**УДК 004.056 (45)**
**DOI https://doi.org/10.32689/maup.it.2023.2.8**

***Антон СИСОЄНКО***
*аспірант кафедри інформаційної безпеки та комп'ютерної інженерії, Черкаський державний технологічний університет, бульвар Шевченко, 460, Черкаси, Україна, індекс 18006 (Ampere859@gmail.com)*
**ORCID:** 0000-0002-6154-8411

***Віра БАБЕНКО***
*доктор технічних наук, професор, професор кафедри інформаційної безпеки та комп'ютерної інженерії, Черкаський державний технологічний університет, бул. Шевченка, 460, Черкаси, Україна, індекс 18006 (v.babenko@chdtu.edu.ua)*
**ORCID:** 0000-0003-2039-2841

***Світлана СИСОЄНКО***
*кандидат технічних наук, доцент, доцент кафедри інформаційної безпеки та комп'ютерної інженерії, Черкаський державний технологічний університет, бул. Шевченка, 460, Черкаси, Україна, індекс 18006 (s.sysoienko@chdtu.edu.ua)*
**ORCID:** 0000-0002-0009-337X

***Anton SYSOIENKO***
*PhD student at the Department of Information Security and Computer Engineering, Cherkasy State Technological University Shevchenko blvd, 460, Cherkasy, Ukraine,postal code 18006, (Ampere859@gmail.com)*

***Vira BABENKO***
*Doctor of Technical Sciences, Professor,Professor of the Department of Computer Information Systems and Technologies, Cherkasy State Technological University Shevchenko blvd, 460, Cherkasy, Ukraine,postal code 18006, (v.babenko@chdtu.edu.ua)*

***Svitlana SYSOIENKO***
*Ph.D., associate professor, Associate Professor, Associate Professor of the Department of Computer Information Systems and Technologies, Cherkasy State Technological University Shevchenko blvd, 460, Cherkasy, Ukraine,postal code 18006, (s.sysoienko@chdtu.edu.ua)*

**Бібліографічний опис статті:** Сисоєнко А., Бабенко В., Сисоєнко С. Особливості реалізації та застосування методів обфускації коду (Аналітичне оглядове дослідження). *Інформаційні технології та суспільство*. 2023. Вип. 2 (8). С. 69–78. DOI: https://doi.org/10.32689/maup.it.2023.2.8

**Bibliographic description of the article:** Sysoienko, A, Babenko, V. & Sysoienko, S. (2023). Osoblyvosti realizatsii ta zastosuvannia metodiv obfuskatsii kodu (Analitychne ohliadove doslidzhennia). [Features of the implementation and application of code obfuscation methods.(Analytical and review research)]. *Informatsiini tekhnolohii ta suspilstvo – Information technology and society*, 2 (8), 69–78. DOI: https://doi.org/10.32689/maup.it.2023.2.8

## ОСОБЛИВОСТІ РЕАЛІЗАЦІЇ ТА ЗАСТОСУВАННЯ МЕТОДІВ ОБФУСКАЦІЇ КОДУ (АНАЛІТИЧНЕ ОГЛЯДОВЕ ДОСЛІДЖЕННЯ)

*Стаття присвячена основним способам захисту програмного коду від дослідження, зокрема методам обфускації коду.Сучасний темп розвитку IT-індустрії зробив пріоритетним задачу підвищення безпеки програмної продукції, адже зниження ризику незаконного використання та поширення розробленого програмного продукту для розробників програмного забезпечення є надважливим аспектом діяльності. Проте, в зв'язку з розвитком інформаційних технологій завжди були і будуть залишатися невирішеними задачі піратства програмного забезпечення. Одним із способів полегшити цю проблему з технічної точки зору є використання методів захисту програмного забезпечення, особливо обфускації коду. Наведено неформальне та формальне визначення процесу обфускації. Проаналізовано основні цілі та задачі застосування методів обфускації як розробниками програмного забезпечення так і зловмисниками. Комп'ютерні технології суттєво розширили можливості як легальних користувачів програмного забезпечення, так і зловмисників щодо використання методів і засобів несанкціонованого доступу до програмних продуктів. Окремо це питання стосується байт-код орієнтованого програмного забезпечення, яке має ряд специфічних особливостей розробки та використання у сучасних комп'ютерних системах. Оскільки аналіз сучасних програмних продуктів показав тенденцію розвитку використання байт-код орієнтованого програмного забезпечення, то на демонстраційному прикладі показано реалізацію одного із методів обфускації для створеного Python-скрипта за*

*допомогою бібліотеки pyArmor. Показано, що обфускацію прийнято використовувати разом із іншими методами захисту. Зроблено висновок про те, що сумісне використання нових підходів та модифікацій існуючих технологій обфускаційних методів захисту програмного коду необхідно підсилювати за допомогою шифрування, що додасть ще один рівень захисту пристроїв і даних.*

***Ключові слова:*** *обфускація, деобфускація, програмний код, перетворення, розробники програмної продукції.*

## FEATURES OF THE IMPLEMENTATION AND APPLICATION OF CODE OBFUSCATION METHODS (ANALYTICAL AND REVIEW RESEARCH)

*The article is devoted to the main methods of protecting the software code from research, in particular the methods of code obfuscation. The modern pace of development of the IT industry has made the task of improving the security of software products a priority, because reducing the risk of illegal use and distribution of the developed software product is an extremely important aspect of activity for software developers. However, in connection with the development of information technologies, there have always been and will remain unsolved problems of software piracy. One way to alleviate this problem from a technical point of view is to use software security techniques, especially code obfuscation. An informal and formal definition of the obfuscation process is given. The main goals and objectives of the application of obfuscation methods by both software developers and attackers are analyzed. Computer technologies have significantly expanded the capabilities of both legal software users and criminals to use methods and means of unauthorized access to software products. Separately, this problem concerns the byte code-oriented software, which has a number of specific features of development and use in modern computer systems. Since the analysis of modern software products showed the development trend of using byte code-oriented software, the demonstration example shows the implementation of one of the obfuscation methods for the created Python script using the pyArmor library. It is shown that obfuscation is commonly used together with other protection methods. It was concluded that the combined use of new approaches and modifications of existing technologies of obfuscation methods of software code protection must be strengthened with the help of encryption, which will add another level of protection of devices and data.*

