Применение диверсифицирующих и обфусцирующих преобразований для изменения сигнатуры программного кода*

А.Р. Нурмухаметов <oleshka@ispras.ru>
Институт системного программирования РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. Развитие методов обнаружения вредоносных приложений привело к развитию специальных технологий, которые помогают вредоносным приложениям оставаться незамеченными. Многие из них направлены на изменение сигнатуры программного кода. Для академического изучения этих методов и кода, получаемого с их помощью, необходимо разработать инфраструктуру для автоматизированного изменения сигнатуры имеющейся программы. В данной работе приводится результаты работы по разработке и реализации такого автоматизированного инструмента на базе компиляторной инфраструктуры LLVM и инфраструктуры инструментации и трансформации бинарного кода Syzygy. В рамках этих инфраструктур были реализованы диверсифицирующие и обфусцирующие преобразования, которые были направлены на изменение сигнатуры исполняемого файла. Для работы этих инструментов требуется соответственно наличие исходного кода или отладочной информации. Были разработаны и реализованы следующие преобразования: вставка мертвого кода, перестановка местами инструкций, перестановка местами базовых блоков, перестановка местами функций в модуле, замена инструкций на эквивалентные, шифрование константных буферов данных. По результатам проведенных работ была продемонстрирована работоспособность данного подхода на реальных примерах, а также выявлены недостатки предложенного подхода и пути дальнейшего развития.

Ключевые слова: диверсификация; обфускация; сигнатурный поиск; вредоносные приложения.

DOI: 10.15514/ISPRAS-2016-28(5)-5

Для цитирования: Нурмухаметов А.Р. Применение диверсифицирующих и обфусцирующих преобразований для изменения сигнатур программного кода. Труды ИСП РАН, том 28, вып. 5, стр. 93-104, 2016 г. DOI: 10.15514/ISPRAS-2016-28(5)-5

* Работа поддержана грантом РФФИ 14-01-00462 А

1. Введение

Изучение методов защиты вредоносных приложений от средств обнаружения – важная область компьютерной безопасности. В то время как многие другие области тестирования на проникновения хорошо изучены и их методы документированы, то о тестировании и обходе защит от вредоносных приложений опубликовано не так много работ. До настоящего времени инциденты, связанные с заражениями внутренних сетей крупных организаций, часто встречаются и приносят ощутимые убытки этим организациям. Кроме этого, такие виды заражений являются подходящим средством для проведения целенаправленных атак на конкретные организации. Вредоносные приложения используются как для вывода из строя элементов инфраструктуры, так и для получения доступа к непубличной информации. Исходя из этого, важно уметь оценивать эффективность защитных механизмов программного обеспечения, направленного на обнаружение вредоносных приложений.

Обнаруженное вредоносное приложение не представляет собой никакой опасности. Поэтому активно развиваются методы сокрытия такого кода от систем обнаружения. Для решения этой трудной задачи применяются следующие технологии: полиморфизм, метаморфизм, упаковщики, крипторы [10-14]. Большинство из этих методов сосредоточены на том, чтобы снабдить вредоносное приложение дополнительной функциональностью, позволяющей изменять себя во время распространения или работы для ухода от обнаружения. В данной работе исследуется возможность изменения сигнатур программного кода с помощью обфусцирующих и диверсифицирующих компиляторных преобразований.

Методы обнаружения вредоносных приложений основаны на сигнатурном поиске, эмуляции кода, эвристических анализах. Сигнатурный поиск [3, 30] основывается на поиске известных шаблонов вредоносного кода в бинарном файле. Хотя этот метод имеет свои ограничения, но также он имеет неоспоримые преимущества: низкое количество ложных срабатываний и высокая скорость работы. Благодаря своей простоте и аккуратности сигнатурный поиск имеет широкое применение для обнаружения вредоносных приложений. Сигнатурные методы плохо применимы для обнаружения новых видов вредоносных приложений, поскольку требуют предварительного создания сигнатуры и добавления ее в базу знаний инструмента поиска, и доставки всем его пользователям.

