https://doi.org/10.31891/2219-9365-2025-83-16
УДК 004.056:004.852:004.75

ГОЛОВКО Ігор
Хмельницький національний університет
https://orcid.org/0000-0002-4925-9713
e-mail: i85.golovko@gmail.com

ІНТЕЛЕКТУАЛЬНІ ПІДХОДИ ДО ЗАХИСТУ ВИХІДНОГО КОДУ

У статті розглядається інтегрована технологія захисту вихідного коду, яка поєднує традиційні методи обфускації з можливостями штучного інтелекту для оптимізації процесу захисту. Представлено методологію, що базується на аналізі проміжного коду (IL) у додатках .NET, де AI використовується для автоматичного підбору та застосування найбільш ефективних стратегій обфускації. Система реалізована з використанням сучасних інструментів для роботи з IL-кодом, таких як Mono.Cecil, у поєднанні з фреймворками машинного навчання (.NET ML, TensorFlow.NET, та ніші), що дозволяє адаптувати процес обфускації до характеристик конкретного коду.

Методологія передбачає поетапний аналіз вхідного коду, де на першому етапі проводиться синтаксичний і семантичний аналіз для виявлення критичних ділянок, які потребують посиленого захисту. Наступний етап полягає у застосуванні AI-модуля, який, використовуючи рекурентні нейронні мережі (наприклад, LSTM) та глибокі автоенкодери, що об'єднані в ансамблеві структури, дає змогу прогнозувати оптимальну стратегію обфускації для кожного сегмента коду. Інтеграція ансамблевих підходів дозволяє комбінувати прогнози кількох моделей, що значно покращує точність і стійкість системи до реверс-інжинірингу.

Проведені експерименти демонструють, що інтеграція AI значно підвищує стійкість коду до реверс-інжинірингу, зберігаючи при цьому функціональність програмного забезпечення. Стаття розглядає теоретичні засади, описує архітектуру розробленої системи та демонструє результати експериментальної перевірки запропонованого підходу, що підтверджують його ефективність у сучасних умовах розробки програмного забезпечення.

Ключові слова: централізація, багатокомп'ютерні системи, обманні методи (deceps), шкідливе програмне забезпечення, комп'ютерні атаки, приманки, пастки.

GOLOVKO Igor
Khmelnytskyi National University

INTELLIGENT APPROACHES TO SOURCE CODE PROTECTION

The article considers an integrated source code protection technology that combines traditional obfuscation methods with the capabilities of artificial intelligence to optimize the protection process. A methodology based on the analysis of intermediate code (IL) in .NET applications is presented, where AI is used to automatically select and apply the most effective obfuscation strategies. The system is implemented using modern tools for working with IL code, such as Mono.Cecil, in combination with machine learning frameworks (.NET ML, TensorFlow.NET, and niches), which allows you to adapt the obfuscation process to the characteristics of a specific code. The methodology involves a phased analysis of the input code, where at the first stage syntactic and semantic analysis is performed to identify critical areas that require enhanced protection. The next stage is the application of an AI module, which, using recurrent neural networks (e.g., LSTM) and deep autoencoders combined into ensemble structures, allows predicting the optimal obfuscation strategy for each code segment. The integration of ensemble approaches allows combining predictions from several models, which significantly improves the accuracy and resistance of the system to reverse engineering.

The experiments conducted demonstrate that the integration of AI significantly increases the resistance of the code to reverse engineering, while maintaining the functionality of the software. The article considers the theoretical foundations, describes the architecture of the developed system, and demonstrates the results of experimental verification of the proposed approach, which confirm its effectiveness in modern software development conditions.

