На правах рукописи

Лубкин Иван Александрович

МЕТОД СНИЖЕНИЯ ПОДВЕРЖЕННОСТИ ПРИЛОЖЕНИЙ К РЕАЛИЗАЦИИ УЯЗВИМОСТЕЙ ЗА СЧЕТ ОБФУСКАЦИИ МАШИННОГО КОДА

Специальность: 2.3.6 − методы и системы защиты информации, информационная безопасность

АВТОРЕФЕРАТ

диссертации на соискание ученой степени кандидата технических наук

Красноярск – 2023

Работа выполнена в Федеральном государственном бюджетном образовательном учреждении высшего образования «Сибирский государственный университет науки и технологий имени академика М.Ф. Решетнева» (СибГУ им. ак. М.Ф. Решетнева)

Научный руководитель: кандидат технических наук, доцент
Золотарев Вячеслав Владимирович

Официальные оппоненты: Максимова Елена Александровна,
доктор технических наук, доцент,
МИРЭА – Российский технологический университет, доцент кафедры КБ-2 «Прикладные информационные технологии»

Абрамов Евгений Сергеевич,
кандидат технических наук, доцент,
Южный федеральный университет,
заведующий кафедрой БИТ ИКТИБ

Ведущая организация: Новосибирский государственный технический университет

Защита состоится «21» декабря 2023 г., в ___ часов на заседании диссертационного совета Д 24.2.415.04 при Томском государственном университете систем управления и радиоэлектроники (ТУСУР) по адресу: 634050, г. Томск, пр. Ленина, 40, ауд. 201.

С диссертацией можно ознакомиться в библиотеке ТУСУР по адресу: 634045, г. Томск, ул. Красноармейская, 146, а также на сайте ТУСУР:

https://postgraduate.tusur.ru/urls/7k78iv8t

Автореферат разослан «___» _____ 20__ г.

Ученый секретарь диссертационного совета — Костюченко Евгений Юрьевич

Общая характеристика работы

Актуальность темы исследования. Значимым аспектом информационной безопасности являются уязвимости программных средств. Среди них одним из наиболее опасных классов являются уязвимости удаленного исполнения кода. Они позволяют атакующему выполнить необходимый ему алгоритм в контексте уязвимого приложения. Это создает предпосылки для нарушения конфиденциальности, целостности и доступности защищаемой информации, а также компрометации вычислительной системы для дальнейших атак на инфраструктуру организации. Высокая оценка опасности рассматриваемых уязвимостей связана с возможностью проводить атаки, находясь за периметром информационной системы, реализуя угрозы нарушения информационной безопасности в открытых компьютерных сетях, включая Интернет.

Выбор AMD64 в качестве рассматриваемой архитектуры процессоров позволяет сделать предлагаемые решения применимыми к широкой области использования. Реализуемая в таких ЭВМ Гарвардская архитектура не позволяет атакующему передать в программу набор машинных команд, выполняющих необходимый ему алгоритм для удаленного исполнения кода. В такой ситуации злоумышленник может попытаться «собрать» необходимый ему алгоритм из фрагментов текущей программы. Такие фрагменты получили наименование «гаджетов», а сам подход – возвратно-ориентированного программирования (далее – *RoP*). Для передачи управления между фрагментами используются определяемые атакующим данные. Такой подход получил наименование атак повторного использования.

В рамках рассматриваемой архитектуры и использующих ее операционных систем (далее – ОС) разработаны и применяются различные технологии защит от *RoP*-уязвимостей, такие как, например: *ASLR*, контроль целостности метаданных кучи, различные системы защиты, интегрируемые в ПО. В целом, количество средств защиты можно оценить как превышающее три десятка. Это свидетельствует об актуальности данного направления, но вместе с тем и отсутствии универсальных программных средств защиты.

Для существующих систем обычно не предусмотрено способов повышения защищенности без глубокой модернизации или перекомпиляции программ. В качестве ответных мер развиваются способы нападения, такие как различные виды перехвата управления, раскрытие информации о содержимом памяти программ, прицельное противодействие конкретным системам защиты. В совокупности это свидетельствует об отсутствии решения описанных проблем и актуальности проведения исследований и создания системы защиты, учитывающей слабые места существующих подходов. Чтобы быть эффективным, предлагаемый метод должен:

- снижать производительность на уровне существующих аналогов;
- для отсутствия сужения области применимости не требовать наличия исходных кодов защищаемых приложений;
- обеспечивать отсутствие искажения логики защищаемого приложения;
- снижать подверженность приложений к *RoP*-атакам за счет устранения гаджетов;
- быть устойчивым к раскрытию информации о содержимом исполнимой памяти защищаемой программы.

Выполнение указанных требований усложняет или делает невозможным построение цепочки операций, приводящей к эксплуатации уязвимости. Вследствие этого происходит объективное снижение возможности проведения *RoP*-атак. Указанные защитные преобразования допустимо назвать обфусцирующими, так как они, с одной стороны, сохраняют функциональные характеристики программ, а с другой стороны, делают невозможным или чрезвычайно трудоемким достижение атакующим его целей. Это выражается в затруднении получения в ходе анализа программ участков, пригодных для проведения атак.

Предлагаемая методика не требует обязательного наличия исходных текстов для повышения защищенности эксплуатируемых программ. Это является ценным в ситуации, когда в эксплуатируемом ПО выявлена уязвимость, но устранить ее перекомпиляцией нет возможности.

Степень проработки темы исследования. Проблемой защиты от *RoP*-уязвимостей активно занимаются в следующих университетах: Флориды (Д. Сливан, О. Аэриас, Д. Генс, Л. Дэви), Дармштадта (Т. Фрасетто), Джорджии (К. Лу, В. Ли), Саар (С. Нюрнберг, М. Бэйкс), Колумбийском университете (Д. Вильямс-Кинг), в подразделениях, занимающихся исследованиями и разработками корпораций *Microsoft Research, Intel* (Д. Гупта, Р. Сэйтэ).

