УДК 004.056.5:004.42

О.О. Щепінська
В.А. Гарнага

СИСТЕМА ЗАХИСТУ АВТОРСЬКИХ ПРАВ НА ВИКОНУВАНІ ФАЙЛИ

Вінницький національний технічний університет

Анотація

Розглянуто підхід до проєктування системи захисту авторських прав на виконувані файли, що поєднує криптографічний захист, контроль цілісності, програмну обфускацію, протидію статичному та динамічному аналізу, а також вбудовування цифрових водяних знаків. Запропоновано використання гібридного механізму, у якому частина захисних дій виконується на етапі збірки програмного продукту, а частина — під час його запуску. Особливу увагу приділено можливості ідентифікації джерела витоку програмного забезпечення за допомогою невіддільних цифрових міток, що можуть зберігатися у ресурсах, бінарній структурі файлу або структурі графа потоку керування.

Ключові слова: Виконуваний файл, захист авторських прав, обфускація, цифровий водяний знак, контроль цілісності, SHA-256, SHA-3, TLS, ліцензування програмного забезпечення.

Abstract

T*he paper considers an approach to designing a copyright protection system for executable files that combines cryptographic protection, integrity control, software obfuscation, resistance to static and dynamic analysis, and embedding of digital software watermarks. A hybrid mechanism is proposed, in which part of the protection procedures is performed at the build stage, while another part is executed at runtime. Special attention is paid to the possibility of identifying the source of software leakage using inseparable digital marks that may be stored in resources, binary file structure, or control flow graph topology.*

Keywords: Executable file, copyright protection, obfuscation, software watermarking, integrity control, SHA-256, SHA-3, TLS, software licensing.

Вступ

У сучасному світі інформаційних технологій програмне забезпечення є одним із ключових об’єктів інтелектуальної власності. Виконувані файли містять реалізовану логіку програмного продукту, алгоритми обробки даних, ліцензійні механізми та інші компоненти, які мають комерційну або науково-технічну цінність. Саме тому вони постійно перебувають під ризиком несанкціонованого копіювання, модифікації, реверс-інжинірингу та незаконного розповсюдження.

Звичні методи захисту програмного забезпечення, зокрема серійні номери, прості перевірки ліцензії або базове шифрування окремих ресурсів, не завжди забезпечують достатній рівень стійкості. Зловмисник може дослідити виконуваний файл за допомогою дизасемблерів, відлагоджувачів, емуляторів або інструментів статичного аналізу. Саме тому актуальним є створення комплексного механізму, який не лише ускладнює аналіз і модифікацію файлу, а й дозволяє встановити джерело витоку у випадку появи несанкціонованої копії.

Перспективним напрямом є поєднання кількох методів захисту: криптографічного перетворення частин коду, перевірки цілісності, програмної обфускації, захисту стаба, протидії відлагодженню, захищеного обміну з сервером ліцензування та вбудовування цифрових водяних знаків.

Основні особливості захисту виконуваних файлів у запропонованій системі наведено на рис. 1.