В данной работе описывается подход, основанный на применении методов диверсификации и обфускации и позволяющий разработать инструмент для автоматизированного изменения сигнатуры приложения в исследовательских целях. Предложенные методы позволяют сгенерировать потенциально неограниченное количество различных версий бинарного файла, которые имеют различные сигнатуры, но одинаковую функциональность. Кроме того, приводятся результаты экспериментальной проверки, обсуждаются преимущества и недостатки использованного подхода. Предлагаемый подход

требует для защиты наличия исходного кода приложения или отладочной информации.

Данная работа состоит из шести частей. Первая глава представляет собой введение, в котором обосновывается актуальность данной работы. Во второй главе приводится обзор методов защиты вредоносных приложений. В третьей главе приводится описание предлагаемого подхода. В четвертой главе рассказывается про используемые преобразования и обсуждаются детали реализации. В пятой главе приведены результаты экспериментальной проверки разработанных инструментов. В шестой главе подводятся итоги и намечаются планы дальнейшего развития.

2. Обзор методов защиты

В данном разделе приводится описание методик, применяемых при разработке вредоносного программного обеспечения для обхода методов обнаружения, а также обсуждаются сильные и слабые стороны каждого способа с точки зрения способов защиты.

Крипторы. Широко применяются различные крипторы [9] для обхода сигнатурных средств поиска. Общая идея их применения заключается в использовании шифрования кода вредоносного приложения. Шифрование применяется в качестве обратимого преобразования, для которого криптографическая стойкость, как правило, не является необходимой в контексте изменения сигнатуры бинарного кода. Зашифрованный двоичный код снабжается специальным расшифровщиком. При запуске управление передается на код расшифровщика, который производит дешифрование непосредственно в памяти в процессе выполнения. Код расшифровщика остается неизменным, поэтому средства сигнатурного поиска могут составлять характерные сигнатуры для расшифровщиков и производить их поиск в подозрительном программном обеспечении.

Олигоморфизм. Последовательным шагом в развитии предыдущей технологии является техника олигоморфизма [9, 10]. Она характеризуется изменением кода декриптора при самовоспроизведении от одной версии к другой. Простейший вариант – наличие различных вариантов декриптора, написанных вручную. В процессе генерации новых поколений происходит случайный выбор декриптора. В работе [11] приводится пример, использующий различные декрипторы. Данный подход ограничен тем, что средство сигнатурного поиска может иметь в своем распоряжении шаблоны для самых популярных декрипторов и успешно их обнаруживать. **Полиморфизм.** Техника полиморфизма [9, 10, 11] является логичным развитием идей, на которых основаны крипторы и олигоморфные вредосные приложения. Она обладает способностью генерировать больше разных декрипторов. Цель состоит в том, чтобы генерировать уникальный код для каждой новой версии, не оставляя общей сигнатуры. Применяются методы трансформации программного кода: замена инструкций на эквивалентные, перестановка

местами инструкций, переименование регистров, вставка мертвого кода для достижения этой цели. Для противодействия такому подходу используется метод эмуляции кода. Во время эмуляции происходит расшифровка кода и сигнатурный поиск позволяет найти характерные куски кода.

Метаморфизм. Техника метаморфизма [9, 11, 12, 23] расширяет возможности предыдущих этапов и генерирует не только новый декриптор, но и полностью новое тело вредоносного приложения при сохранении его поведения. Основой для формирования следующего поколения является код в некотором его представлении, доступном для анализа и трансформации (зачастую это некоторое промежуточное представление). Данный код содержит всю необходимую последовательность действий и транслируется в исполняемый двоичный код. При этом процесс трансляции включает в себя некоторую неопределенность, которая позволяет генерировать различные сигнатуры. Для обнаружения таких вредоносных приложений используется поведенческий и эвристический анализ. Существенным минусом эвристического анализа является его точность, т.е. количество ложных срабатываний.

