УДК 004.054

А.Л. Ершов

ИДЕНТИФИКАЦИЯ АЛГОРИТМОВ ПРЕОБРАЗОВАНИЯ ДАННЫХ В ИСПОЛНЯЕМЫХ МОДУЛЯХ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Ершов А.Л. **Идентификация алгоритмов преобразований данных в исполняемых модулях программного обеспечения.**

Аннотация. Стремительное развитие информационных технологий в последние десятилетия стало причиной значительного увеличения объема исходных текстов программного обеспечения, а также его сложности. Данный факт обусловливает высокую сложность анализа программного обеспечения с целью понимания логики его функционирования, необходимость выполнения которого является важным моментом при проведении судебной компьютерно - технической экспертизы. В статье раскрывается один из подходов к автоматизации процесса идентификации стандартизированных алгоритмов преобразования данных в исполняемых модулях в условиях отсутствия исходных текстов за счет учета их внутренних информационных связей с целью упрощения понимания программ.

Ключевые слова: идентификация алгоритмов, исполняемый модуль, программное обеспечение, линейный блок, слайсинг, информационный граф.

Ershov A.L. **Identification of Data Conversion Algorithms in Executable Software Modules.**

Abstract. The rapid development of information technology in recent decades has led to a substantial increase in the source code of software, as well as its complexity. This fact points to the high complexity of the assessment of its compliance with the requirements of information security. An important point in this is to ensure reliability as the hardware component and software. Ensuring confidence in the software tools are often carried out in the absence, for whatever reasons of their source. The article deals with an approach to automate the identification of standardized algorithms for data conversion in accordance with their internal information links.

Keywords: identification algorithms, executable module, software, linear unit, slicing, information graph.

1. Введение. В настоящее время основу большинства современных российских информационно-телекоммуникационных систем составляют программно-аппаратные средства импортного производства, которые, как правило, не имеют в своем составе полного набора исходных текстов программного обеспечения. Кроме того, зачастую и отечественные программисты используют в составе своих продуктов модули так называемых "сторонних" разработчиков, которые по тем или иным причинам также не предоставляют исходные тексты своих программ.

Указанная ситуация зачастую существенным образом осложняет проведение судебной компьютерно-технической экспертизы, а точнее, одной из её составляющих – программно-компьютерной экспертизы (ПКЭ). ПКЭ предназначена для осуществления экспертного исследования программного обеспечения. Её основными целями являются изучение функционального предназначения, характеристики реализуемых требований, алгоритма и структурных особенностей, текущего

состояния представленного на исследование программного обеспечения компьютерной системы [1]. Одними из основных вопросов, на которые должна дать ответ ПКЭ, являются следующие [1]:

1. Какое общее функциональное предназначение имеет программное средство?
2. Имеются ли на носителях информации программные средства для реализации определенной функциональной задачи?
3. Используется ли данное программное средство для решения определенной функциональной задачи?
4. Имеет ли программное средство защитные возможности (программные, аппаратно-программные) от несанкционированного доступа и копирования?
5. Каким образом организованы защитные возможности программного средства?
6. Каков общий алгоритм данного программного средства?
7. Имеются ли в программном средстве враждебные функции, которые влекут уничтожение, блокирование, модификацию либо копирование информации, нарушение работы компьютерной системы?

Другими словами, ПКЭ подразумевает, в той или иной степени, проведение анализа программного обеспечения с целью понимания логики его функционирования. Одной из особенностей подобного анализа является необходимость анализа программного обеспечения, имея в наличии только машинный код. Важная роль для понимания логики функционирования программного обеспечения в указанных условиях отводится задаче идентификации в исполняемых модулях стандартизированных алгоритмов преобразования данных.

Сложность решения задачи идентификации алгоритмов в исполняемых модулях определена "однонаправленностью" процесса формирования машинного кода (рисунок 1) [2].