Keywords: centralization, multicomputary systems, deceps, malicious software, computer attacks, baits, traps

Стаття надійшла до редакції / Received 20.07.2025
Прийнята до друку / Accepted 23.08.2025

ПОСТАНОВКА ПРОБЛЕМИ У ЗАГАЛЬНОМУ ВИГЛЯДІ ТА ЇЇ ЗВ'ЯЗОК ІЗ ВАЖЛИВИМИ НАУКОВИМИ ЧИ ПРАКТИЧНИМИ ЗАВДАННЯМИ

Захист програмного забезпечення в умовах цифрової економіки є одним із найважливіших завдань сучасності. Щорічне зростання кількості кібератак та вдосконалення методів зловмисників призводить до значного збільшення ризиків несанкціонованого доступу до вихідного коду програмного забезпечення. Захист інтелектуальної власності та конфіденційних алгоритмів є актуальним завданням у сучасній розробці програмного забезпечення. Зі зростанням потужності засобів реверс-інжинірингу традиційні методи обфускації виявляють свої обмеження. У цьому контексті застосування штучного інтелекту для динамічного підбору та оптимізації стратегій обфускації відкриває нові можливості у сфері захисту вихідного коду.

АНАЛІЗ ДОСЛІДЖЕНЬ ТА 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. У роботах Крістіана Колберга(Christian Collberg) і Кларка Томпсона (Clark Thomborson) [1] 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] 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 цикломатину 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. Обфускація та мінімізація часто використовуються для приховування коду у веб-додатках[3,4].

Наразі зростає увага до застосування методів машинного навчання в процесах обфускації коду. Зокрема, у роботі [5] описують процес обфускації .NET-коду з інтегрованим модулем машинного навчання, що демонструє потенціал AI-орієнтованих методів захисту коду. Даний підхід використовує ансамблеві алгоритми для вивчення закономірностей у трансформованому коді і передбачення найефективніших схем обфускації. Такий підхід дозволяє систематично оцінювати різні методи обфускації за різними показниками [6](наприклад, стійкості до аналізу і вплив на продуктивність), після чого модель пропонує оптимальні стратегії для нового коду.

Крім безпосередньо обфускації, у контексті комплексного захисту ПЗ важливе значення мають методи виявлення прихованих загроз та вірусів. Так у роботі [7] біло запропонувано багатокомп’ютерну систему виявлення з використанням метаморфічних технік, яка поєднує різні сенсори й пастки у розподіленому середовищі для протидії адаптивним вірусам. Аналогічно, у роботі [8] розробили архітектуру з частковою централізацією, де локальні вузли можуть змінювати власну топологію та алгоритми детектування, ускладнюючи роботу зловмисникам і підвищуючи ймовірність своєчасного виявлення загрози. Автори іншої роботи [9] сконцентрувалися на мобільному середовищі: вони розробили метод виявлення Android-загроз на основі конволюційної нейронної мережі з «змішаним» набором даних (API-виклики й список дозволів), що дозволяє досягти високої точності класифікації завдяки поєднанню семантичного векторного подання викликів API та бінарних ознак дозволів. А у роботі[10] розробили метод на основі машинного навчання для виявлення стеганографічних змін у зображеннях, що ілюструє можливості сучасних моделей розпізнавання прихованих патернів і підкреслює важливість захисту даних на різних рівнях представлення.

Також сучасні роботи підкреслюють важливість обфускації коду саме під час його виконання, оскільки захист програмних станів стає все більш актуальним. У роботі [11], запропонували методи динамічної обфускації для захисту станів програми під час виконання. Запропоновані авторами стратегії доповнюють попередні ШІ-орієнтовані методи статичної обфускації та окреслюють нову перспективу — обфускацію на фазі виконання. Цей підхід дозволяє не лише заплутувати вихідний код, але й ускладнювати аналіз динамічних процесів, що відбуваються під час виконання програми. Окрім цього, стаття [12] пропонує математичну модель обфускації трас виконання між потоками, що дозволяє значно ускладнити аналіз динамічного виконання багатопоточних додатків. Цей підхід є особливо актуальним для сучасних багатоядерних систем, де аналіз взаємодії потоків може розкрити критичну інформацію про логіку роботи програми. Обфускація відіграє важливу роль у захисті конфіденційних даних, зокрема в IoT додатках, де виникають значні ризики порушення приватності. У роботі [13] 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.

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. У статті [14] 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 [15] 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 [16] автори досліджують, як AI може бути використаний для посилення методів обфускації, зокрема для ускладнення виявлення шкідливого ПЗ за допомогою обфускованого коду.

