УДК 004.04: 004.4'2: 004.43: 004.49
DOI https://doi.org/10.32782/2663-5941/2024.5.1/18

***Головко І.Г.***
Хмельницький національний університет

***Савенко О.С.***
Хмельницький національний університет

***Медзатий Д.М.***
Хмельницький національний університет

***Іванченко О.В.***
Національний технічний університет «Дніпровська політехніка»

# МЕТОДИ ОБФУСКАЦІЇ ПРОГРАМНОГО КОДУ ЗА ДОПОМОГОЮ ШТУЧНОГО ІНТЕЛЕКТУ

*У статті розглядаються сучасні підходи до обфускації .NET коду з використанням технологій штучного інтелекту (AI). Обфускація коду є важливим інструментом для захисту програмного забезпечення від зворотного інжинірингу та несанкціонованого доступу до вихідного коду. Традиційні методи обфускації, такі як перейменування ідентифікаторів, зміна потоку управління та шифрування рядків, мають певні обмеження, оскільки вони є статичними і можуть бути легко подолані сучасними декомпіляційними інструментами. Ці методи часто потребують значних витрат часу і ресурсів на ручне налаштування та підтримку. У зв'язку з цим виникає необхідність впровадження більш інтелектуальних підходів, здатних автоматично адаптуватися до нових загроз.*

*У цій роботі пропонується підхід до обфускації .NET коду на основі штучного інтелекту, який включає в себе використання машинного навчання для аналізу та вибору оптимальних стратегій обфускації. Основні компоненти запропонованої інтелектуальної системи обфускації включають модуль введення, модуль попередньої обробки, AI-базований модуль обфускації, модуль перевірки якості та модуль зберігання даних. Модуль обфускації на основі AI використовує алгоритми глибинного навчання для аналізу структури коду і вибору найбільш ефективних методик обфускації, що забезпечує значне підвищення складності та захищеності коду від зворотного інжинірингу.*

*У статті також представлені результати експериментального дослідження, спрямованого на оцінку ефективності AI-базованого підходу до обфускації. Експеримент проводився на вибірці з 200 файлів, скомпільованих з дапомогою MSBuild інструменту, розділених на контрольну та експериментальну групи. Отримані результати показали значне зростання опору реверс-інжинірингу в експериментальній групі у порівнянні з контрольної, при цьому зберігається висока функціональність та мінімальний вплив на продуктивність програмного забезпечення.*

*Таким чином, дослідження підтверджує ефективність використання AI у процесі обфускації .NET коду. Це відкриває нові можливості для створення більш надійних систем захисту програмного забезпечення, здатних адаптуватися до нових кіберзагроз і забезпечувати високий рівень конфіденційності та цілісності даних. У роботі зроблено висновки про перспективи подальших досліджень у напрямку вдосконалення алгоритмів навчання і розробки нових методик обфускації з використанням штучного інтелекту.*

***Ключові слова:*** *обфускація коду, безпека програмного забезпечення, захист програмного коду, складність коду.*

**Постановка проблеми. Актуальність проблеми захисту програмного забезпечення та обфускації коду.** Захист програмного забезпечення є однією з ключових проблем в сучасних умовах цифрової економіки. З кожним роком зростає кількість кібератак та методів зловмисників, спрямованих на отримання несанкціонованого доступу до вихідного коду програм. Це включає в себе зворотний інжиніринг, аналіз коду для пошуку вразливостей, а також спроби несанкціонованого використання або зміни програмного забезпечення. Зважаючи на такі загрози, важливість забезпечення конфіденційності та цілісності коду стає очевидною.

Традиційні методи обфускації коду спрямовані на ускладнення його розуміння та аналізу шляхом зміни ідентифікаторів, структури коду, введення "мертвого" коду та інших технік. Однак ці методи мають певні обмеження, адже вони зазвичай є статичними і часто легко піддаються аналізу за допомогою сучасних декомпіляційних інструментів. Крім того, вони вимагають ручного налаштування, що може бути трудомістким процесом та призводити до виникнення помилок.

Зважаючи на обмеження традиційних методів, виникає потреба у впровадженні більш інтелектуальних підходів до обфускації коду, які можуть динамічно адаптуватися до змін середовища та нових загроз. Штучний інтелект, зокрема методи машинного навчання, пропонують нові можливості для автоматизації процесу обфускації. Використання AI дозволяє не лише автоматично вибирати оптимальні стратегії обфускації, але й постійно вдосконалювати ці стратегії на основі зворотного зв'язку та нових даних. Це робить процес обфускації більш гнучким, ефективним та стійким до сучасних методів зворотного інжинірингу.

Обфускація коду є особливо важливою для програм, що розробляються на платформі .NET, оскільки їх вихідний код компілюється в проміжну мову IL (Intermediate Language), яка є більш доступною для аналізу і реверс-інжинірингу. Це робить .NET програми вразливими до атак, особливо якщо вони містять критичні або конфіденційні дані. Застосування обфускації, особливо з використанням AI, дозволяє значно підвищити рівень захисту таких програм, ускладнюючи їх аналіз та використання зловмисниками.

