# Дослідження підходу захисту програмних продуктів на основі маскування вихідного коду

Ганна Олійник[1], Сергій Грибков[2]

1. Кафедра інформаційних систем, Національний університет харчових технологій, УКРАЇНА, м.Київ, вул.Володимирська, 68, E-mail: AnnY_o@bigmir.net

2. Кафедра інформаційних систем, Національний університет харчових технологій, УКРАЇНА, м.Київ, вул.Володимирська, 68, E-mail: sergio_nuft@ukr.net

***The work under consideration examines applied aspect of the programme code masking task and its practical realization technique. The usage of the given method ensures programme product source code protection from reverse engineering without any functionality changes. The method under consideration obtains the following characteristic features: functions thread of execution graph considerable amplification by means of arcs addition that breaks graph structure but is obligatory passed in the course of execution; amplification of data flows and their interdependence. Introduction of data dynamic structures and dead arcs absence in graph provide method resistance to static methods of analysis. The drawback of the unfolded method is presented by source code increase.***

Ключові слова – захист програмного забезпечення, маскуючі перетворення, додатковий код, потік управління.

## I. Вступ

Сучасний розвиток інформаційних технологій вимагає використання систем захисту програмного забезпечення, що обумовлено рядом чинників, серед яких слід виділити: незаконне використання алгоритмів, що є інтелектуальною власністю автора, несанкціонований доступ, використання і модифікація, нелегальне розповсюдження та збут програмного забезпечення.

Важливим аспектом захисту є протидія зворотній інженерії та внесенню деяких змін з метою порушення існуючої функціональності програмного продукту. Важливим та найпершим кроком при створенні системи захисту програмного продукту є ускладнення та мінімізація можливості відновлення вихідного програмного коду при її дослідженні, а також недопустимість простого доступу до контрольних параметрів, встановлення своїх значень змінним, блокування роботи певних програмних блоків. Адже призначені для рішення таких задач інтерактивні програмні засоби дозволяють не тільки отримати повний доступ, а й точно відновити алгоритми роботи та внести будь-які зміни.

Дослідження програмного продукту може відбуватися у статичному або динамічному режимі. Суть першого зводиться до дослідження програмного коду, для отримання якого вихідний програмний модуль дизасемблюють. Динамічний режим передбачає трасування програми, тобто виконання з використанням спеціальних засобів, які включають можливість покрокового режиму, отримання доступів до регістрів, областей пам'яті, виконання зупинки програми за певною адресою тощо.

Одним з методів захисту від зворотної інженерії є маскування або заплутування вихідного коду програмного продукту, тобто таке його перетворення, за якого залишається незмінною функціональність, проте розуміння, зворотна інженерія, модифікація стають задачами неприйнятно високої вартості.

## II. Маскування вихідного коду

Підхід маскування програмних продуктів включає три основні аспекти: теоретичний, прикладний, психологічний. Теоретичний аспект базується на розробці нових алгоритмів перетворення графу потоку управління або певним чином представленій трансформації вихідних даних, а також включає теоретичну оцінку складності їх аналізу. Прикладний аспект являє собою розробку конкретних методів маскування, тобто виявлення найефективніших комбінацій існуючих алгоритмів, їх удосконалення, розробку нових, а також емпіричний порівняльний аналіз, аналіз стійкості та надійності методів тощо. Третій аспект – психологічний, не є чітко формалізованим, але не може бути проігнорованим, адже зворотна інженерія – процес, результат якого залежить від особистих якостей та властивостей особи, яка досліджує код програмного продукту [1].

Маскуючі перетворення на прикладному рівні поділяються на декілька груп в залежності від націленості на конкретні складові програмного коду. Перетворення форматування вносять зміни тільки у зовнішній вигляд коду, прикладом яких є зміна імен ідентифікаторів. Перетворення структур даних включають зміни ієрархії наслідування класів або об'єднання скалярних змінних одного типу даних в єдиний масив. Перетворення потоку керування програмного продукту направлені на зміни графів потоку управління окремо взятих функцій. Такі перетворення направлені проти окремих способів декомпіляції або використовують помилки інструментальних засобів декомпіляції.

Досліджена методика маскуючих перетворення визначає головним чином перетворення графу потоку управління. Як наслідок замаскований програмний код значно відрізняється від вихідного коду. Структури даних вихідного програмного продукту залишаються незмінними, проте з'являється значна кількість несуттєвих залежностей між даними.

