© Stepanenko I., Kinzeryavyy V., Nagi A., Lozinskyi I. Modern obfuscation methods for secure coding // Ukrainian Scientific Journal of Information Security, 2016, vol. 22, issue 1, p. 32-37.

СУЧАСНІ ОБФУСКАЦІЙНІ МЕТОДИ ЗАХИСТУ ПРОГРАМНОГО КОДУ

Ірина Степаненко[1], Василь Кінзерявий[1], Абду Наджі[1,2], Іван Лозінський[1]

[1]Національний авіаційний університет, Україна
[2]Міністерство нафти, газу і корисних копалин Ємену, Республіка Ємен

СТЕПАНЕНКО Ірина Віталіївна

Рік та місце народження: 1994 рік, м. Херсон, Херсонська область, Україна.
Освіта: Національний авіаційний університет, 2012 рік.
Посада: студент кафедри безпеки інформаційних технологій з 2012 року.
Наукові інтереси: інформаційна безпека, комп'ютерна безпека, захист програмного забезпечення.
E-mail: stepanenko.iryna.v@gmail.com

КІНЗЕРЯВИЙ Василь Миколайович, к.т.н.

Рік та місце народження: 1985 рік, м. Кам'янець-Подільський, Хмельницька область, Україна.
Освіта: Національний авіаційний університет, 2007 рік.
Посада: доцент кафедри засобів захисту інформації.
Наукові інтереси: криптографія та криптоаналіз блокових симетричних шифрів.
Публікації: більше 80 друкованих наукових праць, серед яких наукові статті у вітчизняних та міжнародних фахових виданнях, патенти та авторські свідоцтва.
E-mail: v.kinzeryavyy@gmail.com

НАДЖІ Абду Ахмад Алі

Рік та місце народження: 1970 рік, м. Тайз, Республіка Ємен.
Освіта: Московська державна академія приладобудування та інформатики, 1997 рік.
Посада: адміністратор комп'ютерних технологій, здобувач кафедри безпеки інформаційних технологій НАУ з 2014 року.
Наукові інтереси: інформаційні технології моделювання, теорія комплексних мереж, інформаційна безпека, сучасні системи дистанційної освіти.
Публікації: більше 20 наукових статей у провідних фахових виданнях.
E-mail: abdonagi@hotmail.com

ЛОЗІНСЬКИЙ Іван Любомирович

Рік та місце народження: 1995 рік, смт. Козова, Тернопільська область, Україна.
Освіта: Національний авіаційний університет, 2012 рік.
Посада: студент кафедри безпеки інформаційних технологій з 2012 року.
Наукові інтереси: інформаційна безпека, комп'ютерна безпека, криптографія, комп'ютерні мережі.
Публікації: 3 тези доповідей на конференціях.
E-mail: i.l.lozinsky@gmail.com

Анотація. *У даній статті проаналізовано існуючі класифікації обфускаційних методів захисту програмного коду. На підставі аналізу даних класифікації встановлено, що захист програм відбувається не належним чином, не враховані певні методи заплутування, що можуть значною мірою підвищити стійкість коду. Тому у статті наведено сучасні обфускаційні методи захисту програмного коду та розроблена узагальнена класифікація даних методів. У подальшому, на основі розробленої класифікації, планується створити програмне забезпечення, відповідно до представленого алгоритму, що дозволить заплутувати код, ускладнити процес аналізу програми та забезпечити захист програмного коду від несанкціонованого розкриття. Отриманні результати розширять знання з використання обфускаційних методів захисту, забезпечать розробку сучасних та ефективних систем захисту програм.*

Ключові слова: *захист програмного коду, обфускація, класифікація обфускаційних методів; обфускаційний алгоритм.*

Вступ

Сучасні компанії, розробники програмного забезпечення, мають надавати своїм клієнтам якісний та захищений інформаційний продукт. Проте при постійному зростанні кількості такого продукту, постає питання про забезпечення його захищеності. Одним із способів захисту є використання обфускаційних методів захисту програмного коду. Обфускаційні методи дозволяють заплутувати код програми, тобто приводити вихідний текст до виду, що зберігає функціональність програми, але ускладнює аналіз, розуміння алгоритмів роботи та проведення модифікації програми.

