УДК 004.056.52 DOI: 10.31673/2412-4338.2019.047180

Семенов С.Г., Давидов В.В., Волошин Д.Г., Гребенюк Д.С.
Національний технічний університет «Харківський політехнічний інститут», Харків

МЕТОД ЗАХИСТУ МОДУЛЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ НА ОСНОВІ ПРОЦЕДУРИ ОБФУСКАЦІЇ

Мета статті: дослідження та розробка методу обфускації коду програмного модуля ліцензування з використанням особливостей презентації строкових виразів, механізму викликів функцій та доступу до ідентифікаторів в байткод-орієнтованих мовах програмування. ***Результати****. У статті наведена класифікація мов програмування. Доведено доцільність досліждення саме байткод-орієнтованих мов програмування (на базі Java Virtual Machine, Common Language Runtime тощо), що в сучасному світі використовуються для створення Enterprise-застосунків. З основних послуг інформаційної безпеки для аналізу було обрано конфіденційність, що забезпечується шляхом використання процедури обфускації. Розглянуто існуючі методи захисту програмного продукту на основі процедури обфускації. Розглянуто їхні недоліки та запропоновано методи на основі процедури обфускації, що працюють з ідентифікаторами, а не змінюють алгоритм виконання програми. Проведені дослідження дозволили сформулювати завдання для створення методів обфускації. Таким чином, було розроблено 2 засоби обфускації. По-перше, це обфускація строкових літералів з використанням особливостей генерації псевдовипадкових чисел, що дає можливість описати одну й ту саму строкову константу різними значеннями. По-друге, це обфускація імен ідентифікаторів. Об'єднання розроблених двох методів дозволило створити метод, що обфускує ідентифікатори таким чином, що кожний доступ до ідентифікатора є унікальним. Це позбавляє зловмисника можливості в стислі терміни знайти відповідні імена ідентифікаторів для аналізу алгоритму роботи модуля.* ***Висновки****. Проведено експеримент, в якому ІТ-фахівцям було запропоновано проаналізувати необфускований та обфускований програмний код розробленого модуля програмного забезпечення, основна мета якого – генерація та верифікація ліцензійного ключа. Результати показали, що для аналізу обфускованого коду зловмиснику буде необхідно до 5 разів більше часу для аналізу коду. Це доводить доцільність розроблених методів.*

Ключові слова*: захист програмного забезпечення, обфускація, байткод-орієнтовані мови програмування.*

Semenov S.G., Davydov V.V, Voloshyn D.G., Hrebeniuk D.S.
National Technical University "Kharkiv Polytechnic Institute", Kharkiv

SOFTWARE MODULE PROTECTION METHOD BASED ON OBFUSCATION PROCEDURE

Purpose of the article: research and development of the software licensing module code obfuscation method with usage of presentation the string expressions features, function calls mechanism and access to identifiers in the bytecode-oriented programming languages. Results. The classification of programming languages is presented. The expediency of bytecode-oriented programming languages (based on JVM, CLR, etc.) that are used in the modern world to create Enterprise applications research is proved. Privacy, which is ensured through the obfuscation procedure usage, was selected for analysis from the main information security services. Existing software product protection methods based on the obfuscation procedure were considered. Their limitations were considered and the methods based on the obfuscation procedure, which work with identifiers and do not change the algorithm of program execution, were proposed. The provided researches made it possible to formulate the task for obfuscation methods development. Thus, two

obfuscation methods were developed. Firstly, it is the obfuscation of string literals using the features of pseudorandom number generation, which makes it possible to describe the same string constant with different values. Secondly, it is an obfuscation of identifier names. The combination of this two developed methods allowed to create a tool that obfuscates the identifiers in such a way that each access to the identifier is unique. This takes away the opportunity of the attacker to find the appropriate identifier names in a short time to analyze the module's algorithm. Conclusions. An experiment was conducted in which IT-specialists were offered to analyze the unobfuscated and obfuscated program code of the developed software module, the main purpose of which was the license key generation and verification. The results showed that it would take up to 5 times more time to analyze the obfuscated code. This proves the advisability of the developed methods.