В РФ теоретическими вопросами противодействия уязвимостям и созданием средств защиты занимается ИСП РАН (А.Р. Нурмухаметов, Ш.Ф. Курмангалеев). Проблема обфусцирующих преобразований программного обеспечения активно изучается на базе Томского государственного университета систем управления и радиоэлектроники (А.А. Шелупанов).

На сегодняшний день авторы применяют следующие методы защиты:

- размещение участков программ случайным для атакующего образом;
- снижение числа пригодных для атаки гаджетов;
- затруднение получения контроля над графом потока управления (ГПУ).

При этом подавляющее большинство авторов не рассматривают вопрос защиты приложений, которые находятся в эксплуатации и для которых отсутствует исходный код, ограничиваясь внедрением кода на этапе компиляции. Кроме того, для обеспечения безопасности авторы традиционно ставят условие – отсутствие у атакующего информации о размещении фрагментов программы. Такой подход становится уязвим после получения атакующим информации о содержимом исполнимой памяти программы.

Целью работы является обеспечение защищенности программ за счет снижения количества пригодных для реализации *RoP*-уязвимостей участков без требования наличия их исходных текстов.

Для достижения указанной цели необходимо решить **следующие задачи:**

1. Провести поиск, анализ и систематизацию публично доступных *RoP*-уязвимостей и используемых в них шаблонных участков и конструкций программ, а также систем защиты от них и подходов к оценке их эффективности.

2. Разработать формальную модель вычисления выходных данных программ, позволяющую сформировать критерий их неразличимости до и после встраивания средств защиты.

3. Разработать алгоритм резервирования мест для встраивания кода средств защиты, обеспечивающий сохранение логики работы защищаемого приложения без необходимости наличия его исходных текстов.

4. Разработать алгоритм выполнения замен фрагментов кода программы, неразличимых с точки зрения результатов работы, которые могут быть использованы для проведения *RoP*-атак, использующий свободные ресурсы процессора.

5. Разработать алгоритм расчета метрик защищенности, позволющий определить потенциальную возможность проведения успешных атак.

6. Разработать средство защиты, реализующее предложенные алгоритмы, для проверки неразличимости оригинальных и модифицированных программ, оценки накладных расходов и определения числа пригодных для использования в рамках эксплуатации уязвимостей участков.

Объектом исследования являются программные модули, скомпилированные для архитектур, для которых актуальны *RoP*-атаки.

Предметом исследования являются средства модификации программ, предназначенные для повышения защищенности приложений от *RoP*-эксплойтов.

Методы исследования. Использовались элементы теории алгоритмов, вычислимых функций Чёрча, теории компиляции, теории графов, методы компьютерной алгебры, статистики, понятия и методы теории сложности.

Достоверность работы подтверждается результатами, полученными с использованием предлагаемого в работе средства защиты, и их сопоставлением с результата-

ми других авторов, проводящих исследования в этой области, а также использованием для перекрестной проверки корректности инструментов анализа программ и средств повышения защищенности от *RoP*-атак, являющихся стандартами де-факто в отрасли.

Научная новизна состоит из предложенных в работе:

1. Модифицированного метода снижения числа пригодных для проведения RoP-атак участков в программах, отличающегося от аналогов встраиванием кода системы защиты в программные модули без требования наличия исходных текстов и с обеспечением неразличимости алгоритма защищенной и оригинальной программы.
2. Модифицированной методики контроля целостности графа потока управления, отличающейся от аналогов использованием псевдослучайных значений для контроля целостности адресов возврата и защитой фреймов стека вызывающих подпрограмм.
3. Модифицированного метода оценки эффективности систем защиты программ от *RoP*-атак, отличающегося от аналогов определением достижимости конечного состояния системы, необходимого атакующему, путем анализа номенклатуры содержащихся в программе гаджетов.

Практическая значимость работы. Разработанные в ходе работы алгоритмы и средство защиты, реализующее модифицированный метод защиты, могут быть использованы в автоматизированных системах с требованиями устойчивости к *RoP*-атакам при возможности взаимодействия атакующего с уязвимым приложением. Наиболее актуальным является проведение атаки через открытые телекоммуникационные сети (включая Интернет). Для защищаемого приложения не требуется наличие исходных текстов.

Результаты работы по повышению защищенности от *RoP*-уязвимостей были использованы для сервиса sshd на границе сетевого периметра АО «РТК-Сибирь», использованы в рамках повышения защищенности разрабатываемого ФГАОУ ВО «Сибирский федеральный университет» ПО от уязвимостей, а также внедрены в образовательный процесс ФГБОУ ВО СибГУ им. М.Ф. Решетнева

для студентов кафедры безопасности информационных технологий в рамках изучения уязвимостей и защиты от них.

Положения, выносимые на защиту:

1. Предложенные технические решения по созданию нового средства защиты, основанного на методе снижения числа пригодных для проведения *RoP*-атак участков в программах, обеспечивают для программ в условиях отсутствия их исходных текстов уменьшение числа уникальных гаджетов на 98–100 %, а гаджетов, пригодных для атак, – на 100 %.

Соответствует пункту 15 паспорта специальности 2.3.6. Принципы и решения (технические, математические, организационные и др.) по созданию новых и совершенствованию существующих средств защиты информации и обеспечения информационной безопасности.

2. Средство противодействия *RoP*-атакам, реализующее методику контроля целостности графа потока управления, обеспечивает защищенность при проведении атак через открытые компьютерные сети в условиях наличия у атакующего всей информации о программе, кроме используемых для защиты адресов возврата псевдослучайных значений.

Соответствует пункту 5 паспорта специальности 2.3.6. Методы, модели и средства (комплексы средств) противодействия угрозам нарушения информационной безопасности в открытых компьютерных сетях, включая Интернет.

3. Программная реализация предлагаемого модифицированного метода оценки эффективности средств защиты программ от RoP-атак позволяет определить техническую реализуемость уязвимостей соответствующего типа путем определения возможности атакующего задавать аргументы необходимых ему подпрограмм в случае перехвата контроля над ГПУ.

Соответствует пункту 11 паспорта специальности 2.3.6. Модели и методы оценки эффективности систем (комплексов), средств и мер обеспечения информационной безопасности объектов защиты.

