УДК 004.41 DOI: 10.31673/2412-4338.2020.045051

Бондарчук А.П. *Державний університет телекомунікацій, Київ*
Корнага Я.І., Базалій М.Ю., Сергієнко П.А. *Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Київ*
Ільїн О.Ю. *Державний університет телекомунікацій, Київ*

МЕТОД ЗАХИСТУ ПРОГРАМНОГО КОДУ ВІД АНАЛІЗУ ЗАСОБАМИ ОБФУСКАЦІЇ

Анотація: *Розглянутий метод захисту програмного коду з використанням механізмів обфускації дозволяє проводити заплутування коду при розробці та при проходженні рефакторингу. Даний метод захищає від методів декомпіляції, які можуть застосовуватися як в мануальному режимі так і автоматично. Метою дослідження є розробка методу обфускації програмного коду для забезпечення захищеності від декомпіляції. Для досягнення поставленої мети вирішено такі завдання: проаналізовані алгоритми деобфускації в різних практичних методах; запропоновано новий метод обфускації програмного коду; представлено обфускований коду та визначено його властивості. Сформульовано основні етапи пропонованого підходу до обфускації програм: лексичний розбір; деструктурування коду; обфускація коду, обфускація змінних, обфускація констант. Визначено різні стратегії при синтезі обфускованних ідентифікаторів: генерація імен, що складаються з допустимих випадкових (псевдовипадкових) символів, довжиною із заданого інтервалу (фіксованою довжиною); генерація імен, що складаються з певної кількості повторюваних допустимих символів, в умовах, коли множину символів задано і заданий інтервал довжин ідентифікаторів; змішана стратегія з рівноймовірнісним вибором стратегій 1 і 2.*

Сформульовано дві оптимізаційні задачі: завдання мінімізації кількості операцій при породженні заданого набору констант при фіксованій множині безпосередньо визначених констант; завдання мінімізації кількості безпосередньо визначених констант серед варіантів з мінімальною складністю формул. Запропонований підхід може бути використаний в програмах, які мають певну кількість підпрограм з однаковим інтерфейсом. При цьому, незалежно від складності реалізації, код кожної підпрограми може бути перетворений в деструктурований код. Після цього можливо обчислити сумарну кількість виконаних операторів з урахуванням (оператора) виходу для всіх підпрограм.

Ключові слова: *обфускація, захист програмного коду, рефакторинг.*

Bondarchuk A.P. *State University of Telecommunications, Kyiv*
Kornaga Ya.I., Basaliy M.Yu., Sergienko P.A. *National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv*
Ilin O.A. *State University of Telecommunications, Kyiv*

METHOD OF PROTECTING SOFTWARE CODE FROM ANALYSIS BY OBFUSCATION MEANS

Abstract: *The considered method of program code protection with the use of obfuscation mechanisms allows to confuse the code during development and refactoring. This method protects against decompilation methods that can be applied both manually and automatically. The aim of the study is to develop a method of obfuscating software code to provide protection against decompilation. To achieve this goal, the following tasks are solved: analyzed algorithms of deobfuscation in various practical methods; a new method of obfuscating program code is proposed; submit obfuscated code. The main stages of the proposed approach to obfuscation of programs are formulated: lexical analysis; code destructuring; code obfuscation; obfuscation of variables; obfuscation of constants. Different strategies for the synthesis of obfuscated identifiers are defined: generation of names, consisting of admissible random (pseudo-random) symbols, length from the set interval (fixed length); generating names consisting of a certain number of repeated valid characters, in*

conditions where the plurality of characters is specified and the specified length of the lengths of the identifiers; mixed strategy with equal choice of strategies 1 and 2.

Two optimization problems are formulated: the task of minimizing the number of operations when generating a given set of constants with a fixed set of directly defined constants; the task of minimizing the number of directly defined constants among the options with minimal complexity of formulas. The proposed approach can be used in programs that have a number of routines with the same interface. In this case, regardless of the complexity of implementation, the code of each subroutine can be converted into destructured code. After that, it is possible to calculate the total number of feasible operators, taking into account (operator) output for all routines.

