МИНОБРНАУКИ РОССИИ
федеральное государственное автономное образовательное учреждение высшего образования «Самарский национальный исследовательский университет имени академика С.П. Королева»
(Самарский университет)

Институт информатики, математики и электроники
Факультет механико-математический
Кафедра безопасности информационных систем

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

МАТЕМАТИЧЕСКИЕ МОДЕЛИ ОБФУСКАЦИИ И ИХ УСТОЙЧИВОСТЬ

по направлению подготовки 10.03.01 Информационная безопасность
(уровень бакалавриата)
направленность (профиль) №2 «Организация и технология защиты информации»

Обучающийся _____ О. М. Тютюгина

Руководитель ВКР,

д.ф.-м.н., профессор _____ С. Я. Новиков

Нормоконтролер _____/_____/

Самара 2021

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

По статистике в России количество использования лицензионного программного обеспечения с нарушением интеллектуальной собственности доходит до 95%, в Соединенных Штатах Америки этот показатель меньше и держится на уровне 35%. В последние годы так называемое «пиратство» приобретает массовый характер, многие программные обеспечения взламываются и используются незаконно и без лицензии.

Актуальность темы определяется тем, что большое количество интеллектуальной собственности, в частности программные средства, используется незаконно из-за недостаточной стойкости и явности исходного кода, что позволяет злоумышленникам спокойно взламывать и пользоваться этими программами. С каждым годов количество преступлений по части 2 статьи 146 УК РФ «Нарушение авторских и смежных прав» увеличивается. Если в 2017 году количество осужденных равнялось – 90 человек, то уже в 2020 году – 323.

К средствам защиты программных средств относятся криптографические средства защиты (шифрование каналов передачи информации, шифрование и обфусцирование исходного кода), стеганографические средства защиты (цифровой водяной знак, электронная подпись).

Целью работы является анализ математических моделей обфсукации и программ-обфускаторов, построенных на этих моделях.

Для достижения поставленной цели необходимо решить следующие задачи:

– проанализировать математические методы обфускации;

– провести анализ стойкости программ-обфускаторов;

– разработать программу-обфускатор на основе анализа уже существующих;

– провести сравнительный анализ программ-обфускаторов.

1 Обфускация

Обфускация в дословном переводе с латинского переводится как затенять, затемнять. Само слово «обфускация» - от английского *obfuscate*, что значит делать неочевидным, сбивать с толку. Сейчас обфускация или запутывание кода – это приведение исходного текста или исполняемого кода программы к виду, сохраняющему её функциональность, но затрудняющему анализ, понимание алгоритмов работы и модификацию при декомпиляции [1].

Основной целью обфускации программы является её затруднение декомпиляции, отладки и изучения для предотвращения обнаружения её функциональности. Это может применяться и для защиты интеллектуальной собственности программы, и для скрытия вредоносных функций программы.

Также при помощи обфускации программного кода создатель может оставлять скрытые данные, такие как: дата, время, авторские права и т.д.

Задачи запутывания и анализа обфусцированных программ имеют три аспекта [1]:

1) **теоретический.** Он включает разработку новых алгоритмов преобразования графа потока управления или трансформации данных программы, а также теоретическую оценку сложности их анализа и раскрытия;

2) **прикладной.** Включает в себя разработку конкретных методов запутывания, а именно наилучших комбинаций алгоритмов, эмпирический сравнительный анализ различных методов, эмпирический анализ устойчивости методов, и т.д.;

3) **психологический.** Является не до конца сформированным, но важным при обфускации программ. При обратной инженерии происходит исследование некоторого готового устройства или программы, а также документации на него с целью понять принцип его работы [2]. Методы запутывания должны использовать слабости человеческой психики для получения более эффективного результата.

1.1 Запутывающие преобразования

Для того, чтобы запутать программный код, применяют различные виды воздействия на него. В зависимости от того, на какой компонент программы нацелено воздействие, запутывающие преобразования классифицируют на:

– преобразования на уровне исходного кода. Такой вид преобразования изменяют только внешний вид программы. Преобразование может заключаться в удалении комментариев, отступов в тексте программы и переименовании идентификаторов;

– преобразования структур данных, изменяющие структуры данных, с которыми работает программа;

– преобразования потока управления программы, которые изменяют структуру её потока управления.

1.1.1 Преобразования форматирования

Как было сказано ранее, к преобразованиям форматирования относятся удаление комментариев, переформатирование программы, удаление отладочной информации, а также изменение имён идентификаторов.

Такие способы как удаление комментариев и переформатирование используются только при запутывании на уровне исходного кода программы. Хотя комментарии не являются обязательной частью программы, но их отсутствие значительно затрудняет обратную инженерию программы. При переформатировании программы исходное форматирование теряется безвозвратно. Однако, существует средства автоматического форматирования.

При запутывании на уровне объектной информации применяется удаление отладочной информации. Данное удаление приводит к тому, что имена локальных переменных невосстановимы.

Изменение имен переменных и функций требует семантического анализа и анализа межмодульных связей. Данное преобразование может заменять имена на более короткие или наоборот на длинные и бессмысленные,

в расчете на то, что длинные имена хуже воспринимаются человеком.

1.1.2 Преобразование структур данных

К этой группе относятся, например, преобразование, изменяющее иерархию наследования классов в программе, или преобразование, объединяющее скалярные переменные одного типа в массив. Также, как и случай изменение имен переменных, данное преобразование затрудняет процесс анализа и восприятия кода программы.

1.1.3 Преобразование потока управления

Потоком управления программы является последовательность, в которой команды выполняются динамически во время работы программы

Графом потока управления является множество всевозможных путей исполнения программы, представленном в виде графа.

Преобразования потока управления изменяют граф потока управления одной функции.

Методов, основанных на преобразовании потока управления, огромное количество. К ним также относятся методы создания в программе новых функций.

Для начала следует ввести понятие непрозрачных предикатов. Предикат является непрозрачным тогда, когда его значение известно в момент запутывания программы, но трудноостанавливаемо после того, как запутывание завершено.

Существует три вида непрозрачных предикатов: P^F – предикат, который всегда имеет значение «ложь», P^T – предикат, который всегда имеет значение «истина», $P^?$ – предикат, который может принимать оба значения, и в момент запутывания значение известно. Данное определение поможет нам для дальнейшего рассмотрения методов:

– **встраивание функций.** Достаточно простой метод, который заключается во вставке тела функции в точку её вызова;

– **вынос группы операторов (вынос функций).** Метод, обратный методу встраивания функций. Группа операторов выделяется в отдельную функцию;

– **вставка недостижимого кода.** Когда в программе есть функция, в которой фигурируют непрозрачные предикаты видов P^F и P^T и ветка условий, со значениями, соответствующими «истина» и «ложь», которые никогда не будут выполняться, то такая часть кода является недостижимой. Ветки условий могут быть заполнены произвольными вычислениями, которые могут быть похожи на исполняемый код, но они не будут выполняться;

– **вставка мертвого кода.** Заключается в том, что в код программы вставляется другой код такой, что на его выполнение затрачиваются ресурсы и в отличие от недостижимого кода происходит его выполнение, но результат никак не будет влиять на саму программу;

– **вставка избыточного кода.** В данном случае, код выполняется и его результат используется в дальнейшем, однако данный код можно упростить или вообще удалить, т.к. можно изначально внести в программу, как константное значение;

– **переплетение функции.** Две или более функции объединяются в одну, вместе со всеми списками параметров, в который добавляется ещё один параметр, показывающий, какая из функций выполняется;

– **клонирование функций.** При разборе кода программы можно определить, как эта функция используется, в каких местах и с какими параметрами они вызываются. Если замаскировать вызов данной программы, как вызов другой, то анализ можно затруднить;

– **устранение библиотечных вызовов.** Некоторые программы используют стандартные библиотеки, которые хорошо известны продвинутому пользователю. Использование стандартных библиотек облегчает анализ программы, поэтому, чтобы затруднить извлечение информации применяют данный метод;

– **развертка циклов.** Тело циклов размножается два или более раз до полного разворачивания в зависимости от того известно ли изначально количество повторений цикла;

– **помимо развертки циклов используется их разложение.** Используется, если цикл имеет сложную структуру, которую можно разбить на циклы с простой структурой;

– **преобразования с использованием переменных.** Это могут быть преобразования, связанные с локализацией переменных, что приводит к появление большого числа новых переменных. Также используется расширение области действия переменных, что заметно сокращает количество переменных. Оба методы затрудняют анализ программы.

Наиболее значимыми методами в преобразовании графов потока управления являются преобразование сводимого графа потока управления к несводимому и реструктуризация графа.

Сводимым графом является граф, который можно получить из графа с единственной вершиной путём операций дополнения и объединения графов. Если же такое преобразование невозможно, то граф считается несводимым [3]. Иногда, при написании кода можно использовать преобразования, которые противоречат структуре языка. К примеру, трансформировать структурный цикл в цикл с множественными заголовками с использованием непрозрачных предикатов.