Протекторы. Другим примером защиты являются протекторы [15-17]. Они представляют собой виртуальные машины, построенные на основе некоторых случайно-генерированных машинных архитектур, могут содержать приемы антиотладки и методы запутывания бинарного кода. Самыми известными примерами являются: VMProtect [18], Themida [19], ASProtect [20]. Опишем подробнее подход, осуществлённый при реализации одного из протекторов. Он основан на построении виртуальной машины некоторого несуществующего процессора и преобразовании обычного кода в код для этого процессора с последующим выполнением его на этой виртуальной машине. Эта технология была реализована на основе LLVM и QEMU. Случайным образом машинные коды инструкций x86 менялись друг на друга с соблюдением ряда нетривиальных ограничений на взаимозаменяемость. Для полученной архитектуры набора инструкций автоматически генерировались свои файлы описания целевой архитектуры компилятора LLVM, после чего он собирался для новой архитектуры. Кроме этого, вносились соответствующие изменения в QEMU для поддержания возможности эмуляции новой архитектуры. Исходный код вредоносного приложения транслировался полученным компилятором, снабжался эмулятором и логикой, запускающей данный код на эмуляторе. Авторы статьи [20] приводят цифры, показывающие практически неограниченную мощность диверсификации сигнатуры исходного кода данного подхода.

3. Описание предлагаемого подхода

Методы трансформации программ, используемые компиляторами, хорошо изучены и развиты в инструментальных средствах. Существующие промышленные компиляторные инфраструктуры Clang/LLVM [1, 22], GCC [2] содержат в себе большое количество преобразований, направленных на

оптимизацию программного кода. Они используют собственные внутренние представления программы.

Компиляторные инфраструктуры предоставляют возможность для удобной реализации новых преобразований над кодом программы в виде промежуточного представления. Преобразования запускаются друг за другом. Каждое из них изменяет промежуточное представление, после чего запускается следующее преобразование. Автоматическое изменение кода программы внутри компиляторных инфраструктур с целью изменения сигнатуры представляется наиболее логичным подходом.

В данной работе предлагается использовать компиляторную инфраструктуру для реализации специальных преобразований над промежуточным представлением, которые бы приводили к изменению кода приложения таким образом, чтобы изменить характерные признаки его сигнатуры. За основу такого решения был взят разработанный обфусцирующий компилятор [26]. Стоит отметить, что данный компилятор разработан на основе Clang/LLVM. Это накладывает некоторые ограничения на его использование. Основное ограничение заключается в недостаточной поддержке платформозависимых особенностей, например, ограниченность поддержки кода, специфичного для Microsoft Visual Studio Compiler. Данная компиляторная инфраструктура является закрытой и не предоставляет возможности для своего расширения.

Для изменения вредоносных приложений под Windows, не компилируемых Clang/LLVM, в данной работе предлагается использовать инструмент Syzygy от Google [24]. Syzygy - набор утилит для послесборочной инструментации и оптимизации 32-битных приложений для Windows в условиях наличия отладочной информации. Выполняет задачи инструментации кода для подсчета метрик, покрытия и профилирования кода. Собранная информация используется для оптимизации запуска приложения и поиска ошибок с памятью. Часть преобразований из обфусцирующего компилятора перенесена в Syzygy для обеспечения возможности генерации различных исполняемых файлов. Отметим, что дизассемблирование в отсутствие отладочной информации является неразрешимой задачей в общем случае, поэтому наличие отладочной информации необходимо для работы этого инструмента.

4. Используемые методы трансформации кода

Диверсифицирующие и обфусцирующие трансформации [7, 8, 10] напоминают компиляторные по своему устройству, но преследуют другие цели. Цель оптимизирующих преобразований – увеличить скорость работы программы или уменьшить занимаемое место. Цель диверсифицирующих преобразований – изменить код приложения, не изменяя его функциональности. Целью обфусцирующих преобразований является запутывание приложения таким образом, чтобы осложнить понимание принципов работы алгоритмов приложения и структур данных. Диверсифицирующие и обфусцирующие трансформации зависят от случайной величины, получаемой от генератора

случайных чисел. Все описываемые ниже преобразования были реализованы в рамках обфусцирующего компилятора, разработанного в ИСП РАН [27, 28]. Часть из них была также реализована в инструменте бинарной инструментации Syzygy.