Keywords: *obfuscation, code protection, refactoring.*

Бондарчук А.П. *Государственный университет телекоммуникаций, Киев*
Корнага Я.И., Базалий М.Ю., Сергиенко П.А. *Национальный технический университет Украины «Киевский политехнический институт имени Игоря Сикорского», Киев*
Ильин О.И. *Государственный университет телекоммуникаций, Киев*

МЕТОД ЗАЩИТЫ ПРОГРАММНОГО КОДА ОТ АНАЛИЗА СРЕДСТВАМИ ОБФУСКАЦИЯ

Аннотация: *Рассмотренный метод защиты программного кода с использованием механизмов обфускации позволяет проводить запутывание кода при разработке и при прохождении рефакторинга. Данный метод защищает от методов декомпиляции, которые могут применяться как в ручном режиме, так и автоматически. Целью исследования является разработка метода обфускации кода для обеспечения защищенности от декомпиляции. Для достижения поставленной цели решены следующие задачи: проанализированы алгоритмы деобфускации в различных практических методах; предложен новый метод обфускации программного кода, представлен обфусцированный код и определены его свойства. Сформулированы основные этапы предлагаемого подхода к обфускации программ: лексический разбор, деструктурирование кода, обфускация кода, обфускация переменных, обфускация констант. Определены различные стратегии при синтезе обфусцированных идентификаторов: генерация имен, состоящих из допустимых случайных (псевдослучайных) символов, длиной из заданного интервала (фиксированной длиной); генерация имен, состоящих из определенного количества повторяющихся допустимых символов, в условиях, когда множество символов задано и задан интервал длин идентификаторов; смешанная стратегия с равновероятностным выбором стратегий 1 и 2.*

Сформулированы две оптимизационные задачи: задача минимизации количества операций при порождении заданного набора констант при фиксированной множестве непосредственно определенных констант; задача минимизации количества непосредственно определенных констант среди вариантов с минимальной сложностью формул. Предложенный подход может быть использован в программах, которые имеют определенное количество подпрограмм с одинаковым интерфейсом. При этом, независимо от сложности реализации, код каждой подпрограммы может быть преобразован в деструктурированный код. После этого можно вычислить суммарное количество выполнимых операторов с учетом (оператора) выхода для всех подпрограмм.

Ключевые слова*: обфускация, защита программного кода, рефакторинг.*

Вступ. Проблема захисту інтелектуальної власності за останні роки стає все більш і більш актуальною. Пов'язано це з великим зростанням кількості нових ідей, продуктів і технологій, зі спрощенням процедур їх опису та реалізації. Інформаційні технології є каталізатором інноваційного процесу. Вони запропонували безліч простих для користувача інструментів підтримки інноваційної діяльності і технології захисту інтелектуальної власності. У той же час прогрес в розвитку простих і зручних в експлуатації інструментальних засобів підтримки інновацій випереджає створення і розвиток інструментарію захисту інтелектуальної власності.

Аналіз літературних даних і постановка задачі. Інструментальні засоби для захисту інтелектуального коду розробляються багатьма компаніями: Microsoft, Corel, MindJet, CA Technologies і т.д. Ці програмні продукти завдяки підтримці різних мов програмування взяли на себе роль метасистем - інструментів створення інновацій в області програмного забезпечення з боку прикладних фахівців[1-4]. Також захист програмного коду проводиться і на рівні апаратної частини. Поява і розвиток таких мов опису апаратури як Verilog і VHDL, з одного боку, надзвичайно спростило і прискорило створення безлічі апаратних рішень, а, з іншого, зробило ряд вельми складних, інтелектуальноємних проектів уразливими до несанкціонованого використання [2-6,11].

В деяких випадках перешкодою на шляху витоку інтелектуальної власності в описаному контексті є технології обфускації. Неформально під обфускацією розуміється приведення вихідного тексту або виконуваного коду програми до виду, що зберігає її функціональність, але ускладнює аналіз, розуміння алгоритмів роботи і модифікацію після декомпіляції [7-9]. Таким чином, розробка нових методів обфускації з різними заданими властивостями становить значний інтерес [10].

Стаття присвячена розробленню методу обфускації програм. Особливістю запропонованого методу є використання лінійних конгруентних послідовностей в якості основи для відображення порядку розташування операторів мови на визначений функціональністю порядок виконання програми.

