Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації

Магістерська кваліфікаційна робота на тему:
«Метод та програмна бібліотека захисту програм від дампінгу»

Виконав: студент 2 курсу групи 2БС-22м
спеціальності 125 Кібербезпека
_____ Олександр ПАЛАМАРЧУК

Керівник: к. т. н., доцент каф. ЗІ
_____ Юрій БАРИШЕВ
«11» 12 2023 р.

Опонент: к. т. н., доцент каф. ПЗ
_____ Денис КАТЄЛЬНІКОВ
«12» 12 2023 р.

Допущено до захисту
Завідувач кафедри ЗІ
д. т. н., проф.
_____ Володимир ЛУЖЕЦЬКИЙ
«13» 12 2023 р.

Вінниця ВНТУ – 2023 року

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації
Рівень вищої освіти II (магістерський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 125 Кібербезпека
Освітньо-професійна програма – Безпека інформаційних і комунікаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри ЗІ,
д. т. н., проф.
_____Володимир ЛУЖЕЦЬКИЙ
«19» 09 2023 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Паламарчуку Олександру Руслановичу

1. Тема роботи: «Метод та програмна бібліотека захисту програм від дампінгу» керівник роботи: Баришев Юрій Володимирович, к. т. н., доцент кафедри ЗІ, затверджені наказом ректора ВНТУ від 18 вересня 2023 №247.
2. Строк подання студентом роботи: 13 грудня 2023р.
3. Вихідні дані до роботи:
   - метод захисту – шифрування;
   - мінімальна довжина ключа – 128 біт;
   - максимальний обсяг пам'яті – 4 ГБ.
4. Зміст текстової частини: Вступ. 1. Аналіз методів захисту від дампінгу. 2. Метод захисту від дампінгу. 3. Засіб захисту від дампінгу. 4. Тестування засобу захисту від дампінгу. 5. Економічна частина. Висновки. Перелік використаних джерел. Додатки.
5. Перелік ілюстративного матеріалу: Наукова новизна (плакат А4). Предмет, об'єкт, мета та задачі дослідження (плакат А4). Метод блокування переривань копіювання пам'яті (плакат А4). Математичний опис процесу захисту від дампінгу (плакат А4). Метод захисту від дампінгу (плакат А4). Метод шифрування (плакат А4). Архітектура засобу (плакат А4). Результати блокового тестування (плакат А4). Результати статичного тестування (плакат А4). Економічні показники розробки (плакат А4).

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Юрій БАРИШЕВ, к. т. н., доц. каф. ЗІ	01.09	23.09
2	Юрій БАРИШЕВ, к. т. н., доц. каф. ЗІ	01.09	10.10
3	Юрій БАРИШЕВ, к. т. н., доц. каф. ЗІ	01.09	15.11
4	Юрій БАРИШЕВ, к. т. н., доц. каф. ЗІ	01.09	21.11
5	Ольга РАТУШНЯК, к. т. н., доц. каф. ЕПВМ	01.09	17.11

7. Дата видачі завдання 1 вересня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз завдання. Вступ	01.09.2023 – 10.09.2023	
2	Аналіз літературних джерел за напрямком магістерської кваліфікаційної роботи	10.09.2023 – 15.09.2023	
3	Науково-технічне обгрунтування	16.09.2023 – 22.09.2023	
4	Розробка технічного завдання	23.09.2023 – 29.09.2023	
5	Розробка рішень	30.09.2023 – 12.10.2023	
6	Практична реалізація, моделювання, експериментування, результати	14.10.2023 – 10.11.2023	
7	Розробка розділу економічного обгрунтування доцільного розробки	11.11.2023 – 17.11.2023	
8	Аналіз виконання ТЗ, висновки	18.11.2023 – 24.11.2023	
9	Оформлення пояснювальної записки	25.11.2023 – 30.11.2023	
10	Попередній захист та доопрацювання МКР	28.11.2023 – 10.12.2023	
11	Перевірка магістерської роботи на наявність плагіату	02.12.2023 – 10.12.2023	
12	Представлення МКР до захисту	11.12.2023 – 14.12.2023	
13	Захист МКР	14.12.2023 – 21.12.2023	

Студент _____ Олександр ПАЛАМАРЧ

Керівник роботи _____ Юрій БАРИШ

АНОТАЦІЯ

Магістерська кваліфікаційна робота складається з 104 сторінки формату А4, на яких є 26 рисунків, 8 таблиць, 26 формул, список використаних джерел містить 30 найменувань.

Магістерська кваліфікаційна робота присвячена розробці методу та програмній бібліотеці захисту програм від дампінгу. У роботі проаналізовано способи атак на основі дампінгу, а також основні методи захисту від них. У межах роботи виконано теоретико-множинний опис методу дампінгу, описано метод захисту від дампінгу, метод шифрування та метод блокування переривань копіювання пам’яті. Спроектовано архітектуру засобу, а також описано основні алгоритми роботи. Після розробки архітектури кожного модулю та алгоритму кожної процедури здійснено програмну реалізацію. Проведено блокове та інтеграційне тестування, статичне тестування безпеки та тестування продуктивності.

Ключові слова: дамп пам’яті, шифрування, захист застосунків, тестування безпеки, математичний опис.

ABSTRACT

The master's qualification work consists of 104 pages of A4 format, which contains 26 figures, 8 tables, 26 formulas, the list of references contains 30 items.

The master's qualification work is devoted to the development of method and program library for protecting from memory dumping. The work analyses methods of cyber attacks that using memory dumping and how to protect from them. The paper provides a set-theoretic description of the memory protection method from dumping, describes the encryption method and method of interrupting execution of memory copying functions. The architecture of the application is designed, and the basic operation algorithms are described. After developing the schemes of functioning of the software modules and algorithms of all procedures, the software implementation was carried out. Automated unit and integration testing, static security testing, and productivity testing of the application were performed.

Keywords: memory dumping, encryption, application protection, security testing, set-theoretic description.

ЗМІСТ

ВСТУП

З підвищенням рівня глобалізації, кількості пристроїв підключених до мережі Інтернет, росте кількість ризиків, що зв’язані з використанням інформаційних технологій. Хоча з початком повномасштабного вторгнення в Україну збільшилась саме кількість DDOS атак [1], що спрямовані на порушення доступності, проте вони можуть бути лише складовою більш комплексної атаки, яка може бути націлена на викрадення даних користувачів тієї чи іншої інформаційної системи.

Дана магістерська дипломна робота є актуальною, оскільки випадки з порушенням конфіденційності користувачів, де зловмисники тим чи іншим чином використовують техніки дампінгу регулярно відбуваються у світі. Наприклад, один з підрозділів Broadcom Software, який відстежує загрози від програм вимагачів станом на 2022 рік фіксує [2], що всі основні сімейства програм вимагачів використовують бібліотеки rundll32.dll та comsvcs.dll для дампу пам’яті LSASS. Також можуть створюватися дампи файлів SAM, SECURITY та SYSTEM для того щоб витягнути з них облікові дані локальної машини [2]. Також, 4-го січня 2023 в компанії CircleCI, що займається CI/CD, стався витік даних користувачів і хоча всі дані були зашифровані, але зловмисники витягли ключі шифрування з запущеного процесу, що дозволило їм отримати доступ до даних [3].

Об’єктом даної роботи є процес розробки програмного забезпечення.

Предметом — методи та засоби захисту програм від дампінгу оперативної пам’яті.

Метою магістерської кваліфікаційної роботи є покращення захисту програм від дампінгу.

Для досягнення мети необхідно розв’язати такі задачі:

- проаналізувати існуючі методи для захисту програм від дампінгу;
- проаналізувати відомі засоби захисту програм від дампінгу;
- розробити власний комплексний метод захисту;
- розробити власний комплексний засіб захисту.

- протестувати власний комплексний засіб захисту;
- обчислити економічну доцільність.

Науковою новизною є те, що отримано подальший розвиток методів захисту від дампінгу, що відрізняється від відомих модульністю та можливістю створення зашифрованих контейнерів, що дозволяє використовувати метод при розробці інших застосунків.

Практична цінність роботи полягає у розробленій програмній бібліотеці, яка може бути використана під час розробки інших програм.

Результати отримані у магістерській кваліфікаційній роботі були представлені на таких конференціях:

- LI Науково-технічна конференція ІТКІ 2022 року, Вінницького національного технічного університету [4];
- LIII Науково-технічна конференція ІТКІ 2023 року, Вінницького національного технічного університету [5].

1 АНАЛІЗ МЕТОДІВ ЗАХИСТУ ВІД ДАМПІНГУ

1.1 Аналіз загроз застосункам

Програмний застосунок – комп'ютерна програма створена для обробки даних користувача та вирішення певної задачі для якої вона була створена [6]. Програмні застосунки можна умовно розділити на застосунки загального та спеціального призначення [7]. До застосунків загального призначення можна віднести програми, які можуть бути використані в багатьох видах людської діяльності, наприклад, текстові та табличні процесори, програми для створення презентацій та керування базами даних тощо. До програмних застосунків спеціального призначення відносяться програми, які можуть бути використані тільки для виконання задач, що існують в межах певного вузькоспеціалізованого роду занять. До них можна віднести програмне забезпечення для керування військовими об'єктами, медичними пристроями, проектування машин та механізмів тощо [7].

Розробка програмного застосунку складається з декількох етапів [8]. Першим кроком є аналіз вимог, що висуваються до програмного застосунку. Метою аналізу є визначення всіх особливостей розроблюваного застосунку і досягнення компромісу між замовником та розробником. Наступним кроком розробки програмного забезпечення є проектування під час якого розробляється загальний дизайн програми, визначаються технології, які будуть використані під час розробки та перелік програмних компонентів, які потрібно реалізувати. Третім кроком є безпосередня розробка та програмування, яка ділиться на розробку алгоритмів, написання коду, його тестування та зневадження. Останнім кроком розробки є створення документації, введення в експлуатацію та супровід програмного застосунку до виведення його з експлуатації [8].

Програмні застосунки можуть бути атаковані різними способами, в залежності від цілей зловмисника, дизайну архітектури програмного засобу, використаних технологій, в тому числі технологій безпеки, тощо.

Основними загрозами безпеки для програмних застосунків є такі [9-11]:

- вторгнення в роботу програми для модифікації її поведінки;

- несанкціоноване копіювання програмного застосунку;
- несанкціоноване дослідження алгоритмів захисту програми з метою їх обходу;
- використання хуків при роботі з програмою;
- дампінг пам’яті програми;
- спроба зламу облікових даних іншого користувача;
- викрадення та подальша модифікація файлів програми з метою обходу авторських прав на продукт;
- недоліки контролю доступу;
- криптографічні збої в алгоритмах захисту;
- ін̣’єкції баз даних, що вбудовані в програмний застосунок;
- небезпечний архітектурний дизайн;
- використання застарілих та вразливих бібліотек при розробці програмного забезпечення.

З перечислених вище загроз, найбільш значущими є недоліки контролю доступу, використання застарілих та вразливих бібліотек при розробці програмного забезпечення та створення його дампу пам’яті. Дампінг пам’яті може бути використаний як один із засобів дослідження цілі, що використовуються зловмисником для визначення вразливостей в програмному забезпеченні та пошуку точки входу для реалізації атаки, тому захист від несанкціонованого створення дампу пам’яті є одним з найважливіших методів захисту від несанкціонованого дослідження програмного забезпечення.

1.2 Аналіз атак на основі дампінгу

Одним з найефективніших методів боротьби з навісним захистом є створення дампів пам’яті, що являють собою копію оперативної пам’яті всієї операційної системи, її частини, або тільки одного процесу [12].

Дампи пам’яті першочергово призначені для збору інформації для діагностики збоїв у роботі програм та можуть допомогти у визначенні причини

збою цілої операційної системи так і окремої програми [10]. Зняття дампу програми може здійснюватися як за допомогою сторонніх програм [13] так і за допомогою засобів операційної системи [14].

Один з підрозділів Broadcom Software, який відстежує загрози від програм вимагачів фіксує, що останнім часом всі основні сімейства програм цього типу використовують бібліотеки rundll32.dll та comsvcs.dll для дампу пам’яті LSASS. Також можуть створюватися дампи файлів SAM, SECURITY та SYSTEM для того щоб витягнути з них облікові дані локальної машини [2]. Також деколи шкідливе програмне забезпечення використовувало системний процес taskmgr.exe для зняття дампу пам’яті LSASS [2].

Також однією з найпоширеніших програм для зняття дампу пам’яті з метою отримання паролів користувачів є Mimikatz. З його допомогою можна отримати паролі користувачів як і з пам’яті Windows напряму, або через віддалений доступ так і з файлів глибокого сну системи та файлів віртуальних машин [15].

Якщо зловмисник отримав фізичний доступ до розблокованої машини він може одразу зняти дамп цікавого йому процесу за допомогою диспетчера задач, або офіційної утиліти від Microsoft за допомогою всього однієї команди [16]. На заблоковану машину можна провести атаку Cold Boot за допомогою звичайного USB накопичувача, здійснивши апаратне перезавантаження комп’ютера вручну, або викликавши BSOD (синій екран смерті). Після чого отриманий дамп оперативної пам’яті можна проаналізувати на іншому комп’ютері [17].

Зловмисник також може здійснити дамп пам’яті використовуючи вразливості в апаратному забезпеченні. Одна з вразливостей серії чипів Intel з кодом CVE-2022-21233 була пов’язана з витоком конфіденційних даних з ізольованих блоків Intel Software Guard eXtensions та при наявності у зловмисника прав адміністратора, дозволяла зняти повний дамп пам’яті програми, що використовувала у своїй роботі дану технологію [18, 19].

Як показав аналіз, дампінг пам’яті достатньо поширена атака на системи під керуванням різних операційних систем, може бути здійснена як програмними так і

апаратними засобами з використанням вбудованих, або сторонніх утиліт. Метою даної атаки зазвичай є отримання даних для автентифікації інших користувачів.

1.3 Аналіз методів захисту від дампінгу

Більшість дамперів написаних для Windows використовують певний набір стандартних функцій призначених для роботи з процесами, такі як: OpenProcess(), ReadProcessMemory(), VirtualQueryEx(), які описані в заголовкових файлах processthreadsapi.h та memoryapi.h [10]. Для захисту програми від створення дампу логічно здійснити перехоплення даних функцій при зверненні до процесу програми. Даний метод захисту від дампінгу називається антидампінгом у нульовому кільці [10]. Перехоплення системної функції у Windows можна здійснити двома шляхами: впровадження призначеної для користувача функції в адресний простір процесу, який моніториться, або підміна точки входу системної функції на функцію створену користувачем [20].

Іншим способом захисту від дампінгу є динамічне розпакування під час якого процес ніколи не перебуває в оперативній пам’яті повністю, оскільки антидампінговий модуль залишає розшифрованими тільки ті частинки програми, які зараз безпосередньо працюють, а ті які вже відпрацювали видаляються з пам’яті [10].

Одним з найпростіших рішень для простого користувача буде використання антивірусів, які мають функції для захисту від найбільш поширеного шкідливого програмного забезпечення, яке в тому числі може здійснювати дампінг [15]. Системи запобігання вторгненням теж можуть допомогти з пошуком дамперів відслідковуючи незвичну поведінку програм, їхні спроби отримати доступ до системних функцій та аналізуючи трафік в мережі на наявність пересилання дампів через мережу Інтернет [15]. Також якщо користувач не використовує в своїй роботі утиліти типу wmic чи PsExec, то варто заборонити їх використання за допомогою політик безпеки. Варто також оновлювати програми для захисту як можна частіше,

щоб розширювати антивірусні бази даних для пошуку шкідливого програмного забезпечення на основі сигнатурного методу пошуку.

Для захисту від атак, що здійснюються за допомогою такого інструменту як Mimikatz варто обмежити привілей SeDebugPrivilege для локальної групи адміністраторів BUILTINAdministrators у режимі debug в групових політиках [16]. Також варто відключити можливість використання протоколу WDigest, який може використовувати пароль користувача у відкритому вигляді. Захист LSA від підключення сторонніх модулів дозволить значно покращити захист пам’яті LSA, а заборона на використання LM та NTLM хешів дозволить уникнути їх використання для автентифікації Windows. Для захисту від викрадення хешів паролів з гілки реєстру HKEY_LOCAL_MACHINESECURITYCache, які спрощують доступність входу в систему при відсутності доступу до контролера домену потрібно внести заборону на кешування облікових даних в політиках безпеки [16].

Від атаки методом можна захиститись за допомогою використання EFI замість класичного BIOS’у, оскільки одразу після апаратного перезавантаження він записує випадкові байти по всій оперативній пам’яті через що дамп пам’яті втрачає сенс. Хоча інструменти для збору чутливої інформації існують також і для систем, які використовують EFI, тому найкращим захистом від подібної атаки буде контроль над фізичним доступом до машини [17].

Для захисту від атак, що можуть бути реалізовані через апаратні вразливості можна захиститися лише слідкуючи за оновленнями безпеки даного апаратного забезпечення та встановлюючи їх як тільки вони стають доступними [19].

На даний момент засоби захисту від дампінгу достатньо розвинені і можуть запропонувати методи на різних рівнях: починаючи від низькорівневих функцій, що працюють на рівні операційної системи до програмних засобів, які слідкують за роботою усієї системи. Найефективнішими з методів є максимальне обмеження привілеїв груп користувачів, відключення всіх невикористовуваних функцій, які потенційно можуть бути використані для зняття дампу та вчасні оновлення програмного та апаратного забезпечення.

1.4 Аналіз блокових шифрів

Шифрування – це криптографічне перетворення даних за допомогою ключових даних з метою їх приховання для неавторизованих користувачів [21]. В результаті відкритий текст перетворюється в шифротекст, з якого можна отримати оригінальний текст, тобто розшифрувати, тільки при знанні ключових даних та методу зашифрування. Шифрування здійснюється з такою метою [21]:

- конфіденційність – лише особи, які знають ключ можуть отримати оригінальне повідомлення;
- цілісність даних – навіть при найменшій зміні шифротексту блок даних буде пошкоджений і можна буде дати запит на повторне пересилання пошкоджених даних;
- нормативно-правові акти – практично всі установи згідно державного законодавства мають забезпечувати певні стандарти безпеки.

Розрізняють два типи шифрування [22]:

- симетричне;
- несиметричне.

Різниця між ними в тому, що симетричне для своєї роботи використовує лише один ключ, який знають лише відправник та адресат, тоді як несиметричне використовує два ключі: відкритий для зашифрування та секретний для розшифрування. Симетричне шифрування працює значно швидше, але має проблему захищеного поширення ключа між всіма особами, які повинні мати доступ, а несиметричне навпаки – працює довше, проте більш захищене оскільки проблема поширення ключів відсутня.

Симетричні алгоритми шифрування можна поділити на [22]:

- потокові – кожен біт даних шифрується незалежно від інших за допомогою гамування;
- блокові – шифрується одразу частина даних певного розміру.

В свою чергу блокові шифри діляться на [22]:

- шифри перестановки;
- шифри заміни;

Шифри перестановки переставляють біти вихідного тексту в певному наперед визначеному порядку, а шифри заміни замінюють певний набір даних інший в заздалегідь визначеній таблиці.

Найпопулярніші блокові шифри зазвичай будують на основі таких схем [24]:

- мережі Фейстеля;
- SP-мережі, або мережі змін-перестановок;

Мережа Фейстеля складається з однакових комірок на вхід кожної з яких надходять дані та раундовий ключ, які змінюються на виході. Розшифрування здійснюється так само як і зашифрування зі зміною порядку раундових ключів на протилежний [21]. Більшість шифрів на основі мережі Файстеля мають фіксовану кількість операцій в раунді, довжину ключа, розмір блоку та кількість раундів. Кількість раундів для шифрування коливаються від 4-х до 64-х. Для збільшення рівня стійкості до зламу необхідно збільшувати кількість раундів, що може значно знизити швидкість зашифрування [24].

SP-мережа для роботи використовує певний набір S- та P-блоків, які чергуються для отримання шифротексту з відкритого набору даних. Кількість раундів, що зазвичай використовують шифри побудовані на основі SP-мереж варіюється від 8-и до 16-и, довжина ключа від 128-и до 512-и біт, розмір блоку від 64-и до 128-и біт. Структура SP-мереж забезпечує високу стійкість до різних видів криптоаналізу, оскільки S-блоки зазвичай нелінійні та можуть займати великі розміри, проте це також може зменшити швидкість шифрування. Процедура зашифрування та розшифрування не є однаковими, що також ускладнює реалізацію шифрів, побудованих на даній основі [22].

В загальному, при побудові блокових шифрів потрібно орієнтуватися на кількість даних для зашифрування/розшифрування та критичність витрат часу на ці процеси, щоб знайти компроміс між захищеністю та доступністю користувача. Наприклад, усі шифри побудовані на основі мереж Фейстеля масово використовують бітові функції, такі як: додавання (Blowfish, LOKI97, RC5) та

множення (DFC, E2) за модулем 2 в певному ступені, циклічні зсуви (Camellia, HPC, KASUMI) та логічні операції “І” та “АБО” (Camellia, KASUMI, MISTY1), які можуть бути ефективно оптимізовані на сучасних процесорах [21]. Хоча в той же час шифри на основі SP-мереж також часто використовують бітові операції (SAFER+, Serpent, Threefish), проте в якості захисту від криптоаналізу більше покладаються на табличні перестановки (CS-Cipher, Serpent, Threefish) та заміни (ARIA, Khazad, Hierocrypt-L1) стійкість яких сильно залежить від їх розміру, а також множення на матрицю (ARIA, SC2000, SAFER+), що може негативно вплинути на кількість оперативної пам’яті та швидкість обчислень, що використовуються під час шифрування та розшифрування [24].

Згідно вище наведених методів було визначено, що найбільш ефективною схемою для побудови власного блокового шифру буде використання мережі Фейстеля через її вищу швидкодію, оскільки сучасні комп’ютери та процеси на них можуть використовувати велику кількість оперативної пам’яті одночасно і уповільнення її використання може сильно погіршити доступність для користувачів.

