Об особенностях применения методов обфускации программного кода языка JavaScript

А.О. Кузнецова
Факультет информатики и робототехники
Уфимский государственный авиационный технический университет
Уфа, Россия
e-mail: Anna.Kuznetsova111@yandex.ru

Г.Н. Верхотурова
Факультет информатики и робототехники
Уфимский государственный авиационный технический университет
Уфа, Россия
e-mail: verhoturova.gn@yandex.ru

Аннотация[1]

Данная статья посвящена проблеме защиты программного кода от несанкционированного использования. Процесс обфускации на сегодняшний день является одним из самых популярных и наиболее часто используемых способов защиты веб-приложений, поэтому в данной работе выполнен детальный анализ существующих инструментов и методов обфускации программного кода языка JavaScript. Целью проводимого исследования является построение обфускатора кода для защиты JavaScript-программ от нелегального доступа. Для достижения поставленной цели были проанализированы режимы обфускации и подробно рассмотрены заложенные в их основе запутывающие алгоритмы.

1. Введение

Интернет-пространство является популярным источником информации и способом общения для многих людей в мире. Оно охватывает большую аудиторию и очень быстро растет. К середине 2015 года число пользователей достигло 3,3 млрд человек. Во многом это обусловлено широким распространением сотовых сетей с доступом в Интернет стандартов 3G и 4G, развитием социальных сетей и удешевлением стоимости интернет-трафика. Такое стремительное развитие порождает риск снижения информационной безопасности. Исключительная важность обеспечения защиты приложения от несанкционированного доступа обуславливает распространение и постоянное усовершенствование средств защиты информации и расширение их номенклатуры. На сегодняшний день эту задачу можно обеспечить за счет целого комплекса разносторонних решений.

Труды Седьмой всероссийской научной конференции "Информационные технологии интеллектуальной поддержки принятия решений", 28-30 мая, Уфа-Ставрополь-Ханты-Мансийск, Россия, 2019

2. Обфускация программного кода

2.1. Определение обфускации

Для обхода защиты программы в большинстве случаев требуется изучение некоторого готового устройства или программного кода, а также документации на него с целью понять принцип его работы. Этот процесс называется «реверсивная (обратная) инженерия».

Самым известным и популярным способом защиты от реверс-инжиниринга является обфускация. Что же собой представляет обфускация? Обфускацией программы называется всякое ее преобразование, которое сохраняет вычисляемую программой функцию (эквивалентное преобразование), но при этом придает программе такую форму, что извлечение из текста программы (программного кода) ключевой информации об алгоритмах и структурах данных, реализованных в этой программе, становится трудоемкой задачей [1].

Процесс преобразования исходного кода является процессом обфускации, если он удовлетворяет следующим условиям:

- Визуально код модифицированной программы отличается от исходного кода, однако при тех же входных данных выдает тот же самый результат.
- Анализ модифицированного кода более сложный и трудоемкий, чем анализ исходного кода.
- При каждом процессе модификации результирующий код является различным.
- Обратное преобразование кода является неэффективным.

Цель обфускации программного кода заключается в том, чтобы затруднить понимание и анализ программного кода и воспрепятствовать целенаправленной его модификации.

Обфусцированной программой называется программа, которая после применения обфусцирующих преобразований на всех допустимых для исходной программы входных данных выдает тот

же самый результат, что и оригинальная программа, но более трудна для анализа, понимания и модификации [2].

2.2. Уровни обфускации

Выделяют следующие уровни процесса обфускации:

- низший уровень, когда процесс обфускации осуществляется над ассемблерным кодом программы, или даже непосредственно над двоичным файлом программы хранящим машинный код.
- высший уровень, когда процесс обфускации осуществляется над исходным кодом программы написанном на языке высокого уровня [3].

Осуществление обфускации на низшем уровне считается менее сложным процессом, но при этом более трудно реализуемым по ряду причин. Одна из этих причин заключается в том, что должны быть учтены особенности работы большинства процессоров, так как способ обфускации, приемлемый на одной архитектуре, может оказаться неприемлемым на другой.

Большинство существующих алгоритмов и методов обфускации (включая те, которые будут рассмотрены ниже) применяются для осуществления процесса обфускации как на низшем, так и на высшем уровне.

Также иногда может быть нецелесообразным подвергать обфускации весь код программы (например, из-за того, что в результате может значительно снизиться время выполнения программы), в таких случаях желательно производить обфускацию только наиболее важных участков кода.

2.3. Виды обфускации

Обфускацию программного кода можно разделить на виды, отталкиваясь от способа преобразования кода. Рассмотрим некоторые из них.

