосконалено метод динамічного завантаження програмного продукту або його частин, який передбачає збереження виконуваного коду на сервері, а не на локальній машині користувача, та динамічне завантаження бібліотек у робочий процес, що забезпечує високий рівень захисту виконуваних файлів від несанкціонованого копіювання;
- розроблено динамічно завантажувану бібліотеку, що буде завантажуватись в виконуваний процес та структуровано копіювати вхідну користувацьку бібліотеку;
- розроблено модуль для завантаження користувацької бібліотеки з сервера;
- розроблено модуль для управління процесом для користувацького додатку;
- проведено тестування програмного продукту.

Слайд 20 – Висновки по завершенню виконання дипломної роботи

Апробація результатів роботи

- на міжвузівському студентському вебінарі «Інноваційні та інформаційні технології в бізнесі та освіті». – Вінниця: ВТЕІ КНТЕУ. – 21 жовтня 2015 р.
- на науково-практичній Інтернет-конференції «Електронні інформаційні ресурси: створення, використання, доступ – 2015». – Вінниця: ВНТУ, 2015 р.;
- на міжнародній науково-практичній Інтернет-конференції «Електронні інформаційні ресурси: створення, використання, доступ – 2018». – Вінниця: ВНТУ, січень 2018 р.

Результати роботи впроваджено

- на ТОВ «Он-лайн».

Слайд 21 – Апробація та впровадження

Публікації

- Войтко В.В. Система захисту програмних додатків з використанням динамічного завантаження виконуваного коду в робочий процес / В.В. Войтко, С.В. Бевз, С.М. Бурбело, А.О. Андреєв // Електронні інформаційні ресурси: створення, використання, доступ // Збірник наукових праць міжнародної науково-практичної Інтернет-конференції. – Вінниця: ВНТУ, січень-2018. – С.248-253.
- Войтко В.В. Розробка та провадження на ринок програмного додатку з вивчення хімії / В.В. Войтко, А.О. Андреєв, О.В. Дажура, С.В. Козловський, В.В.Туйчев // Інноваційні та інформаційні технології в бізнесі та освіті // Матеріали міжвузівського студентського вебінару / відп. ред. Л.Б. Ліщинська. – Вінниця: ВТЕІ КНТЕУ. – 21 жовтня 2015р. – С. 6-7.
- Войтко В.В. Автоматизація процесів формування новин у різних соціальних мережах / В.В. Войтко, С.В. Бевз, А.О. Андреєв, О.В. Дажура, В.В.Туйчев, О.В. Дикий // Електронні інформаційні ресурси: створення, використання, доступ // Збірник наукових праць міжнародної науково-практичної Інтернет-конференції. – Вінниця: ВНТУ, 2015. – С. 91-92
- Войтко В.В. Комп'ютерна програма «Програмний продукт для автоматизації роботи з різними соціальними мережами» / В.В. Войтко, А.О.Андреєв, О.В. Дикий, О.В. Дажура, В.В. Туйчев // Свідоцтво про реєстрацію авторського права на твір № 64732 від 01.04.2016.

Слайд 22 – Публікації

Нагороди

Результати роботи були подані на Всеукраїнський та Міжнародний студентські конкурси, де отримано призові місця:

- диплом переможця II ступеня у Всеукраїнському конкурсі студентських наукових робіт з напрямку «Інформатика і кібернетика»
- диплом за зайняте II місце у Фіналі Міжнародної студентської олімпіади з інформаційних технологій IT-UNIVERSE в конкурсі «Кращий диплом з кібербезпеки».

Слайд 23 – Нагороди