1.5 Постановка задачі

В ході дослідження методів для захисту від дампінгу було виявлено, що переважно з цією метою використовують антивірусні засоби, системи виявлення вторгнень та налаштування групових політик з метою блокування функцій для здійснення дампу. На рівні програмного коду необхідно уникати програмних інструментів, що використовують дампінг програм для зневадження, наприклад збирач сміття, та використовувати методи для ручного керування пам’яттю, такі як: збереження секретних даних в динамічному масиві з подальшою його очисткою після використання даних, уникання збереження секретних даних в рядках, обфускація даних в оперативній пам’яті до моменту використання, наприклад за допомогою гешування, та шифрування. Також було встановлено значні недоліки

стандартних методів, у вигляді низької ефективності проти обфускованих атак для зняття дампу [25-27].

Аналіз засобів показав, що доцільно розробити вбудований метод захисту від дампінгу для усунення недостатньої стійкості стандартних антидампінгових методів із збереженням продуктивності захищуваної програми і стійкості до обходу антидампінгового захисту.

1.6 Висновки з розділу

Розглянуто та проаналізовано загрози програмним застосункам. Визначено, що для них є суттєвою загрозою дампінг оперативної пам’яті. Відповідно, розглянуто методи атак та захисту від них за допомогою антивірусних засобів, систем виявлення вторгнень та налаштування групових політик. Також було проаналізовано блокові шифри та основні підходи до їх побудови. Аналіз показав необхідність удосконалення вбудованих методів захисту від дампінгу.

Встановлено необхідність покращення загального рівня захищеності програми від дампінгу порівняно із стандартними методами захисту від нього.

2 МЕТОД ЗАХИСТУ ВІД ДАМПІНГУ

2.1 Математичний опис процесу дампінгу

Математичний опис процесу дампінгу необхідний для того, щоб визначити всі компоненти, що будуть брати участь в захисті від дампінгу, тобто входи, які будуть позначені як I, внутрішні перетворення IT, що будуть здійснені над ними, внутрішні стани IS, в яких компоненти будуть перебувати в процесі захисту, результати цих перетворень O та взаємозв’язки між ними.

При множинному підході до опису системи можна виділити такі вхідні дані: сегмент пам'яті з програмним кодом програми C, в якому описані дії над даними, які будуть захищатися, сегмент пам'яті даних D, в якому зберігаються дані, що потребують захисту. Крім того, у випадку застосування шифрування на вході цього процесу потрібно отримувати множину ключів для симетричного шифрування K. Таким чином множина входів має вигляд:

$$I = \{C, D, K\} \tag{2.1}$$

Внутрішні перетворення в процесі захисту від дампінгу потрібні для того, щоб захистити дані та сам процес захисту, коли вони перебувають в оперативній пам’яті. Тому до них належать: встановлення хуків для функції дампінгу HF, для створення додаткового рівня захисту від дампінгу, створення множини ключів для шифрування KC, зашифрування даних DE на множині симетричних ключів K, очищення ділянок оперативної пам’яті MC, в яких містяться оригінальні дані. Таким чином множина внутрішніх перетворень має вигляд:

$$IT = \{HF, KC, DE, MC\} \tag{2.2}$$

Внутрішні стани відображають стан входів та проміжкових даних в процесі виконання захисту від дампінгу. До них належать: стан встановленості хуків на функції дампінгу SH, стан очищеності даних в оперативній пам’яті SC. Таким чином множина внутрішніх станів має вигляд:

$$IS = \{SH, SC\} \tag{2.3}$$

Результатом виконання цього процесу буде той же програмний код C, що й на вході, проте дані вже будуть зашифровані в оперативній пам’яті ED, множина симетричних ключів K, що також будуть використовуватися для розшифрування.

$$O = \{C, ED, K\} \quad (2.4)$$

Розглянувши всі елементи математичного опису було створено узагальнений опис захисту від дампінгу, який виглядає таким чином:

$$(I) \rightarrow [IT, IS] \rightarrow (O) \quad (2.5)$$

Використовуючи математичний опис було визначено всі компоненти, що будуть брати участь в захисті від дампінгу, внутрішні перетворення, що будуть здійснені над ними, внутрішні стани, в яких компоненти будуть перебувати в процесі захисту, результати цих перетворень та взаємозв’язки між ними.

2.2 Узагальнений опис методу

Метод захисту від дампінгу передбачає створення зовнішньої бібліотеки, яка буде містити в собі функції для захищеного зберігання даних в оперативній пам’яті та очищати ділянку пам’яті після закінчення роботи з нею.

Метод захисту даних передбачає такі етапи:

Крок 1. Встановлення хуків для функції дампінгу;

Крок 2. Визначення розміру даних для захисту та створення масиву відповідного розміру;

Крок 3. Розбиття даних на блоки та доповнення останнього блоку при необхідності;

Крок 4. Генерування одноразового сесійного ключа для шифрування;

Крок 5. Поблокове зашифрування даних та їх запис в створений масив;

Крок 6. Очищення ділянки оперативної пам’яті, де була записані дані до цього;

Крок 7. Очікування використання даних та їх розшифрування.

Крок 8. Очікування закінчення роботи з даними та очищення оперативної пам’яті, де вони знаходились;

Крок 9. Зняття хуків з функції дампінгу.

Використання зовнішньої бібліотеки дозволить вільно змінювати вихідний код для захисту та мінімально впливати на вже готові застосунки, що використовують дану бібліотеку. Використання одноразових ключів для шифрування дозволить зменшити ризики викрадення ключа та з часом вносити зміни в спосіб їх генерації. Збереження захищуваних даних у зашифрованому вигляді в оперативній пам’яті дозволить зменшити ризик отримання цих даних зловмисником.

2.3 Метод шифрування даних

Шифрування та розшифрування даних може бути достатньо трудомістким процесом, особливо при використанні об’єктів великого розміру, якими сьогодні оперують різні програмні засоби. Враховуючи це, а також роботу користувачів із цими даними в режимі реального часу, високий рівень доступності грає важливу роль в розроблюваному програмному засобі.

Саме тому для створення власного методу шифрування за основу було обрано симетричний блоковий шифр, що використовує мережу Фейстеля з двома гілками, під назвою DES [23]. Проте через застарілість та низький рівень захисту, зокрема через використання ключа та блоку малого розміру, даний шифр необхідно покращити, шляхом збільшення розмірності ключа до достатніх на сьогодні 128-ми біт та блоку до 256-ти біт, а також здійснити зміни в самому шифрі, щоб збільшити час грубого зламу до такого рівня, щоб дані на момент свого дешифрування втрачали свою актуальність.

Алгоритм одного раунду шифрування складається з таких етапів:

Крок 1. Вхідний блок ділиться на дві однакові частини;

Крок 2. Обчислюються ліва та права частини:

$$L_i = R_{i-1} \oplus f(L_{i-1}, K_{i-1}), \tag{2.6}$$

$$R_i = L_{i-1}, \tag{2.7}$$

де f – певна функція,

K_{i-1} – ключ і-го раунду,

$\oplus$ – результат суми за модулем 2.

Для розшифрування даних необхідно повторити всі операції у зворотньому порядку з використання раундових ключів у зворотньому порядку.

Алгоритм роботи шифру на основі мережі Фейстеля з двома гілками зображено на рисунку 2.1.

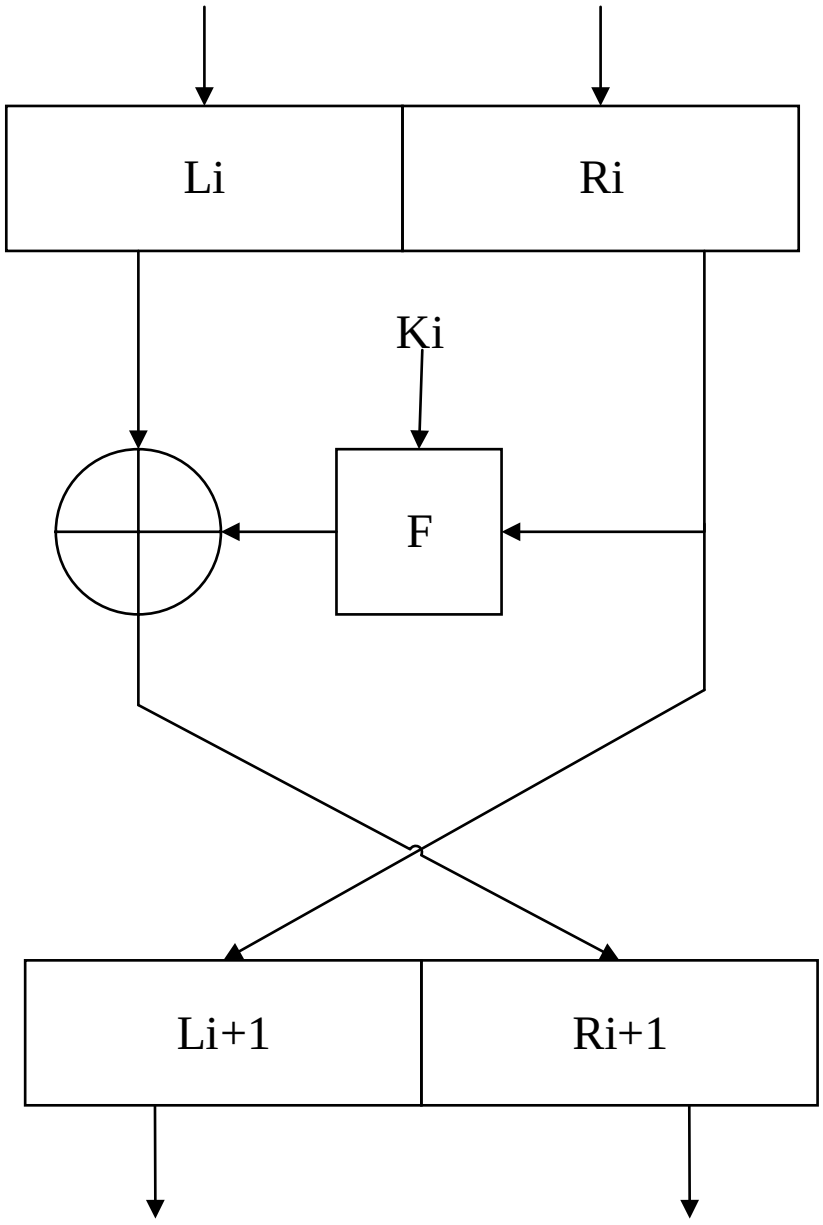


Рисунок 2.1 – Алгоритм одного циклу в мережі Фейстеля з двома гілками

Функція f, використовується на кожному раунді шифрування та розшифрування та не змінює свою. Вона складається з декількох кроків:

1) Перестановка з розширенням. Тут відбувається розширення 32-х вхідних бітів до 48-ми та їх перемішування;

2) Сума за модулем 2 з підключем, який також має розмірність 48 біт;

3) Результат отриманий на попередньому кроці ділиться на 6 рівних частин та переставляється за допомогою власної таблиці заміни;

4) Останнім кроком є перестановка бітів в певному порядку, результатом чого є 32-бітна послідовність.

Алгоритм роботи даної функції можна зображений на рисунку 2.2.

Рисунок 2.2 – Алгоритм роботи раундової функції

Для свого функціонування алгоритм DES потребує 16 підключів, які будуть використовуватися протягом 16-ти раундів. Порядок їх генерації такий:

1) Стиснення 64-бітного ключа до 56-ти бітного за допомогою фіксованої таблиці перестановки;

2) Розділ ключа на дві різні частини по 28 біт кожна;

3) Для ключів під номерами 1, 2, 9, 16 відбувається зсув на один біт вліво, для інших здійснюється зсув на 2;

4) Далі дві половинки ключа конкатенуються;

5) Останнім кроком є перестановка бітів в певному порядку, результатом чого є 48-бітна послідовність, яка використовується як підключ у відповідному раунді шифрування.

Використання мережі Фейстеля, як основи для шифру, значно спростить програмну реалізацію шифру та дозволить мати високу швидкість шифрування та розшифрування.

2.4 Метод блокування переривань копіювання пам'яті

Для захисту від використання системних функцій для створення дампу оперативної пам’яті було реалізовано переривання функцій ReadProcessMemory та WriteProcesssMemory, які зазвичай використовуються для створення дампу пам’яті процесу. Перехоплення функції було реалізовано за допомогою програмної бібліотеки MinHook, що дозволяє здійснювати перехоплення функцій в сімействі операційних систем Windows, які використовують як x86, так і x64 архітектуру [28]. Головна ідея роботи даної бібліотеки полягає в тому, що здійснюється заміна прологу перехоплюваної функції на машинному рівні на безумовний стрибок на замінену функцію.

Також в бібліотеку вбудовано систему станів за допомогою яких можна відслідковувати поточний стан процесу перехоплення, що значно спрощує етап розробки та зневадження.

На даний момент бібліотека вже не розвивається і всі її оновлення стосуються лише її використання з новими версіями компіляторів.

Для роботи функцій цієї бібліотеки необхідно мати:

- Функцію, яка буде викликатися замість перехопленої і містити ту ж саму сигнатуру, для того щоб уникнути помилок на етапі компіляції та вказівник на неї;
- Вказівник на функцію, яку потрібно перехопити;
- Вказівник на функцію-трамплін, що може бути використана, якщо необхідно буде викликати оригінальну функцію, проте це не обов’язковий параметр.

Порядок використання бібліотеки для створення хуків складається з таких кроків:

1) Створення функції, що буде використовуватися замість перехоплюваної та містити такий самий набір параметрів;
2) Ініціалізація програмної бібліотеки, яка повинна бути зроблена лише один раз, перед початком використання хуків;
3) Створення хуку в неактивному стані;
4) Активація хуку;
5) Виконання коду, який потребує перехоплення окремих функцій;
6) Деактивація хуку;
7) Виконання коду, який не потребує перехоплення функцій;
8) Якщо знову потрібно здійснювати перехоплення, то відбувається перехід до пункту 4;
9) Деініціалізація бібліотеки.

Візуально алгоритм використання хуків можна зобразити наступним чином, як показано на рисунку 2.3.

Рисунок 2.3 – Алгоритм перехоплення функцій копіювання пам'яті

Після того як програма закінчить роботу з даними, які необхідно захистити, необхідно деактивувати хуки та деініціалізувати бібліотеку для уникнення виключних ситуацій та помилок в роботі програми.

2.5 Висновки з розділу

Створення математичного опису процесу дампінгу дозволило вказати на важливі математичні аспекти, які лежать в основі процесу дампінгу та оцінити всі етапи створення дампу процесу, на які потрібно зважати при створенні захисту. Він включив в себе формалізовані алгоритми, рівняння та математичні моделі, що використовуються для забезпечення безпеки та ефективності під час здійснення захисту від дампінгу. Створення узагальненого опису методу захисту від дампінгу дозволило розбити задачу побудови захисту на менші підзадачі, що спростило зневадження окремих модулів бібліотеки та покращило адаптивність коду для внесення додаткової функціональності в майбутньому. Також узагальнений опис вказав на різноманітні підходи і методи, які можуть бути використані для захисту від дампінгу. Це може включати технології, алгоритми, принципи та стратегії, які допомагають забезпечити надійність і конфіденційність даних під час обміну. Розробка методу шифрування даних дозволила створити власний метод для шифрування області пам’яті із захищуваною інформацією та спростити програмну реалізацію шифру. Опис методу блокування переривань копіювання пам'яті дозволив спростити додавання ще одного рівня захисту від дампінгу пам’яті та визначити сильні та слабкі сторони використовуваного методу.

3 ЗАСІБ ЗАХИСТУ ВІД ДАМПІНГУ

3.1 Узагальнена архітектура засобу

Архітектура захисту не відрізняється при встановленні та знятті захисту. В процесі захисту пам’яті беруть участь:

- Дані, які обирає користувач;
- Інструкції для очищення оперативної пам’яті оригінальних даних;
- Інструкції для перехоплення функцій системних бібліотек, що призначені для зняття дампу оперативної пам’яті;
- Інструкції для шифрування даних;
- Оперативна пам’ять.

Узагальнену архітектуру засобу можна зобразити наступним чином, як показано на рисунку 3.1.

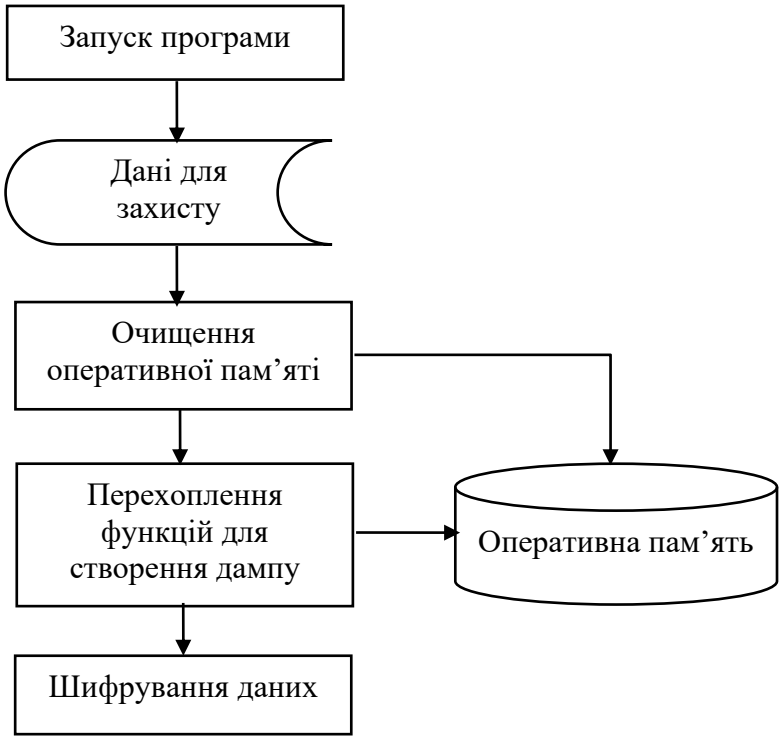


Рисунок 3.1 – Узагальнена архітектура засобу

Робота бібліотеки починається не одразу після запуску програми, що її використовує, а лише після того як користувач створить об’єкт шифроконтейнеру та заповнить всі поля даного об’єкту. Далі метод захисту використовує поля об’єкту для визначення місця в оперативній пам’яті, куди потрібно здійснити запис та обчислення розміру даних, які потрібно туди записати. Наступним кроком методу захисту є встановлення перехоплення функцій, які намагаються отримати доступ до оперативної пам’яті процесу. В цих процесах об’єктом виступає оперативна пам’ять, куди відбувається запис та здійснюється перехоплення при спробі отримати доступ до ділянки, що виділена для процесу. Останнім кроком встановлення захисту є шифрування даних, які потрібно захистити.

Створення узагальненої архітектури засобу дозволяє виділити ключові елементи, на які потрібно звернути увагу при програмній реалізації бібліотеки.

3.2 Архітектура модулю очищення оперативної пам’яті

Архітектура модулю очищення оперативної пам’яті не відрізняється протягом всього часу роботи програмної бібліотеки. В процесі очищення оперативної пам’яті беруть участь:

- Дані, які обирає користувач;
- Об’єкт для роботи з оперативною пам’яттю;
- Інструкції для обчислення розміру старих даних для їх послідуючої заміни в оперативній пам’яті;
- Інструкції для запису незначущих даних в оперативну пам’ять замість оригінальних;
- Оперативна пам’ять.

Архітектуру модулю очищення оперативної пам’яті можна зобразити наступним чином, як показано на рисунку 3.2.

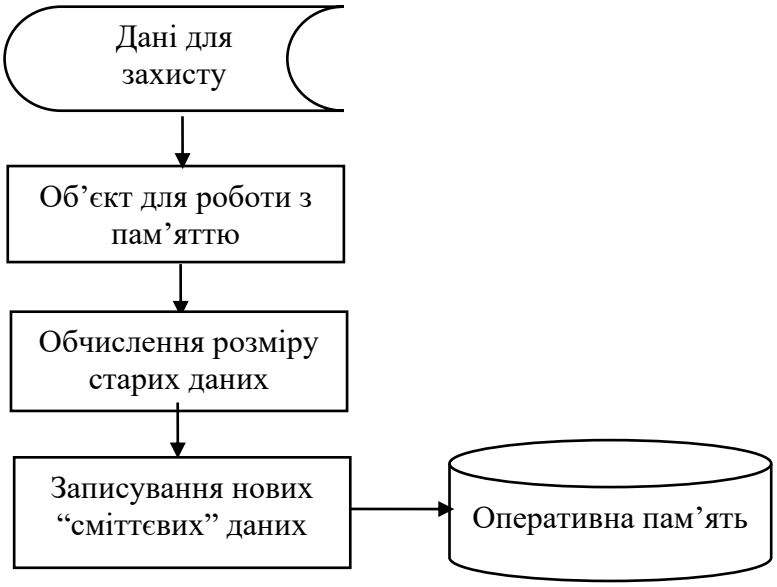


Рисунок 3.2 – Узагальнена архітектура модулю очищення оперативної пам’яті

Для правильної роботи модулю очищення оперативної пам’яті потрібно правильний тип даних та адреса за якими вони зберігаються, щоб уникнути виключних ситуацій при роботі з пам’яттю. Далі повинен бути створений об’єкт для роботи з пам’яттю, який має методи для роботи з пам’яттю. Наступним кроком є обчислення розміру “сміттєвих даних”, що будуть записані на місце оригінальних та мати той же самий розмір, щоб уникнути змін у структурі оперативної пам’яті даного процесу. Останнім кроком очищення оперативної пам’яті є записування раніше обчислених “сміттєвих даних” на те ж саме місце в оперативній пам’яті.

Створення архітектури модулю очищення оперативної пам’яті дозволяє виділити ключові елементи, на які потрібно звернути увагу при програмній реалізації бібліотеки.

3.3 Архітектура модулю перехоплення функцій для створення дампу

Архітектура модулю перехоплення функцій для створення дампу не відрізняється протягом всього часу роботи програмної бібліотеки. В процесі перехоплення функцій для створення дампу беруть участь:

- Функції для дампінгу та їх заміни;
- Об’єкт для роботи з перехопленнями;
- Інструкції для створення хуків;
- Стек оперативної пам’яті.

Архітектуру модулю перехоплення функцій для створення дампу можна зобразити наступним чином, як показано на рисунку 3.3.