Лексическая обфускация. Является наиболее простой. Заключается в форматировании кода программы, изменении его структуры таким образом, чтобы он стал нечитаемым, менее информативным и более трудным для изучения.

Обфускация такого вида содержит:

- удаление всех комментариев в коде программы или изменение их на дезинформирующие;
- удаление различных пробелов, отступов, символов табуляции, используемых для лучшего визуального восприятия кода программы;
- замену имен идентификаторов (имен переменных, массивов, структур, функций, процедур и т.д.) на произвольные длинные наборы непонятных человеку символов;
- добавление различных мусорных (ненужных) операций;
- изменение расположения блоков (функций, процедур) программы таким образом, чтобы это никоим образом не повлияло на ее работоспособность.

Обфускация данных. Такой вид обфускации связан с модификацией структур данных. Он считается более сложным и замысловатым.

Обфускацию данных принято делить на три основные группы, которые описаны ниже.

a) *Обфускация хранения.* Суть состоит в преобразовании хранилищ данных, а также самих типов данных (например, создание и использование необычных типов данных, изменение представления существующих и т.д.). Существуют различные методы, позволяющие осуществить такую обфускацию. Например, изменение интерпретации данных определенного типа, изменение срока использования хранилищ данных, преобразование статических (неменяющихся) данных в процедурные, разделение переменных, изменение представления (или кодирование).

b) *Обфускация соединения.* Используется в процессе обратной инженерии программ, основан на изучении структур данных. Является необходимым для того, чтобы в процессе запутывания усложнить представление структур данных, используемых программой. Например, при использовании обфускации соединения это достигается благодаря соединению независимых данных или разделению зависимых. Основными методами для осуществления такого вида обфускации являются объединение переменных, реструктурирование массивов и изменение иерархий наследования классов

c) *Обфускация переупорядочивания.* Идея состоит в том, чтобы изменить последовательность объявления переменных, внутреннего расположения хранилищ данных, а также переупорядочивании методов, массивов, определенных полей в структурах и т. д.

Обфускация управления. Позволяет осуществить обфускацию потока управления, то есть последовательности выполнения программного кода. Основывается на использовании непрозрачных предикат, в качестве которых выступают последовательности операций, результат работы которых сложно определить (само понятие "предикат" выражает свойство одного объекта (аргумента), или отношения между несколькими объектами).

Превентивная обфускация. Рассчитана на предотвращение применения злоумышленником деобфускаторов, декомпиляторов и остальных

инструментов деобфускации. Нацелена на использование недостатков, особенностей, присутствующих в наиболее распространенных программных средствах, часто используемых злоумышленниками в процессе деобфускации [4].

2.4. Области применения обфускации

В современное время исследования в области запутывания программного кода проводятся по направлениям, которые очень мало взаимодействуют друг с другом:

- Системное программирование
- Математическая криптография

В первом случае обфускация используется для предотвращения нарушения авторских прав на программный продукт, для его защиты от обратной инженерии, для обеспечения безопасности мобильных агентов в информационных сетях, при применении цифровых водяных знаков. Однако стойкость обфускации в данном случае не гарантируется.

С позиции математической криптографии разработка эффективных алгоритмов позволяет решить целый ряд серьезных вопросов, например, с ее помощью можно преобразовать криптосистему с секретным ключом к криптосистеме с открытым ключом, проводить вычисления над зашифрованными данными, реализовывать системы функционального шифрования, доверенные схемы перешифрования и электронно-цифровой подписи, создавать верифицируемые системы тайного голосования и схемы двойственного шифрования [5].

2.5. Определение обфускатора

Обфускатор O представляет собой «компилятор», который принимает на вход программу P и на выходе выдает обфусцированную программу O(P). Вероятностный алгоритм O является обфускатором, если выполняются следующие три условия:

1. Для любой машины Тьюринга M, O(M) описывает машину Тьюринга, которая вычисляет ту же функцию, что и M;

2. Имея доступ к O(M) злоумышленник не сможет узнать об M больше, чем можно узнать об M, имея к ней доступ как к «черному ящику»;

3. Размер описания и время выполнения O(M) полиномиально зависят от соответствующих величин машины Тьюринга M. То есть должен существовать полином p такой, что для каждой машины Тьюринга M:

$$|O(M)| < p(|M|)$$

где |M| – это размер описания машины Тьюринга M. И если M завершает выполнение через t шагов при некотором входе x, то O(M) при том же входе x завершает выполнение через p(t) шагов.