***Keywords**: software protection, obfuscation, bytecode-oriented programming languages.*

Семенов С.Г., Давыдов В.В., Волошин Д.Г., Гребенюк Д.С.
Национальный технический университет «Харьковский политехнический институт», Харьков

СПОСОБ ЗАЩИТЫ МОДУЛЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ НА ОСНОВЕ ПРОЦЕДУРЫ ОБФУСКАЦИИ

Цель статьи: исследование и разработка метода обфускации кода программного модуля лицензирования с использованием особенностей презентации строковых выражений, механизма вызовов функций и доступа к идентификаторам в байткод-ориентированных языках программирования. Результаты. В статье приведена классификация языков программирования. Доказана целесообразность исследования именно байткод-ориентированных языков программирования (на базе Java Virtual Machine, Common Language Runtime и т.д.), в современном мире используемых для создания Enterprise-приложений. Из основных услуг информационной безопасности для анализа была выбрана конфиденциальность, которая обеспечивается путем использования процедуры обфускации. Рассмотрены существующие методы защиты программного продукта на основе процедуры обфускации. Рассмотрены их недостатки и предложены методы на основе процедуры обфускации, которые работают с идентификаторами, а не меняют алгоритм выполнения программы. Проведенные исследования позволили сформулировать задачи для создания методов обфускации. Таким образом, были разработаны 2 способа обфускации. Во-первых, это обфускация строковых литералов с использованием особенностей генерации псевдослучайных чисел, что дает возможность описать одну и ту же строковую константу различными значениями. Во-вторых, это обфускация имен идентификаторов. Объединение разработанных двух методов позволило создать метод, который обфусцирует идентификаторы таким образом, что каждый доступ к идентификатору является уникальным. Это лишает злоумышленника возможности в сжатые сроки найти соответствующие имена идентификаторов для анализа алгоритма работы модуля. Выводы. Проведен эксперимент, в котором IT-специалистам было предложено проанализировать необфусцированный и обфусцированный программный код разработанного модуля программного обеспечения, основная цель которого – генерация и верификация лицензионного ключа. Результаты показали, что для анализа обфусцированного кода злоумышленнику потребуется до 5 раз больше времени для анализа кода. Это доказывает целесообразность разработанных методов.

***Ключевые слова**: защита программного обеспечения, обфускация, байткод-ориентированные языки программирования.*

1. Вступ

В сучасному світі з підвищенням кількості атак на програмне забезпечення (ПЗ) все більшої актуальності набуває питання його захисту [1]. Для аналізу питання захисту ПЗ необхідно виявити основні завдання інформаційної безпеки. До них належать [2]:

– конфіденційність;

– цілісність;

– доступність;

– невідмовність.

В межах даної роботи присвячується увага конфіденційності. З точки зору розробки програмного забезпечення, конфіденційність забезпечується шляхом використання механізму обфускації.

Аналіз літератури показав, що для ПЗ, написаного на різних мовах програмування, існують різні механізми обфускації. Також, аналіз літератури показав, що мови програмування можна поділити на 4 групи [3]:

– скриптові (наприклад, Javascript, PHP) та ті, що інтерпретуються (наприклад, PHP, Perl, Ruby, Python). Дана категорія мов програмування характеризується тим, що програмний продукт поставляється у відкритому вигляді та обфускація виконується для початкового коду (source code);

– ті, що компілюються (наприклад, C, C++, Delphi, Assembler). Дана категорія мов програмування характеризується тим, що програмний продукт поставляється у вигляді бінарних (скомпільованих) файлів (файли, що виконуються, бібліотеки), які: 1) важко декомпілювати (перетворювати назад до початкового коду); 2) мають можливість вбудовування системно- та процесорно-специфічних трюків, що дозволяє підняти якість обфускованості коду на високий рівень. Незважаючи на свої беззаперечні переваги, дана категорія мов має наступні недоліки: 1) дороговизна підтримки коду, 2) відсутність кросплатформеності, 3) витіснення мовами програмування з проміжним кодом з ринку прикладних програмних продуктів;

– байткод (наприклад, Java, C#). Ця категорія потребує особливої уваги, так як програмні продукти, які написані на мовах цієї категорії, мають «проміжний» код відповідної віртуальної машини (JVM, CLR), який близький до компільованого, але простіше декомпілюється. Так як проміжний код виконується певною відповідною віртуальною машиною, то використання хитрощів, як в компільованому коді, не є можливим.