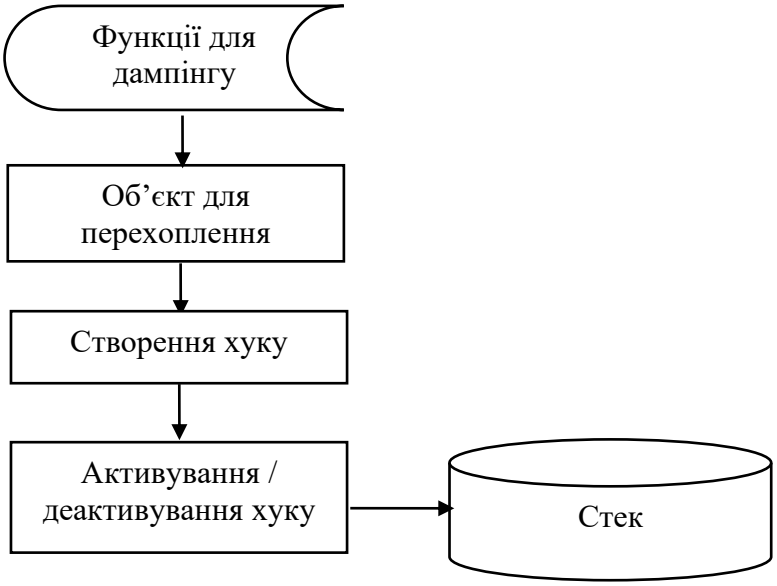


Рисунок 3.3 – Узагальнена архітектура модулю перехоплення функцій для створення дампу

Для того, щоб вдало здійснити перехоплення функцій, що призначені для створення дампу, необхідно отримати посилання на них на функції-замінника. Хоча в бібліотеці за замовчуванням визначено перехоплення функцій ReadProcessMemory та WriteProcessMemory, користувач також може здійснювати перехоплення інших функцій, наприклад інших системних функцій, що лежать в їх основі. Далі повинен бути створений об’єкт для перехоплення функцій, який має методи для роботи з бібліотекою для здійснення хуків. Після цього потрібно створити хуки для перехоплення системних функцій з правильним набором параметрів. Активація, або деактивація хуку буде здійснюватися на вибір користувача, коли він буде вмикати, або вимикати захист. В свою чергу бібліотека,

яка здійснює перехоплення взаємодіє зі стеком, щоб перезаписувати адресу точки старту функції.

Створення архітектури модулю перехоплення функцій для створення дампу дозволяє виділити ключові елементи, на які потрібно звернути увагу при програмній реалізації бібліотеки.

3.4 Архітектура модулю шифрування пам’яті

Архітектура модулю шифрування пам’яті не відрізняється протягом всього часу роботи програмної бібліотеки. В процесі очищення оперативної пам’яті беруть участь:

- Дані, які обирає користувач;
- Об’єкт для роботи з оперативною пам’яттю;
- Інструкції для обчислення розміру старих даних для їх послідуючої заміни в оперативній пам’яті;

Архітектуру модулю шифрування пам’яті можна зобразити наступним чином, як показано на рисунку 3.4.

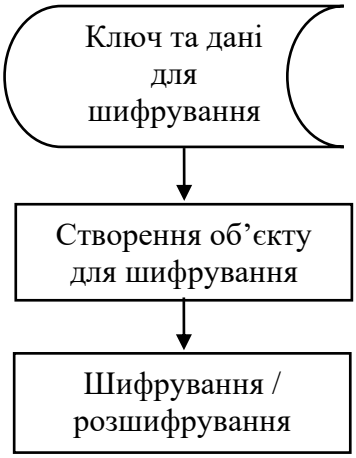


Рисунок 3.4 – Узагальнена архітектура модулю шифрування пам’яті

Для того, щоб правильно зробити шифрування та розшифрування потрібно отримати дані та правильно їх перетворити у форму, що підходить для зашифрування та безпечний ключ правильної довжини. Далі повинен бути

створений об’єкт для шифрування, який має методи для шифрування, розшифрування та всіх внутрішніх перетворень. Вже даний об’єкт здійснює шифрування чи розшифрування, в залежності від встановлення чи зняття захисту та записує його собі в пам’ять в зашифрованому вигляді.

Створення архітектури модулю шифрування пам’яті дозволяє виділити ключові елементи, на які потрібно звернути увагу при програмній реалізації бібліотеки.

3.5 Узагальнений алгоритм роботи засобу

Початок та закінчення роботи бібліотеки визначається та керується користувачем в процесі виконання самої програми. Після початку роботи програми відбувається створення нового захищеного об’єкту, який буде містити зашифровані дані, поки користувачу не знадобляться оригінальні дані. Після цього оригінальні дані будуть перезаписані на інші, що не мають ніякого сенсу, проте мають той же розмір, щоб не здійснювати ніяких змін в оперативній пам’яті процесу. Далі відбувається встановлення хуків для перехоплення системних функцій, які використовуються для зняття дампу зловмисними програмами. Під час роботи захисту користувач не зможе використовувати дані функції, проте вони знову будуть доступні при зупинці захисту чутливих даних.

Після цього необхідно згенерувати одноразовий ключ шифрування розміром в 128 біт, який буде використовуватися для симетричного шифрування до наступної активації захисту. При повторній активації захисту буде згенеровано новий ключ, на якому буде відбуватися подальше шифрування та розшифрування.

Далі буде відбуватися зашифрування даних, що потребуватимуть захисту за допомогою новоствореного ключа шифрування. У якості методу шифрування буде використовуватися модифікований симетричний блоковий шифр DES зі збільшеними розмірами ключа та блоку. В результаті всіх вище описаних дій дані за оригінальною адресою будуть видозмінені, що унеможливить їх використання

до зняття захисту. Оригінальні дані будуть зберігатися у зашифрованому вигляді, до моменту коли користувач знову захоче їх використовувати та зніме захист.

При знятті захисту будуть відбуватися кроки зворотні до вищеописаних при встановленні захисту, а саме:

1) Зупинка перехоплення системних функцій, які використовуються для зняття дампу;

2) Розшифрування захищених даних за допомогою симетричного ключа створеного раніше;

3) Запис розшифрованого значення за попередньою адресою, щоб користувач міг використовувати ці дані як і до цього.

Якщо користувач знову захоче захищати дані, він може відновити захист за допомогою одного методу, або створити новий об’єкт для захисту інших даних. При відновленні захисту того ж об’єкту в оперативній пам’яті буде змінено:

- При зміні розміру оригінальних даних буде також змінено розмір даних, що були перезаписані на місце оригінальних;
- Об’єкт для генерації ключа;
- Ключ для шифрування;
- Шифротекст, отриманий при зашифруванні даних.
- Системний час.

Системний час має важливе значення, оскільки від нього значно залежить генерація псевдовипадкового ключа.

Подібних ітерацій може здійснюватися стільки потрібно користувачу.

При закінченні роботи програми, що використовує дану програмну бібліотеку буде здійснено повну зупинку її роботи та деініціалізація внутрішніх об’єктів, які вона використовує в процесі своєї роботи. Алгоритм роботи бібліотеки показано на рисунку 3.5.

Рисунок 3.5 – Узагальнений алгоритм роботи бібліотеки

Робота бібліотеки залишається незмінною для кожного проходження, окрім даних користувача, які він може змінювати за власними потребами. При даній структурі важливо використовувати дані, які представляють лише текстові типи даних, проте користувач може здійснити перетворення даних з необхідного йому типу даних до текстового.

3.6 Процедура створення захищеного контейнеру

Головною задачею процедури створення захищеного контейнеру є ініціалізація всіх необхідних полів та допоміжних об’єктів, методи яких будуть використовуватись для внутрішніх перетворень. До полів, що містить шифроконтейнер входять:

- Копія даних, що будуть захищатися;
- Адреса, де ці дані зберігаються;
- Ключ, що буде використовуватися для шифрування.

До допоміжних об’єктів захисту належать:

- Об’єкт для перехоплення функцій системних бібліотек, що здійснюють дампінг;
- Об’єкт для здійснення криптографічних перетворень.

Алгоритм створення захищеного контейнеру показано на рисунку 3.6.

Першим кроком створення є запис всіх даних у відповідні поля, які в даному випадку мають рядковий тип даних, проте при використанні іншого типу даних алгоритм не зміниться, тому користувач може написати власну обгортку під власний тип даних. Далі повинна відбутися перевірка чи справді за отриманною адресою знаходяться ті дані, що передані в поле даних. Вкінці створення об’єкту буде ініціалізовано допоміжні об’єкти для захисту, а саме об’єкт для створення хуків системних функцій, що здійснюють дамп оперативної пам’яті та об’єкт для здійснення шифрування та розшифрування.

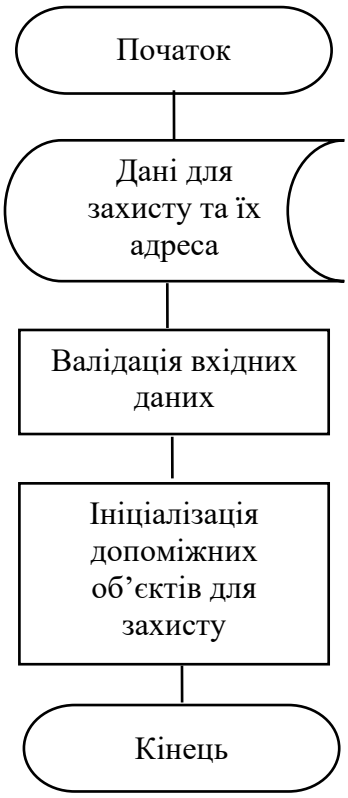


Рисунок 3.6 – Алгоритм створення захищеного контейнера

Захищений контейнер призначений лише для того, щоб зберігати дані, тому його структура є незмінною протягом усього виконання програми.

3.7 Процедура перезаписування оригінальних даних

Головною задачею процедури перезаписування оригінальних даних є перезапис даних за адресою оригінальних на сміттєве значення, тієї ж розмірності, що й оригінальні, щоб зберегти структуру оперативної пам’яті процесу. До полів, що використовує процедура перезаписування оригінальних даних входять:

- Ідентифікатор процесу, в даному випадку – ідентифікатор власного процесу;
- Дескриптор процесу, який можна отримати за допомогою його ідентифікатора;
- Адреса, куди буде відбуватися запис;
- Нове значення, яке буде записане замість оригінального.

Алгоритм процедури перезаписування оригінальних даних показано на рисунку 3.7.

Рисунок 3.7 – Алгоритм процедури перезаписування оригінальних даних

Першим кроком процедури перезаписування є отримання адреси для запису та нове значення, яке вона буде містити. Ці далі були отримані в результаті виконання попередніх процедур. Ідентифікатор процесу можна отримати за допомогою системної функції _getpid, а дескриптор за допомогою функції OpenProcess з параметром у вигляді раніше отриманого ідентифікатору. Далі викликається функція WriteProcessMemory, яка отримує всі вхідні дані та дані, що були отримані на попередніх кроках процедури та здійснює запис нового значення в оперативну пам’ять.

Процедура перезапису оригінальних даних призначена для того, щоб зробити обгортку навколо системної функції запису за адресою оперативної

пам’яті, зменшити кількість вхідних параметрів, зробити їх самообчислюваними та досягнути модульності в загальній структурі програмної бібліотеки, тому його структура є незмінною протягом усього виконання програми.

3.8 Процедура створення та активація хуку

Головною задачею процедури створення та активації хуку є перехоплення системних функцій, що здійснюють дампінг операційної системи та часто використовуються зловмисниками для дампінгу своїх цілей. До полів, що використовує процедура створення та активація хуку входять:

- Оригінальні функції, що будуть перехоплюватися;
- Вказівники на оригінальні функції;
- Функції, що будуть викликані замість оригінальних;
- Вказівники на функції-замінники;
- Функції-трампліни, що можуть використовуватися замість оригінальних функцій;
- Вказівники на функції-трампліни;
- Допоміжні дані про функції;
- Стан перехоплення.

Алгоритм процедури створення та активації хуку показано на рисунку 3.8.

Першим кроком процедури створення та активації хуку є отримання всіх вхідних даних, які були наведені вище. Далі необхідно здійснити ініціалізацію та отримати код MH_OK, щоб переконатися, що операція завершилася успішно. Наступним кроком є створення хуків в неактивному вигляді. Для цього на вхід функції подається:

- Вказівник на функцію-ціль;
- Вказівник на функцію-замінник;
- Вказівник на функцію-трамплін.

Результатом виконання даного кроку теж має бути код MH_OK. Далі при активації чи деактивації захисту буде виконано функцію EnableXHook, або

DisableXHook, що активують, або деактивують хук відповідно. В кінці роботи програми необхідно викликати функцію Uninitialize, щоб деініціалізувати виконання бібліотеки для перехоплення.

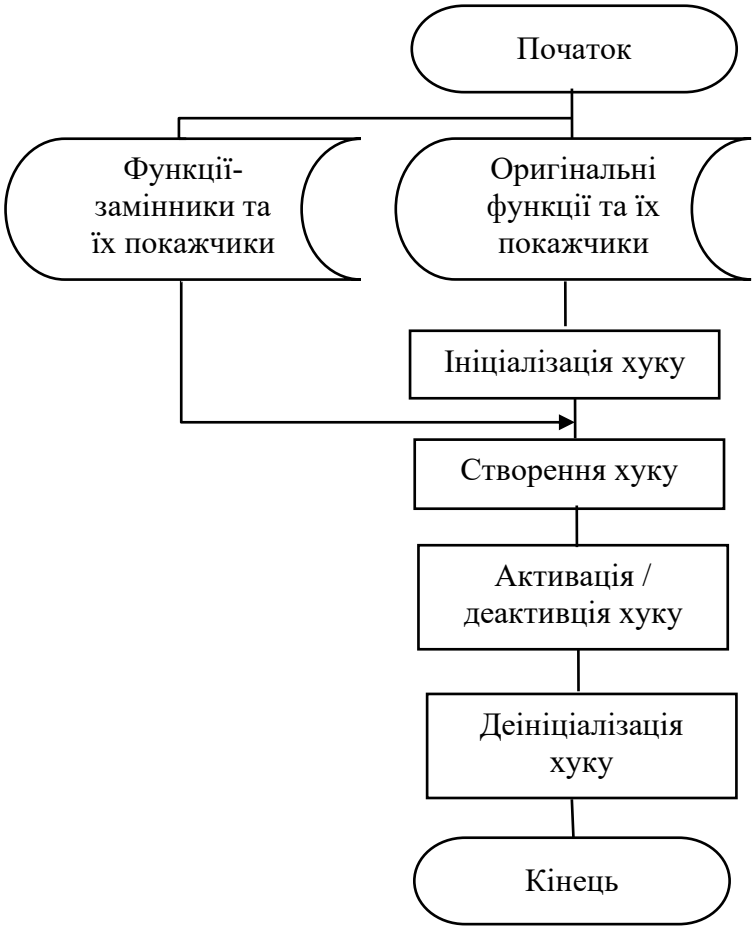


Рисунок 3.8 – Алгоритм процедури створення та активації хуку

Процедура створення та активації хуку призначена для того, щоб зробити обгортку навколо бібліотеки перехоплення, зменшити кількість вхідних параметрів, зробити їх самообчислюваними та досягнути модульності в загальній структурі програмної бібліотеки, тому його структура є незмінною протягом усього виконання програми.

3.9 Процедура створення одноразового ключа для шифрування

Головною задачею процедури створення одноразового ключа для шифрування є створення ключа шифрування, що буде використовуватись до призупинення захисту користувачем. Процедура створення одноразового ключа для шифрування не використовує вхідних параметрів, окрім системного часу.

Алгоритм процедури створення одноразового ключа для шифрування показано на рисунку 3.9.

Рисунок 3.9 – Алгоритм створення одноразового ключа

Першим кроком процедури створення одноразового ключа є отримання системного часу, на основі якого буде генеруватися псевдовипадковий ключ. Після цього буде відбуватися генерування псевдовипадкових чисел та їх перевірка на

парність в результаті чого до ключа буде дописуватися нульовий, або одиничний біт. Наступним кроком є перевірка на максимальну кількість однакових бітів підряд. Якщо згенерований ключ пройшов перевірку, то він визнається стійким та використовується в шифруванні / розшифруванні в подальшому, інакше генерується заново.

Процедура створення одноразового ключа призначена для того, щоб створити одноразовий ключ для шифрування, тому його структура є незмінною протягом усього виконання програми.

3.10 Процедура зашифрування / розшифрування

Головною задачею процедури зашифрування / розшифрування є здійснення криптографічних перетворень для забезпечення конфіденційності даних. До полів, що використовує дана процедура входять:

- Ключ для шифрування;
- Дані для шифрування;
- Масив раундових ключів;
- Розмір доповнення до блоку;

Алгоритм процедури зашифрування / розшифрування показано на рисунку 3.10.

Першим кроком даної процедури є отримання ключа для шифрування та даних, які були закодовані в двійковому вигляді. Далі на основі ключа шифрування буде створено масив з 16 підключів, які будуть використовуватися протягом наступних 16-ти раундів. Потрібно також зазначити, що вся криптостійкість алгоритму залежить від секретності ключа шифрування та відповідно методу утворення масиву додаткових підключів. Наступним кроком необхідно розбити вхідний текст на блоки розміром в 256 бітів, щоб алгоритм працював тільки з невиликими порціями даних. В кінці здійснюється шифрування, або розшифрування в залежності від того, що зараз відбувається – встановлення, або зняття захисту. Етапи зашифрування та розшифрування відрізняються між собою

лише порядком використання ключів, а також тим, що розшифрування не потребує вхідних даних.

Рисунок 3.10 – Алгоритм процедури шифрування / розшифрування

Процедура шифрування / розшифрування призначена для того, щоб зробити обгортку навколо механізму шифрування, зменшити кількість вхідних параметрів, зробити їх самообчислюваними та досягнути модульності в загальній структурі програмної бібліотеки, тому його структура є незмінною протягом усього виконання програми.

3.11 Висновки з розділу

Створення узагальненої архітектури засобу дозволило виділити ключові елементи, на які потрібно звернути увагу при програмній реалізації бібліотеки. Створення архітектури модулю очищення оперативної пам’яті дозволило виділити ключові елементи, на які потрібно звернути увагу при програмній реалізації очищення оперативної пам’яті. Створення архітектури модулю перехоплення функцій для створення дампу дозволило виділити ключові елементи, на які потрібно звернути увагу при програмній реалізації перехоплення функцій для створення дампу. Створення архітектури модулю шифрування пам’яті дозволило виділити ключові елементи, на які потрібно звернути увагу при програмній реалізації шифрування пам’яті. Розробка узагальненого алгоритму роботи засобу дозволила сформувати модульну структуру засобу та розділити програмний засіб та процедури на складові, що дозволило значно спростити проектування та програмну реалізацію засобу. Розробка процедури створення та роботи захищеного контейнеру дозволила зберігати дані протягом усього виконання програми. Розробка процедури перезапису оригінальних даних дозволила зробити обгортку навколо системної функції запису за адресою оперативної пам’яті, зменшити кількість вхідних параметрів, зробити їх самообчислюваними та досягнути модульності в загальній структурі програмної бібліотеки. Розробка процедури створення та активації хуку дозволила зробити обгортку навколо бібліотеки перехоплення, зменшити кількість вхідних параметрів, зробити їх самообчислюваними та досягнути модульності в загальній структурі програмної бібліотеки.

4 ТЕСТУВАННЯ ЗАСОБУ ЗАХИСТУ ВІД ДАМПІНГУ

4.1 Обґрунутвання вибору засобів розробки та тестування

Програмну бібліотеку можна реалізувати за допомогою різних мов програмування: Go, Java, JavaScript, Pascal, Fortran. Проте через специфічність поставленої задачі необхідно використовувати низькорівневу мову програмування, що може маніпулювати оперативною пам’яттю, використовувати вказівники та мати можливість компілювати як статичні, так і динамічні бібліотеки. Тому було обрано C++, як мову програмування для розробки магістерської дипломної роботи. Крім виконання всіх вищенаведених вимог C++, має можливість використання підходу ООП, що дозволяє створювати складні об’єкти в процесі написання коду, зробити структуру більш модульною, що спростить їх написання та відлагодження, а також використовує заголовкові файли, що дозволяє вільно замінювати файли з вихідним кодом, що мають той же набір класів та методів з однаковими сигнатурами.

Для розробки програми було використано середовище Visual Studio 2022 для мови програмування C++, як фреймворк тестування було використано платформу модульного тестування Microsoft для C++.

Visual Studio 2022 – середовище розробки, яке може використовуватися для найрізноманітніших проектів, починаючи від розробки, збирання та тестування програм до зміни веб-сторінок у вбудованому веб-конструкторі [29].

4.2 Блокове тестування

Метою блокового тестування є перевірка працездатності всіх окремих модулів програми та для того щоб при виникненні помилок в більш широкому тестуванню можна було бути впевненим, що проблема знаходиться не в вихідному коді.

Перш за все було протестовано функціональність, яка забезпечує генерування ключа, оскільки від правильності та стійкості ключа залежить чи

будуть дані захищені під час активного захисту та чи зможе користувач отримати доступ до захищених даних після зняття захисту.

При генерації ключа потрібно переконатися в декількох речах:

- Чи згенерований ключ має правильну довжину;
- Чи складається ключ з двійкових символів;
- Чи задовільняє ключ вимогам максимальної довжини серії однакових символів;

Результати блокового тестування модулю генерування ключів можна побачити на рисунку 4.1.

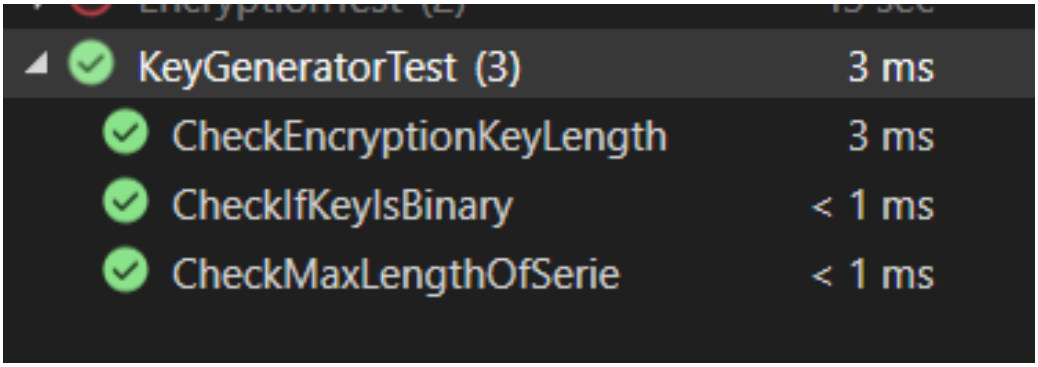


Рисунок 4.1 – Результат тестування модуля генерації ключів

Перш за все було протестовано функціональність, яка забезпечує генерування ключа, оскільки від правильності та стійкості ключа залежить чи будуть дані захищені під час активного захисту та чи зможе користувач отримати доступ до захищених даних після зняття захисту.