***Key words:*** *obfuscation, deobfuscation, program code, transformation, software developers.*

**Актуальність проблеми.** Розвиток інформатизації у суспільстві останніх років висунув на одне з перших місць проблему захисту величезної кількості інформації, що формується, оброблюється і передається в комп'ютерних системах, а також підвищення рівня безпеки програмного забезпечення щодо різних дестабілізуючих факторів. Комп'ютерні технології суттєво розширили можливості як легальних користувачів програмного забезпечення, так і зловмисників щодо використання методів і засобів несанкціонованого доступу до програмних продуктів. Окремо це питання стосується байт-код орієнтованого програмного забезпечення, яке має ряд специфічних особливостей розробки та використання у сучасних комп'ютерних системах. Аналіз сучасних програмних продуктів показав тенденцію розвитку використання байт-код орієнтованого програмного забезпечення. Це пов’язано з можливістю створення платформо-незалежного коду, а також сучасних та ефективних механізмів роботи з пам’яттю (наприклад, через використання garbage collectors). Завдяки архітектурі платформо-незалежного коду (а саме зберігання проміжного коду, який можна декомпілювати), байт-код орієнтоване програмне забезпечення вразливе до кібератак, що пов’язані з порушенням конфіденційності та автентичності [1].

Крім того, сучасний темп розвитку ІТ-індустрії зробив пріоритетним задачу підвищення безпеки програмної продукції. Адже зниження ризику незаконного використання та поширення розробленого програмного продукту для розробників програмного забезпечення є надважливим аспектом діяльності. Проте, в зв’язку з розвитком інформаційних технологій завжди були і будуть залишатися невирішеними задачі піратства програмного забезпечення. Одним із способів полегшити цю проблему з технічної точки зору є використання методів захисту програмного забезпечення, особливо обфускації коду.

**Аналіз останніх досліджень і публікацій.** Із швидким поширенням Інтернету речей (IoT) кінцеві точки як хости в мережі, які можуть отримати доступ до інших вузлів, а також, до яких можуть отримати доступ інші вузли в мережі, кожна кінцева точка потенційно є способом проникнення зловмисного програмного забезпечення у мережу.Обчислення навіддаленому хості, як правило, виконуються через посилання. Таким чином, виникає необхідність підвищення рівня безпеки програмного коду, коли код знаходиться на шляху до хосту призначення, оскільки саме в цей часвона є надзвичайно вразливою для підробки або порушення цілісності (модифікації).

В умовах розподіленої роботи, коли офісні, віддалені й гібридні працівники використовують усе більше пристроїв у різних точках світу, захищати кінцеві точки стало ще складніше.Система захисту кінцевих точок включає низку процесів, служб і рішень, які забезпечують їх захист від кіберзагроз. Першим інструментом для забезпечення кінцевих точок стало традиційне програмне забезпечення для захисту від вірусів і зловмисних програм [2].

Одним із ефективних способів захисту програмних продуктів є використання обфускаційних методів захисту програмного коду. Заплутаною (obfuscated) називається програма, рекомендована OWASP (онлайн спільнотою, яка працює в галузі безпеки веб-застосунків), що на всіх припустимих для вихідної програми вхідних даних видає той же самий результат, що й оригінальна програма, але більш складна

для аналізу, розуміння й модифікації. Загрозу становить не лише нелегальне поширення програмних продуктів, а й нелегальні модифікації зі сторони хакерів.Заплутаною програма виходить у результаті застосування щодо вихідноїпрограми перетворень, що заплутують (obfuscatingtransformations). Саме тому ряд сучасних наукових публікацій присвячені класифікації методів захищеності програмного коду, зокрема методам обфускації.

У роботі [3] проводилась розробка узагальненої класифікації обфускаційних методів захисту програмного коду, яка може дозволити розробляти надійні алгоритми захисту програмного коду від деобфускації.Обфускаційні методи дозволяють заплутувати код програми, тобто приводити вихідний текст до виду, що зберігає функціональність програми, але ускладнює аналіз, розуміння алгоритмів роботи та проведення модифікації програми.

У [4] запропонованообфускатор, який перетворює вихідний код мобільного агента у незрозумілий код. Дослідження в основному зосереджене наобфускації мобільних агентів, в той час як техніка може бути використанадля обфускації будь-якого програмного забезпечення.

Також із сумісним використанням нових підходів та модифікацій існуючих технологій обфускаційних методів захисту програмного коду необхідно постійно підсилювати безпеку кінцевих точок за допомогою шифрування [5], що додасть ще один рівень захисту пристроїв і даних.

У [6] автори пропонують метод обфускації для захисту програмного забезпечення, який забезпечує захист від зворотної інженерії. Метод базується на новій послідовності перетворень обфускації. Також розроблено програмний інструмент StiK, і на основі представленої послідовності операцій створено псевдокод для методу захисту. Проведено експериментальне дослідження за представленою методикою.

У [7] пропонується обфускатор байт-коду Micropython, заснований на обфускації потоку управління, який має перевагу в ефективності та зручності, автори реалізували та експериментували на платформі STM32L4. Результати тестування доводять, що обфускатор може значно підвищити складність злому байт-коду Micropython.

Пропонований підхід до обфускації в роботі [8] може бути використаний в програмах, які мають певну кількість підпрограм з однаковим інтерфейсом. При цьому, незалежно від складності реалізації, код кожної підпрограми може бути перетворений в деструктурований код. Особливістю запропонованого методу є використання лінійних конгруентних послідовностей в якості основи для відображення порядку розташування операторів мови на визначений функціональністю порядок виконання програми.