Аналіз літератури [4, 5] показав, що саме байткод-орієнтовані мови програмування мають найвищий рівень популярності серед інших. До того ж, саме з використанням цих мов програмування розробляються Enterprise-застосунки, що формують найбільшу частину фінансової складової ринку IT-індустрії [6] у зв'язку з тим, що вартість розробки та підтримки може в 100 разів перевищувати розробку простих застосунків та сайтів.

Висока вартість програмного забезпечення потребує:

– підвищеного рівня безпеки самого програмного забезпечення для уникнення витоку даних користувача та корпорації;

– підвищеного рівня безпеки модуля авторизації програмного забезпечення, зокрема, модуля ліцензування. У зв'язку з тим, що вартість ліцензій може сягати сотні тисяч доларів, гостро стоїть проблема захисту авторських прав компаній-розробників програмного забезпечення. Одним з напрямів захисту програмного забезпечення та авторських прав на нього є обфускація.

Таким чином, дослідження методів обфускації програмного забезпечення, яке написане на байткод-орієнтованих мовах програмування, є актуальним та потребує проведення досліджень в цьому напрямку.

2. Аналіз літературних даних і постановка проблеми

Аналіз літератури [7-12] показав, що методи обфускації, які застосовуються в байткод-орієнтованих мовах програмування, можна поділити на такі типи:

– вставка мертвого коду. Цей код або нічого не робить, або до нього не доходить процес керування. Цей метод має можливість сконцентрувати увагу зловмисника на аналізі

коду, що не має сенсу з точки зору бізнес-логіки. Це підвищує складність та дає можливість збільшити час на аналіз коду в цілому;

– лексичні перетворення. Цей тип найпримітивніший та включає до себе видалення символів, що не відносяться до коду (наприклад, коментарі, відступи);

– зміна послідовності виконання коду;

– заміна інструкцій на еквівалентні вирази. До цього засобу можна віднести заміну ідентифікаторів на довільний набор символів;

– фрагментація даних. Наприклад, код

System.out.println("48656c6c6f20576f726c6421");

замінити на

System.out.println(“48”,”65en”,”6c”,”6c(fd”,”6f”,”2054”,
”57g”,”6f5h”,”72 __t”,”6c”,”64’h”,”21”).

З точки зору мови програмування Java, функція System.out.println у даному випадку буде виводити лише по 2 символи кожної строки, що дасть нам ідентичний результат;

– кодування/шифрування коду тексту алгоритму (олігоморфічне, поліморфічне, метаморфічне).

Аналіз показав, що в більшості випадків увага приділяється обфускації процесу захисту алгоритму, та практично не приділяється захисту самих даних. У даній роботі здійснено спробу розв’язання цієї проблеми.

3. Мета і задачі дослідження

Метою дослідження є аналіз існуючих методів обфускації та розробка методу обфускації коду програмного модуля ліцензування з використанням особливостей презентації строкових виразів, механізму викликів функцій в байткод-орієнтованих мовах програмування.

Для досягнення поставленої мети вирішуються наступні задачі:

– аналіз існуючих методів обфускації, основним недоліком яких є спрямованість на обробку послідовності дій, а не на дані та їх зберігання;

– розробка алгоритму обфускації ідентифікаторів і презентації строкових констант.