Даючи користувачам доступ до виконуючих файлів певного програмного забезпечення, компанія потенційно розкриває дані про розроблений код програми, його структуру. Зловмисники чи конкуренти можуть проаналізувати виконуючі файли програми, відновити алгоритм, знайти незахищені місця у коді та провести його модифікацію. Даний процес можливий за допомогою деобфускаційних програм: IDA Pro, Ariadne, JSNice, iMPROVE .NET, De4dot тощо, що дозволяють читати та модифікувати виконувані файли, переводити їх у машинний код або перетворювати частину коду в зручне для аналізу проміжне представлення. Для компаній-розробників програмного забезпечення виникає певна проблема: написаний програмний код може бути зрозумілий, проаналізований та розібраний зловмисником чи конкурентом. Тому є необхідність у розробці сучасних та вдосконалення існуючих методів захисту програмного коду, щоб він був незрозумілий для аналізу зловмисником, і, при цьому, програма виконувала свої функції.

Аналіз відомих досліджень і постановка завдання

Відомі програми для обфускаційних перетворень: ThinApp, Obfuscar, .NET Reactor, CilSecure, C# Source Code Obfuscator тощо, виконують лише такі перетворення як зміна порядку виклику даних, перетворення імені змінних, вставлення частини коду, який має нульовий ефект. Ці методи захисту програмного коду не можна вважати досить ефективними. Необхідно розробити механізм обфускації, який буде виконувати не тільки вище перераховані перетворення, а й перетворення основних елементів коду, використання методів клонування, мертвого коду, змін умов циклу тощо. Однак для створення сучасного обфускаційного механізму необхідно визначити повний перелік (класифікацію) обфускаційних перетворень, що можуть бути застосовані для захисту програмного коду.

Багато зарубіжних авторів розробляють різні алгоритми захисту програми за допомогою обфускаційних методів. У роботах [1-10] наведені переліки (класифікації) основних обфускаційних методів, показані приклади реалізації деяких з них, проте дані переліки обфускаційних методів на сьогодні є неповними. Тому **актуальним завданням** є аналіз обфускаційний методів та розробка їх узагальненої класифікації.

Метою даної роботи є аналіз та розробка узагальненої класифікації обфускаційних методів захисту програмного коду, що дозволить у майбутньому розробляти надійні алгоритми захисту програмного коду від деобфускації.

Основна частина

Провівши аналіз існуючих робіт, необхідно зазначити, що обфускаційні методи розділялися на певні види (категорії). У роботі Крістіана Коллберга, Кларка Томборсона та Дугласа Лоу [4] методи заплутування коду були розподілені за чотирма критеріями: обфускація даних, обфускація керування, обфускація трасування та запобіжні перетворення. Проте авторами не враховано використання міток у коді програми, реструктурування даних масивів, габаритне оголошення змінних. Використання даних методів заплутування збільшить нечитабельність, незрозумілість коду та ускладнить процес деобфускації.

Корейські вчені Ілсун Ю та Кенгбін Юім [7] створили класифікацію методів заплутування коду, яка базується на тих методах, що використовуються в процесі створенні шкідливого програмного забезпечення. До них відносять: метод мертвого коду, реєстрація перерозподілу, реорганізація підпрограм, інструкція заміни, метод перестановки коду, метод об'єднання. Але не були враховані пунктуаційні перетворення програмного коду, метод клонування, зміни умов циклу та використання міток у програмі.

Вчені Арізонського університету запропонували класифікувати методи [1] наступним чином: вхідний-вихідний потоки даних, трасування спрощення, управління потоками структури коду. Дана класифікація спрямована на перетворення структури коду, перенаправлення потоку даних, порушення логічного порядку виконання операцій. Однак в даній класифікації невраховані методи заплутування змінних та пунктуаційні перетворення у програмному коді.

У своїх роботах [1-10] вчені використовують різні обфускаційні перетворення, по різному визначають критерії розподілу. Проте, при побудові класифікацій не були враховані певні методи, які значною мірою можуть підвищити рівень стійкості програмного коду при деобфускації.

