УДК 004.056.55

Н.Е. Губенко, канд. техн. наук, доцент,
А.В. Чернышева, ст. преподаватель,
Д.Д. Моргайлов, магистрант
Донецкий национальный технический университет, г. Донецк, Украина
gubenko@cs.dgtu.donetsk.ua, alla@pmi.dgtu.donetsk.ua, kolleganin@yandex.ru

Система обфускации программного кода для языка PHP

Описаны концепции обфускации для защиты программного кода. Приведено формальное определение процесса обфускации. Представлена классификация существующих методов запутывания, выполнен их анализ. Предложен авторский алгоритм лексического анализа. Спроектирована и реализована система обфускации программного кода для языка PHP, показаны результаты ее работы.

Ключевые слова: Защита программных продуктов, обфускация, обратная инженерия, лексический анализ, язык PHP

Введение

В настоящее время защита программных систем от нелегального использования и изучения является актуальной задачей, поскольку зачастую их исходные коды составляют коммерческую тайну или являются «ноу-хау». Кроме того, возможность получения закрытых сведений о внутреннем устройстве программы позволяет злоумышленнику использовать ее уязвимые места в деструктивных целях. Таким образом, с целью продления срока использования пробной версии программы, обхода процедур регистрации и активации либо получения прав доступа более высокого уровня, подсистемы безопасности программного продукта могут быть подвергнуты модификации или удалению.

Законодательство Украины в сфере авторского права обеспечивает право собственности на любые виды компьютерных программ. Однако наличие такой законодательной базы не способно полностью исключить заимствование логических принципов, реализованных в этих программах, архитектуры построения программы и тому подобное, что порождает проблему воспроизведения функциональных возможностей чужих компьютерных программ за короткий срок без нарушения международных соглашений и национального законодательства [1].

Поэтому одним из наиболее действенных на данный момент способов защиты программных продуктов от подобных нарушений является обфускация – выполнение ряда преобразований над исходным кодом программы, затрудняющих процесс обратной инженерии и в то же время полностью сохраняющих семантику программы.

Обфускация должна обеспечивать такой уровень сложности трансформированного кода, чтобы его анализ и изучение стали экономически невыгодными (а физически трудновыполнимыми) по сравнению с разработкой собственного аналога программного продукта.

Целью исследования является программная реализация выбранного подмножества методов обфускации.

Задачами исследования являются анализ существующих методов обфускации и разработка авторского алгоритма лексического анализа.

Концепции обфускации для защиты программного кода

Приведем формальное определение процесса обфускации с позиции защиты программного продукта путем односторонней трансформации исходного кода, исключая возможность восстановления его первоначального вида.

Пусть TR – трансформирующий процесс, а PR1 – исходная программа. Тогда результатом преобразования вида TR(PR1) будет программа PR2, представляющая собой трансформированный код программы PR1. Процесс TR называется процессом обфускации, если выполняются следующие требования [2]:

– код программы PR2 значительно отличается от кода программы PR1 и при этом полностью сохраняет функциональность исходной программы;

– процесс детрансформации и изучения программы PR2 будет крайне сложным, трудоемким и экономически нецелесообразным;

− результаты выполнения трансформации над одним и тем же программным кодом будут различаться.

Обфускация может применяться для демонстрации неочевидных возможностей языка программирования или среды выполнения, оптимизации программы и уменьшения ее размера, затруднения изучения принципов работы программного продукта и их воспроизведения, защиты системы от взлома, создания и защиты вредоносных программ, реализации сетевых атак и сокрытия спама.

Запутывание кода может осуществляться на уровне алгоритма, исходного и ассемблерного текста программы.

Обфускация на высшем уровне осуществляется над исходным кодом и заключается в изменении внешнего вида программы: переформатировании и замене имен идентификаторов.

Запутывание на уровне машинного кода на данный момент недостаточно исследовано и не получило широкой популярности. Оно является труднореализуемым и менее комплексным, требует учета особенностей работы большинства существующих процессоров. Поскольку быстродействие программы при этом значительно снижается, такой вид обфускации следует применять в критичных к безопасности, но не критичных к времени выполнения участках кода.