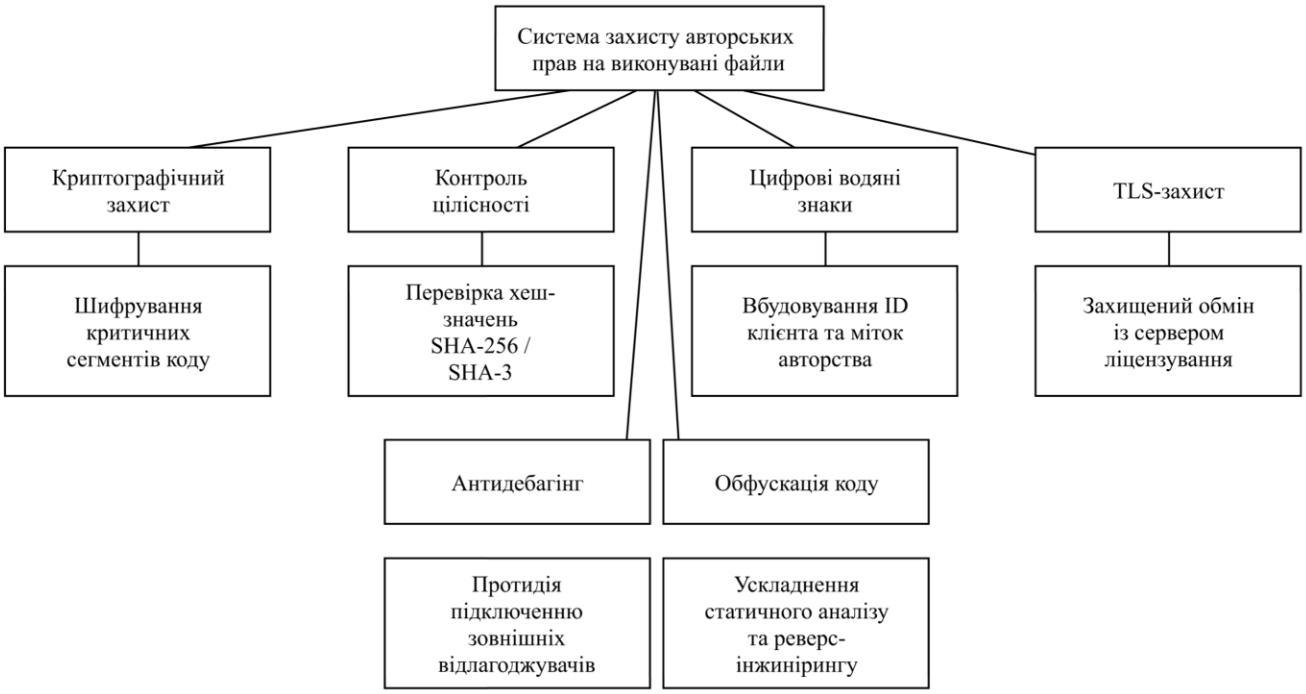

Рис 1.1 - Особливості захисту виконуваних файлів у запропонованій системі

Як видно зі схеми, запропонована система базується на поєднанні криптографічних, структурних та мережевих механізмів захисту. Такий гібридний підхід дозволяє підвищити складність аналізу виконуваного файлу та створити умови, за яких несанкціонована модифікація призводить до втрати працездатності програми.

Результати дослідження

Головна ідея системи полягає у створенні адаптивного захисту, що працює на двох рівнях: на етапі збірки програмного продукту та на етапі його виконання. Такий підхід дає змогу розділити захисну логіку між білдером і стабом, що ускладнює повне відновлення алгоритму захисту шляхом аналізу лише одного компонента системи.

На етапі збірки передбачається криптографічне перетворення критичних фрагментів виконуваного файлу. Окремі ділянки коду можуть зберігатися у зашифрованому вигляді, а їх використання здійснюється лише під час виконання програми. Для перевірки незмінності захищених сегментів доцільно застосовувати криптографічні хеш-функції, наприклад SHA-256 або SHA-3. Стандарт FIPS 180-4 визначає алгоритми сімейства SHA-2, які використовуються для формування дайджестів повідомлень та виявлення змін у даних [1]. Стандарт FIPS 202, своєю чергою, описує сімейство функцій SHA-3, побудованих на основі алгоритму KECCAK [2].

Перевірку цілісності захищеного сегмента коду можна подати у вигляді формули:

$$H = SHA256\left(C_{\text{зашфр}}\right), \tag{1}$$

де $C_{\text{зашфр}}$ – зашифрований сегмент коду, а H – еталонне хеш-значення, сформоване на етапі збірки.
Під час виконання стаб повторно обчислює значення:

$$H' = SHA256\left(C_{\text{зашфр}}\right). \tag{2}$$

Якщо виконується умова $H \neq H'$ то це сигналізує про втручання, і програма має негайно припинити роботу. Такий механізм дозволяє виявити спроби модифікації файлу, втручання у захищені ділянки коду або заміну окремих компонентів. При цьому контроль цілісності має охоплювати не лише основний виконуваний код, а й сам стаб, оскільки саме він відповідає за запуск, перевірку та розшифрування захищених фрагментів.