У роботі [1] синтезовано комплекс алгоритмів обфускації і деобфускації програмних модулів, який відрізняється від відомих урахуванням варіативності типів даних. Розроблено уніфіковану математичну модель процесу обфускації програмних модулів на базі методу графічної оцінки. В межах моделі розроблено алгоритми обфускації лексем, обфускації строкових виразів, обфускації імен ідентифікаторів та обфускації логічних виразів. Розроблено GERT-модель процесу обфускації програмних модулів тадосліджено уніфіковану GERT-модель зі зміненою кількістю вузлів, проведено оцінку якості обфускації програмних модулів. Синтезовано апарат оцінки якості обфускації програмних модулів на основі показників якості програмного продукту. Також розроблено алгоритм отримання метрик якості коду програмного продукту.

З огляду на коло досліджень щодо сучасної класифікації методів захищеності програмного коду, можна стверджувати, що впровадження засобів обфускації й досягнення мети підвищення швидкості, якості та захищеності програмних продуктів наразі є актуальною проблемою.

**Метою статті** є аналіз використання обфускаційних методів захисту програмного коду, виокремлення особливостей їх реалізації та застосування, що дозволить у майбутньому розробляти методи та інформаційні технології підтримки ефективного розроблення захищеного коду.

**Виклад основного матеріалу.** Одним із поширених сучасних способів захисту програмного коду є використання обфускаційних методів. Неформально підобфускацією розуміється наведення вихідного тексту або виконуваного коду програми до виду, що зберігає її функціональність, але ускладнює аналіз, розуміння алгоритмів роботи та модифікацію після декомпіляції [3, 8].

Головна мета процесу обфускації у тому, щоб заплутати програмний код і приховати у ньому логічні зв'язки, тобто змінити його так, щоб він був складний для вивчення сторонніми особами [8].

Деобфускація – процес, зворотний процесу обфускації, тобто. він дозволяє, наскільки це можливо, повернути код у початковий вигляд, цим самим спростити процес реверсивної інженерії [3].

Формалізацію поняття обфускації зручно представити комутативною діаграмою (рис. 1). Дана формалізація відображає результати досліджень у роботах [9, 10] в термінах теорії множин.

На рис. 1 використані наступні позначення:

P – множина програм;

X – множина вихідних даних для програм із множини P;

Y – множина результатів виконання програм із множини P із вихідними даними із множини X;

υ –відображення, що реалізує обчислювач (комп'ютер);

O – множина обфускованих програм;

θ – відображення, яке реалізується обфускатором (спеціальною програмою деякого обчислювача);

X1 – деяка підмножина множини X, 1X⊆,X на діаграмі це відношення подано відповідним подовженим знаком,

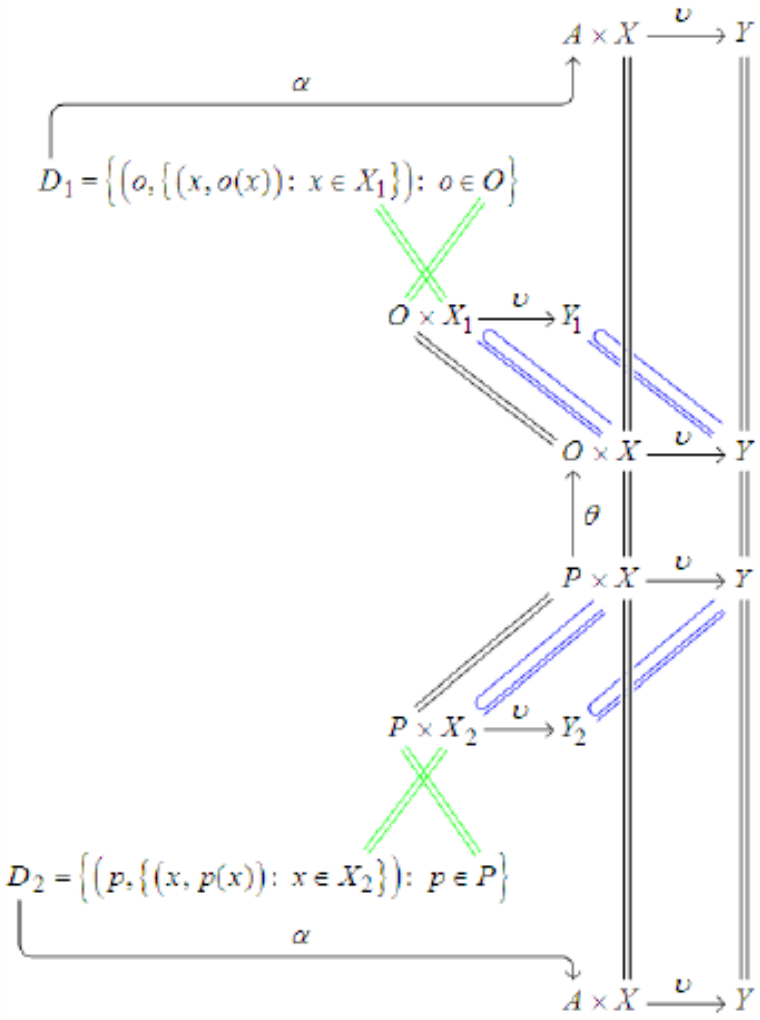


**Рис. 1. Комутативна діаграма формального подання процесу обфускації**

Y1 – підмножина множини Y, 1Y⊆,Y, що містить результати виконання обфускованих програм із множини O із вихідними даними із множини X1;

D1 – множина обфускованих програм з деякою частиною прикладів їх виконання, що визначається множиною X1, та спостерігається суб'єктом, що здійснює атаку;

A – множина реконстуйованих програм суб'єктом, що здійснює атаку, на основі дослідження елементів множини D1 (або D2) з використанням алгоритму реконструкції (рефакторингу), що представлений на діаграмі відображенням α ;

X2 – деяка підмножина множини X, 2X⊆ ;X

Y2 – підмножина множини Y, 2Y⊆,Y містить результати виконання програм із множини P із вихідними даними із множини X2;

D2 – множина обфускованих програм з деякою частиною прикладів їх виконання, що визначається множиною X2, та спостерігається суб'єктом, що здійснює атаку;

дві паралельні лінії (подовжений знак рівності) означає, що поєднання цим знаком множини – це одна і та ж сама множина, що подано синхронно, умовно, таке, що подається в кожний момент часу в кожній своїй позиції присутності одним і тим же своїм елементом.