Простейший способ обфускации машинного кода – вставка в него несуществующих команд и инструкций.

Наиболее популярные на сегодняшний день языки программирования компилируют исходный код в промежуточный (байт-код). Подобное представление содержит достаточно информации для адекватного восстановления исходного текста программы и поэтому также нуждается в защите.

Большинство существующих алгоритмов и методов обфускации могут применяться как на низшем, так и на высшем уровне [2].

Для сохранения размера программы и скорости ее работы целесообразно прибегать к обфускации только наиболее важных фрагментов.

Виды и методы обфускации

В зависимости от способа модификации кода программы, процессы обфускации можно условно разделить на несколько групп, каждая из которых направлена на преобразование конструкций определенного типа [2, 3]:

− лексическая обфускация;
− обфускация данных;
− преобразования потока управления;
− превентивная обфускация.

Лексическая обфускация является наиболее простой и распространенной и предназначена для снижения информативности программного кода с целью затруднения его анализа человеком. Этот вид обфускации предполагает удаление лишних пробелов и отступов, удаление комментариев или изменение их на неинформативные, замену идентификаторов произвольными последовательностями заданной длины из символов определенного набора, удаление отладочной информации, изменение расположения функциональных блоков программы [2].

Переформатирование программы и преобразование комментариев являются наиболее простыми методами обфускации, поскольку не требуют синтаксического и семантического анализа. Для их реализации достаточно лишь разбить исходный код программы на последовательность токенов и выполнить модификацию токенов соответствующих типов. Несмотря на односторонний характер таких преобразований, они не способны значительно затруднить процесс обратной инженерии. Наличие в программе поясняющих комментариев зачастую является лишь рекомендацией, а при условии высокого качества кода и вовсе не обязательно. Восстановить исходное форматирование программы можно без особых временных и материальных затрат с использованием автоматизированных инструментальных средств.

Переименование идентификаторов требует семантического анализа. Для определения типа идентификатора (переменная, функция, имя класса и т.д.) и привязки их к месту объявления и области действия, необходимо выполнить анализ межмодульных связей программной системы. Идентификаторы, принадлежащие внешним библиотекам, изменяться не могут. В противном случае данные библиотеки также должны подвергаться трансформации. Тогда для обеспечения переносимости, набор трансформированных библиотек должен поставляться вместе с программой, что является безусловным недостатком такого подхода.

Обфускация данных нацелена на преобразование структур данных и считается более сложной и продвинутой, чем лексическая. Она включает в себя изменение интерпретации данных определенного типа и срока их использования, преобразование статических (неменяющихся) данных в процедурные, разделение и объединение переменных и массивов, шифрование переменных,

переупорядочение хранилищ данных, изменение иерархии наследования [2].

Преобразования потока управления изменяют граф потока управления одной функции и могут приводить к созданию в программе новых функций. В качестве методов, реализующих данное преобразование, выступают расширение условий цикла, добавление недостижимого и «мертвого» кода, параллелизирование кода, встраивание, извлечение, объединение и клонирование функций, трансформация, развертка и разделение циклов [2].

Анализ программ, которые были подвергнуты преобразованиям данных или потока управления, осуществляется автоматически (без участия человека) либо полуавтоматически (с минимальным участием пользователя) в зависимости от конкретного метода обфускации.

Проектирование и реализация обфускатора для языка PHP

Большинство веб-сайтов в сети Интернет, созданных с использованием CMS (систем управления контентом), имеет множество уязвимостей, которые позволяют удаленному злоумышленнику получить доступ к конфиденциальной информации, расположенной в файловой системе сервера. Поэтому защита PHP-скриптов от копирования и анализа является важной проблемой.

Для осуществления процесса обфускации над исходным кодом программы требуется выделить некоторое подмножество конструкций языка, которые впоследствии будут подвергнуты преобразованиям. Поэтому разработке лексического анализатора непосредственно предшествует описание данных конструкций, а также языка в целом.

