УДК 004.056.5

Программная реализация алгоритмов обфускации программного кода языка JavaScript

С.В. Медгаус, А.В. Чернышова
Донецкий национальный технический университет
medgaus-sergey@yandex.ru, alla@donntu.org

Медгаус С.В., Чернышова А.В. Программная реализация алгоритмов обфускации программного кода языка JavaScript. *В тексте данной статьи представлен краткий обзор существующих обфускаторов для языка JavaScript. После их анализа предложено реализовать собственный обфускатор, применяющий сложные преобразования исходного кода. В статье подробно описаны применяющиеся алгоритмы запутывания программного кода языка JavaScript.*

Введение

На данный момент проблема сохранения прав собственности на исходный код достаточно актуальна. В отличие от настольных приложений, в которых существуют различные средства защиты и лицензирования, в web-приложениях такие способы защиты не применимы и весь исходный код скриптов доступен для просмотра любому желающему через браузер.

Подробный анализ существующих способов защиты web-приложений рассмотрен в [1]. Исходя из изложенного в статье материала, можно сказать, что обфускация – это наименее ресурсоёмкий способ защиты исходного кода web-приложений. «Обфускация (от лат. obfuscare — затенять, затемнять; и англ. obfuscate — делать неочевидным, запутанным, сбивать с толку) или запутывание кода — приведение исходного текста или исполняемого кода программы к виду, сохраняющему его функциональность, но затрудняющему анализ и понимание алгоритмов работы. Существуют специальные программы, производящие обфускацию, так называемые обфускаторы» [2].

На текущий момент существуют различные программные продукты, которые предлагают обфускацию исходного кода, однако эффективные алгоритмы обфускации предлагают только платные продукты, бесплатные же работают на уровне минификации кода (уменьшение размера исходного кода, путём сокращения имён переменных и функций, удаление символов форматирования кода). В работе [1] были рассмотрены и проанализированы такие полнофункциональные и эффективные программные продукты как YUI Compressor [3], Packer [4], JavaScript Obfuscator [5] и Google Closure Compiler [6]. После анализа существующих обфускаторов было принято решение разработать комбинацию эффективных обфусцирующих алгоритмов для защиты программного кода языка JavaScript .

Целью работы являются разработка и программная реализация алгоритмов обфускации для языка JavaScript с последующим объединением их в комплексный алгоритм обфусцирующего преобразования и использование его при создании обфускатора.

1 Проектирование программного продукта

После анализа существующих обфускаторов для языка JavaScript, была разработана диаграмма прецедентов для проектируемого обфускатора (см. рис. 1).

Диаграмма прецедентов UML – это диаграмма, показывающая связи между актёрами и действиями (прецедентами), которые они могут выполнять.[7].

Как видно из рис. 1, пользователь может загружать файл с исходным кодом, сохранять обфусцированный код в файл, запускать обфускацию, а также выбирать режимы обфускации:

- удаление форматирования;
- преобразование условных конструкций;
- логическое преобразование;
- сокращение констант;
- кодирование числовых констант;
- кодирование строковых констант;
- переименование переменных.

Все вышеперечисленные режимы обфускации применяются в заданном программой порядке, который невозможно изменить, то есть являются этапами. Это позволяет максимально эффективно запутать исходный код, так как появляются связи ещё и между самими режимами обфускации.