Загальна ідея методу фактично складається з двох кроків. Першим кроком є значне збільшення складності графа потоку управління, але з умовою, що усі його дуги будуть пройдені при виконанні програмного коду. Другий крок – ускладнення потоків даних програмного продукту за допомогою «накладання» на програмний код додаткового коду, що не впливає на функції, які маскуються, і як наслідок, не впливає на коректність алгоритму виконання операцій програмного коду. Така функція

будується з фрагментів початкової функції, семантичні якості яких відомі заздалегідь, а також фрагментів бібліотеки маскуючого транслятора. Для ускладнення задачі виявлення замаскованої функції використовуються конструкції, що недостатньо піддаються аналізу, а саме вказівники та математичні тотожності [2].

До основних складових даного методу належать наступні: використання відкритих вставок функцій; винесення груп операторів; внесення додаткового коду декількох видів; поєднання функцій; клонування функцій.

Використання відкритої вставки функцій полягає в тому, що тіло функції підставляється у точку виклику функції. Таке перетворення є стандартним для оптимізуючих компіляторів і являється одностороннім, тобто з перетвореного програмного коду відновити такі функції неможливо.

Винесення групи операторів – метод обернений до попереднього. Деяка група операторів виділяється в окрему функцію, а при виникненні необхідності створюються формальні параметри.

Внесення додаткового коду таких видів: недосяжного коду, який ніколи не буде виконаний, а лише впливає на розмір вихідних програмних модулів, але не на швидкодію; несуттєвого коду, що виконується, проте не впливає на основний алгоритм роботи та не має жодних побічних ефектів; надлишкового коду, результати виконання якого використовуються в процесі подальшого виконання програми.

Поєднання функцій: ідея полягає в об'єднанні двох або більше функцій в одну. Списки параметрів початкових функцій поєднуються та додається ще один, значення якого визначає функцію, що буде виконуватись в конкретний момент.

Клонування функцій є засобом протидії дослідження місць її виклику, а також визначення параметрів виклику, оскільки створення декількох копій функції та застосування до них різних перетворень призведе до представлення виклику однієї фукції як до виклику різних.

Для реалізації розглянутого методу програмний код необхідно розглядати з точки зору системного програмування, а об'єктами маскування є вихідні коди реальних програмних продуктів. Замаскований програмний код збільшується в розмірі, що в результаті ускладнює застосування ручного аналізу при його демаскуванні за рахунок часових і вартісних обмежень.

## III.Стійкість методу до відомих методів аналізу

При застосуванні методів аналізу програмного коду для виявлення залежностей між даними вимагається міжпроцедурний аналіз програмного коду. При наявності великої кількості глобальних змінних та великого числа функцій, аналіз виявиться або неточним, або буде вимагати занадто багато ресурсів. Також це стосується аналізу вказівників розташованих у динамічній пам'яті.

Напівстатичний аналіз замаскованого програмного коду не дозволяє виявити явних закономірностей, відсутність яких робить такий аналіз значно менш ефективним.

Інструкції, що забезпечують стійкість замаскованої функції, розподілені серед усіх базових блоків, а не сконцентровані на невеликій ділянці. Тому демаскування вимагає аналізу всієї функції, а не певної її частини.

Збільшений розмір замаскованих функцій навіть для відносно невеликих функцій вихідного програмного коду є вагомою перешкодою для ручного аналізу.

Кожне маскуюче перетворення параметризується в широких межах з використанням випадкового підходу. Знання, отримане в результаті аналізу однієї замаскованої функції, може бути тільки частково застосовано для аналізу іншої замаскованої функції.

Після застосування методу граф потоку управління набуває такої структури, що його візуалізація може дати негативний результат, значно ускладнить розуміння, або такий граф може взагалі не відобразитися.

В наш час не існує математичного апарату, придатного для оцінки складності напівстатичного та динамічного аналізу програмного коду, що має істотну евристичну компоненту.

## Висновок

Використання розглянутого методу забезпечує захист вихідного коду програмного продукту від зворотної інженерії без зміни його функціональності. Особливістю дослідженого методу є суттєве ускладнення графу потоку управління функцій за рахунок додавання дуг, що порушують його структурність, але обов'язково будуть пройдені в процесі виконання, а також ускладнення потоків даних та залежностей між ними. Внесення динамічних структур даних і відсутність «мертвих» дуг в графі забезпечують стійкість методу до статичних методів аналізу. З недоліків дослідженого методу необхідно виділити збільшення вихідного коду. Практичні випробовування розглянутого методу продемонстрували забезпечення високого рівня захисту від зворотної інженерії вихідного коду програмного продукту.

## Література

[1] Золотарев В.В. Метод исследования программных средств защиты информации на основе компонентной модели информационной среды / В. В. Золотарев // Изв. ЮФУ: Технич. науки, 2008. – № 8. – С. 87–94.

[2] Чернов А.В. Об одном методе маскировки программ // В сб. «Труды Института системного программирования», под. ред. В. П. Иванникова. М.: ИСП РАН, 2003. – Т. 3. – С. 85–119.