Реструктуризация графа применяется тогда, когда на семантическом уровне можно определить, какие применялись преобразования. В качестве метода реструктуризации можно использовать приведение графа к однородному виду, который затрудняет анализ программы.

Как видно, все методы направлены на усложнение исследования кода, однако в интерпретируемых языках программирования обфусцировнный код занимает меньше, чем исходный и иногда выполняется быстрее.

1.2 Математические модели обфускации

Обфускация является одним из современных направлений развития криптографии, поэтому многие модели и математические описания тесно связаны друг с другом.

Математическое описание модели обфускации формулируется следующим образом [4]. За обфсукатор понимается вероятностный алгоритм O, который получает на вход программу P, и преобразует её в программу O(P), так что удовлетворяются следующие требования:

1) функциональность. Программы P и O(P) вычисляют одну и ту же функцию, т.е. эквивалентны;

2) эффективность. Расходы на переход от P к O(P) незначительны, т.е. увеличение длины программы и времени её выполнения невелики;

3) стойкость. Полученная программа O(P) трудна для понимания.

Классификацию обфускации программ проводят относительно возможных моделей противника с учетом различных форм представления программ.

Модель поведения противника определяют следующие параметры:

1) цель противника. Ею может быть получение информации об устройстве программы, вычисляемой ею функции, содержащихся алгоритмов и структур данных;

2) предварительные знания;

3) вычислительные ресурсы: набор инструментальных средств анализа программ;

Этих параметров недостаточно для введения понятий математических моделей обфсукации, поэтому для более точного описания каждого определения моделей необходимо ввести ряд положений:

1) программа представляется в виде машины Тьюринга (*MT*);

2) модель противника представляется в полиномиальной вероятностной машины Тьюринга (*PPT*).

Исходя из этих данных математические модели классифицируются по стойкости:

1) стойкие в модели «черный ящик»;
2) стойкие в модели «серый ящик»;
3) стойкие в модели с дополнительным входом.

1.2.1 Модель «черного ящика»

В данной модели противник не обладает какими-либо знаниями о программе. Он стремится извлечь какую-нибудь информацию об исходной программе.

Рабочая схема поведения противника выглядит следующим образом: противник видит входные данные и выходные данные, сама программа представляется ему неким «черным ящиком» (рис. 1).

Рисунок 1 – Визуальное представление модели «черный ящик»

Злоумышленник не имеет физического доступа к ключу, а может только наблюдать за внешней информацией и поведением.

Вероятностный алгоритм O называется обфускатором программ, стойким в модели «черного ящика», если он удовлетворяет следующим трем требованиям:

1) функциональности;
2) эффективности;

3) стойкости: для любой полиномиальной вероятностной машины Тьюринга A (противник) существует полиномиальная вероятностная машина Тьюринга S (симулятор) и пренебрежимо малая функция ν, удовлетворяющие для любой машины Тьюринга M соотношению:

$$| \Pr[A(O(M)) = 1] - \Pr[S^M (1^{|M|}) = 1] | \leq \nu(|M|) \qquad (1)$$

где, Pr[A(O(M)) =1] – вероятность того, что полиномиальная вероятностная машина Тьюринга противника (A), которой на вход подали слово O(M), выдаст 1;

$\Pr[S^M (1^{|M|}) = 1]$ – вероятность того, что полиномиальная вероятностная машина S при переборе всевозможных вариантов M слов длиной |M|, выдаст 1.

ν(|M|) – значение малой функции ν, которая вычисляется относительно длины слова |M|.

1.2.2 Модель «серого ящика»

Модель «серого ящика» является побочным продуктом реализации «черного ящика». Отличием от предыдущей модели является то, что злоумышленник имеет физический доступ к ключу, какие-то входные данные (рис. 2). Такими данными могут быть: время выполнения алгоритма, информация о потреблении энергии, трасса вычисления исходной программы.

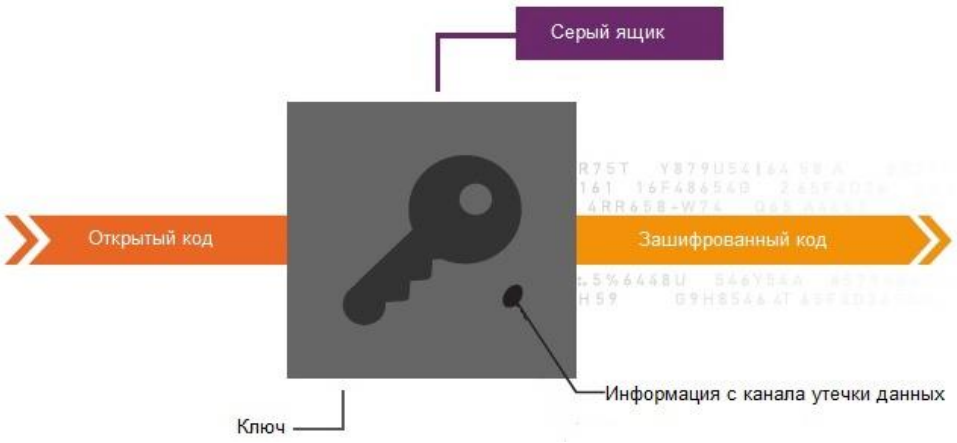

Рисунок 2 – Визуальное представление модели «серый ящик»

Чтобы сформулировать требование стойкости обфускации для данной модели необходимо ввести оракул, который как раз и представляет собой математическую модель «серый ящик».

Вероятностный алгоритм O называется обфускатором программ, стойким в модели «серого ящика», если он удовлетворяет требованиям:

1) функциональности;
2) эффективности;
3) стойкости: для любой полиномиальной вероятностной машины Тьюринга A существует полиномиальная вероятностная машина Тьюринга S и пренебрежимо малая функция ν, удовлетворяющие для любой машины Тьюринга M соотношению (1), однако с некоторыми поправками.

Вторым слагаемым будет $Pr[S^{Tr(M)}(1^{|M|}) = 1]$, где Tr(M) – результат работы оракула Tr(M), которому на вход подаётся машина Тьюринга M.

1.2.3 Модель с дополнительным входом

При приближении к реальности разумно полагать, что злоумышленнику могут быть известны такие дополнительные сведения как: пользовательская инструкция, описание какой-либо функции или вычисляемой программы и т.д. Такие знания усиливают возможности противника, поэтому данная модель (рис. 3) является наиболее сложной для обеспечения стойкости.

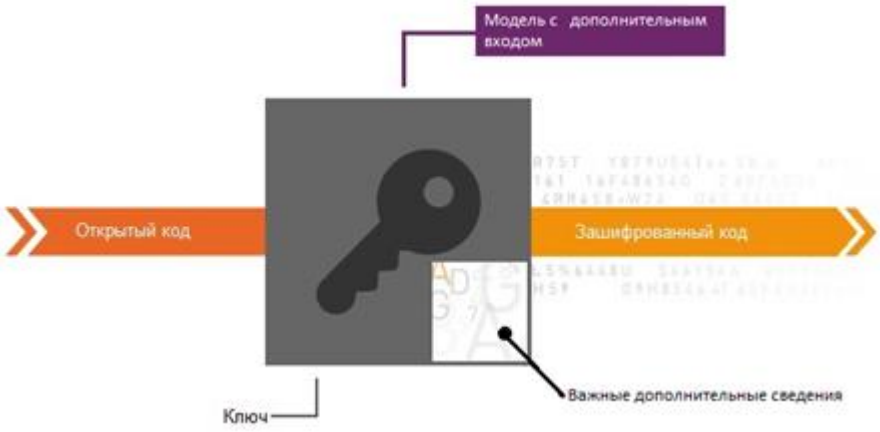

Рисунок 3 – Визуальное представление модели с дополнительным входом

Вероятностный алгоритм O называется обфускатором с дополнительным входом, если он удовлетворяет требованиям:

1) функциональности;

2) эффективности;

3) стойкости: для любой полиномиальной вероятностной машины Тьюринга A существуют такие полиномиально-вероятностные машины Тьюринга S и пренебрежимо малая функция ν, что для любого полинома q(·) и битовой строки z, длина которой не превосходит q(|M|), выполняется соотношение (1). Однако на вход программам A и S^M будет подаваться значения соответственно (O(M), z) и ($1^{|M|}$, z).

1.3 Доказательство стойкости обфускации

Классификация основывается на стойкости обфускаторов. Чтобы отнести существующие программы-обфсукаторы к той или иной модели, необходимо доказать возможности реализации и существования моделей. Для этого следует доказать или опровергнуть возможность реализации данных моделей.

Исследование стойкости обфускаторов началось в работах Б. Барака, который и ввёл понятие обфускации в 2001 году [4].

В своей диссертации Б. Барак доказал теорему, согласно которой, не существует универсальных стойких обфускаторов в модели «черный ящик».

Формулировка теоремы звучит следующим образом «Стойких обфсукаторов в модели черный ящик не существует».