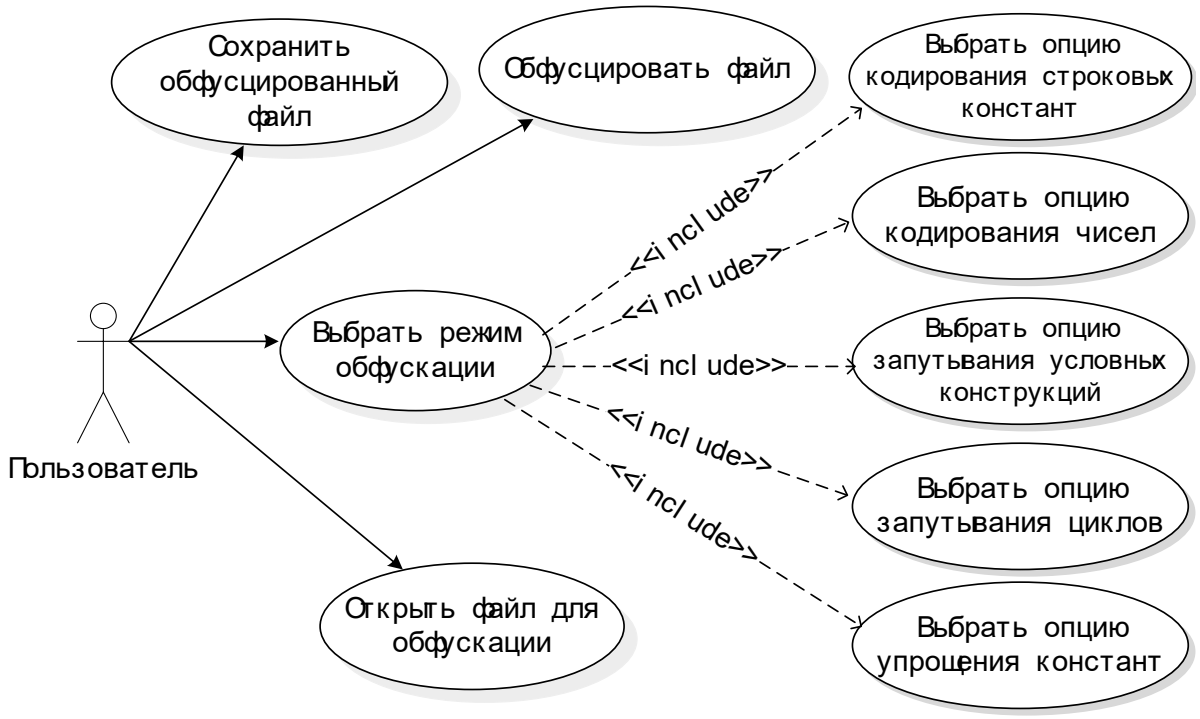

Рисунок 1 – Диаграмма вариантов использования обфускатора

Исходя из спроектированной диаграммы вариантов использования, была разработана диаграмма потоков данных обфускатора, которая представлена на рис. 2.

Приложение будет состоять из одной формы, в которой будет содержаться главный рабочий класс Обфускатор, который будет выполнять всю работу. В свою очередь, в зависимости от выбранных режимов, обфускатор будет использовать различные запутыватели, которые предоставляет программа. После обфускации программа будет выдавать выходной файл, который будет содержать обфусцированный JavaScript код.

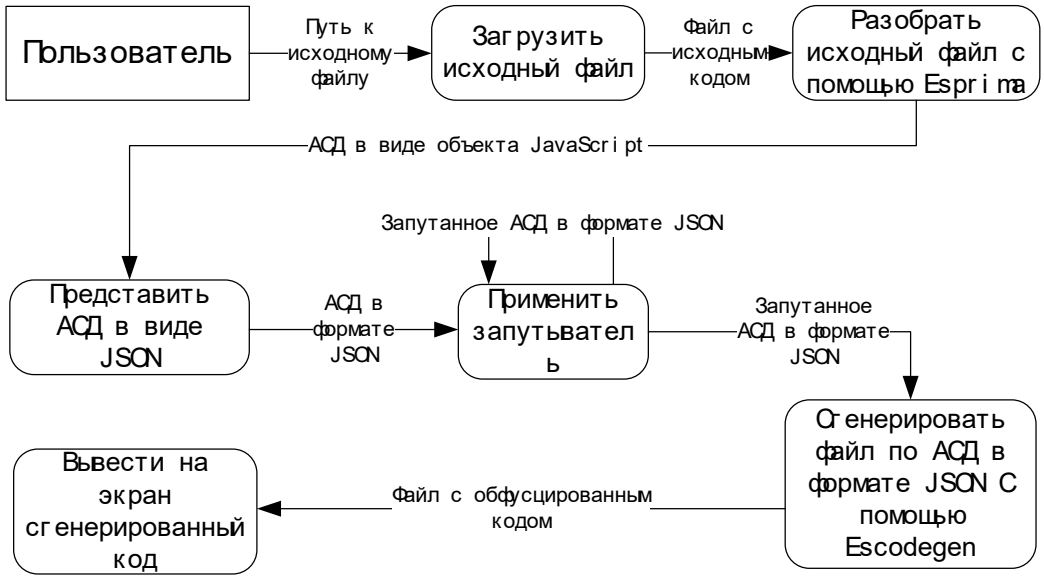

Рисунок 2 – Диаграмма потоков данных работы обфусцирующей программы

Все остальные диаграммы, разработанные при проектировании обфускатора, представлены в [8].