**Аналіз останніх досліджень і публікацій.** У сучасних умовах інформаційної безпеки обфускація коду є важливим інструментом захисту програмного забезпечення від несанкціонованого доступу та зворотного інжинірингу. Ця методика була предметом багатьох наукових досліджень, які значно розширили можливості програмного захисту.

Традиційні підходи до обфускації коду були детально вивчені та описані в численних дослідженнях. Наприклад, у роботах Крістіана Колберга (Christian Collberg) і Кларка Томпсона (Clark Thomborson) [1, с. 15] була розроблена таксономія обфускаційних трансформацій, яка стала базисом для подальших досліджень у цій галузі. Вони запропонували класифікацію методів обфускації на основі ступеня їх ефективності та впливу на продуктивність програмного забезпечення.

Ці методи включають перейменування ідентифікаторів, зміну потоку управління, вставку «мертвого» коду та шифрування рядків. Однією з найбільш ефективних технік обфускації є зміна потоку управління, яка значно ускладнює аналіз та реверс-інжиніринг програм. У роботі [2, с. 2] автори досліджують можливості оптимізації компілятора для обфускації потоку управління, що дозволяє створити більш стійкі до аналізу програми на етапі компіляції. Кожен із цих методів має свої переваги та недоліки. Перейменування ідентифікаторів є простим, але не дуже ефективним способом захисту, оскільки його можна легко подолати за допомогою автоматизованих інструментів декомпіляції. Зміна потоку управління, з іншого боку, значно ускладнює розуміння логіки програми, але може призводити до значних втрат у продуктивності(для обчислення складності розуміння логіки програми було використано циклопатину складність Томаса Маккейба (Thomas McCabe)). Цикломатична складність є одним із основних показників для оцінки ефективності обфускації. Підвищення цього показника свідчить про збільшення складності аналізу коду. У роботі [3, с. 480] автори описують роль цикломатичної складності у визначенні рівня складності продукту, що є корисним для оцінки ефективності захисних заходів, таких як обфускація. Також обфускація та мінімізація часто використовуються для приховування коду у вебдодатках. У роботі [4, с. 7–10] досліджується, як обфускований та мінімізований код у вебдодатках може використовуватися як для захисту, так і для приховування потенційно шкідливого коду. Обфускація відіграє важливу роль у захисті конфіденційних даних, зокрема в IoT додатках, де виникають значні ризики порушення приватності. У роботі [5, с. 6–14] представлено підхід подвійної обфускації для захисту місцезнаходження в додатках на основі IoT, що дозволяє підвищити рівень безпеки та знизити ризики витоку інформації. З розвитком технологій штучного інтелекту (AI) відкрилися нові можливості для вдосконалення процесу обфускації коду. Однією з головних переваг AI є його здатність адаптуватися до нових загроз і автоматично вибирати найбільш ефективні стратегії захисту. У традиційних методах обфускації часто використовуються статичні підходи, які вимагають ручного налаштування та контролю. Вони можуть бути ефективними на певному етапі, але їхня ефективність знижується, коли змінюються атаки або інструменти для зворотного інжинірингу.

Використання штучного інтелекту дозволяє зробити процес обфускації динамічним. За допомогою машинного навчання та глибинного навчання можна створювати моделі, які аналізують вихідний код і автоматично підбирають найкращі методики для його захисту. Такий підхід не тільки зменшує необхідність у ручній роботі, але й забезпечує більш стійкий захист, оскільки обфускація стає адаптивною і здатною змінюватися відповідно до нових загроз. Штучний інтелект (ШІ) стає невід'ємною частиною сучасних методик обфускації, дозволяючи адаптувати стратегії захисту до нових загроз.