4. Основна частина

4.1 Обфускація строкових виразів

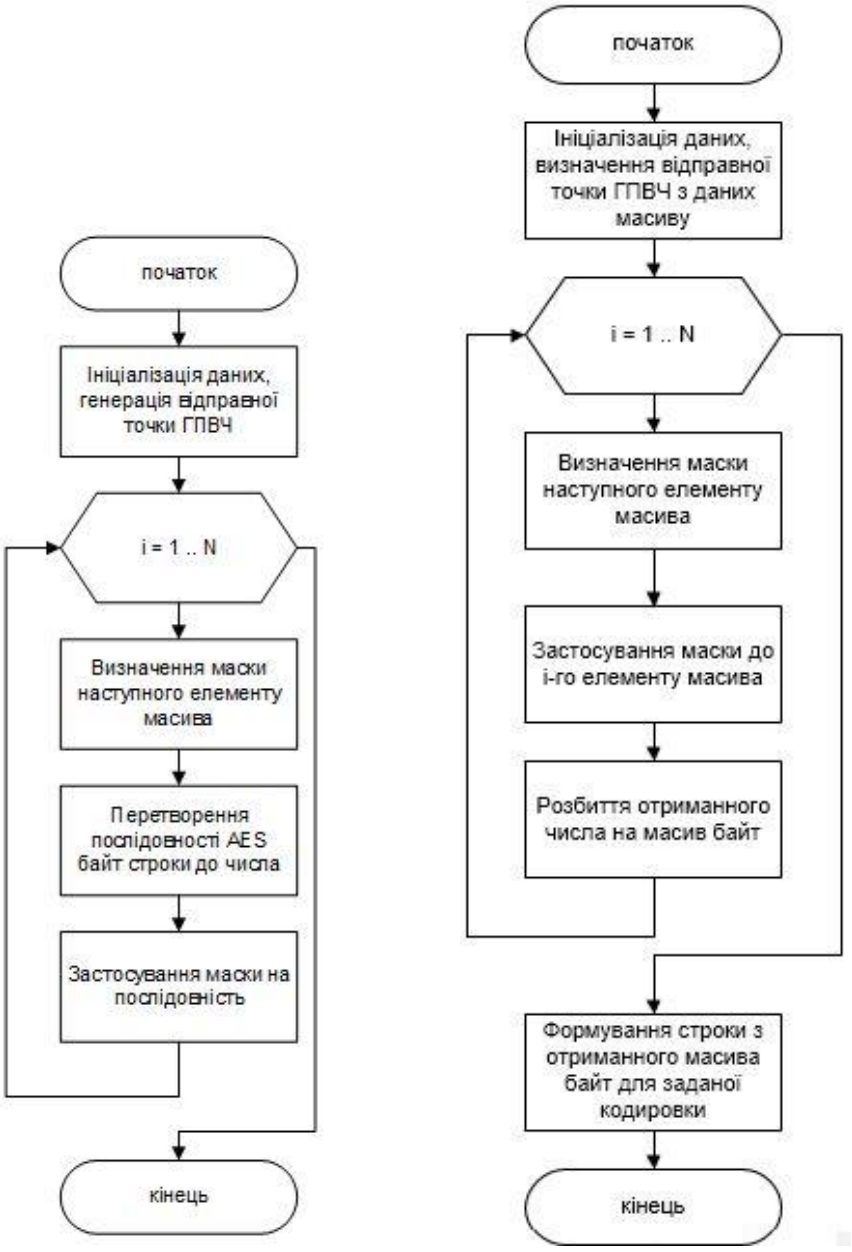
Запропонований метод полягає в особливостях роботи генератора псевдовипадкових чисел (ГПВЧ): при одному і тому ж початковому значенні зерна (*seed*) послідовність чисел завжди виходить однакова. В результаті роботи методу виходить масив чисел, який відображає масив, і завдяки симетричності алгоритму може бути назад перетворений в рядок.

В сучасних операційних системах найбільший за розміром і поширенням тип даних *long*, який займає в пам'яті 8 байт. В залежності від мовних особливостей тексту в наявному обсязі пам'яті можна зберігати до 8 символів (за умови, що всі вони будуть однобайтові, наприклад, символи латинського алфавіту, цифри). При використанні кодування *UTF-8* кількість символів, які можна записати у 8 байт, зменшується.

Для того, щоб перетворити рядок в число, необхідна кількість змінних типу *long* у 8 разів менша, ніж обсяг рядка в байтах.

У зв'язку з тим, що метод, який розроблюється, базується на початковому значенні для функції генерації псевдовипадкових чисел, то результуючий масив буде містити на одне значення більше очікуваного.

Алгоритм обфускації. Схематичний опис алгоритму обфускації строкових літералів на основі особливостей роботи ГПВЧ наведено на рис. 1а.



