**УДК 681.3.06**

**А.О. Стороженко, А.Я. Горпенюк[1], Н.М. Лужецька[2]**
Національний університет „Львівська політехніка”,
[1]кафедра захисту інформації,
[2]кафедра безпеки інформаційних технологій

# МЕТОДИКА ЗАХИСТУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ШЛЯХОМ ВПРОВАДЖЕННЯ ЦВЗ В АСЕМБЛЕРНИЙ КОД ПРОГРАМИ



**Various techniques for software protection are discussed in this paper, in particular – software watermarking for copyright protection. Also the algorithms for dynamically implementation and extraction software watermark are presented.**

**Keywords - digital watermarking, software protection, reverse engineering.**

**Розглянуто методи захисту програмного забезпечення. Зокрема, цифрові водяні знаки для захисту авторських прав на програмне забезпечення. Також запропоновано алгоритми для впровадження та вилучення водяних знаків.**

**Ключові слова – цифрові водяні знаки, захист програмного забезпечення, реверс-інжиніринг.**

## Вступ

На сьогодні комп’ютерне піратство є однією з головних проблем у світовій програмній індустрії. За деякими підрахунками загальна сума збитку, яку воно приносить розробникам програмного забезпечення (ПЗ), сягає десятки мільярдів доларів на рік [1]. У зв’язку з цим, зусиллями багатьох вчених та працівників галузі створено багато етичних, правових і технічних рішень для захисту програм. До етичних та правових методів відносяться закони про авторське право, які сприяють усвідомленню користувачами неправомірності використання неліцензованого програмного забезпечення. Серед технічних виділяють апаратні, апаратно-програмні та програмні засоби. В апаратних засобах використовується спеціальне обладнання (наприклад, електронні ключі), або фізичні особливості носіїв інформації (CD, DVD і т.д.), для ідентифікації оригінальних версій програм і захисту продуктів від нелегального використання. Програмні засоби захисту реалізуються програмним шляхом, без використання фізичних характеристик носіїв інформації чи спеціального обладнання.

**Мета роботи** – проаналізувати існуючі методи захисту комп’ютерних програм від пірацтва; запропонувати підхід для забезпечення захисту авторських прав у сфері програмного забезпечення.

## Огляд та аналіз програмних методів захисту ПЗ

Програмні методи захисту є перспективними, оскільки для них є характерними гнучкість у проектуванні, адаптованість до алгоритмів програм-об’єктів захисту та, в порівнянні з апаратними засобами, нижча вартість реалізації. До програмних методів захисту відносяться шифрування та заплутування або обфускація (obfuscation) програмного коду. Такі методи ускладнюють структуру програми і, тим самим, процес зворотнього або реверс-інжинірингу (reverse engineering). Не менш ефективними у боротьбі з піратством є стеганографічні методи, а саме цифрові водяні знаки, які впроваджують в коди програм для підтвердження авторських прав. Наприклад, якщо в кожен примірник програми додати водяний знак у вигляді ідентифікаційного номера, то у випадку незаконного копіювання і розповсюдження програми можна визначити недобросовісного користувача. Прикладом такого підходу є «Автоматизована система ідентифікації комп’ютерних програм» [2]. У даній роботі для впровадження ідентифікаційних номерів використовуються особливості форматів виконуваних файлів, а саме наявність зарезервованих полів у заголовках цих

файлів. Схожий підхід пропонується авторами в роботі [3], які записують секретне повідомлення в кінець секції програмного файлу замість нульових байт вирівнювання. Перевагою даного методу є те, що розмір перетвореного файлу залишається незмінним, а саме перетворення не впливає на виконання програми. Проте, як показано у статті [4], помітними є розбіжності статистичних властивостей коду програми і зашифрованого повідомлення. Отже, даний метод є нестійким до атак статистичного стегоаналізу.