У зв'язку з останнім зауваженням звернемо увагу на відсутність цього знака між множиною А, присутньою і у верхній, і нижній частині діаграми (це та сама множина, але представлена в кожній окремої ситуації різними своїми елементами, інакше кажучи, множина, представлена на діаграмі асинхронно).

Процес обфускації може бути здійснений на нижчому та вищому рівнях подання програмного коду: на нижчому рівні процес обфускації здійснюється над асемблерним кодом програми або безпосередньо над двійковим файлом програми, що зберігає машинний код; на вищому рівні процес обфускації здійснюється над вихідним кодом програми, написаним мовою високого рівня.

Залежно від способу модифікації коду програми розрізняють [8-13]:

– лексичну обфускацію;

– обфускування структур даних;

– обфускування потоку керування;

– превентивне обфускування.

Коротко охарактеризуємо вищезгадані методи [8-13]. Лексична обфускація – найпростіший метод обфускації, що полягає у форматуванні коду програми та модифікації його структури так, щоб він став нечитабельним, менш інформативним та складним для вивчення.

Обфускація цього виду включає: зміна коментарів на дезінформуючі або видалення в коді програми; видалення пробілів, відступів, які використовуються для кращого візуального сприйняття коду програми; заміну назв ідентифікаторів на випадкові набори символів, які людині важко сприйняти; додавання різних непотрібних операцій; зміна місць блоків програми так, щоб це не вплинуло на її здатність працювати.

Обфускація структур даних найчастіше використовується. Цей вид обфускації поділяють на три основні групи: обфускація зберігання; обфускування з'єднання; обфускування переупорядкування.

Обфускація зберігання передбачає перетворення сховищ даних та самих типів даних, наприклад створення та використання незвичних типів даних, зміна подання вже існуючих типів даних тощо.

Обфускація з'єднання ґрунтується на ускладненні представлення структур даних, що використовуються програмою. При цьому заплутування досягається шляхом з'єднання незалежних даних або розділення залежних:

Обфускація переупорядкування полягає у зміні черговості оголошення змінних, внутрішнього розміщення сховищ даних, а також у переупорядкуванні певних полів у структурах, масивах тощо.

Обфускація потоку управління [12] змінює послідовність виконання програмного коду (потоку управління). Існує кілька способів цього виду обфускації: додавання недосяжного коду; додавання «мертвого» коду; додавання надлишкового коду; «переплетення» функцій; «клонування» функцій; розгортання циклів; розпаралелювання коду; усунення бібліотечних викликів.

Превентивне обфускування призначене для запобігання використанню зловмисником деобфускаторів, декомпіляторів та інших програмних засобів деобфускації. Превентивна обфускація націлена на використання особливостей та недоліків, властивих найбільш відомим програмним засобам, які часто використовуються зловмисниками у процесі деобфускації.

Обфускація вихідного коду є особливо затребуваною для інтерпретованих мов програмування (Python, PHP, JavaScript), тобто для мов, де оператори транслюються та виконуються послідовно один за одним. Це зумовлено тим, що програми на таких мовах є скрипт-програмним сценарієм, що описує послідовність дій, що виконується інтерпретатором, а не виконуваний машинний код, що найчастіше складніше піддається аналізу, як у випадку з мовами з компілюванням (C, C++).

Для обфускації вихідних кодів, написаних на JavaScript, може використовуватися JS Obfuscator Tool [14]. Принцип його дії полягає у лексичному перетворенні, заміні імен функцій та змінних, видаленні коментарів та символів пропуску. Крім того, виконується конвертування рядків у шістнадцяткові послідовності та їх подальше кодування у base64.

Ще один обфускатор, що використовується для JavaScript, – це JS-obfuscator [15]. Дії, які він виконує, аналогічні JS Obfuscator Tool. Крім цього, він здійснює заплутування вбудованого HTML, PHP та іншого коду.

Як один із інструментів обфускації вихідних кодів на Python застосовувався «Opy» [16], що робить просту лексичну обфускацію. Він виконує заміну імен функцій та змінних на послідовності символів «l» та «1», а також перетворює рядки на набори випадкових символів. Як альтернативний інструмент використовується обфускатор Pyarmor [17], що здійснює заплутування на більш високому, ніж попередній розглянутий обфускатор. Обфускація відбувається у процесі виконання байт-коду кожного об'єкта, а очищення локальних змінних відбувається відразу після виконання функції.

Наведемо демонстраційний приклад застосування методу обфускації саме для байт-код орієнтованого програмного забезпечення. Мовою програмування Python та за допомогою бібліотек os, datetime, hashlib, tkinter, було розроблено тестову програму, що має функцію зміни дати редагування файлу і має обмежену кількість запусків.Кількість запусків записується до файлу id.pasy зашифрованому вигляді, якщо дата редагування цього файлу відрізняється від 06.12.1998, програма завершує свою роботу. На рисунку 2 наведено фрагмент лістингу створеної програми.

```
importos
importdatetime
importhashlib
importtkinterastk

defget_computer_info():
computer_info = os.environ.get('COMPUTERNAME', '') +
os.environ.get('USERNAME', '') +
os.environ.get('PROCESSOR_IDENTIFIER', '')
returncomputer_info

defencrypt_counter(counter):
computer_info = get_computer_info()
key = hashlib.sha256(computer_info.encode()).digest()
encrypted_counter = bytearray()
counter_bytes = counter.to_bytes(4, 'big')
key_len = len(key)
fori, byteinenumerate(counter_bytes):
encrypted_byte = byte ^ key[i % key_len]
encrypted_counter.append(encrypted_byte)
returnbytes(encrypted_counter)

defdecrypt_counter(encrypted_counter):
computer_info = get_computer_info()
key = hashlib.sha256(computer_info.encode()).digest()
decrypted_counter = bytearray()
key_len = len(key)
fori, byteinenumerate(encrypted_counter):
decrypted_byte = byte ^ key[i % key_len]
decrypted_counter.append(decrypted_byte)
returnint.from_bytes(bytes(decrypted_counter), 'big')
defcreate_counter_file():
filename = 'id.pas'
counter = 10
date = datetime.datetime(1998, 12, 6)
try:
ifnotos.path.exists(filename):
encrypted_counter = encrypt_counter(counter)
withopen(filename, 'wb') asf:
f.write(encrypted_counter)
os.utime(filename, (date.timestamp(), date.timestamp()))
exceptExceptionase:
print(f'Помилка: {e}')
```