Личный вклад. Все результаты, изложенные в диссертации, получены автором самостоятельно. Уточнение цели и задач, подходов к их решению и способов представления результатов выполнены автором совместно с научным руководителем. Программная реализация предложенных в работе алгоритмов в виде средства противодействия *RoP*-атакам также выполнена автором самостоятельно.

Апробация результатов работы. Результаты работы докладывались в 2021 году на семинаре кафедры БИТ СибГУ им. М.Ф. Решетнева и на семинаре кафедры КИБЭВС Томского государственного университета систем управления, а также на следующих конференциях:

XXV Международная научная конференция «Решетневские чтения». Красноярск, 10–12 ноября 2021 г.

2021 IEEE International Conference «Quality Management, Transport and Information Security, Information Technologies» (IT&QM&IS).

XII International scientific and technical conference "Dynamics of Systems, Mechanisms and Machines" (Dynamics), 13–15 November 2018, Omsk, Russia.

XX Международная научная конференция «Решетневские чтения». Красноярск, 1–13 ноября 2016 г.

VII Всероссийский конкурс-конференция студентов и аспирантов по информационной безопасности SibInfo – 2007. Томск, 17–18 апреля 2007.

По теме работы было получено **2 свидетельства** на регистрацию программы для ЭВМ. Работа по теме диссертации проводилась в рамках гранта Минобрнауки России № 21/2020 на 2020–2021 гг. «Метод снижения подверженности приложений к реализации уязвимостей за счет обфускации машинного кода». Грант выполнялся автором единолично. Работа выполнена при финансовой поддержке Министерства науки и высшего образования РФ в рамках базовой части государственного задания ТУСУРа на 2023–2025 гг. (проект № FEWM-2023-0015).

Публикации. Основные результаты диссертации изложены в 14 печатных изданиях, 7 из которых изданы в журналах, рекомендованных ВАК [1–7]. Публикации [8–10] входят в международную систему цитирования *Scopus*.

Объем и структура работы. Диссертация содержит: введение, 5 глав, заключение, список литературы (104 наименования) и 9 приложений. Общий объем диссертации составляет 179 страниц, включающих в себя 13 таблиц и 11 рисунков.

Основные положения работы

Во введении обосновывается актуальность темы диссертации, формулируются цель и задачи исследования, обсуждаются научная новизна и практическая ценность выносимых на защиту результатов.

Первая глава посвящена проблеме возникновения *RoP*-уязвимостей в машинном коде и существующим подходам по защите от них. В главе проанализирована выборка из уязвимостей соответствующего типа, показано отсутствие универсального решения данной проблемы даже с появлением технологии аппаратной защиты *CET*, разработанной Intel. Для выборки уязвимостей проанализированы состав используемых в эксплойтах гаджетов, статистика инструкций входящих в их вычислительную часть и виды управляющих инструкций.

Далее в главе рассмотрены существующие подходы к защите для определения их слабых мест и ограничений применимости, которые должны быть учтены при формировании требований к разрабатываемому решению. Используются подходы:

1) рандомизации размещения, безопасность которого основана на неизвестности атакующему информации о содержимом исполнимой памяти;

2) снижения числа пригодных для атаки гаджетов на этапе компиляции;

3) затруднения получения контроля над ГПУ, выполняемого на этапе компиляции.

Из анализа сделан вывод о том, что разрабатываемый метод должен обеспечивать снижение пригодных для атаки гаджетов при отсутствии исходных текстов, для того чтобы иметь максимальную применимость для эксплуатируемых программ.

В заключение главы приводятся сведения о рассматриваемой предметной области и ограничениях. В качестве архитектуры рассматривается AMD64, операционной системы – GNU/Linux, бинарного интерфейса приложений – «System V Application

Binary Interface. AMD64 Architecture Processor Supplement». Рассматриваются компилируемые приложения, написанные на языках Си и Си++, не содержащие ассемблерные вставки и встроенные системы защиты. Также приводится информация о структурах прологов и эпилогов подпрограмм, о структуре инструкций процессора.

Во второй главе рассматривается формальная модель вычислений выходных данных программ, на основании которой строится алгоритм резервирования мест для встраивания кода средств защиты, а также порядок контроля качества выполняемых преобразований. Использование данного алгоритма в рамках предложенного модифицированного метода снижения числа пригодных для проведения RoP-атак участков в программах существенно отличает его от аналогов.

В рамках модели вычислений выходных данных исследуемая программа рассматривается как последовательность операций над ячейками совокупной памяти (СП) вычислительной системы (ОЗУ и регистрами процессора). Состояние системы определяется данными в СП и для системы может быть представлено на i-м шаге выполнения программы как массив значений в регистрах процессора $R^i = (r_0{}^i, r_1{}^i, \dots, r_{|R|-1}{}^i)$, $r_k \in E$, $0 \le k < |R|$ и ОЗУ $M^i = (m_0{}^i, m_1{}^i, \dots, m{}^i{}_{|M|-1})$, $m_l \in E$, $0 \le l < |M|$ соответственно, где $|R|$ и $|M|$ – число элементов (байтов) массивов R и M соответственно. Здесь и далее верхний индекс показывает шаг вычислений, соответствующий некоему моменту времени от запуска программы, если не оговорено иного.

С каждой конкретной ячейкой ОЗУ m_l может быть сопоставлен ее адрес a. Сопоставление задается функцией *addr*, определяемой метаданными программы и параметрами работы загрузчика. Индекс l ячейки m_l в рамках предлагаемого решения является не только математической абстракцией, но и прообразом адреса для описания модели алгоритма при модификации программы.

Разобьем всё M на множество регионов ячеек ОЗУ так, чтобы в каждом регионе если адреса двух ячеек различаются на единицу, то и индексы их отличались бы на единицу (далее – смежные). Региону ячеек ОЗУ $M_j^i = (m_j^i, m_{j+1}^i, \dots, m_{j+n_v-1}^i)$ размером $\left|M_j^i\right| = n_v$ сопоставляется численное значение v_j^i посредством функции

val: $M_j^i \to v_j^i, val\left(M_j^i\right) = \prod_{k=0}^{n_v-1} m_{j+k} \times 256^k$, где j – индекс первой ячейки в регионе памяти (здесь и далее).