Далі було протестовано роботу модулю шифрування. Серед публічних функцій необхідно було протестувати:

- Чи правильно декодуються двійкові рядкові значення до цілочисельних;
- Чи правильно декодуються цілочисельні значення до двійкових рядкових значень;
- Чи здійснюються зашифрування та розшифрування, тобто чи збігаються зашифровані та розшифровані значення;

Результати блокового тестування модулю шифрування можна побачити на рисунку 4.2.

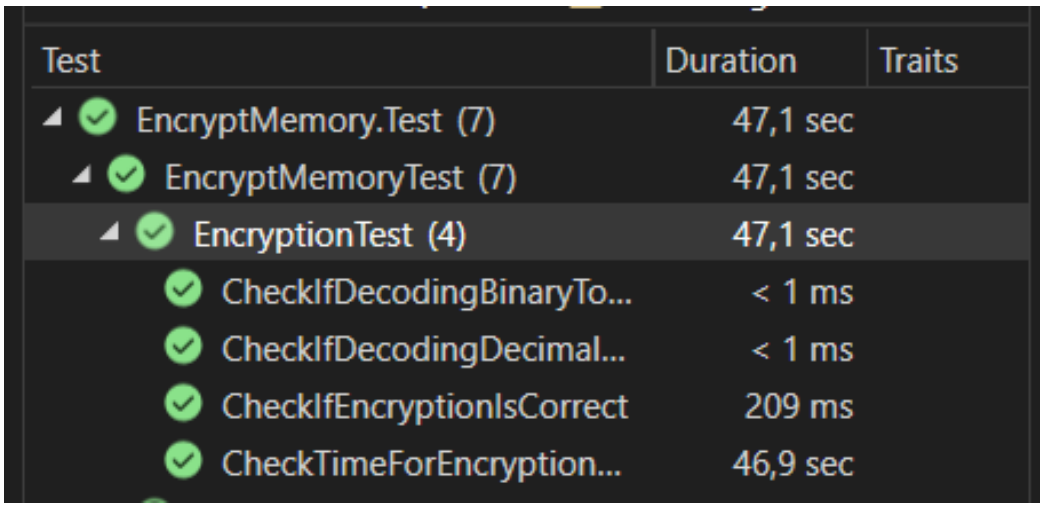


Рисунок 4.2 – Результат тестування модуля шифрування

Далі було протестовано роботу модулю перехоплення. Серед публічних функцій необхідно було протестувати:

- Чи здійснюється перехоплення функції читання оперативної пам’яті;
- Чи здійснюється перехоплення функції запису в оперативну пам’ять.

Результати блокового тестування модулю перехоплення можна побачити на рисунку 4.3.

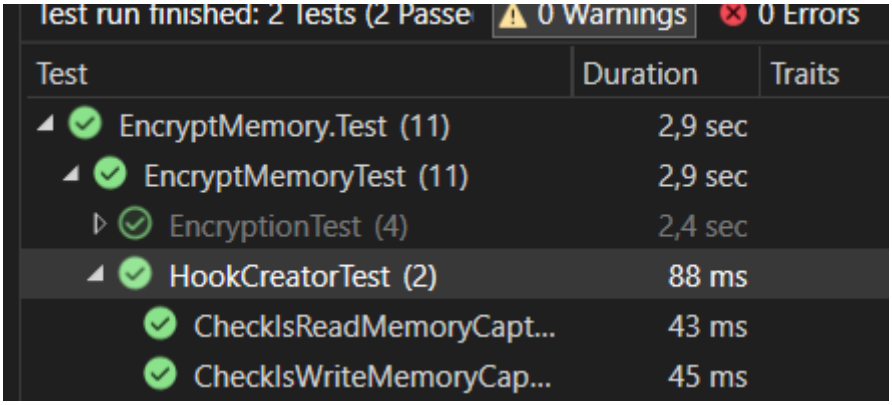


Рисунок 4.3 – Результат тестування модуля перехоплення

Блокове тестування дозволило протестувати працездатність кожного модуля програмної бібліотеки окремо, що зменшило кількість помилок в програмному забезпеченні та утворення виключних ситуацій, що були спричинені неправильною реалізацією модулів.

4.3 Статичне тестування безпеки

Для перевірки безпеки розробленої системи та виявлення помилок у її роботі було проведено статичне тестування виконуваного коду програмного застосунку. Для проведення статичного тестування було використано розширення для Visual Studio 2022, PVS Studio. З його допомогою можна здійснювати аналіз застосунків написаних на мовах C, C#, C++ та Java. Основною перевагою у роботі даної системи є можливість переглядати причину виникнення помилок в іншому вікні програмного середовища.

Результати статичного тестування, зображено на рисунку 4.4.

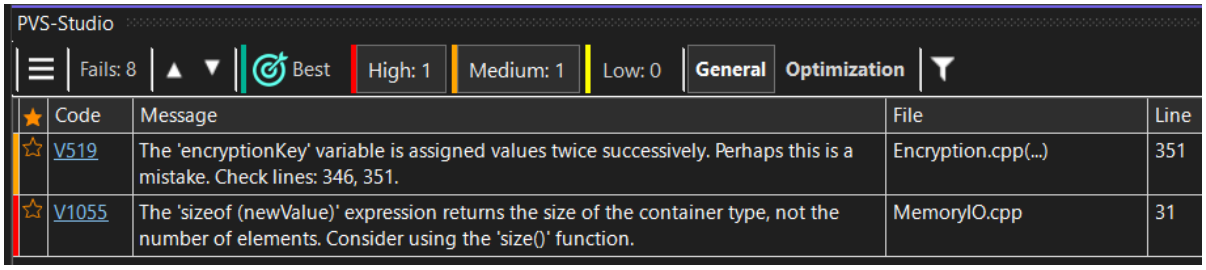


Рисунок 4.4 – Результат статичного тестування програми

Було виявлено декілька помилок, а саме:

- Подвійне присвоєння значення одній змінній;
- Використання функції sizeof, яка повертає розмір об’єкту, а не кількість елементів в колекції.

Виправлення першої помилки дозволило спростити умовну конструкцію при валідації криптографічного ключа в файлі з інструкціями до шифрування. Друга помилка не має сенсу, оскільки там використовується рядковий тип даних, що містить елементи розмірністю в 1 байт, тому розмір рядка є також кількістю його

елементів, проте щоб уникнути хибного розуміння коду дана помилка була усунута.

Також PVS Studio має вбудований у середовище браузер та може відобразити детальніше пояснення помилки, як це показано на рисунку 4.5.

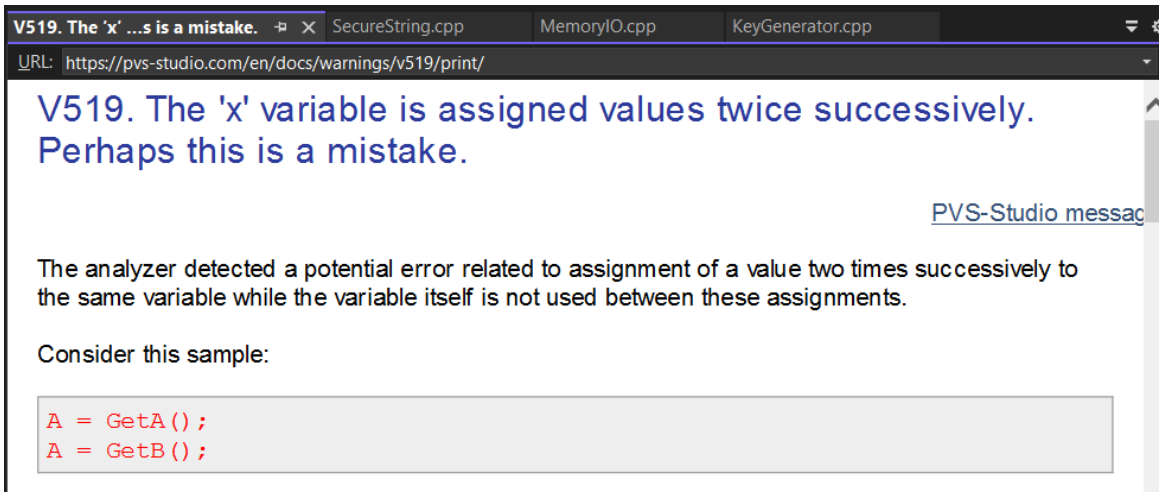


Рисунок 4.5 – Відображення пояснення помилки у вбудованому в середовище розробки браузері

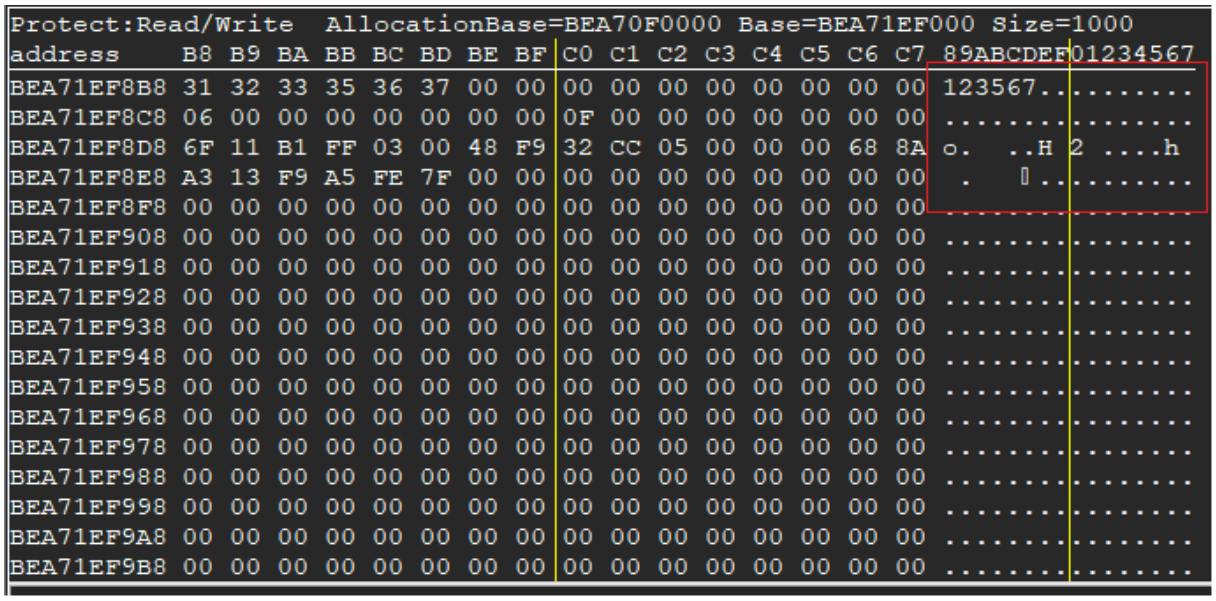
Приклад показаний вище демонструє опис помилки, приклади, коли вона може виникати, вірогідний варіант вирішення даної проблеми та номер її класифікації.

4.4 Інтеграційне тестування

Метою інтеграційного тестування є перевірка працездатності всієї системи повністю, в такому вигляді як її буде використовувати кінцевий користувач бібліотеки.

В рамках даного тестування буде створено зміну рядкового типу даних, увімкнено захист, вимкнено захист, змінено значення цих даних, знову увімкнено захист та знову його деактивовано.

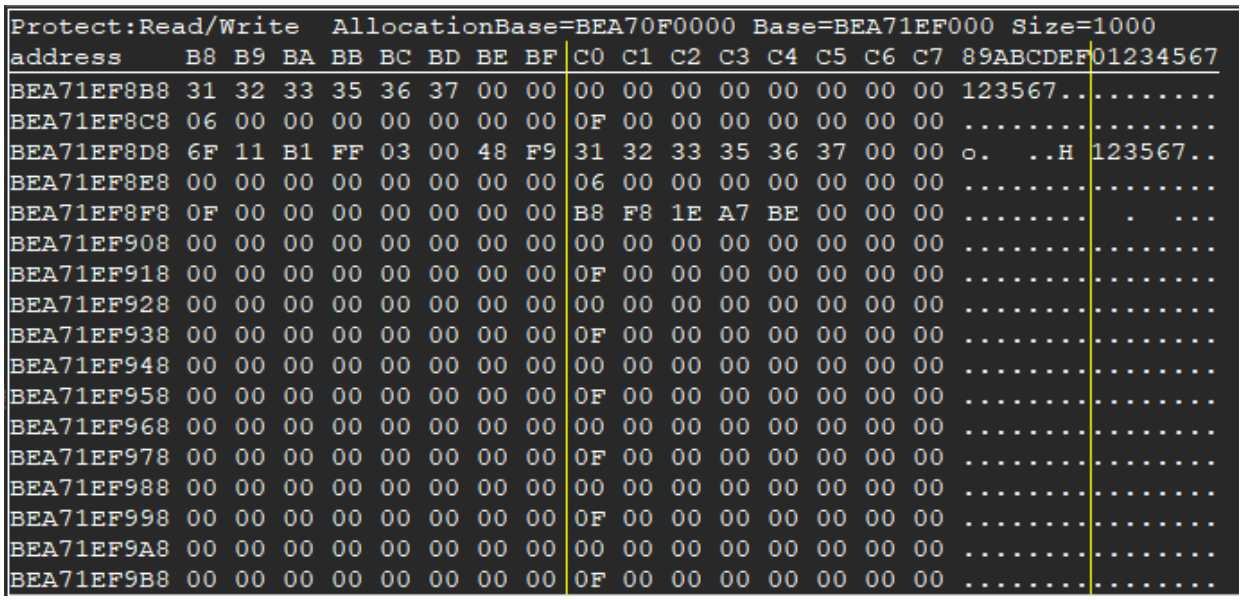
На рисунку 4.6, що показаний нижче можна побачити стан оперативної пам’яті до увімкнення захисту.



```
Protect:Read/Write  AllocationBase=BEA70F0000 Base=BEA71EF000 Size=1000
address     B8 B9 BA BB BC BD BE BF C0 C1 C2 C3 C4 C5 C6 C7 89ABCDEF01234567
BEA71EF8B8 31 32 33 35 36 37 00 00 00 00 00 00 00 00 00 00 123567..........
BEA71EF8C8 06 00 00 00 00 00 00 00 0F 00 00 00 00 00 00 00 ................
BEA71EF8D8 6F 11 B1 FF 03 00 48 F9 32 CC 05 00 00 00 68 8A o.   ..H 2 ....h
BEA71EF8E8 A3 13 F9 A5 FE 7F 00 00 00 00 00 00 00 00 00 00   .   ..........
BEA71EF8F8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
BEA71EF908 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
BEA71EF918 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
BEA71EF928 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
BEA71EF938 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
BEA71EF948 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
BEA71EF958 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
BEA71EF968 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
BEA71EF978 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
BEA71EF988 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
BEA71EF998 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
BEA71EF9A8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
BEA71EF9B8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

Рисунок 4.6 – Вміст оперативної пам’яті до увімкнення захисту

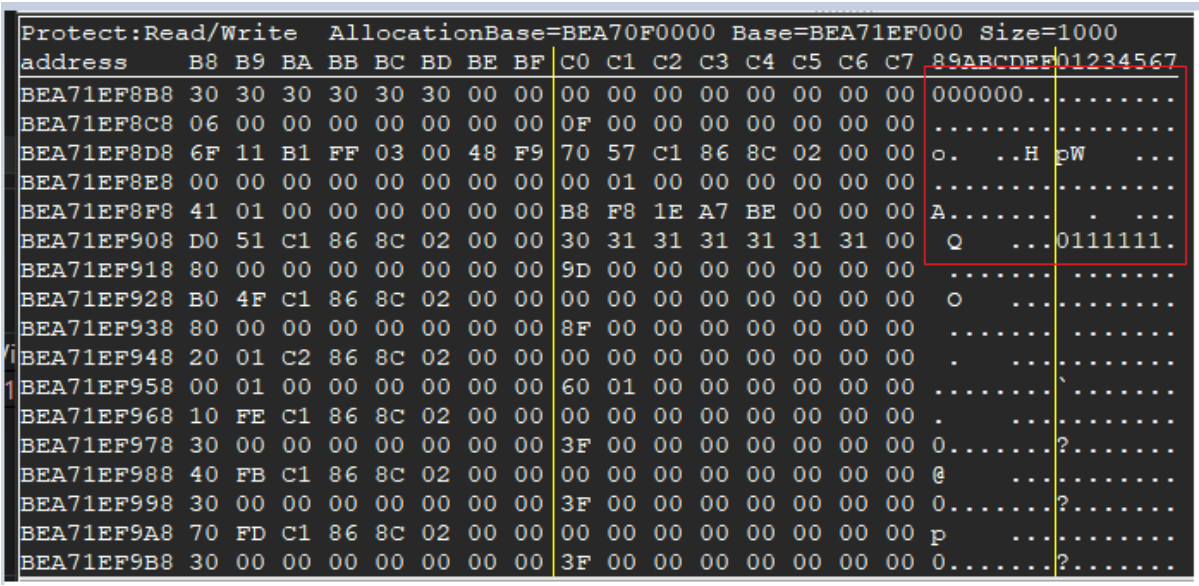
На даному зображені видно адресу в пам’яті, де зберігається змінна та її вміст. На рисунку 4.7 показано як змінюється стан оперативної пам’яті після створення шифроконтейнеру.



```
Protect:Read/Write  AllocationBase=BEA70F0000 Base=BEA71EF000 Size=1000
address     B8 B9 BA BB BC BD BE BF C0 C1 C2 C3 C4 C5 C6 C7 89ABCDEF01234567
BEA71EF8B8 31 32 33 35 36 37 00 00 00 00 00 00 00 00 00 00 123567..........
BEA71EF8C8 06 00 00 00 00 00 00 00 0F 00 00 00 00 00 00 00 ................
BEA71EF8D8 6F 11 B1 FF 03 00 48 F9 31 32 33 35 36 37 00 00 o.   ..H 123567..
BEA71EF8E8 00 00 00 00 00 00 00 00 06 00 00 00 00 00 00 00 ................
BEA71EF8F8 0F 00 00 00 00 00 00 00 B8 F8 1E A7 BE 00 00 00 ........   .   ...
BEA71EF908 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
BEA71EF918 00 00 00 00 00 00 00 00 0F 00 00 00 00 00 00 00 ................
BEA71EF928 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
BEA71EF938 00 00 00 00 00 00 00 00 0F 00 00 00 00 00 00 00 ................
BEA71EF948 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
BEA71EF958 00 00 00 00 00 00 00 00 0F 00 00 00 00 00 00 00 ................
BEA71EF968 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
BEA71EF978 00 00 00 00 00 00 00 00 0F 00 00 00 00 00 00 00 ................
BEA71EF988 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
BEA71EF998 00 00 00 00 00 00 00 00 0F 00 00 00 00 00 00 00 ................
BEA71EF9A8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
BEA71EF9B8 00 00 00 00 00 00 00 00 0F 00 00 00 00 00 00 00 ................
```

Рисунок 4.7 – Вміст оперативної пам’яті після створення шифроконтейнеру

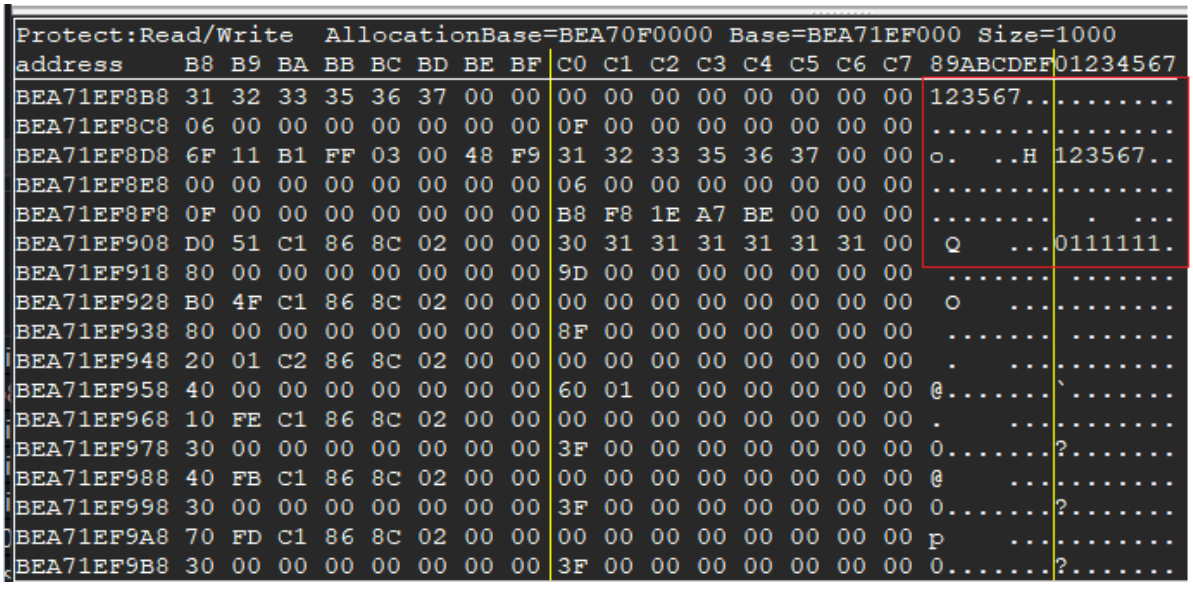
На даному зображені видно, що шифроконтейнер був вдало створений, а дані для захисту вдало в нього скопійовані. На рисунку 4.8 показано як змінюється стан оперативної пам’яті після увімкнення захисту.