Описание синтаксиса и основных возможностей языка PHP присутствует в [4].

В результате анализа существующих методов обфускации для программной реализации были выбраны следующие виды трансформаций:

− преобразование статических литералов в процедурные данные с возможностью кодирования в формат base64;

− преобразование числовых данных в процедурные;

− удаление однострочных и многострочных комментариев или замена их на дезинформирующие;

− удаление лишних пробелов и отступов.

Применение этих преобразований в указанном порядке позволяет значительно ухудшить читабельность программы и не приводит к увеличению ее размера. В то же время обфускация статических строк и числовых значений является обратимой, а также существенно уменьшает быстродействие PHP-скрипта.

Таким образом, разработанная система обфускации реализует данные методы и предоставляет возможность выбора из них некоторого подмножества для осуществления преобразований над исходной PHP-программой.

Основная функциональность системы отражена на диаграмме вариантов использования (рис. 1).

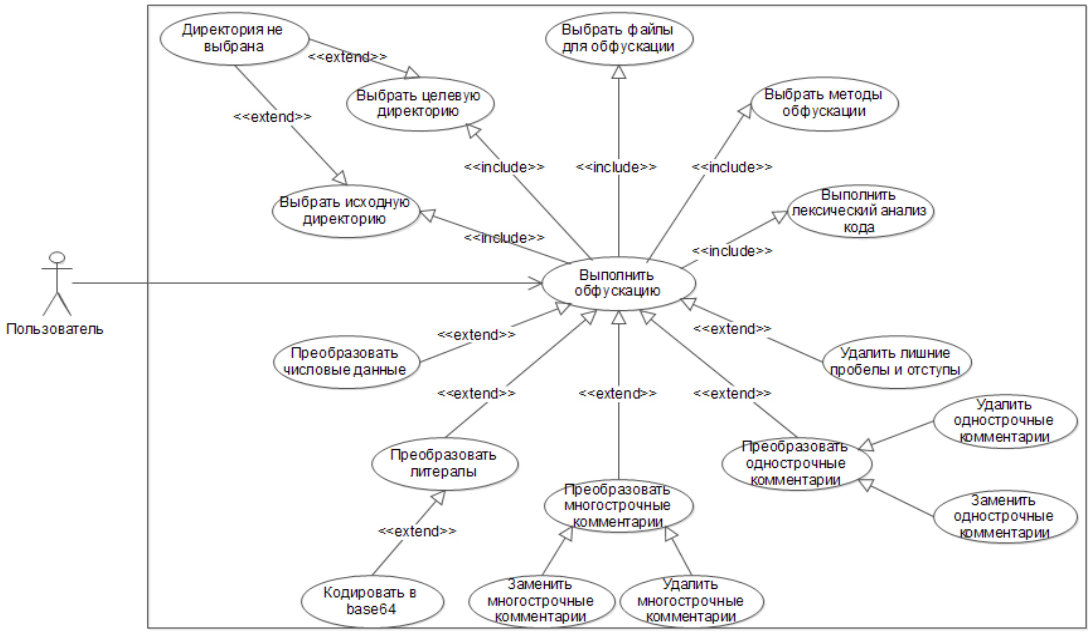

Рисунок 1 – Диаграмма вариантов использования обфускатора

Для программной реализации выбранных методов обфускации необходимо разработать следующие классы (табл. 1).

Диаграммы классов и последовательности, отражающие соответственно структуру и взаимосвязи выделенных классов, приведены на рис. 2 и 3 соответственно.

Представленная диаграмма последовательности описывает основной поток событий прецедента «Выполнить обфускацию» и отражает взаимодействие объектов классов MainForm, FileSearcher, Obfuscator и Scanner.