В контексті дослідження обфускації для мов програмування високого рівня такі як Java та мови програмування на базі платформи .NET, які використовують проміжний код (IL/Byte code) для процесу компіляції. Проміжні представлення, як описано в роботі [17], є важливим етапом у процесі компіляції коду, 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 реверс-інжинірингу, обфускація також може використовуватися для захисту інтелектуальної власності та цифрових контрактів. У роботі [18] 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.

Незважаючи на значні успіхи традиційних методів обфускації, сучасні засоби реверс-інжинірингу дозволяють досить ефективно аналізувати захищений код. Основні проблеми, що стоять перед розробниками, можна сформулювати наступним чином:

– Обмежена адаптивність: Традиційні методи обфускації часто використовують фіксовані алгоритми, які не здатні динамічно реагувати на нові типи атак та інструменти аналізу.

– Необхідність ручного налаштування: Більшість існуючих підходів вимагають значних людських зусиль для підбору оптимальних параметрів обфускації, що збільшує ризик помилок та знижує ефективність захисту.

– Баланс між безпекою та продуктивністю: Надмірна обфускація може призводити до значного збільшення складності коду, що негативно впливає на продуктивність та підтримку програмного забезпечення.

Ще однією проблемою сучасної обфускації є також вразливість до атак на проміжний код. У середовищах, таких як платформа .NET/JAVA(мови програмування високого рівня), програми компілюються у проміжну мову IL (Intermediate Language)/Byte Code, яка легко піддається аналізу за допомогою різного роду інструментів(наприклад JetBrains dotPeek або Microsoft Ildasm тощо). Це створює загрозу для програмного забезпечення, яке містить конфіденційну інформацію або критично важливу бізнес-логіку. На рисунку 1. показано основний процес перетворення вихідного програмного коду на мові програмування високого рівня (в даному прикладі мова C#) у виконуваний машинний код.

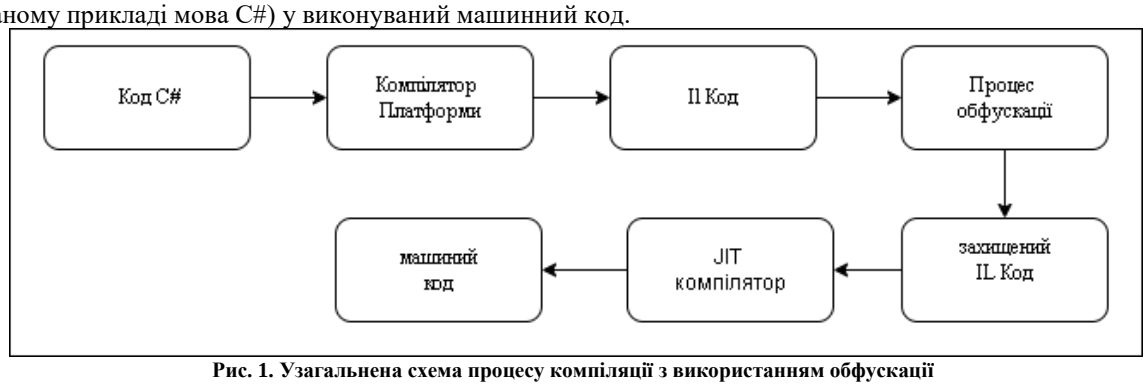

Рис. 1. Узагальнена схема процесу компіляції з використанням обфускації

Ці проблеми вимагають розробки нових технологій, які поєднують традиційні методи захисту з адаптивними можливостями штучного інтелекту для забезпечення високої стійкості вихідного коду без шкоди для функціональності додатків. 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. 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 [19],

використання глибокого підкріплення для оптимізації вставки NOP інструкцій значно ускладнює зворотний інжиніринг за допомогою програмного забезпечення. Крім того, робота [20] демонструє, як обфускація може бути використана не лише для заплутування коду, а й для приховування даних. Інтеграція концепції сховищ даних із застосуванням обфускації дозволяє додатково захищати конфіденційну інформацію, впроваджену у вихідний код, що є важливим аспектом сучасних підходів до захисту інтелектуальної власності.