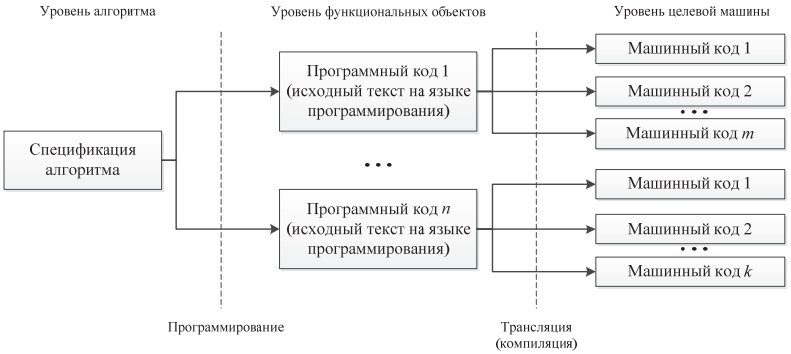

Рис. 1. Этапы формирования машинного кода исполняемого модуля

Программист, имея в качестве исходных данных алгоритм, представленный в виде блок -схемы или словесного описания, может написать программу на высокоуровневом языке программирования, его реализующую, в общем случае, множеством различных способов (уровень функциональных объектов на рисунке 1). В связи с этим машинный код, реализующий один и тот же алгоритм, будет отличаться в зависимости от исходного текста программы. Более того, даже в случае идентичности исходных текстов программы, машинные коды, реализующие их, могут отличаться друг от друга, т.к. они также зависят и от используемого компилятора и его настроек. В частности такими настройками могут быть необходимость оптимизации кода с целью минимизации его объема, скорости выполнения или вообще не использовать оптимизацию и т.д. (уровень целевой машины на рисунке 1). Таким образом, возможные варианты машинного кода, реализующего один и тот же алгоритм, могут значительно отличаться друг от друга (рисунок 2).

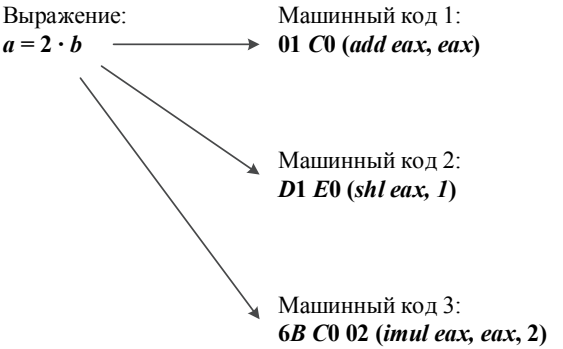

Рис. 2. Варианты машинного кода, реализующего одно и то же математическое выражение

2. Анализ известных способов решения задачи идентификации алгоритмов преобразования данных в исполняемых модулях программного обеспечения. В настоящее время для идентификации стандартизированных алгоритмов преобразования данных в исполняемых модулях используют набор характерных признаков программного или машинного кода, их реализующих. В основном, в качестве таких признаков принимают либо сигнатуры [3, 4], либо статистические характеристики машинного кода [5, 6]. Встречаются также подходы к идентификации алгоритмов с использованием их структурных особенностей [7–11], а также метода "черного ящика" [12].

В качестве сигнатур, например, для криптографических алгоритмов, могут выступать константы или таблицы подстановок, которые прописаны в стандарте на сам алгоритм. Сигнатурами также могут быть последовательности байт машинного кода [4], инструкций процессора или совокупностей инструкций и их операндов-констант [5] и т.д. Использование сигнатур позволяет быстро и достаточно точно идентифицировать известный алгоритм, но это применимо только в случае, если для него возможно их создание. Также недостатком данного подхода является невозможность в ряде случаев определить "границы" и конкретное местонахождение реализации алгоритма, так как обнаружение сигнатуры не всегда позволяет обнаружить саму реализацию алгоритма. Например, при использовании в качестве сигнатуры константы (таблицы констант) криптографического алгоритма, обнаружение её положения в исполняемом модуле не всегда позволяет обнаружить сам исполняемый код, реализующий алгоритм.

В качестве статистических характеристик, также используемых, например, для идентификации криптографических алгоритмов, можно использовать частотность появления тех или иных инструкций процессора (или их последовательностей) на участке кода определенной длины. Дополнительно могут рассматриваться изменение энтропии выходных значений по отношению к энтропии входных [13] и т.д. Оценка статистических характеристик машинного кода достаточно успешно используется для определения наличия в машинном коде элементов криптографических алгоритмов, т.к. в коде, их реализующем, гораздо чаще, чем в обычном, встречаются арифметические и битовые инструкции [14]. В то же время, данный подход зачастую приводит к появлению большого количества ложно - положительных результатов при идентификации криптографических алгоритмов, т.е. он позволяет предположить присутствие в машинном коде некоторого криптографического примитива, но определить, к какому конкретно алгоритму он принадлежит, не всегда возможно. В дополнение к этому, существуют определенные сложности в определении длины окна, в котором необходимо вычислять частоты появления инструкций.