Таблица 1 – Описание классов системы

Класс	Назначение
MainForm	Класс главной формы. Реализует интерфейс пользователя.
ScannerStates	Перечисление. Описывает состояния лексического анализатора.
TokenTypes	Перечисление. Представляет список возможных типов токенов.
FileSearcher	Возвращает список файлов указанной директории с учетом вложенных каталогов.
RandomGenerator	Генерирует случайные целые неотрицательные числа размерностью 32 бита, не превышающие указанное значение.
Token	Служит для хранения базовой информации о токене. Инкапсулирует тип токена и его значение.
Scanner	Класс лексического анализатора. Осуществляет лексический разбор исходного PHP-кода и хранит представление кода в виде последовательности токенов различного типа.
TokenOperations	Статический класс-библиотека, предоставляющий набор операций для генерации имен идентификаторов различного вида, определения типа указанного токена, а также шифрования строковых значений в формат base64 и поиска интерполируемых переменных в литералах, заключенных в двойные кавычки.
Obfuscator	Непосредственно класс обфускатора. Реализует выбранные методы обфускации и возвращает код преобразованной программы в текстовом представлении.

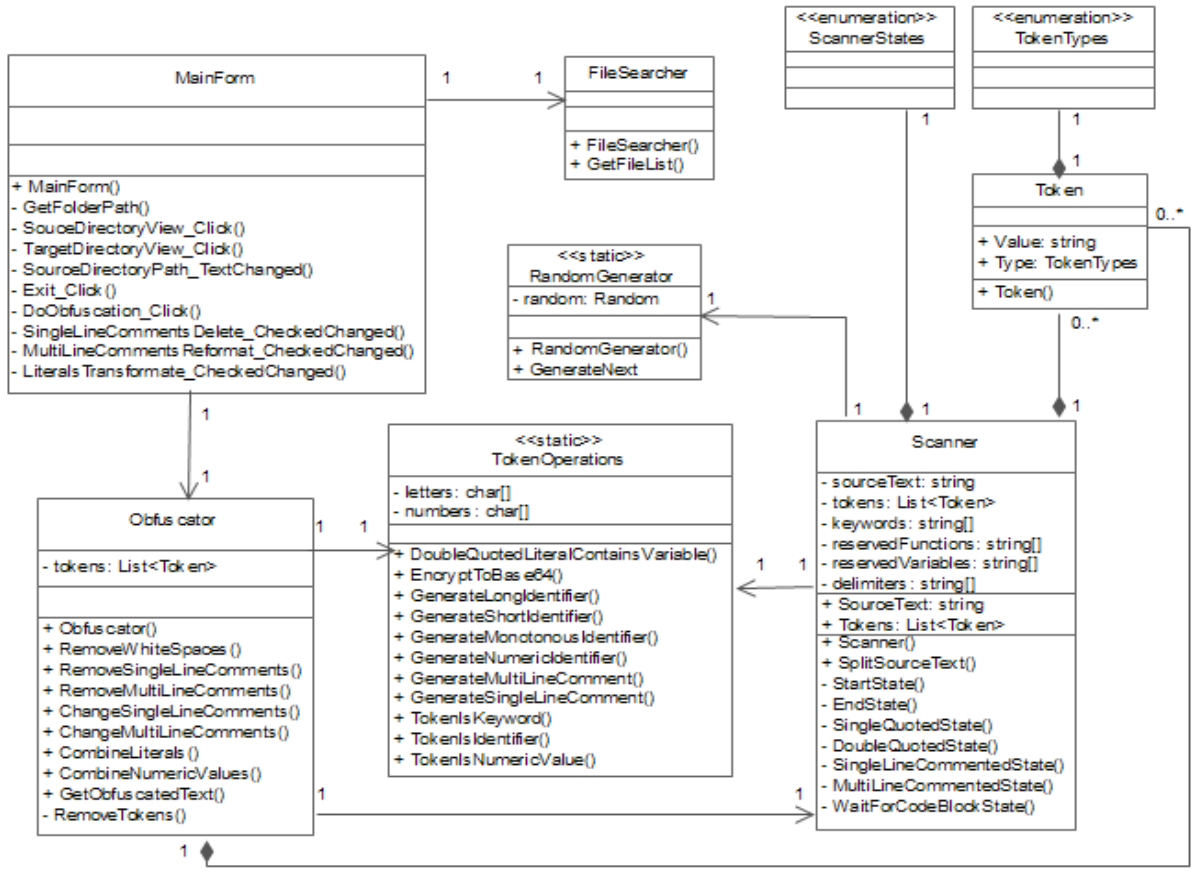

Рисунок 2 – Диаграмма классов обфускатора

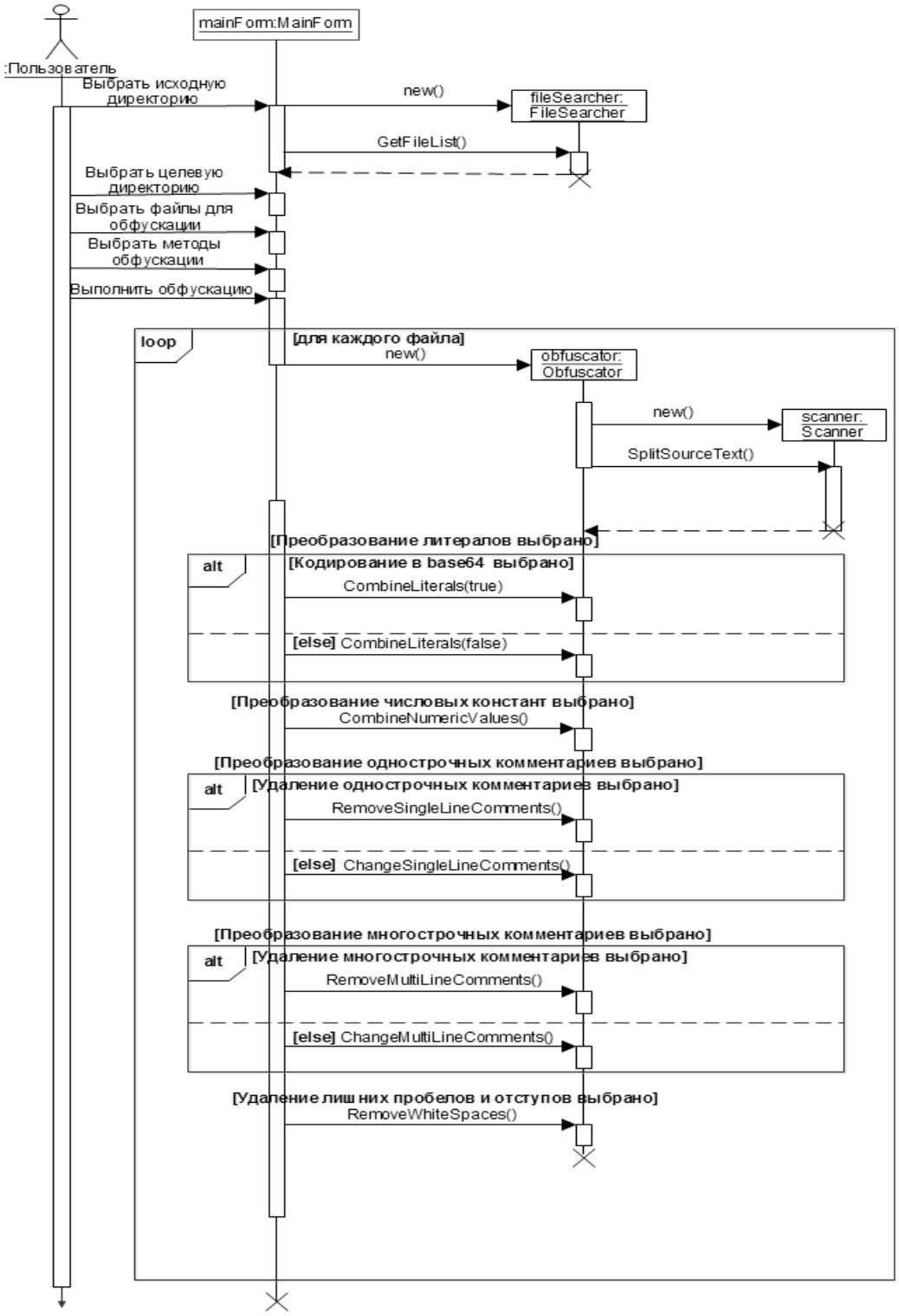

Рисунок 3 – Диаграмма последовательности обфускатора

Этапом, предшествующим реализации выбранных методов обфускации, является разработка алгоритма лексического анализа исходного кода программы.

Основная задача лексического анализатора – разбить входной текст, состоящий из одиночных символов, на последовательность токенов (лексем), т.е. выделить эти слова из непрерывной последовательности символов [5]. Все символы входной последовательности подразделяются на символы, принадлежащие каким-либо лексемам, и символы-разделители. В