```
Protect:Read/Write  AllocationBase=BEA70F0000 Base=BEA71EF000 Size=1000
address     B8 B9 BA BB BC BD BE BF C0 C1 C2 C3 C4 C5 C6 C7 89ABCDEF01234567
BEA71EF8B8 30 30 30 30 30 30 00 00 00 00 00 00 00 00 00 00 000000..........
BEA71EF8C8 06 00 00 00 00 00 00 00 0F 00 00 00 00 00 00 00 ................
BEA71EF8D8 6F 11 B1 FF 03 00 48 F9 70 57 C1 86 8C 02 00 00 o.  ..H pW    ...
BEA71EF8E8 00 00 00 00 00 00 00 00 00 01 00 00 00 00 00 00 ................
BEA71EF8F8 41 01 00 00 00 00 00 00 B8 F8 1E A7 BE 00 00 00 A...........  ...
BEA71EF908 D0 51 C1 86 8C 02 00 00 30 31 31 31 31 31 31 00  Q   ...0111111.
BEA71EF918 80 00 00 00 00 00 00 00 9D 00 00 00 00 00 00 00  ........ ........
BEA71EF928 B0 4F C1 86 8C 02 00 00 00 00 00 00 00 00 00 00  O   ............
BEA71EF938 80 00 00 00 00 00 00 00 8F 00 00 00 00 00 00 00  ........ ........
BEA71EF948 20 01 C2 86 8C 02 00 00 00 00 00 00 00 00 00 00  .   ............
BEA71EF958 00 01 00 00 00 00 00 00 60 01 00 00 00 00 00 00 ........`.......
BEA71EF968 10 FE C1 86 8C 02 00 00 00 00 00 00 00 00 00 00 .   ............
BEA71EF978 30 00 00 00 00 00 00 00 3F 00 00 00 00 00 00 00 0.......?.......
BEA71EF988 40 FB C1 86 8C 02 00 00 00 00 00 00 00 00 00 00 @   ............
BEA71EF998 30 00 00 00 00 00 00 00 3F 00 00 00 00 00 00 00 0.......?.......
BEA71EF9A8 70 FD C1 86 8C 02 00 00 00 00 00 00 00 00 00 00 p   ............
BEA71EF9B8 30 00 00 00 00 00 00 00 3F 00 00 00 00 00 00 00 0.......?.......
```

Рисунок 4.8 – Вміст оперативної пам’яті після увімкнення захисту

На даному зображені видно, що захист був вдало увімкнений, а дані за оригінальною адресою перезаписані, як і дані самого шифроконтейнеру. На рисунку 4.9 показано як змінюється стан оперативної пам’яті після вимкнення захисту.

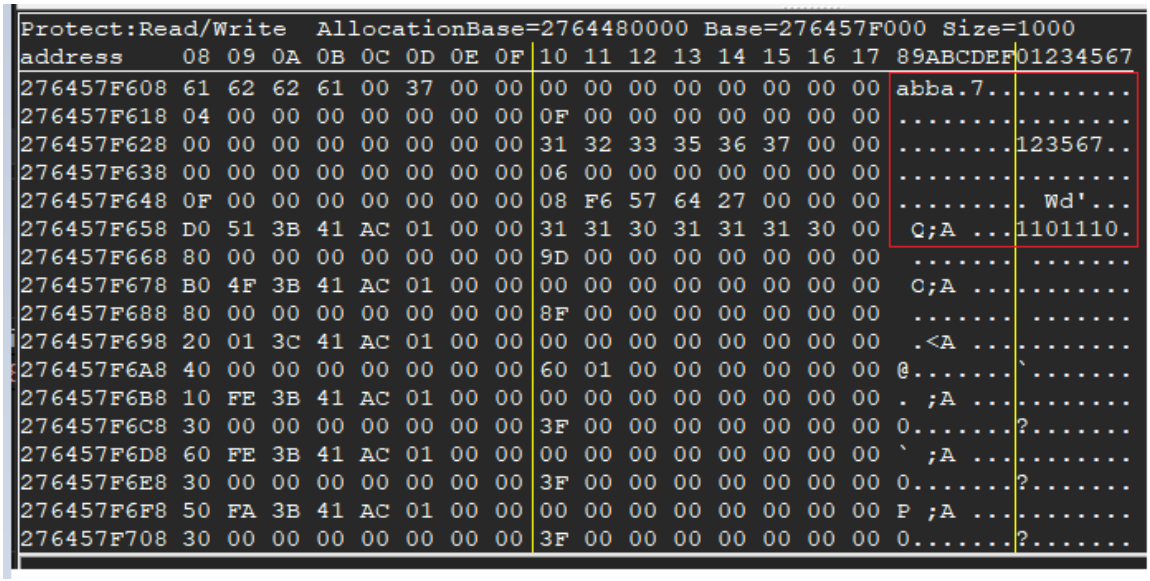


```
Protect:Read/Write  AllocationBase=BEA70F0000 Base=BEA71EF000 Size=1000
address     B8 B9 BA BB BC BD BE BF C0 C1 C2 C3 C4 C5 C6 C7 89ABCDEF01234567
BEA71EF8B8 31 32 33 35 36 37 00 00 00 00 00 00 00 00 00 00 123567..........
BEA71EF8C8 06 00 00 00 00 00 00 00 0F 00 00 00 00 00 00 00 ................
BEA71EF8D8 6F 11 B1 FF 03 00 48 F9 31 32 33 35 36 37 00 00 o.  ..H 123567..
BEA71EF8E8 00 00 00 00 00 00 00 00 06 00 00 00 00 00 00 00 ................
BEA71EF8F8 0F 00 00 00 00 00 00 00 B8 F8 1E A7 BE 00 00 00 ...........  ...
BEA71EF908 D0 51 C1 86 8C 02 00 00 30 31 31 31 31 31 31 00  Q   ...0111111.
BEA71EF918 80 00 00 00 00 00 00 00 9D 00 00 00 00 00 00 00  ........ ........
BEA71EF928 B0 4F C1 86 8C 02 00 00 00 00 00 00 00 00 00 00  O   ............
BEA71EF938 80 00 00 00 00 00 00 00 8F 00 00 00 00 00 00 00  ........ ........
BEA71EF948 20 01 C2 86 8C 02 00 00 00 00 00 00 00 00 00 00  .   ............
BEA71EF958 40 00 00 00 00 00 00 00 60 01 00 00 00 00 00 00 @.......`.......
BEA71EF968 10 FE C1 86 8C 02 00 00 00 00 00 00 00 00 00 00 .   ............
BEA71EF978 30 00 00 00 00 00 00 00 3F 00 00 00 00 00 00 00 0.......?.......
BEA71EF988 40 FB C1 86 8C 02 00 00 00 00 00 00 00 00 00 00 @   ............
BEA71EF998 30 00 00 00 00 00 00 00 3F 00 00 00 00 00 00 00 0.......?.......
BEA71EF9A8 70 FD C1 86 8C 02 00 00 00 00 00 00 00 00 00 00 p   ............
BEA71EF9B8 30 00 00 00 00 00 00 00 3F 00 00 00 00 00 00 00 0.......?.......
```

Рисунок 4.9 – Вміст оперативної пам’яті після вимкнення захисту

На даному зображені видно, що захист був вдало вимкнений, а дані за оригінальною адресою повернули свою початкову форму, дані самого

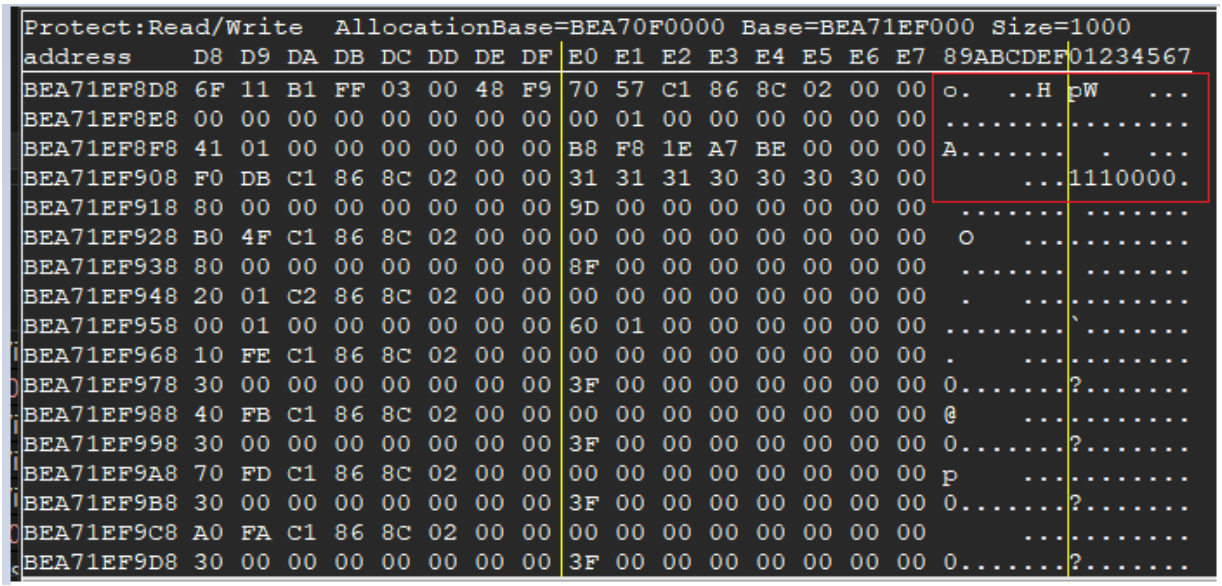
шифроконтейнеру не змінилися із зашифрованих. На рисунку 4.10 показано як змінюється стан оперативної пам’яті після того як користувач змінює значення змінної.



```
Protect:Read/Write  AllocationBase=2764480000 Base=276457F000 Size=1000
address    08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 89ABCDEF01234567
276457F608 61 62 62 61 00 37 00 00 00 00 00 00 00 00 00 00 abba.7..........
276457F618 04 00 00 00 00 00 00 00 0F 00 00 00 00 00 00 00 ................
276457F628 00 00 00 00 00 00 00 00 31 32 33 35 36 37 00 00 ........123567..
276457F638 00 00 00 00 00 00 00 00 06 00 00 00 00 00 00 00 ................
276457F648 0F 00 00 00 00 00 00 00 08 F6 57 64 27 00 00 00 ..........Wd'...
276457F658 D0 51 3B 41 AC 01 00 00 31 31 30 31 31 31 30 00  Q;A ...1101110.
276457F668 80 00 00 00 00 00 00 00 9D 00 00 00 00 00 00 00  ....... .......
276457F678 B0 4F 3B 41 AC 01 00 00 00 00 00 00 00 00 00 00  O;A ............
276457F688 80 00 00 00 00 00 00 00 8F 00 00 00 00 00 00 00  ....... .......
276457F698 20 01 3C 41 AC 01 00 00 00 00 00 00 00 00 00 00  .<A ............
276457F6A8 40 00 00 00 00 00 00 00 60 01 00 00 00 00 00 00 @.......`.......
276457F6B8 10 FE 3B 41 AC 01 00 00 00 00 00 00 00 00 00 00 . ;A ............
276457F6C8 30 00 00 00 00 00 00 00 3F 00 00 00 00 00 00 00 0.......?.......
276457F6D8 60 FE 3B 41 AC 01 00 00 00 00 00 00 00 00 00 00 ` ;A ............
276457F6E8 30 00 00 00 00 00 00 00 3F 00 00 00 00 00 00 00 0.......?.......
276457F6F8 50 FA 3B 41 AC 01 00 00 00 00 00 00 00 00 00 00 P ;A ............
276457F708 30 00 00 00 00 00 00 00 3F 00 00 00 00 00 00 00 0.......?.......
```

Рисунок 4.10 – Вміст оперативної пам’яті після зміни значення змінної

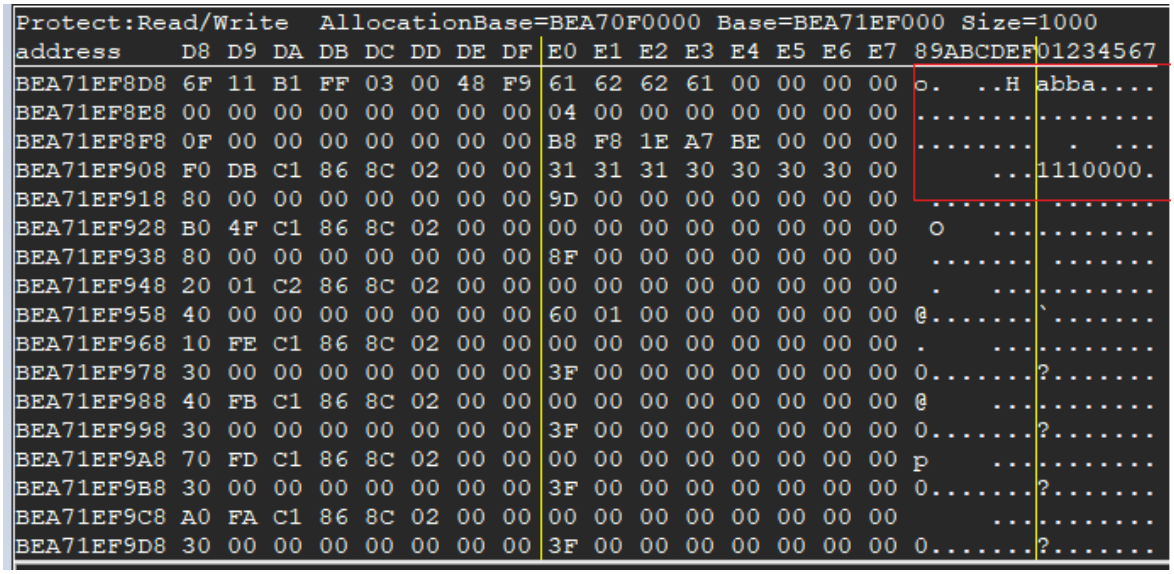
На даному зображені видно, що вміст змінної був успішно перезаписаний. На рисунку 4.11 показано як змінюється стан оперативної пам’яті після повторного увімкнення захисту.



```
Protect:Read/Write  AllocationBase=BEA70F0000 Base=BEA71EF000 Size=1000
address    D8 D9 DA DB DC DD DE DF E0 E1 E2 E3 E4 E5 E6 E7 89ABCDEF01234567
BEA71EF8D8 6F 11 B1 FF 03 00 48 F9 70 57 C1 86 8C 02 00 00 o.  ..H pW   ...
BEA71EF8E8 00 00 00 00 00 00 00 00 00 01 00 00 00 00 00 00 ................
BEA71EF8F8 41 01 00 00 00 00 00 00 B8 F8 1E A7 BE 00 00 00 A.......  .   ...
BEA71EF908 F0 DB C1 86 8C 02 00 00 31 31 31 30 30 30 30 00      ...1110000.
BEA71EF918 80 00 00 00 00 00 00 00 9D 00 00 00 00 00 00 00  ....... .......
BEA71EF928 B0 4F C1 86 8C 02 00 00 00 00 00 00 00 00 00 00  O   ............
BEA71EF938 80 00 00 00 00 00 00 00 8F 00 00 00 00 00 00 00  ....... .......
BEA71EF948 20 01 C2 86 8C 02 00 00 00 00 00 00 00 00 00 00  .   ............
BEA71EF958 00 01 00 00 00 00 00 00 60 01 00 00 00 00 00 00 ........`.......
BEA71EF968 10 FE C1 86 8C 02 00 00 00 00 00 00 00 00 00 00 .   ............
BEA71EF978 30 00 00 00 00 00 00 00 3F 00 00 00 00 00 00 00 0.......?.......
BEA71EF988 40 FB C1 86 8C 02 00 00 00 00 00 00 00 00 00 00 @   ............
BEA71EF998 30 00 00 00 00 00 00 00 3F 00 00 00 00 00 00 00 0.......?.......
BEA71EF9A8 70 FD C1 86 8C 02 00 00 00 00 00 00 00 00 00 00 p   ............
BEA71EF9B8 30 00 00 00 00 00 00 00 3F 00 00 00 00 00 00 00 0.......?.......
BEA71EF9C8 A0 FA C1 86 8C 02 00 00 00 00 00 00 00 00 00 00      ............
BEA71EF9D8 30 00 00 00 00 00 00 00 3F 00 00 00 00 00 00 00 0.......?.......
```

Рисунок 4.11 – Вміст оперативної пам’яті після повторного увімкнення захисту

На даному зображені видно, що захист був вдало увімкнений, а дані за оригінальною адресою перезаписані, як і дані самого шифроконтейнеру. На рисунку 4.12 показано як змінюється стан оперативної пам’яті після вимкнення захисту.



