Технология полиморфной модификации программного кода для повышения надежности функционирования программного обеспечения в недоверенной среде

**Человек:** В современном информационном пространстве, когда критические системы подключены к сетям общего доступа, в том числе и глобальной сети Интернет, а программное обеспечение запускается в недоверенных средах, возникает потребность в наличии средств противодействия информационным угрозам, а также повышения надежности функционирования программного обеспечения. Особенно актуальным это становится в условиях стремительного развития кибероружия, которое обладает возможностью по перехвату управления или разрушению критически важных для функционирования общества и государства сетей и объектов производственной, социальной, военной и финансовой инфраструктуры, а также высокоавтоматизированных производственных, технологических линий. В данной работе рассматривается подход к повышению надежности функционирования, с точки зрения компьютерной безопасности, программного обеспечения в недоверенной среде при помощи полиморфной модификации программного кода. Результаты исследования могут применяться для защиты программных компонентов критически важных инфраструктур как от непреднамеренного воздействия со стороны программного окружения, так и от преднамеренного со стороны вредоносного ПО.

**Key words:** обфускация, транспайлер, компилятор, интерпретатор, модификация программного кода, запутывающие преобразования, запутывающий компилятор, компиляторная инфраструктура, языки программирования, программное обеспечение

=================================

**FastText\_KMeans\_Clean:** Исследовательский проект HES-SO/RCSO "Obfuscator – LLVM". перенос локальных переменных в глобальную область видимости, шифрование строк, вставка фиктивных циклов, диспетчеризация, переплетение функций, сокрытие вызовов функций, разбиение констант, клонирование функций, формирование непрозрачных предикатов. Преобразование форматирования. Преобразование потока управления. Превентивное преобразование. Наиболее близким к данному техническому решению подход — это использование виртуальной машины, однако он обладает следующими недостатками: восстановление исходного фрагмента после полного анализа и обратная трансляция в систему команд целевой архитектуры, широкий спектр инструментов для анализа и декомпиляции, ограничение применимости только для компилируемых языков.

**Key words part:** 0.4230769230769231

=================================

**FastText\_KMeans\_Raw/:** перенос локальных переменных в глобальную область видимости, шифрование строк, вставка фиктивных циклов, диспетчеризация, переплетение функций, сокрытие вызовов функций, разбиение констант, клонирование функций, формирование непрозрачных предикатов. Преобразование форматирования. Преобразование потока управления. Превентивное преобразование. Наиболее близким к данному техническому решению подход — это использование виртуальной машины, однако он обладает следующими недостатками: восстановление исходного фрагмента после полного анализа и обратная трансляция в систему команд целевой архитектуры, широкий спектр инструментов для анализа и декомпиляции, ограничение применимости только для компилируемых языков. Антивирусное программное обеспечение для проведения эксперимента выбиралось из списка доступных контрибьюторов (производителей предоставляющих свои разработки) на сервисе Virustotal, с учетом известности их производителя: AVG Antivirus Free, Avira Antivirus Free, BitDefender Internet Security, COMODO Antivirus, G Data Antivirus, ESET NOD32 Antivirus, Kaspersky Antivirus, McAfee Antivirus, Norton Security, Trend Micro Antivirus Plus.

**Key words part:** 0.5384615384615384

=================================

**FastText\_PageRank\_Clean/:** Esprima, SpiderMonkey, UglifyJS, Traceur, Acorn. Python ast module, RedBaron, Astor, Astcheck, AST Optimizer. System.Management.Automation, Language.Parser, Metaspec, Roslyn, srcML, NRefactory. 2) [27, 32, 33, 34]. Наименование проекта. Исследовательский проект "confuse". Исследовательский проект ИСП РАН "Обфускатор". Язык программирования.

**Key words part:** 0.4230769230769231

=================================

**FastText\_PageRank\_Raw/:** Esprima, SpiderMonkey, UglifyJS, Traceur, Acorn. Python ast module, RedBaron, Astor, Astcheck, AST Optimizer. Javaparser, Spoon, JDT, srcML, Javac api, Javapoet. 2) [27, 32, 33, 34]. Наименование проекта. Исследовательский проект "confuse". Коммерческий проект C/C++ "Obfuscator". Язык программирования.

**Key words part:** 0.4230769230769231

=================================

**Mixed\_ML\_TR/:** В данной работе рассмотрена технология полиморфного изменения программного кода, основывающаяся на преобразованиях абстрактного синтаксического дерева с помощью алгоритмов запутывания кода (обфускации) и транспайлеров (интерпретаторов, преобразующих код программы, написанной на одном языке программирования (ЯП), в аналогичный код другого языка программирования). В рамках исследования методов обфускации рассматривалась как базовая функциональность, необходимая для разнообразия получаемых вариантов программного кода, а также методы затруднения анализа программного обеспечения. Вычислительная обфускация это группа методов, приводящих к изменению структуры графа потока управления, например, добавление недостижимого кода, расширение и добавление новых условий циклов, условных операторов, добавление избыточных операций. Данные преобразования предназначены для противодействия определенным методам декомпиляции программ [23], в некоторых случаях могут использовать недостатки или особенности работы определенных инструментальных средств анализа кода и декомпиляции. Реализованные методы обфускации. Непрозрачные предикаты, вставка функций, диспетчеризация, выделения функций, вставка мертвого кода, переплетение функций. Преобразование потока данных. Таким образом, разработанные к настоящему времени практические методы и алгоритмы обфускации программ способны затруднить понимание и модификацию программ, но не способны в полной мере решить задачу защиты доверенного ПО. Исходя из этих недостатков были разработаны методы, которые помогают обнаружить факт работы доверенного ПО в режиме эмуляции. Сложность анализа программного кода данные преобразования не изменяют.