Мета і задачі дослідження. Метою дослідження є розробка методу обфускації програмного коду для забезпечення захищеності від декомпіляції.

Для досягнення поставленої мети вирішено такі завдання:

1. Проаналізовані алгоритми деобфускації в різних практичних методах.
2. Запропоновано новий метод обфускації програмного коду.
3. Представлено та досліджено обфускований код.

Метод обфускації програмного коду. Метод обфускації полягає в тому, що будь-який базовий блок програми імперативною мовою програмування можна представити у вигляді еквівалентної конструкції.

Позначимо:

$g : N \to N$ - бієкція, яка є елементом циклічної групи з кількістю елементів не менше $N+1$, при чому $\forall k(\mathrm{k}=1,2,...,\mathrm{N}+1)[\mathrm{n}_i = \mathrm{k} \Rightarrow \mathrm{x}_i = \mathrm{g}^k(\mathrm{x}_0)]$, де $g^k(\mathrm{x}) = \mathrm{g}\circ \mathrm{g}\circ \mathrm{g}\circ \mathrm{g}(\mathrm{x})(\mathrm{k}-\text{штук})$;

N - множина натуральних чисел;

x_0 - деяке початкове число з N;

x_i - значення ключа, по якому відбувається вибір виконуваного оператору з N;

n_i - номер оператора, що відповідає ключу x_i, $i = 1,2,...,N+1$.

Зауважимо, що різні циклічні утворювачі g можуть породжувати різні функціонально еквівалентні конструкції. Додаткову різноманітність у безліч цих конструкцій може внести можливість варіювання числа x_0.

Для того, щоб обговорюваний підхід на основі функціонально еквівалентного перетворення став дійсно практичним, необхідно підібрати таку бієкцію g, яка забезпечувала б заданий порядок породжуваної для неї циклічної групи і реалізація якої, з одного боку, була б обчислювально проста, а з іншого, легко б могла бути сама піддана обфускації.

Хорошим кандидатом на роль такого перетворення g є рекурентне співвідношення, що породжує так звані лінійні неконгруентні послідовності (ЛКП). Перетворення, що породжує ЛКП, має всього три параметри, відповідний вибір яких гарантує максимальний період послідовності, рівний одному з параметрів, званого модулем. Оскільки період ЛКП визначає порядок циклічної групи, то вимогу забезпечення будь-якого заданого порядку групи виконано.

Сформулюємо основні етапи пропонованого підходу до обфускації програм: лексичний розбір; деструктурування коду; обфускація коду; обфускація змінних; обфускація констант.

З перерахованих кроків алгоритму тільки лексичний розбір є типовим алгоритмом для певної конкретної мови програмування, на інших етапах слід зупинитися докладніше. В основу алгоритму деструктурування коду покладені принципи еквівалентного перетворення конструкцій програмування.

У коді потрібно зберігати відступи, що дозволяють простежити минулу структурованість програми. Також в цьому коді елементи замінити коментарями, що містять вихідні елементи, а усі оператори мови пронумеровати. Останній номер буде відповідати виходу з функції.

Візьмемо програму, де пронумеруємо оператори від 0 до 16, а 17 оператор буде пустий та відповідатиме за завершення програми.

Таким чином, для обфускації програми необхідна ЛКП з періодом не менш 18. Виберемо в якості модуля лінійного конгруентного перетворення число 18. Це означає, що перетворення $g(x) = |ax + c|_m$, де $m = 18$, а $|a|_m$ позначає найменше не негативне віднімання числа a за модулем m, може породити ЛКП максимального періоду 18, якщо множник a і приріст c будуть відповідати умовам, наприклад, $a = 7$, $c = 11$. В якості значення x_0 можна взяти будь-яке число від 0 до 17. Наприклад при $x_0 = 8$ ми отримуємо наступний період:

(13, 12, 5, 10, 9, 2, 7, 6, 17, 4, 3, 14, 1, 0, 11, 16, 15, 8).

Використовуючи ці значення ми можемо перетворити програмний код в обфускований код, який буде важко декомпілювати.