Вставка недостижимого кода. *Недостижимый код* – это код, который не выполняется в программе ни при каких входных данных. Вставка недостижимого кода может быть реализована различными способами [25]. Недостижимый код вставляется в граф потока управления в виде нового базового блока. Вход в этот базовый блок закрывается непрозрачным предикатом [26]. Это преобразование может быть использовано с целью увеличения метрики сигнатурного различия оригинального и трансформированного приложения.

Вставка бесполезных инструкций. Данное преобразование привносит в код программы инструкции, которые, в отличии от недостижимого кода выполняются, но никак не влияют выходные данные или наблюдаемое поведение приложения [29]. Количество привносимых инструкций может быть любым, как и их сложность, поэтому данной трансформацией можно добиться континуального множетсва вариантов программы. В обоих инструментах была реализована вставка инструкций и их последовательностей, которые ничего не делают: nop; xchg eax, eax; xchg eax, ebx, xchg eax, ebx.

Замена инструкций на эквивалентные. Существуют различные машинные инструкции для выполнения одних и тех же действий, поэтому возможно заменять инструкции друг на друга. Данное преобразование было реализовано внутри компиляторной инфраструктуры LLVM на уровне машинно-зависимого кода как peephole оптимизации. Трансформация использует набор шаблонов для замены инструкций. Большая часть шаблонов связана с заменой логико-арифметических выражений на эквивалентные, но выраженные через другие операции.

Перестановка местами инструкций. С помощью стандартного планировщика инструкций перестановка местами инструкций была реализована внутри компиляторной инфраструктуры LLVM. Кроме того, данное преобразование было реализовано и в инфраструктуре Syzygy. Для этого был реализован дополнительный анализ зависимостей между инструкциями. Результаты данного анализа позволяют определять вид зависимостей между инструкциями и на основании этого определять те инструкции, которые можно менять друг с другом местами внутри одного базового блока.

Перестановка базовых блоков. Базовый блок – линейная последовательность инструкций, оканчивающаяся единственной инструкцией передачи управления следующему базовому блоку. Последовательность раскладки базовых блоков в образе программы можно менять, что позволяет менять сигнатуру приложения. Перестановка базовых блоков в случайном порядке в пределах одной функции была реализована внутри инструмента Syzygy.

Перестановка местами функций в модуле. Перестановка местами функций дает дополнительную возможность по изменению сигнатуры программы на более крупнозернистом уровне по сравнению с перестановкой базовых блоков. [21]. Данное преобразование было реализовано внутри компиляторной инфраструктуры LLVM. Функции в пределах одного модуля трансляции переставлялись в случайном порядке следования.

Шифрование константных буферов. Константные буферы, хранящиеся в памяти процесса, могут быть источником характерных сигнатур и причиной обнаружения. Преобразование [26], маскирующее такие буферы, предназначено для изменения их сигнатуры. Шифрование выполняется следующим образом: все константные буферы кроме тех, для которых невозможно гарантировать консервативность, шифруются; в модуль добавляются функции расшифровки и шифровки. Перед каждым использованием буфера вставляется расшифровывающая функция, а после шифрующая. Данное преобразование было реализовано в обфусцирующем компиляторе, разрабатываемом в ИСП РАН [27, 28].

5. Результаты

Для экспериментальной проверки работоспособности описанного подхода были проведены исследования по запутыванию реальных образцов вредоносных приложений. В данной работе в качестве средства сигнатурного поиска использовался проект с открытым исходным кодом Clam [5]. Были взяты приложения из открытого набора вредоносных приложений с исходным кодом theZoo [6], который содержит в себе примеры под Windows. Стоит отметить, что не все примеры из этого набора успешно собираются и обнаруживаются средствами сигнатурного поиска. Для двух из них ‘NullBot’, ‘xTBot’ удалось получить запутанные образцы с помощью инструмента на основе Syzygy, которые не обнаруживались средством сигнатурного поиска. Для проверки обфусцирующего компилятора были использованы примеры вредоносных приложений с исходным кодом на языке C для Linux с ресурса [4]. К сожалению, из двадцати примеров только один собирается без ошибок и после оригинальной сборки обнаруживается средством сигнатурного поиска, а именно – vit. Запутывание его обфусцирующим компилятором привело к исчезновению из его кода характерной сигнатуры.