З огляду на це, була запропонована нова класифікації методів захисту програмного коду, в якій враховані усі сучасні методи обфускації. Дані методи заплутування програмного коду було розділено за трьома критеріями: пунктуаційні перетворення, перетворення змінних та перетворення структури коду. На рис. 1 показана схема класифікації даних методів.

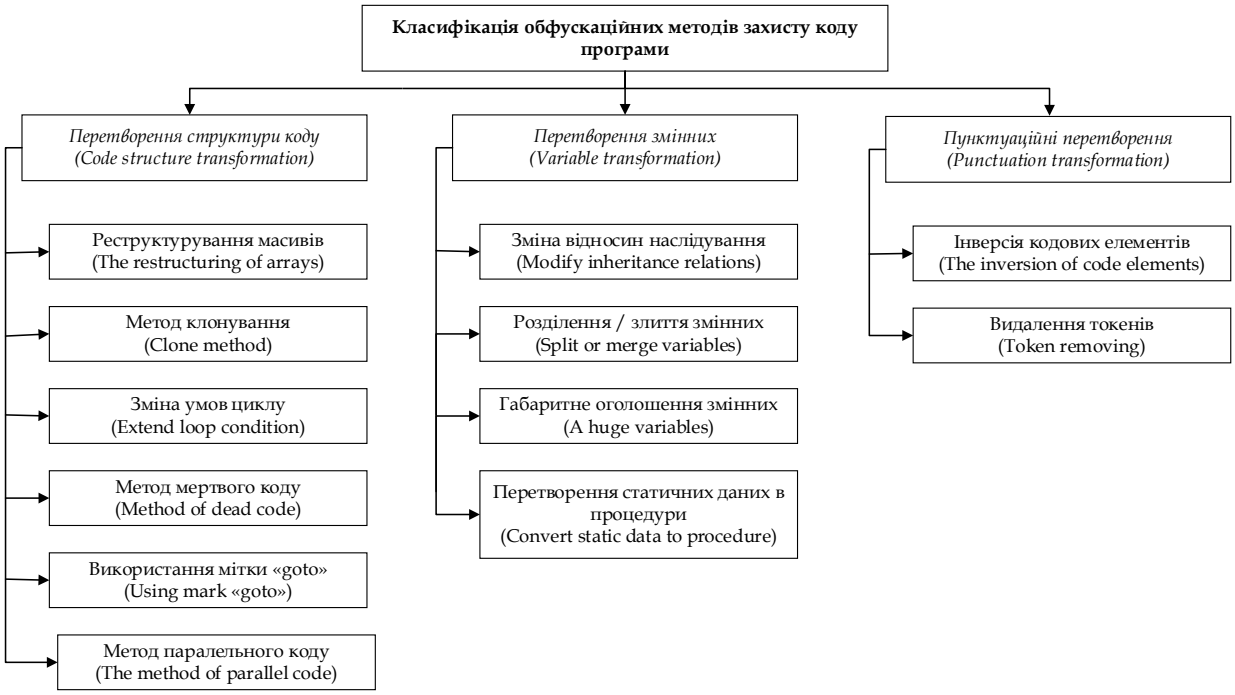

Рис. 1. Класифікація обфускаційних методів захисту коду програми

Для захисту програмного коду необхідно використовувати більшу кількість зазначених обфускаційних методів. Наприклад, можна використати наступний алгоритм, який складається з трьох етапів.

Етап. 1. Розробники програмного коду в першу чергу повинні провести механізм обфускації з самою структурою коду, для цього з представлених шести методів необхідно обрати не менше трьох методів перетворень та почергово чи одночасно використати їх для зміни коду.

Етап. 2. Далі необхідно виконати перетворення змінних. Аналогічно попередньому етапу, обирається декілька методів перетворення змінних та почергово чи одночасно відбувається обфускаційний процес.

Етап. 3. Необхідно виконати пунктуаційні перетворення: провести інверсію кодових елементів та видалити усі токени. Використання обфускаційних методів у такій послідовності повинно забезпечити надійний захист програмного коду, що сильно ускладнить проведення аналізу структури та роботи програми.

Коротко опишемо усі вище зазначені обфускаційні методи та наведемо приклади їх використання (приклади написані мовою С++).