Кроме описанных выше, в [7] описывается подход к идентификации стандартизированных алгоритмов преобразования данных с применением элементов морфологического анализа. Суть данного подхода состоит в сравнении фрагментов графов потока управления функциональных объектов дизассемблированного листинга исполняемого модуля с фрагментами графов потока управления известных алгоритмов (шаблонами). Основным недостатком подобного подхода является большое число ложно - положительных результатов в случае

«простых» графов потоков управления, т.е. когда в графе либо отсутствуют ветвления потока управления, либо их количество невелико (примером алгоритма с подобным графом потока управления является криптографический алгоритм *Rijndael*).

В статье [8] авторы описывают использование методов синтаксического анализа для распознавания файловых вирусов. В основе работы лежит предположение о том, что вероятности порождения цепочек машинных команд незараженного и вредоносного кода соответствующими (заранее полученными на этапе обучения) стохастическими грамматиками будут отличаться. Т.е., имея в наличии восстановленные (на этапе обучения) стохастические грамматики, порождающие незараженный и вредоносный машинный код, можно определить принадлежность машинного кода к соответствующему классу (вредоносному или незараженному). Подобный подход можно использовать и для идентификации алгоритмов преобразования данных, однако, основным его недостатком для рассматриваемой задачи является необходимость наличия большого (порядка нескольких тысяч) количества прецедентов на этапе обучения, что в реальных условиях обеспечить практически невозможно.

Анализ вышеописанных результатов исследований, проводимых в данном направлении, а также приведенных в [5] показал, что существующие решения задачи идентификации алгоритмов преобразования данных в исполняемых модулях имеют существенный недостаток, заключающийся в отсутствии анализа информационных связей между аргументами, внутренними переменными и результатами алгоритмов, что и приводит к достаточно высоким значениям вероятностей ошибок первого и второго рода при идентификации. В связи с этим, для построения системы идентификации алгоритмов преобразования данных, предварительно необходимо разработать модель, которая бы учитывала их внутренние информационные связи.

3. Аналитическая модель алгоритма преобразования данных. По результатам анализа этапов формирования исполняемых модулей (рисунок 1) можно сделать вывод, что единственным инвариантом в этой последовательности является спецификация алгоритма. Данный факт послужил основой для синтеза системы их идентификации на основе использования модели, построенной с учетом структурных свойств алгоритмов. В качестве таких свойств первоначально были выбраны:

1. Количество входных аргументов.
2. Распределение входных аргументов по операторам.
3. Количество используемых констант.

4. Минимально необходимый объем памяти (количество внутренних переменных).

5. Число операторов промежуточного представления алгоритма в виде трехадресного кода.

6. Число операторов промежуточного представления алгоритма в SSA – форме.

7. Количество циклов.

8. Количество значимых операторов (операторов, непосредственно влияющих на результат).

Исходя из невозможности предсказать, каким именно образом программист реализует исходный алгоритм на языке программирования, введем ограничение: будем предполагать, что программист не будет целенаправленно реализовывать несколько алгоритмов преобразования данных в одной программе (или в одном функциональном объекте) Учитывая данное допущение, можно выделить среди перечисленных выше характеристик те, которые будут устойчивы к изменению параметров процесса формирования исполняемых модулей. В связи с тем, что структура исходного текста программы, реализующей алгоритм, может, не смотря на введенное ограничение, меняться, имеет смысл рассмотреть исходный алгоритм, как совокупность составляющих его линейных частей (блоков). В этом случае перечисленные характеристики необходимо рассматривать применительно к линейным блокам исходного алгоритма. Таким образом, распределение входных аргументов по операторам линейного блока, также как и количество операторов в промежуточных представлениях линейных блоков в качестве устойчивых признаков алгоритма не подходят, т.к. данные характеристики кода исполняемого модуля зависят от используемого компилятора и его настроек. Количество циклов при рассмотрении алгоритма в виде совокупности линейных блоков не имеет смысла само по себе. С другой стороны, не смотря на то, что количество входных аргументов, используемых констант и количество значимых операторов линейных блоков в исполняемом модуле также зависят от используемого компилятора и его настроек, их минимальные значения все равно остаются неизменными и в связи с этим их можно использовать в качестве признаков для идентификации алгоритмов преобразования данных. По такому же принципу можно использовать и минимально необходимое количество внутренних переменных линейного блока.