**Рис. 2. Фрагмент лістингу розробленого програмного забезпечення**

Розробники скриптів знають, що код Python підтримує аналіз байт-коду, що дозволяє прискорювати роботу інтерпретатора. А сам код Python дуже складно захистити від небажаного перегляду третіми особами, тому що легко можна отримати вихідний скрипт.py з файлу.exe. Саме тому виникла необхідність реалізації засобів захисту програмного коду написаного на Python. Для цього випадку передбачена бібліотека PyArmor [17], за допомогою якої можна скористатися всіма функціями захисту скрипту від небажаного злому, і яка реалізує метод обфускації.

На рисунку 3 подано варіант з автоматизованим обфускованим кодом, що був отриманий за допомогою бібліотеки pyArmor.

```
frompyarmor_runtime_000000 import__pyarmor__
__pyarmor__(__name__, __file__,
b'PY000000\x00\x03\n\x00o\r\r\n\x80\x00\x01\x00\x08\x00\x00\x00\x04\x00\x00\x00@
\x00\x00\x00\x16\x18\x00\x00\x12\t\x04\x001\n\xce\x1a\x7f\xa6\xf2\x0fDc\xfe`\xb1
\xab\xdf\xc7\x00\x00\x00\x00\x00\x00\x00\x00|J\x15\xd0s\xa0.\xd3\xb4~\xbc\xf6\xb
d\xc64_v\xceE\xcbk@\xb9\xe5\xa0n`s\xd0\xdf1\xb7\x8b\x9c\x9aH\xc8\x11\xbf\xd6:\xa
a\xe7\xd4\xad\xd3Q\x03j\x89\x9eCXRI\xb7\xab\xc9\xe0_i\xc8\xfc\xaasc\r+\xcc\xce94
\xd2\xbf5\xe1\x19\x08\\l[}[\x05\xcf%]\xd5$n\xa9/\x02\xa0!%\xca\xa8\xdc\xc2\xe1i\
x88\x05\x1a\xde\xa0r\xb7\xbf\xf0\xfe\x07kZ\x87\x00\xb9\xc5i\xe8\x1e$\x9c<\x9bd\x
82c*\xc9\xfe\x95\x8a\xc5\xcb\xa2\x8a\xcd\xe4\x9e)\xed\x19\xc1U.B\xe1\xe5|\xcb;\x
f2Fp\xe5\xde\x1cSW\x85jJ\x91C\x8cV"Y\xd22\xf4^?nb\x15\xad
\xb01\x0fU~\xa0\x17\x85\xa4.\x18\xc7\x03,\x8a\xd6\xdb[\x172\xa4\xc5D\x1eA\x9d\xb
c\x05\x7f\x7f\xe2\xffF\xfd\xfewB\x81\xc3aD\x02u\xc4\xf8\x13\xd9K~,Q\x84.5\xe5J\x
f5\t\x9e\xce\x85\x9d\x9e\t\xbb\xac/\xee\x15\x86\xdaU2n\x8eh\x0f\x1f\xa8ap_E\xb6\
x95\x1b.W|\x0f\x1eS\xd0z_g\xf5\xbb\xd7\xf6\x94p\xe6\xa9]\xedJ\xe3\xc8\x84\xfe\xd
a?\xc5\x9a\x99|\xb2\xcd\xfa*\x94\xa6g\xdb\xf0\xf7j\xf1C\xc6\x80k\xeb\x9f\xff\x97
pUCp.\xf1h\xef\xff\xe7[\xdc1\x9e\x93$\xe5\xea\x80_\x1b8\xfaT\x05\xe8&\xc3\xa5\xc
2\\\xd6\xe4d\xcf\xeb\xf6\x14\xcf*1\x83\x95_\x11)w\x0e\x8a\x00{\xa5LZ\xc5rD\x1f\x
e1!1\xbc\xe6\x1f\x04\x85R\xc4\x07\x1eY~\xac\x92\xcf\x9f\x0b\xea\x00\x99wk\x05@K\
x04\x95En\xa9\n\x0cB53*\x027=\xf3\xba\xd0\xbb\xd3\xff\'\x828\xbf\xf7\xfd\x90\x10
\x99\x7f\xaf8."d\x9b\x98\xf8\xa7\xce0\xfc\x99\xe8
\xd2\xdb\xc4\xa2\xc4\x82.:\xab\xf1\x99O-
\x93uJq\x86\x19\x97Y\x07s]\x0c\x94\xb0ED\xb7\xf5\xc55\x12
\x03\xb3\xbe\xbde\xf9\xcb\xf1n\xaap\xe7\x86\xfc\x03\rk\xc8\x0bW7\xa9\x03a>\xed-
#?\xd6\xbfK\xca\xce\xc8\xae\xb1\xff-
8\x1a\x85x$\xede\xbd\xd6\xc1\xae\x05H\xcf\xea\xff\x1e|\xba\xa1\xc20\x98\xc5\xb2\
x9e\xa5\xde\xea3\xdc\xe3\xe2\xbe\x85\x88\xcc\xc1\xfc\xef.\xd3<\x01\x8f\x87a\xfc\
xed0\xd8."/\x9a\x81\xca\x04\xa7O\xc6a\xa7\x85\xc6\xb4\xc6\x9e\x8a\xca\xd9\xee\x9
c?u\x16d\xae\xa7)\x80\x8f\x07]\xb2\x00p\x1c\xfdAn\xac\xb4\xb1\xf6\xeaz\xb3\xa0\x
b5\xd1}\'bW\x1fJ\xd9\x1bx\xf9\xba\x10\x95\xb3\xab0\x87\x98\xea5\xc6\xc9=diB\x0f\
x87\x12\xba\x17\xe8\x0fQi\xf8\x1d\xae){\xfd\xd0\xc1BJ}\xc5\xfc\xc8\x02q\x19\x0b;
\xf1\xf6|](\xcd\x94\x03\xe4\xac|\x80\xaf\x83\x0e\xc8\x1f\xc6\x98[\xf2\xfc)\xf6\x
17\xd4`\x0f\xd0`r\xcd\xe5\xd3\xd0\x1c\x96\x13\x19\xb1TL=\x7f\x8b\x93\xe1\xfequ\x
8e\x91\xa8\xdc\xdcd\x14\xaa(D\xfa\xd0\x93\x86L{\x8e\xc0\\+r\xbbmJ\x08+\x1f\x7f\r
\xd10\x87o\x8a\x95P>_\xaa5>\xb1\x84w\xd5\x9b\xfb\xa3\x90\xcc\xa3\xf3\xea\xf4\xb5
.\xc4\x0cYD*\xc1\xcd\xc5~\xfbYS\x14\xa2\x96{\xde4\x8e,\x88H\x86,v}\xa4\x05\xf3\x
d4\xa4#_\x83\xd5\xfc-
\x96\x84\x94?\x7f\xc8\xf0`C\x04\xd5.\xed\xee\xfc\x19o\xcb\x87\xd4\x99\xfb\x92\x8
a\xfb$\x1a\xb1C\xcf\x0e\xc6\x04Q\xb6Lma\x8bx\x1d\xd4\xb1X\xdd*\xe4\xd9\xa2\xdf!\
x91\xbb\x04\xb4I+\x8c\xd9\xdbSY\x1c\xdc\xb9\xa4\t\xd4b\xd0>6\xf2\x99\x1a\xd9X\x1
e+\xfa\xfc\x8c\xe3\x017\xa8\xcd&3\xf4\xfd\xa8\x98\xc2\xe3a\x8b\xa8\x8dE\xef\x03i
\xa9\xee\xc2\xa5w\xed16N`\xee\x02!\x06n\xbc\xc0>~\x84\xd3\x97d\xc4\x96\xc6\x1c\x
01\x86\\\x15kGY\xbfQ\xd9\xf0!q\x19\x9b\xd3\x7f\xd4\xd9\xd3)m\'\x8c\xb38O\xb1\x86
<c9\xceU\xbc\xb0\x05\x1aJr\x03\xf5\x19\xb6\xa1\xffVa\xdar\xe5\xcc\xd9AF\xfc\xab\
xd1\xa8\xd1\x8a\xef\xb3\'\x91\xcb\xa6\xdf\xee\x84-
\x93\x9b\xaf\x11\x83v\xcd\t:XC\xb6I\xcbW\xa4\xd5)\xb1Z\xdeo\xb1TC0y\xf0lZ\xdak\x
ca\x885\xc9\xc6\xbc\xe6;\x8e\xf4\xca\x8d\x83\x1ee\xe4>\x85}w\xf0\x99\x90\xbc\x9a
\x9b[%\x0c\x9d\xdf7\xed\x92yj;\xdf\xa2~L4\x94Is\xff\x03\x06<\xb3\nu\xbf\xee-
\x9f@0\xe2\xf6\xdf\x9b<,\x1e`jZI\xd2\xe1\xb8\x98\xae\xc1\x8e\xc0\xfc\xf9\xfe\xd7
2\x13\xa9\x853E\xe1\xa5f+\xc8B\xa27\x94#\x12\xb7\xb1\x04K\xf2`\xbf\xd3\xfb(\xc6\
xc9Zq\xa8\xa09\xa4S\xfee\xbd\x83\x91Q]
```