Крім того, AI дозволяє впровадити елемент самонавчання у процес обфускації. Моделі на основі машинного навчання можуть аналізувати успішність застосованих стратегій, адаптуватися до нових умов і вдосконалюватися на основі отриманих даних. Це значно підвищує стійкість програмного забезпечення до атак, оскільки системи обфускації можуть швидко реагувати на нові типи зворотного інжинірингу. Дослідження [6, с. 8] демонструє можливість використання нейронних мереж для аналізу вихідного коду і динамічного створення обфускаційних алгоритмів. Такі моделі забезпечують адаптацію обфускаційних стратегій у реальному часі, залежно від типу коду та виявлених вразливостей. Також слід відзначити внесок роботи [7, с. 22–25], яка описує переваги інтеграції методів ШІ у процесі обфускації на різних етапах розробки програмного забезпечення. Зокрема, використання алгоритмів машинного навчання дозволяє знизити ручне втручання та підвищити ефективність обфускації без значного впливу на продуктивність програм. У статті [8, с. 8] зазначено, що обфускація часто використовується для приховування шкідливого програмного забезпечення в нелегально клонованих додатках, особливо на платформі Android. Однак обфускація також може використовуватися для захисту легітимних додатків від копіювання коду. Автори пропонують кілька моделей глибинного навчання для виявлення та класифікації обфускації у додатках Android. Вони використовують гібридну модель, яка поєднує підходи обробки природної мови та розпізнавання зображень, і досягають значних покращень у порівнянні з попередніми методами виявлення обфускації. Окрім методів захисту, варто враховувати і можливості атак на обфускацію за допомогою машинного навчання. У роботі [9, с. 47–55] розглядається обфускація маршрутизації та аналіз атак з використанням машинного навчання, що показує можливі слабкі місця в системах обфускації. Штучний інтелект дозволяє не лише покращувати методи обфускації, але й сприяє виявленню шкідливого програмного забезпечення. У роботі [10, с. 80–88] автори досліджують, як AI може бути використаний для посилення методів обфускації, зокрема для ускладнення виявлення шкідливого ПЗ за допомогою обфускованого коду. Штучний інтелект також може бути використаний для виявлення ботнетів у розподілених системах. У роботі [11, с. 190–198] розглядається підхід до виявлення ботнетів, що може бути інтегрований з методами обфускації для підвищення загального рівня безпеки. Машинне навчання є ефективним інструментом для виявлення багатовекторних кіберзагроз. У роботі [12, с. 239] досліджується застосування алгоритмів машинного навчання для аналізу трафіку та виявлення кіберзагроз в середовищі Інтернету речей (IoT), що можна застосувати у поєднанні з методами обфускації для підвищення безпеки. Також AI-методи можуть бути використані для виявлення специфічних загроз, таких як DNS тунелювання ботнетів. У роботі [13] обговорюється підхід до виявлення ботнетів через аналіз DNS тунелювання, що може бути корисним у поєднанні з методами обфускації для захисту розподілених систем від атак. Інші методи штучного інтелекту, такі як згорткові нейронні мережі (CNN), можуть бути ефективними у виявленні шкідливого ПЗ на мобільних платформах. У роботі [14, с. 198–213] описується метод виявлення шкідливого ПЗ на платформі Android, заснований на моделі CNN змішаних даних, що дозволяє ідентифікувати шкідливі програми навіть після їх обфускації.

В результаті використання AI в обфускації дозволяє підвищити рівень захисту, зробити процес більш автоматизованим і ефективним, зменшуючи при цьому вплив на продуктивність програмного забезпечення.

Розглянемо дослідження обфускації в контексті мов програмування високого рівня. Мови програмування високого ріння(такі як Java, мови платформи .NET) використовують проміжний код (IL). Проміжні представлення, як описано в роботі [15, с. 159–207, 221–223], є важливим етапом у процесі компіляції, оскільки вони забезпечують високий рівень абстракції та полегшують аналіз коду. У даній роботі детально описується роль IR(Intermediate Representations) у компіляторах і те, як рішення щодо того, як представляти код, впливає на ефективність обфускації. Це робить їх ключовою точкою вразливості для атак з боку

зловмисників. Саме тому обфускація на рівні IL є критично важливою для захисту програмного забезпечення. Окрім захисту від зворотного інжинірингу, обфускація також може використовуватися для захисту інтелектуальної власності та цифрових контрактів. У роботі [16, с. 439–448] розглядається метод водяних знаків для смарт-контрактів, заснований на обфускації коду, що забезпечує додатковий рівень безпеки та захисту даних у блокчейн-системах. Щоб захистити IL код, використовуються різні методи обфускації, які ускладнюють аналіз коду, але не впливають на його функціональність. На рис. 1 представлена загальна модель компіляції високорівневої мови програмівання в IL Code з використанням обфускації. Для прикладу було взято мову програмування високого рівня C#.

Одним із найпоширеніших методів обфускації є перейменування ідентифікаторів. Цей метод змінює назви змінних, класів, методів і інших ідентифікаторів на випадкові або нерозбірливі імена. Це робить аналіз коду складнішим, оскільки зникає семантична інформація, яка могла б допомогти в розумінні призначення компонентів коду.

Ще одним потужним методом є обфускація потоку управління. Вона змінює логіку виконання програми таким чином, що код залишається функціональним, але його виконання стає менш зрозумілим. Наприклад, використовуються фальшиві цикли або додаткові умовні оператори, що ускладнюють аналіз послідовності виконання програми.

Вставка «мертвого» або зайвого коду також є ефективною технікою обфускації. У код додаються блоки, які не впливають на функціональність програми, але роблять її структуру більш заплутаною. Це ускладнює процес реверс-інжинірингу, оскільки додатковий код збільшує складність програми без зміни її поведінки.

Для захисту даних, таких як рядки тексту, використовується шифрування рядків. Це дозволяє уникнути витоку важливої інформації з коду, наприклад, повідомлень про помилки або URL-адрес, що можуть бути витягнуті під час аналізу файлів.

Обфускація ресурсів також відіграє важливу роль у захисті програмних активів, таких як зображення або аудіофайли. Шифрування або модифікація цих ресурсів захищає їх від несанкціонованого доступу або змін.

Використання цих методів обфускації на рівні IL коду є критично важливим для захисту програм. Поєднання різних технік, таких як перейменування ідентифікаторів, обфускація потоку управління та шифрування рядків, забезпечує вищий рівень захисту, особливо коли ці методи використовуються разом із технологіями штучного інтелекту. Штучний інтелект дозволяє автоматизувати процес обфускації, вибираючи оптимальні стратегії для конкретного коду та адаптуючись до нових загроз.