**Key words part:** 0.7307692307692307

=================================

**MultiLingual\_KMeans/:** В рамках исследования методов обфускации рассматривалась как базовая функциональность, необходимая для разнообразия получаемых вариантов программного кода, а также методы затруднения анализа программного обеспечения. Вычислительная обфускация это группа методов, приводящих к изменению структуры графа потока управления, например, добавление недостижимого кода, расширение и добавление новых условий циклов, условных операторов, добавление избыточных операций. Реализованные методы обфускации. Непрозрачные предикаты, вставка функций, диспетчеризация, выделения функций, вставка мертвого кода, переплетение функций. Преобразование потока данных. Таким образом, разработанные к настоящему времени практические методы и алгоритмы обфускации программ способны затруднить понимание и модификацию программ, но не способны в полной мере решить задачу защиты доверенного ПО. Исходя из этих недостатков были разработаны методы, которые помогают обнаружить факт работы доверенного ПО в режиме эмуляции.

**Key words part:** 0.6153846153846154

=================================

**Multilingual\_PageRank/:** Esprima, SpiderMonkey, UglifyJS, Traceur, Acorn. Использование внешнего инструмент построения АСД обусловлено тем, что создание собственного инструмента является крайне трудоемкой задачей и не имело практического смысла в данной работе. Отсутствие комментариев не влияет существенно на анализ. 2) [27, 32, 33, 34]. Общим их недостатком является отсутствие гарантированной надежности. В рамках исследования было установлено, что патентованные решения или аналоги данной технологии отсутствуют. Данные преобразования не рассматривались для исследования метрических характеристик. Прототип технологии не обрабатывает данную информацию, что является основной причиной недостижения значений экспериментальной оценки прототипа до 100% для всех серий экспериментов.

**Key words part:** 0.38461538461538464

=================================

**RuBERT\_KMeans\_Without\_ST/:** В данной работе рассмотрена технология полиморфного изменения программного кода, основывающаяся на преобразованиях абстрактного синтаксического дерева с помощью алгоритмов запутывания кода (обфускации) и транспайлеров (интерпретаторов, преобразующих код программы, написанной на одном языке программирования (ЯП), в аналогичный код другого языка программирования). Преобразования форматирования. Преобразование потока управления. Стоит заметить, что так как транспайлирование это перевод среды исполнения целевого языка программирования в среду исполнения другого языка программирования, само по себе это не позволяет добиться повышения выживаемости доверенного программного обеспечения в недоверенной среде и не решает проблем, выявленных для классических методов обфускации.

**Key words part:** 0.6923076923076923

=================================

**RuBERT\_KMeans\_With\_ST/:** В данной работе рассмотрена технология полиморфного изменения программного кода, основывающаяся на преобразованиях абстрактного синтаксического дерева с помощью алгоритмов запутывания кода (обфускации) и транспайлеров (интерпретаторов, преобразующих код программы, написанной на одном языке программирования (ЯП), в аналогичный код другого языка программирования). Esprima, SpiderMonkey, UglifyJS, Traceur, Acorn. Вычислительная обфускация это группа методов, приводящих к изменению структуры графа потока управления, например, добавление недостижимого кода, расширение и добавление новых условий циклов, условных операторов, добавление избыточных операций. Преобразование форматирования. Преобразование потока управления. Скремблирование идентификаторов, преобразования функций.

**Key words part:** 0.6538461538461539

=================================

**RUBERT\_page\_rank\_Without\_ST/:** Название ЯП. Средства построения АСД. 2) [27, 32, 33, 34]. Наименование проекта. Сокрытие импорта исполняемых файлов(Linux). 3. Описание технологии.

**Key words part:** 0.3461538461538461

=================================

**RUBERT\_page\_rank\_With\_ST/:** Название ЯП. Средства построения АСД. Изменяют только внешний вид программы. На рисунке 1 представлена схема классического механизма работы транспайлера. Рисунок 1 – Механизм работы транспайлера. Важно заметить, что также можно разделять среды физически, организовав подобие клиент-серверного общения, а на каждый получившийся фрагмент кода можно использовать результаты обфусцирующих преобразований от классических обфускаторов.