Перетворення структури коду (Code structure transformation)

1. Перетворення структури коду – це метод, заснований на вивченні структури коду програми та внесення необхідних перетворень для унеможливлення аналізу коду. Даний вид обфускації заснований на реструктуруванні даних, зміни умов роботи циклів, злиття та розділення масивів. Нижче наведені основні методи даного виду.

1.1. *Реструктурування даних* [1-3, 6-10] – це метод, змінена традиційного порядку проходження елементів у масиві, циклах. Для нього характерне розбиття великих елементів в малі масиви (цикли), підмасиви (підцикли) або згортання багатовимірних масивів у одномірні. Однак при такій змінні необхідно враховувати отримані результати перетворення, щоб вони не порушували функціональну організацію програми. Приклад даного обфускаційного методу наведений на рис. 2.

```
int main()                          int main()
{float a[6] = { 10.0, 30.0,         {float a[2][3] = { { 10.0, 36.0,
5.0, 12.0,20.0,35.0}, m, d;         5.0}, {72.0, 20.0, 38.6}}, m, d;
int i;                              int i,j;
for (i = 0; i <= 5; i++)            for (i = 0; i <= 1; i++){
{a[i] = a[i] / 10;          ⇨       for (j = 0; j <= 2; j++)
m = modf(a[i], &d);                 {a[i][j] = a[i][j] / 10;
if (m != 0) a[i] = a[i] * 10;       m = modf(a[i][j], &d);
else a[i] = 0;                      if (m != 0) a[i][j]= a[i][j] * 10;
cout<<"a["<<i<<"]="<<a[i]           else a[i][j] = 0;
<<"\n";}                            cout<<"a["<<i<<"]["<<j<<"]="
return 0;}                          <<a[i][j]<<"\n";}}
                                    return 0;}
```

Рис. 2. Приклад роботи методу реструктурування даних

1.2. *Метод клонування* [2, 4, 10] – це метод, що дозволяє ускладнити аналіз використаних функцій, змінних, класів у коді програми. Процес клонування полягає у виділенні певної функції X, яка часто зустрічається в програмі. Наприклад, за допомогою неявного предикату (безумовно вірного ствердження) пов'язується з клоном X1. В результаті цього, у зловмисника створюється уявлення, що данні функції є різними та виконують різні операції, хоча насправді буде не важливо, яка функція буде обрана. Приклад див. на рис. 3.

```
int main()                          int main()
{float a=10, ab;                    {int a=10,ac=1,ab=0, t=1;
ab=2*a-(17-(a-7));          ⇨       if ((((t*a + 20)/6) % 19) != 0)
cout<<ab<<"\n";                     ab = 2*a - (17 - (a - 7)); if ((2
return 0;}                          * a % 2) == 0) ac = 3*t - 24;
                                    cout<<ab<<"\n";
                                    cout<<ac<<"\n";
                                    return 0;}
```

Рис. 3. Приклад роботи методу клонування

1.3. *Зміна умови циклу* [4, 6, 10]. Основною особливістю даного методу заплутування програми – приховання умови виконання циклу, шляхом створення більш складної умови. Основна ідея полягає в тому, щоб розширити умову початкового циклу так, аби вона не впливала на обрахунок необхідної змінної. Далі наведений приклад, коли для існуючої умови додається предикат (умова, яка безумовно вірна), який в свою чергу ніяк не буде впливати на значення необхідної змінної. Приклад представлений на рис. 4.

```
int main()                          int main()
{int a, n = 0, b=1, s;              {int a, n = 0, b = 1, s, l=0;
cout << "Enter the                  cout << "Enter the
number\n";                          number\n";
cin >> a;                           cin >> a;
cout << "Enter limit\n";            cout << "Enter limit\n";
cin >> s;                           cin >> s;
do { n = n +1; b = b + 1; }         do { n = n + 1; b = b + 1;}
while (s > pow(a, b));              while ((s >= pow(a, b))
cout << "Quantity = "               && ((a*a*(a + 1) *(a + 1)
<< n << "\n";                       % 4 == 0)));
return 0;}                          cout << "Quantity = "
                                    << n << "\n";
                                    return 0;}
```