определенных случаях между лексемами может и не быть разделителей или наоборот следовать несколько разделителей подряд. Некоторые из них могут быть незначащими (например, лишние пробелы и отступы).

Лексемы делятся на следующие классы:

- идентификатор (Identifier);
- ключевое слово (Keyword);
- литерал в одинарных кавычках (SingleQuotedLiteral);
- литерал в двойных кавычках (DoubleQuotedLiteral);
- числовое значение (NumericValue);
- однострочный комментарий (SingleLineComment);
- многострочный комментарий (MultiLineComment);
- разделитель (Delimiter);
- текст, находящийся за пределом блока кода (NotPHPCode);
- токены, не вошедшие в перечисленные классы (Other).

Сначала выделяется отдельная лексема, заключенная между символами-разделителями. Выделенная лексема проверяется на принадлежность какому-либо из перечисленных классов, а затем ее значение и признак соответствующего класса сохраняются (добавляются в список лексем). Например, ключевые слова распознаются путем выделения идентификатора и последующей проверки его на принадлежность множеству ключевых слов.

Работа лексического анализатора задается некоторым конечным автоматом. Однако, непосредственное описание конечного автомата неудобно с практической точки зрения. Во-первых, такое представление порождает огромное число состояний лексического анализатора. Вторым важным недостатком является возможность анализировать только один символ входного потока, в то время как часто для выделения и точной идентификации токена требуется «заглянуть вперед» во входной поток. Поэтому в данной работе для разбиения исходного текста на токены будет применяться авторский алгоритм с «заглядыванием вперед».