Схема доказательства данной теоремы выглядит следующим образом. Доказательство идет от обратного. Предположим, что существует такой стойкий обфускатор. Тогда для $\alpha, \beta \in \{0,1\}^k$, определим $C_{\alpha,\beta}$, $D_{\alpha,\beta}$ и Z_k следующим образом:

$$C_{\alpha,\beta} = \begin{cases} \beta, \text{если } x = \alpha \\ 0, \text{если } x \neq \alpha \end{cases}$$

$$D_{\alpha,\beta} = \begin{cases} 1, \text{если } C(\alpha) = \beta \\ 0, \text{если } C(\alpha) \neq \beta \end{cases}$$

Комбинируя данные функции можно получить следующие функции $F_{\alpha,\beta} = C_{\alpha,\beta}, \times D_{\alpha,\beta}$, $G_{\alpha,\beta} = Z_k \times C_{\alpha,\beta}$. Противник A, получая на вход данные функции, запускает вторую функцию на первой. То есть A (F, G) = F(G). Тогда для данного случая получаются следующие соотношения:

$$Pr\,[A(O(F_{\alpha,\beta})) = 1] = 1,$$

$$Pr\,[A(O(G_{\alpha,\beta})) = 1] = 0,$$

$$| Pr[S^{F\alpha,\beta}(1^k) = 1] - Pr[S^{G\alpha,\beta}(1^k) = 1] | \leq 2^{-\Omega(k)}$$

В данном случае вероятности берутся равновероятно из $\alpha, \beta \in \{0,1\}^k$ и подбрасываются монетки A, S, O. Это противоречит определению обфускатора, стойкого в модели черный ящик (п. 1.2.1).

Однако, реализация возможна, если ввести дополнительные условности:

- обфускация представлена в виде отдельного класса функций, так называемые точечные функции;

- введены дополнительные меры защиты, а именно криптографические параметры.

Следует отметить следующие работы по теории обфускации:

– «О перспективах решения задачи обфускации компьютерных программ» [5]. В данной работе авторы анализируют возможные направления применения обфускации программ;

– «Положительные результаты и методы обфускации» от Линн Б., Прабхакаран М., Сахай А. [6]. В данной работе авторы приводят положительные результаты по обфусцированию программ, которые можно

считать начальным прогрессом в развитии обфускации;

– «Обфускация для криптографических целей» от Хофхейнц Д., Малон-Ли Дж., Стэм М. [7]. В данной работе авторы вводят новое понятие обфускации, предлагают безопасный алгоритм обфусцирования.

Наиболее значительное продвижение в доказательстве существования обфускаторов в модели «черный ящик» было достигнуто в работе Хохенбергера С. и других. Авторы доказали осуществимость стойкой обфускации программ перешифрования сообщений [8].

В 2016 году в работе «Теоретическое обоснование стойкости неразличимой обфускации» Козачок А.В., Бочков М.В. и Лай М. Т. представили алгоритм декомпозиции элементов и описали модель обфускатора в виде модели со случайным оракулом [9].

Таким образом, пока ещё не найдена настолько сложная функция, программы вычисления которой допускают доказуемо стойкую обфускацию в модели «черного ящика».

Из-за больших требований для моделей «серого ящика» и с дополнительным входом, нахождение достаточно сложной функции для данных моделей не достижимо [10].

2 Стойкость современных программ-обфускаторов

По введенному определению математической стойкости обфускации существование стойких обфускаторов невозможно. Для решения данной проблемы стойкость будет считаться, как и для всех обычных программ.

Для оценки стойкости программы будет использоваться действующий ГОСТ 28195-89 «Оценка качества программных средств. Общие положения» [11].

В данном ГОСТе описываются критерии оценивания программных средств по следующим показателям:

- надежности ПС;
- сопровождения;
- удобства применения;
- эффективности;
- универсальности;
- корректности.

Именно к показателям надежности относится устойчивость функционирования программы.

Устойчивость функционирования – это способность обеспечивать продолжение работы программы после возникновения отклонений, вызванных сбоями и ошибками во время работы программы.

Так как сбои и ошибки могут инициироваться злоумышленниками для нанесения ущерба, то данный показатель является важным для оценивания средств по защите информации, к которым относятся программы обфсукации.

Показатель устойчивости к искажающим воздействиям является расчётным показателем и вычисляется по следующей формуле [11]:

$$P(Y) = 1 – D/K, \quad (2)$$

где D – число экспериментов, в которых искажающие воздействия приводили

к отказу,

K – число экспериментов, в которых имитировались искажающие воздействия.

В данной работе в качестве средств защиты информации будут рассматриваться следующие программы обфускаторы:

- бесплатное программное средство – *JavaScript Obfuscator Tool*;
- платное программное средство – *Premium Javascript Obfuscator*;
- собственная программа-обфускатор (приложение А).

Эксперименты будут проводиться для двух видов: ошибки входных данных, искусственная генерация сбоя работы программы. В качестве входных данных будет использоваться скрипт *JavaScript*, представленный в приложении Б.

Пусть ошибки входных данных будут являться ошибками 1 типа. Показатель устойчивости к искажениям 1 типа будет считаться:

$$P(Y1) = 1 - D1/K1, \quad (3)$$

где D1 – число экспериментов, в которых ошибки 1 типа приводили к отказу,

K1 – число экспериментов, в котором имитировалась ошибка 1 типа.

Тогда ошибки, связанные со сбоями программы, являются ошибкам 2 типа. Показатель устойчивости к искажениям 2 типа будет считаться:

$$P(Y2) = 1 - D2/K2, \quad (4)$$

где D2 – число экспериментов, в которых ошибки 2 типа приводили к отказу,

K2 – число экспериментов, в котором имитировалась ошибка 2 типа.

2.1 Программа-обфускатор *JavaScript Obfuscator Tool*

JavaScript Obfuscator Tool является бесплатным обфускатором,

разработанным российским программистом Тимофеем Качаловым [12]. Данный обфускатор имеет открытый исходный код, а это значит, что с ним можно подробнее ознакомиться.

Данное приложение имеет легкодоступный интерфейс с возможностью загрузки файла с устройства или же вставки программного кода в окно ввода информации (рис. 4).

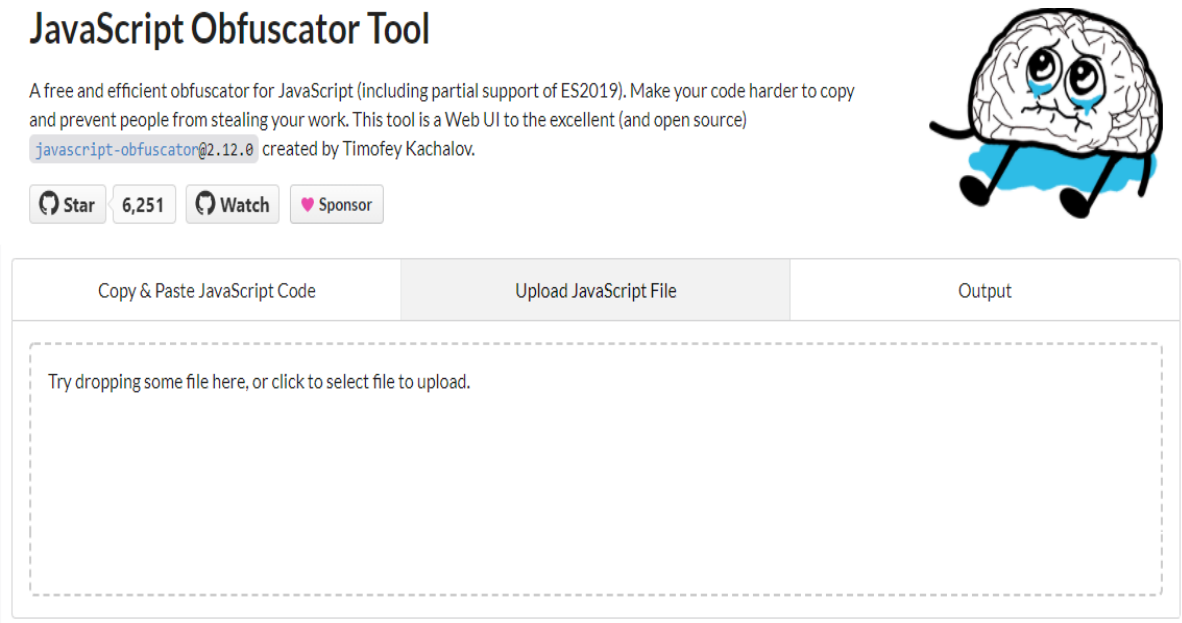

Рисунок 4 – Интерфейс загрузки файла в *JavaScript Obfuscator Tool*

Данная программа использует следующие преобразования переименование переменных, функций, аргументов, удаление комментариев, упрощение и другое. Все преобразования настраиваются опционально в зависимости от требований пользователя в отдельном интерфейсе (рис. 5).

При проведении экспериментов будут использоваться стандартные настройки программы, представленные на рисунке 5.

Для расчёта показателей устойчивости будут проводиться по несколько десятков экспериментов для генерации каждого вида ошибок и рассчитываться на каждый этап показатель устойчивости. Далее будет посчитано среднее значение показателя для данной программы.