Лістинг обфускованого коду (функція обчислення мультиплікативно зворотнього елементу в системі найменших невід'ємних залишків $a^{-1}(b)$ з використанням розширеного алгоритму Евкліда):

```
public void Original(long a, long b, ref long GCD)
{
   long r;
   long r_pre;
   long r_pre_pre;
   long y;
   long y_pre;
   long y_pre_pre;
   r_pre_pre = b;
   r_pre = a;
   y_pre_pre = 0;
   y_pre = 1;
   while (r_pre != 0)
   {
      r = r_pre_pre % r_pre;
      y = y_pre_pre - y_pre * (r_pre_pre / r_pre);
      r_pre_pre = r_pre;
      r_pre = r;
      y_pre_pre = y_pre;
      y_pre = y;
   }
   GCD = r_pre_pre;
}
```

Лістинг обфускованого коду:

```
public void Obfuscation(long a, long b, ref long GCD)
{
   long r;
```

```
        long r_pre;
        long r_pre_pre;
        long y;
        long y_pre;
        long y_pre_pre;
        long xx;
        xx = 8;
        while (true)
        {
          xx = (7 * xx + 11) % 18;
          switch (xx)
          {
            case 0: if (y_pre_pre >= 0) xx = 16; break;
            case 1: GCD = r_pre_pre; break;
            case 2: r = r_pre_pre % r_pre; break;
            case 3: y_pre = y; break;
            case 4: y_pre_pre = y_pre; break;
            case 5: y_pre_pre = 0; break;
            case 6: r_pre_pre = r_pre; break;
            case 7: y = y_pre_pre - y_pre * (r_pre_pre / r_pre); break;
            case 8: break;
            case 9: if (!(r_pre != 0)) xx = 14; break;
            case 10: y_pre = 1; break;
            case 11: y_pre = b + y_pre_pre; break;
            case 12: r_pre = a; break;
            case 13: r_pre_pre = b; break;
            case 14: xx = 10; break;
            case 15: y_pre = y_pre_pre; break;
            case 16: xx = 15; break;
            case 17: r_pre = r; break;
            default: Console.WriteLine("xx=" + xx); break;
          }
        }
      }
```

На лістингу обфускованого коду присутні відступи, що демонструють структурну організацію нового еквівалентного коду. Крім того, додані мітки і аналог оператора goto, а також номери операторів розміщенні в порядку виконання і значень елементів періоду, що передуються. Для даного випадку алгоритм перетворення виявляється дуже простим та забезпечує обфускацію функції.

Проблеми з семантикою ідентифікаторів при рефакторінгу збільшують час, необхідний для розуміння алгоритму, в середньому в два рази по відношенню до ситуації, коли потік управління залежить від безлічі «непрозорих» предикатів. Ще більшого ефекту можна домогтися при реалізації ідеї повторного використання ідентифікаторів, коли семантично обумовлено різні ідентифікатори, але вони не перетинаються по області використання, при збігу типів, об'єднуються за обфускованим ім’ям.

При синтезі обфускованих ідентифікаторів можуть застосовуватися різні стратегії:

1) генерація імен, які складаються з допустимих випадкових (псевдовипадкових) символів, довжиною із заданого інтервалу (фіксованою довжиною);

2) генерація імен, що складаються з певної кількості повторюваних допустимих символів, в умовах, коли безліч символів задано і заданий інтервал довжин ідентифікаторів;

3) змішана стратегія з рівноймовірнісним вибором стратегій 1 і 2.