Окремим напрямом є впровадження цифрових водяних знаків у програмний продукт. Software watermarking використовується для підтвердження авторства, доведення факту придбання або

ідентифікації джерела нелегального розповсюдження програмного забезпечення [3]. У межах запропонованої системи цифровий водяний знак може містити ідентифікатор клієнта, номер ліцензії, дату видачі копії або інші службові метадані. Такі дані можуть вбудовуватися у ресурси виконуваного файлу, службові секції, бінарну структуру або структуру графа потоку керування.

Особливий інтерес становить метод модифікації графа потоку керування. Його суть полягає у створенні додаткових структур, які не впливають на основну функціональність програми, але формують унікальну топологічну сигнатуру. Наприклад, до програми можуть додаватися фрагменти умовно «мертвого коду», які під час побудови графа потоку керування утворюють певну послідовність вузлів і переходів. Така структура може кодувати унікальне число, пов'язане з конкретним користувачем або ліцензією. У разі витоку програмного продукту forensic-аналіз скомпрометованої копії дозволить встановити, якій саме ліцензії вона відповідала.

Основні методи, які доцільно поєднати у межах системи захисту виконуваних файлів, наведено в табл.1.

Таблиця 1 – Основні методи захисту виконуваних файлів

Метод захисту	Сутність методу	Призначення
Шифрування сегментів коду	Критичні ділянки виконуваного файлу зберігаються у зашифрованому вигляді	Ускладнення статичного аналізу та приховування логіки програми
Контроль цілісності	Обчислення та перевірка хеш-значень захищених сегментів	Виявлення модифікації коду або стаба
Цифрові водяні знаки	Вбудовування унікальних ідентифікаторів у структуру програми	Встановлення джерела витоку та підтвердження авторства
Обфускація	Перетворення коду у складнішу для аналізу форму	Ускладнення реверс-інжинірингу
Антидебагінг	Використання засобів протидії відлагодженню	Ускладнення динамічного аналізу
TLS-захист	Захищений обмін даними з сервером ліцензування	Захист ліцензійних даних від перехоплення та підміни

Для ускладнення статичного аналізу доцільно використовувати обфускацію. Вона дає змогу перетворити програму у форму, складнішу для розуміння, але зі збереженням її функціональності [4]. У контексті захисту авторських прав обфускація може застосовуватися для приховування ліцензійної логіки, ускладнення аналізу алгоритму перевірки цілісності, маскування критичних фрагментів коду та захисту цифрових водяних знаків від видалення.

Обфускацію доцільно поєднувати з криптографічним захистом, самоперевіркою, поліморфізмом стаба та контролем середовища виконання. Комерційні програмні протектори, зокрема VMProtect і Themida, демонструють ефективність комплексного підходу, який поєднує віртуалізацію, мутацію, захист від аналізу та модифікації виконуваних файлів [5, 6].

Ще одним важливим елементом є протидія динамічному аналізу. Динамічний аналіз передбачає дослідження програми під час її виконання, зокрема за допомогою відлагоджувачів. У середовищі Windows існують механізми створення процесів у режимі налагодження, що описані в документації Microsoft [7]. У межах захисної системи ця особливість може бути використана як додатковий бар'єр: стаб може створювати дочірній процес у режимі DEBUG_PROCESS і виступати для нього як керуючий відлагоджувач. Це ускладнює підключення сторонніх інструментів аналізу, наприклад OllyDbg або x64dbg.

Для захищеного обміну даними з сервером ліцензування доцільно використовувати протокол TLS. TLS 1.3 визначений у RFC 8446 і призначений для захищеної взаємодії клієнтських і серверних застосунків із протидією прослуховуванню, підміні та фальсифікації повідомлень [8]. У запропонованій системі TLS може застосовуватися для передавання ліцензійних даних, перевірки статусу користувача, оновлення параметрів захисту або підтвердження автентичності сервера.

Запропонована архітектура системи може включати білдер, стаб, модуль криптографічного перетворення, модуль контролю цілісності, модуль цифрового водяного знаку, модуль перевірки середовища виконання та модуль взаємодії із сервером ліцензування. Білдер відповідає за підготовку захищеної версії виконуваного файлу, вбудовування міток, шифрування окремих ділянок і формування еталонних хеш-значень. Стаб забезпечує запуск програми, перевірку цілісності, відновлення необхідних фрагментів коду в пам'яті та контроль умов виконання.