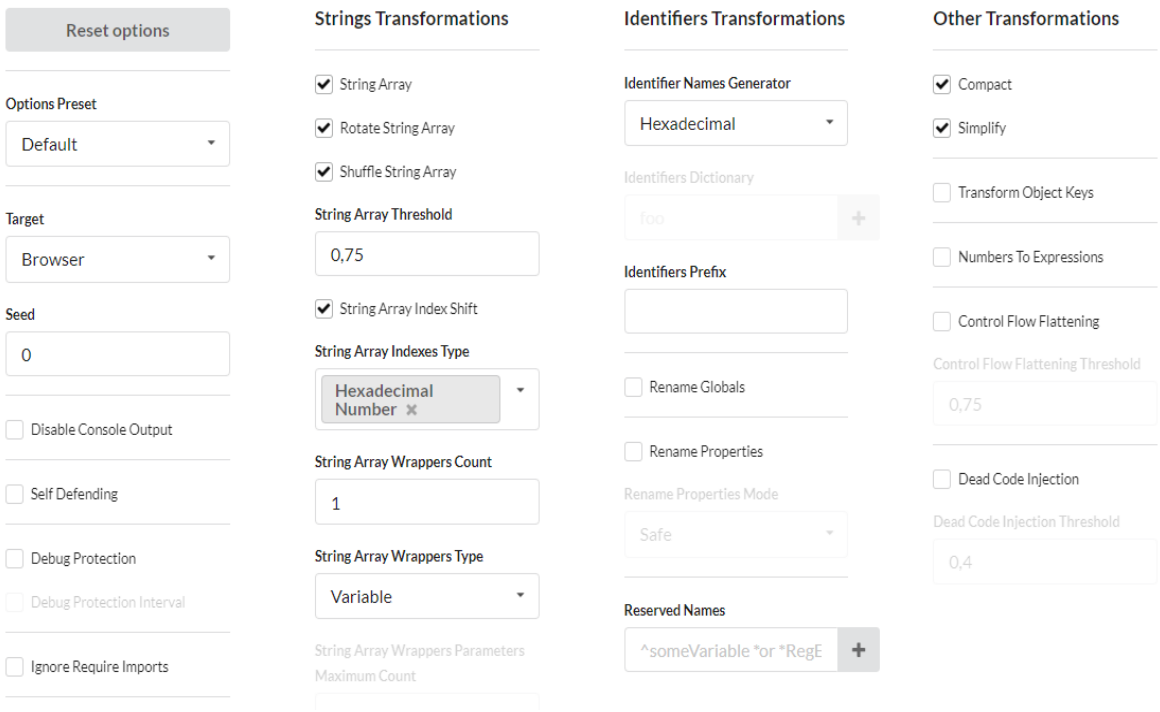

Рисунок 5 – Интерфейс выбора преобразований

Эксперименты при симуляции ошибок 1 типа.

В ходе проведения экспериментов с генерацией ошибок 1 типа были получены следующие результаты (рис. 6):

1) после проведения пяти экспериментов было выявлено три отрицательных результата. Показатель для пяти случаев равен: $P_5(Y1) = 1 – 3/5 = 0,4$;

2) после проведения десяти экспериментов было выявлено пять отрицательных результата. Показатель для десяти случаев равен: $P_{10}(Y1) = 1 – 5/10 = 0,5$;

3) после проведения пятнадцати экспериментов было выявлено восемь отрицательных результатов. Показатель для пятнадцати случаев равен: $P_{15}(Y1) = 1 – 8/15 = 0,47$;

4) после проведения двадцати экспериментов было выявлено одиннадцать отрицательных результатов. Показатель для двадцати случаев равен: $P_{20}(Y1) = 1 – 11/20 = 0,45$;

5) после проведения двадцати пяти экспериментов было выявлено тринадцать отрицательных результатов. Показатель для двадцати пяти

случаев равен: $P_{25}(Y1) = 1 – 13/25 = 0,48$.

Среднее значение показателя устойчивости равняется $P(Y1) = 0,46$.

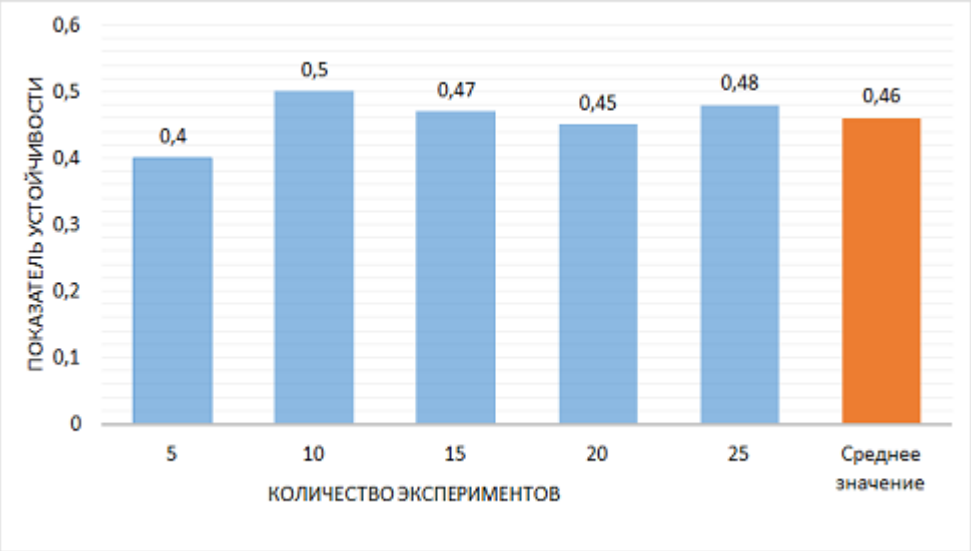

Рисунок 6 – Гистограмма показателей устойчивости к ошибкам 1 типа для программы *JavaScript Obfuscator Tool*

Эксперименты при симуляции ошибок 2 типа.

В ходе проведения экспериментов с генерацией ошибок 2 типа были получены следующие результаты (рис. 7):

1) после проведения пяти экспериментов было выявлено четыре отрицательных результата. Показатель для пяти случаев равен: $P_5(Y2) = 1 – 4/5 = 0,2$;

2) после проведения десяти экспериментов было выявлено семь отрицательных результата. Показатель для десяти случаев равен: $P_{10}(Y2) = 1 – 7/10 = 0,3$;

3) после проведения пятнадцати экспериментов было выявлено двенадцать отрицательных результатов. Показатель для пятнадцати случаев равен: $P_{15}(Y2) = 1 – 12/15 = 0,2$;

4) после проведения двадцати экспериментов было выявлено семнадцать отрицательных результатов. Показатель для двадцати случаев равен: $P_{20}(Y2) = 1 – 17/20 = 0,15$;

5) после проведения двадцати пяти экспериментов было выявлено девятнадцать отрицательных результатов. Показатель для двадцати пяти случаев равен: $P_{25}(Y2) = 1 – 19/25 = 0,24$.

Среднее значение показателя устойчивости равняется $P(Y2) = 0,23$.

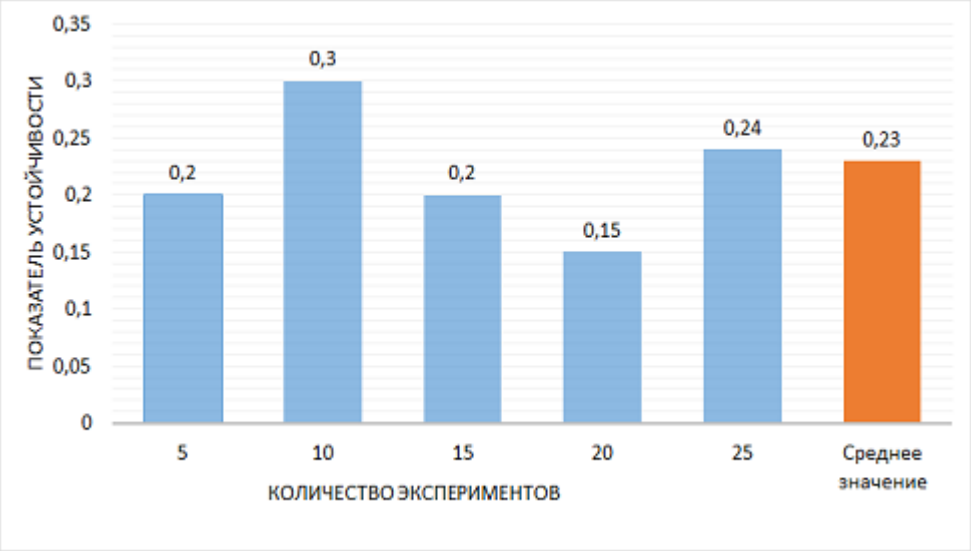

Рисунок 7 – Гистограмма показателей устойчивости к ошибкам 2 типа для программы *JavaScript Obfuscator Tool*

Эксперименты при симуляции ошибок 1 и 2 типа.