Рис. 4. Приклад роботи методу зміни умови циклу

1.4. *Метод мертвого коду* [1, 2, 3, 7] – використання надлишкових операції, для розширення арифметичних виразів, що знаходяться в коді програми. Мертвий код може зустрічатися в найрізноманітніших частинах програми. Це може бути як процедури та функції, класи, змінні, константи так і мертві файли або модулі програм. На рис. 5 наведений приклад, де елементом мертвого коду буде змінна z.

```
int main()                          int main()
{int x[25], i = 0, max;             {int x[25], i = 0, max, z;
for (i = 0; i < 25; i++)            for (i = 0; i < 25; i++)
{srand(unsigned(time(N              {srand(unsigned(time(N
ULL)));                             ULL)));
while (i < 25)                      while (i < 25)
{x[i] = rand() % 30;                {x[i] = rand() % 30;
i++;} }                             i++; } } max = x[0];
max = x[0];                         for (i = 0; i < 25; i++)
for (i = 0; i < 25; i++)            {if (x[i] > max)
{if (x[i] > max) max =              {max = x[i];
x[i]; }                             z = x[i] * 12 + 25;}}
cout<<"\n max ="<<                  cout<<"\n max ="<<
max;                                max;
return 0;}                          return 0;}
```

Рис. 5. Приклад роботи методу мертвого коду

1.5. *Використання міток* [3, 8, 9]. У програмуванні рідко використовують команда goto, проте вона може бути корисною при обфускації. Велика кількість таких міток може з легкістю заплутати зловмисника у структурі роботи програми та зробити код нечитабельним. Приклад наведений на рис. 6.

```
int main()                          int main()
{float a, b, c, x, y;               {float a, b, c, x, y;
int k = 0;                          int k = 0;
cout<<"Enter the sizes a            cout<<"Enter the sizes a
brick\n";                           brick\n";
cin>>a>>b>>c;                       cin>>a>>b>>c;
cout<<"Enter the sizes              cout<<"Enter the sizes
the framei\n";                      the framei\n";
cin>>x>> y;                         cin>>x>>y;
if ((x >= a &&y >= b) |             if ((x >= a &&y >= b) |
(x >= b && y >= a) | (x             (x >= b && y >= a) | (x
>= a && y >= c) | (x >=             >= a && y >= c) | (x >=
c && y >= a) | (x >= b              c && y >= a) | (x >= b
&& y >= c )| (x >= c &&             && y >= c) | (x >= c &&
y >= b)) cout<<"Brick               y >= b)) goto m1;
will held\n";                       else goto m2;
else cout<<"Brick will              m1: cout<<"Brick will
not pass\n";                        held\n"; goto m3;
return 0;}                          m2: cout<<"Brick will
                                    not pass\n";
                                    m3: return 0;}
```

Рис. 6. Приклад роботи методу використання міток

1.6. *Метод паралельного коду* [3, 4] полягає в поділі коду на окремі незалежні ділянки, які під час роботи програми будуть виконуватися паралельно. Це робиться для того, щоб збільшити час декомпілювання. Розглянемо приклад на рис. 7.

```
int main()                          int main()
{int a, b, c=0, c1=0, c2=0;         {int a, b, c=0, c1=0, c2=0,
cout<<"Enter two                    c21=0, c22=0;
numbers\n";                         cout<<"Enter two
cin >> a >> b ;                     numbers\n");
if (a >= b) {c1 = a + b;            cin >> a >> b ;
cout<<"Answer:"<< c1;}              if (a >= b) {c1 = a + b;
if (a<b) {c2 = pow(b, 2)            cout << "Answer: " <<
+ 2 * (a*b); cout <<                c1;}
"Answer:"<< c2;}                    if (a < b) { c21 = pow(b,
return 0;}                          2); c22 = 2 * (a*b); c2 =
                                    c21 + c22; cout <<
                                    "Answer: " << c2; }
                                    return 0;}
```

Рис. 7. Приклад роботи методу паралельного коду

Перетворення змінних (Variable transformation)

2. Перетворення змінних – це метод заплутування коду за допомогою певних трансформацій безпосередньо зі значенням змінних. Дані перетворення направлені на зміну потоку даних та на трансформацію самих типів даних.