```
Protect:Read/Write  AllocationBase=BEA70F0000 Base=BEA71EF000 Size=1000
address     D8 D9 DA DB DC DD DE DF E0 E1 E2 E3 E4 E5 E6 E7 89ABCDEF01234567
BEA71EF8D8 6F 11 B1 FF 03 00 48 F9 61 62 62 61 00 00 00 00 o.  ..H abba....
BEA71EF8E8 00 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00 ........ ........
BEA71EF8F8 0F 00 00 00 00 00 00 00 B8 F8 1E A7 BE 00 00 00 ........ .  ...
BEA71EF908 F0 DB C1 86 8C 02 00 00 31 31 31 30 30 30 30 00      ...1110000.
BEA71EF918 80 00 00 00 00 00 00 00 9D 00 00 00 00 00 00 00  ....... ........
BEA71EF928 B0 4F C1 86 8C 02 00 00 00 00 00 00 00 00 00 00  O   ... ........
BEA71EF938 80 00 00 00 00 00 00 00 8F 00 00 00 00 00 00 00  ....... ........
BEA71EF948 20 01 C2 86 8C 02 00 00 00 00 00 00 00 00 00 00  .   ... ........
BEA71EF958 40 00 00 00 00 00 00 00 60 01 00 00 00 00 00 00 @.......`.......
BEA71EF968 10 FE C1 86 8C 02 00 00 00 00 00 00 00 00 00 00 .    ... ........
BEA71EF978 30 00 00 00 00 00 00 00 3F 00 00 00 00 00 00 00 0.......?.......
BEA71EF988 40 FB C1 86 8C 02 00 00 00 00 00 00 00 00 00 00 @    ... ........
BEA71EF998 30 00 00 00 00 00 00 00 3F 00 00 00 00 00 00 00 0.......?.......
BEA71EF9A8 70 FD C1 86 8C 02 00 00 00 00 00 00 00 00 00 00 p    ... ........
BEA71EF9B8 30 00 00 00 00 00 00 00 3F 00 00 00 00 00 00 00 0.......?.......
BEA71EF9C8 A0 FA C1 86 8C 02 00 00 00 00 00 00 00 00 00 00     ... ........
BEA71EF9D8 30 00 00 00 00 00 00 00 3F 00 00 00 00 00 00 00 0.......?.......
```

Рисунок 4.12 – Вміст оперативної пам’яті після повторного увімкнення захисту

На даному зображені видно, що тестування пройшло успішно і після повторного зняття захисту користувач може продовжувати користуватися змінною.

4.5 Тестування продуктивності

Метою тестування продуктивності є перевірка наскільки добре система справляється з граничним навантаженням на яке вона розрахована. В рамках постановки задачі було визначено, що програмна бібліотека повинна вміти захищати об’єкти розміром до 4 ГБ. В результаті тестування було визначено, що хоча система може захищати об’єкти такого розміру, проте час для активації та зняття захисту не є допустим для кінцевого користувача, оскільки може тривати декілька годин. Основною проблемою в даному випадку є не сам алгоритм, а швидкість обчислення на сучасних процесорах. Тому рекомендовано використовувати дану бібліотеку з даними, що мають розмір не більше 1МБ.

Результати даного тестування зображені на рисунку 4.13.

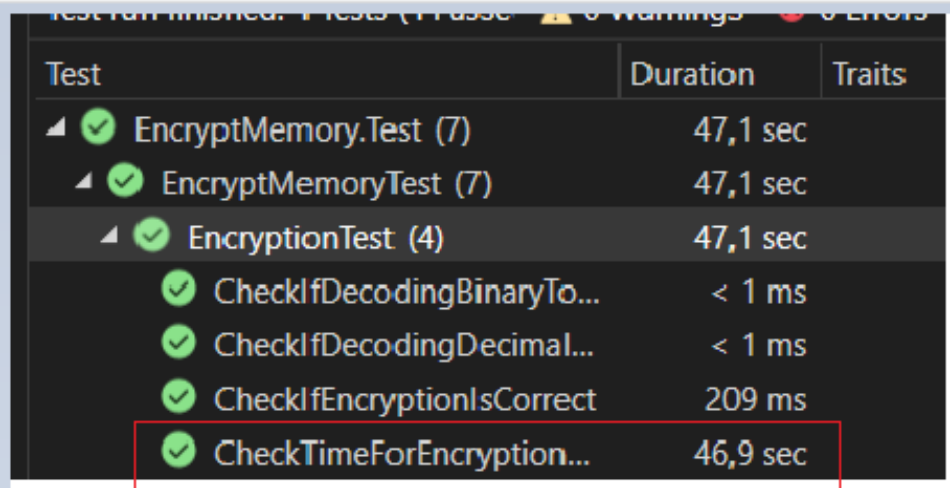


Рисунок 4.13 – Результат тестування продуктивності

Навіть дані такого невеликого розміру шифруються близько 50-ти секунд, приблизно такий самий час іде на розшифрування, тому дану бібліотеку можна покращити використовуючи більш потужні процесор, розпаралелення обчислювальних потоків чи використовувати графічний процесор для обчислень даних такого розміру.

4.6 Висновки з розділу

Блокове тестування дозволило протестувати працездатність кожного модуля програмної бібліотеки окремо, що зменшило кількість помилок в програмному забезпеченні та утворення виключних ситуацій, що були спричинені неправильною реалізацією модулів. Здійснення статичного тестування вихідного коду бібліотеки дозволило провести перевірку коректності, захищеності та безпеки виконуваного коду, що забезпечило стабільне виконання функціональності бібліотеки. Проведення інтеграційного тестування дозволило переконатися у працездатності всієї бібліотеки та коректності взаємодії всіх її елементів. Здійснення тестування продуктивності дозволило перевірити програмну бібліотеку на продуктивність під час захисту даних великого розміру та оцінити рівень ефективності використовуваних системних ресурсів впродовж захисту.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Метою проведення комерційного і технологічного аудиту дослідження за темою «Метод та програмна бібліотека захисту програм від дампінгу» є покращення захисту програм від дампінгу, шляхом створення зашифрованих контейнерів, що містять захищувані дані.

Для проведення комерційного та технологічного аудиту було залучено 3 незалежні експерти.

Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендовано здійснювати із застосуванням 5-ти бальної системи оцінювання за 12-ма критеріями, що наведені в таблиці 5.1 [30]. Результати оцінки було зведено до таблиці 5.2.

Таблиця 5.1 – Рекомендовані критерії оцінювання науково-технічного рівня і комерційного потенціалу розробки та бальна оцінка

Бали (за п’ятибальною шкалою)					
	0	1	2	3	4
1	2	3	4	5	6
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
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
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижча за ціни аналогів	Ціна продукту значно нижча 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

Продовження таблиці 5.1

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
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 незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються 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 більший 5- ти років	Термін реалізації ідеї менший 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менший 3-х років. Термін окупності інвестицій менший 3-х років

Продовження таблиці 5.1

12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що потребує значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту потребує незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Таблиця 5.2 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	Експерт (ПІБ, посада)		
	Олександр Вентичук Developer. ТОВ «БРОКСТАР»	Влад Куран Middle .NET Developer. ТОВ «БРОКСТАР»	Володимир Тарасюк QA Директор ТОВ «БРОКСТАР»
	Бали, виставлені експертами:		
1. Технічна здійсненність концепції	4	3	3
2. Ринкові переваги (наявність аналогів)	3	5	2
3. Ринкові переваги (ціна продукту)	4	4	5
4. Ринкові переваги (технічні властивості)	2	2	1
5. Ринкові переваги (експлуатаційні витрати)	3	2	4
6. Ринкові перспективи (розмір ринку)	4	4	5
7. Ринкові перспективи (конкуренція)	3	4	2
8. Практична здійсненність (наявність фахівців)	4	5	4
9. Практична здійсненність (наявність фінансів)	2	4	5
10. Практична здійсненність (необхідність нових матеріалів)	4	4	4
11. Практична здійсненність (термін реалізації)	3	3	3
12. Практична здійсненність (розробка документів)	4	4	5
Сума балів	$СБ_1=40$	$СБ_2=44$	$СБ_3=43$

Середньоарифметична сума балів буде рівна (40+44+43)/3 = 42.3

За результатами розрахунків, наведених в таблиці 5.2 та використовуючи дані з таблиці 5.3 [30], було зроблено висновок про те, що науково-технічний рівень і рівень комерційного потенціалу розробки - високий.

Таблиця 5.3 – Науково-технічні рівні та комерційні потенціали розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	Науково-технічний рівень та комерційний потенціал розробки
41…48	Високий
31…40	Вищий середнього
21…30	Середній
11…20	Нижчий середнього
0…10	Низький

5.2 Розрахунок витрат на проведення науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи на тему «Метод та програмна бібліотека захисту програм від дампінгу», під час планування, обліку і калькулювання собівартості науково-дослідної роботи було згруповано за відповідними статтями.

До статті «Витрати на оплату праці» належать витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп, науковим, інженерно-технічним працівникам, конструкторам, технологам, креслярам, копіювальникам, лаборантам, робітникам, студентам, аспірантам та іншим працівникам, безпосередньо зайнятим виконанням конкретної теми, обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з чинними в організаціях системами оплати праці.

Основна заробітна плата дослідників

Витрати на основну заробітну плату дослідників ($З_о$) розраховано у відповідності до посадових окладів працівників, за формулою [30]:

$$З_о = \sum_{i=1}^{k} \frac{M_{ni} \cdot t_i}{T_p}, \qquad (5.1)$$

де k – кількість посад дослідників залучених до процесу досліджень;

M_{ni} – місячний посадовий оклад конкретного дослідника, грн;

t_i – число днів роботи конкретного дослідника, дн.;

T_p – середнє число робочих днів в місяці, T_p=21 дні.

$$З_o = 25000 \cdot 21 / 21 = 25000 \text{ грн.}$$

Проведені розрахунки зведемо до таблиці.

Таблиця 5.4 – Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн
Керівник проекту	25000	1190,5	21	25000
Програміст	18000	857,1	42	36000
Всього				61000

Основна заробітна плата робітників

Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт НДР на тему «Метод та програмна бібліотека захисту програм від дампінгу» розраховано за формулою:

$$З_p = \sum_{i=1}^{n} C_i \cdot t_i , \quad (5.2)$$

де C_i – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i – час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду C_i можна визначити за формулою:

$$C_i = \frac{M_M \cdot K_i \cdot K_c}{T_p \cdot t_{зм}} , \quad (5.3)$$

де M_M – розмір прожиткового мінімуму працездатної особи, або мінімальної місячної заробітної плати (в залежності від діючого законодавства), приймемо $Мм$=6700 грн;

K_i – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду (табл. Б.2, додаток Б) [30];

K_c – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об'єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати.

T_p – середнє число робочих днів в місяці, приблизно T_p = 21 дн;

$t_{зм}$ – тривалість зміни, год.

$C_1 = 6700{,}00 \cdot 1 \cdot 1{,}65 / (21 \cdot 8) = 65{,}8$ грн.
$З_{р1} = 63{,}8 \cdot 1 = 65{,}8$ грн.

Таблиця 5.5 – Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Погодинна тарифна ставка, грн	Величина оплати на робітника грн
1.Підготовчі	16	1	65,8	1052,8
2.Створення архітектури	40	3	88,8	3553,4
3.Вдосконалення алгоритму	40	5	111,9	4474,6
4.Розробка застосунку	40	2	72,4	2895,4
5.Тестування	16	4	59,8	957,1
Всього				12933,5

Додаткова заробітна плата дослідників та робітників

Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \quad (5.4)$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати. Приймемо 11%.

$З_{дод}$ = (61000 + 12933,5) · 11 / 100% = 8132,68 грн.

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%}, \quad (5.5)$$

де $Н_{зп}$ – норма нарахування на заробітну плату. Приймаємо 22%.

$З_н$ = (61000 + 12933,5 + 8132,68) · 22 / 100% = 18054,56 грн.

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби і предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень за темою «Метод та програмна бібліотека захисту програм від дампінгу».

Витрати на матеріали ($М$), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою:

$$M = \sum_{j=1}^{n} Н_j \cdot Ц_j \cdot К_j - \sum_{j=1}^{n} В_j \cdot Ц_{вj}, \quad (5.6)$$

де $Н_j$ – норма витрат матеріалу j-го найменування, кг;

n – кількість видів матеріалів;

$Ц_j$ – вартість матеріалу j-го найменування, грн/кг;

$К_j$ – коефіцієнт транспортних витрат, ($К_j$ = *1,1 ... 1,15*);

$В_j$ – маса відходів j-го найменування, кг;

$Ц_{вj}$ – вартість відходів j-го найменування, грн/кг.

Проведені розрахунки зведемо до таблиці.

Таблиця 5.6 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за 1 кг, грн	Норма витрат, кг	Вартість витраченого матеріалу, грн
Папір канцелярський офісний (А4)	175	1	175
Папір для заміток (А5)	115	1	115
Ручка	10	1	10
Флешка	250	1	250
Всього			550
З врахуванням коефіцієнта транспортування			632,5

Витрати на комплектуючі *($К_в$)*, які використовують при проведенні НДР на тему «Метод та програмна бібліотека захисту програм від дампінгу».

Витрати на комплектуючі К, що були використані під час виконання даного етапу роботи, розраховуються за формулою:

$$К = \sum_{1}^{n} Н_i \cdot Ц_i \cdot К_i \text{ грн.}, \quad (5.7)$$

де $Н_i$ – кількість комплектуючих *i*-го виду, шт.;

$Ц_i$ – ціна комплектуючих *i*-го виду, грн.;

$К_i$ – коефіцієнт транспортних витрат, $К_i = (1{,}1 \ldots 1{,}15)$;

n – кількість видів комплектуючих.

Зроблені розрахунки бажано звести до таблиці:

Таблиця 5.7 – Витрати на комлектуючі

Найменування комплектувальних	Кількість	Ціна за штуку, грн.	Сума, грн.
Intel Core i9-13900KF 3.0 - 5.8GHz Turbo, 64 ГБ оперативної пам'яті, 1 ТБ SSD	2	120000	240000
Мереже обладнання	1	10000	10000
Visual Studio 2022	2	1850	3700
SSL-сертифікат	1	2000	2000
DataStorage	2	60000	120000
Всього з врахування коефіцієнт транспортних витрат			432055

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, розраховуємо з використанням прямолінійного методу амортизації за формулою:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12}, \quad (5.8)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$T_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$A_{обл} = (75000 \cdot 1) / (2 \cdot 12) = 6250$ *грн.*

Проведені розрахунки зведемо до таблиці.

Таблиця 5.8 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Ноутбук Acer Predator Helios 300	75000	2	2	6250
Робоче місце розробника ПЗ	150000	20	2	1250
Всього				7500

Витрати на силову електроенергію ($B_е$) розраховуємо за формулою:

$$B_е = \sum_{i=1}^{n} \frac{W_{yi} \cdot t_i \cdot Ц_е \cdot K_{вni}}{\eta_i}, \quad (5.9)$$

де W_{yi} – встановлена потужність обладнання на визначеному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_е$ – вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), приймемо $Ц_е$ = 7,5 грн;

$K_{вni}$ – коефіцієнт, що враховує використання потужності, $K_{вni} < 1$;

η_i – коефіцієнт корисної дії обладнання, $\eta_i < 1$.

$B_е = 0,25 \cdot 310,0 \cdot 7,5 \cdot 0,5 / 0,8 = 363,28$ грн.

До статті «Службові відрядження» дослідної роботи на тему «Метод та програмна бібліотека захисту програм від дампінгу» належать витрати на відрядження штатних працівників, працівників організацій, які працюють за договорами цивільно-правового характеру, аспірантів, зайнятих розробленням

досліджень, відрядження, пов’язані з проведенням випробувань машин та приладів, а також витрати на відрядження на наукові з’їзди, конференції, наради, пов’язані з виконанням конкретних досліджень.

Витрати за статтею «Службові відрядження» розраховуємо як 20…25% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{св} = (З_о + З_р) \cdot \frac{Н_{св}}{100\%}, \quad (5.10)$$

де $Н_{св}$ – норма нарахування за статтею «Службові відрядження», приймемо $Н_{св}$ = 20%.

$В_{св}$ = (61000 + 12933,5) · 25 / 100% = 18483,37 грн.

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками.

Витрати за статтею «Інші витрати» розраховуємо як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$І_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (5.11)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати», приймемо $Н_{ів}$ = 50%.

$І_в$ = (61000 + 12933,5) · 75 / 100% = 55450,12 грн.

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов’язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов’язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов’язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін.

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуємо як 100…150% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (5.12)$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати», приймемо $Н_{нзв}$ = 100%.

$В_{нзв}$ = (61000 + 12933,5) · 100 / 100% = 73933,5 грн.

Витрати на проведення науково-дослідної роботи на тему «Метод та програмна бібліотека захисту програм від дампінгу» розраховуємо як суму всіх попередніх статей витрат за формулою:

$$B_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + B_{спец} + B_{прг} + А_{обл} + B_е + B_{св} + B_{сп} + І_в + B_{нзв}. \quad (5.13)$$

$В_{заг}$=61000+12933,5+8132,68+18054,56+632,5+432055+7500+363,28+18483,37 +55450,12+73933,5 = 688538,5 грн.

Загальні витрати *ЗВ* на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховується за формулою:

$$ЗВ = \frac{B_{заг}}{\eta}, \quad (5.14)$$

де η - коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи, приймемо η=0,7.

ЗВ = 688538,5 / 0,7 = 983626,44 грн.

В ринкових умовах узагальнюючим позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку.

Результати дослідження проведені за темою «Метод та програмна бібліотека захисту програм від дампінгу» передбачають комерціалізацію протягом 3-х років реалізації на ринку.

В цьому випадку основу майбутнього економічного ефекту будуть формувати:

ΔN – збільшення кількості споживачів продукту, в аналізовані періоди часу, від покращення його певних характеристик;

N – кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки, приймемо 1 особа

$Ц_б$ – вартість програмного продукту у році до впровадження результатів розробки, приймемо 10000,00 грн;

$\pm\Delta Ц_о$ – зміна вартості програмного продукту від впровадження результатів науково-технічної розробки в аналізовані періоди часу, приймемо зростання на 1000,00 грн.

Можливе збільшення чистого прибутку у потенційного інвестора $\Delta П_i$ для кожного із 3-х років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховуємо за формулою [30]:

$$\Delta П_i = (\pm\Delta Ц_о \cdot N + Ц_о \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \qquad (5.15)$$

де λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2021 році ставка податку на додану вартість складає 20%, а коефіцієнт $\lambda = 0{,}8333$;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту). Приймемо ρ =30%;

ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2023 році $\vartheta = 18\%$;

Збільшення чистого прибутку 1-го року:

$\Delta П_1 = (1 \cdot 1000 + 10000 \cdot 800) \cdot 0{,}83 \cdot 0{,}3 \cdot (1\text{-}0{,}18/100\%) = 1988662{,}95$ грн.

Збільшення чистого прибутку 2-го року:

$\Delta П_2 = (1 \cdot 1000 + 10000 \cdot (800 + 1000)) \cdot 0{,}83 \cdot 0{,}3 \cdot (1 - 0{,}18/100\%) = 4474932{,}4$ грн.

Збільшення чистого прибутку 3-го року:

$\Delta П_3 = (1 \cdot 1000 + 10000 \cdot (800 + 1000 + 900)) \cdot 0{,}83 \cdot 0{,}3 \cdot (1 - 0{,}18/100\%) = 6711898{,}6$ грн.

Приведена вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \tag{5.16}$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

T – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ =18%;

t – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$ПП = 1988662{,}95 / (1+0{,}18)^1 + 4474932{,}4 / (1+0{,}18)^2 + 6711898{,}6 / (1+0{,}18)^3 =$

$= 8984203$ грн.

Величина початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки:

$$PV = k_{інв} \cdot ЗВ, \tag{5.17}$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, приймаємо $k_{інв}$ =2,5;

ЗВ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, приймаємо 983626,44 грн.

$PV = k_{інв} \cdot ЗВ = 2{,}5 \cdot 983626{,}44 = 2459066{,}1$ грн.

Абсолютний економічний ефект $E_{абс}$ для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV, \quad (5.18)$$

де $ПП$ – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, 8984203 грн;

PV – теперішня вартість початкових інвестицій, 2637313,13 грн.

$E_{абс} = ПП - PV$ = 8984203 - 2459066,1 = 6525136,9 грн.

Внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$E_в = \sqrt[Тж]{1 + \frac{E_{абс}}{PV}} - 1, \quad (5.19)$$

де $E_{абс}$ – абсолютний економічний ефект вкладених інвестицій, грн;

PV – теперішня вартість початкових інвестицій, грн;

$T_ж$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, 3 роки.

$$E_в = \sqrt[Тж]{1 + \frac{E_{абс}}{PV}} - 1 = (1+6525136,9 /2459066,1)^{1/3} - 1 = 0,54.$$

Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{мін}$:

$$\tau_{мін} = d + f, \quad (5.20)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2023 році в Україні d =0,1;

f – показник, що характеризує ризикованість вкладення інвестицій, приймемо 0,2.

$\tau_{мін}$ = 0,1+0,2 = 0,3 < 0,54 свідчить про те, що внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки вища мінімальної внутрішньої дохідності. Тобто інвестувати в науково-дослідну роботу за темою «Метод та програмна бібліотека захисту програм від дампінгу» доцільно.

Період окупності інвестицій $T_{ок}$ які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$T_{ок} = \frac{1}{E_в}, \tag{5.21}$$

де $E_в$ – внутрішня економічна дохідність вкладених інвестицій.

$T_{ок}$ = 1 / 0,54 = 1,8 р.

$T_{ок}$ < 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

5.3 Висновки з розділу

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Метод та програмна бібліотека захисту програм від дампінгу» становить 42,3 бали, що, свідчить про комерційну важливість проведення даних досліджень оскільки рівень комерційного потенціалу розробки високий. Також термін окупності становить 1,8 роки, що менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Отже можна зробити висновок про доцільність проведення науково-дослідної роботи за темою «Метод та програмна бібліотека захисту програм від дампінгу».

ВИСНОВКИ

В даній магістерській дипломній роботі, відповідно до індивідуального завдання, розроблено програмну бібліотеку для захисту від дампінгу шляхом створення зашифрованих контейнерів в оперативній пам’яті робочої станції.

Необхідність захисту від дампінгу обумовила актуальність даної магістерської дипломної роботи. Визначено актуальність, мету та задачі, об’єкт та предмет дослідження, новизну отриманих результатів.

Здійснено аналіз загроз до яких можуть бути вразливі застосунки, а також детально описано та проаналізовано атаки, які мають дампінг, як один з кроків реалізації атаки. На основі проаналізованих даних було описано та проаналізовано методи захисту від дампінгу, що можуть бути використанні для перешкоджанню реалізації атак. Також було проаналізовано блокові шифри, які можуть бути використані в процесі захисту від дампінгу для захисту від дампінгу та розробки даної магістерської дипломної роботи. Після проведення аналізу було виконано постановку задачі для магістерської кваліфікаційної роботи.

Здійснено математичний опис процесу дампінгу, який дозволив сформувати математичну модель дампінгу та визначити вхідні та вихідні дані, внутрішні стани, а також перетворення, які допомагають досягти кінцевої мети. Узагальнений опис методу дозволив виділити конкретні кроки, які дозволили б досягнути стану захищеності. Опис методу шифрування даних дозволив уточнити деталі реалізації шифру та визначити приорітети при його побудові. Опис методу блокування переривань копіювання оперативної пам’яті дозволив описати додатковий рівень захисту від дампінгу та визначити сильні та слабкі сторони використовуваного методу.

Розроблено узагальнені архітектуру та алгоритми як програмного засобу, так і усіх процедур, що будуть виконані в процесі захисту. Реалізація кожного з них дозволила сформувати модульну структуру засобу та розділити програмний засіб та процедури на складові, що дозволило значно спростити проектування та програмну реалізацію засобу.

Здійснено програмну реалізацію та тестування програмної бібліотеки для захисту від дампінгу. Постановка задачі допомогла визначити використовувані мову програмування та засоби для тестування. Блокове тестування дозволило протестувати працездатність кожного модуля програмної бібліотеки окремо, що зменшило кількість помилок в програмному забезпеченні та утворення виключних ситуацій, що були спричинені неправильною реалізацією модулів. Здійснення статичного тестування вихідного коду бібліотеки дозволило провести перевірку коректності, захищеності та безпеки виконуваного коду, що забезпечило стабільне виконання функціональності бібліотеки. Проведення інтеграційного тестування дозволило переконатися у працездатності всієї бібліотеки та коректності взаємодії всіх її елементів. Здійснення тестування продуктивності дозволило перевірити програмну бібліотеку на продуктивність під час захисту даних великого розміру та оцінити рівень ефективності використовуваних системних ресурсів впродовж захисту.

Програмна бібліотека, що була створена в межах даної магістерської кваліфікаційної роботи може бути використана в освітньому процесі вищих навчальних закладів та на підприємствах, які розробляють програмні продукти, при розробці програм, які потребують захисту від дампінгу. Подальшим розвитком програмного засобу може стати покращення його масштабованості, а саме розробка бібліотек на основі запропонованого методу, які підтримуватимуть різні мови, фреймворки та обчислювальні платформи.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Потужні DDoS-атаки та інші кіберзагрози кінця 2023 року: до чого готуватися бізнесу та як від них захищатися. URL : https://speka.media/ponad-2500-potuznix-ddos-atak-ta-novi-prognozi-kiberekspertiv-do-cogo-gotuvatisya-biznesu-pndqwv (дата звернення 20.10.2023).

2. Програми-вимагачі: як зловмисники зламують корпоративні мережі. URL : https://channel4it.com/publications/programi-vimagach-yak-zlovmisniki-zlamuyut-korporativn-merezh.html (дата звернення 20.10.2023).

3. CircleCI incident report for January 4, 2023 security incident. URL : https://circleci.com/blog/jan-4-2023-incident-report/ (дата звернення 20.10.2023).

4. Паламарчук О. Алгоритм захисту від несанкціонованого копіювання. Науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії ВНТУ – 2022. 2 с. URL : https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2022/paper/view/15525/13112 (дата звернення 20.10.2023).

5. Паламарчук О. Метод захисту програм від дампінгу. Науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії ВНТУ – 2024. 2 с. URL : https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2024/paper/view/19735/16365 (дата звернення 17.12.2023).

6. Definftion of application program. URL : https://www.pcmag.com/encyclopedia/term/application-program (дата звернення 20.10.2023).

7. Горбенко А. А. Інформатика та комп'ютерна техніка. Електронний навчально-методичний посібник. Тема 3. Програмне забезпечення та його класифікація. URL : https://kppk.com.ua/ELLIB/ebook/Gorbenko/IKT/3/3.htm (дата звернення 20.10.2023).

8. Стадії циклу розробки ПЗ. URL : https://qalight.ua/baza-znaniy/stadiyi-tsiklu-rozrobki-pz/ (дата звернення 20.10.2023).

9. Каплун В.А., Дудатьєв А.В., Семеренко В.П. Захист програмного забезпечення. Частина 1 : навч. посіб. Вінниця : ВНТУ, 2005. 139 с.

10. Каплун В.А., Дмитришин О.В., Баришев Ю.В. Захист програмного забезпечення. Частина 2 : навч. посіб. Вінниця : ВНТУ, 2014. 105 с.

11. OWASP Top Ten. URL : https://owasp.org/www-project-top-ten/ (дата звернення 20.10.2023).

12. Memory dump. URL : https://www.techtarget.com/whatis/definition/memory-dump (дата звернення 20.10.2023).

13. Інструменти для комп'ютерної криміналістики (форензики). URL : https://hackyourmom.com/kibervijna/instrumenty-dlya-kompyuternoyi-kryminalistyky-forenzyky/ (дата звернення 20.10.2023).

14. Creating a memory dump from a specific process with Task Manager. URL : https://m-files.my.site.com/s/article/Creating-a-memory-dump-from-a-specific-process-with-Task-Manager (дата звернення 20.10.2023).

15. Отримання облікових даних Адміністратора або кілька способів роботи з Mimikatz. URL : https://hackyourmom.com/kibervijna/otrymannya-oblikovyh-danyh-administratora-abo-kilka-sposobiv-roboty-z-mimikatz/ (дата звернення 20.10.2023).

16. Як розпізнати крадіжку облікових даних та запобігти інциденту. URL : https://dou.ua/forums/topic/43308/ (дата звернення 20.10.2023).

17. New Kids On The Block: Understanding Cold Boot Attacks. URL : https://hackernoon.com/new-kids-on-the-block-understanding-cold-boot-attacks-vd2k34or (дата звернення 20.10.2023).

18. Знайдена чергова вразливість чипів Intel. URL : https://ko.com.ua/znajdena_chergova_vrazlivist_chipiv_intel_141805 (дата звернення 20.10.2023).

19. The latest security information on Intel® products. INTEL-SA-00657. URL: https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00657.html (дата звернення 20.10.2023).

20. Перехоплення системних функцій в середовищі Windows. URL : https://devzone.org.ua/post/perexoplennia-sistemnix-funkcii-v-seredovishhi-windows (дата звернення 20.10.2023).

21. Що таке шифрування та як воно працює? : URL : https://www.kingston.com/ua/blog/data-security/what-is-encryption (дата звернення 20.10.2023).

22. Глинчук Л.Я. Криптологія : навч. посіб. Луцьк : Вежа-Друк, 2014. 164 с.

23. Тарнавський Ю.А. Технології захисту інформації : навч. посіб. Київ : КПІ ім. Ігоря Сікорського, 2018. 162 с.

24. Лужецький, В. А. Остапенко, А. В. Аналіз алгоритмів симетричного блокового шифрування: Інформаційні технології та комп’ютерна інженерія №3. , 2012. 55-64 с.

25. Safeguarding Memory in Higher-Level Programming Languages? : URL : https://www.praetorian.com/blog/safeguarding-memory-in-higher-level-programming-languages/ (дата звернення 20.10.2023).

26. Raw Memory Dump/Access Outside Memory : URL : https://cplusplus.com/forum/beginner/28588/ (дата звернення 20.10.2023).

27. Looking for C++ object in a memory dump : URL : https://www.amongbytes.com/post/201201-how-find-c++-object-address-of-a-class-in-the-core-file/ (дата звернення 20.10.2023).

28. MinHook - The Minimalistic x86/x64 API Hooking Library : URL : https://www.codeproject.com/Articles/44326/MinHook-The-Minimalistic-x-x-API-Hooking-Libra (дата звернення 20.10.2023).

29. Visual Studio 2022 : URL : https://visualstudio.microsoft.com/downloads/ (дата звернення 20.10.2023).

30. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. – Вінниця : ВНТУ, 2021. – 42 с.

ДОДАТКИ

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації

УЗГОДЖЕНО

Директор ТОВ «БРОКСТАР»

_____ Володимир ТАРАСЮК

«13» грудня 2023 р.

ЗАТВЕРДЖУЮ

Завідувач кафедри ЗІ

д. т. н., проф.

_____ Володимир ЛУЖЕЦЬКИЙ

«11» 12 2023 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської кваліфікаційної роботи

на тему: "Метод та програмна бібліотека захисту програм від дампінгу"

08-20.МКР.020.00.000 ТЗ

Керівник магістерської кваліфікаційної роботи
к. т. н., доц.

_____ Юрій БАРИШЕВ

Вінниця 2023

1 Назва та область використання

Метод та програмна бібліотека захисту програм від дампінгу
Область використання: захист програмних застосунків від дампінгу
Розробка виконується на основі наказу ВНТУ №247 від 18.09.23.

2 Мета та призначення розробки

Покращення захисту програм від дампінгу.

3 Джерела розробки

3.1 Каплун В.А., Дмитришин О.В., Баришев Ю.В. Захист програмного забезпечення. Частина 2 : навч. посіб. Вінниця : ВНТУ, 2014. 105 с.
3.2 Глинчук Л.Я. Криптологія : навч. посіб. Луцьк : Вежа-Друк, 2014. 164 с.
3.3 Тарнавський Ю.А. Технології захисту інформації : навч. посіб. Київ : КПІ ім. Ігоря Сікорського, 2018. 162 с.
3.4 Лужецький, В. А. Остапенко, А. В. Аналіз алгоритмів симетричного блокового шифрування: Інформаційні технології та комп'ютерна інженерія №3. , 2012. 55-64 с.

4 Виконавці МКР

Студент групи 2БС-22м Паламарчук Олександр Русланович

5 Вимоги до програми

5.1 Вимоги до функціональних характеристик:
- бібліотека повинна підтримувати заміну модулів з однаковими сигнатурами;
- можливість компіляції на різних розрядностях операційної системи;
- можливість працювати з рядковими типами даних.

5.2 Вимоги до нефункціональних характеристик:
- можливість перехоплення фунцій різної розрядності;
- застосування симетричного шифрування з довжиною ключа не менше 128 біт та блоком 256 біт для захисту данних.

5.3 Вимоги до складу і параметрів технічних засобів:
- тактова частота процесора – від 2,5 ГГц;
- обсяг оперативної пам'яті – не менше 4 Гб.

5.4 Вимоги до інформаційної та програмної сумісності:
- програма вимагає наявність засобів для розробки програм мовою C++ на комп'ютері користувача (Visual Studio 2022, VS Code, XCode)

6 Стадії та етапи розробки

№	Зміст	Початок - закінчення	Результат
1	Аналіз завдання. Вступ	01.09.2023 - 10.09.2023	Чернетка вступу
2	Аналіз літературних джерел за напрямком магістерської кваліфікаційної роботи	10.09.2023 - 15.09.2023	Чернего першого розділу
3	Науково-технічне обґрунтування	16.09.2023 - 22.09.2023	Чернетка першого розділу
4	Розробка технічного завдання	23.09.2023 - 29.09.2023	Технічне завдання
5	Розробка рішень	30.09.2023 - 12.10.2023	Чернетка другого розділку
6	Практична реалізація, моделювання, експериментування, результати	14.10.2023 - 10.11.2023	Чернетка третього та четвертих розділів
7	Розробка розділу економічного обґрунтування доцільності розробки	11.11.2023 - 17.11.2023	Чернетка економічного розділу
8	Аналіз виконання ТЗ, висновки	18.11.2023 - 24.11.2023	Чернетка висновків
9	Оформлення пояснювальної записки	25.11.2023 - 30.11.2023	Пояснювальна записка

7 Порядок контролю та приймання

7.1 До приймання МКР представляється:

- пояснювальна записка до МКР;
- програма для реалізації захисту;
- результати функціонального тестування програми;
- ілюстративні матеріали для захисту.

7.2 Попередній захист на кафедрі: 28 листопада – 1 грудня 2023 р.

7.3 Представлення МКР до захисту: 11 – 14 грудня 2023р.

7.4 Захист МКР: 14 – 21 грудня 2023.

Розробив студент групи 2БС-22м _____ Олександр ПАЛАМАРЧУК

ДОДАТОК Б

Текст програми

Encryption.h:

```
#pragma once

#include <iostream>
#include <Windows.h>
#include <iostream>
#include <process.h>

using namespace std;

class Encryption {
private:
	string encryptionKey;
	string plaintext;

	string roundKeys[16];
	void generateKeys(string initialKey);
	void reverseRoundKeysSequence();
	int padding = 0;

	string shiftLeftOnce(string keyChunk);
	string shiftLeftTwice(string keyChunk);
	string xorStrings(string a, string b);

	string encryptText64(string pt);
	string decryptText64(string et);
public:
	void setEncryptionKey(string key);
	void setPlaintext(string text);

	string convertDecimalToBinary(int decimal);
	int convertBinaryToDecimal(string binary);

	string encryptText();
	string decryptText(string encryptText);
};
```

Encryption.cpp:

```
#include <iostream>
#include "Encryption.h"

using namespace std;

string Encryption::shiftLeftOnce(string keyChunk) {
    string shifted = "";
    for (int i = 1; i < 28; i++) {
        shifted += keyChunk[i];
    }
    shifted += keyChunk[0];
    return shifted;
}

string Encryption::shiftLeftTwice(string keyChunk) {
    string shifted = "";
```

```
    for (int i = 0; i < 2; i++) {
        for (int j = 1; j < 28; j++) {
            shifted += keyChunk[j];
        }
        shifted += keyChunk[0];
        keyChunk = shifted;
        shifted = "";
    }
    return keyChunk;
}

string Encryption::xorStrings(string a, string b)
{
    string result = "";
    int size = b.size();
    for (int i = 0; i < size; i++) {
        if (a[i] != b[i]) {
            result += "1";
        }
        else {
            result += "0";
        }
    }
    return result;
}

string Encryption::convertDecimalToBinary(int decimal)
{
    string binary;
    while (decimal != 0) {
        binary = (decimal % 2 == 0 ? "0" : "1") + binary;
        decimal = decimal / 2;
    }
    while (binary.length() < 4) {
        binary = "0" + binary;
    }
    return binary;
}

int Encryption::convertBinaryToDecimal(string binary)
{
    int decimal = 0;
    int counter = 0;
    int size = binary.length();
    for (int i = size - 1; i >= 0; i--)
    {
        if (binary[i] == '1') {
            decimal += pow(2, counter);
        }
        counter++;
```

```
    }
    return decimal;
}

void Encryption::generateKeys(string initialKey) {
    // The PC1 table
    int pc1[56] = {
    57,49,41,33,25,17,9,
    1,58,50,42,34,26,18,
    10,2,59,51,43,35,27,
    19,11,3,60,52,44,36,
    63,55,47,39,31,23,15,
    7,62,54,46,38,30,22,
    14,6,61,53,45,37,29,
    21,13,5,28,20,12,4
    };
    // The PC2 table
    int pc2[48] = {
    14,17,11,24,1,5,
    3,28,15,6,21,10,
    23,19,12,4,26,8,
    16,7,27,20,13,2,
    41,52,31,37,47,55,
    30,40,51,45,33,48,
    44,49,39,56,34,53,
    46,42,50,36,29,32
    };
    // 1. Compressing the key using the PC1 table
    string perm_key = "";
    for (int i = 0; i < 56; i++) {
        perm_key += initialKey[pc1[i] - 1];
    }
    // 2. Dividing the key into two equal halves
    string left = perm_key.substr(0, 28);
    string right = perm_key.substr(28, 28);
    for (int i = 0; i < 16; i++) {
        // 3.1. For rounds 1, 2, 9, 16 the key_chunks
        // are shifted by one.
        if (i == 0 || i == 1 || i == 8 || i == 15) {
            left = shiftLeftOnce(left);
            right = shiftLeftOnce(right);
        }
        // 3.2. For other rounds, the key_chunks
        // are shifted by two
        else {
            left = shiftLeftTwice(left);
            right = shiftLeftTwice(right);
        }
        // Combining the two chunks
        string combined_key = left + right;
```

```
        string round_key = "";
        // Finally, using the PC2 table to transpose the key bits
        for (int i = 0; i < 48; i++) {
            round_key += combined_key[pc2[i] - 1];
        }
        roundKeys[i] = round_key;
    }
}

string Encryption::encryptText64(string pt) {
    // The initial permutation table
    int initial_permutation[64] = {
    58,50,42,34,26,18,10,2,
    60,52,44,36,28,20,12,4,
    62,54,46,38,30,22,14,6,
    64,56,48,40,32,24,16,8,
    57,49,41,33,25,17,9,1,
    59,51,43,35,27,19,11,3,
    61,53,45,37,29,21,13,5,
    63,55,47,39,31,23,15,7
    };
    // The expansion table
    int expansion_table[48] = {
    32,1,2,3,4,5,4,5,
    6,7,8,9,8,9,10,11,
    12,13,12,13,14,15,16,17,
    16,17,18,19,20,21,20,21,
    22,23,24,25,24,25,26,27,
    28,29,28,29,30,31,32,1
    };
    // The substitution boxes. The should contain values
    // from 0 to 15 in any order.
    int substition_boxes[8][4][16] =
    { {
        14,4,13,1,2,15,11,8,3,10,6,12,5,9,0,7,
        0,15,7,4,14,2,13,1,10,6,12,11,9,5,3,8,
        4,1,14,8,13,6,2,11,15,12,9,7,3,10,5,0,
        15,12,8,2,4,9,1,7,5,11,3,14,10,0,6,13
    },
    {
        15,1,8,14,6,11,3,4,9,7,2,13,12,0,5,10,
        3,13,4,7,15,2,8,14,12,0,1,10,6,9,11,5,
        0,14,7,11,10,4,13,1,5,8,12,6,9,3,2,15,
        13,8,10,1,3,15,4,2,11,6,7,12,0,5,14,9
    },
    {
        10,0,9,14,6,3,15,5,1,13,12,7,11,4,2,8,
        13,7,0,9,3,4,6,10,2,8,5,14,12,11,15,1,
        13,6,4,9,8,15,3,0,11,1,2,12,5,10,14,7,
        1,10,13,0,6,9,8,7,4,15,14,3,11,5,2,12
```

```
},
{
   7,13,14,3,0,6,9,10,1,2,8,5,11,12,4,15,
   13,8,11,5,6,15,0,3,4,7,2,12,1,10,14,9,
   10,6,9,0,12,11,7,13,15,1,3,14,5,2,8,4,
   3,15,0,6,10,1,13,8,9,4,5,11,12,7,2,14
},
{
   2,12,4,1,7,10,11,6,8,5,3,15,13,0,14,9,
   14,11,2,12,4,7,13,1,5,0,15,10,3,9,8,6,
   4,2,1,11,10,13,7,8,15,9,12,5,6,3,0,14,
   11,8,12,7,1,14,2,13,6,15,0,9,10,4,5,3
},
{
   12,1,10,15,9,2,6,8,0,13,3,4,14,7,5,11,
   10,15,4,2,7,12,9,5,6,1,13,14,0,11,3,8,
   9,14,15,5,2,8,12,3,7,0,4,10,1,13,11,6,
   4,3,2,12,9,5,15,10,11,14,1,7,6,0,8,13
},
{
   4,11,2,14,15,0,8,13,3,12,9,7,5,10,6,1,
   13,0,11,7,4,9,1,10,14,3,5,12,2,15,8,6,
   1,4,11,13,12,3,7,14,10,15,6,8,0,5,9,2,
   6,11,13,8,1,4,10,7,9,5,0,15,14,2,3,12
},
{
   13,2,8,4,6,15,11,1,10,9,3,14,5,0,12,7,
   1,15,13,8,10,3,7,4,12,5,6,11,0,14,9,2,
   7,11,4,1,9,12,14,2,0,6,10,13,15,3,5,8,
   2,1,14,7,4,10,8,13,15,12,9,0,3,5,6,11
} };
// The permutation table
int permutation_tab[32] = {
16,7,20,21,29,12,28,17,
1,15,23,26,5,18,31,10,
2,8,24,14,32,27,3,9,
19,13,30,6,22,11,4,25
};
// The inverse permutation table
int inverse_permutation[64] = {
40,8,48,16,56,24,64,32,
39,7,47,15,55,23,63,31,
38,6,46,14,54,22,62,30,
37,5,45,13,53,21,61,29,
36,4,44,12,52,20,60,28,
35,3,43,11,51,19,59,27,
34,2,42,10,50,18,58,26,
33,1,41,9,49,17,57,25
};
//1. Applying the initial permutation
```

```
string perm = "";
for (int i = 0; i < 64; i++) {
    perm += pt[initial_permutation[i] - 1];
}
// 2. Dividing the result into two equal halves
string left = perm.substr(0, 32);
string right = perm.substr(32, 32);
// The plain text is encrypted 16 times
for (int i = 0; i < 16; i++) {
    string right_expanded = "";
    // 3.1. The right half of the plain text is expanded
    for (int i = 0; i < 48; i++) {
        right_expanded += right[expansion_table[i] - 1];
    };  // 3.3. The result is xored with a key
    string xored = xorStrings(roundKeys[i], right_expanded);
    string res = "";
    // 3.4. The result is divided into 8 equal parts and passed
    // through 8 substitution boxes. After passing through a
    // substituion box, each box is reduces from 6 to 4 bits.
    for (int i = 0; i < 8; i++) {
        // Finding row and column indices to lookup the
        // substituition box
        string row1 = xored.substr(i * 6, 1) + xored.substr(i * 6 + 5, 1);
        int row = convertBinaryToDecimal(row1);
        string col1 = xored.substr(i * 6 + 1, 1) + xored.substr(i * 6 + 2, 1) + xored.substr(i * 6 + 3, 1) + xored.substr(i * 6 + 4, 1);;
        int col = convertBinaryToDecimal(col1);
        int val = substition_boxes[i][row][col];
        res += convertDecimalToBinary(val);
    }
    // 3.5. Another permutation is applied
    string perm2 = "";
    for (int i = 0; i < 32; i++) {
        perm2 += res[permutation_tab[i] - 1];
    }
    // 3.6. The result is xored with the left half
    xored = xorStrings(perm2, left);
    // 3.7. The left and the right parts of the plain text are swapped
    left = xored;
    if (i < 15) {
        string temp = right;
        right = xored;
        left = temp;
    }
}
// 4. The halves of the plain text are applied
string combined_text = left + right;
string ciphertext = "";
// The inverse of the initial permuttaion is applied
for (int i = 0; i < 64; i++) {
    ciphertext += combined_text[inverse_permutation[i] - 1];
```

```
    }
    //And we finally get the cipher text
    return ciphertext;
}

string Encryption::decryptText64(string et)
{
    plaintext = et;
    string decryptedText = encryptText64(et);

    return decryptedText;
}

void Encryption::reverseRoundKeysSequence() {
    int i = 15;
    int j = 0;
    while (i > j)
    {
        string temp = roundKeys[i];
        roundKeys[i] = roundKeys[j];
        roundKeys[j] = temp;
        i--;
        j++;
    }
}

string DecimalToBinary(int dec) {
    if (dec < 1) return "0";

    string binStr = "";

    while (dec > 0)
    {
        binStr = binStr.insert(0, string(1, (char)((dec % 2) + 48)));

        dec /= 2;
    }

    return binStr;
}

string ASCIIToBinary(string str) {
    string bin = "";
    int strLength = str.length();

    for (int i = 0; i < strLength; ++i)
    {
        string cBin = DecimalToBinary(str[i]);
        int cBinLength = cBin.length();
```

```
        if (cBinLength < 8) {
            for (size_t i = 0; i < (8 - cBinLength); i++)
                cBin = cBin.insert(0, "0");
        }

        bin += cBin;
    }

    return bin;
}

void Encryption::setPlaintext(string text)
{
    plaintext = ASCIIToBinary(text);
    padding = 0;
    while (plaintext.size() % 256 != 0) {
        plaintext += "0";
        padding++;
    }
}

void Encryption::setEncryptionKey(string key)
{
    int keySize = key.size();
    if (keySize > 128)
        encryptionKey = key.substr(0, 128);
    else if (keySize < 128) {
        while (key.size() != 128)
            key += "0";
    }
    encryptionKey = key;
}

string Encryption::encryptText()
{
    string key1 = encryptionKey.substr(0, 64);
    string key2 = encryptionKey.substr(64, 64);

    string block1 = plaintext.substr(0, 64);
    string block2 = plaintext.substr(64, 64);
    string block3 = plaintext.substr(128, 64);
    string block4 = plaintext.substr(192, 64);

    string encryptedBlock = "";
    generateKeys(key1);
    encryptedBlock += encryptText64(block1);
    encryptedBlock += encryptText64(block2);
    generateKeys(key2);
    encryptedBlock += encryptText64(block3);
    encryptedBlock += encryptText64(block4);
```

```
    return encryptedBlock;
}

string Encryption::decryptText(string encryptedText)
{
    plaintext = encryptedText;

    string block1 = plaintext.substr(0, 64);
    string block2 = plaintext.substr(64, 64);
    string block3 = plaintext.substr(128, 64);
    string block4 = plaintext.substr(192, 64);

    string decryptedText = "";

    string key1 = encryptionKey.substr(0, 64);
    string key2 = encryptionKey.substr(64, 64);

    generateKeys(key1);
    reverseRoundKeysSequence();
    decryptedText += decryptText64(block1);
    decryptedText += decryptText64(block2);

    generateKeys(key2);
    reverseRoundKeysSequence();
    decryptedText += decryptText64(block3);
    decryptedText += decryptText64(block4);

    while (padding != 0) {
        decryptedText.pop_back();
        padding--;
    }

    return decryptedText;
}
```

HookCreator.h:

```
#pragma once

#include <iostream>
#include <Windows.h>
#include <iostream>
#include <process.h>

class HookCreator {
private:
        typedef bool (WINAPI* WRITEPROCESSMEMORY)(HANDLE, LPVOID, LPCVOID, SIZE_T, OUT SIZE_T);
        WRITEPROCESSMEMORY fpWriteProcessMemory = NULL;
        typedef bool (WINAPI* READPROCESSMEMORY)(HANDLE, LPVOID, LPCVOID, SIZE_T, OUT SIZE_T);
        WRITEPROCESSMEMORY fpReadProcessMemory = NULL;
```

```
public:
        void InitializeHook();

        void CreateWriteMemoryHook();
        void CreateReadMemoryHook();

        void EnableWriteMemoryHook();
        void EnableReadMemoryHook();

        void DisableWriteMemoryHook();
        void DisableReadMemoryHook();

        void Uninitialize();
};
```

HookCreator.cpp:

```
#include "HookCreator.h"
#include "MinHook.h"
#include <thread>
#pragma comment(lib, "libMinHook.x64.lib")

// Detour function which overrides original
bool WINAPI DetourWriteProcessMemory(HANDLE hProcess, LPVOID lpBaseAddress, LPCVOID lpBuffer, SIZE_T nSize,
   OUT SIZE_T* lpNumberOfBytesWritten)
{
   //#ifdef _WIN32
   //    MessageBox(nullptr, TEXT("WriteProcessMemory function was called!"), TEXT("Alert"), MB_ICONWARNING);
   //#endif
   std::cout << "WriteMemory was captured" << std::endl;
   return 1;
}

bool WINAPI DetourReadProcessMemory(HANDLE hProcess, LPVOID lpBaseAddress, LPCVOID lpBuffer, SIZE_T nSize,
   OUT SIZE_T* lpNumberOfBytesWritten)
{
   //#ifdef _WIN32
   //    MessageBox(nullptr, TEXT("WriteProcessMemory function was called!"), TEXT("Alert"), MB_ICONWARNING);
   //#endif
   std::cout << "ReadMemory was captured" << std::endl;
   return 1;
}

// Initialize MinHook.
void HookCreator::InitializeHook()
{
   if (MH_Initialize() != MH_OK)
   {
      return;
```