а б

Рис. 1. Алгоритм обфускації (а) та деобфускації (б) строкових літералів

Крок 1. Ініціалізація даних. На даному етапі виділяється пам'ять під результуючий масив згідно з формулою:

$$cnt = GVS + (Size / AES) + ((Size\% AES == 0)? 0: 1),$$

де *Size* - розмір початкового рядка в байтах; *GVS* (*Generated Value Size*) - кількість елементів масиву, які будуть займати значення відправної точки генератора псевдовипадкових чисел (в поданому прикладі - 1 елемент); *AES* (*Array Element Size*) - розмір одного елемента результуючого масиву в байтах (для типу *long* - 8 байт).

Також генерується псевдовипадкове число, яке буде є «відправною» точкою для роботи генератора псевдовипадкових чисел. Дане число генерується в залежності від поточного часу, а це означає, що дві вихідних послідовності одного і того ж рядка будуть повністю відрізнятися.

Перші *GVS* елементів масиву заповнюються псевдовипадковими числами – відправною точкою ГПВЧ.

Крок 2. Заповнення елементів, що залишилися від вихідного масиву.

Крок 2.1. Для кожного наступного елемента масиву відбувається обчислення наступного псевдовипадкового числа, яке буде виступати в ролі «бітової маски» для приховування даних.

Крок 2.2. Перетворення наступних *AES* байт рядка в число шляхом подання їх як складових байт числа. Наприклад, рядок *ABCD* перетворюється в число: 0x44434241

Крок 2.3. Накладення маски, отриманої на кроці 2.1 на число, отримане на кроці 2.2.

Алгоритм деобфускації. Для проведення процедури деобфускації з метою відновлення рядків використовується послідовність дій, описана на рис. 1б. Даний алгоритм можна охарактеризувати наступною послідовністю дій.

Крок 1. Ініціалізація даних. На даному етапі виділяється пам'ять під результуючий рядок (в байтах) згідно з формулою:

$$Size = (cnt - GVS) * GVS,$$

де *GVS* (*Generated Value Size*) - кількість елементів масиву, які будуть приймати значення відправної точки генератора псевдовипадкових чисел (в поданому прикладі - 1 елемент); *AES* (*Array Element Size*) - розмір одного елемента результуючого масиву в байтах (для типу *long* - 8 байт); *cnt* - кількість елементів у вихідному масиві.

ГПВЧ ініціалізується згідно до відправної точки, яка розраховується з даних перших *GVS* елементів масиву. Так, в поданому прикладі, 0-й елемент масиву і є відправною точкою.

Крок 2. Формування масиву байт результуючого рядка.

Крок 2.1. Для кожної наступної послідовності з *AES* байт формується "маска" згідно з алгоритмом роботи ГПВЧ.

Крок 2.2. Маска накладається на послідовність з необроблених *AES* байт. В результаті виходить масив, аналогічний отриманому в результаті роботи Кроку 2.2 процесу обфускації.

Крок 2.3. Розбиття числа розміром *AES* байт на *AES* окремих байт і додавання їх в результуючий масив байт.

Крок 3. Формування рядку з масиву байт відповідно до обраного кодування. У засобу перевірки правопису *UTF-8* немає чіткої відповідності «1 байт = 1 символ», тому розмір результуючого рядка може бути менше.

Розвитком даного алгоритму є модифікація вихідної послідовності не як масиву чисел (2,4,8 - байтних), а як байтового масиву або *BigInteger* числа. Використовуючи даний підхід, необхідність вирівнювання буде усунена, що дає можливість зробити значення відправної точки ГПВЧ практично необмеженої довжини (в тому числі і для використання власного ГПВЧ), а також використання маски, розмір якої не є кратним до обраного типу елемента масиву (*AES*) - так, при типі елемента результуючого масиву *long*, кожен елемент буде займати 8 байт, і для узгодження вирівнювання необхідно, щоб довжина маски становила 1,2,4,8 байт.

Наступним розвитком запропонованого алгоритму є використання перетворення числових значень на рядок і повторення алгоритму задану кількість разів.