В ходе проведения экспериментов с генерацией ошибок двух типов одновременно были получены следующие результаты (рис. 8):

1) после проведения пяти экспериментов было выявлено четыре отрицательных результата. Показатель для пяти случаев равен: $P_5(Y) = 1 – 4/5 = 0,2$;

2) после проведения десяти экспериментов было выявлено восемь отрицательных результата. Показатель для десяти случаев равен: $P_{10}(Y) = 1 – 8/10 = 0,2$;

3) после проведения пятнадцати экспериментов было выявлено тринадцать отрицательных результатов. Показатель для пятнадцати случаев равен: $P_{15}(Y) = 1 – 13/15 = 0,13$;

4) после проведения двадцати экспериментов было выявлено

восемнадцать отрицательных результатов. Показатель для двадцати случаев равен: $P_{20}(Y) = 1 – 18/20 = 0,1$;

5) после проведения двадцати пяти экспериментов было выявлено двадцать один отрицательный результат. Показатель для двадцати пяти случаев равен: $P_{25}(Y) = 1 – 21/25 = 0,16$.

Среднее значение показателя устойчивости равняется $P(Y) = 0,16$.

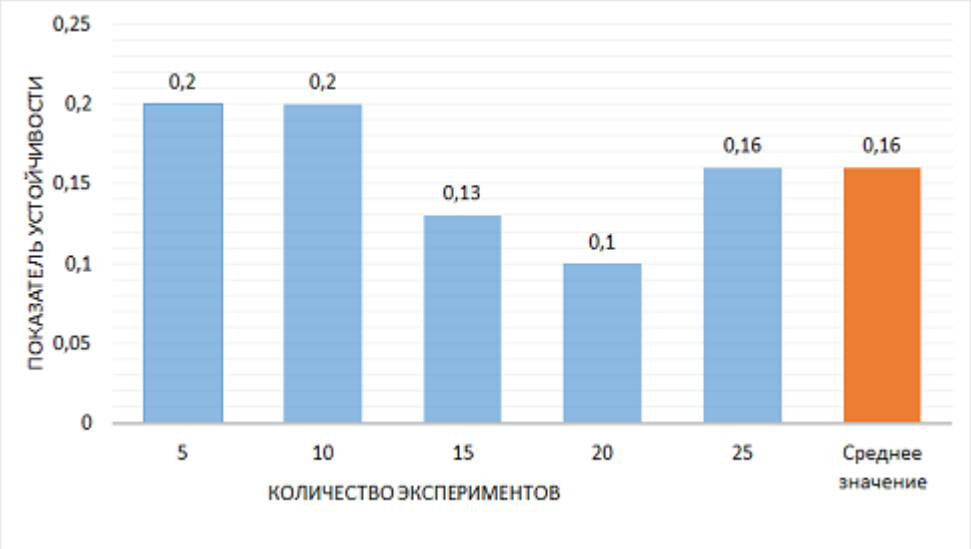

Рисунок 8 – Гистограмма показателей устойчивости к ошибкам для программы *JavaScript Obfuscator Tool*

2.2 Программа-обфускатор *Premium JavaScript Obfuscator*

Данная программа, как и предыдущая, обеспечивает обфускацию программного кода, однако, уже скрывает свой исходный код и является платным [13]. Стоимость ежемесячной подписки на данный товар с основными функциями, удовлетворяющими частным пользователям, составляет 19 долларов США. Программа имеет ряд преимуществ над бесплатным приложением:

– интерфейс программы интуитивно понятен для восприятия (рис. 9);

– поддерживает загрузку программного кода извне или ввода в самой программе более 1000 Мб;

– использование более сложных алгоритмов обфускации.

Также программа имеет защищенный доступ, требует аутентификации в сети. Среди методов обфускации используются: замена переменных, изменение порядка функций, вставка мертвого кода, замена глобальных переменных, индивидуальные алгоритмы обфускации, скрытые от пользователей.

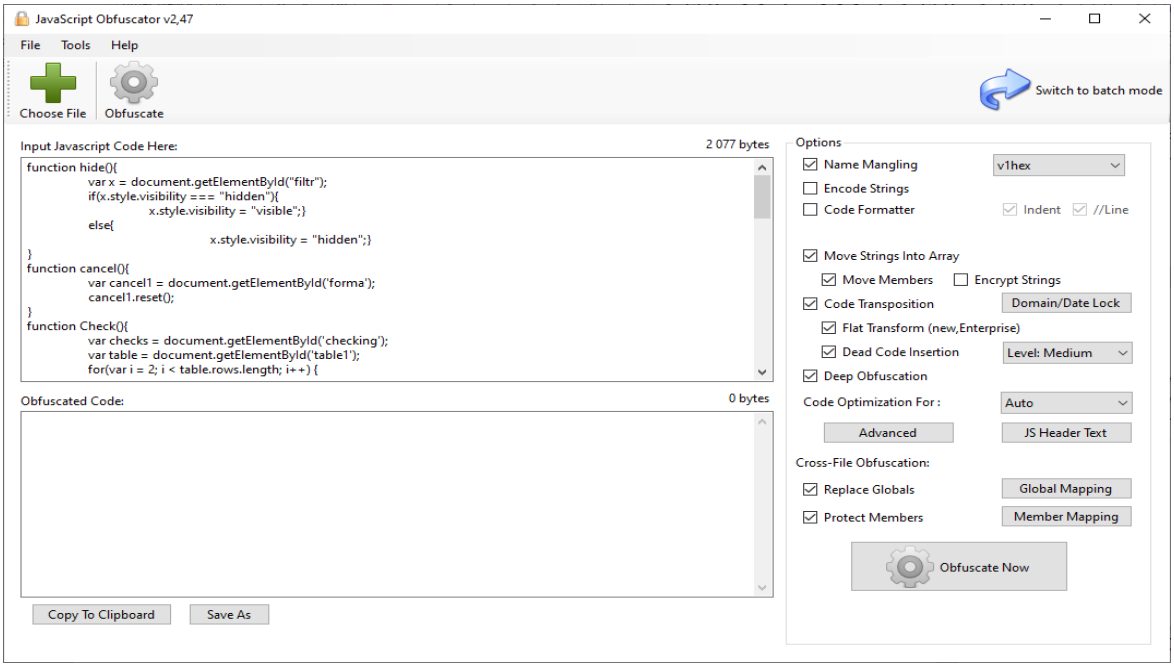

Рисунок 9 – Интерфейс программы-обфускатора *Premium JavaScript Obfuscator*

Эксперименты при симуляции ошибок 1 типа.

В ходе проведения экспериментов с генерацией ошибок 1 типа были получены следующие результаты (рис. 10):

1) после проведения пяти экспериментов было выявлено два отрицательных результата. Показатель для пяти случаев равен: $P_5(Y1) = 1 – 2/5 = 0,6$;

2) после проведения десяти экспериментов было выявлено три отрицательных результата. Показатель для десяти случаев равен: $P_{10}(Y1) = 1 – 3/10 = 0,7$;

3) после проведения пятнадцати экспериментов было выявлено пять

отрицательных результатов. Показатель для пятнадцати случаев равен: $P_{15}(Y1) = 1 – 5/15 = 0,67$;

4) после проведения двадцати экспериментов было выявлено семь отрицательных результатов. Показатель для двадцати случаев равен: $P_{20}(Y1) = 1 – 7/20 = 0,65$;

5) после проведения двадцати пяти экспериментов было выявлено девять отрицательных результатов. Показатель для двадцати пяти случаев равен: $P_{25}(Y1) = 1 – 9/25 = 0,67$.

Среднее значение показателя устойчивости равняется $P(Y1) = 0,65$.

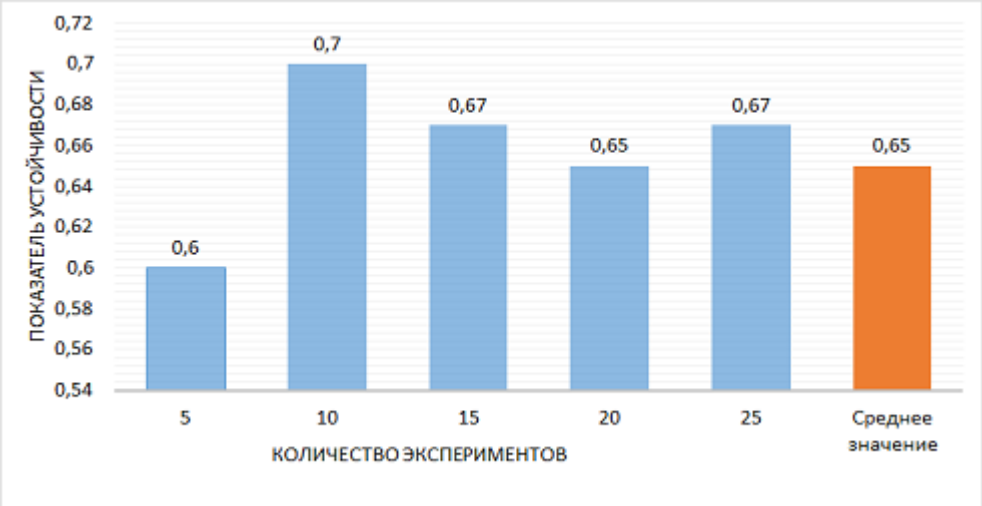

Рисунок 10 – Гистограмма показателей устойчивости к ошибкам 1 типа для программы *Premium JavaScript Obfuscator*

Эксперименты при симуляции ошибок 2 типа.