Перевагою такого підходу є те, що кожна захищена копія програмного продукту може бути унікальною. Навіть якщо функціональність для різних користувачів залишається однаковою, внутрішня структура виконуваного файлу, цифрові мітки та контрольні значення можуть відрізнятися. Це ускладнює створення універсального способу обходу захисту та дозволяє проводити ідентифікацію джерела витоку.

Висновки

Отже, система захисту авторських прав на виконувані файли має будуватися як комплексний механізм, що поєднує кілька взаємодоповнюючих методів. Найбільш перспективним є гібридний підхід, у якому криптографічний захист, контроль цілісності, обфускація, антидебагінг, цифрові водяні знаки та захищений обмін із сервером ліцензування працюють як єдина система.

Запропонований підхід дозволяє не лише ускладнити несанкціоновану модифікацію або аналіз програмного продукту, а й забезпечити можливість ідентифікації джерела витоку. Особливу роль у цьому відіграють цифрові водяні знаки, вбудовані у бінарну структуру файлу або граф потоку керування.

Практична цінність дослідження полягає у можливості використання запропонованих рішень для захисту комерційного, навчального або спеціалізованого програмного забезпечення від несанкціонованого копіювання, модифікації та розповсюдження. Подальший розвиток роботи може бути спрямований на програмну реалізацію прототипу системи, експериментальне оцінювання її стійкості до статичного й динамічного аналізу, а також порівняння ефективності запропонованого механізму з існуючими засобами захисту виконуваних файлів.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. NIST FIPS 180-4. Secure Hash Standard (SHS). *National Institute of Standards and Technology*. URL: https://csrc.nist.gov/pubs/fips/180-4/upd1/final (дата звернення: 27.04.2026).
2. NIST FIPS 202. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. *National Institute of Standards and Technology*. URL: https://csrc.nist.gov/pubs/fips/202/final (дата звернення: 27.04.2026).
3. Myllymaki J. A practical implementation of the SHA-1 hash function. *Electronic Commerce Research*. 2006. Vol. 6. P. 39–54. URL: https://link.springer.com/article/10.1007/s10660-006-6955-z (дата звернення: 27.04.2026).
4. Thomborson C., Collberg C. A taxonomy of obfuscating transformations. *Technical Report*. University of Auckland, 1997. URL: https://www.cs.auckland.ac.nz/~cthombor/Pubs/112393-2a.pdf (дата звернення: 27.04.2026).
5. Jakobsen C. R., Jacobsen K. L. Obfuscation of Software. *ACM Computing Surveys*. 2016. Vol. 49, No. 2. P. 1–35. URL: https://www.plai.ifi.lmu.de/publications/csur16-obfuscation.pdf (дата звернення: 27.04.2026).
6. Themida: Advanced Windows Software Protection System. *Oreans Technologies*. URL: https://www.oreans.com/Themida.php (дата звернення: 27.04.2026).
7. Debugging Functions. *Microsoft Learn*. URL: https://learn.microsoft.com/en-us/windows/win32/debug/process-functions-for-debugging (дата звернення: 27.04.2026).
8. Rescorla E. The Transport Layer Security (TLS) Protocol Version 1.3. *RFC 8446*. 2018. URL: https://datatracker.ietf.org/doc/html/rfc8446 (дата звернення: 27.04.2026).

***ЩЕПІНСЬКА* Оксана Олександрівна** - студентка групи 1БС-22б, факультету інформаційних технологій та комп'ютерної інженерії, Вінницький національний технічний університет, Вінниця, e-mail: oksiksu2005@gmail.com

***ГАРНАГА* Володимир Анатолійович** – доцент кафедри Захисту Інформації, Вінницький національний технічний університет, м. Вінниця, Україна, e-mail: garnaga.volodymyr@vntu.edu.ua

***SCHEPINSKA* Oksana Oleksandrivna** - student of group 1BS-22b, faculty of information technologies and computer engineering, Vinnytsia National Technical University, Vinnytsia, e-mail: oksiksu2005@gmail.com

***HARNAHA* Volodymyr Anatoliyovych** - associate professor of the Department of Information Protection, Vinnytsia National Technical University, Vinnytsia, Ukraine, e-mail: garnaga.volodymyr@vntu.edu.ua