УДК 004.056.53

Я. І. Насталенко

ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ЗАХИСТУ ПРОГРАМНОГО КОДУ ШЛЯХОМ ЛЕКСИЧНОЇ ОБФУСКАЦІЇ

Вінницький національний технічний університет

Анотація. *В даній статті проведено аналіз різних видів захисту програмного коду від несанкціонованого дослідження. Запропоновано та розроблено засіб для захисту програмного коду шляхом лексичної обфускації, реалізований мовою програмування Java та досліджено його ефективність.*

Ключові слова: *захист програмного коду, лексична обфускація, Java*

Abstract. *This article analyzes the different types of protection of software code from unauthorized research. Was developed a tool for protecting program code by lexical obfuscation, implemented in Java programming language, and investigated its effectiveness.*

Keywords: *code protection, lexical obfuscation, Java.*

Вступ

Сучасний світ не можливо уявити без розробки та використання програмного забезпечення (ПЗ) з метою полегшення життя людству. З розвитком ПЗ все більшої актуальності набуває питання його захисту: від несанкціонованого доступу, дослідження, копіювання, піратства та промислового шпигунства. Існує безліч алгоритмів заплутування, мутації, компресії даних та шифрування, методи утруднення налагодження, дизасемблювання та дослідження, серед яких є метод обфускації – заплутування програмного коду.

Отже, метою роботи є розробка засобу для захисту програмного коду від статичного дослідження шляхом здійснення лексичної обфускації та аналіз його ефективності.

Результати дослідження

Обфускація – це один з методів захисту програмного коду, який дозволяє ускладнити процес реверсивної інженерії. Метод полягає в навмисному додаванні неоднозначної, заплутаної або оманливої інформації задля створення труднощів при дослідженні [1]. Основними умовами, яким повинен відповідати обфускований код, є збереження його функціональності та істотне ускладнення аналізу.

Обфускація може здійснюватись на двох рівнях: вищому та нижчому. На нижчому рівні здійснюється обфускація асемблерного коду, або безпосередньо двійкового коду програми. Цей процес є доволі клопітким, і вимагає врахування особливостей роботи більшості процесорів. Вищий рівень – це здійснення процесу обфускації над вихідним кодом програми, написаної мовою високого рівня [2]. Оскільки більшою популярністю користуються мови програмування високого рівня, то спрямувати всю увагу варто на них.

Обфускація буває декількох видів: лексична (символьна), обфускація даних та графа потоку керування. Лексична обфускація полягає в наступному: код програми повинен стати нечитабельним та менш інформативним. Саме тому для реалізації лексичної обфускації частіше всього використовують символьну обфускацію – заміну імен змінних, масивів, функцій, методів, класів на набір випадкових символів, імена, що міняють сенс змінних, об’єктів, класів, або їх порядкову нумерацію. Крім того, в комплексі з символьною обфускацією використовують зміну розміщення блоків програми (методів та функцій, де їх розташування в коді не відіграє ніякого значення) та додавання сміттєвих операцій [3].

За декілька останніх десятиліть було створено досить велику кількість програмних засобів для здійснення обфускації – обфускаторів (наприклад, NET Reactor, Babel, BitHelmet, SilSecure, Desa Ware та інші). Основними їх недоліками є занадто висока ціна, що не завжди виправдовує себе, або ж їх занадто низька якість, яка або псує програмний код, унеможливлюючи його подальше виконання, або ж взагалі майже нічого не змінює внаслідок обфускації.

Розглянемо перелік дій, що в сукупності являють собою лексичну обфускацію:

- додавання сміттєвих операцій;

– видалення необов’язкових конструкцій мови програмування. Це – коментарі (однорядкові та багаторядкові), зайві пробіли та всі відступи;
– заміна імен ідентифікаторів на випадкові набори символів/порядкові номери.

Для реалізації лексичної обфускації розроблено власний програмний засіб. Розглянемо алгоритм реалізації обфускації, використаний у цьому засобіс (рис. 1).

Рисунок 1 – Схема реалізації лексичної обфускації

Першим кроком розробленого алгоритму є видалення коментарів, оскільки вони є необов’язковими конструкціями, ш їх наявність, ймовірніше всього, допоможе зловмиснику в аналізі коду, а видалення не вплине на виконуваність програмного коду. Крім того, за видалення коментарів, обфускація буде проводитись ефективніше.