**Постановка завдання.** Метою цієї статті є розробка, впровадження та аналіз ефективності системи обфускації коду на базі штучного інтелекту для програм на платформі .NET. В умовах постійного зростання кількості атак, спрямованих на реверс-інжиніринг програмного забезпечення, традиційні методи обфускації коду стають все менш ефективними. Тому необхідно розробити новий підхід, який дозволить динамічно адаптувати обфускаційні стратегії до конкретних загроз та підвищити загальний рівень захисту програмного забезпечення.

Основними завданнями статті є:

1. Аналіз існуючих методів обфускації та їхніх обмежень: Необхідно провести огляд традиційних технік обфускації коду, таких як перейменування

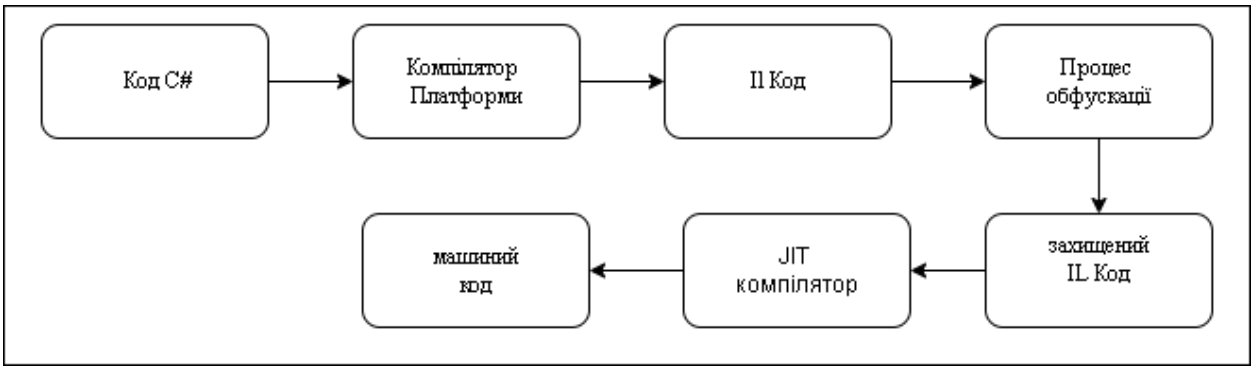


**Рис. 1. Узагальнена схема процесу компіляції з використанням обфускації**

ідентифікаторів, зміна потоку управління, шифрування рядків та вставка «мертвого» коду. Важливо також виявити недоліки цих методів у контексті захисту програмного коду.

2. Розробка AI-базованої системи обфускації: Метою є створення інтелектуальної системи, що використовує алгоритми машинного та глибинного навчання для вибору та застосування оптимальних методик обфускації в залежності від специфіки вихідного коду. Система має автоматично адаптуватися до нових загроз та покращувати ефективність захисту.

3. Впровадження динамічних методик обфускації на рівні IL коду: Необхідно розробити рішення для обфускації проміжного коду .NET (IL), що ускладнить процес реверс-інжинірингу та декомпіляції. Оскільки IL код є одним із найбільш вразливих елементів .NET програм, його обфускація є критично важливою для підвищення загальної безпеки.

4. Аналіз та інтерпретація отриманих результатів: Оцінка ефективності запропонованої системи обфускації базуватиметься на отриманих експериментальних даних. Важливо виявити, чи AI-базовані методики обфускації покращують захист, зберігаючи при цьому мінімальні втрати продуктивності та функціональності.

5. Визначення напрямків подальших досліджень: На основі отриманих результатів важливо визначити, як можна вдосконалити запропоновану систему. Це може включати дослідження нових моделей машинного навчання, розширення можливостей обфускації для різних типів програмного забезпечення та впровадження додаткових методик захисту.

Основні результати, які передбачається досягти:

1. Підвищення рівня захисту програмного забезпечення: AI-базована обфускація повинна ускладнити декомпіляцію та аналіз програмного забезпечення, підвищуючи стійкість до реверс-інжинірингу.

2. Автоматизація процесу обфускації: Інтелектуальна система обфускації повинна забезпечити автоматичний вибір стратегій обфускації без необхідності ручного втручання, що дозволить знизити витрати часу та ресурсів.

3. Збереження продуктивності та функціональності: AI-базовані методи повинні зберігати стабільність роботи програмного забезпечення після обфускації, забезпечуючи мінімальний вплив на його швидкодію та функціональні можливості.

**Виклад основного матеріалу.** Процес обфускації коду є ключовою стратегією захисту програмного забезпечення від зворотного інжинірингу та несанкціонованого аналізу. У традиційних методах обфускації використовуються статичні техніки, такі як перейменування ідентифікаторів, зміна потоку управління, вставка «мертвого» коду або шифрування рядків. Ці методи, хоча й ефективні у певних випадках, часто вимагають ручного втручання та можуть бути обмеженими, коли мова йде про сучасні декомпіляційні інструменти, здатні легко розпізнавати такі зміни. Кожен із методів має свої сильні та слабкі сторони, які залежать від контексту застосування та типу загроз. У цьому розділі проведено порівняльний аналіз найбільш відомих технік обфускації, що допоможе визначити їх ефективність залежно від конкретних вимог до безпеки. Розгляд таких аспектів, як стійкість до атак, вплив на продуктивність, гнучкість та легкість впровадження, дозволяє зробити комплексну оцінку кожного методу. У таблиці нижче наведено основні переваги та недоліки методів, що найчастіше використовуються для захисту коду в різних середовищах програмування.