Результат обфускації імен змінних при фіксованій довжині ідентифікатора, що дорівнює 8, наведено в лістингу:

```
public void jfU5puyE(long quEvc8xr, long kRaZ2Ipi, ref long M5XX4pmi)
{
  long I08HujHx;
  long jBsfK8YG;
  long swKZJf6e;
  long FQYIO5Bp;
  long p7e5LjKj;
  long jJBgVzoy;
  long RcmiOrCO;
  RcmiOrCO = 8;
  while (true)
  {
    RcmiOrCO = (7 * RcmiOrCO + 11) % 18;
    switch (RcmiOrCO)
    {
      case 0: if (jJBgVzoy >= 0) RcmiOrCO = 16; break;
      case 1: M5XX4pmi = swKZJf6e; break;
      case 2: I08HujHx = swKZJf6e % jBsfK8YG; break;
      case 3: p7e5LjKj = FQYIO5Bp; break;
      case 4: jJBgVzoy = p7e5LjKj; break;
      case 5: jJBgVzoy = 0; break;
      case 6: swKZJf6e = jBsfK8YG; break;
      case 7: FQYIO5Bp = jJBgVzoy - p7e5LjKj * (swKZJf6e / jBsfK8YG); break;
      case 8: break;
      case 9: if (!(jBsfK8YG != 0)) RcmiOrCO = 14; break;
      case 10: p7e5LjKj = 1; break;
      case 11: p7e5LjKj = kRaZ2Ipi + jJBgVzoy; break;
      case 12: jBsfK8YG = quEvc8xr; break;
      case 13: swKZJf6e = kRaZ2Ipi; break;
      case 14: RcmiOrCO = 10; break;
      case 15: p7e5LjKj = jJBgVzoy; break;
      case 16: RcmiOrCO = 15; break;
      case 17: jBsfK8YG = I08HujHx; break;
      default: Console.WriteLine("RcmiOrCO=" + RcmiOrCO); break;
    }
  }
}
```

В цілому ряді випадків, знання констант, використовуваних в програмі, дозволяє істотно полегшити розуміння семантики алгоритму. Тому приховування явного використання констант може стати однією з перешкод на шляху рефакторингу програм при ручному аналізі коду. Для автоматизованого аналізу обфускація констант навряд чи може стати серйозною перешкодою, але для непрофесійного ручного аналізу вона може створити помітні проблеми.

Зауважимо, що константи основних базових типів мов програмування легко можуть бути представлені цілими невід'ємними числами. Константа цілого типу – це ціле невід'ємне зі знаком, дійсного типу – це ціле невід'ємне зі знаком для мантиси і ціле невід'ємне зі знаком для порядку, константа рядкового типу – це набір відповідних символів, інакше, набір цілих невід'ємних чисел. Таким чином, завдання обфускації констант базових типів можна звести до задачі обфускації цілих невід'ємних констант.

Нехай C - безліч цілих невід'ємних констант, використовуваних в програмі, $C \subset Z^+$. Z^+ - множина цілих невід'ємних чисел. Нехай $m = \max C + 1$, тобто $C \subseteq Z = \{0,1,...,m-1\}$. Нехай R - множина (допоміжних) констант, які представлені в програмі не тільки своїми ідентифікаторами, а й явно записаними цілими числами без знаку. Тепер можна ввести безліч констант, використовуваних в програмі і заданих явно,

$$C_0(R) = C \cap R .$$

На першому кроці, використовуючи дану безліч констант, можна її розширити за допомогою ряду лінійних комбінацій:

$$C_1(R, A_0, x_0) = C_0(R) \cup \{c : c \in C \setminus C_0(R), c = \sum_{a \in A_0(c)} x_0(c,a)\, a\},$$

де $A_0 : C \setminus C_0(R) \to 2^{C_0(R)}$ - деяка функція, що представляє собою підмножину констант з множини $C_0(R)$, які використовуються в лінійній комбінації для породження деяких констант з множини $C \setminus C_0(R)$ на першій ітерації;

$x_0 : [C \setminus C_0(R)] \times C_0(R) \to C_0(R)$ - функція, яка при породженні деяких констант c з множини $C \setminus C_0(R)$ зіставляє кожному члену лінійної комбінації з множини $A_0(c)$ коефіцієнт з множини $C_0(R)$.

На наступних кроках ми можемо поступово розширювати множину констант, визначених через константи попередніх кроків:

$$C_{k+1}(R, A_k, x_k) = C_k(R, A_{k-1}, x_{k-1}) \cup \{c : c \in C \setminus C_k(R, A_{k-1}, x_{k-1}), c = \sum_{a \in A_k(c)} x_k(c,a)\, a\},$$

де $A_k : C \setminus C_k(R, A_{k-1}, x_{k-1}) \to 2^{C_k(R, A_{k-1}, x_{k-1})}$ - деяка функція, що представляє підмножину констант з множини $C_k(R, A_{k-1}, x_{k-1})$, використовуваних в лінійній комбінації для породження деяких констант з множини $C \setminus C_k(R, A_{k-1}, x_{k-1})$ на $(k+1)$ ітерації.