**Key words part:** 0.4615384615384616

=================================

**RUSBERT\_KMeans\_Without\_ST/:** Преобразование форматирование. Ключевой особенностью, отличающей предлагаемый подход, является использование технологии транспайлирования как средства запутывающих преобразований, а новизна данного подхода состоит в том, что, после трансляции части программы в другой язык программирования, обеспечивается сохранение связей между компонентами для их взаимодействия во время исполнения программы. ﹣ преобразования форматирования: являются не только тривиальными с точки зрения их реализации и преодоления, но и не представляют интереса в контексте исследования метрических характеристик. Следует отметить, что рассмотренное преобразование может возникать и в качестве побочного эффекта замены инструкции возврата из функции инструкцией перехода – тогда точка выхода из функции не всегда может быть распознана корректно и часть кода вызывающей функции ошибочно относится к вызываемой.

**Key words part:** 0.5769230769230769

=================================

**RUSBERT\_KMeans\_With\_ST/:** Преобразования потока управления. Также предусматривается разделение сред исполнения между преобразованными и не преобразованными частями программного кода. В ходе теоретических и экспериментальных исследований по разработке технологии полиморфной модификации программного кода были исследованы и выявлены преимущества и недостатки существующих решений. Разработан новый подход к обфускации программного кода, который в значительной степени ограничивает возможность понимания алгоритмов функционирования программного обеспечения.

**Key words part:** 0.6153846153846154

=================================

**RUSBERT\_page\_rank\_Without\_ST/:** Листья являются пустыми операторами и представляют только переменные и константы [23]. Исследовательский проект "confuse". Наименование метода. Вставка функций, выделение функций. Общим их недостатком является отсутствие гарантированной надежности.

**Key words part:** 0.3461538461538461

=================================

**RUSBERT\_page\_rank\_With\_ST/:** Преобразования структур данных. Наименование проекта. Исследовательский проект "confuse". Наименование метода. Преобразования структур данных.

**Key words part:** 0.38461538461538464

=================================

**Simple\_PageRank/:** В рамках исследований необходимо получить различные способы построения запутывающих преобразований, в основе которых лежит принцип полиморфной модификации абстрактного синтаксического дерева (АСД). В результате сравнительного анализа средств получения АСД для интерпретируемых языков были выбраны инструменты, которые предоставляют необходимую функциональность и максимально покрывают язык программирования, что оказывает первичное влияние на скорость и качество исследований. Стоит заметить, что так как транспайлирование это перевод среды исполнения целевого языка программирования в среду исполнения другого языка программирования, само по себе это не позволяет добиться повышения выживаемости доверенного программного обеспечения в недоверенной среде и не решает проблем, выявленных для классических методов обфускации. На этапе проверки технологии была произведена аналитическая оценка эффективности методов запутывающих преобразований с использованием метрических характеристик, описывающих сложность кода программы [30]. ﹣ транспайлирование фрагментов кода увеличивает трудоемкость анализа кода в целом, но при этом может уменьшать значение характеристики сложности потока управления, например, транспайлирование фрагмента кода большой и сложной функции уменьшает ее количественные характеристики, а также характеристики сложности потока управления. При проведении экспериментов ставилась задача показать, что средства защиты, выступая как враждебная среда, не смогут существенно нарушить, либо прекратить вообще функционирование экземпляра ПО после применения разработанной технологии, что позволит говорить об успешном получении экспериментальной оценки эффективности.

**Key words part:** 0.7692307692307693

=================================

**TextRank/:** В данной работе рассмотрена технология полиморфного изменения программного кода, основывающаяся на преобразованиях абстрактного синтаксического дерева с помощью алгоритмов запутывания кода (обфускации) и транспайлеров (интерпретаторов, преобразующих код программы, написанной на одном языке программирования (ЯП), в аналогичный код другого языка программирования). Данные преобразования предназначены для противодействия определенным методам декомпиляции программ [23], в некоторых случаях могут использовать недостатки или особенности работы определенных инструментальных средств анализа кода и декомпиляции. При исследовании методов обфускации рассматривались также возможности классических транспайлеров (тип компилятора, который использует исходный код программы, написанной на одном языке программирования, в качестве исходных данных и производит эквивалентный исходный код на другом языке программирования), которые особенно хорошо реализуются для интерпретируемых языков программирования. Ключевой особенностью, отличающей предлагаемый подход, является использование технологии транспайлирования как средства запутывающих преобразований, а новизна данного подхода состоит в том, что, после трансляции части программы в другой язык программирования, обеспечивается сохранение связей между компонентами для их взаимодействия во время исполнения программы. На этапе проверки технологии была произведена аналитическая оценка эффективности методов запутывающих преобразований с использованием метрических характеристик, описывающих сложность кода программы [30]. Метрические характеристики сложности кода программы ставят в соответствие каждой программе некоторое численное значение, а запутывающие преобразования должны приводить к существенному увеличению численного показателя данных характеристик [29].