Для глибшого розуміння ролі AI у процесі обфускації важливо розглянути основні типи методів, які використовуються для захисту коду.

Таблиця 1

**Порівняння переваг та недоліків відомих методів обфускації**

| Метод обфускації | Переваги | Недоліки |
|---|---|---|
| Перейменування ідентифікаторів | Легко реалізується, швидке виконання | Легко піддається реверс-інжинірингу з сучасними інструментами |
| Зміна потоку управління | Ускладнює логіку програми, підвищує захист від реверс-інжинірингу | Може знизити продуктивність та ускладнює підтримку |
| Вставка «мертвого» коду | Збільшує складність коду та ускладнює аналіз | Може збільшити розмір програми без суттєвого підвищення захисту |
| Шифрування рядків | Захищає чутливі дані у програмі | Потребує ресурсів для розшифрування під час виконання |
| Обфускація з використанням ШІ | Динамічна адаптація до нових загроз, підвищена стійкість | Складність реалізації, потребує великих обчислювальних ресурсів |

Нижче наведено класифікацію методів обфускації на основі їх характерних рис:

1. За типом трансформації:

1.1. Синтаксичні методи: Включають перейменування ідентифікаторів, що змінює назви змінних, класів, методів, роблячи їх менш зрозумілими. Ці методи є простими у реалізації, але мають обмежену ефективність проти сучасних інструментів реверс-інжинірингу.

1.2. Семантичні методи: Зміна потоку управління і введення «мертвого» коду є прикладами цих методів. Вони ускладнюють аналіз логіки програми, але можуть негативно впливати на продуктивність.

1.3. Шифрування рядків: Метод, що захищає конфіденційні дані всередині програми шляхом їх шифрування.

2. За рівнем коду:

2.1. Обфускація на рівні вихідного коду: Застосовується до коду високого рівня, що полегшує приховування логіки програми на рівні розробки.

2.2. Обфускація на рівні проміжного коду (IL/bytecode): Особливо корисна для таких платформ, як .NET або Java, де програмний код компілюється у проміжний байт-код, що легко декомпілюється.

2.3. Обфускація на рівні машинного коду: Обфускація безпосередньо в бінарних файлах, що знижує ймовірність реверс-інжинірингу на найнижчому рівні.

3. За цілями:

3.1. Заплутування коду: Орієнтоване на те, щоб зробити код менш зрозумілим і важким для аналізу.

3.2. Захист конфіденційних даних: Використовується для шифрування важливих рядків, конфігурацій та даних, що зберігаються всередині програми.

3.3. Захист від реверс-інжинірингу: Комплексні методи, що поєднують кілька технік для ускладнення декомпіляції та подальшого аналізу коду.

4. За динамічністю:

4.1. Статична обфускація: Застосовується під час компіляції і не змінюється під час виконання програми.

4.2. Динамічна обфускація: Використовує AI для модифікації коду під час його виконання, що робить код менш передбачуваним і ускладнює його аналіз в реальному часі.

Метод обфускації за допомогою ШІ може бути класифікований як динамічна комбінована обфускація, оскільки він використовує кілька підходів одночасно і динамічно адаптується до нових загроз. AI-методи також застосовуються як для вихідного, так і для проміжного коду, що забезпечує універсальність і підвищує рівень захисту програмного забезпечення. Використання штучного інтелекту (AI) у процесі обфускації коду дозволяє значно підвищити ефективність захисту шляхом впровадження динамічної системи, яка адаптується до змінних загроз і нових методів атак. Обфускація з використанням AI включає автоматизоване застосування машинного навчання для вибору та оптимізації обфускаційних стратегій у реальному часі, на основі характеристик проміжного коду (IL). Окрім адаптивних методів обфускації, глибинне навчання може використовуватися для вставки непомітних NOP інструкцій, які не впливають на функціональність програми, але збільшують складність коду. Як зазначено в роботі [17, с. 10], використання глибокого підкріплення для оптимізації вставки NOP інструкцій значно ускладнює зворотний інжиніринг шкідливого програмного забезпечення.

Розглянемо архітектуру системи обфускації на основі штучного інтелекту. Система обфускації, яка використовує штучний інтелект, складається з кількох ключових модулів, кожен з яких відіграє важливу роль у процесі захисту коду. Ці модулі працюють в унісон для забезпечення безпеки та цілісності обфускованого коду.