ВИКЛАД ОСНОВНОГО МАТЕРІАЛУ

У системі застосовуються класичні методи обфускації, адаптовані для роботи з IL-кодом, а також розроблено адаптивну частину систему, яка за допомогою машинного навчання:

– аналізує патерни коду;
– автоматично підбирає стратегії (зміна ідентифікаторів, модифікація логіки виконання тощо);
– адаптується до нових загроз шляхом перенавчання моделі на нових даних.

Система побудована на основі моделей штучного інтелекту, які використовують ансамблеві методи машиного навчання. Ансамблеві методи комбінують прогнозування кількох моделей для досягнення більш стабільного та точного результату. Розглянемо модель, яка побудована на основі ансамблевого методу стекінг(stacking). Нехай P позначає програмний модуль (наприклад, набір IL-інструкцій, що реалізують певну функцію). Мета – перетворити P у $P' = T(P)$, де T – застосована обфускаційна трансформація, така що P' функціонально еквівалентний P, але більш захищений від аналізу. Нехай множина доступних обфускаційних перетворень $\{T_i\}$ містить різні методики (перейменування, перестановка і вставка фальшивих інструкцій, шифрування даних, розбиття математичних виразів тощо). Тоді задачу вибору трансформації можна подати як задачу багатокласової класифікації: $\phi: X \to \{1,2,\dots,N\}$, де кожному фрагменту коду $x \in X$ ставить у відповідність індекс оптимальної обфускації $T_{\phi(x)}$. Ансамбль стекінгу наближує цю невідому функцію ϕ своїм рішенням вибудовучи своє передбачення $\hat{\phi}(x)$ (наближення до істинної $\phi(x)$за допомогою мета-моделі G, яка на вході отримує результати (ознаки) від кожної з базових моделей h_i.

$$\hat{\phi}(x) = G(h_1(x), h_2(x), \dots, h_m(x)) \tag{1}$$

де $\hat{\phi}(x)$– передбачення ансамблю для фрагмента коду x, $h_i(x)$– вихід i-ї базової моделі.

Виходи $h_i(x)$слугують ознаковим описом для мета-класифікатора G. На практиці $h_i(x)$можуть бути як безпосередніми передбаченнями класів (наприклад, «LSTM передбачає, що найкраще – шифрування рядків з імовірністю 0.7»), так і неперервними ознаками (напр., «автоенкодер дав помилку відновлення 0.35, що свідчить про незвичність структури коду»). Мета-модель G може бути реалізована як, логістична регресія або невелика нейромережа, яка на виході формує розподіл ймовірностей по класах перетворень і вибирає аргмаксимум. Таким чином, формально стекінг-ансамбль реалізує відображення:

$$\hat{\phi}(x) = arg_{j \in \{1,\dots,N\}} max\ G_j(h_1(x), h_2(x), \dots, h_m(x)) \tag{2}$$

де G_j– це оцінка, яку мета-модель присвоює класу (трансформації) j(тобто міра переваги трансформації T_jдля коду x), $h_i(x)$– вихід i-ї базової моделі.

Тобто це означає значення індексу j з множини $\{1,\dots,N\}$, при якому вираз $G_j(\dots)$приймає максимальне значення. Якщо кілька j дають однакове максимальне значення, то беруть одне з них за заздалегідь узгодженим правилом (наприклад, найменше чи випадкове).

Розглянемо приклад обфускації 100 файлів різного коду на C# (.NET 8) за допомогою ансамблевого методу стекінг. В табл. 1 наведено результати експерименту обфускації коду з використанням наступних метрик:

1. Ступінь стійкості до реверс-інжинірингу (відсоток невдалих спроб декомпіляції):

$$\text{Відсоток невдач} = \frac{\text{кількість невдалих спроб}}{\text{загальна кількість спроб}} \times 100\% \tag{3}$$

Чим вище це значення, тим складніше зловмиснику отримати зрозумілий вихідний код.