В ходе проведения экспериментов с генерацией ошибок 2 типа были получены следующие результаты (рис. 11):

1) после проведения пяти экспериментов было выявлено три отрицательных результата. Показатель для пяти случаев равен: $P_5(Y2) = 1 – 3/5 = 0,4$;

2) после проведения десяти экспериментов было выявлено пять отрицательных результата. Показатель для десяти случаев равен: $P_{10}(Y2) = 1 – 5/10 = 0,5$;

3) после проведения пятнадцати экспериментов было выявлено семь отрицательных результатов. Показатель для пятнадцати случаев равен: $P_{15}(Y2) = 1 – 7/15 = 0,53$;

4) после проведения двадцати экспериментов было выявлено девять отрицательных результатов. Показатель для двадцати случаев равен: $P_{20}(Y2) = 1 – 9/20 = 0,55$;

5) после проведения двадцати пяти экспериментов было выявлено двенадцать отрицательных результатов. Показатель для двадцати пяти случаев равен: $P_{25}(Y2) = 1 – 12/25 = 0,52$.

Среднее значение показателя устойчивости равняется $P(Y2) = 0,5$.

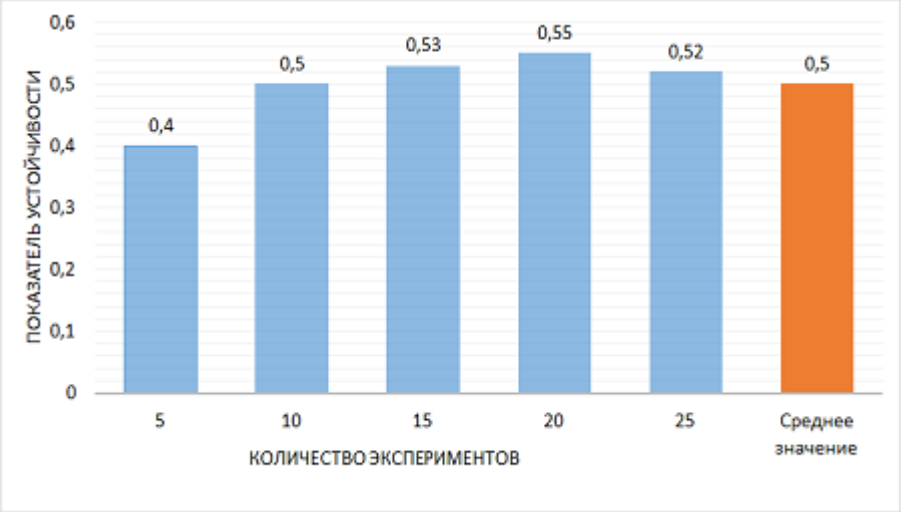

Рисунок 11 – Гистограмма показателей устойчивости к ошибкам 2 типа для программы *Premium JavaScript Obfuscator*

Эксперименты при симуляции ошибок 1 и 2 типа.

В ходе проведения экспериментов с генерацией ошибок двух типов одновременно были получены следующие результаты (рис. 12):

1) после проведения пяти экспериментов было выявлено четыре отрицательных результата. Показатель для пяти случаев равен: $P_5(Y) = 1 – 4/5 = 0,2$;

2) после проведения десяти экспериментов было выявлено семь

отрицательных результата. Показатель для десяти случаев равен: $P_{10}(Y) = 1 – 7/10 = 0,3$;

3) после проведения пятнадцати экспериментов было выявлено девять отрицательных результатов. Показатель для пятнадцати случаев равен: $P_{15}(Y) = 1 – 9/15 = 0,4$;

4) после проведения двадцати экспериментов было выявлено тринадцать отрицательных результатов. Показатель для двадцати случаев равен: $P_{20}(Y) = 1 – 13/20 = 0,35$;

5) после проведения двадцати пяти экспериментов было выявлено восемнадцать отрицательных результатов. Показатель для двадцати пяти случаев равен: $P_{25}(Y) = 1 – 18/25 = 0,28$.

Среднее значение показателя устойчивости равняется $P(Y) = 0,31$.

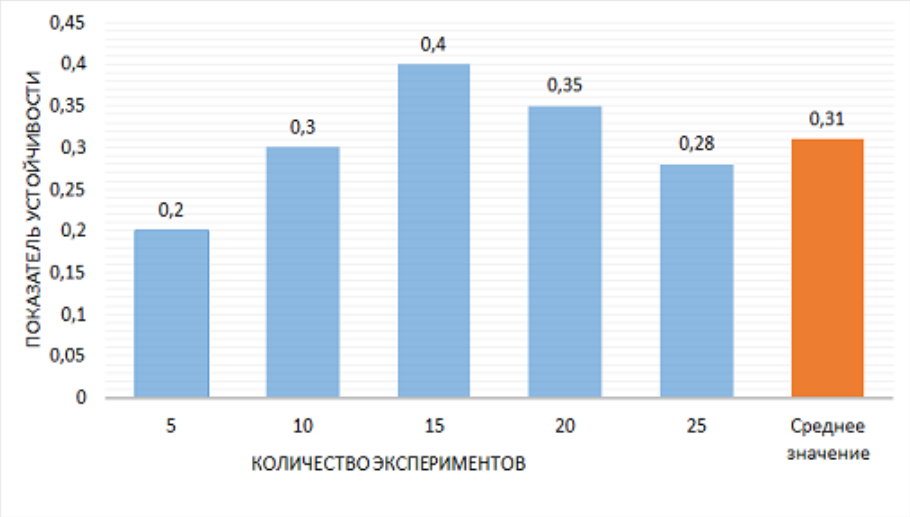

Рисунок 12 – Гистограмма показателей устойчивости к ошибкам для программы *Premium JavaScript Obfuscator*

2.3 Программа-обфускатор, разработанная на основе анализа существующих

На основании оценки выше описанных программ, разработана собственная программа-обфускатор, полный код которой представлен в приложении А.

В качестве языка программирования будет использоваться такой язык программирования как *C#*.

Требования к разрабатываемой программе:

– поддержка, т.е. возможность обфусцирования, кода, написанного на языке программирования *Javascript*;

– использование методов обфускации, используемых в современных программах-обфускаторах.

В программе будут использоваться следующие методы обфускации:

– переименование переменных;

– переименование функций;

– лексическая обфускация;

– вставка мертвого кода;

– изменение порядка функции.

Для реализации данных методов используются следующие элементы:

– словари для переименования функций;

– словари для переименования переменных;

– функции добавления бессмысленного кода, не влияющего на работу программы (вставка мертвого кода);

– словари для изменения порядка функций.

Также, как и для предыдущих программ, следует оценить показатели устойчивости для ошибок различного типа.

Эксперименты при симуляции ошибок 1 типа.

В ходе проведения экспериментов с генерацией ошибок 1 типа были получены следующие результаты (рис. 13):

1) после проведения пяти экспериментов было выявлено два отрицательных результата. Показатель для пяти случаев равен: $P_5(Y1) = 1 – 2/5 = 0,6$;

2) после проведения десяти экспериментов было выявлено пять

отрицательных результата. Показатель для десяти случаев равен: $P_{10}(Y1) = 1 – 5/10 = 0,5$;

3) после проведения пятнадцати экспериментов было выявлено семь отрицательных результатов. Показатель для пятнадцати случаев равен: $P_{15}(Y1) = 1 – 7/15 = 0,53$;

4) после проведения двадцати экспериментов было выявлено десять отрицательных результатов. Показатель для двадцати случаев равен: $P_{20}(Y1) = 1 – 10/20 = 0,5$;

5) после проведения двадцати пяти экспериментов было выявлено тринадцать отрицательных результатов. Показатель для двадцати пяти случаев равен: $P_{25}(Y1) = 1 – 13/25 = 0,48$.

Среднее значение показателя устойчивости равняется $P(Y1) = 0,52$.