В ходе экспериментального тестирования разработанных преобразований был выявлен их недостаток. Он заключается в том, что данные преобразования носят случайный характер. Они производятся над теми местами в коде, над которыми их можно осуществить. Это приводит к тому, что, преобразования могут давать разный эффект в зависимости от начальной затравки. Например, вредоносное приложение NullBot было запутано сто раз с разными значениями начальной затравки для линейного генератора псевдослучайных чисел. Характерная сигнатура обнаруживалась в этом приложении только в 19

случаях, тогда как в 81 случае ее в коде трансформированного приложения не присутствовало.

Направленность диверсифицирующих и обфусцирующих преобразований может быть исправлена путем создания специальных метрик, численно показывающих степень различия двух исполняемых файлов с точки зрения сигнатурного поиска.

6. Заключение

В данной работе описывается подход к разработке инструментального средства и соответствующей инфраструктуры, предназначенного для изучения возможных способов обфускации и диверсификации, которыми могут пользоваться вредоносные приложения. Данная инфраструктура может быть полезна для последующего поиска способов противодействия. Проведен обзор существующих методов обхода сигнатурных типов защиты. Предложен и реализован набор преобразований, направленных на достижение поставленной цели. Приведены результаты экспериментальной проверки предложенного подхода. Они показывают, что диверсифицирующие и обфусцирующие преобразования могут применяться для решения задачи изменения сигнатуры программного кода. Кроме того, обнаружено ограничение представленного подхода – ненаправленность проводимых преобразований. Обсуждаются способы исправления этих недостатков и перспективные пути развития предлагаемого подхода.

Список литературы

[1]. LLVM Compiler Infrastructure. http://llvm.org/
[2]. GCC Compiler Infrastructure. https://gcc.gnu.org/
[3]. Radare2. http://radare.org/r/
[4]. Malware Source Collection. http://vxheaven.org.
[5]. Clam. http://www.clamav.net/
[6]. Malware Open Source Collection. https://github.com/ytisf/theZoo
[7]. Source-Free Binary Mutation for Offense and Defense. V. Mohan. 2014.
[8]. Algorithmic Diversity for Software Security. https://arxiv.org/abs/1312.3891
[9]. You and K. Yim, "Malware Obfuscation Techniques: A Brief Survey," Broadband, Wireless Computing, Communication and Applications (BWCCA), 2010 International Conference on, Fukuoka, 2010, pp. 297-300.
[10]. Ashu Sharma and S K Sahay. Article: Evolution and Detection of Polymorphic and Metamorphic Malwares: A Survey. International Journal of Computer Applications 90(2):7-11, March 2014.
[11]. Rad, B., Masrom, M. and Ibrahim, S. “Camouflage in Malware: From Encryption to Metamorphism”, International Journal of Computer Science and Network Security, 2012, 12: 74-83.
[12]. P. OKane, S. Sezer and K. McLaughlin, "Obfuscation: The Hidden Malware," in IEEE Security & Privacy, vol. 9, no. 5, pp. 41-47, Sept.-Oct. 2011. doi: 10.1109/MSP.2011.98