В зависимости от обрабатываемой в данной момент лексемы, лексический анализатор может находиться в одном из шести состояний (табл. 2).

Таблица 2 – Описание состояний лексического анализатора

Состояние	Характеристика
Start	Осуществляется проверка, являются ли очередные символы входного потока (т.е. начальные символы выделяемой лексемы) началом литерала в одинарных (') или двойных (") кавычках либо однострочного (// или #) или многострочного (/*) комментария. Если совпадение обнаружено, сканер переходит в соответствующее состояние для выделения лексемы данного типа. Иначе выполняется переход в конечное (End) состояние, где проверяется предположение о принадлежности символа к классу разделителей. Если же текущий символ не является первым в выделяемой лексеме, то анализируются следующие символы входного потока.
SingleQuoted	Выделяется литерал, заключенный в одинарные кавычки. Допускается использование экранированного символа одинарной кавычки в выделяемой строке.
DoubleQuoted	Выделяется литерал, заключенный в двойные кавычки. Двойные кавычки, используемые в теле строки, должны быть экранированы.
SingleLineCommented	Выделяется однострочный комментарий. Об окончании комментария сигнализирует конец строки, блока кода или файла.
MultiLineCommented	Выделяется многострочный комментарий. Комментарий должен заканчиваться символами */
End	Если во входном потоке обнаружен символ-разделитель, выделяемая лексема считается завершенной и определяется ее тип. Информация о лексеме (ее тип и значение) вместе с разделяющим символом добавляется в список токенов. Если текущий блок кода завершен, осуществляется переход в состояние WaitForCodeBlock.
WaitForCodeBlock	Ожидается начало следующего блока кода. Текст, находящийся вне блока кода, сохраняется с пометкой NotPHPCode.