4.2 Обфускація імен ідентифікаторів

Історично, обфускація імен ідентифікаторів використовувалася при виробництві *J2ME*-застосунків для зменшення обсягу результуючого програмного застосунку шляхом скорочення будь-яких ідентифікаторів до 1-2 буквених назв.

Так як сучасні компілятори підтримують імена ідентифікаторів в *UTF-8* форматі, розвитком обфускації ідентифікаторів було скорочення імен ідентифікаторів до одного *UTF-8* символу, що візуально «читалося» гірше. Наприклад символ з кодом \u0001 не має репрезентаційного символу, що погіршує сприймання коду, так як зловмисник не бачить

назву змінної. Однак, даний підхід до обфускації імен ідентифікаторів мав недолік - була можливість взяти декомпільований код, внести в нього зміни (зокрема, заміну всіх ідентифікаторів з кодом \u0001 на щось більш логічне) і без труднощів перекомпілювати. Розвитком даного підходу стало перейменування імен ідентифікаторів в ключові слова мови, такі як "do", "while", "for", які перешкоджають перекомпіляції.

Аналіз описаних реалізацій обфускації імен ідентифікаторів показав їх спільний недолік - в рамках області видимості вони мають одну й ту саму назву, що дає можливість простежити набір дій, які виконуються з одним і тим же ідентифікатором.

Мови програмування не дають можливості конструювання імені ідентифікатора - воно повинно бути відомо і повноцінно на етапі компіляції. В рамках підвищення рівня обфускації коду програми пропонується використання глобального асоціативного масиву, де ключ - рядок - ім'я ідентифікатора. Це дає нам можливість отримувати доступ на читання і запис елемента асоціативного масиву по ключу, який буде генеруватися «на льоту», що ускладнить зловмиснику можливість простежити хід програми без налагодження.

Наприклад, конструкція

```
int value = 10;
System.out.println(value);
```

при переході на асоціативний масив перетвориться в конструкцію:

```
private static final Map<String, Object> GLOBAL_VARIABLES = new HashMap()<>;
...
GLOBAL_VARIABLES.put("value", 10);
System.out.println(GLOBAL_VARIABLES.get("value");
```

Цей варіант використання може бути ще більше обфускований, якщо виконати обфускацію строкових літералів з використанням особливостей генератора псевдовипадкових чисел:

```
GLOBAL_VARIABLES.put(ConvertString(new long[] {1256, 6521}), 10);
System.out.println(GLOBAL_VARIABLES.get(ConvertString(new long[] {126, 321}));
```

Слід зазначити, що глобальний асоціативний масив має таку особливість: всі ключі даного масиву повинні бути унікальні, а при розробці програмного забезпечення:

– в різних методах і класах можуть бути присутніми ідентифікатори з однаковим ім'ям;

– клас може мати кілька екземплярів, а отже, необхідно ідентифікувати екземпляр класу;

– функції, в яких використовуються змінні, можуть бути перевантажені.

У зв'язку з вищезазначеними обмеженнями, було прийнято рішення використовувати наступний формат ідентифікатора:

classname @ hash:global_class_id

де *classname* - повне ім'я класу (разом з пакетом); *hash* - геш поточного екземпляру класу. У разі, якщо ідентифікатор глобальний, - значення порожнє; *global_class_id* - унікальний ідентифікатор в контексті поточного класу. Це дає можливість уникнути необхідності визначення області видимості і наявності перевантажених функцій.

Алгоритм заплутування імен ідентифікаторів таким чином можна представити схемою алгоритму, представленою на рис. 2.

Рис. 2. Схема алгоритму обфускації імен ідентифікаторів

5. Обговорення результатів дослідження

В рамках дослідження було проведено експеримент. Був розроблений модуль для створення та верифікації ліцензійного ключа на основі характеристик системи [13], що був обфускований розробниками. Мета експерименту – визначити, скільки часу необхідно IT-фахівцям для того, щоб розплутати алгоритм і автоматизувати процес отримання необхідних даних. Була зроблена вибірка зі 100 осіб.

Результат показав, що для розплутування алгоритму в середньому витрачається 5.3 годин з значенням середньоквадратичного відхилення рівного 1.1 години. На рис. 3 наведено порівняльну діаграму часу (вісь ОУ) аналізу обфускованого та необфускованого коду для кожної особи (вісь ОХ).