Існують методи, які приховують інформацію безпосередньо у вихідних кодах програм. Наприклад, розглянемо функцію:

```
int f (int x)
{
        int a, b;
        a = x + 1;
        b = x - 1;
        return a + 2 * b;
}
```

Ми можемо впровадити два біти водяного знака шляхом зміни вибору порядку оголошення змінних (1), і зміни порядку проходження незалежних інструкцій у цій функції (2) :

0→ int a, b; 1→ int b, a; (1)

0→ a = x + 1; 1→ b = x – 1; (2)

b = x – 1; a = x + 1;

Однак, такий метод може виявитися недієвим, якщо зловмисник заволодіє вихідним кодом програми. Наприклад, провівши декомпіляцію виконуваного файлу, зловмисник може отримати найбільш схожий до початкового код на мові високого рівня. Це дозволить спростити процес реверс-інжинірингу. Отже, можна зробити висновок, що прихована інформація буде більш захищеною, якщо перетворення в програмних кодах здійснювати на нижчому рівні, використовуючи об’єктний чи асемблерний код. Це дозволить ускладнити можливий процес декомпіляції програмного коду. Наприклад, в роботах [5], [6] використано методи для впровадження водяних знаків, що діють на рівні асемблера.

**Опис запропонованого методу**

Як правило, програмне забезпечення поширюють у вигляді скомпільованого файлу відповідного формату (EXE- виконувані файли в операційній системі DOS і PE – у Windows), без вихідних кодів. Програма подається в бінарному коді для обробки процесором. На сьогодні відомо методи реверс-інжинірингу, що застосовуються до двійкових програм з метою розуміння їх структури і алгоритмів. Це досягається шляхом перетворення машинного коду в асемблерний. Далі відбувається процес декомпіляції цього коду у програму на мові високого рівня. Реверс-інжиніринг часто використовується в законних цілях. Зокрема, під час аналізу шкідливих програм – вірусів, з метою розуміння їх поведінки, визначення вразливих місць. Але здебільшого така технологія застосовується для злому ПЗ, внесення несанкціонованих змін у програму, викрадення інтелектуальної власності.

Одним із основних методів запобігання зворотному інжинірингу є обфускація або заплутування коду [7]. В нашому підході методика захисту програм полягає у комплексному застосуванні методів заплутування кодів та цифрових водяних знаків. Така методика може служити для захисту авторських прав та додатково ускладнювати реверс-інжиніринг.

За способом впровадження водяні знаки поділяють на два основні типи: статичні і динамічні. Статичні водяні знаки можуть бути вилучені безпосередньо з програми, без її виконання. Найпростіша форма статичних водяних знаків - рядок в стрічках з коментарем чи фактична змінна в рядках програми. Звичайно, такий водяний знак можна легко ідентифікувати і знищити.

Динамічні водяні знаки є більш стійкими в цьому аспекті. Процес їх впровадження та вилучення, як правило, вимагає виконання програми. Наприклад, для створення водяного знаку

може використовуватись конкретний набір команд, які виконуються в окремій частині програми [8].

У запропонованому методі з метою впровадження водяного знака ми змінюємо асемберний код програми. Припустимо, що всі інструкції (або команди) у програмі якимось чином впорядковані, тобто при виконанні програми кожна з них має свій порядковий номер або адресу. Перетворення методу полягають у тому, що можна змінювати адреси інструкцій, вставляючи між ними додаткові команди.

У мові асемблера є багато команд, які можна використовувати для перетворення коду. Проте необхідно, щоб програма працювала коректно, незважаючи на внесені зміни. Тому, як водяні знаки використаємо команди, які будуть виконуватись всередині конструкцій програми, але результат їх виконання жодним чином не вплине на алгоритм її роботи. При цьому будуть змінюватися лише адреси оригінальних інструкцій програми.

Приклади таких інструкцій наведено в Таблиці 1.