Модель алгоритмов, построенная только на основе перечисленных признаков, не отражает особенности их внутренних информационных связей. Проведенный анализ показал, что наиболее эффектив-

ным способом представления внутренних информационных связей алгоритма является информационный граф [15], который, в случае линейности соответствующих ему операторных схем, является инвариантом относительно всех других схем, описывающих тот же алгоритм. Но это справедливо только тогда, когда ни одна из дуг исходного информационного графа не может быть удалена без нарушения корректности вычисления результата алгоритмом. Исходя из этого, в качестве дополнительного признака алгоритма целесообразно использовать информационный граф, построенный для его линейных блоков, оптимизированных с целью минимизации используемой памяти. Подводя итог вышесказанному, модель алгоритма предлагается представить в виде множества векторов признаков его линейных блоков:

$$M = \{< a, v, s, c, I >_n\}, n = \overline{1, N}, \quad (1)$$

где: a – количество входных аргументов;
v – минимальное количество внутренних переменных, необходимое для выполнения линейного блока алгоритма;
s – минимальное количество значимых операторов линейного блока алгоритма;
c – количество используемых констант в линейном блоке алгоритма;
I – информационный граф линейной части алгоритма;
N – количество линейных блоков.

4. Модель системы идентификации стандартизированных алгоритмов преобразования данных. Совокупность вышеописанных особенностей формирования машинного кода и модели алгоритма позволила разработать систему идентификации стандартизированных алгоритмов преобразования данных (рисунок 3).

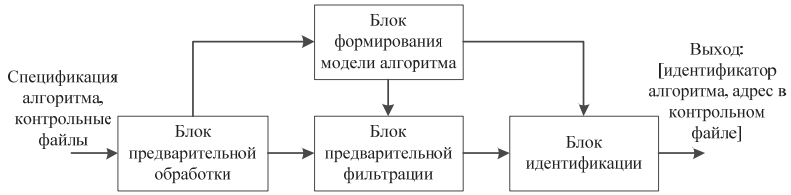

Рис. 3. Структурная схема системы идентификации стандартизированных алгоритмов преобразования данных

Функционирование системы происходит в 2 этапа. На первом этапе на вход блока предварительной обработки поступает спецификация известного алгоритма преобразования данных. В данном блоке эксперт приводит описание алгоритма в единую форму его промежуточного представления (трехадресный код, *SSA* – форма и т.д.), что и является

выходом блока на данном этапе. Затем полученное промежуточное представления алгоритма поступает на блок формирования его модели, где выполняются оптимизирующие преобразования и выделяются линейные блоки. По полученным результатам определяются параметры модели исходного алгоритма, представленной в выражении (1).

На втором этапе на вход блока предварительной обработки поступает исполняемый модуль (контрольный файл). В данном блоке записывается трасса его выполнения. Трасса записывается с помощью системы динамического анализа бинарного кода *Pin* (разработчик – компания *Intel*) и специально разработанного для этого модуля расширения системы *Pin*. Из полученной трассы выделяются линейные блоки ("*basic block*" в трактовке *Pin*). Далее, в каждом из линейных блоков определяется наличие "развернутых" циклов и, при их наличии, линейный блок заменяется телом выявленного цикла (первой его итерацией) или их совокупностью. После этого определяются входные аргументы для каждого из линейных блоков. За входные аргументы принимаются операнды, читаемые первый раз операторами линейного блока при его выполнении. Затем, над каждым из линейных блоков выполняется операция прямого "разреженного" слайсинга [16] (критерием слайсинга является каждый из входных аргументов линейного блока). Данная операция оставляет в составе линейных блоков только те операторы, которые непосредственно изменяют значения, связанные по информации с одним из входных аргументов, и удаляет остальные. Далее по полученным результатам определяются остальные значимые признаки линейных блоков, перечисленные ранее для модели алгоритма (v, s, c, I). Выходом блока предварительной обработки на втором этапе является множество моделей линейных блоков трассы выполнения контрольного файла. Полученное множество моделей линейных блоков поступает на вход блока предварительной фильтрации, который пропускает на свой выход только те линейные блоки, модели которых удовлетворяют выражению (2):

$$L_{out} = L_{in} | (a_{in} \geq a_m \wedge v_{in} \geq v_m \wedge s_{in} \geq s_m \wedge c_{in} \geq c_m), \qquad (2)$$