$x_k : [C \setminus C_k(R, A_{k-1}, x_{k-1})] \times C_k(R, A_{k-1}, x_{k-1}) \to C_k(R, A_{k-1}, x_{k-1})$ - функція, яка при породженні деяких констант c з множини $C \setminus C_k(R, A_{k-1}, x_{k-1})$ зіставляє кожному члену лінійної комбінації з множини $A_k(c)$ коефіцієнт з множини $C_k(R, A_{k-1}, x_{k-1})$, $k = 1, 2, ...$

Очевидно, що для деяких R на деякій ітерації k' буде досягнуто визначення всіх необхідних констант:

$$C_{k'}(R, A_{k'-1}, x_{k'-1}) = C .$$

Визначимо множину можливих параметрів породження заданої множини констант при фіксованій множині безпосередньо заданих констант:

$$V(R) = \{(k', (A_k, x_k)_{k=0}^{k=k'-1}) : C_{k'}(R, A_{k'-1}, x_{k'-1}) = C\} .$$

Тепер можна сформулювати ряд оптимізаційних задач.

1) Завдання мінімізації кількості операцій при породженні заданого набору констант C при фіксованій множині безпосередньо визначених констант R:

$$\pi(R) = (k'', (A'_k, x'_k)_{k=0}^{k=k''-1})(R) = \underset{V(R)}{\operatorname{argmin}} \sum_{k=1}^{k'} \sum_{c \in C_k(R, A_{k-1}, x_{k-1})} |A_k(c)|$$

2) Завдання мінімізації кількості безпосередньо визначених констант серед варіантів з мінімальною складністю формул:

$$R^{'} = \arg\min_{R \in Dom\pi} |R|,$$

де $Dom\pi$ - область визначення функції π.

Значення функції $\pi(R^{'})$ можна розглядати як оптимальну (в певному сенсі) схему обфускації констант. Слід, однак, відзначити, що алгоритм точного розв'язання задачі обчислення $\pi(R^{'})$ лежить в класі EXPSPACE. Тому на практиці завдання обчислення $\pi(R^{'})$ вирішується будь-яким наближеним емпіричним методом.

Висновок. Пропонований підхід до обфускації може бути використаний в програмах, які мають певну кількість підпрограм з однаковим інтерфейсом. При цьому, незалежно від складності реалізації, код кожної підпрограми може бути перетворений в деструктурований код. Після цього можливо вичислити сумарну кількість здійсненних операторів з урахуванням (оператора) виходу для всіх підпрограм і побудувати ЛКП з відповідною довжиною періоду. Якщо об'єднати ці підпрограми під одним ім'ям, а в їх стандартний інтерфейс додати ознаки вибору конкретної підпрограми і, далі, використовуючи ці ознаки ініціювати x_0 значеннями з періоду з кроком в довжину кожної з підпрограм, то в одну обфускаційну конструкцію можливо укласти весь набір цих підпрограм. Ризик реалізації загрози в методі було запропоновано піддати декомпозиції в рамках деякої логіко-ймовірнісної моделі паралельно з декомпозицією об'єктної моделі системи. При цьому ймовірність реалізації активної загрози (атаки з боку раціонального зловмисника) тут тим вище, чим вище ступінь рефакторингу цільового класу (об'єкта). У цьому сенсі запропонований підхід до обфускації коду може помітно знизити ймовірність реалізації цілеспрямованої атаки.

Список використаної літератури

1. Бородин А.В., Долгушев Е.Д. Обфускация пула констант как задача построения минимальной системы целочисленных линейных комбинаций. Образование, наука, бизнес: развитие и перспективы: материалы III международной научно-практической конференции. 2016. С. 7 – 13.

2. Бородин А.В., Долгушев Е.Д. Постановка задачи обфускации пула констант. Новое слово в науке: перспективы развития: материалы IX Международной научно–практической конференции. 2016. № 3 (9). С. 89 – 93.