**Рис. 3. Фрагмент лістингу обфускованого коду програми, що був згенерований за допомогою pyArmor**

Сучасними методами підвищення безпеки програмного коду [3, 6, 8], зокрема методами обфускації, цікавляться не лише розробники програмних продуктів. Великий інтерес до цього методу захисту програмного коду виявляють і зловмисники, які мають на меті якомога довше залишатися непоміченими при скоєнні комп'ютерних атак, заснованих на використанні шкідливого програмного забезпечення, наприклад, вірусних програм. Проаналізуємо цілі, які ставлять перед собою зловмисники, а також розробники ПЗ, та задачі, які вони можуть реалізувати, використовуючи методи обфускації та деобфускації.

Серед основних цілей застосування методів обфускації зловмисниками можна виокремити такі [3, 6, 8-17]:

1. Приховування сигнатур вірусного програмного забезпечення для затруднення його виявлення антивірусними комплексами. Часто більшість антивірусів використовують такі прийоми виявлення, як розпізнавання вірусів за їх сигнатурою, виявлення аномалій поведінки програм та інше. Так, наприклад, метод виявлення вірусу по сигнатурах корисний тільки в тому випадку, коли чергова сигнатура,

що знайшлася, за принципом дії практично не відрізняється від шаблонної. Для обходу такого способу виявлення антивірусом може використовуватися обфускація: при кожній новій реплікації вірус використовує обфускаційні перетворення, які дозволяють отримувати на виході різні за структурою модифікації одного і того ж вірусу, які не розпізнаються антивірусом як та сама сигнатура вірусної програми.

2. Захист програмного коду від аналізу з метою приховування вірусного програмного забезпечення.

Оскільки, майже усі цілі деобфускації безпосередньо залежать від цілей обфускації, то основні цілі застосування деобфускаторів розробниками програмного забезпечення такі:

1. Виявлення антивірусами сигнатур вірусного ПЗ. Як було зазначено вище, за допомогою методів обфускації можна приховати сигнатури вірусного ПЗ у вигляді зміни його структури під час проходження чергової реплікації. Якщо антивірус для виявлення сигнатур буде використовувати засоби деобфускації, ймовірність виявлення сигнатур вірусного програмного забезпечення зросте, отже, зросте стійкість антивірусу;

2. Дослідження програми щодо наявності у ньому шкідливого коду. Деобфускація шкідливого коду та його аналіз може проводитись фахівцями із захисту інформації.