2 Алгоритмы запутывающих преобразований

Эффективность программы-обфускатора зависит от её алгоритмов, которые преобразовывают (запутывают) исходный код программ [9].

2.1 Обобщённый алгоритм работы обфускатора

Необходимо отметить, что основным объединяющим алгоритмом программы является алгоритм, в котором объединяются все запутывающие алгоритмы и используются для приведения исходного кода в непонятный для человека вид (см. рис. 3).

```
подсчитать количество объектов-запутывателей;
подключить движок Nashorn;
распознать код с помощью Esprima;
перевести из формата Object в JSON с
    помощью встроенной библиотеки JSON;
распознать с помощью JSONSimple дерево в формате JSON;

FOR (для каждого запутывателя) {
    получить текстовое представление этапа;
    установить прогресс в зависимости
        от номера запутывателя;
    оповестить текстовым представлением;
    запутать код;
}

установить прогресс на последний этап;
опубликовать этап "Построение кода";

вызвать метод "Запутать" у объекта
    класса ПеремешивательФункций;
представить АСД в формате Object;
задать опции генерации кода, учитывая форматирование;
сгенерировать код с помощью библиотеки Escodegen;
оповестить об окончании обфускации;

вернуть обфусцированный код;
```

Рисунок 3 – Обобщённый алгоритм работы программы

2.2 Алгоритм преобразования выражений условного оператора if-else

Суть этого преобразования заключается в том, чтобы преобразовать допустимые выражения оператора if-else в тернарный оператор. Этот алгоритм нацелен на ухудшение читаемости кода, без каких-либо фундаментальных преобразований. Алгоритм преобразования на псевдоязыке представлен на рис. 4.

2.3 Алгоритм логического преобразования

Данный алгоритм представляет собой простое инвертирование сравнений таким образом, чтобы вместо исходного оператора использовалось отрицание противоположного оператора с отрицаемыми операндами (см. рис. 5) [6]. На схеме алгоритма указана только часть логических операторов. Операторы, не попавшие на схему, преобразуются по такому же алгоритму.

Необходимо отметить, что можно использовать и другие логические преобразования, применяя логические эквиваленции, однако в JavaScript результатом логического выражения может быть не только true или false, но и ещё и объект, поэтому использование более сложных формул преобразований может приводить к ошибкам.

```
выполнить проход по АСД (
    IF (есть условные конструкции с пустой веткой else) THEN
        устанавливаем флаг необходимости доп. функций;
)

IF (флаг необходимости доп. функций) {
    добавляем в массив функций случайное кол-во
        случайно сгенерированных функций;
    FOR (каждой функции) {
        сгенерировать случайное арифметическое выражение
            из нескольких арифметических операндов;
        представить выражение в виде строки;
        внести в тело функции в виде получения кода из
            символьных кодов;
        занести тело функций в конец исходного кода программы;
    }
}

выполнить проход по АСД (
    IF (текущий узел = "условная конструкция" И
            (выражение не содержит вызовов методов объектов
                ИЛИ системных вызовов)) THEN {
        IF (ветка else пустая) {
            занести туда вызов сгенерированной функции;
        }
        поменять местами ветки if  и else;
        поменять тип узла на "тернарный оператор";
    }
)
```