2.6. «Неразличимая обфускация» программного кода

Рассмотрим специальный вид обфусцирующего преобразования, которое подойдет для любых программ. Такое преобразование называется преобразованием «неразличимой обфускации».

Пусть даны две программы C1 и C2, вычисляющие одну и ту же функцию. При этом при одинаковых значениях входных параметров, данные программы имеют одинаковое выходное значение. Внутри же эти данные могут быть реализованы совершенно по-разному. Таким образом, при преобразовании программного кода посредством «неразличимой обфускации» не существует эффективного алгоритма, который сможет отличить O(C1) от O(C2).

Хотя данный термин был представлен достаточно давно - долгое время никто не знал, как можно построить преобразование подобного типа. И наконец, в 2013 году был представлен способ построения обфусцирующих преобразований такого типа.

2.7. «Различающая обфускация» программного кода

Другим перспективным направлением исследований в данной области является преобразование «различающая обфускация». «Различающий обфускатор» гарантирует, что если нарушитель все-таки различит обфусцированные программы O(C1) и O(C2), то он сможет найти такие значения входных параметров, при которых O(C1) от O(C2) дадут различные выходные значения.

Данные виды преобразований криптографической обфускации являются лучшими из всех возможных для программ в целом.

3. Деобфускация программного кода

Как уже упоминалось, существует процесс, обратный процессу обфускации. Он позволяет вернуть наиболее похожий первоначальный код программы, то есть код до обфускации.

Процесс деобфускации может быть реализован различными способами. Рассмотрим самые распространенные.

В первом случае берутся несколько программ, которые были модифицированы посредством обфускации. Их код может быть визуально разным, однако производить они будут аналогичные действия. Для выявления мусорного кода, добавленного в процессе запутывания, сравниваются кусочки кода этих программ. Затем лишний код просто удаляется.

Во втором случае программный код проверяется на наличие наиболее распространенных конструкций,

применяемых в процессе запутывания. Также в коде выделяются фрагменты, которые никоим образом не связаны с основными выполняемыми программой задачами, то есть обнаружение ненужных участков кода.

4. Язык программирования JavaScript

Авторами статьи разрабатывается обфускатор для защиты программ, написанных на языке Javascript.

JavaScript является объектно-ориентированным языком. Однако используемое в языке прототипирование обуславливает отличия в работе с объектами по сравнению с традиционными класс-ориентированными языками. Кроме того, JavaScript имеет ряд свойств, присущих функциональным языкам — функции как объекты первого класса, объекты как списки, карринг, анонимные функции, замыкания — что придаёт языку дополнительную гибкость.

Существуют как минимум три особенности языка JavaScript, которые делают его поистине уникальным:

- Полная интеграция с HTML/CSS.
- Простые вещи делаются просто.
- Поддерживается всеми распространёнными браузерами и включён по умолчанию

Ввиду того, что этих трёх особенностей одновременно нет больше ни в одной браузерной технологии, JavaScript является самым распространённым средством создания браузерных интерфейсов.

Существует множество методов обфускации JavaScript-программ, но они обладают недостатками и уязвимостями. Поэтому задача совершенствования средств обфускации программ на JavaScript является актуальной.

5. Обзор существующих методов обфускации JavaScript-программ

Все современные методы обфускации используют алгоритм, основанный на последовательном применении нескольких функций преобразования исходного кода.

Схематично данный алгоритм можно изобразить следующим образом (Рис. 1):

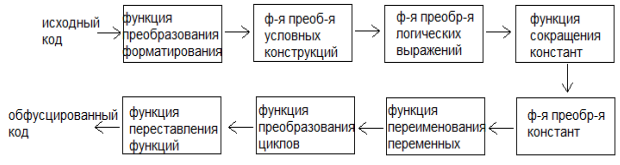

Рис. 1. Алгоритм обфускации

Основные различия между методами состоят во внутренних алгоритмах – алгоритмах функций преобразования.

Рассмотрим это подробнее на примере алгоритма, лежащего в основе функции преобразования логических выражений. В большинстве случаев используется алгоритм, суть которого заключается в том, чтобы вместо исходного оператора использовалось отрицание противоположного оператора с отрицаемыми операндами.

Необходимо отметить, что 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.

Таким образом, на примере алгоритма функции преобразования логических выражений, легко заметить, что существующие методы обфускации не учитывают многие конструкции синтаксиса языка JavaScript последней версии.

6. Анализ существующих инструментов обфускации JavaScript-программ

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) [6].