2.1. *Зміна відносин наслідування* [2, 4, 6, 10] використовується для перетворення ієрархій успадкування змінних у певному класі, що здійснюється шляхом ускладнення ієрархії успадкування за допомогою створення додаткових класів або використання помилкового поділу класів.

2.2. *Розділення / злиття змінних* [2, 4, 7, 9] – необхідне для ускладнення використовуваних структур даних в програмі. Загальний принцип для даного методу: з'єднання незалежних даних або поділ залежних, при цьому створюється новий потік даних. Змінні можуть бути об'єднані лише у тому випадку, якщо загальний розмір двох змінних, не перевищує розміру об'єднаної змінної. Для прикладу буде використовувати спосіб розділення змінних для обрахунку логічних операцій наведений на рис. 8.

2.3. *Габаритне оголошення змінних* – даний метод обфускації використовується для нагромадження коду програми, тим самим аналіз програми програмістом буде ускладнюватись. Також для даного методу можна використовувати метод додавання, віднімання, множення та ділення

змінних на великі числа. Проте необхідно враховувати, що внесені зміни не повинні виплавати на результат роботи програми.

```
int main()                          int main()
{bool a, b, c, c1;                  {int a1, a2, b1, b2, c1, c2;
a = true; b = false;                a1 = 1; a2 = 0; b1 = 0; b2 = 0;
c = false; c1 = false;              c1 = ((a1 || a2) && (b1 ||
c = a && b;                  ⇨      b2));
c1 = a || b;                        c2 = ((a1 || a2) || (b1 ||
cout << "c = " << c;                b2)); cout << "c1 = " << c1;
cout << "c1 = "<< c1;               cout << "c2 = " << c2;
return 0;}                          return 0;}
```

Рис. 8. Приклад роботи методу розділення/ злиття змінних

Розглянемо приклад знаходження значення двох змінних (див. рис. 9). Для обфускації використовуються числа, складні для сприйняття.

```
int main()                          int main()
{int a[10] = { 117, 30, 5,          {int a[10] = { 117, 30, 5,
15, 21, 35, 110, 18, 150,           15, 21, 35, 110,18,150, 6};
6};                                 int i, n = 0, k = 0, m = 0;
int i, n = 0, k = 0, m = 0;         for (i = 0; i <= 9; i++)
for (i = 0; i <= 9; i++)            {m = a[i] % 5; n = a[i]
{m = a[i] % 5;               ⇨      if ((m ==
n = a[i] % 10;                      (493017974%563))&& (n
if ((m == 0) && (n != 0))           != (10943411116%1276)))
k = k + 1;}                         k = k + 1;}
cout<<"Amount of                    cout<<"Amount of
numbers, where the last             numbers, where the last
number is 5 = "<< k;                number is 5 = "<<k;
return 0;}                          return 0;}
```

Рис. 9. Приклад роботи методу габаритного оголошення змінних

2.4. *Перетворення статичних даних в процедури* [2-5]. При створенні будь-якої програми, програмісти працюють з певним потоком даних, які найчастіше представлені в коді у вигляді статичних даних таких, як рядки, що дозволяють візуально орієнтуватись зловмиснику в роботі програми та у функціях, які виконуються. Дані значення бажано заміняти на ASCII код, або генерувати необхідний рядок відповідно до аргументів. Після цього рядок видаляється, і на його місце записується виклик функції для іншого значенням аргументів. Також до статичних даних належать числові константи, які можуть бути також трансформовані (див. рис. 10).

```
int main()                          int main()
{int b=23;                          {int b=23, c=2; float a;
float a;                            const double pi =
const double pi =            ⇨      3.1415926535897932385;
3.1415926535897932385;              a = 2*c+b*pi-(b-3)/5;
a = b*pi; cout<<a;                  cout<<a;
return 0;}                          return 0;}
```

Рис. 10. Приклад роботи методу перетворення статичних даних в процедури

Пунктуаційні перетворення (Punctuation transformation)

3.Пунктуаційні перетворення – це метод заплутування коду, коли навмисно порушують загальні правила оформлення програми. До даного виду можна віднести:

3.1. *Інверсія кодових елементів* [3-5, 7-9] (зміна порядку виклику змінних, функцій, масивів, класів, методів, процедур). Значення та тип змінних, масивів не визначається безпосередньо перед її використанням, вони оголошуються у коді хаотично. Необхідно також переприсвоювати значення змінних, подібних за назвою. Якщо в програмі виконуються певні математичні перетворення, то доцільно розбивати обрахунок даних операцій на частини. Цей метод дозволить зменшити можливість аналізу коду програми за рахунок нестандартного об'явлення змінних. На рис. 11 наведений приклад обрахунку коренів квадратного рівняння.

```
int main()                          int main()
{double a, b, c, x;                 {double a, b, c;
cout<<"a: ";                        cout<<"a: "; cin>>a;
cin>>a;                             double x;
cout<<"b: ";                        x = 4 * a;
cin>>b;                             double y;
cout<<"c: ";                        cout<<"b: "; double xx;
cin>>c;                             cin>>b; y = b*b;
if ((b*b - 4 * a*c) >= 0) {  ⇨      double l; cout<<"c: ";
x = (-1 * b + sqrt(b*b - 4          l = -1 * b; cin>>c;
* a*c)) / (2 * a);                  xx = 4 * a*c;
cout<<"1="<< x <<"\n";              if ((y - x*c) >= 0)
x = (-1 * b - sqrt(b*b - 4 *        {x = (l + sqrt(y - xx)) / (2
a*c)) / (2 * a);                    * a);
cout<<"2="<<x<<"\n"}                cout<<"1="<<x <<"\n";
else{cout<<"No roots of             x = (l - sqrt(y - xx)) / (2 *
the equation\n";}                   a);
return 0; }                         cout<<"2="<<x<<"\n";}
                                    else{cout<<"No roots of
                                    the equation\n";}
                                    return 0;}
```

Рис. 11. Приклад роботи методу інверсії кодових елементів

3.2. *Видалення токенів* [4, 5, 8, 10] (видалення коментарів, пробілів, символів горизонтальної та вертикальної табуляції, символів переносу рядка, символів переносу сторінки). Головне завдання цього методу – зробити код програми нечитабельним. Це зроблено для того, щоб витрачалося більше часу для безпосереднього аналізу тексту, перетворення коду на загальноприйнятий. Приклад наведений на рис. 12.

```
int main()                          int main()
//variable declaration              {int
{int x[5][5], i = 0, j = 0, b       x[5][5],i=0,j=0,b=0;srand(u
= 0; //function rand()              nsigned(time(NULL)));for(
srand(unsigned(time(N               i=0;i<5;i++){for(j=0;j<5;j++
ULL)));                      ⇨      ){x[i][j]=rand()%10;cout<<
//randomly fill the                 x[i][j]<<"";}
array                               cout<<"\n";}cout<<"\n";if(
for (i = 0; i<5; i++)               x[0][0]>x[4][4])cout<<"\n"
{for (j = 0; j<5; j++)              ; else cout<<"2 \n";return
{x[i][j] = rand() % 10;             0;}
cout<<x[i][j]<<" ";}
cout<<"\n";}
cout<<"\n";
//conditional test
if (x[0][0] > x[4][4])
cout<<"the upper left
corner is more than an
in the lower right \n";
else cout<<"in the right
nizhnm corner is more
than an in the lower
right \n";
return 0;
```

Рис. 12. Приклад роботи методу видалення токенів

Висновки

У даній статі наведено аналіз існуючих класифікацій обфускаційних методів захисту програмного коду та розроблено узагальнену класифікацію методів захисту. Використовуючи розроблену систематизацію, можна побудувати алгоритм (обфускатор), який буде протидіяти реверс-інжинірингу, проводити відповідні заміни елементів кодів так, щоб вони зберігали свою функціональність та ставали складнішими для розуміння, аналізу та модифікації.

Далі необхідно розробити такий обруска-ційний алгоритм, який надійним чином буде захищати програмний код від крадіжки, несанкціонованої модифікації та буде вико-ристовувати сучасні методи обфускаційного захисту програмного коду, що представлено у запропонованій класифікації.