**Key words part:** 0.8076923076923077

=================================

**TF-IDF\_KMeans/:** Преобразования форматирования. Превентивные преобразования. 2) [27, 32, 33, 34]. Скремблирование идентификаторов. Преобразование потока управления. Непрозрачные предикаты, вставка функций, диспетчеризация, выделения функций, вставка мертвого кода, переплетение функций. При исследовании методов обфускации рассматривались также возможности классических транспайлеров (тип компилятора, который использует исходный код программы, написанной на одном языке программирования, в качестве исходных данных и производит эквивалентный исходный код на другом языке программирования), которые особенно хорошо реализуются для интерпретируемых языков программирования.

**Key words part:** 0.6153846153846154

=================================

**Текст:** Современное программное обеспечение (далее ПО) подвергается различному риску, в том числе при использовании в недоверенных средах. Особенно это критично для отечественного программного обеспечения, функционирующего в условиях, когда в управляющих (операционных) системах широко используются зарубежные компоненты, а полный переход на отечественные реализации пока не возможен. Разработка программного обеспечения – сложная, многоступенчатая работа, требующая тщательной проработки на каждом этапе. В современных реалиях одним из таких этапов становится учет среды исполнения, а также возможных внешних средств анализа программного обеспечения. Существующие в настоящее время подходы не позволяют решать данную задачу в соответствии с предъявляемыми требованиями на всех этапах жизненного цикла программного обеспечения [3]. Особенно это критично когда ПО должно функционировать совместно со сторонними средствами защиты, например, брандмауэрами и антивирусами, которые непреднамеренно могут нарушить корректный цикл выполнения ПО. Одним из существующих подходов по повышению выживаемости в недоверенной среде является использование обфусцирующих преобразований, например, алгоритмов и подходов, раскрываемых в работах К. Колберга [4,5]. В данном исследовании также были рассмотрены эти алгоритмы, использована их классификация [4]. Однако, применение одних лишь обфусцирующих преобразований не позволяет достичь такой вариативности получаемых результатов, когда программное обеспечение не будет поддаваться анализу со стороны автоматизированных и ручных средств анализа кода. Яркими представителями программного обеспечения, которые стараются противодействовать недоверенной среде исполнения, являются компьютерные вирусы и их изучение может быть достаточно полезным с инженерной и математической точек зрения [6,7]. В них используются различные технологии противодействия анализу: антиотладочные и антиэмуляционные приемы, полиморфизм, метаморфизм, упаковщики, крипторы [8-18]. Известны работы в которых исследуются данные методы, в том числе и с целью обеспечения требуемой безопасности доверенного ПО [19]. Анализ существующих результатов исследований [24, 25] показывает, что предлагаемые методы, по большей части, работают с двоичным кодом и нацелены на изменение только определенной части ПО – декриптора, либо используют виртуализацию – виртуальную машину некоторого несуществующего процессора и осуществляют преобразование кода в код данного процессора с последующим выполнением на виртуальной машине.. В данной работе рассмотрена технология полиморфного изменения программного кода, основывающаяся на преобразованиях абстрактного синтаксического дерева с помощью алгоритмов запутывания кода (обфускации) и транспайлеров (интерпретаторов, преобразующих код программы, написанной на одном языке программирования (ЯП), в аналогичный код другого языка программирования). Технология реализуема с помощью автоматизированных средств для популярных компилируемых и интерпретируемых языков программирования и позволяет повысить выживаемость доверенного ПО в условиях использования в недоверенных средах. Разработанные прототипы автоматизированных средств преобразования были протестированы с использованием автоматизированных средств анализа ПО - антивирусных средств.. . 1. Обзор средств получения абстрактного синтаксического дерева. В рамках исследований необходимо получить различные способы построения запутывающих преобразований, в основе которых лежит принцип полиморфной модификации абстрактного синтаксического дерева (АСД). Объектом исследования являются различные средства построения АСД для выбранных языков программирования. Под абстрактным синтаксическим деревом понимают конечное размеченное ориентированное дерево, в котором внутренние вершины сопоставлены (помечены) с операторами языка программирования, а листья с соответствующими операндами. Листья являются пустыми операторами и представляют только переменные и константы [23].. Для большинства современных языков программирования (ЯП) уже реализованы средства получения АСД из исходного кода программ. В данной работе для выбранных ЯП были рассмотрены средства построения АСД, указанные в табл. 1.. . Таблица 1 – Языки программирования и средства построения АСД. Название ЯП. Средства построения АСД. JavaScript. Esprima, SpiderMonkey, UglifyJS, Traceur, Acorn. Python. Python ast module, RedBaron, Astor, Astcheck, AST Optimizer. PHP. Astkit, PhpParser. PowerShell. . System.Management.Automation, Language.Parser, Metaspec, Roslyn, srcML, NRefactory. Java. Javaparser, Spoon, JDT, srcML, Javac api, Javapoet. . В результате сравнительного анализа средств получения АСД для интерпретируемых языков были выбраны инструменты, которые предоставляют необходимую функциональность и максимально покрывают язык программирования, что оказывает первичное влияние на скорость и качество исследований.. Для компилируемых языков программирования рассматривалась компиляторная инфраструктура LLVM/Clang [26].. Использование внешнего инструмент построения АСД обусловлено тем, что создание собственного инструмента является крайне трудоемкой задачей и не имело практического смысла в данной работе.. . 