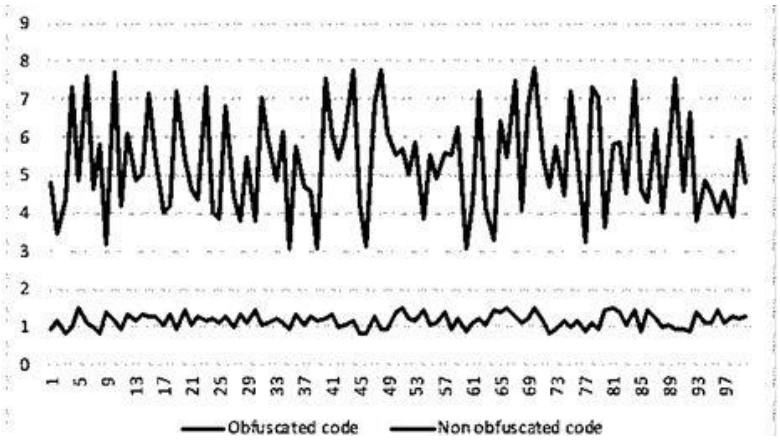


Рис. 3. Порівняльна діаграма часу аналізу обфускованого (зверху) та необфускованого (знизу) коду розробленого модуля

6. Висновки

Світ розробки програмного забезпечення спрямований на використання байткод-орієнтованих мов програмування. Аналіз літератури показав, що методам обфускації, які спрямовані на заплутування даних та ідентифікаторів не приділяється достатньо уваги. В даній роботі розроблені методи обфускації строкових літералів та імен ідентифікаторів, доцільність використання яких підтверджено експериментом, в якому наведено, що час, який витрачається на деобфускацію з використанням розроблених засобів обфускації, до 5 разів більше.

Список використаної літератури

1. Мельников В.П. Информационная безопасность и защита информации, 3-е изд, Для студентов высших учебных заведений / Мельников В.П., Клейменов С.А., Петраков А.М. – М.: Издательский центр «Академия», 2008. – 336с.

2. Основы информационной безопасности: курс лекций: учебное. 3-е изд. Под редакцией академика РАН В.Б. Бетелина. – М.: ИНТУИТ.РУ "Интернет-университет Информационных Технологий", 2006. – 208 с. – ISBN 5-9556-0052-3.

3. Programming Concepts: Compiled and Interpreted Languages. https://thecodeboss.dev/2015/07/programming-concepts-compiled-and-interpreted-languages/ (last access 01-dec-2019).

4. A Look At 5 of the Most Popular Programming Languages of 2019. https://stackify.com/popular-programming-languages-2018/ (last accessed 01-dec-2019).

5. Top Computer Languages. http://statisticstimes.com/tech/top-computer-languages.php (last accessed 01-dec-2019).

6. What's The Average Web App Development Cost. https://perfectial.com/blog/average-web-app-development-cost/ (last accessed 01-dec-2019).

7. Collberg C. A Taxonomy of Obfuscating Transformations / C. Collberg, C. Thomborson, D. Low. – Auckland: Department of Computer Science, The University of Auckland, 1997. – 36 p.

8. Barak B. On the (im) possibility of obfuscating programs / Barak B., Goldreich O., Impagliazzo R., Rudich S., Sahai A., Vadhan S. and Yang K. // CRYPTO-2001.

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

10. Goldwasser S. On best-possible obfuscation / Goldwasser S., and Guy N. R. // TCC-2007.

11. Schrittwieser S. Protecting Software through Obfuscation: Can It Keep Pace with Progress in Code Analysis? / Schrittwieser S., Katzenbeisser S., Kinder J., Merzdovnik G., and Weippl E. // *ACM Comput. Surv.* 49, 1, Article 4 (April 2016) - 40 p.

12. Faruki, P. Android Code Protection via Obfuscation Techniques: Past, Present and Future Directions / Faruki P., Fereidooni H., Laxmi V., Conti M., Gaur M. // November, 2016. – 37 p.