Рисунок 4 – Алгоритм преобразования выражений условного оператора if-else

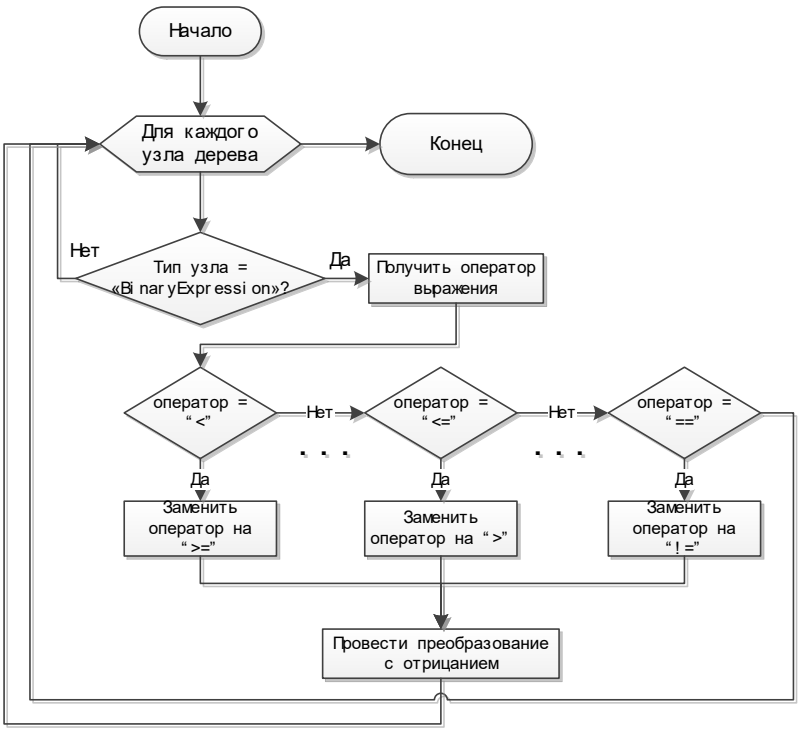

Рисунок 5 – Алгоритм преобразования логических выражений

2.4 Алгоритм сокращения констант

По стандарту ECMA Script 5.0 в языке JavaScript ещё нет констант, которые бы обеспечивал сам язык, поэтому программа сама анализирует переменные, которые не изменяют своего значения в исходном коде пользовательской программы.

Данное преобразование состоит из следующих этапов:

– найти возможные константы в коде;
– исключить меняющиеся переменные;
– исключить переменные, чьё значение не является простым, то есть не является числом, строкой или булевым значением;
– заменить в местах использования констант их значением.

Детальное описание алгоритма представлено на рис. 6.

```
FOR (для каждого узла дерева) {
    IF (узел - это объявление переменной) THEN {
        получить имя переменной;
        поместить в хеш-таблицу имя переменной и её значение;
    }
}
FOR (для каждого узла дерева) {
    IF (узел используется с оператором
            инкремента или декремента) THEN {
        удалить переменную из хеш-таблицы;
    }
    IF (узел - присвоение переменной значения) THEN {
        IF (переменная неинициализированная) THEN {
            присвоить новое значение и занести в хеш-таблицу;
        } ELSE {
            удалить переменную из хеш-таблицы;
        }
    }
}
FOR (для каждой переменной в хеш-таблице) {
    IF (инициализированное значение не простое) THEN {
        удалить переменную из хеш-таблицы;
    }
}
FOR (для каждого узла дерева) {
    IF (используется переменная, которая есть
            в хеш-таблице) THEN {
        заменить использование перменной её значением;
    }
}
FOR (для каждого узла дерева) {
    IF (объявление переменной, которая есть в хеш-таблице) THEN {
        удалить объявление переменной;
    }
}
```

Рисунок 6 – Алгоритм сокращения констант

2.5 Алгоритм кодирования чисел

Данный алгоритм очень важен для программной системы, так как большая часть исходного кода в любом языке программирования использует огромное количество чисел для счётчиков различного рода, для циклов и для числовых констант.

Суть этого алгоритма заключается в том, чтобы заменить числа некоторым набором арифметических выражений, и после этого представить все числа в 16-ичной системе счисления [6]. Детальный алгоритм представлен на рис. 7.