```
    }
}

// Create a hook in disabled state.
void HookCreator::CreateWriteMemoryHook()
{
    if (MH_CreateHook(&WriteProcessMemory, &DetourWriteProcessMemory,
        reinterpret_cast<LPVOID*>(&fpWriteProcessMemory)) != MH_OK)
    {
        return;
    }
}

// Create a hook in disabled state.
void HookCreator::CreateReadMemoryHook()
{
    if (MH_CreateHook(&ReadProcessMemory, &DetourReadProcessMemory,
        reinterpret_cast<LPVOID*>(&fpReadProcessMemory)) != MH_OK)
    {
        return;
    }
}

// Enable the hook
void HookCreator::EnableWriteMemoryHook()
{
    if (MH_EnableHook(&WriteProcessMemory) != MH_OK)
    {
        return;
    }
}

// Enable the hook
void HookCreator::EnableReadMemoryHook()
{
    if (MH_EnableHook(&ReadProcessMemory) != MH_OK)
    {
        return;
    }
}

// Disable the hook
void HookCreator::DisableWriteMemoryHook()
{
    if (MH_DisableHook(&WriteProcessMemory) != MH_OK)
    {
        return;
    }
}
```

```
// Disable the hook
void HookCreator::DisableReadMemoryHook()
{
   if (MH_DisableHook(&ReadProcessMemory) != MH_OK)
   {
      return;
   }
}

// Uninitialize MinHook.
void HookCreator::Uninitialize()
{

   if (MH_Uninitialize() != MH_OK)
   {
      return;
   }
}
```

KeyGenerator.h:

```
#pragma once

#include <iostream>

class KeyGenerator {
public:
		std::string createKey();
		std::string createKey128();
};
```

KeyGenerator.cpp:

```
#include "KeyGenerator.h"

#include <iostream>

std::string KeyGenerator::createKey() {
   srand(time(NULL));
   std::string key = "";

   for (int i = 0; i < 64; i++) {
      key += (rand() % 2) + '0';
   }

   return key;
}
```

```
std::string KeyGenerator::createKey128() {
    srand(time(NULL));
    std::string key = "";

    for (int i = 0; i < 128; i++) {
        key += (rand() % 2) + '0';
    }

    return key;
}
```

MemoryIO.h:

```
#pragma once

#include <iostream>
#include "HookCreator.h"

class MemoryIO {
public:
        std::string readString(std::string* addressForReading, int size);
        void writeString(std::string* addressForWriting, std::string newValue);
};
```

MemoryIO.cpp:

```
#include "MemoryIO.h"

std::string MemoryIO::readString(std::string* addressForReading, int size) {
    DWORD procID = _getpid(); // A 32-bit unsigned integer, DWORDS are mostly used to store Hexadecimal Addresses
    HANDLE handle = OpenProcess(PROCESS_VM_READ, FALSE, procID); // Opening the Process with All Access
    std::string readBuffer = "";

    if (procID == NULL) {
        exit(-1); // Exit the program if it did not find the Window
    }
    else
        // Read the Process Memory, we read the Value from and save it in readBuffer
        ReadProcessMemory(handle, (PBYTE*)addressForReading, &readBuffer, size, 0);
    return readBuffer;
}

void MemoryIO::writeString(std::string* addressForWriting, std::string newValue) {
    DWORD procID = _getpid(); // A 32-bit unsigned integer, DWORDS are mostly used to store Hexadecimal Addresses
    HANDLE handle = OpenProcess(PROCESS_ALL_ACCESS, FALSE, procID); // Opening the Process with All Access

    if (procID == NULL) {
        exit(-1); // Exit the program if it did not find the Window
```

```
    }
    else
        // Write the newValue into the Process Memory, 03007640 is the Address
        WriteProcessMemory(handle, (PBYTE*)addressForWriting, &newValue, sizeof(newValue), 0);
}
```

SecureString.h:

```
#pragma once

#include <iostream>
#include "HookCreator.h"
#include "Encryption.h"

class SecureString {
public:
        void setValue(std::string inputValue, std::string* address);
        std::string getValue();

        void startProtect();
        int binaryToDecimal(std::string n);
        std::string setStringtoASCII(std::string str);
        void pauseProtect();
        void stopProtect();
private:
        std::string protectedValue;
        std::string* oldAddress;
        std::string encryptionKey;

        Encryption enc;
        HookCreator hc;
        bool rewriteOldAddress();
};
```

SecureString.cpp:

```
#include "SecureString.h"
#include "KeyGenerator.h"
#include "Encryption.h"
#include "MemoryIO.h"

void SecureString::setValue(std::string inputValue, std::string* address)
{
        protectedValue = inputValue;
        oldAddress = address;

        hc.InitializeHook();
}
```

```
std::string SecureString::getValue()
{
	return string();
}

void SecureString::startProtect()
{
	bool result = rewriteOldAddress();
	if (result) {
		hc.CreateWriteMemoryHook();
		hc.EnableWriteMemoryHook();
		hc.CreateReadMemoryHook();
		hc.EnableReadMemoryHook();

		KeyGenerator kg;
		encryptionKey = kg.createKey128();

		enc.setPlaintext(protectedValue);
		enc.setEncryptionKey(encryptionKey);

		protectedValue = enc.encryptText();
	}
	else
		return;
}

int SecureString::binaryToDecimal(string n)
{
	string num = n;

	// Stores the decimal value
	int dec_value = 0;

	// Initializing base value to 1
	int base = 1;

	int len = num.length();
	for (int i = len - 1; i >= 0; i--) {

		// If the current bit is 1
		if (num[i] == '1')
			dec_value += base;
		base = base * 2;
	}

	// Return answer
	return dec_value;
}
string SecureString::setStringtoASCII(string str)
```

```
{
	// To store size of s
	int N = int(str.size());

	// If given string is not a
	// valid string
	if (N % 8 != 0) {
		return "Not Possible!";
	}

	// To store final answer
	string res = "";

	// Loop to iterate through string
	for (int i = 0; i < N; i += 8) {
		int decimal_value = binaryToDecimal((str.substr(i, 8)));

		// Apprend the ASCII character
		// equivalent to current value
		res += char(decimal_value);
	}
	// Return Answer
	return res;
}

void SecureString::pauseProtect()
{
	hc.DisableWriteMemoryHook();
	hc.DisableReadMemoryHook();
	string binary = enc.decryptText(protectedValue);
	protectedValue = setStringtoASCII(binary);
	MemoryIO mio;
	mio.writeString(oldAddress, protectedValue);
}

void SecureString::stopProtect()
{
	hc.Uninitialize();
}

bool SecureString::rewriteOldAddress()
{
	MemoryIO mio;
	std::string trashValue = "";
	for (int i = 0; i < protectedValue.size(); i++) {
		trashValue += "0";
	}
	mio.writeString(oldAddress, trashValue);
	return true;
}
```

ЗАТВЕРДЖУЮ
Директор ТОВ «БРОКСТАР»
_____ Володимир ТАРАСЮК
«13» грудня 2023 р.

АКТ
про впровадження результатів магістерської кваліфікаційної роботи
Паламарчука Олександра Руслановича

Комісія у складі: голова комісії – директор Володимир Тарасюк, розглянувши матеріали магістерської роботи Олександра Паламарчука на тему «Метод та програмна бібліотека захисту програм від дампінгу», склав цей акт про те, що у ТОВ «БРОКСТАР» впроваджено результати цієї роботи, а саме програмну бібліотеку захисту програм від дампінгу.

Впроваджені результати дозволяють здійснювати захист даних в оперативній пам'яті, що дало змогу підвищити захищеність використовуваних програмних засобів.

Голова комісії:
Директор Володимир ТАРАСЮК

ПРОТОКОЛ ПЕРЕВІРКИ
МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Метод та програмна бібліотека захисту програм від дампінгу

Автор роботи: Паламарчук Олександр Русланович
Тип роботи: магістерська кваліфікаційна робота

Підрозділ кафедра захисту інформації ФІТКІ
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність – 96,73 %. Схожість – 3,27 %.

Аналіз звіту подібності (відмітити потрібне):

√ 1. Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

□ 2. Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

□ 3. Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку (підпис) Валентина КАПЛУН

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи (підпис) Олександр ПАЛАМАРЧУК

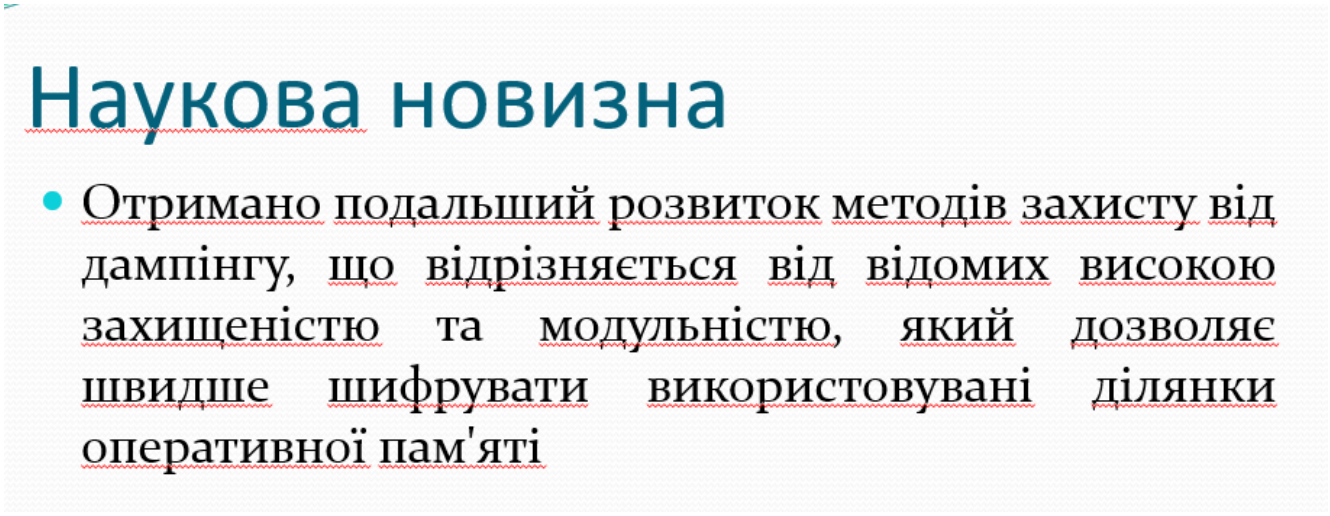
Керівник роботи (підпис) Юрій БАРИШЕВ

Додаток Д

ІЛЮСТРАТИВНА ЧАСТИНА

МЕТОД ТА ПРОГРАМНА БІБЛІОТЕКА ЗАХИСТУ ПРОГРАМ ВІД ДАМПІНГУ

Наукова новизна



Предмет, об’єкт, мета та задачі дослідження

Мета та задачі

- Об’єктом даної роботи є процес захисту програм від дампінгу.
- Предметом — методи та засоби захисту програм від дампінгу.

Метою роботи є покращення існуючих методів захисту програм від дампінгу.

Для досягнення мети потрібно розв’язати такі задачі:

- проаналізувати існуючі методи для захисту програм від дампінгу;
- проаналізувати відомі засоби захисту програм від дампінгу;
- розробити власний комплексний метод захисту.

Мета та задачі

Для досягнення мети потрібно розв’язати такі задачі:

- розробити власний комплексний засіб для захисту;
- протестувати власний програмний засіб;
- обчислити економічну доцільність.

Метод блокування переривань копіювання пам'яті

Математичний опис процесу захисту від дампінгу

$$I = \{C, D, K\},$$

де I – входи, C – сегмент пам'яті з програмним кодом програми, D – сегмент пам'яті даних, K – множина ключів для симетричного шифрування

$$IT = \{HF, KC, DE, MC\},$$

де IT – внутрішні перетворення, HF – встановлення хуків для функції дампінгу, KC – створення множини ключів для шифрування, DE – зашифрування даних

$$IS = \{SH, SC\},$$

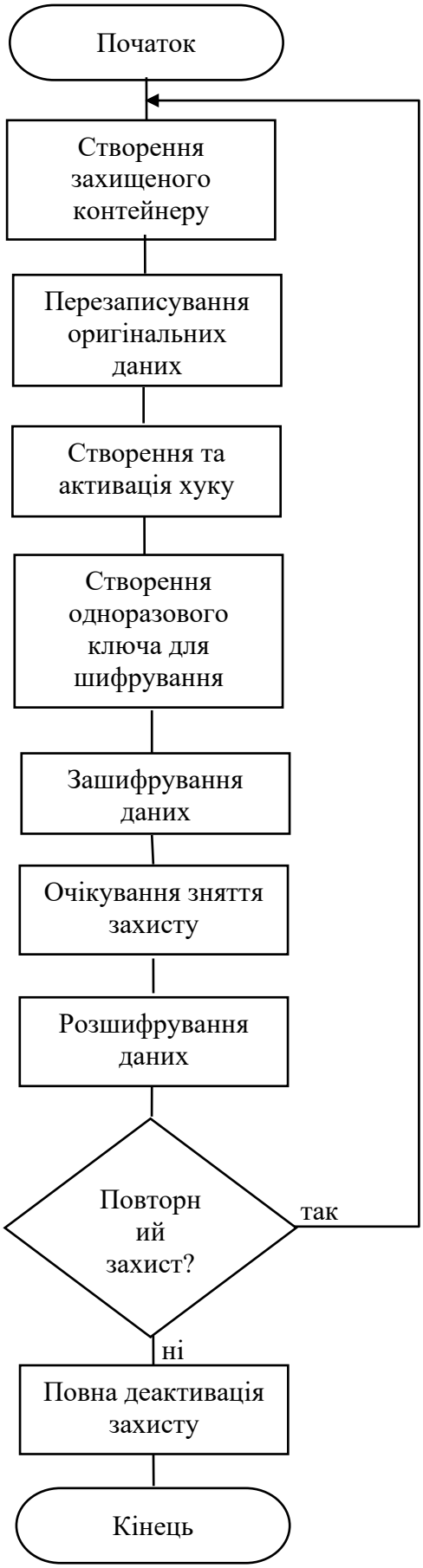
де IS – внутрішні стани, SH – стан встановленості хуків на функції дампінгу, SC – стан очищеності даних в оперативній пам’яті

$$O = \{C, ED, K\}$$

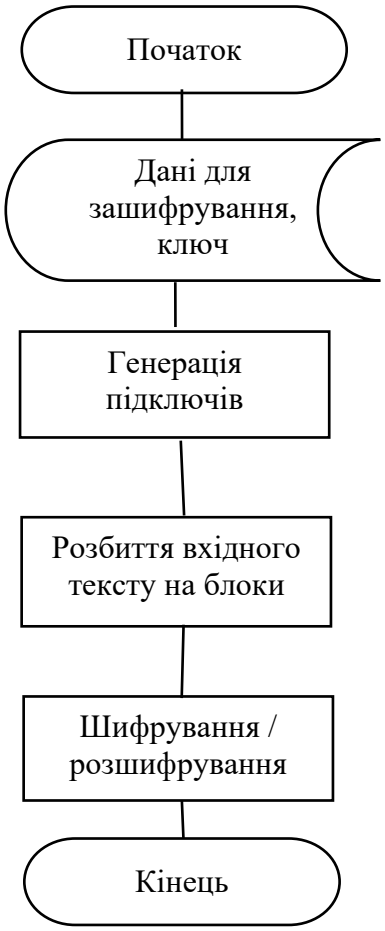
де O – виходи, ED – дані зашифровані в оперативній пам’яті

$$(I) \rightarrow [TS, IS] \rightarrow (O)$$

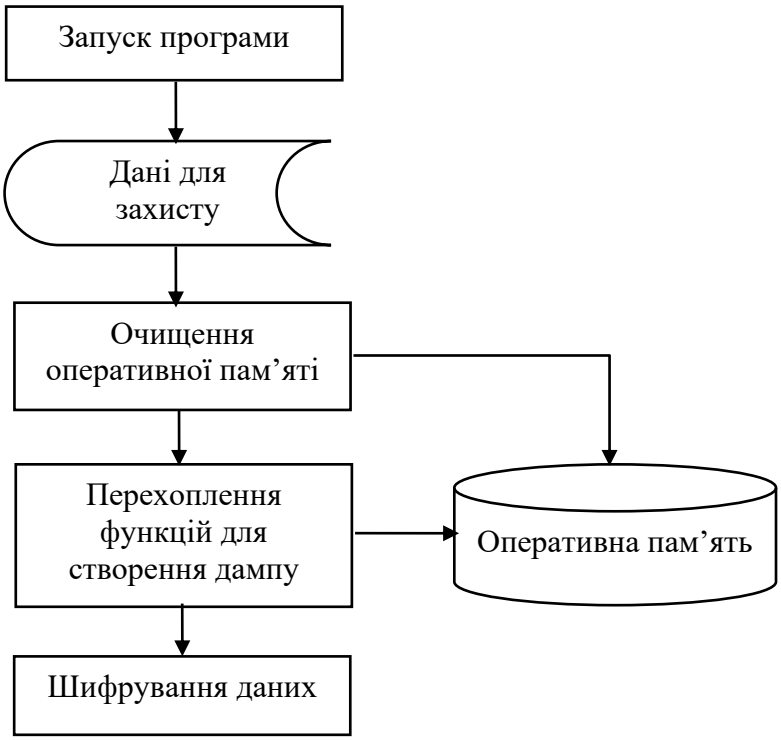
Метод захисту від дампінгу



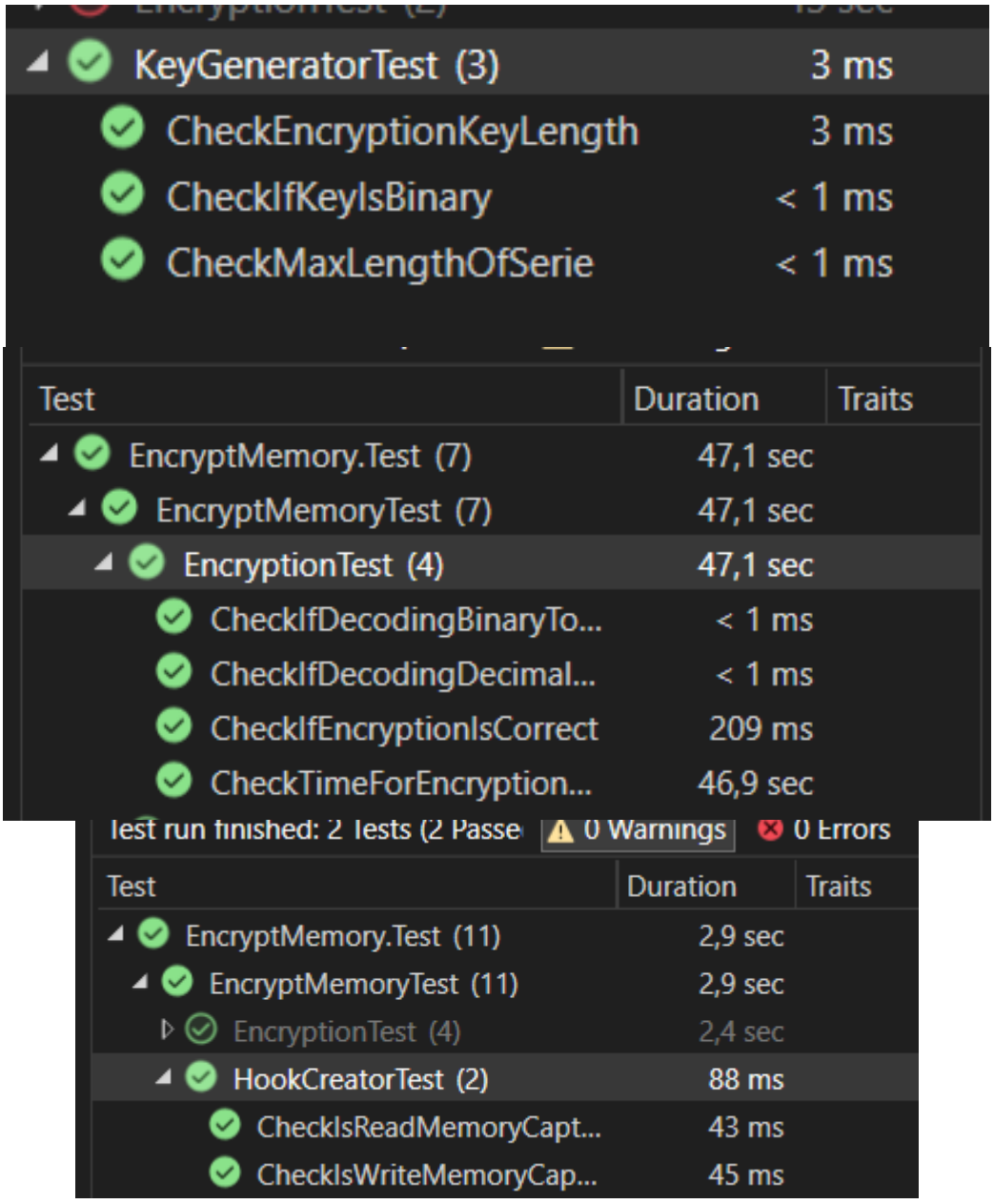
Метод шифрування



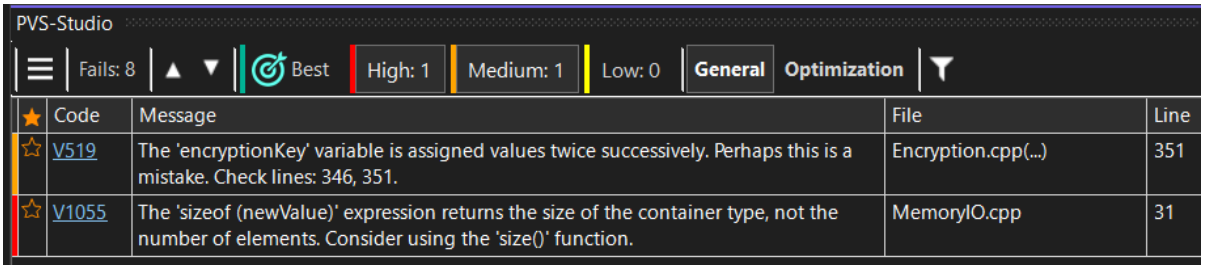
Архітектура засобу



Результати блокового тестування



Результати статичного тестування



Економічні показники розробки

Економічна частина

- рівень комерційного потенціалу розробки високий і становить 42,3 бали
- Термін окупності становить 1,8 роки