Наступним є додавання сміттєвих операцій, а саме фрагментів недосяжного коду, наявність якого не впливає на виконання програмного коду, але впливає на емоційний стан зловмисника. В якості недосяжного коду спеціально розроблені функції програмного засобу випадковим чином додають в текст методи, що здійснюють різні математичні операції та вирішують деякі задачі (транспонування матриць, їх множення, операції з рядами Тейлора та ін.).

Третім кроком є перейменування змінних, методів та класів. Це відбувається за рахунок заміни найменувань випадковим набором символів. Одним із варіантів реалізації даного кроку є заміна найменувань на порядкові номери, іншим же є заміна їх на безглузді найменування – обидва варіанти є гіршими за перший, оскільки велика кількість наборів випадкових символів мають сильніший вплив на дезорієнтацію зловмисника.

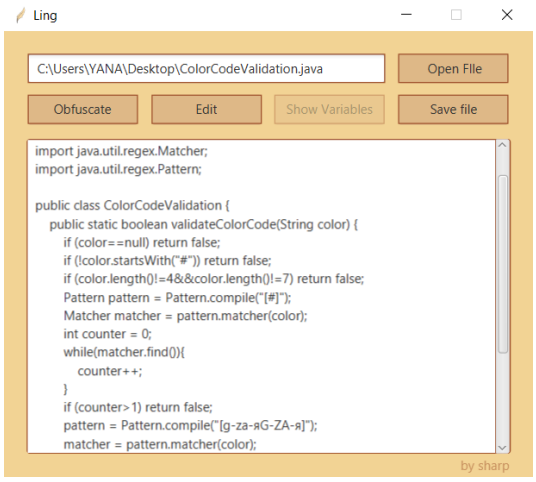

Рисунок 2 – Вигляд вікна реалізації обфускації

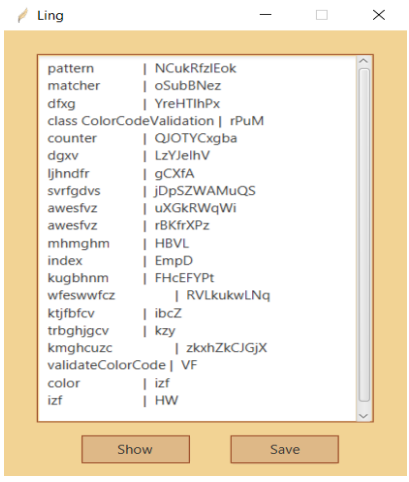

Рисунок 3 – Вікно виведення найменувань змінних

Видалення зайвих пробілів доцільніше реалізовувати останнім кроком, вже після здійснення всіх попередніх змін, оскільки в такому разі програмна реалізація буде значно легшою.

В результаті виконання вищевказаних кроків, програмний код наюбуває вигляду, який неможливо зрозуміти, або на його аналіз знадобиться занадто багато часу (рис. 4).

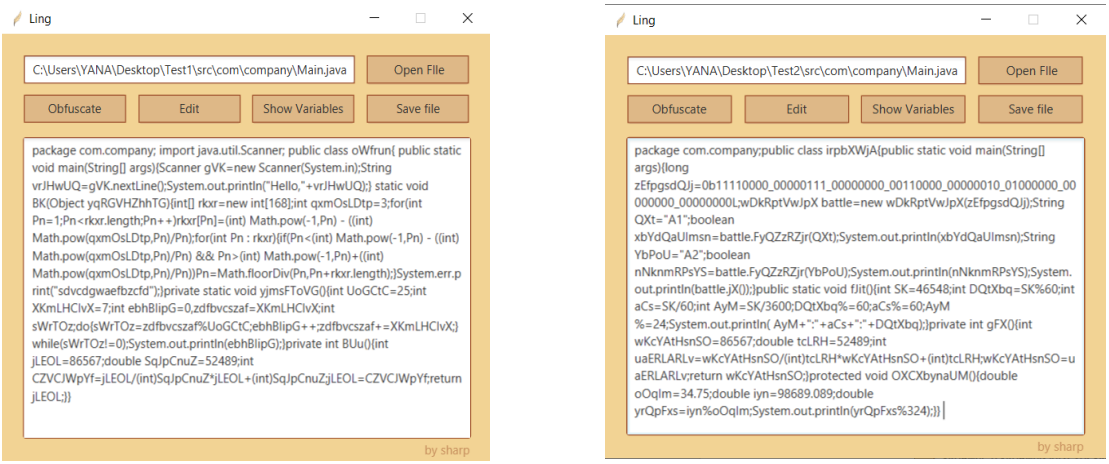

Рисунок 4 – Результат обфускації програмного коду двох різних програм

Оскільки однією із найпопулярніших мов програмування є Java, і оскільки вона має в наявності такі класи, як String та StringBuilder, зручні для реалізації роботи з тектовими даними, то реалізацію програмного засобу здійснено саме мовою Java. Програмним середовищем обрано середовище IntelliJ IDEA та середовище SceneBuilder для реалізації інтерфейсної частини.