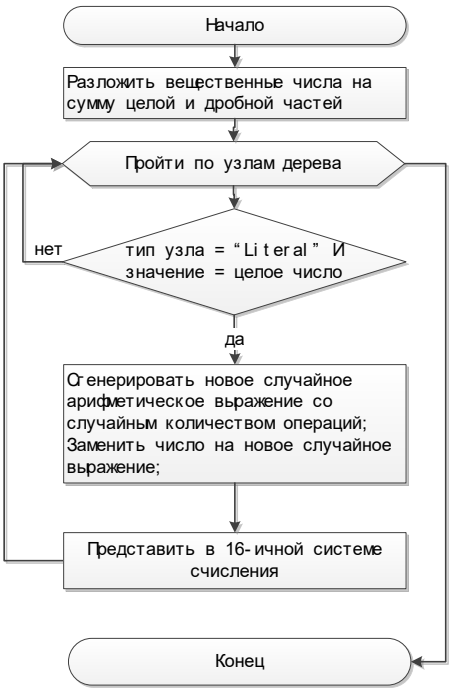

Рисунок 7 – Алгоритм кодирования чисел

2.6 Алгоритм кодирования строк

Данный алгоритм также является важным в работе программной системы. Почти во всех программах используются какие-либо строковые константы, будь-то сообщения об ошибках или текстовые элементы интерфейса. Для того, чтобы труднее было разобрать исходный код программы, необходимо преобразовать и скрыть явное использование таких строковых констант.

В данном преобразовании строки представляются в виде конкатенации вызовов различных функций, а также в виде кодировки BASE64. Всё разбиение строковых констант происходит каждый раз случайно, обеспечивая различный исходный код на выходе обфускатора [7].

Детальный алгоритм представлен на рис. 8.

2.7 Алгоритм переименования переменных

Данное преобразование исходного кода является самым не затратным с точки зрения ресурсов преобразования, но с точки зрения конечного запутывания является одним из самых эффективных алгоритмов обфускации.

Главная особенность этого алгоритма в том, чтобы представить имена переменных и функций в виде непонятного для человека набора символов [6]. Подробный разбор алгоритма представлен на рис. 9.

```
FOR (для каждого узла дерева) {
    случайно получить флаг "необходимо ли делить слово";
    IF (узел - это строковый литерал И
            есть флаг "делить слово") THEN {
        получить длину строки;
        получить случайное число до длины строки;
        разбить слово на два по случайному числу;
    }
}
FOR (для каждого узла дерева) {
    IF (узел - это строковый литерал) THEN {
        занести каждую строковую константу в массив констант;
    }
}
FOR (для каждого узла дерева) {
    случайно получить флаг "необходимо
        ли выносить слово в отдельную функцию";
    IF (узел - это строковый литерал И
            есть флаг "выносить слово") THEN {
        заменить использование строковой константы на
            вызов отдельной функции;
        установить флаг "отдельная функция использовалась";
    }
}
IF ("отдельная функция использовалась") {
    занести в конец исходного кода тело отдельной функции
        для строковых констант;
}
FOR (для каждого узла дерева) {
    случайно получить флаг "необходимо ли преобразовать в Base64";
    IF (узел - это строковый литерал И
            есть флаг "необходимо преобразовать") THEN {
        заменить использование строковой константы на
            закодированное значение в Base64 и вызов функции
            декодирования;
    }
}
```