*Таблиця 1.*

| Команда | Опис та приклад використання |
|---|---|
| NOP | Невиконання операції - це «фіктивна» команда, яка не впливає на функціональність програми. Процесор пропускає цю інструкцію. На практиці команда NOP використовується, щоб внести затримку під час виконання інструкцій програми [9]. |
| JUMP | За аргумент команда приймає адресу пам'яті. В асемблері цей аргумент вказано у вигляді мітки. JUMP змушує перейти на вказану адресу. Можна використовувати JUMP для переходу інструкції до іншої адреси, залишаючи при цьому програмний потік незмінним. |
| XOR,<br>OR,<br>ADD /SUB,<br>INC /DEC,<br>NEG,<br>NOT | Логічні і арифметичні інструкції.<br>• Логічні команди, які залишають значення регістра незмінним:<br>XOR eax, 0;<br>OR eax, 0;<br>• сполучення протилежних команд<br>ADD ebx,8<br>SUB ebx, 8;<br>INC ebx<br>DEC ebx;<br>• дублювання однакових команд<br>NEG ecx<br>NEG ecx;<br>NOT ecx<br>NOT ecx. |
| PUSH / POP | В мові асемблера виклик цих двох інструкцій здійснюється досить часто. PUSH поміщає значення регістра в стек і POP повертає ці значення зі стека. Отже, можна вставляти пари PUSH / POP інструкцій для зміни цільової адреси інструкції:<br>PUSH eax<br>MOV eax, 2CH<br>POP eax. |

Візьмемо десять таких команд і присвоїмо кожній з них номер 0-9. Нехай водяним знаком, який вноситься у програму є число «25». Припустимо, що цифрі «2» відповідає команда XOR eax, 0, а «5» - ADD ebx, 8 SUB ebx, 8.

Щоб детальніше проілюструвати наш підхід розглянемо його на прикладі. Нехай контейнером є програма на мові С “hello.c”:

```
 # include <stdio.h>
int main()
{
```

```
 printf("Hello, world!\n");
 return 0;
}
```

**Алгоритм впровадження водяних знаків:**

1. Першим кроком є компіляція програми.
2. До виконуваного файлу застосовуємо дизасемблер, наприклад «IDA Pro», щоб отримати асемблерний код програми.
3. Водяний знак «25», який будемо вставляти в ассемблерний файл, представлений у вигляді інструкцій:
   XOR eax, 0 → «2»
   ADD ebx, 8 → «5»
   SUB ebx, 8
4. Крім водяного знака, в асемблерний файл помістимо додаткові перетворення у вигляді інструкцій, які створюватимуть ефект «заплутування» коду з метою приховування місця розташування водяного знака та ускладнення процесу дослідження програмного коду.
5. Розташування цих перетворень, тобто значення номерів адрес, за якими будуть вставлятися водяні знаки і додаткові інструкції, будуть вибиратися випадковим чином.
6. Далі компілюємо вже модифікований асемблерний файл. В результаті утворюється новий виконуваний файл програми з водяним знаком.

Таким чином, можна створити N копій програмного забезпечення з різними водяними знаками ( ідентифікаційними номерами), повторюючи ці кроки для кожного екземпляра. Заново створені виконувані файли будуть функціонально еквівалентними, оскільки в результаті перетворень не змінюється граф проходження програми.

У Таблиці 2 показано як змінився код програми на мові асемблера після впровадження ЦВЗ.

*Таблиця 2.*

| Асембрерний код програми hello.c | Код програми hello.c з водяним знаком |
|---|---|
| _main: | _main: |
| pushl %ebp | pushl %ebp |
| movl %esp, %ebp | movl %esp, %ebp |
| subl $8, %esp | subl $8, %esp |
| andl $-16, %esp | andl $-16, %esp |
| movl $0, %eax | movl $0, %eax |
| | **xor $0, %eax** |
| | **add $8, %ebp** |
| Водяний знак “25” | **sub $8, %ebp** |
| movl %eax, -4(%ebp) | movl %eax, -4(%ebp) |
| movl -4(%ebp), %eax | movl -4(%ebp), %eax |
| | **xor $0, %ebp** |
| call __alloca | call __alloca |
| call ___main | call ___main |
| movl $LC0, (%esp) | movl $LC0, (%esp) |
| Обфускація | **push %ebp** |
| | **movl $23, %ebp** |
| | **pop %ebp** |
| call _printf | call _printf |
| movl $0, %eax | movl $0, %eax |
| leave | leave |
| ret | ret |
| .def _printf; .scl 2; | .def _printf; .scl 2; |
| .type 32; .endef | .type 32; .endef |