3. Варновский Н.П., Захаров В.А., Кузюрин Н.Н. Математические проблемы обфускации. Труды конференции "Математика и безопасность информационных технологий". 2014. С. 54 – 72.

4. Варновский Н.П., Захаров В.А., Подловченко Р.И., Щербина В.С., Кузюрин Н.Н., Шокуров А.В. О применении методов деобфускации программ для обнаружения сложных компьютерных вирусов. Известия ЮФУ. Технические науки. 2006. №7(62). С. 18 – 27.

5. Варновский Н.П., Захаров В.А., Кузюрин Н.Н., Шокуров А.В. Современное состояние исследований в области обфускации программ: определения стойкости обфускации. Труды Института системного программирования РАН. 2014. Т. 26. В. 3. С. 167 – 198.

6. Коробейников А.Г., Кутузов И.М. Алгоритм обфускации. Кибернетика и программирование. 2015. №3. С. 1 – 8.

7. Коробейников А.Г. , Кутузов И.М., Колесников П.Ю. Анализ методов обфускации. Кибернетика и программирование. 2017. №1. С. 31 – 37.

8. Лифшиц Ю.М. Запутывание (обфускация) программ. Санкт-Петербургское отделение математического института им. В. А. Стеклова РАН., 2018.

9. Barak B., Goldreich O., Impagliazzo R., Rudich S., Sahai A., Vadhan S., Yang K. On the (im)possibility of obfuscating programs. Journal of the ACM. 2018. № 6.P. 40 – 48 p.

10. Drape S. , Voiculescu I. Creating transformations for matrix obfuscation. Static Analysis: 16th International Static Symposium. 2019. P. 273 – 292.

11. Drape S. , Voiculescu I. The Use of Matrices in Obfuscation. Oxford University Computing Laboratory. 2018. № 08. P. 12 – 28.

References

1. Borodin A.V., Dolgushev E.D. (2016) Obfuscation of a pool of constants as a problem of constructing a minimal system of integer linear combinations. *Education, Science, Business: Development and Prospects: Proceedings of the III International Scientific and Practical Conference.* P. 7 - 13.

2. Borodin A.V., Dolgushev E.D. (2016) Statement of the problem of constant pool obfuscation. *A new word in science: development prospects: materials of the IX International scientific and practical conference.* No. 3 (9). P. 89 - 93.

3. Varnovsky N.P., Zakharov V.A., Kuzyurin N.N. (2014) Mathematical problems of obfuscation. *Proceedings of the conference "Mathematics and Security of Information Technologies".* P. 54 - 72.

4. Varnovsky N.P., Zakharov V.A., Podlovchenko R.I., Shcherbina V.S., Kuzyurin N.N., Shokurov A.V. (2006) On the application of software deobfuscation methods to detect complex computer viruses. *Izvestia SFedU. Technical science*. No. 7 (62). P. 18 - 27.

5. Varnovsky N.P., Zakharov V.A., Kuzyurin N.N., Shokurov A.V. (2014) The current state of research in the field of program obfuscation: determining the persistence of obfuscation. *Proceedings of the Institute for System Programming, RAS.* Vol. 26. Issue.3. P. 167 - 198.

6. Korobeynikov A.G., Kutuzov I.M. (2015) Obfuscation algorithm. *Cybernetics and Programming.* No. 3. P. 1 - 8.

7. Korobeynikov A.G., Kutuzov I.M., Kolesnikov P.Yu. (2017) Analysis of obfuscation methods. *Cybernetics and Programming*. No. 1. P. 31 - 37.

8. Lifshits Yu.M. Obfuscation of programs. (2018) *St. Petersburg Department of the Mathematical Institute.* V. A. Steklov RAN.

9. Barak B., Goldreich O., Impagliazzo R., Rudich S., Sahai A., Vadhan S., Yang K. (2018) On the (im)possibility of obfuscating programs. *Journal of the ACM*. No 6. P. 40 – 48.

10. Drape S., Voiculescu I. (2109) Creating transformations for matrix obfuscation. *Static Analysis: 16th International Static Symposium.* P. 273 – 292.

11. Drape S., Voiculescu I. (2018) The Use of Matrices in Obfuscation. *Oxford University Computing Laboratory.* No 08. P. 12 – 28.