Рисунок 8 – Алгоритм кодирования строковых констант

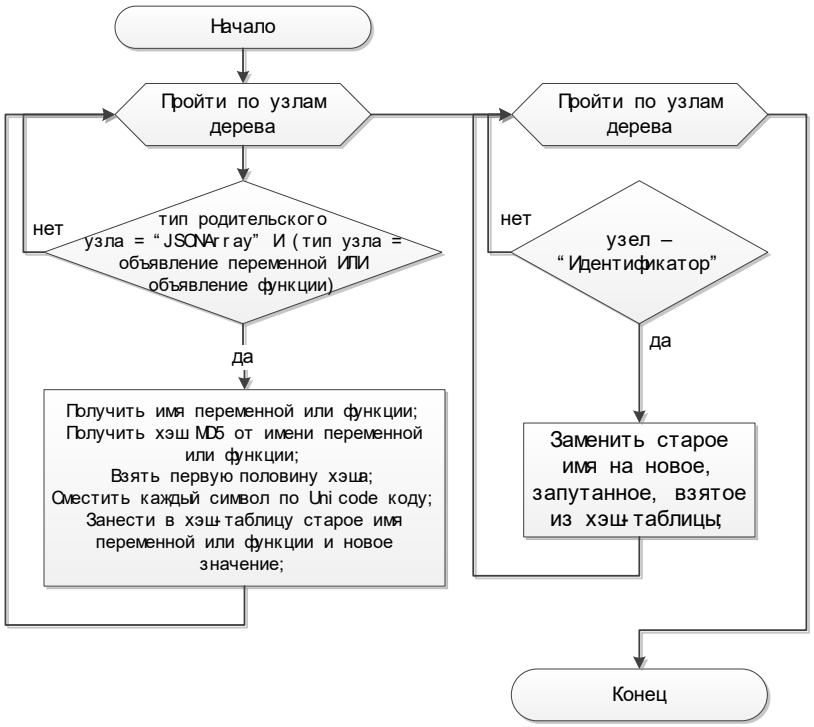

Рисунок 9 – Алгоритм переименования переменных

2.8 Алгоритм переставления функций

Данный алгоритм не является напрямую запутывающим преобразованием, его нельзя выбрать через интерфейс пользователя. Он служит для того, чтобы перемешивать глобальные функции, которые могли появиться вследствие других запутывающих преобразований (мусорные функции). Иначе, если это преобразование не использовать, то мусорные функции располагались бы в конце документа с исходным кодом и могли бы легко быть удалены человеком, который взялся бы деобфусцировать исходный код. Детальное описание алгоритма представлено на рис. 10.

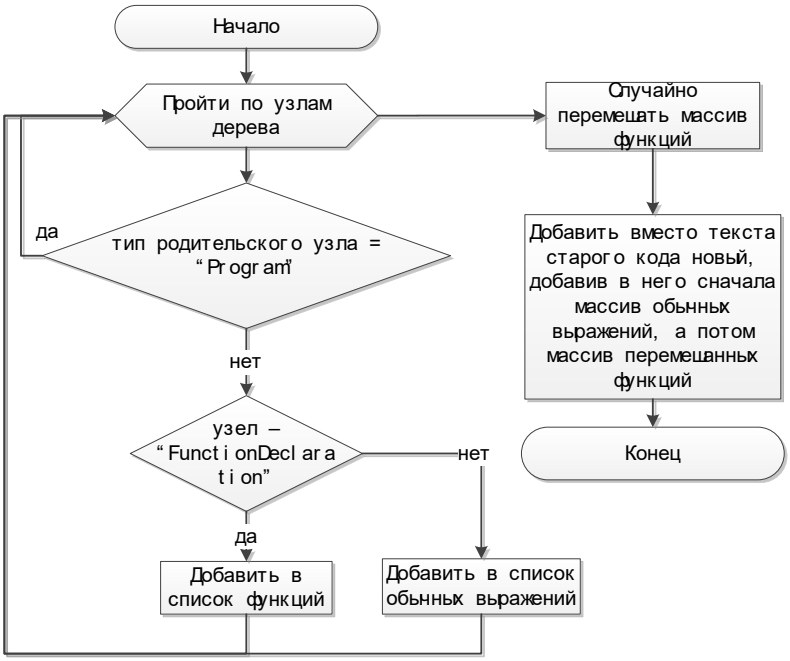

Рисунок 10 – Алгоритм переставления функций

3 Результат разработки алгоритмов и их реализации в программном продукте

После разработки алгоритмов, они были реализованы на языке Java, где и были объединены в один главный обфусцирующий алгоритм.

На рис. 11 представлена экранная форма обфускатора, на которой видно, что пользователь выбрал все доступные режимы обфускации. Также видно, что исходный код существенно изменился.

Как показало тестирование на экспертных группах, код стал более запутанным. Эксперты сделали субъективное заключение – сложность понимания исходного кода увеличилась в 3 раза.

Выводы

В ходе выполнения данной работы были проанализированы существующие обфускаторы языка JavaScript. После их детального изучения, были выявлены присущие им недостатки, учитывая которые, было решено разработать и реализовать собственные эффективные алгоритмы обфускации программного кода языка JavaScript.

Также была разработана последовательность объединения запутывающих преобразований (общий алгоритм работы обфускатора) с максимальной степенью запутанности.