Таким образом, переходы между состояниями определяются появлением во входном потоке специфичной для каждого состояния последовательности символов.

Диаграмма состояний лексического анализатора показана на рис. 4.

Описание условий переходов приведено в табл. 3.

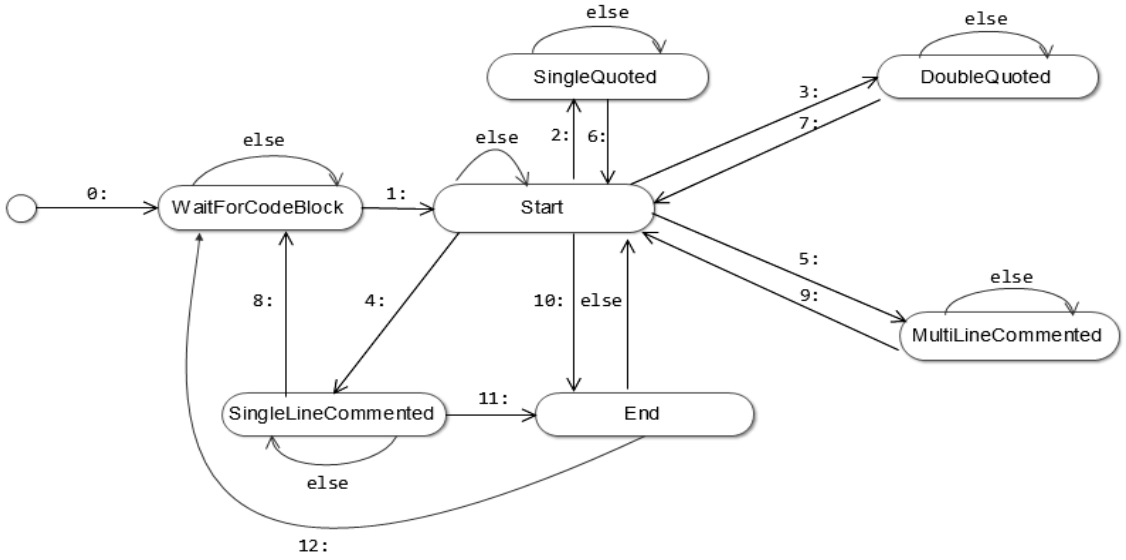

Рисунок 4 – Диаграмма состояний лексического анализатора

Таблица 3 – Описание условий переходов

Переход	Описание
0:	Ожидание первого блока кода – начальное состояние лексического анализатора
1:	Достигнут очередной блок кода
2:	Выделяемая лексема – литерал в одинарных кавычках
3:	Выделяемая лексема – литерал в двойных кавычках
4:	Выделяемая лексема – однострочный комментарий
5:	Выделяемая лексема – многострочный комментарий
6:	Литерал в одинарных кавычках завершен
7:	Литерал в двойных кавычках завершен
8:	Однострочный комментарий и текущий блок кода завершены
9:	Многострочный комментарий завершен
10:	Текущий символ не является символом начала литерала или комментария
11:	Однострочный комментарий и текущая строка кода завершены
12:	Текущий блок кода завершен

Удаление лишних пробелов и отступов осуществляется следующим образом. На каждой итерации прохода по всем токенам программы символы табуляции, перехода на новую строку и возврата каретки заменяются пробелами. Символ перехода на новую строку не заменяется пробелом, если предыдущим токеном был однострочный комментарий. Затем текущий токен удаляется, если он является пробелом и находится между двумя другими разделителями.

Однострочные и многострочные комментарии, находящиеся в теле PHP-программы, удаляются или заменяются неинформативной последовательностью букв латинского алфавита. Структура комментариев (пробелы, табуляции, переходы на новую строку и возврат каретки) при замене сохраняется.