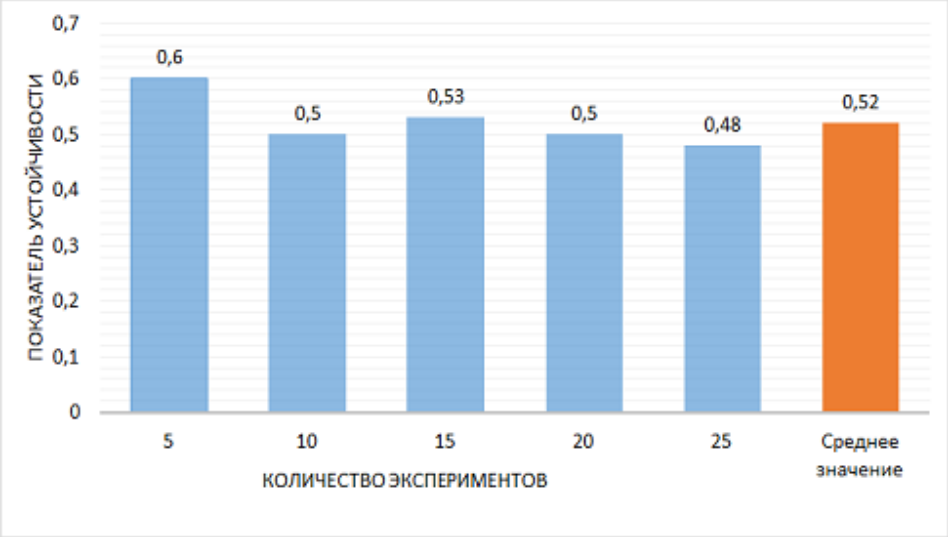

Рисунок 13 – Гистограмма показателей устойчивости к ошибкам 1 типа для авторской программы

Эксперименты при симуляции ошибок 2 типа.

В ходе проведения экспериментов с генерацией ошибок 2 типа были получены следующие результаты (рис. 14):

1) после проведения пяти экспериментов было выявлено три отрицательных результата. Показатель для пяти случаев равен:

$P_5(Y2) = 1 – 3/5 = 0,4$;

2) после проведения десяти экспериментов было выявлено семь отрицательных результата. Показатель для десяти случаев равен: $P_{10}(Y2) = 1 – 7/10 = 0,3$;

3) после проведения пятнадцати экспериментов было выявлено девять отрицательных результатов. Показатель для пятнадцати случаев равен: $P_{15}(Y2) = 1 – 9/15 = 0,4$;

4) после проведения двадцати экспериментов было выявлено тринадцать отрицательных результатов. Показатель для двадцати случаев равен: $P_{20}(Y2) = 1 – 13/20 = 0,35$;

5) после проведения двадцати пяти экспериментов было выявлено восемнадцать отрицательных результатов. Показатель для двадцати пяти случаев равен: $P_{25}(Y2) = 1 – 18/25 = 0,28$.

Среднее значение показателя устойчивости равняется $P(Y2) = 0,35$.

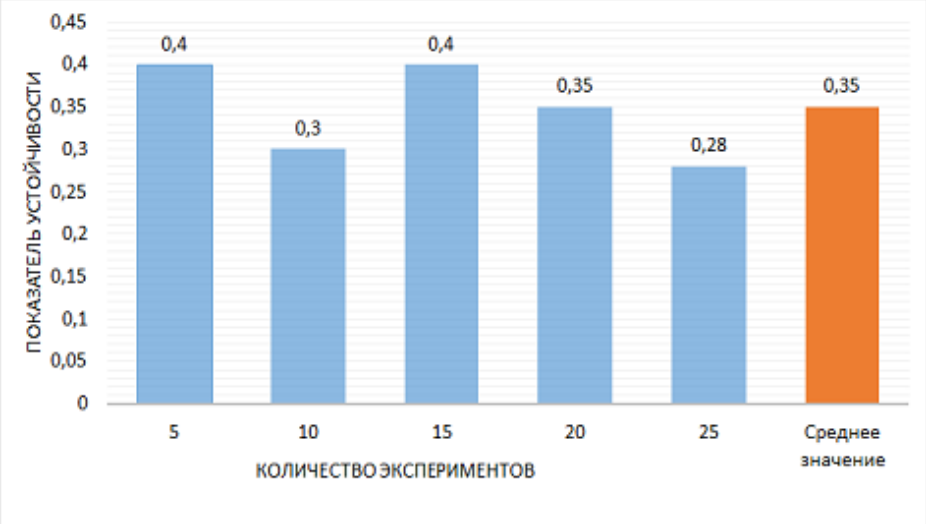

Рисунок 14 – Гистограмма показателей устойчивости к ошибкам 2 типа для авторской программы

Эксперименты при симуляции ошибок 1 и 2 типа.

В ходе проведения экспериментов с генерацией ошибок двух типов одновременно были получены следующие результаты (рис. 15):

1) после проведения пяти экспериментов было выявлено четыре

отрицательных результата. Показатель для пяти случаев равен: $P_5(Y) = 1 – 4/5 = 0,2$;

2) после проведения десяти экспериментов было выявлено семь отрицательных результата. Показатель для десяти случаев равен: $P_{10}(Y) = 1 – 7/10 = 0,3$;

3) после проведения пятнадцати экспериментов было выявлено одиннадцать отрицательных результатов. Показатель для пятнадцати случаев равен: $P_{15}(Y) = 1 – 11/15 = 0,27$;

4) после проведения двадцати экспериментов было выявлено пятнадцать отрицательных результатов. Показатель для двадцати случаев равен: $P_{20}(Y1) = 1 – 15/20 = 0,25$;

5) после проведения двадцати пяти экспериментов было выявлено двадцать один отрицательный результат. Показатель для двадцати пяти случаев равен: $P_{25}(Y) = 1 – 21/25 = 0,16$.

Среднее значение показателя устойчивости равняется $P(Y) = 0,24$.

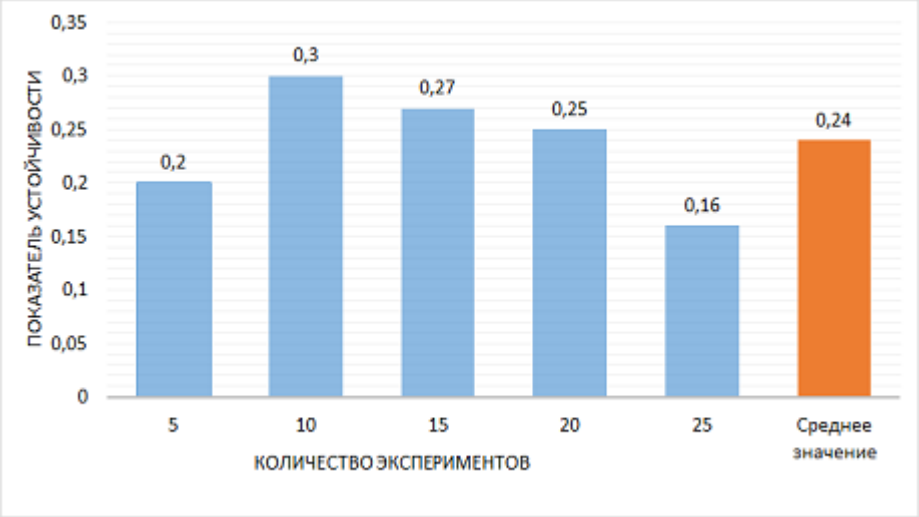

Рисунок 15 – Гистограмма показателей устойчивости к ошибкам для программы *JavaScript Obfuscator Tool*

2.4 Сравнительный анализ программ-обфускаторов

По полученным результатам в таблице 1 видно, что наиболее

устойчивым обфускатором является платная программа *Premium Javascript Obfuscator*. Данная программа показывает лучшие результаты среди её бесплатных аналогов. Её использование разумно для обфусцирования кодов в коммерческих целях для предотвращения кражи информации, когда утеря или хищение информации может понести весомый материальный ущерб разработчикам.

Таблица 1 – Устойчивость обфускаторов.

	Устойчивость к ошибке 1 типа	Устойчивость к ошибке 2 типа	Общая устойчивость
JavaScript Obfuscator Tool	0,46	0,23	0,16
Premium Javascrtipt Obfusctator	0,65	0,5	0,31
Авторский	0,52	0,35	0,24

Стоит отметить, что самодельное приложение имеет лучшую устойчивость к ошибкам, чем бесплатное приложение. Это связано с тем, что в самодельном используются дополнительные методы обфускации.

Таким образом, для обеспечения большей устойчивости кода в индивидуальных целях следует использовать разработанный обфускатор.

ЗАКЛЮЧЕНИЕ

Так как с каждым годом увеличивается количество доступных сайтов и программ, то для защиты их исходного кода от несанкционированного доступа применяются средства обфускации. Если для увеличения стойкости программы используются криптографические и стеганографические методы, то для защиты от несанкционированного доступа, взлома программы и обхода лицензии используются методы запутывания, которые включают в себя методы обфускации.

В результате проведённой работы получены следующие результаты:

– проанализированы математические методы обфускации;

– проведен анализ стойкости программ-обфускаторов;

– разработана программа-обфускатор на основе анализа уже существующих;

– проведен сравнительный анализ программ-обфускаторов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Чернов, А. В. Анализ запутывающих преобразований программ [Электронный ресурс] / А.В. Чернов // Труды ИСП РАН. – 2002. – URL: https://cyberleninka.ru/article/n/analiz-zaputyvayuschih-preobrazovaniy-programm (дата обращения: 10.11.2020).