2. Обзор методов обфускации. Обфускация программы есть всякое ее преобразование, которое сохраняет вычисляемую программой функцию (эквивалентное преобразование), но при этом придает такую форму, что извлечение из программного кода ключевой информации становится трудоемкой задачей [20]. В рамках исследования методов обфускации рассматривалась как базовая функциональность, необходимая для разнообразия получаемых вариантов программного кода, а также методы затруднения анализа программного обеспечения. По результатам анализа различных отечественных и зарубежных работ [5, 19, 20, 21, 22] были выбраны такие методы обфускации, которые могут быть наиболее эффективными для достижения поставленной цели. Запутывающие преобразования можно разделить на группы в зависимости от того, на трансформацию какой из компонент программы они нацелены [23].. Преобразования форматирования. Изменяют только внешний вид программы. К этой группе относятся преобразования удаляющие комментарии, отступы в тексте программы, меняющие имена идентификаторов. Данные методы направлены только на усложнение анализа кода программы человеком. Отсутствие комментариев не влияет существенно на анализ. Форматирование текста программы в удобный для восприятия вид может быть легко проведено вручную или с помощью автоматизированных средств. Имена идентификаторов так же могут быть изменены в процессе деобфускации на удобные и значимые для исследователя, а автоматизированные средства делают собственные именования.. Преобразования структур данных. Изменяют структуры данных с которыми работает программа. Преобразования структур данных можно разделить на классы в зависимости от целевого объекта: обфускация хранения, обфускация соединения, обфускация переупорядочивания. Обфускация хранения заключается в трансформации хранилищ данных, а также самих типов данных (разделение переменных, конвертирование статичных данных в процедуры, локализация переменных, расширение области действия переменных). Обфускация соединения заключается в соединении независимых данных или разделении зависимых – разделение массива, объединение массива, увеличение размерности, уменьшение размерности, факторизация классов. Обфускация переупорядочивания заключается в изменении последовательности объявления переменных, внутреннего расположения хранилищ данных, а также переупорядочивания методов, массивов, классов, определенных полей в структурах и т.п.. Преобразование потока управления. Главным образом предназначено для изменения структуры графа потока управления одной функции или программы в целом. Использование методов данного класса может приводить к созданию в программе новых функций, методов и процедур. Основная задача преобразований этого класса состоит в запутывании последовательности исполнения программного кода, что в значительной степени усложняет понимание реализованных алгоритмов. Преобразование потока управления можно разделить на классы: вычислительная обфускация, обфускация соединения, обфускация переупорядочивания. Вычислительная обфускация это группа методов, приводящих к изменению структуры графа потока управления, например, добавление недостижимого кода, расширение и добавление новых условий циклов, условных операторов, добавление избыточных операций. В группу методов обфускации соединения входят методы, производящие объединение или разделение фрагментов кода для того, чтобы убрать логические связи между ними (вставка функций, выделение функций, переплетение функций, клонирование функций, развертка циклов, разложение циклов). Методы группы обфускации переупорядочивания влияют на переупорядочивание блоков, циклов, выражений (диспетчеризация, переупорядочивание операторов и выражений).. Превентивные преобразования. Данные преобразования предназначены для противодействия определенным методам декомпиляции программ [23], в некоторых случаях могут использовать недостатки или особенности работы определенных инструментальных средств анализа кода и декомпиляции.. Для компилируемых языков были рассмотрены некоторые коммерческие и исследовательские реализации запутывающих компиляторов (табл. 2) [27, 32, 33, 34]. Среди существующих проектов запутывающих компиляторов для компилируемых языков можно выделить два основных типа: первые предоставляют возможность запутывания программ с применением широкого набора разных взаимодополняющих методов, а другие разработаны для реализации и проверки одного конкретного метода [27].. . Таблица 2 – Запутывающие компиляторы. Наименование проекта. Реализованные методы обфускации. Исследовательский проект HES-SO/RCSO «Obfuscator – LLVM». замена инструкций, вставка ложных ветвлений, вставка непрозрачных предикатов, диспетчеризация. Исследовательский проект «confuse». обфускация строк, вставка избыточного кода, запутывание графа потока управления. Исследовательский проект ИСП РАН «Обфускатор». . перенос локальных переменных в глобальную область видимости, шифрование строк, вставка фиктивных циклов, диспетчеризация, переплетение функций, сокрытие вызовов функций, разбиение констант, клонирование функций, формирование непрозрачных предикатов. Коммерческий проект C/C++ «Obfuscator». замена инструкций, обфускация строк, обфускация числовых констант, переименование файлов и каталогов с исходным кодом, преобразования форматирования. . Сгруппировав по рассматриваемым языкам программирования, были получены следующие распределения методов, указанные в табл. 3, при этом для языков, поддерживаемых компиляторной инфраструктурой LLVM, распределение происходило по используемым операционным системам.. . Таблица 3 – Группы методов для рассматриваемых языков программирования. . Язык программирования. Группа запутывающего преобразования. Наименование