где: L_{out} – линейные блоки, прошедшие на выход блока предварительной фильтрации;

L_{in} – линейные блоки, поступающие на вход блока предварительной фильтрации;

$a_{in}, v_{in}, s_{in}, c_{in}$ – соответствующие параметры модели линейного блока трассы выполнения контрольного файла;

a_m, v_m, s_m, c_m – соответствующие параметры модели известного алгоритма преобразования данных.

С выхода блока предварительной фильтрации прошедшие на его выход линейные блоки поступают на вход блока идентификации, в котором происходит проверка, является ли один из информационных графов линейных блоков модели известного алгоритма подграфом информационного графа модели линейного блока из трассы выполнения контрольного файла. В случае положительного решения считается, что данный линейный блок принадлежит программному субъекту, реализующему известный алгоритм преобразования данных. Идентификатор алгоритма вместе с адресом линейного блока являются выходом системы.

5. Заключение. Предложенная система является предварительным решением задачи идентификации алгоритмов преобразования данных. Для оценки её эффективности была разработана её программная реализация. Основой программной реализации системы является система динамического анализа бинарного кода *Pin* компании *Intel*. Автором был разработан модуль расширения комплекса *Pin*, который реализует описанный в статье механизм идентификации. Предварительные результаты проведенных экспериментов показали, что вероятность принятия системой правильного решения составляет значение не меньше 0,96 при точности оценки 0,04 и её надежности 0,95. В ходе эксперимента проводилась идентификация алгоритма расчета криптографической хэш - функции *SHA* – 256 [17] в 20 файлах различного назначения как содержащих, так и не содержащих в себе реализацию искомого алгоритма (системные библиотеки, офисные и системные приложения, "хэш – калькуляторы" и т.д.). Вместе с этим, в ходе проведения экспериментов был выявлен и основной недостаток предлагаемой системы идентификации, а именно – зависимость качества идентификации от значений параметров моделей известных алгоритмов. В настоящее время ведется работа по уточнению граничных условий применимости данной системы на практике. В дальнейшем также необходимо проработать процедуру принятия решения в случае отнесения одного и того же линейного блока к нескольким алгоритмам, линейных блоков одного функционального объекта к разным алгоритмам, а также в других неопределенных ситуациях. Несмотря на это, полученные уже на данном этапе результаты позволяют снизить временные затраты на выполнение поиска интересующего алгоритма преобразования данных в исполняемых модулях программного обеспечения в условиях отсутствия исходных текстов.

Литература

1. *Российская Е.Р., Галяшина Е.И.* Настольная книга судьи: судебная экспертиза // М.: Проспект. 2010.