Модуль введення отримує вихідний код .NET та готує його для подальшого аналізу та обфускації. Модуль попередньої обробки аналізує вихідний код, виконуючи синтаксичний та семантичний аналіз. Модуль ідентифікує критичні частини коду, що потребують додаткового захисту, такі як конфіденційні дані або вразливі ділянки, які можуть бути використані зловмисниками. Модуль обфускації на основі AI є основним компонентом системи. Використовує алгоритми машинного навчання та глибинного навчання для аналізу коду і вибору обфускаційних стратегій. Цей модуль використовує бібліотеку правил обфускації та модуль адаптивного навчання для постійного вдосконалення методик обфускації. Модуль перевірки якості забезпечує те, що обфускований код зберігає свою оригінальну функціональність та виконує тести на стійкість до реверс-інжинірингу. Модуль також перевіряє, чи зросла складність коду. Модуль звітності генерує детальні звіти про процес обфускації, включаючи інформацію про використані методики, ефективність захисту та можливі вразливості.

Взаємодія між компонентами є ключовою для ефективної роботи системи (Рис. 2). Процес розпочинається з отримання вихідного коду в модулі

введення, після чого код передається до модуля попередньої обробки для первинного аналізу. Модуль обфускації, заснований на AI, вибирає відповідні стратегії обфускації та застосовує їх до вихідного коду. Протягом усього процесу бібліотека правил обфускації та модуль адаптивного навчання забезпечують, що обфускаційні методи постійно оновлюються і вдосконалюються відповідно до нових загроз.

Після завершення обфускації, код передається до модуля перевірки якості для тестування на функціональність та стійкість до атак. Після цього результати перевірки зберігаються в модулі зберігання даних разом із обфускованим кодом. Кінцевим етапом процесу є модуль звітності, який генерує звіти, що включають опис використаних стратегій обфускації та їх ефективність. Модуль AI аналізує вихідний код та автоматично вибирає найкращі методи захисту, враховуючи специфіку коду та ймовірні загрози. Крім того, адаптивний модуль навчання постійно вдосконалює методики обфускації на основі нових даних та зворотного зв'язку, що дозволяє системі ефективно протистояти новим атакам.

Використання AI в обфускації коду має кілька важливих переваг:

1. Динамічність і адаптивність. Система може автоматично підлаштовувати свої стратегії обфускації під нові загрози.

2. Автоматизація процесу. Відсутність необхідності у ручному налаштуванні робить процес швидшим і менш схильним до помилок.

3. Поліпшена стійкість до реверс-інжинірингу. Завдяки використанню AI, код стає більш стійким до сучасних методів зворотного інжинірингу.

4. Підтримка високої функціональності. Незважаючи на підвищену складність коду, система забезпечує збереження функціональності обфускованого програмного забезпечення.

**Експеримент.** Для визначення ефективності обфускації використаєм критерії оцінювання, які описані в роботі [18, с. 10–11].

Для оцінки ефективності AI-базованого підходу був проведений експеримент з використанням 200 .dll та .exe файлів, скомпільованих за допомогою .NET 8. Файли були випадковим чином розділені на дві групи по 100 файлів: контрольна група та експериментальна. Експериментальна група використовувала обфускацію з використанням AI (RNN - TensorFlow.NET [19]), тоді як контрольна група використовувала традиційні методи. Для мануального аналізу і реверс-інжинірингу були використані інструменти [20, 21].

В результаті проведених експериментальних досліджень отримано результати, що підтверджують ефективність запропонованого підходу. За основними критеріями отримано.

1. Стійкість до аналізу. Успішні спроби реверс-інжинірингу в контрольній групі склали 18%, тоді як у експериментальній – лише 5%.

2. Зміна продуктивності. Середній час виконання програм збільшився на 6% у контрольній групі та на 8% у експериментальній.

3. Збереження функціональності. Втрата функціональності склала 2% у контрольній групі та 2,3% у експериментальній.

4. Складність патернів. Обфускація у експериментальній групі ускладнила розпізнавання патернів на 43% у порівнянні з контрольним результатом.

**Висновки.** Результати дослідження підтверджують ефективність використання AI у процесі обфускації .NET коду. Використання AI дозволяє створювати більш стійкі до зворотного інжинірингу та аналізу програми, оскільки штучний інтелект адаптується до особливостей коду і нових загроз, забезпечуючи динамічний вибір оптимальних стратегій захисту. Результати експе-