Самыми популярными инструментами для обфускации кода клиентской части web-приложения являются Google ClosureCompiler и YUI Compressor.

YUI Compressor — компрессор, разработанный Yahoo на языке Java, который гарантирует сохранение работоспособности кода наряду со снижением веса файла. Он начинает с анализа исходного файла JavaScript, чтобы понять, как он структурирован. Затем печатает поток токенов, опуская как можно больше символов пробела и заменяя все локальные символы символом буквы 1 (2 или 3), где бы такая замена ни была подходящей.

Google ClosureCompiler - инструмент по сжатию JavaScript от Google, написан на языке Java. В отличие от YUI Compressor он имеет официальный онлайн-инструмент для сжатия и предоставляет 3 уровня обфускации:

1. Whitespace only удаляет комментарии, лишние разрывы строк и другие ненужные места. Этот уровень обеспечивает наименьший уровень сжатия.

2. Simple— данный вид обфускации похож на предыдущий, но также выполняет обфускацию внутри выражений и функций, включая переименование переменных и параметров функции к более коротким именам.

3. Advanced — последний вид обфускации, похож на «Simple», но добавляет множество более агрессивных глобальных преобразований, чтобы достигнуть самого высокого уровня обфускации.

Для сравнения YUI Compressor и Google ClosureCompiler проведем эксперимент по сжатию кода, отвечающего за случайный вывод на JavaScript:

```
function r_out01() {
        var b=[];
        b[0]='Test-1';
        b[1]='Test-2';
        vari=Math.floor(Math.random()*b.length);
        document.write( b[i] );}
```

В результате работы YUI Compressor на выходе получаем:

```
function r_out01(){var a=[];a[0]="Test-1";a[1]="Test-2";
varc=Math.floor(Math.random()*a.length);document.write(a[c])};
```

Как видно, были удалены переводы строк, лишние пробелы, переменные заменены на другие в алфавитном порядке. В примере были использованы переменные из одного символа, если исходный код будет содержать переменные из нескольких символов (например, слов на транслите), то они также будут минимизированы. При наличии комментариев они также удаляются. Суммарный выигрыш составил 11%.

В эксперименте с Google ClosureCompiler участвует все тот же код. Выбираем уровень обфускации Simple.На выходе получаем:

```
function r_out01(){var a=[];a[0]="Test-1";a[1]="Test-
2";document.write(a[Math.floor(Math.random()*a.length
)])};
```

Суммарный выигрыш составил 12.6%, что выше YUI Compressor на 1,6%. Были удалены пробелы, заменены переменные, отличия в полученном коде заметны. Однако стоит отметить, что оба сервиса по-разному подсчитали оригинальный размер кода.

К сожалению, рассмотренные инструменты имеют существенные недостатки, например, не работают с кодом, содержащим конструкции JS Flow - распространенным в наше время статическим анализатором типов, что обосновывает проводимую разработку обфускатора кода.

7. Проектирование программного продукта

Анализ существующих обфускаторов и выявленные недостатки обосновывают решение разработать собственное веб-приложение для осуществления эффективной обфускации программного кода, написанного на языке JavaScript.

Схема работы с веб-приложением следующая:

- пользователь загружает файл с исходным JavaScript кодом,
- выбирает режимы обфускации,
- затем запускает процесс обфускации,
- после чего ему предлагается сохранить файл с обфусцированным кодом.

В зависимости от выбранных режимов обфускатор использует заложенные в программе запутывающие функции. Режимы обфускации, предлагаемые программой:

- Удаление пробельных символов.
- Удаление однострочных и многострочных комментариев.
- Замена имен идентификаторов.
- Преобразование выражений условного оператора if-else.
- Логическое преобразование.
- Сокращение констант.
- Кодирование чисел.
- Кодирование строк.

Язык программирования JavaScript используется преимущественно для гипертекстовых документов. Однако существует большое число деобфускаторов для гипертекстовых документов как распространяемых свободно, так и коммерческих, что упрощает несанкционированное использование, обратную инженерию и модификацию информации. Для борьбы с этим оправдано применение эффективных алгоритмов запутывания исходного кода.

Рассмотрим алгоритмы, заложенные в упомянутых ранее режимах.

Удаление пробельных символов, однострочных и многострочных комментариев

Данный вид преобразования исходного кода реализуется в JavaScript с помощью регулярных выражений. Работают они через специальный объект RegExp. Кроме того, у строк есть свои методы search, match, replace.