2. *Долгова К.Н., Чернов А.В.* О некоторых задачах обратной инженерии // Труды Института системного программирования РАН. 2008. Т. 15. С. 119–134.
3. *Zhao R. et al.* Detection and analysis of cryptographic data inside software // Information Security. NY: Springer-Verlag Berlin Heidelberg. 2011. vol. 1. no. 1. pp. 182–196.
4. IDA F.L.I.R.T. Technology: In-Depth. URL: https://www.hex-rays.com/products/ida/tech/flirt/in_depth.shtml (дата обращения: 28.05.2015)
5. *Gröbert F.* Automatic Identification of Cryptographic Primitives in Software // Ruhr-University Bochum. 2010. 115 p.
6. *Быков А.В.* Поиск реализаций криптографических алгоритмов в дизассемблированном программном обеспечении // Инновации в информационно-аналитических системах: сб. научн. трудов. Вып. 2. 2011. С. 56–64.
7. *Thierry A.* Recognition of binary patterns by Morphological analysis // Reverse Engineering Conference. 2012.
8. *Мацкевич А. Г., Козачок В. И.* Математическая модель системы стохастического структурного распознавания файловых вирусов // Безопасность информационных технологий. 2007. № 3. С. 44–49.
9. *Булычев П.Е.* Алгоритмы вычисления отношений подобия в задачах верификации и реструктуризации программ // М.: МГУ им. М.В. Ломоносова. 2010. 169с.
10. *Ахин М.Х., Ицыксон В.М.* Слайсинг над деревьями: метод обнаружения разорванных и переплетенных клонов в исходном коде программного обеспечения // Моделирование и анализ информационных систем. 2012. Т. 19. № 6. С. 69–78.
11. *Подымов В.В.* Быстрые алгоритмы проверки эквивалентности программ в моделях с полугрупповой семантикой // М.: МГУ им. М.В. Ломоносова. 2014. 164с.
12. *Calvet J.* Cryptographic function identification in obfuscated binary programs // Reverse Engineering Conference. 2012. URL: https://recon.cx/2012/schedule/attachments/46_Joan_CryptographicFunctionIdentification.pdf (дата обращения: 28/05/2015).
13. *Lutz. N.* Towards Revealing Attackers' Intent by Automatically Decrypting Network Traffic // Fritz Kutter Fonds. 2008. URL: http://www.kutter-fonds.ethz.ch/App_Themes/default/datalinks/NoeLutz-08.pdf (дата обращения: 28.05.2015).
14. *Wang Z., Jiang X., Cui W., Wang X.* ReFormat: Automatic Reverse Engineering of Encrypted Messages // Proceedings of the 14th European conference on Research in computer security (ESORICS'09). 2009. pp. 200–215.
15. *Ершов А.П.* Введение в теоретическое программирование (беседы о методе) // М.: Издательство «Наука». 1977. 288 с.
16. *Sridharan M., Fink S. J., Bodík R.* Thin Slicing // ACM SIGPLAN Notices. 2007. vol. 42. no. 6. pp. 112–122.
17. FIPS PUB 180 – 4. Security Hash Standard (SHS) // NIST. Computer Security Resource Center. 2012. URL: http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf (дата обращения: 28.05.2015).

References

1. Rossijskaja E.R., Galjashina E.I. *Nastol'naja kniga sud'i: sudebnaja jekspertiza*. [Handbook of the judge: forensic examination]. Moscow: Prospekt. 2010. 458 p. (In Russ.).
2. Dolgova K.N., Chernov A.V. [About some problems of reverse engineering]. *Trudy Instituta sistemnogo programmirovanija RAN – Proceedings of Institute for system programming of RAS*. Moscow. 2008. vol. 15. pp. 119–134. (In Russ.).
3. Zhao R. et al. Detection and analysis of cryptographic data inside software. *Information Security*. NY: Springer-Verlag Berlin Heidelberg. 2011. vol. 1. no. 1. pp. 182–196.

4. IDA F.L.I.R.T. Technology: In-Depth. Available at: https://www.hex-rays.com/products/ida/tech/flirt/in_depth.shtml (accessed 28.05.2015).
5. Gröbert F. Automatic Identification of Cryptographic Primitives in Software. Ruhr-University Bochum. 2010. 115 p.
6. Bykov A.V. [Searching for implementations of cryptographic algorithms in disassembly software]. *Innovacii v informacionno-analiticheskih sistemah: sb. nauchn. trudov* [Innovations in information-analytical systems: Collected papers]. Kursk: Naukom. 2011. vol. 2. pp. 56–64 (In Russ.).
7. Thierry A. Recognition of binary patterns by Morphological analysis. Reverse Engineering Conference. 2012.
8. Mackevich A. G., Kozachok V. I. [Mathematical model of stochastic structural recognition file viruses]. *Bezopasnost' informacionnyh tehnologij – Security of information tecnologies*. Voronez. 2007. vol. 3. pp. 44–49. (In Russ.).
9. Bulychev P.E. *Algoritmy vychislenija otnoshenij podobija v zadachah verifikacii i restrukturizacii program* [Algorithms for computing similarity relationships in problems of verification and restructuring programs]. Moscow: MGU im. M.V. Lomonosova. 2010. 169 p. (In Russ.).
10. Ahin M.H., Icykson V.M. [Slicing over the trees: the method of detection of broken and twisted clones in source code software] *Modelirovanie i analiz informacionnyh system – Modeling and Analysis of Information Systems*. Jaroslavl'. 2012. vol. 19. no. 6. pp. 69–78. (In Russ.).
11. Podymov V.V. *Bystrye algoritmy proverki jekvivalentnosti programm v modeljah s polugruppovoj semantikoj* [Fast algorithms for checking the equivalence of programs in models with semantics semigroup]. Moscow: MGU im. M.V. Lomonosova. 2014. 164 p. (In Russ.).
12. Calvet J. Cryptographic function identification in obfuscated binary programs. Reverse Engineering Conference. 2012. Available at https://recon.cx/2012/schedule/attachments/46_Joan_CryptographicFunctionIdentification.pdf (accessed: 28/05/2015).
13. Lutz. N. Towards Revealing Attackers' Intent by Automatically Decrypting Network Traffic. Fritz Kutter Fonds. 2008. Available at http://www.kutter-fonds.ethz.ch/App_Themes/default/datalinks/NoeLutz-08.pdf (accessed: 28.05.2015).
14. Wang Z., Jiang X., Cui W., Wang X. ReFormat: Automatic Reverse Engineering of Encrypted Messages. North Carolina State University. 2008. 16 p.
15. Ershov A.P. *Vvedenie v teoreticheskoe programmirovanie (besedy o metode)* [Introduction to theoretical programming (talk about the method)]. Moscow: Izdatel'stvo "Nauka". 1977. 288 p. (In Russ.).
16. Sridharan M., Fink S. J., Bodik R. Thin slicing. ACM SIGPLAN Notices. 2007. vol. 42. No. 6. pp. 112–122.
17. FIPS PUB 180-4. Secure Hash Standard (SHS). National Institute of Standards and Technology. Gaithersburg. 2012. 31 p.