В составе M могут быть выделены следующие подмножества: исполнимый код O и данные программы D. Состояние вычислений на шаге i с учетом такого разбиения может быть представлено как

$$S^i = \left(s_0^i, s_1^i, \dots, s_{|S|-1}^i\right) = \left(R^i, D^i\right).$$

Множество O содержит данные, которые в ходе штатного функционирования программы интерпретируются процессором как набор команд (операций, инструкций), которые можно разбить на два класса: детерминированные и недетерминированные. Результат выполнения детерминированных инструкций представим как $S^{i+1} = f_{j^i}(S^i)$, т.е. выполнение некоторой команды f_{j^i} с начальным адресом a_{j^i} переводит систему из некоторого состояния i в следующее, i + 1-е состояние (что отражается в изменении СП). Для недетерминированных инструкций результат их работы представлен как $S^{i+1} = \tilde{f}_{j^i}^i\left(f_{j^i}(S^i)\right) = \tilde{f}_{j^i}^i \circ f_{j^i}(S^i)$, где « ∘ » – обозначение композиции функций, а $\tilde{f}_{j^i}^i$ – фиктивная инструкция. Она отражает наступающие сторонние эффекты, а результат преобразования состояния зависит от контекста выполнения программы.

Разобьем множество O на последовательности линейно исполняющихся инструкций – базовых блоков (ББ). Выполнение программы может быть представлено ориентированным ГПУ. В нем из каждого узла исходит как минимум одно ребро. Для ББ фиктивная инструкция будет разная в зависимости от шага исполнения i, поэтому свяжем их по индексу j^i. Тогда блок таких инструкций в символьном виде обозначим как $\tilde{b}_{j^i} = \tilde{f}_{j^i}^i \circ b_j$, где b означает последовательность инструкций и соответствует ББ. Несмотря на то, что такой блок становится зависимым от шага выполнения (не только его индекс), будем опускать верхний индекс, но далее везде, где появляется такая зависимость от сторонних эффектов, будем указывать символ тильда.

Для обеспечения возможности внесения модификаций в защищаемую программу (для резервирования участков, в которые будет вставляться код системы защиты) сформулируем модель вычисления выходных данных программ. Она обеспечивает сравнение результатов работы оригинальной и защищенной программ. Если эти результаты семантически совпадают, то делается вывод об идентичности программ.

Программа в начальном состоянии может быть представлена как

$P = R^0 \cup D^0 \cup B$, где $B = (\dots, b_j, \dots), \forall j$

Результат вычислений на момент наступления шага i является композицией операций из состава ББ с учетом сторонних эффектов:

$$\tilde{S}^i = \left(\tilde{b}_{j\,i-1} \circ \tilde{b}_{j\,i-2} \circ \cdots \circ \tilde{b}_{j\,0}\right)(S^0)$$

$$\tilde{b}_{j\,t} = \tilde{f}_{(j+\left|b_{j\,t}\right|-1)^t} \circ f_{(j+\left|b_{j\,t}\right|-1)^t} \circ \cdots \circ f_{j\,t}, \forall t \in Z_+$$

Определим критерий неразличимости двух программ с точки зрения вычисленных выходных данных: две программы будут неразличимы, если результаты их работы будут неразличимы при одинаковых входных данных. Это, в свою очередь, выполняется при условии одинаковых ячеек для помещения одинаковых входных данных и одинаковыми сформированными значениями в одинаковых выходных ячейках. Здесь и далее наличие символа «′» означает модифицированную программу, его отсутствие – оригинальную программу. Запишем критерий неразличимости:

$$P \rightleftarrows P' \leftrightarrow \text{для } \forall i: S^i \rightleftarrows S'^i$$

$$S^i \rightleftarrows S'^i \leftrightarrow \forall j: s_j^i = s'^i_j, \text{если } val\left(s_j^i\right) \notin A^i \text{ и } val(s'^i_j) \notin A'^i$$

Для формулирования критерия неразличимости программ должна быть учтена специфика работы инструкций процессора. Каждая инструкция обрабатывает лишь часть S^i. Исключим из рассмотрения те ячейки состояния S^i, значения которых не участвуют в вычислении S^{i+1} и не модифицируются в ходе него. Для отражения этого явно укажем аргументы операции: $f^i_j(S^i_p)$, где $S^i_p \subset S^i$. Для обрабатываемых инструкцией подмножества СП будем добавлять нижний индекс p. Рассмотрим типы аргу-

ментов $S^i_{jp} \subset S^i_p$ с точки зрения того, содержат ли они адреса и происходит ли его разыменование, если оно обрабатывается.

Тип аргумента 1: $S^i_{jp} \subset R^i, val(S^i_{jp}) \notin A^i$. Аргумент является регистровой памятью или непосредственным операндом инструкции, в котором не содержится адресов какой-либо ячейки ОЗУ.

Тип аргумента 2: $val(S^i_{jp}) \subset A^i, \nexists m^i_k \in S^i_p : m^i_k = [val(S^i_{jp})]$ для $\forall k$. Адреса обрабатываются, но не разыменовываются.

Тип аргумента 3: пара участков ячеек R^i_{jp} и M^i_{kp}, таких, что: $val(R^i_{jp}) \in A^i, !\exists m^i_k = [val(R^i_{jp})]$. Значение обрабатываемых данных определяется в результате разыменования аргумента, содержащего адрес.

Так как программа самонеразличима с собой при одинаковых входных данных, то допустимо принять одинаковость сторонних эффектов для запусков как для одной и той же программы, так и для оригинальной с модифицированной. Также рассмотрение запуска одной и той же программы с учетом работы *ASLR* позволяет сделать вывод о том, что изменение базы загрузки (и, как следствие, изменение значений всех адресов в программе) не приводит к формированию иного семантически результата при одинаковости сторонних эффектов. На основании этого введем критерий неразличимости программ. Два некоторых состояния неразличимы ($S^i \rightleftarrows S'^i$) при условиях:

1. Для $\forall j$: если $a^i_j \rightleftarrows a'^i_j$, то $m^i_j \rightleftarrows m'^i_j$, т.е. задано новое сопоставление абсолютных адресов с конкретными ячейками памяти.