метода. . . Powershell(C#). Преобразование форматирования. Скремблирование идентификаторов. Преобразование потока управления. Непрозрачные предикаты, вставка функций, диспетчеризация, выделения функций, вставка мертвого кода, переплетение функций. Преобразование структур данных. Кодирование. . . PHP. Преобразование форматирования. Скремблирование идентификаторов, преобразования функций. Преобразование потока управления. Диспетчеризация. Превентивное преобразование. Транспайлирование. . . JavaScript. Преобразования форматирования. Скремблирование идентификаторов. Преобразования потока управления. Вставка функций, выделение функций. Превентивное преобразование. Антиэмуляция, траспайлирование JavaScript. . . Python. Преобразование форматирование. Скремблирование идентификаторов, преобразования строк, необратимое изменение форматирования кода. Преобразование потока управления. Трансформация путем сжатия. Превентивное преобразование. Транспайлирование. . . Java. Преобразование форматирование. Скремблирование идентификаторов. Преобразование потока управления. Непрозрачные предикаты, вставка функций, диспетчеризация, выделение функций, переплетения функций, расширение функций. Преобразования структур данных. Кодирование, замена константных выражений. Превентивное преобразование. Транспайлирование, метод перестановок сущностей в программном коде. . . CC++. . Windows. Преобразование потока данных. Шифрование строк, замена инструкций. Преобразование потока управления. Вставка ложных ветвлений и непрозрачных предикатов, диспетчеризация, сокрытие нулевых литералов. Превентивное преобразование. Антиэмуляция, сокрытие испорта исполняемых файлов. . \*nix(Linux, macOS). Преобразование потока данных. Шифрование строк, замена инструкций. Преобразование потока управления. Вставка ложных ветвлений, непрозрачных предикатов, диспетчеризация, сокрытие нулевых литералов, разбиение побитовых операций, сокрытие операции xor. Превентивное преобразование. Сокрытие импорта исполняемых файлов(Linux). . Рассмотренные классы и методы, входящие в данные классы, представляют собой эвристические алгоритмы. Общим их недостатком является отсутствие гарантированной надежности. Данные методы обфускации довольно успешно противостоят автоматическим инструментам статического анализа и декомпиляции программных кодов, однако в случае применения методов динамического анализа программ и привлечения квалифицированных экспертов в области системного программирования, стойкости данных методов обфускации программ оказывается недостаточно. Таким образом, разработанные к настоящему времени практические методы и алгоритмы обфускации программ способны затруднить понимание и модификацию программ, но не способны в полной мере решить задачу защиты доверенного ПО. В результате проведенных исследований был сделан вывод о необходимости разработки специфических методов наряду с рассмотренными.. . 3. Описание технологии. При исследовании методов обфускации рассматривались также возможности классических транспайлеров (тип компилятора, который использует исходный код программы, написанной на одном языке программирования, в качестве исходных данных и производит эквивалентный исходный код на другом языке программирования), которые особенно хорошо реализуются для интерпретируемых языков программирования. На сегодняшний день для большинства высокоуровневых языков программирования существуют средства транспайлирования, которые используются для различных целей и задач [28]. На рисунке 1 представлена схема классического механизма работы транспайлера.. . . Рисунок 1 – Механизм работы транспайлера.. . При использовании классического механизма транспайлирования для решения прикладных задач необходимо учитывать некоторое особенности, ограничивающие область применения: информационная среда, разрешение зависимостей, проблему полного покрытия кода. Стоит заметить, что так как транспайлирование это перевод среды исполнения целевого языка программирования в среду исполнения другого языка программирования, само по себе это не позволяет добиться повышения выживаемости доверенного программного обеспечения в недоверенной среде и не решает проблем, выявленных для классических методов обфускации. Для эффективного использования средств транспайлирования в целях обфускации программного кода необходимо произвести соответствующие доработки.. Предлагаемый способ обфускации с использованием транспайлирования заключается в разделении сред исполнения для определенных сущностей программного кода(рис 2). Благодаря этому при стороннем анализе ПО необходимо будет исследовать несколько сред исполнения, которые могут значительно отличаться друг друга ввиду различий языков программирования, что в итоге помогает повысить выживаемость доверенного ПО недоверенной среде. Блоки транслируемых фрагментов могут варьироваться различным образом, а организация сообщений, передачи результата может реализовываться в виде трамплинов, асинхронной модели, передачи событий, сообщений. Важно заметить, что также можно разделять среды физически, организовав подобие клиент-серверного общения, а на каждый получившийся фрагмент кода можно использовать результаты обфусцирующих преобразований от классических обфускаторов.. . . Рисунок 2 - Обфускация с использованием транспайлирования и разделения сред. . Данный подход сильно повышает вероятность выживания в недоверенной среде, затрудняет ручной анализ и анализ с помощью применения автоматизированных средств. Наиболее близким к данному техническому решению подход — это использование виртуальной машины, однако он обладает следующими недостатками: восстановление исходного фрагмента после полного анализа и обратная трансляция в систему команд целевой архитектуры, широкий спектр инструментов для анализа и декомпиляции, ограничение применимости только для компилируемых языков.. В рамках исследования было