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

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.

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.

Key words: digital watermarking, software protection, reverse engineering.

Вступ

Сьогодні комп'ютерне піратство є однією з головних проблем у світовій програмній індустрії. За деякими підрахунками, загальна сума збитків, яких воно завдає розробникам програмного забезпечення (ПЗ), сягає десятків 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 належать 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 тощо), для ідентифікації оригінальних версій програм і захисту продуктів від нелегального використання. Програмні засоби захисту реалізуються програмно, без використання фізичних характеристик носіїв інформації чи спеціального обладнання.

Мета роботи – проаналізувати різні методи захисту комп'ютерних програм від піратства; 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 характерні гнучкість у проектуванні, адаптованість до алгоритмів програм-об'єктів захисту та, порівняно з апаратними засобами, нижча вартість реалізації. До програмних методів захисту належать шифрування та заплутування, або обфускація (obfuscation), програмного коду. Такі методи ускладнюють структуру програми і, тим самим, процес зворотного 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 цій роботі для впровадження ідентифікаційних номерів використано особливості форматів виконуваних файлів, а саме наявність зарезервованих полів у заголовках цих файлів. Схожий підхід пропонують автори роботи [3], які записують секретне повідомлення в кінець секції програмного файла замість нульових байтів вирівнювання. Перевагою цього методу

є те, що розмір перетвореного файла залишається незмінним, а саме перетворення не впливає на виконання програми. Проте, як показано у статті [4], помітні розбіжності статистичних властивостей коду програми і зашифрованого повідомлення. Отже, такий метод нестійкий 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, змінивши вибір порядку оголошення змінних (1) і порядок 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 файла, зловмисник може отримати найбільш схожий до початкового код мовою високого рівня. Це спростить процес реверс-інжинірингу. Отже, можна зробити висновок, що прихована інформація буде захищенішою, якщо перетворення в програмних кодах здійснювати на нижчому рівні, використовуючи об'єктний чи асемблерний код. Це дасть змогу ускладнити можливий процес декомпіляції програмного коду. Наприклад, у 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 файла відповідного формату (EXE – виконувані файли в операційній системі DOS і PE – у Windows), без вихідних кодів. Програма подається в бінарному коді для обробки процесором. Сьогодні відомі методи реверс-інжинірингу, що застосовуються до двійкових програм з метою розуміння їх структури і алгоритмів. Це досягається перетворенням машинного коду на асемблерний. Далі відбувається процес декомпіляції цього коду у програму мовою 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]. За нашим підходом методика захисту програм полягає у комплексному застосуванні методів заплутування кодів та цифрових водяних знаків. Нею можна скористатись для захисту авторських прав та додаткового ускладнення реверс-інжинірингу.

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

Динамічні водяні знаки є стійкішими в цьому аспекті. Процес їх впровадження та вилучення, як правило, вимагає виконання програми. Наприклад, для створення водяного знака може використовуватись конкретний набір команд, які виконуються в окремій частині програми [8].

У запропонованому методі з метою впровадження водяного знака ми змінюємо асемблерний код програми. Припустимо, що всі інструкції (або команди) у програмі у певний спосіб

впорядковані, тобто під час виконання 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 виконуватимуться всередині конструкцій програми, але результат їх виконання ніяк не вплине на алгоритм її роботи. Змінюватимуться лише адреси оригінальних інструкцій програми.

Приклади таких інструкцій наведено в табл. 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,
OR,
ADD /SUB,
INC /DEC,
NEG,
NOT
	Логічні і арифметичні інструкції.
	• Логічні команди, які залишають значення регістра незмінним:
	XOR eax, 0;
	OR eax, 0;
	• сполучення протилежних команд
	ADD ebx,8
	SUB ebx, 8;
	INC ebx
	DEC ebx;
	• дублювання однакових команд
	NEG ecx
	NEG ecx;
	NOT ecx
	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: PUSH eax MOV eax, 2CH 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 мовою 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 файла застосовуємо дизасемблер, наприклад, “IDA Pro”, щоб отримати асемблерний код програми.
3. Водяний знак “25”, який будемо вставляти в асемблерний 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, щоб приховати місце розташування водяного знака та ускладнити процес дослідження програмного коду.
5. Розташування цих перетворень, тобто значення номерів адрес, за якими будуть вставлятися водяні знаки і додаткові інструкції, вибиратимуться випадково.
6. Далі компілюємо вже модифікований асемблерний файл. В результаті утворюється новий виконуваний файл програми з водяним знаком.

Отже, 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 табл. 2 показано, як змінився код програми мовою асемблера після впровадження ЦВЗ.

Таблиця 2

Асембрерний код програми hello.c	Код програми hello.c з водяним знаком

_main:
pushl %ebp
movl %esp, %ebp
subl $8, %esp
andl $-16, %esp
movl $0, %eax
 	
movl %eax, -4(%ebp)
movl -4(%ebp), %eax
call __alloca
call ___main
movl $LC0, (%esp)
 	
call _printf
movl $0, %eax
 	
leave
ret
.def _printf; .scl 2;
.type 32; .endef
	_main:
	pushl %ebp
	movl %esp, %ebp
	subl $8, %esp
	andl $-16, %esp
	movl $0, %eax
	xor $0, %eax
	add $8, %ebp
	sub $8, %ebp
	movl %eax, -4(%ebp)
	movl -4(%ebp), %eax
	xor $0, %ebp
	call __alloca
	call ___main
	movl $LC0, (%esp)
	push %ebp
	movl $23, %ebp
	pop %ebp
	call _printf
	movl $0, %eax
	leave
	ret
	.def _printf; .scl 2;
	.type 32; .endef

Водяний знак “25”

Обфускація

Вилучення водяних знаків

Доведення авторських прав та оригінальності програми здійснюється зчитуванням водяного знака. У цьому випадку його потрібно вилучити з перетвореного виконуваного файла.

Алгоритм вилучення водяних знаків:

1. Спочатку розкладаємо виконуваний файл на асемблерний код за допомогою дизасемблера IDA Pro.
2. Вибираємо з цього коду команди, 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 проаналізовано методи захисту програм від піратства та представлено методику для впровадження ЦВЗ в асемблерний код програми. Також запропоновано алгоритми для впровадження та вилучення ЦВЗ.

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

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

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 авторских 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.