Ершов Алексей Леонидович — научный сотрудник, Академия Федеральной службы охраны Российской Федерации. Область научных интересов: тестирование программного обеспечения. Число научных публикаций — 12. al.er@rambler.ru; Наугорское шоссе, д. 70, к. 58, Орел, 302020; р.т.: +7(919)205-34-48.

Ershov Aleksey Leonidovich — researcher, The Academy of Federal Security Guard Service of the Russian Federation. Research interests: software testing. The number of publications — 12. al.er@rambler.ru; 70, Naugorskoe shosse, apt. 58, Orel, 302020, Russia; office phone: +7(919)205-34-48.

РЕФЕРАТ

Ершов А.Л. **Идентификация алгоритмов преобразования данных в исполняемых модулях программного обеспечения.**

На современном этапе развития информационных технологий, в силу отсутствия у конечного пользователя (в подавляющем большинстве случаев) исходных текстов программного обеспечения, существенно затруднено проведение программно-компьютерной экспертизы, для решения большинства задач которой необходимо понимание логики функционирования программ. В работе рассмотрены известные подходы к идентификации алгоритмов преобразования данных в исполняемых модулях программного обеспечения, которая является одним из основных этапов процесса исследования программного обеспечения с целью понимания его назначения и логики функционирования. Автором предложено решение для упрощения данного этапа за счёт его автоматизации.

В основе предложенного подхода лежит предположение, что исходный алгоритм преобразования данных можно представить в виде совокупности его линейных частей, чьи характеристики можно использовать в качестве признаков для автоматического распознавания неизвестных функциональных объектов в исполняемых модулях программного обеспечения. В дополнение к выделенным признакам для учета внутренних информационных связей предлагается использовать информационный граф, построенный для операторных схем линейных частей алгоритма, оптимизированных с целью экономии памяти. Предложенный подход в определенной степени позволяет абстрагироваться от вариативности реализаций в машинном коде алгоритмов преобразования данных, вносимой разработчиками программного обеспечения.

SUMMARY

Ershov A.L. **Identification of Data Conversion Algorithms in Executable Software Modules.**

At the present stage of the development of information technologies an end user (in most cases) lacks the source code of software, which greatly complicates software forensics, to solve the majority of problems of which we need to understand the logic of the functioning of programs. The paper discusses the known approaches to the identification of data conversion algorithms in the executable software modules, which is one of the main stages of the research of software to understand its purpose and logic of functioning. The author provides a solution to simplify this stage due to its automation.

At the heart of the approach is the assumption that the initial data conversion algorithm can be represented as a combination of its linear parts, whose characteristics can be used as signs for the automatic recognition of unknown functional objects in the executable software modules. In addition to the shown features to account for the internal data connections, author offers to use the information graph built for the operator schemes of linear algorithm parts, optimized to save memory. The proposed approach allows us, at a certain degree, to abstract from variability of implementations in machine code of data conversion algorithms, introduced by software developers.