Алгоритм замены имен идентификаторов

Преобразование исходного кода с помощью замены имен идентификаторов (переменных, функций, процедур и т.д.) является одним из самых эффективных алгоритмов, поскольку не требует особых ресурсных затрат и вместе с тем затрудняет понимание программы. Суть алгоритма в том, чтобы привести названия переменных и функций в набор символов, сложный для человеческого восприятия.

Алгоритм преобразования условных конструкций

Идея данного алгоритма состоит в том, чтобы допустимые выражения условного оператора if-else представить в виде тернарного оператора, который выглядит как «условие? значение1: значение2».

Алгоритм логического преобразования

Суть этого преобразования заключается в том, чтобы вместо исходного оператора использовать отрицание противоположного оператора с отрицаемыми операндами.

Алгоритм сокращения констант

Алгоритм преобразования с помощью сокращения констант выглядит следующим образом: найти в коде константы, 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.

Алгоритм кодирования чисел

Преобразование заключается в том, что числа заменяются арифметическими выражениями, после чего представляются в другой системе счисления.

Алгоритм кодирования строк

В данном преобразовании представляем строки в виде конкатенации вызовов различных функций.

8. Заключение

В данной статье рассмотрены различные методы и алгоритмы обфускации, которые можно применять для высокоэффективного преобразования исходного кода программ, написанных на языке JavaScript.

Стоит отметить, что в будущем возможно усовершенствование данных алгоритмов обфускации, в том числе за счет их комплексного применения в заданном программой порядке, который невозможно изменить. Это позволит модифицировать исходный код с максимальной эффективностью, поскольку появятся связи еще и между самими режимами запутывания.

Также в работе приведены результаты анализа существующих инструментов для обфускации программного кода языка JavaScript. В ходе их детального изучения были выявлены присущие им недостатки.

Во-первых, и Google ClosureCompiler, и YUI Compressor написаны на языке Java и запускаются при помощи Java. Это значит, что для использования этих средств разработчики должны обладать хотя бы минимальными знаниями этого языка. Данный аспект может вызвать значительные затруднения, так как фронтэнд-разработчики (основной специализацией которых является JavaScript) будут вынуждены либо устанавливать на свои компьютеры и осваивать Java на минимальном для запуска программ уровне, либо частично передавать ответственность за свою часть приложения на бэкэнд-разработчиков. Все это существенно увеличивает порог вхождения для разработчиков, желающих использовать данные инструменты.

Во-вторых, YUI Compressor поддерживает только старые версии языка JavaScript (ES3 и ES5), а Google ClosureCompiler всего лишь частично поддерживает набор функций версии ES6, что обусловлено недочетами алгоритмов, заложенных в их основе.

Кроме этого, оба рассмотренных программных средства не работают с кодом, содержащим конструкции JS Flow - распространенным в наше время статическим анализатором типов.

Опираясь на проведенное исследование, был предложен собственный обфускатор кода, имеющий в основе эффективные алгоритмы обфускации и удовлетворяющий следующим минимальным требованиям:

1) быть написанным на языке JavaScript (чтобы не увеличивать порог вхождения для потенциальных пользователей);

2) уметь работать со всеми распространенными в наше время версиями языка JavaScript: ES3, ES5 и ES6;

3) уметь работать с кодом, содержащим конструкции статического типизатора JS Flow.

Список используемых источников

1. Варновский Н.П., Захаров В.А., Кузюрин Н.Н., Шокуров А.В. Современное состояние исследований в области обфускации программ: определения стойкости обфускации // Труды Института системного программирования РАН (электронный журнал), том 26, № 3, с. 167-198.
2. Козачок А.В. Комплекс алгоритмов контролируемого разграничения доступа к данным, обеспечивающий защиту от несанкционированного доступа / А.В. Козачок, Л.М. Туан // Системы управления и информационные технологии. – Воронеж, 2015. – № 3(61). – С. 58–61.
3. Никольская К. Ю., Хлестов А. Д. Обфускация и методы защиты программных продуктов. Вестник УрФО «Безопасность в информационной сфере».– 2015.–Том 16.– С.7-10
4. Обфускация и защита программных продуктов.– 2005.–URL: http://citforum.ru/security/articles/obfus/
5. Козачок А. В. Подход к защите файлов документальных форматов от несанкционированного доступа на основе применения неразличимой обфускации программного кода / А. В. Козачок, Л. М. Туан // Доклады ТУСУР. – 2015. – № 4(38). – С. 113–118.
6. Медгаус С.В., Чернышова А.В. Обфускатор программного кода языка JavaScript. Информатика и кибернетика, № 4(6), – Донецк: ДонНТУ, 2016. с. 59 – 66.