установлено, что патентованные решения или аналоги данной технологии отсутствуют. Ключевой особенностью, отличающей предлагаемый подход, является использование технологии транспайлирования как средства запутывающих преобразований, а новизна данного подхода состоит в том, что, после трансляции части программы в другой язык программирования, обеспечивается сохранение связей между компонентами для их взаимодействия во время исполнения программы. Также предусматривается разделение сред исполнения между преобразованными и не преобразованными частями программного кода.. Для компилируемых языков также были рассмотрены методы противодействия автоматизированным средствам, использующим технику эмуляции программного кода, когда эмулируется API используемый в ПО. Для повышения устойчивости к такому виду анализа было проведено исследование, в ходе которого был составлен список наиболее часто эмулируемых функций и выявлены ограничения, которые существуют у эмуляторов, используемых в автоматизированных средствах анализа. Установлено, что практически все эмуляторы имеют два существенных недостатка – время имитации команды и сложность правильного имитирования всевозможных конструкций и функций ОС. Исходя из этих недостатков были разработаны методы, которые помогают обнаружить факт работы доверенного ПО в режиме эмуляции.. . 4. Результаты. На этапе проверки технологии была произведена аналитическая оценка эффективности методов запутывающих преобразований с использованием метрических характеристик, описывающих сложность кода программы [30]. Метрические характеристики сложности кода программы ставят в соответствие каждой программе некоторое численное значение, а запутывающие преобразования должны приводить к существенному увеличению численного показателя данных характеристик [29]. Рассматривались методы в отдельности, а также их различные комбинации, что позволило выявить группы методов, которые максимизируют эффект. Причем максимизация данного эффекта показывала также оптимальный порядок методов, когда взаимное влияние усиливалось, а негативные влияния нивелировались. Перечень метрических характеристик сложности программ был выбран следующий: ABC-метрика, метрики Холстеда, метрика Пивоварского, цикломатическая сложность. Новые значения метрических характеристик сравнивались после преобразования и до применения обфускации.. В результате исследования было выявлено, что, в зависимости от последовательности применения запутывающих преобразований, значение характеристики либо уменьшается, либо увеличивается:. ﹣ преобразования форматирования: являются не только тривиальными с точки зрения их реализации и преодоления, но и не представляют интереса в контексте исследования метрических характеристик. Запутывающие преобразования «Скремблирование идентификаторов» не влияют на значение характеристик. Оказывают влияние лишь на значение таких мер, как среднее количество строк, количество пустых строк, количество или процент комментариев, оценка стилистики. Сложность анализа программного кода данные преобразования не изменяют. Данные преобразования не рассматривались для исследования метрических характеристик.. ﹣ преобразование структур данных меняют размещение в памяти, тип кодирования, агрегацию или порядок элементов в объектах данных. Данные преобразования влияют на значение количественных характеристик непредсказуемым образом, так как заранее неизвестно, в какой очередности должно быть применено преобразование структур данных для увеличения или уменьшения характеристик сложности кода.. ﹣ преобразования потока управления в отличие от преобразований структур данных, способны сильно увеличить сложность программного кода. Соответственно, учитывающие эти преобразования характеристики сложности являются более актуальными в контексте рассматриваемой задачи. Преобразование «Вставка функций» увеличивает все характеристики сложности потока управления, основанные на цикломатической сложности. Наоборот, преобразование «Выделение функции» уменьшает данные характеристики. Преобразование «Переплетение функций» увеличивает как характеристики размера и сложности управления, так и сложность данных (за счет объединения списков параметров). Следует отметить, что рассмотренное преобразование может возникать и в качестве побочного эффекта замены инструкции возврата из функции инструкцией перехода – тогда точка выхода из функции не всегда может быть распознана корректно и часть кода вызывающей функции ошибочно относится к вызываемой. «Диспетчеризация» эффективно искажает поток управления. «Непрозрачные предикаты» увеличивает цикломатическую сложность программы и все меры, использующие эту величину. Данная группа преобразований показала наибольшее значение метрик, при комбинации с этими преобразованиями значение метрических характеристик всегда увеличивается. Преобразования «Вставка функций», «Вставка непрозрачных предикатов» и «Внесение мертвого кода» увеличивали значение метрических показателей в конце последовательности. Преобразования «Введение диспетчера» и «Выделение функций» уменьшают значение метрических характеристик в начале последовательности. Целесообразно применять данные преобразования в конце последовательности обфускации.. ﹣ превентивное преобразование является одним из наиболее эффективных способов обфускации. Оценка этого способа с помощью известных метрических характеристик представляется затруднительной, так как зависит от сочетания различных факторов.. ﹣ транспайлирование фрагментов кода увеличивает трудоемкость анализа кода в целом, но при этом может уменьшать значение характеристики сложности потока управления, например, транспайлирование фрагмента кода большой и сложной функции уменьшает ее количественные характеристики, а также характеристики сложности потока управления. На практике, однако, данный метод обфускации является не только эффективным, но и очень