Для преобразования литералов в процедурный вид сначала осуществляется просмотр всех токенов, представляющих собой строки в одинарных или двойных кавычках. Последние дополнительно проверяются на наличие в них интерполируемых переменных или выражений, содержащих переменные. Затем формируется функция, содержащая список статических литералов и возвращающая нужное значение по заданному индексу. В программе литералы заменяются вызовом данной функции с соответствующим аргументом.

Преобразование числовых данных производится аналогично предыдущему и в комбинации с ним дает хорошие результаты. Абсолютные значения числовых констант выделяются в отдельную функцию, помещенную в первый блок кода программы.

Заключение

Описаны концепции обфускации для защиты программного кода. Приведено формальное определение процесса обфускации. Представлена классификация существующих методов запутывания, выполнен их анализ. Предложен авторский алгоритм лексического анализа. Спроектирована и реализована система обфускации программного кода для языка PHP. Показаны результаты работы системы.

Непосредственное описание лексического анализатора конечным автоматом порождает большое число состояний и позволяет анализировать только один символ входного потока, в то же время часто для выделения и точной идентификации токена требуется «заглянуть вперед» во входной поток.

Преобразование статических строк и числовых значений позволяет существенно ухудшить читабельность программного кода, однако уменьшает быстродействие программы. Кодирование литералов в формат base64 намного снижает скорость работы программы.

Недостаток большинства запутывающих преобразований заключается в их обратимости, т.е. в возможности получения исходного вида программы с помощью специальных средств (деобфускаторов).

Изменение последовательности применяемых методов запутывания может усилить получаемый от обфускации эффект.

Научная значимость данной работы состоит в анализе существующих методов обфускации и разработке авторского алгоритма лексического анализа.

Практическая ценность выполненных исследований состоит в создании системы обфускации программного кода для языка PHP.

Дальнейшие исследования будут направлены на создание нового метода обфускации, который позволит значительно снизить информативность программного кода и будет устойчивым к процессу деобфускации.

Список литературы

1. Умяров Н.Х. Анализ и выбор методов защиты программного продукта от копирования с использованием обфускации / Н.Х. Умяров, Н.Е. Губенко // Материалы VI международной научно-технической конференции [Информатика и компьютерные технологии-2011], (Донецк 22-23 ноября 2011). – Донецк: ДонНТУ, 2011. – С. 291-295.
2. Обфускация и защита программных продуктов [Электронный ресурс]. - Режим доступа: http://citforum.ru/security/articles/obfus/
3. Чернов А. В. Анализ запутывающих преобразований программ [Электронный ресурс] / А.В. Чернов. - Режим доступа: http://citforum.ru/security/articles/analysis/
4. PHP [Электронный ресурс]. - Режим доступа: http://php.su/

Надійшла до редакції 10.10.2012

Н.Є. ГУБЕНКО, А.В. ЧЕРНИШОВА, Д.Д. МОРГАЙЛОВ
Донецький національний технічний університет

N.Ye. GUBENKO, A.V. CHERNYSHOVA, D.D. MORGAILOV
Donetsk National Technical University

Система обфускації програмного коду для мови PHP

Описано концепції обфускації для захисту програмного коду. Наведено формальне визначення процесу обфускації. Представлена класифікація існуючих методів заплутування, виконано їх аналіз. Запропоновано авторський алгоритм лексичного аналізу. Спроектовано та реалізовано систему обфускації програмного коду для мови PHP, показані результати її роботи.

Ключові слова: захист програмних продуктів, обфускація, зворотна інженерія, лексичний аналіз, мова PHP

PHP Code Obfuscator

The paper describes obfuscation concepts for code protection. The formal definition of obfuscation process is given. The existing obfuscation methods are represented, the analysis of them is done. The author's code parsing algorithm is offered. PHP code obfuscator is designed and implemented, the results of its work are shown.

Keywords: software protection, obfuscation, reverse engineering, code parsing, PHP language