[13]. Tom Brosch, Maik Morgenstern AV-Test GmbH. Runtime Packers: The Hidden Problem? Black Hat USA'06. https://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Morgenstern.pdf
[14]. Ultimate Packer for eXecutables. http://upx.sourceforge.net/
[15]. ASPack. http://www.asprotect.ru/aspack.html
[16]. FSG. https://exelab.ru/_dl-nLh/pack/fsg20.rar
[17]. VMProtect. http://vmpsoft.com/
[18]. Themida. http://www.oreans.com/themida.php
[19]. ASProtect. http://www.asprotect.ru/asprotect64.html
[20]. AlphaPack Protector Report. https://github.com/graulito/alphapack
[21]. Application of Compiler Transformation Against Software Vulnerabilities Exploitation. A. Nurmukhametov, Sh. Kurmangaleev, V. Kaushan, S. Gaisaryan. Programming and Computer Software. 2015. V. 41, № 4. P. 231-236. Doi: 10.1134/S0361768815040052.
[22]. Lattner C. LLVM: An Infrastructure for Multi-Stage Optimization. Master's thesis, Computer Science Dept., University of Illinois at Urbana-Champaign, Urbana, IL.
[23]. Philippe Beaucamps. Advanced Metamorphic Techniques in Computer Viruses. International Conference on Computer, Electrical, and Systems Science, and Engineering - CESSE'07, Nov 2007, Venice, Italy. 2007.
[24]. Syzygy Transformation Toolchain. url: https://github.com/google/syzygy/
[25]. Tamboli Teja. Metamorphic Code Generation from LLVM IR Bytecode: Master's thesis. San Jose State University. San Jose. 2013.
[26]. В. Иванников, Ш. Курмангалеев, А. Белеванцев [и др.]. Реализация запутывающих преобразований в компиляторной инфраструктуре LLVM. Труды ИСП РАН, том 26, вып. 1, 2014, стр. 327-342. DOI: 10.15514/ISPRAS-2014-26(1)-12.
[27]. Ш. Ф. Курмангалеев, В. П. Корчагин, В. В. Савченко [и др.]. Построение обфусцирующего компилятора на основе инфраструктуры LLVM. Труды ИСП РАН, том 23, 2012, стр. 77-92. DOI: 10.15514/ISPRAS-2012-23-5.
[28]. Курмангалеев Ш. Ф., Корчагин В. П., Матевосян Р. А. Описание подхода к разработке обфусцирующего компилятора Труды ИСП РАН, том 23, 2012, стр. 67-76. DOI: 10.15514/ISPRAS-2012-23-4.
[29]. Software Tamper Resistance: Obstructing Static Analysis of Programs: Tech. Rep.: Chenxi Wang, Jonathan Hill, John Knight [и др.]. Charlottesville, VA, USA: 2000.
[30]. Clam. http://www.clamav.net/

The Application of Compiler-based Obfuscation and Diversification for Program Signature Modification.

A.R. Nurmukhametov <oleshka@ispras.ru>
Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.

Abstract. Development of malware detection techniques leads to the evolution of anti-detection techniques. In this paper we discuss possibility of creating an automatic tool for signature modification. In this article we describe our experience in designing and development of such tool. For signature modification in Linux programs we implemented a tool based on LLVM compiler infrastructure and for Windows programs we used post-link instrumentation and optimization tool Syzygy. The former approach requires program source code, while the latter assumes only the presence of debug information. Diversifying and obfuscating transformations were implemented in both cases with the aim of changing the signature of program to prevent matching them the known patterns. Implemented transformations are bogus code insertion, function permutation, instruction substitution, ciphering of constant buffer. As a result we demonstrate proof-of-concept examples which confirm that it is possible to automatically change of program signature for avoiding detection by signature-based analysis. Furthermore we explain drawbacks of this technique and discuss the further ways of development.

Keywords: diversification; obfuscation; signature analysis.

DOI: 10.15514/ISPRAS-2016-28(5)-5

For citation: Nurmukhametov A.R. The Application of Compiler-based Obfuscation and Diversification for Program Signature Modification. Trudy ISP RAN/Proc. ISP RAS, 2016, vol. 28, issue 5, 2016, pp. 93-104 (in Russian). DOI: 10.15514/ISPRAS-2016-28(5)-5

References

[1]. LLVM Compiler Infrastructure. http://llvm.org/
[2]. GCC Compiler Infrastructure. https://gcc.gnu.org/
[3]. Radare2. http://radare.org/r/
[4]. Malware Source Collection. http://vxheaven.org.
[5]. Clam. http://www.clamav.net/
[6]. Malware Open Source Collection. https://github.com/ytisf/theZoo
[7]. Source-Free Binary Mutation for Offense and Defense. V. Mohan. 2014.
[8]. Algorithmic Diversity for Software Security. https://arxiv.org/abs/1312.3891
[9]. You and K. Yim, "Malware Obfuscation Techniques: A Brief Survey," Broadband, Wireless Computing, Communication and Applications (BWCCA), 2010 International Conference on, Fukuoka, 2010, pp. 297-300.