2 Юричев, Д. Введение в reverse engineering для начинающих [Электронный ресурс] / Д. Юричев. – 2020. – URL: https://beginners.re/RE4B-RU.pdf (дата обращения 9.11.2020).

3 Курмангалеев, Ш. Ф. Описание подхода к разработке обфусцирующего компилятора [Электронный ресурс] / Ш.Ф. Курмангалеев, В. П. Корчагин, Р. А. Матевосян // Труды ИСП РАН – 2012. – URL: https://cyberleninka.ru/article/n/opisanie-podhoda-k-razrabotke-obfustsiruyusche-go -kompilyatora (дата обращения: 11.11.2020).

4 Боаз, Б. О (не)возможности обфускации программ [Текст] / Б. Барак, О. Голдрейх, Р. Импальяццо // Конспект лекций по информатике. – 2001. – С. 1-18.

5 Варновский, Н.П. О перспективах решения задачи обфускации компьютерных программ [Текст] / Н.П. Варновский и [др] // Труды конференции (МаБИТ-03), Москва, 22-24 октября 2003 г. – М.,2003. – С. 344-352.

6 Линн, Б. Положительные результаты и методы обфускации [Текст] / Б. Линн, М. Прабхакаран, А. Сахай // Достижения в криптологии – Еврокрип 2004, Лекции по информатике – 2004. – №3027 – С. 20-39.

7 Хофхейнц, Д. Обфускация для криптографических целей [Текст] / Д. Хофхейнц, Дж. Мэлон-Ли, М. Стэм // Конференция по теории криптографии, Конспект лекций по информатике – 2007. – №4932 – С. 214 - 232.

8 Безопасное запутывание повторного шифрования [Текст] / С. Хохенберг [и др.] // Журнал криптологии – 2010. – №24 – С. 694 - 719.

9 Козачок, А.В. Теоретическое обоснование стойкости неразличимой обфускации [Текст] / Вопросы кибербезопасности – 2016. – №1(14) – С. 36-46.

10 Варновский, Н.П. О стойкой обфускации компьютерных программ [Текст] / Н.П. Варновский, В.А. Захаров, Н.Н. Кузюрин, Шокуров А.В. // Экономика. Информатика – 2009. - №15(70). – С. 97-105.

11 ГОСТ 28195-89 Оценка качества программных средств. Общие положения [Текст] – Введен: 1990-01-07. - М.: Издательство стандартов, 1989. – 30 с.

12 JavaScript Obfuscator Tool [Электронный ресурс] / Т. Качалов, Т. Серафим – Прикладная программа. – 2020. – URL: https://www.obfuscator.io/ (дата обращения 15.04.2020).

13 Javascript Obfuscator [Электронный ресурс] – Прикладная программа. – 2020. – URL: https://javascriptobfuscator.com/Javascript-Obfuscator.aspx (дата обращения 17.04.2020).

ПРИЛОЖЕНИЕ А

Программный код обфускатора, разработанного на анализе уже существующих бесплатных программ-обфускаторов.

```
static Dictionary<string, string> names = new Dictionary<string, string>();
static bool testing = false;
static string[] files1 =  @"a.js,b.js,c.js"   .Split(new string[] { Environment.NewLine, " ", "\t", "," }, StringSplitOptions.RemoveEmptyEntries);
static string[] ignore_names =
    @"sin,cos,order,min,max,join,round,pow,abs,PI,floor,random,index,http,__defineGetter__,__defineSetter__,indexOf,isPrototypeOf,length,clone,toString,split,clear,erase
RECT,SIZE,Vect,VectInt,vectint,vect,int,double,canvasElement,text1,text2,text3,textSizeTester,target,Number
    number,TimeStep,images,solid,white,default,cursive,fantasy,".
  Split(new string[] { Environment.NewLine, " ", "\t", "," }, StringSplitOptions.RemoveEmptyEntries);
string[] extra_names = @"a,b,c".Split(new string[] { Environment.NewLine, " ", "\t", "," }, StringSplitOptions.RemoveEmptyEntries);
string src = @"C:\temp";
string dest1 = src + "\all1.js";
string dest2 = src + "\all2.js";
static void Main()
{
  File.Delete(dest1);
  File.Delete(dest2);
  foreach (string s in files1)
    File.AppendAllText(dest1, File.ReadAllText(src + "\" + s) + Environment.NewLine + Environment.NewLine + Environment.NewLine +
```

```
Environment.NewLine + Environment.NewLine + Environment.NewLine, Encoding.UTF8);
  string all = File.ReadAllText(dest1);
  int free_index = 0;
  foreach (string s in extra_names)
  {
    free_index++;
    string free_name = "" + (char)('A' + (free_index % 25)) + (char)('A' + ((free_index / 25) % 25));
    Debug.Assert(free_name != "AA");
    names.Add(s, free_name);
  }
  Regex reg1 = new Regex("(var |function |\.prototype\.)([a-zA-Z0-9_]+)");
  int startat = 0;
  while (startat < all.Length)
  {
    Match match = reg1.Match(all, startat);
    if (!match.Success)
      break;
    string key = all.Substring(match.Groups[2].Index, match.Groups[2].Length);
    if (!ignore_names.Contains(key))
    {
      free_index++;
      string free_name = "" + (char)('A' + (free_index % 25)) + (char)('A' + ((free_index / 25) % 25));
      Debug.Assert(free_name != "AA");
      if (!names.ContainsKey(key))
        names.Add(key, testing ? key + free_name : free_name);
    }
    startat = match.Groups[0].Index + match.Groups[0].Length;
```

```
  }
  Regex reg2 = new Regex(@"/\*.*\*/", RegexOptions.Multiline);
  Regex reg3 = new Regex("([^:\\])//.*\r\n");
  Regex reg4 = new Regex("([a-zA-Z0-9_]+)");
  Regex reg5 = new Regex("(\r\n)*[ \t]+");
  Regex reg6 = new Regex("(\r\n)+");
  all = reg2.Replace(all, eval2);
  all = reg3.Replace(all, eval3);
  all = reg4.Replace(all, eval4);
  all = reg5.Replace(all, eval5);
  all = reg6.Replace(all, eval6);
  File.WriteAllText(dest2, all);
}
public static string eval4(Match match)
{
  return names.ContainsKey(match.Groups[1].Value) ?
names[match.Groups[1].Value] : match.Groups[0].Value;
}
public static string eval5(Match match)
{
  return string.IsNullOrEmpty(match.Groups[1].Value) ? " " :
Environment.NewLine;
}
public static string eval6(Match match)
{
  return Environment.NewLine;
}
public static string eval2(Match match)
{
  return " ";
```

```
}
public static string eval3(Match match)
{
  return match.Groups[1].Value + Environment.NewLine;
}
```

ПРИЛОЖЕНИЕ Б

Обфусцируемый программный код, написанный на языке Javascript

```
function hide(){
	var x = document.getElementById("filtr");
	if(x.style.visibility === "hidden"){
		x.style.visibility = "visible";}
	else{
			x.style.visibility = "hidden";}
}

function cancel(){
	var cancel1 = document.getElementById('forma');
	cancel1.reset();
}

function Check(){
	var checks = document.getElementById('checking');
	var table = document.getElementById('table1');
	for(var i = 2; i < table.rows.length; i++) {
		if(checks.checked && checks.value ==
(table.rows[i].cells[4].innerHTML)){
			table.rows[i].style.visibility = "collapse";}
		else{table.rows[i].style.visibility = "initial"};
	}
}

function Name() {
	var inputName = document.getElementById('FIO');
```

```
        var table = document.getElementById('table1');
        var searchName = new RegExp(inputName.value, 'i');
        var flag = false;
        for(var i = 2; i < table.rows.length; i++) {
                flag = searchName.test(table.rows[i].cells[1].innerHTML);
                if(flag){
                        table.rows[i].style.display = "";
                }else{
                        table.rows[i].style.display = "none";
                }
        }
}

function Diaposon(){
        var table = document.getElementById('table1');
        var min = document.getElementById('ot');
        var max = document.getElementById('do');
        for(var i = 2; i < table.rows.length; i++){
                if(min.value == '' && max.value != ''){
                        min.value=0;}
                if(min.value != '' && max.value ==''){
                        max.value=100000;}
                if(min.value !='' && max.value != ''){
                        if(min.value >
parseInt((table.rows[i].cells[3].innerHTML),10)){
                                table.rows[i].style.display = "none";}
                        if(max.value <
parseInt((table.rows[i].cells[3].innerHTML),10)){
                                table.rows[i].style.display = "none";}
                }
```

```
        }
}

function Tipy(){
        var table = document.getElementById('table1');
        var tips = document.getElementById('tip');
        var type = 10;
        for (var i=0; i<tips.childNodes.length;i++){
                if (tips.childNodes[i].selected){
                        type=tips.childNodes[i].value;}
        }
        if (type != 10){
                for(var i = 2; i < table.rows.length; i++){
                        if (type != table.rows[i].cells[2].innerHTML){
                                table.rows[i].style.display = "none";}
                }
        }
}
```