2. Пусть $\Delta v^i_j : a^i_j = a^i_{basej} + \Delta v^i_j$ – смещение относительно базы a^i_{basej}, тогда для $\forall j$: если $a^i_j \neq a'^i_j$, то $\Delta v'^i_j := a'^i_j - a'^i_{basej}$ – коррекция смещений для относительных адресов; база определяется в зависимости от инструкций, реализующих вычисление абсолютного адреса.

3. Для $\forall k$: если $v^i_k \in A^i$ и $v^i_k = a^i_j$, то $v'^i_k := a'^i_j$ – замена всех абсолютных оригинальных адресов на соответствующие им модифицированные.

4. Для $\forall k: v_k^i \notin A^i \Rightarrow v_k^i = v'^i_k$ – одинаковое содержимое семантически неразличимых ячеек СП, не содержащих адресной информации.

5. Вставка внутрь БД не производится, так как семантический анализ БД требует существенно более сложного анализа, при том что рассматривается вставка данных в пределах *O*.

Выполнение данных условий приводит к тому, что если $S^i \rightleftarrows S'^i$, то для любого из типов аргументов инструкции выполняется $f_{j\,i}(S^i) \rightleftarrows f_{j\,i}(S'^i)$. Из этого также следует одинаковость переходов в ГПУ, так как, согласно условию 3, если $v_{IP}^i \in A^i$, то $val(v_{IP}^i) \rightleftarrows val(v'^i_{IP})$, а согласно используемому в качестве условия самонеразличимости, $b_j = b_j^{'}$. Далее сформируем базу индукции. То есть необходимо проанализировать, при каких условиях $S^0 \rightleftarrows S'^0$. При этом, согласно условию самонеразличимости, $R^0 = R'^0$, из чего следует достаточность обеспечения $D^0 \rightleftarrows D'^0$.

Выполнение условия 1 критерия неразличимости достигается за счет известности перемещения участков кода. Обработка перечисленных выше ситуаций обеспечит выполнение условий 2 и 3. Отсутствие модификаций неадресных данных обеспечит выполнение условия 4. Отсутствие вставки данных в D^0 обеспечит выполнение условия 5. Следовательно, выполняется неразличимость $D^0 \rightleftarrows D'^0$. Учитывая $R^0 = R'^0$, обеспечивается база индукции $S^0 \rightleftarrows S'^0$. С учетом шага индукции $S^i \rightleftarrows S'^i \Rightarrow S^{i+1} \rightleftarrows S'^{i+1}$ можно сделать вывод о том, что при соблюдении описанных правил модификации и одинаковости $b_j = b_j^{'}$ обеспечивается $\forall \mathrm{i}: \mathrm{S^i} \rightleftarrows \mathrm{S'^i}$, а, следовательно, $\mathrm{P} \rightleftarrows \mathrm{P}^{'}$.

Таким образом, вопрос неразличимости программ сводится к вопросу анализа программы с выявлением участков, которые содержат адресную информацию, и определением границ ББ. Для этого может использоваться как отладочная информация в составе программы, так и заимствуемые средства анализа – дизассемблеры. Для контроля корректности анализа выполняется сравнение перечня ББ и перечня адре-

сов, выявленных программной реализацией алгоритма анализа и, например, дизассемблером IDA.

На основании условий неразличимости был сформирован алгоритм резервирования мест для встраивания кода средств защиты (приведен на рисунке 1), позволяющий обрабатывать программы, для которых отсутствует исходный код.