Серед основних цілей застосування методів деобфускації зловмисниками можна виокремити такі [3, 6, 8]:

1. Деобфускація програмного коду продукту з метою незаконного копіювання, зміни, використання у власних цілях вихідного коду програми. Зловмисник після застосування деобфускаторів над обфускованим кодом може використовувати його з наступною метою:

– модифікувати код, тобто внести до програми зміни, які можуть призвести до суттєвих змін або повного блокування алгоритму роботи програми;

– скопіювати код та присвоїти собі авторські права;

– скопіювати та розповсюдити програмний продукт безкоштовно або за плату меншу, ніж плата, оголошена розробником (піратство);

2. Оптимізація коду. І деобфускація, і оптимізація програмного коду тією чи іншою мірою протилежні процесу обфускації. Оскільки у процесі обфускації в програмний код часто здійснюється додавання зайвих операцій, які зазвичай ніяк не впливають на результати роботи самої програми та призначені для утруднення процесу вивчення програмного коду сторонніми особами, то процес деобфускації можливо використовувати з метою оптимізації коду. Крім того, потрібно розуміти, що під час компіляції програмного коду здійснюється і і його оптимізація. Отже, слід зазначити, що більшість компіляторів у процесі компіляції вихідного коду автоматично здійснюють процес оптимізації, тому, якщо процес обфускації здійснюється над вихідним кодом програми (обфускація високого рівня), виникає певна ймовірність того, що її ефективність після процесу компіляції знизиться. Якщо ж такий вихідний код буде підданий обробці інтерпретатором (тобто не буде схильний до компіляції), ефективність виконаного процесу обфускації не зміниться [16, 17].

Як бачимо, застосування методів обфускації коду можуть мати як позитивні так і негативні наслідки щодо програмного забезпечення в залежності від того, хто та з якою ціллю їх використовує.

**Висновки та перспективи подальших досліджень.** Методи захисту програмного забезпечення від дослідження з метою реалізації деструктивних впливів необхідно вивчати для того, щоб розуміти як зловмисник може приховувати шкідливі програми в атакованому програмному коді і, навпаки, розробник програмного забезпечення може, застосовуючи такі методи, захистити програми від зловмисника, який намагається впровадити в них свої шкідливі програми.

Одним із ефективних способів захисту програмних продуктів є використання обфускаційних методів захисту програмного коду.Однак потрібно розуміти, що одна лише обфускація не забезпечує ефективного захисту програм, оскільки вона не запобігає незаконному використанню програмного продукту. Тому обфускацію прийнято використовувати разом із іншими методами захисту, адже проблема захисту інтелектуальної власності набуває сьогодні неабиякої актуальності.

У статті здійснено аналітичний огляд методів обфускації програмного коду, виокремлені особливості їх реалізації та застосування залежно від цілей їх використання. Враховуючи отримані результати проведеного аналізу, розроблені рекомендації щодо технологій застосування обфускації коду. Зокрема розглянуті варіанти спільного використання методів обфускації з іншими методами захисту програмного забезпечення від дослідження та аналізу з метою зниження ризику виникнення та реалізації загроз безпеці розробленого програмного продукту. В перспективі в подальших дослідженнях пропонується модифікувати реалізацію алгоритму одного із розглянутих методів обфускації на основі використання операцій криптографічного кодування для підсилення безпеки кінцевих точок за допомогою шифрування, що додасть ще один рівень захисту пристроїв і даних. Також дане дослідження спонукає у майбутньому розробити методи та інформаційні технології підтримки ефективного розроблення захищеного коду.

**Список використаних джерел:**

1. Давидов В.В. Моделі та методи підвищення безпеки байт-код орієнтованого програмного забезпечення в умовах кібератак : дис. д-ра техн. наук : 05.13.05 / Давидов Вячеслав Вадимович. Харків, 2021. 313 с. Режим доступу: https://er.chdtu.edu.ua/handle/ChSTU/2577

2. Важливість системи захисту кінцевих точок URL: https://www.microsoft.com/uk-ua/security/business/security-101/what-is-an-endpoint

3. Stepanenko I., Kinzeryavyy V., Nagi A., Lozinskyi I. Modern obfuscation methods for secure coding. *Ukrainian Scientific Journal of Information Security.* 2016, vol. 22 (1), issue 1, p. 32-37. Режим доступу: doi.org/10.18372/2225-5036.22.10451

4. Garg, V., Srivastava, A., Mishra, A. Obscuring mobile agents by source code obfuscation. *Int. J. Comput. Appl., 2013.* 61(9), 46–50.

5. Svitlana Sysoienko, Iryna Myronets, Vira Babenko. Practical Implementation Effectiveness of the Speed Increasing Method of Group Matrix Cryptographic Transformation. *CEUR Workshop Proceedings 2353*. 2019. p.402 – 412. (Scopus). ISSN 1613-0073. Режим доступу: doi.org/10.32782/cmis/2353-32

6. Gnatyuk, S., Kinzeryavyy, V., Stepanenko, I., Gorbatyuk, Y., Gizun, A., Kotelianets, V. Code Obfuscation Technique for Enhancing Software Protection Against Reverse Engineering. In: Hu, Z., Petoukhov, S., He, M. (eds) Advancesin Artificial Systems for Medicine and Education II. AIMEE2018 2018. *Advances in Intelligent Systems and Computing*, 2020. Vol 902. pp. 571–580. Springer, Cham. Режим доступу: https://doi.org/10.1007/978-3-030-12082-5_52

7. Lantao Wang; Yun Li; Haitao Zhang; Qigu Han; Lirong Chen. (2021). An Efficient Control-flow based Obfuscator for Micropython Bytecode. 7th International Symposium on System and Software Reliability (ISSSR), 23-24 September 2021. Chongqing, China. *INSPEC Accession Number*: 21481418. Режим доступу: doi: 10.1109/ISSSR53171.2021.00028

8. Бондарчук А.П., Корнага Я.І., Базалій М.Ю., Сергієнко П.А., Ільїн О.Ю. Метод захисту програмного коду від аналізу засобами обфускації. *Телекомунікаційні та інформаційні технології*. 2020. № 4 (69). С. 140-148. Режим доступу: doi: 10.31673/2412-4338.2020.045051

9. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., Yang, K. On the (im)possibility of obfuscating programs. *Journal of the ACM.* 2012. Vol. 59. Iss. 2. Article No. 6. 48 p. Режим доступу: DOI:10.1145/2160158.2160159.