2. Збереження функціональності (аналіз втрат у виконанні програм): кількість пройдених тестів / загальна кількість тестів (у %). Середній відсоток деградації продуктивності (за часом виконання або споживанням пам'яті). Припустімо, що прийнятним вважається показник до 5–10% зростання часу виконання або пам'яті.

3. Зростання цикломатичної складності: порівняння середнього відсоткового збільшення (Δ) для кожної з двох обфускованих версій.

Таблиця 1

Усереднені дані експериментів

Метрика	Оригінал	AI-обфускація за допомогою ансамблю стекінг
Відсоток невдалих спроб декомпіляції	-	96%
Прохідність тестів	100%	98.5% (±2% деградації продуктивності)
Зростання цикломатичної складності	-	+55%

ВИСНОВКИ З ДАНОГО ДОСЛІДЖЕННЯ І ПЕРСПЕКТИВИ ПОДАЛЬШИХ РОЗВІДОК У ДАНОМУ НАПРЯМІ

Таким чином, впровадження AI-модуля дозволяє більш динамічно та ефективно обфускувати код, підлаштовуючись під особливості кожного файлу, що особливо важливо для великих та різнорідних проектів. Перспективи подальших досліджень включають розширення можливостей адаптивного навчання, інтеграцію додаткових метрик безпеки та оптимізацію часу обчислень, що може забезпечити ще вищий рівень захисту програмного забезпечення. Проте виклики, пов’язані з потенційним збільшенням часу компіляції та необхідністю ретельного тестування, залишаються відкритими питаннями для подальших досліджень.

Література

1. Collberg, C., Thomborson, C., Low, D. "A taxonomy of obfuscating transformation", University of Auckland, Technical Report 148, 1997.

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. Skolka P., Staicu C.-A., Pradel M., “Anything to Hide? Studying Minified and Obfuscated Code in the Web”, in www '19: The World Wide Web Conference, San Francisco CA, 2019, pp. 1735 – 1746., https://doi.org/10.1145/3308558.3313752.

4. F. Cerutti, D. B. di San Pietro, F. Gringoli, G. Lamperti. ”Looking for Criminal Intents in JavaScript Obfuscated Code”, Procedia Computer Science, vol 207, pp 867-876, Oct 2022.

5. Igor Golovko, Oleg Savenko, Petro Vizhevskyi and Anatoliy Sachenko Obfuscation process with machine learning module // Proceedings of 2024 IEEE 14th International Conference on Dependable Systems, Services and Technologies (DeSSerT-2024, Athens, Greece, October 11-13, 2024) .

6. I. Golovko, O. Savenko, P. Vizhevskyi, O. Klein, A.-B.M. Salem. ”Obfuscation technologies of high-level source code using artificial intelligence”, CEUR Workshop, 3736, pp. 324–338, Jun 2024.

7. Kashtalian, A., Lysenko, S., Savenko, O., Nicheporuk, A., Sochor, T., & Avsiyevych, V. (2024). Multi-computer malware detection systems with metamorphic functionality. Radioelectronic and Computer Systems, 2024(1), 152-175. doi:https://doi.org/10.32620/reks.2024.1.13.

8. B. Savenko, A. Kashtalian, S. Lysenko and O. Savenko, "Malware Detection By Distributed Systems with Partial Centralization," 2023 IEEE 12th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), Dortmund, Germany, 2023, pp. 265-270, doi: 10.1109/IDAACS58523.2023.10348773.

9. Nicheporuk, A., Savenko, O., A. Nicheporuk, and Y. Nicheporuk. 2020. An android malware detection method based on CNN mixed-data model CEUR Workshop Proceedings Kharkiv, Ukraine. 2732:198–213.

10. D. Denysiuk, O. Savenko, S. Lysenko, B. Savenko and A. Kashtalian, "Method for Detecting Steganographic Changes in Images Using Machine Learning," 2023 13th International Conference on Dependable Systems, Services and Technologies (DESSERT), Athens, Greece, 2023, pp. 1-6, doi: 10.1109/DESSERT61349.2023.10416453.