затратным.. Необходимо отметить, что не существует универсальной метрической характеристики сложности программного кода среди известных. Любые рассмотренные метрические характеристики анализируемых программ должны использоваться совместно. Дальнейшие исследования метрических характеристик сложности обфусцированных программ должны быть направлены на проведение серии экспериментов с вычислением простых характеристик, формирование шкал сложности и трудоемкости, а также построением профилей анализируемых приложений и трасс на основе пригодных для этого характеристик. Результатом таких экспериментов будет уточнение набора элементарных метрических характеристик, позволяющих автоматизировать поиск защитных механизмов в программе.. Для экспериментальной оценки был рассмотрен следующий сценарий: многие методы, которые могут быть использованы для повышения выживаемости доверенного ПО, также зачастую используются в программах, классифицируемых как компьютерные вирусы. По сложившейся практике, в исследовательских целях проведение экспериментов с использованием такого класса программ может быть вполне оправдано [19], для получения экспериментальной оценки может быть использована следующая схема: имеется некая программа, которая классифицируется как вредоносная, она легко доступна в открытом доступе, легко обнаруживается средствами автоматизированного анализа. В качестве средства автоматизированного анализа может выступать антивирусное ПО как класс средств защиты, нацеленных на нарушение работы такого класса ПО и эксперимент сводится к изначальной постановке задачи: имеется ПО и среда функционирования, которая может нарушить работу данного ПО. Для исследования были использованы такие экземпляры вредоносного ПО, которые обнаруживаются средствами защиты (по данным ресурса Virustotal [31]) и легко доступны – исходный код размещен на популярных площадках публикаций исходного кода. Антивирусное программное обеспечение для проведения эксперимента выбиралось из списка доступных контрибьюторов (производителей предоставляющих свои разработки) на сервисе Virustotal, с учетом известности их производителя: AVG Antivirus Free, Avira Antivirus Free, BitDefender Internet Security, COMODO Antivirus, G Data Antivirus, ESET NOD32 Antivirus, Kaspersky Antivirus, McAfee Antivirus, Norton Security, Trend Micro Antivirus Plus. Для нивелирования возможных последствий со стороны выбранных экземпляров и средств защиты, была построена аналогия платформы Virustotal в изолированной сети без доступа в сеть Интернет.. При проведении экспериментов ставилась задача показать, что средства защиты, выступая как враждебная среда, не смогут существенно нарушить, либо прекратить вообще функционирование экземпляра ПО после применения разработанной технологии, что позволит говорить об успешном получении экспериментальной оценки эффективности. Следует отметить, что для вредоносного ПО была исключена возможность потенциально деструктивного действия, оно было заменено на функцию записи в файл, что не оказывает значительного влияния на проведение экспериментов и получения экспериментальной оценки. В ходе экспериментов для каждого тестового экземпляра вредоносного ПО проводились следующие манипуляции в серии из трех экспериментов:. При срабатывании антивирусного средства на шаге 4 проводилась повторная модификация экземпляра с новыми параметрами настройки прототипа технологии. Испытание считалось не пройденным в том случае, когда все модификации в серии из трех попыток были обнаружены антивирусным средством аналогично шагу 2. Под настройкой прототипа понимается выбор конкретных методов и их специфичных параметров, глубина обработки данных. Было определено, что антивирусное средство может производить детектирование основываясь на метаданных ПО, содержащих служебную информацию – так называемый манифест ПО. Манифест подробно описывает процесс получения ПО из исходного кода (сборку), содержит уникальный идентификатор, данные об экспортируемых компонентах, внешних ссылках. Прототип технологии не обрабатывает данную информацию, что является основной причиной недостижения значений экспериментальной оценки прототипа до 100% для всех серий экспериментов. Общее количество экспериментов по всем экземплярам ПО, ЯП составило около 250.. В рамках экспериментов было получено, что более 90% процентов вариаций, которые получаются после применения технологии, показывают положительный результат и задача повышения надежности функционирования программного обеспечения выполнена. Недоверенная среда функционирования не может существенно нарушить, либо прекратить функционирование экземпляра ПО, а функциональные тесты при этом показали, что технология не нарушает исходную функциональность экземпляров.. . Заключение. В ходе теоретических и экспериментальных исследований по разработке технологии полиморфной модификации программного кода были исследованы и выявлены преимущества и недостатки существующих решений. Были выбраны наиболее подходящие технические и программные средства для реализации требований предъявляемых к защите доверенного ПО. Разработан новый подход к обфускации программного кода, который в значительной степени ограничивает возможность понимания алгоритмов функционирования программного обеспечения. Разработанная технология позволяет повысить надежность функционирования доверенного программного обеспечения, используемого в недоверенной среде исполнения, в том числе при угрозах возможного анализа автоматизированными средствами. Результаты исследования могут применяться для защиты программных компонентов критически важных инфраструктур как от непреднамеренного воздействия со стороны программного окружения, так и от преднамеренного со стороны вредоносного ПО. Дальнейшее развитие может быть направлено на создание многоуровневых запутывающих преобразований с учетом метаинформации для целевого языка программирования.