### Вилучення водяних знаків

Доведення авторських прав та оригінальності програми здійснюється шляхом зчитування водяного знака. В даному випадку його потрібно вилучити з перетвореного виконуваного файлу.

**Алгоритм вилучення водяних знаків:**

1. Спочатку розкладаємо виконуваний файл на асемблерний код за допомогою дизасемблера IDA Pro.
2. Вибираємо з даного коду команди, які були додані у програму як водяний знак. При цьому пропускаємо інструкції, що заплутують код.
3. Далі, на основі порівняння команд за відповідним шаблоном (присвоєними їм номерами 0-9), зчитуємо наш водяний знак.

### Висновок

У роботі проведено аналіз існуючих методів захисту програм від пірацтва та представлено методику для впровадження ЦВЗ в асемблерний код програми. Також запропоновано алгоритми для впровадження та вилучення ЦВЗ.

Маючи вихідний код програми, написаний на мові програмування високого рівня, ми розібрали його на асемблерний код. До асемблерного файлу, крім водяного знака, помістили додаткові перетворення у вигляді інструкцій, що створюють ефект обфускації, додатково заплутуючи код. Таким чином отримали новий виконуваний файл з водяним знаком.

Запропонований метод має ряд потенційних переваг. По-перше, автоматизувати процес видалення водяних знаків з асемблерного файлу надзвичайно складно. По-друге, водяні знаки можуть забезпечити надійніший захист, будучи вставленими у програму декілька разів.

### Література

*[1] Ninth Annual BSA Global Software 2011 Piracy Study [веб-портал] / Business Software Alliance. [Washington], 2000–2012. URL: http://portal.bsa.org/globalpiracy2011/ (дата звернення: 30.01.2013). [2] Буза М.К. Автоматизированная система идентификации компьютерных программ / М.К. Буза, Е.Н. Ливак // Автоматизация и современные технологии. – 2002. – № 8. – С. 3-10. [3] Thorpe D. Development System with Methodology Providing Information Hiding in Executable Program // US Patent 20060136875 (2006). [4] Нечта И.В. Эффективный метод стегоанализа исполняемых файлов, базирующийся на коде Хаффмана / И.В. Нечта // Вестник СибГУТИ. – 2010. – № 4. – С. 47-53. [5] Steganography for executables and code transformation signatures / B. Anckaert, B. De Sutter, D. Chanet, K. De Bosschere // Information Security and Cryptology ICISC-2004. – 2005. – № 7. – P. 425-439. [6] Ярмолик В.Н. Современные методы и средства защиты авторських прав разработчиков программного обеспечения / В.Н. Ярмолик, С.С. Потрянко // Доклады БГУИР. – 2004. – № 1. – С. 126-135. [7] Binary Obfuscation Using Signals [Електронний ресурс]: Proceedings of the 16th USENIX Security Symposium / Igor V. Popov, Saumya K. Debray, Gregory R. Andrews //USENIX Security Symposium – 2007. – С. 275–290. – Режим доступу до ресурсу http://static.usenix.org/event/sec07/tech/full_papers/popov/popov_html [8] Collberg C.S. Watermarking, tamper-proofing, and obfuscation-tools for software protection / Christian S. Collberg, Clark Thomborson // Software Engineering, IEEE Transactions. – 2002. – № 8. – P.735-746.[9] Randall Hyde. The art of assembly language / Hyde R. – No Starch Press, Inc., 2003. – 899 с. – ISBN 1-886411-97-2*