Література

[1] A generic approach to automatic deobfuscation of executable code / B. Yadegari, B. Johannesmeyer, B. Whitely, S. Debray. – IEEE Symposium Security and Privacy (S&P). – 2014. – 18 p.

[2] Balakrishnan A. Code Obfuscation Literature Survey / A. Balakrishnan, C. Schulze. – Computer Sciences Department, University of Wisconsin, Madison. – 2005. – 10 p.

[3] Buzatu F. Methods for obfuscating Java programs / F. Buzatu // Journal of Mobile, Embedded and Distributed Systems. – 2012. – vol. 4. – P. 25–30.

[4] Collberg C. A taxonomy of obfuscating transformations / C. Collberg, C. Thomborson, D. Low. – Department of Computer Science, The University of Auckland, New Zealand, 1997. – 36 p.

[5] Effects of code obfuscation on android app similarity analysis / [J. Park, H. Kim, Y. Jeong and etc] // Journal of Wireless Mobile Networks, Ubiquitous Computing and Dependable Applications(JoWUA). – 2015. – Vol. 6. – P. 86–98.

[6] Garg V., Srivastava A., Mishra A. Obscuring Mobile Agents by Source Code Obfuscation / V. Garg, A. Srivastava, A. Mishra. International Journal of Computer Applications. – 2013. – 61(9). – P. 46-50.

[7] Ilsun Y. Malware obfuscation techniques: A brief survey / Y. Ilsun, Y. Kangbin. – Broadband, Wireless Computing, Communication and Applications (BWCCA), 2010. – 4 p.

[8] Leskov D. Protect your Java code - through obfuscators and beyond [Online] : http://www.excelsior-usa.com/articles/java-obfuscators.html (Accessed on December 10, 2015).

[9] Lu G. Automatic Simplification of Obfuscated JavaScript Code: A Semantics-Based Approach / G. Lu, S. Debray. – Proc. ICISTM-12 Workshop on Program Protection and Reverse Engineering (PPREW), 2012. – 10 p.

[10] Wroblewski G. General Method of Program Code Obfuscation (draft) / G. Wroblewski. – Ph.D. dissertation, Institute of Engineering Cybernetics, Wroclaw University of Technology, 2002. – 120 p.

УДК 004.056 (045)

Степаненко И.В., Кинзерявый В.Н, Наджи А.А., Лозинский И.Л. Современные обфускационные методы защиты программного кода

Аннотация. *В данной статье проанализировано существующие классификации обфускационных методов защиты программного кода. Основываясь на проведенном анализе данных классификаций, было установлено, что защита программного обеспечения имеет определенные недостатки: не были учтены некоторые из методов запутывания программного кода, которые могут значительно повысить стойкость программы к процессу деобфускации. Поэтому в статье представлены современные обфускационные методы защиты программного кода и разработана обобщенная классификация данных методов. В дальнейшем, на основе разработанной классификации, планируется создать программное обеспечение в соответствии с представленным алгоритмом, что позволит запутывать код, усложнит процесс анализа программы и обеспечит защиту программного кода от несанкционированного доступа. Полученные результаты расширят знания об использовании обфускационных методов защиты, обеспечат создание современных и эффективных систем защиты программного кода.*

Ключевые слова: *защита программного кода, обфускация, классификация обфускационных методов, обфускационный алгоритм.*

Stepanenko I., Kinzeryavyy V., Nagi A., Lozinskyi I. Modern obfuscation methods for secure coding

Abstract. *In this paper was done analysis of existing classifications for obfuscation software security methods. Based on the above analysis of these classification it was found that software protection has certain drawbacks such as are not taken into account some obfuscation methods that can strongly increase stability of the code. In this work presented modern obfuscation software security methods and developed generalized classification of these methods. Subsequently, based on developed classification, it is planned to create software based on presented algorithm, that will allow to embrangle code, made the process of software analysis more complicated and will provide software security from unauthorized disclosure. The obtained results expand the knowledge on how to use obfuscation security methods and will provide the development of modern and effective software security systems.*

Key words: *secure coding, obfuscation, obfuscation methods classification, obfuscation algorithm.*

Отримано *17 лютого 2016 року*, затверджено редколегією *10 березня 2016 року*