11. Song X., Wang H., Wang Y., Hei X., Ji W., “Code obfuscation protection strategies for program running states”, 2024 International Conference on Networking and Network Applications (NaNA), Yinchuan City, China, 09-12 August 2024

12. Sha Z., Shu, H “Model of execution trace obfuscation between threads”, IEEE Transactions on Dependable and Secure Computing, Volume 19, NOV 2022.

13. 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.

14. 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.

15. 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.

16. 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

17. Cooper K. D., Torczon L., Engineering a Compiler, 3rd ed. Elsevier Inc, August 20 2022, 159-207.

18. 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

19. 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

20. Rajba, P.; Mazurczyk, W. "Data hiding using code obfuscation". In Proceedings of the 16th International Conference on Availability, Reliability and Security, Vienna, Austria, pp. 1–10, 17–20 August 2021

References

1. Collberg, C., Thomborson, C., Low, D. "A taxonomy of obfuscating transformation", University of Auckland, Technical Report 148, 1997.

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. Skolka P., Staicu C.-A., Pradel M., "Anything to Hide? Studying Minified and Obfuscated Code in the Web", in www '19: The World Wide Web Conference, San Francisco CA, 2019, pp. 1735 – 1746., https://doi.org/10.1145/3308558.3313752.

4. F. Cerutti, D. B. di San Pietro, F. Gringoli, G. Lamperti. "Looking for Criminal Intents in JavaScript Obfuscated Code", Procedia Computer Science, vol 207, pp 867-876, Oct 2022.

5. Igor Golovko, Oleg Savenko, Petro Vizhevskyi and Anatoliy Sachenko Obfuscation process with machine learning module // Proceedings of 2024 IEEE 14th International Conference on Dependable Systems, Services and Technologies (DeSSerT-2024, Athens, Greece, October 11-13, 2024) .

6. I. Golovko, O. Savenko, P. Vizhevskyi, O. Klein, A.-B.M. Salem. "Obfuscation technologies of high-level source code using artificial intelligence", CEUR Workshop, 3736, pp. 324–338, Jun 2024.

7. Kashtalian, A., Lysenko, S., Savenko, O., Nicheporuk, A., Sochor, T., & Avsiyevych, V. (2024). Multi-computer malware detection systems with metamorphic functionality. Radioelectronic and Computer Systems, 2024(1), 152-175. doi:https://doi.org/10.32620/reks.2024.1.13.

8. B. Savenko, A. Kashtalian, S. Lysenko and O. Savenko, "Malware Detection By Distributed Systems with Partial Centralization," 2023 IEEE 12th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), Dortmund, Germany, 2023, pp. 265-270, doi: 10.1109/IDAACS58523.2023.10348773.

9. Nicheporuk, A., Savenko, O., A. Nicheporuk, and Y. Nicheporuk. 2020. An android malware detection method based on CNN mixed-data model CEUR Workshop Proceedings Kharkiv, Ukraine. 2732:198–213.

10. D. Denysiuk, O. Savenko, S. Lysenko, B. Savenko and A. Kashtalian, "Method for Detecting Steganographic Changes in Images Using Machine Learning," 2023 13th International Conference on Dependable Systems, Services and Technologies (DESSERT), Athens, Greece, 2023, pp. 1-6, doi: 10.1109/DESSERT61349.2023.10416453.

11. Song X., Wang H., Wang Y., Hei X., Ji W., "Code obfuscation protection strategies for program running states", 2024 International Conference on Networking and Network Applications (NaNA), Yinchuan City, China, 09-12 August 2024

12. Sha Z., Shu, H "Model of execution trace obfuscation between threads", IEEE Transactions on Dependable and Secure Computing, Volume 19, NOV 2022.

13. 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.

14. 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.

15. 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.

16. 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

17. Cooper K. D., Torczon L., Engineering a Compiler, 3rd ed. Elsevier Inc, August 20 2022, 159-207.

18. 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

19. 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

20. Rajba, P.; Mazurczyk, W. "Data hiding using code obfuscation". In Proceedings of the 16th International Conference on Availability, Reliability and Security, Vienna, Austria, pp. 1–10, 17–20 August 2021