Дальнейшие разработка и реализация алгоритмов обфускации являются перспективными, так как разработанные алгоритмы работают напрямую с кодом, меняя внешнее представление кода, в то время как, преобразования потока управления программ не производится. То есть разработанные алгоритмы принадлежат к категориям лёгкой и средней степеням обфускации, а алгоритмы обфускации сложной степени необходимо разрабатывать.

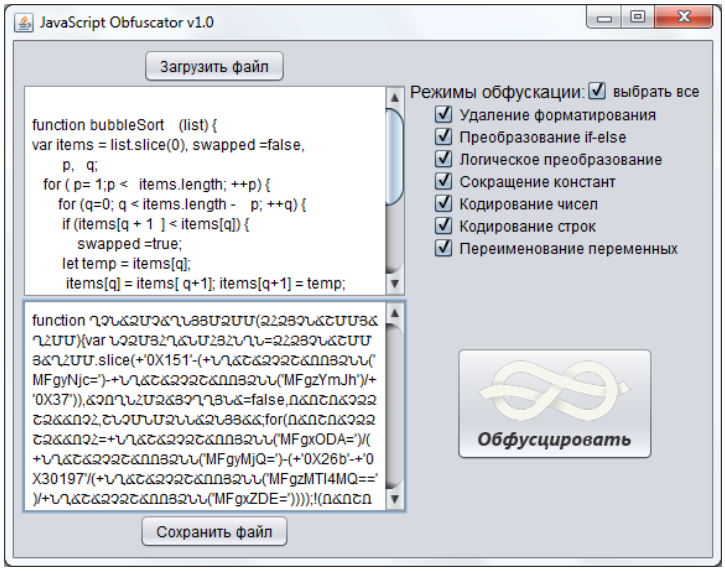

Рисунок 11 – Экранная форма обфускатора

Литература

1. Обзор существующих обфускаторов и их алгоритмов. Компьютерная и программная инженерия - 2015 год: - Донецк: ДонНТУ, 2015. – С.117-119.

2. Википедия. Обфускация [электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Обфускация.

3. YUI Compressor [электронный ресурс]. – Режим доступа: https://github.com/yui/yuicompressor.

4. Packer [электронный ресурс]. – Режим доступа: http://dean.edwards.name/packer.

5. JavaScriptObfuscator. Canada [электронный ресурс]. – Режим доступа: https://www.javascriptobfuscator.com/.

6. Google Closure Compiler [электронный ресурс]. – Режим доступа: https://developers.google.com/closure/compiler/.

7. Диаграммы прецедентов: крупным планом [электронный ресурс]. – Режим доступа: http://www.intuit.ru/studies/courses/1007/229/lecture/5962.

8. Проектирование обфускатора для языка JavaScript. ИУСМКМ - 2016: VII Международная научно-техническая конференция, 26 мая 2016: - Донецк: ДонНТУ, 2016. – С. 167-173.

9. LynX. Обфускация и защита программных продуктов // CITForum. [электронный ресурс]. – Режим доступа: http://citforum.ru/security/articles/obfus.

10. Java Obfuscator – String Encryption // zelix.com [электронный ресурс]. – Режим доступа: http://www.zelix.com/klassmaster/featuresStringEncryption.html.

Медгаус С.В., Чернышова А.В. Программная реализация алгоритмов обфускации программного кода языка JavaScript. *В тексте данной статьи представлен краткий обзор существующих обфускаторов для языка JavaScript. После их анализа, предложено реализовать собственный обфускатор, применяющий сложные преобразования исходного кода. В статье подробно описаны применяющиеся алгоритмы запутывания.*

Ключевые слова: *обфускатор, Java, JavaScript, запутывание, обфускация, алгоритмы обфускации, Esprima, Nashorn*

Medgaus Sergey, Chernyshova Alla Program realization obfuscating algorithms of JavaScript source code. *There is presented short overview of existing obfuscators for JavaScript language in the text of this article. It is proposed to realize own obfuscator that uses complex transformation of source code. There are described different applied mangling algorithms.*

Keywords: *obfuscator, Java, JavaScript, mangling, obfuscation, obfuscating algorithms, Esprima, Nashorn*

Статья поступила в редакцию 20.11.2016
Рекомендована к публикации д-ром физ.-мат.. наук А.С. Миненко