Для аналізу стійкості даного захисту було використано декомпілятор Procyon.

```
package com.company;

import java.math.BigInteger;

public class wDkRptVwJpX
{
    private final long OmHmaDaiZQq;
    private long vMufVVKJ;

    public wDkRptVwJpX(final long OmHmaDaiZQq) {
        this.vMufVVKJ = 0L;
        this.OmHmaDaiZQq = OmHmaDaiZQq;
    }

    public boolean FyQZzRZjr(final String cBkMPsxHa) {
        final int jTzdpRMQ = this.sqyA(cBkMPsxHa);
        final int EBmynF = 64 - jTzdpRMQ;
        String BNSXrggMXIa = "1";
        for (int i = 1; i < EBmynF; ++i) {
            BNSXrggMXIa = invokedynamic(makeConcatWithConstants:(Ljava/lang/String;)Ljava/lang/String;, BNSXrggMXIa);
        }
        final Long cBkMPsxHaL = ELrRxrkHA(BNSXrggMXIa, 2);
        this.vMufVVKJ |= cBkMPsxHaL;
        String jX = Long.toBinaryString(this.OmHmaDaiZQq);
        if (jX.length() < 64) {
            for (int j = 0; j < 64 - jX.length(); jX = invokedynamic(makeConcatWithConstants:(Ljava/lang/String;)Ljava/lang/String;, jX), ++j) {}
        }
        return Character.toString(jX.charAt(jTzdpRMQ)).equals("1");
    }
```

Рисунок 5 – Фрагмент коду, отриманого декомпілятором

Декомпілятор повернув обфускованому коду читабельний вигляд, проте ідентифікатори залишились незмінними, тобто безглуздими наборами символів (рис. 3.12).

Крім того, як компілятор при створенні jar-файлу, так і декомпілятор, не видалили недосяжний код, що є значною перевагою при захисті (рис. 3.13)

Висновки

В результаті виконання роботи розроблено програмний засіб, що реалізує захист програмного коду від статичного дослідження програмного коду шляхом лексичної обфускації. Застосунок було розроблено мовою Java та використано програмні середовища IntelliJ IDEA та Scene Builder.

Засіб протестовано на правильність здійснення обфускації та досліджено якість захисту програмного коду шляхом лексичної обфускації. За допомогою декомпіляції можна подолати лише деякі методи даного способу захисту: програмний код знову набуває читабельного вигляду, проте найменування змінних, методів та класів залишаються не зміненими, що все одно достатньо ускладнює аналіз коду зловмисником.

Отже, з метою захисту програмного коду не варто використовувати лише лексичну обфускацію. Потрібно використовувати комплексний захист – звертатись до шифрування, пакування або інших методів обфускації, таких, як обфускація даних та графа потоку керування.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Finn Brunton, Helen Nissenbaum Obfuscation: a user’s guide for privacy and protest. Gildan Media, LLC, 2015. 123 p.
2. Каплун В. А., Дмитришин О. В., Баришев Ю. В. Захист програмного забезпечення. Частина 2 : навчальний посібник. Вінниця : ВНТУ, 2014. 105 с.
3. Коробейников А.Г., Кутузов И.М. Обфускация сокрытия вызовов при помощи инструкции invokedynamic // Кибернетика и программирование. – 2016. – № 5. – С. 33 - 37. DOI: 10.7256/2306-4196.2016.5.18686 URL: https://nbpublish.com/library_read_article.php?id=18686.

Насталенко Яна Іванівна – студентка групи 1БС-19Б, факультет інформаційних технологій та комп'ютерної інженерії, Вінницький національний технічний університет, Вінниця, e-mail: sof8013@gmail.com.

Nastalenko Yana I. – Department of Information Techologies and computer engineering, Vinnytsia National Technical University, Vinnytsia, email : sof8013@gmail.com.