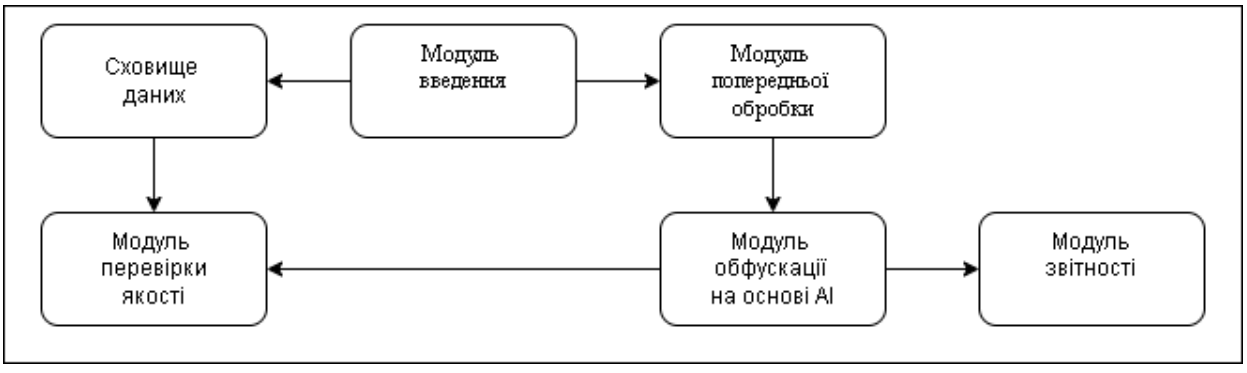


**Рис. 2. Основні компоненти інформаційної інтелектуальної системи з обфускацією**

Таблиця 2

**Критерії оцінювання методів обфускації**

| Критерій | Опис | Формула |
|---|---|---|
| Стійкість до реверс-інжинірингу | Здатність методу ускладнити процес зворотного інжинірингу та аналізу коду | $S = 1 - \frac{K}{M}$ K – кількість успішного анадізу коду, M – загальна кількість спроб |
| Вплив на продуктивність | Ступінь, до якого метод обфускації знижує продуктивність програми | $P = \frac{R - L}{L}$ R – час виконання після обфускації, L – час виконання до обфускації |
| Збереження функціональності | Наскільки добре обфускований код зберігає свою оригінальну функціональність | $F = \frac{N}{M}$ N – кількість не вірних поведінок системи після обфускації, M – загальна кількість функціоналу |
| Виявлення шаблону | Оцінює здатність автоматичних або ручних інструментів виявляти стандартні патерни (шаблони) в обфускованому коді. Чим менше патернів виявлено(більше значення D), тим краще метод обфускації приховує структуру програми, що підвищує її захищеність. | $D = 1 - \frac{N}{M}$ N – кількість виявланих шаблонів, M – загальна кількість шаблонів |

рименту показали, що AI-обфускація значно підвищує рівень безпеки програмного забезпечення без значних втрат у продуктивності та функціональності. Отже основі висноки із цього дослідженяя включають: підвищена стійкість до зворотного інжинірингу, зростання цикломатичної складності та складності патернів, автоматизація процесу обфускації.

Дослідження, проведене в цій роботі, дозволило досягти подальшого вдосконалення процесу обфускації програмного забезпечення з використанням штучного інтелекту. Основними напрямками подальшого розвитку можуть бути: розширення можливостей AI для обфускації різних типів коду, інтеграція AI-обфускації з іншими методами захисту, використання AI для виявлення нових загроз. Однією з перспективних галузей є розробка технологій динамічної обфускації, коли код обфускується в режимі реального часу під час виконання. Це дозволить змінювати стратегії обфускації в залежності від дій користувачів або спроб атак. Наприклад, система може реагувати на підозрілі дії шляхом динамічного застосування більш агресивних методик обфускації або шифрування коду. Штучний інтелект може використовуватися не тільки для обфускації, але й для прогнозування нових видів атак. Використовуючи великі масиви даних про атаки, AI може виявляти потенційні вразливості та пропонувати стратегії їх усунення до того, як атака станеться. Це допоможе створити проактивні стратегії захисту, де система самостійно виявляє загрози і адаптується під нові виклики.

**Список літератури:**

1. Collberg C., Thomborson C., Low D. A taxonomy of obfuscating transformation, The University of Auckland, Department of Computer Science, Technical Report 148, January 1997, https://www.researchgate.net/publication/37987523_A_Taxonomy_of_Obfuscating_Transformations.

2. Hameeza A., Muhammad F. H., Muhammad F. ul H., Paulo C. S. Exploring compiler optimization space for control flow obfuscation, *Computers & Security,*

Volume 139, April 2024, https://doi.org/10.1016/j.cose.2024.103704.

3. Muhammad A. S., Riyanarto S. Cyclomatic Complexity for Determining Product Complexity Level in COCOMO II, *4th Information Systems International Conference 2017, ISICO 2017,* 6-8 November 2017, Bali, Indonesia, Pages 478-486, https://doi.org/10.1016/j.procs.2017.12.180.

4. Philippe S., Cristian-Alexandru S., Michael P. Anything to Hide? Studying Minified and Obfuscated Code in the Web, *www '19: The World Wide Web Conference*, 13-17 May 2019, San Francisco CA USA, Pages 7-10, 13, https://doi.org/10.1145/3308558.3313752.

5. Sami S. A., Adnan A. Abi S., Abdallah N., Nour M. B., Ahmad B. A., Abdullah A. A Double Obfuscation Approach for Protecting the Privacy of IoT Location Based Applications, *IEEE Access,* Volume 8, 14 July 2020, Pages 129415-129431, https://doi.org/10.1109/ACCESS.2020.3009200.

6. Yunqi G., Zhaowei T., Kaiyuan C., Songwu L., Ying N. W. A Model Obfuscation Approach to IoT Security, *2021 IEEE Conference on Communications and Network Security (CNS)*, 04-06 October 2021, Tempe, AZ, USA, https://doi.org/10.1109/CNS53000.2021.9705028.

7. Minjae P., Geunha Y., Seong-je C., Minkyu P., Sangchul H. A Framework for Identifying Obfuscation Techniques applied to Android Apps using Machine Learning, *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications,* Volume 10, No 4, 31 December 2019, Pages 22-30, https://isyou.info/jowua/papers/jowua-v10n4-2.pdf.