10. Goldwasser, S., & Guy, N. R. On best-possible obfuscation. *Fourth IACR Theory of Cryptography Conference*, TCC 2007, February 21-24 2007. Amsterdam: KNAW Trippenhuis, 2007. P. 194-213.

11. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., Yang, K. On the (Im)possibility of Obfuscating Programs. *LNCS*, 2010. P. 1-18.

12. Chow, S., Gu, Y., Johnson, H., Zakharov, V. An approach to the obfuscation of control-flow of sequential computer program. *LNCS*, 2001. P. 144-155.

13. Garg S., Gentry C., Halevi S., Raykova M., Sahai A., and Waters B. Candidate indistinguishability obfuscation and functional encryption for all circuits. FOCS, 2013. P. 22-23.

14. JS Obfuscator Tool. URL: https://obfuscator.io/

15. JS-obfuscator. URL: https://github.com/caiguanhao/js-obfuscator

16. Opy. URL: https://github.com/QQuick/Opy

17. Pyarmor. URL: https://github.com/dashingsoft/pyarmor

**References:**

1. Davydov, V.V. (2021). Modeli ta metody pidvyshchennia bezpeky bait-kod oriientovanoho prohramnoho zabezpechennia v umovakh kiberatak [Models and methods of increasing the bytecode-oriented software security of during cyberattacks]. Doctor's thesis. Kharkiv. Retrieved from https://er.chdtu.edu.ua/handle/ChSTU/2577 [in Ukrainian].

2. Vazhlyvist systemy zakhystu kintsevykh tochok. [The importance of an endpoint security system]. Retrieved from https://www.microsoft.com/uk-ua/security/business/security-101/what-is-an-endpoint [in Ukrainian].

3. Stepanenko I., Kinzeryavyy V., Nagi A., Lozinskyi I. Modern obfuscation methods for secure coding. *Ukrainian Scientific Journal of Information Security.* 2016, vol. 22 (1), issue 1, p. 32-37. Retrieved from doi.org/10.18372/2225-5036.22.10451 [in English].

4. Garg, V., Srivastava, A., Mishra, A. (2013). Obscuring mobile agents by source code obfuscation. *Int. J. Comput. Appl.* 61(9), 46–50 [in English].

5. Svitlana Sysoienko, Iryna Myronets, Vira Babenko. (2019). Practical Implementation Effectiveness of the Speed Increasing Method of Group Matrix Cryptographic Transformation. *CEUR Workshop Proceedings 2353*. p.402 – 412. Retrieved from https://ceur-ws.org/Vol-2353/paper32.pdf [in English].

6. Gnatyuk, S., Kinzeryavyy, V., Stepanenko, I., Gorbatyuk, Y., Gizun, A., Kotelianets, V. (2020). Code Obfuscation Technique for Enhancing Software Protection Against Reverse Engineering. In: Hu, Z., Petoukhov, S., He, M. (eds) Advancesin Artificial Systems for Medicine and Education II. AIMEE2018 2018. *Advances in Intelligent Systems and Computing*, vol 902. pp. 571–580. Springer, Cham. Retrieved from https://doi.org/10.1007/978-3-030-12082-5_52 [in English].

7. Lantao Wang; Yun Li; Haitao Zhang; Qigu Han; Lirong Chen. (2021). An Efficient Control-flow based Obfuscator for Micropython Bytecode. 7th International Symposium on System and Software Reliability (ISSSR), 23-24 September 2021. Chongqing, China. *INSPEC Accession Number*: 21481418. Retrieved from https://doi: 10.1109/ISSSR53171.2021.00028 [in English].

8. Bondarchuk, A.P., Kornaha, Ya.I., Bazalii, M.Yu., Serhiienko, P.A., Ilin, O.A. (2020). Metod zakhystu prohramnoho kodu vid analizu zasobamy obfuskatsii [Method of protecting software code from analysis by obfuscation means]. *Telekomunikatsiini ta informatsiini tekhnolohii – Telecommunication And Information Technologies,* 4(69), 140-148. Retrieved from https://doi: 10.31673/2412-4338.2020.045051 [in Ukrainian].

9. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., Yang, K. (2012). On the (im) possibility of obfuscating programs. *Journal of the ACM.* Vol. 59. Iss. 2. Article No. 6. 48 p. Retrieved from https://doi: 10.1145/2160158.2160159 [in English].

10. Goldwasser, S., & Guy, N. R. (2007). On best-possible obfuscation. *Fourth IACR Theory of Cryptography Conference*, TCC 2007, 194-213 [in English].

11. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., Yang, K. (2010). On the (Im)possibility of Obfuscating Programs. *LNCS*, 1-18 [in English].

12. Chow, S., Gu, Y., Johnson, H., Zakharov, V. (2001). An approach to the obfuscation of control-flow of sequential computer program. *LNCS*, 144-155 [in English].

13. Garg S., Gentry C., Halevi S., Raykova M., Sahai A., and Waters B. (2013). Candidate indistinguishability obfuscation and functional encryption for all circuits. FOCS, 22-23 [in English].

14. JS Obfuscator Tool. Retrieved from https://obfuscator.io/ [in English].

15. JS-obfuscator. Retrieved from https://github.com/caiguanhao/js-obfuscator [in English].

16. Opy. Retrieved from https://github.com/QQuick/Opy [in English].

17. Pyarmor. Retrieved from: https://github.com/dashingsoft/pyarmor [in English].