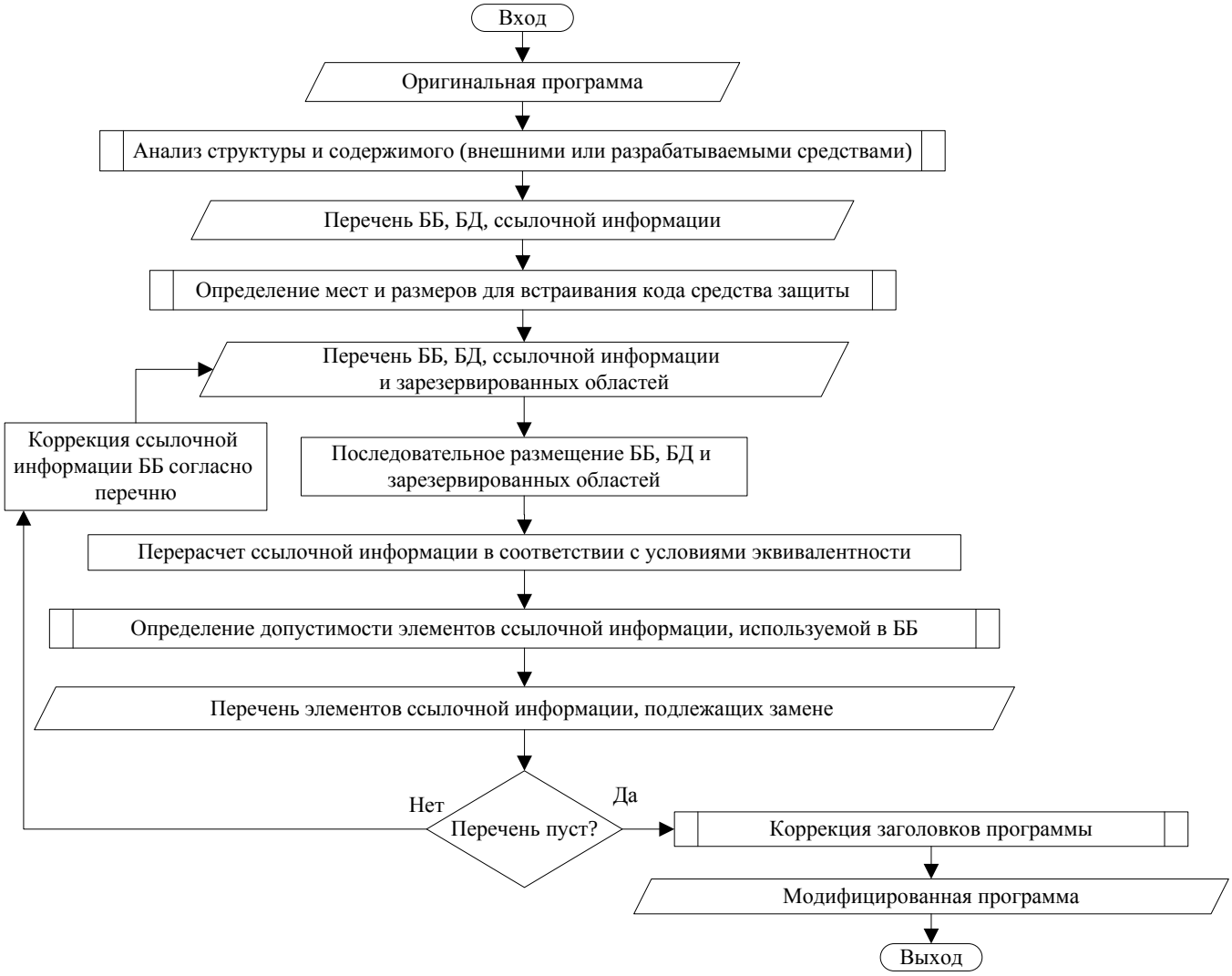

Рисунок 1. Алгоритм резервирования мест для встраивания кода средств защиты

В третьей главе приведено описание модифицированного метода снижения числа пригодных для проведения *RoP*-атак участков в программах в части применения двух его классических составляющих: защиты ГПУ и устранения гаджетов. Далее рассмотрен реализующий его алгоритм устранения опасных инструкций (ОИ) и

опасных значений (ОЗ), которые могут быть использованы в качестве гаджетов, на участках, не включающих проблемного содержимого памяти. В составе данного алгоритма используются методика контроля целостности ГПУ и алгоритм синонимической замены элементов программы, содержащих опасные значения. Алгоритм, учитывающий места размещения ОИ и ОЗ, приведен на Рисунке 2.

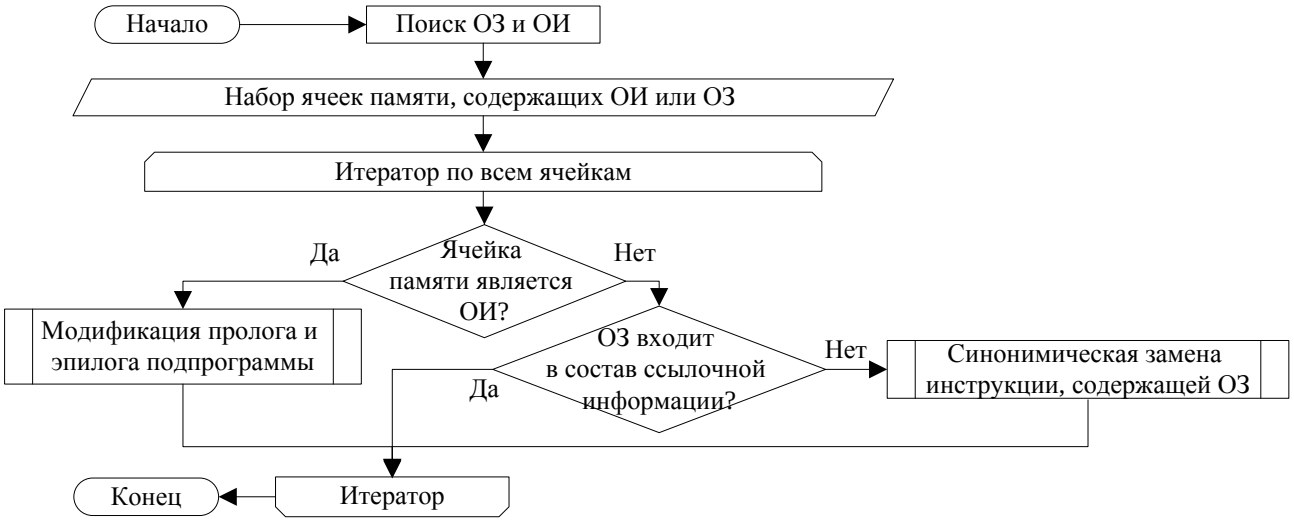

Рисунок 2. Алгоритм устранения ОИ и ОЗ

Методика контроля целостности ГПУ реализуется путем вставки кода средства защиты в зарезервированные участки в каждой подпрограмме π_d приложения. При выполнении модифицированного пролога в рамках пролога формируется случайное значение κ_d, используемое для защиты адреса возврата ρ_d (нижний индекс показывает номер фрейма стека, нумеруются от нуля, d соответствует некоторой текущей рассматриваемой подпрограмме, d-1 – вызывающей, d+1 – вызываемой из текущей). Запишем в формальном виде преобразования, которые выполняются в d и d-1 фреймах стека (для $d > 1$), кодом средства защиты, добавляемым в прологе π_d,:

$$\rho_d \rightarrow \rho_d \, XOR \, \kappa_d$$

$$\begin{cases} \kappa_{init} \rightarrow \kappa_{init} \; XOR \; \kappa_d, \text{если } d = 0 \\ \rho_{d-1} XOR \; \kappa_{d-1} \rightarrow \rho_{d-1} \; XOR \; \kappa_{d-1} XOR \; \kappa_d, \text{если } d > 1 \end{cases}$$

$$\begin{cases} \kappa_{init} \rightarrow \kappa_{init} \; XOR \; \kappa_d, \text{если } d = 0 \\ \kappa_{d-1} \rightarrow \kappa_{d-1} \; XOR \; \kappa_d, \text{если } d > 1 \end{cases}$$

При выполнении пролога π_{d+1} в фрейме d будет выполнена модификация: $\rho_d \rightarrow \rho_d \; XOR \; \kappa_d \; XOR \; \kappa_{d+1}$

В эпилогах выполняются обратные преобразования. В частности, обратная операция размещается непосредственно перед ОИ, что делает невозможным использование штатных инструкций подпрограмм в качестве гаджетов. Попытка повреждения памяти, хранящей ρ_d в защищенном виде, приведет к переходу на непредсказуемый для атакующего адрес и аварийное завершение программы.