8. Mauro C., Vinod P., Alessio V. Obfuscation detection in Android applications using deep learning, *Journal of Information Security and Applications,*
Volume 70, November 2022, https://doi.org/10.1016/j.jisa.2022.103311.

9. Wei Z., Azadeh D., Rasit Onur T. ObfusX: Routing obfuscation with explanatory analysis of a machine learning attack, *Integration,* Volume 89, March 2023, Pages 47-55, https://doi.org/10.1016/j.vlsi.2022.10.013.

10. Christian C., Giorgia S., Nicolò G. T. Enhancing Code Obfuscation Techniques: Exploring the Impact of Artificial Intelligence on Malware Detection, *Product-Focused Software Process Improvement. PROFES 2023. Lecture Notes in Computer Science,* vol 14484. Springer, 2 December 2023, Pages 80–88, https://doi.org/10.1007/978-3-031-49269-3_8.

11. Savenko O., Sachenko A., Lysenko S., Markowsky G., Vasylkiv N. BOTNET DETECTION APPROACH BASED ON THE DISTRIBUTED SYSTEMS, *International Journal of Computing*, 19(2), June 2020, Pages 190-198.

12. Lysenko S., Bobrovnikova K., Kharchenko V., Savenko O. IoT multi-vector cyberattack detection based on machine learning algorithms: traffic features analysis, experiments, and efficiency, *Algorithms* 15(7), 12 July.

13. Savenko B., Lysenko S., Bobrovnikova K., Savenko O., Markowsky G. Detection DNS Tunneling Botnets, *Proceedings of the 2021 IEEE 11th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), IDAACS'2021*, September 2021, Pages 22-25.

14. Savenko O., Nicheporuk A., Nicheporuk Y. An android malware detection method based on CNN mixed-data model, *CEUR Workshop Proceedings*, Vol. 2732, 2020, Pages. 198–213.

15. Keith D. C., Linda T. Intermediate Representations. *Engineering a Compiler (Third Edition),* 20 August 20 2022, Pages 159-207.

16. Teng H., Jiahui H., Yan P., Hongyang Y. Smart contract watermarking based on code obfuscation, *Information Sciences,* Volume 628*,* May 2023, Pages 439-448, https://doi.org/10.1016/j.ins.2023.01.126.

17. Daniel G., Matt F., Carles M., Jordi P., Quan L. Enhancing the insertion of NOP instructions to obfuscate malware via deep reinforcement learning, *Computers & Security*, Volume 113, February 2022, https://doi.org/10.1016/j.cose.2021.102543.

18. Golovko I., Savenko O., Vizhevskyi P., Klein O., Salem, A.-B.M. Obfuscation technologies of high-level source code using artificial intelligence, *CEUR Workshop, 3736*, June 2024, Pages. 324–338.

19. Machine Learning framework for .NET. URL: https://scisharp.github.io/tensorflow-net-docs/#/

20. Low-level .NET (ECMA-335) metadata reader and writer. URL: https://www.nuget.org/packages/System.Reflection.Metadatalibrary

21. Library to generate and inspect programs and libraries in the ECMA CIL format. URL: https://www.mono-project.com/docs/tools+libraries/libraries/Mono.Cecil/

**Golovko I.G., Savenko O.S., Medzatyi D.M., Ivanchenko O.V. ADVANCED TECHNIQUES IN OF SOFTWARE CODE OBFUSCATION USING ARTIFICIAL INTELLIGENCE**

*This paper explores modern approaches to .NET code obfuscation using artificial intelligence (AI) technologies. Code obfuscation is a critical tool for protecting software from reverse engineering and unauthorized access to source code. Traditional obfuscation methods, such as identifier renaming, control flow modification, and string encryption, have certain limitations as they are static and can be easily bypassed by modern decompilation tools. These methods often require significant time and resources for manual configuration and maintenance. Therefore, there is a need to implement more intelligent approaches that can automatically adapt to new threats.*

*In this work, we propose an AI-based approach to .NET code obfuscation, which involves using machine learning to analyze and select optimal obfuscation strategies. The main components of the proposed intelligent obfuscation system include an input module, preprocessing module, AI-based obfuscation module, quality verification module, and data storage module. The AI-based obfuscation module utilizes deep learning algorithms to analyze the code structure and select the most effective obfuscation techniques, ensuring a significant increase in code complexity and protection against reverse engineering.*

*The paper also presents the results of an experimental study aimed at evaluating the effectiveness of the AI-based approach to obfuscation. The experiment was conducted on a sample of 200 .dll and .exe files, compiled by MSBuild tool and divided into control and experimental groups. The obtained results showed a significant increase in resistance to reverse engineering in the experimental group compared to the control group, while maintaining high functionality and minimal impact on software performance.*

*Thus, the study confirms the effectiveness of using AI in the .NET code obfuscation process. This opens up new possibilities for creating more reliable software protection systems capable of adapting to new cyber threats and ensuring a high level of data confidentiality and integrity. The paper concludes with an outlook on further research prospects in improving learning algorithms and developing new obfuscation techniques using artificial intelligence.*

***Key words:*** *Code Obfuscation, Software Security, Software Protection, Code Complexity.*