13. Давыдов В.В. Система формирования цифрового идентификатора программного обеспечения для защиты авторских прав / Давыдов В.В., Семенов С.Г., Мовчан А.В. // XV Міжнародний науковий семінар «Сучасні проблеми інформатики в управлінні, економіці, освіті та подоланні наслідків Чорнобильської катастрофи», Київ – оз. Світязь. - 4–8 липня 2016 року. – К.: Національна академія управління, 2016. – с.110-115.

References

1. Melnikov V.P., Kleimenov S.A., Petrakov A.M. (2008). *"Information Security and Information Security, 3rd ed. Training Allowance for stud. higher studies institutions."* M.,"Academy": 336. Print.

2. Galatenko V.A. (2016). "*Fundamentals of Information Security."* M., National Open University "INTUIT": 267. Print. ISBN 5-9556-0052-3.

3. *Programming Concepts: Compiled and Interpreted Languages.* https://thecodeboss.dev/2015/07/programming-concepts-compiled-and-interpreted-languages/ (01-dec-2019).

4. *A Look At 5 of the Most Popular Programming Languages of 2019.* https://stackify.com/popular-programming-languages-2018/ (01-dec-2019).

5. *Top Computer Languages.* http://statisticstimes.com/tech/top-computer-languages.php (01-dec-2019).

6. *What's The Average Web App Development Cost.* https://perfectial.com/blog/average-web-app-development-cost/ (01-dec-2019).

7. Collberg C., Thomborson C., Low D.. (1997). *"A Taxonomy of Obfuscating Transformations"*. Department of Computer Science, The University of Auckland: 36. Print.

8. Barak B., Goldreich O., Impagliazzo R., Rudich S., Sahai A., Vadhan S. and Yang K. (2001). «On the (im) possibility of obfuscating programs.» *CRYPTO-2001*.

9. Garg S., Gentry C., Halevi S., Raykova M., Sahai A., and Waters B. (2013). «Candidate indistinguishability obfuscation and functional encryption for all circuits.» *FOCS-2013*.

10. Goldwasser S., and Guy N. R. (2007). "On best-possible obfuscation." *TCC-2007*.

11. Sebastian Schrittwieser, Stefan Katzenbeisser, Johannes Kinder, Georg Merzdovnik, and Edgar Weippl (2016). "Protecting Software through Obfuscation: Can It Keep Pace with Progress in Code Analysis?" *ACM Comput. Surv.* 49, 1, Article 4 (April 2016): 40. Print.

12. Faruki, Parvez & Fereidooni, Hossein & Laxmi, Vijay & Conti, Mauro & and, & Gaur, Manoj (2016). "Android Code Protection via Obfuscation Techniques: Past, Present and Future Directions": 37. Print.

13. Davydov V.V., Semenov S.G., Movchan A.V. (2016). "System for software digital identifier generation for copyright protection". *XV International scientific seminar "Contemporary Problems of Informatics in Management, Economics, Education and Overcoming the Consequences of the Chornobyl Catastrophe", Kyiv-Svityaz, July 4-8, 2016, K., National Academy of Management*. 110-115: Print.

Автори статті (Authors of the article)

Семенов Сергій Геннадійович – д.т.н., професор, завідувач кафедри обчислювальної техніки та програмування (Semenov Serhii, Dr.Sci in Technics, Prof., Head of the Department of Computer Science and Programming). Phone: +380 67 579 30 82. E-mail: s_semenov@ukr.net.

Давидов Вячеслав Вадимович – к.т.н., доцент кафедри обчислювальної техніки та програмування (Davydov Viacheslav, PhD in Technics, Associate Professor of the Department of Computer Science and Programming). Phone: +380 63 475 10 41. E-mail: vyacheslav.v.davydov@gmail.com.

Волошин Денис Геннадійович – аспірант кафедри обчислювальної техніки та програмування (Voloshyn Denys, Postgraduate student of the Department of Computer Science and Programming). Phone: +380 67 934 25 76. E-mail: ultrageron@gmail.com.

Гребенюк Дарина Сергіївна – аспірантка кафедри обчислювальної техніки та програмування (Hrebeniuk Daryna, Postgraduate student of the Department of Computer Science and Programming). Phone: +380 67 952 84 80. E-mail: darina.gg1@gmail.com.