Для устранения гаджетов вне эпилогов подпрограмм используется алгоритм синонимической замены элементов программы, приведенный на Рисунке 3. Он сформирован с учетом мест возможного вхождения ОЗ в состав инструкций.

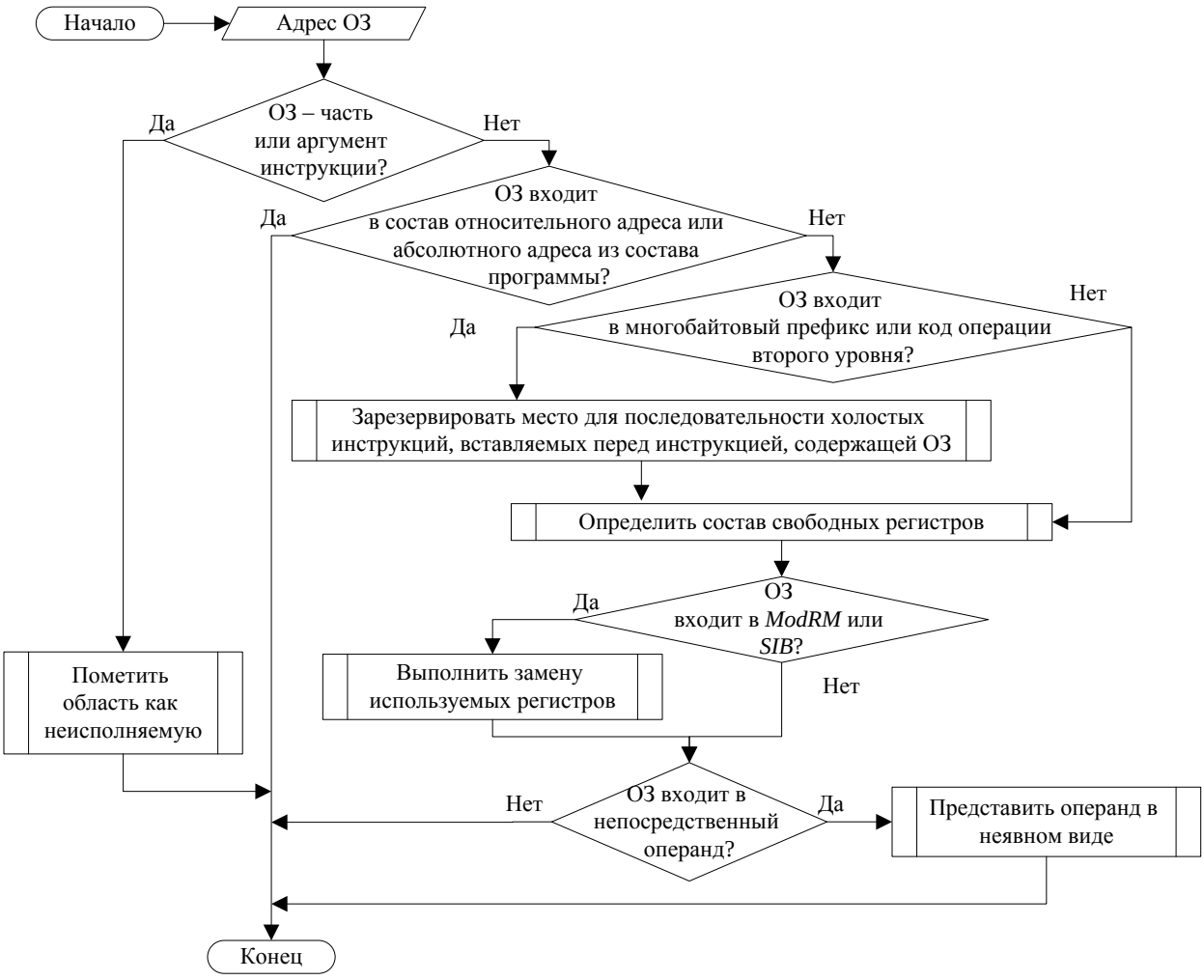

Рисунок 3. Алгоритм синонимической замены инструкций, содержащих ОЗ

В четвертой главе предложен метод для оценки эффективности защит, направленных на снижение числа гаджетов в защищаемых приложениях. Анализ существующих подходов показал, что используются: качественный анализ, подсчет числа гаджетов, применение существующих эксплойтов.

Из анализа сделан вывод, что для предлагаемой защиты необходимо определить количество пригодных для атак гаджетов с учетом их семантики. Такой подход позволяет обойтись без субъективных экспертных оценок и экстраполяции существующих уязвимостей на защищаемую программу. Для определения возможностей атакующего по проведению *RoP*-атаки используется алгоритм, приведенный на Рисунке 4. В результате определяется метрика N_π, которая при значении 0 показывает невозможность проведения атак, иначе атака потенциально возможна.