[10]. Ashu Sharma and S K Sahay. Article: Evolution and Detection of Polymorphic and Metamorphic Malwares: A Survey. International Journal of Computer Applications 90(2):7-11, March 2014.
[11]. Rad, B., Masrom, M. and Ibrahim, S. “Camouflage in Malware: From Encryption to Metamorphism”, International Journal of Computer Science and Network Security, 2012, 12: 74-83.
[12]. P. OKane, S. Sezer and K. McLaughlin, "Obfuscation: The Hidden Malware," in IEEE Security & Privacy, vol. 9, no. 5, pp. 41-47, Sept.-Oct. 2011. doi: 10.1109/MSP.2011.98
[13]. Tom Brosch, Maik Morgenstern AV-Test GmbH. Runtime Packers: The Hidden Problem? Black Hat USA’06.https://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Morgenstern.pdf
[14]. Ultimate Packer for eXecutables. http://upx.sourceforge.net/
[15]. ASPack. http://www.asprotect.ru/aspack.html
[16]. FSG. https://exelab.ru/_dl-nLh/pack/fsg20.rar
[17]. VMProtect. http://vmpsoft.com/
[18]. Themida. http://www.oreans.com/themida.php
[19]. ASProtect. http://www.asprotect.ru/asprotect64.html
[20]. AlphaPack Protector Report. https://github.com/graulito/alphapack
[21]. Application of Compiler Transformation Against Software Vulnerabilities Exploitation. A. Nurmukhametov, Sh. Kurmangaleev, V. Kaushan, S. Gaisaryan. Programming and Computer Software. 2015. V. 41, № 4. P. 231-236. Doi: 10.1134/S0361768815040052.
[22]. Lattner C. LLVM: An Infrastructure for Multi-Stage Optimization. Master’s thesis, Computer Science Dept., University of Illinois at Urbana-Champaign, Urbana, IL.
[23]. Philippe Beaucamps. Advanced Metamorphic Techniques in Computer Viruses. International Conference on Computer, Electrical, and Systems Science, and Engineering - CESSE'07, Nov 2007, Venice, Italy. 2007.
[24]. Syzygy Transformation Toolchain. url: https://github.com/google/syzygy/
[25]. Tamboli Teja. Metamorphic Code Generation from LLVM IR Bytecode: Master’s thesis. San Jose State University. San Jose. 2013.
[26]. Ivannikov V., Kurmangaleev S., Belevantsev A., Nurmukhametov A., Savchenko V., Matevosyan H., Avetisyan A. Implementing Obfuscating Transformations in the LLVM Infrastructure. Trudy ISP RAN/Proc. ISP RAS, vol. 26, issue 1, 2014. pp. 327-342. (in Russian). DOI: 10.15514/ISPRAS-2014-26(1)-12
[27]. Kurmangaleev S.F., Korchagin V.P., Savchenko V.V., Sargsyan S.S. Building obfuscation compiler based on LLVM infrastructure. Trudy ISP RAN/Proc. ISP RAS, vol. 23, 2012, pp. 77-92 (in Russian). DOI: 10.15514/ISPRAS-2012-23-5
[28]. Kurmangaleev S.F., Korchagin V.P., Matevosyan H.A. Description of the approach to development of the obfuscating compiler. Trudy ISP RAN/Proc. ISP RAS, vol. 23, 2012, pp. 67-76 (in Russian). DOI: 10.15514/ISPRAS-2012-23-4
[29]. Software Tamper Resistance: Obstructing Static Analysis of Programs: Tech. Rep.: Chenxi Wang, Jonathan Hill, John Knight [и др.]. Charlottesville, VA, USA: 2000.
[30]. Clam Antivirus Software. http://www.clamav.net/