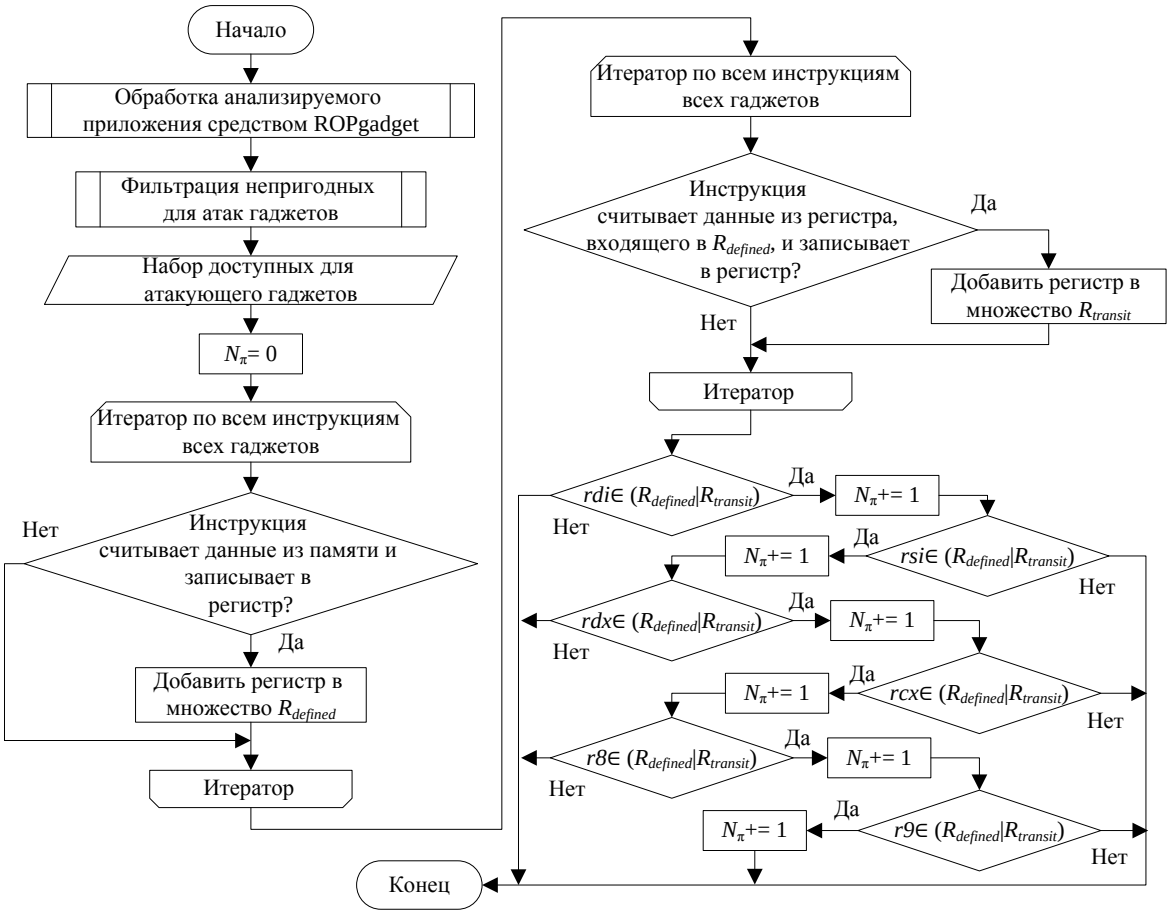

Рисунок 4. Алгоритм расчета метрики защищенности N_π

В пятой главе выполняются апробация и экспериментальная оценка корректности и эффективности предлагаемого подхода к защите. Структура разработанного комплекса приведена на рисунке 5.

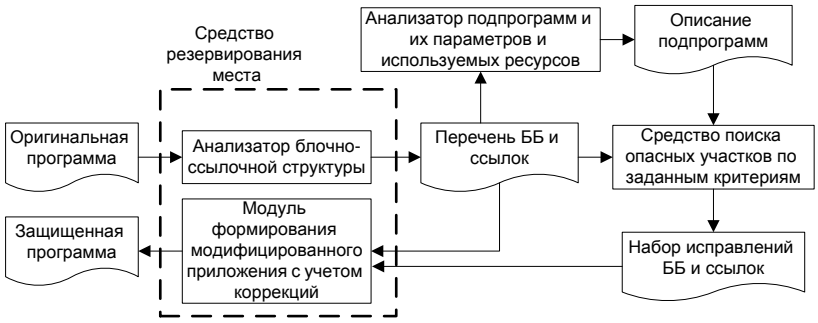

Рисунок 5. Схема взаимодействия подсистем

Для контроля отсутствия искажения алгоритма работы использовались: синтетические тесты; программы, написанные на различных языках программирования; приложениях, снабженных модульными тестами.

Отработка предложенной методики на тестовой выборке показала отсутствие искажения алгоритмов. Для сравнения производительности использовалась специализированная программа «*coremark*». Она была выбрана, так как дополнительно контролирует целостность результатов расчетов. Характеристики тестовой системы: процессор *AMD A*6-6310 1,8 ГГц, 4 ядра, 4 ГБ ОЗУ, НЖМД 7200 об/мин. Результаты определения изменения защищенности приведены в таблице 1. Замеры производительности показали падение относительно оригинальной программы: при использовании *RDRAND* – 91 %, *RDTSC* – 53 %, AESENC – 12,9 % (число запусков 100 для каждого). Сравнение производительности наиболее осмысленно со средством защиты *Shuffler*, так как оно устойчиво к атакам раскрытия содержимого памяти при условии, что атакующий не успеет провести атаку между интервалами переразмещения. Для интервала 50 мс накладные расходы составляют 15 %. Для средства существующего снижения числа гаджетов G-free падение производительности составило 35 %.

В работе приведено качественное сравнение с существующими средствами защит. Для средства снижения числа гаджетов для *RoP*-атак *G-free* приведено сравнение реализаций, показавшее, во-первых, пригодность предложенного метода оценки эффективности защит не только для описанного в работе средства защиты, а во-вторых, лучшее значение метрики N_π для разработанного средства.

Таблица 1. Полученные значения метрик защищенности до и после применения защиты

Приложение	Уник. гаджеты		Регистры, определяемые атакующим		Оперируемые регистры		Дост. арг., N_π (полностью/ частично)		Защита обеспечена
	Ориг.	Защ.	Ориг.	Защ.	Ориг.	Защ.	Ориг.	Защ.	
Синт., нет оптимиз.	387	3	rax, ch, rbx, edx, rsi, rdi, rbp, rsp, r8d, r12, r13, r14, r15	Нет	rcx, rdx, fr7	al	4/5	0	Да
Синт., ур. оптимиз. 2	118	3	eax, rbx, rsi, rdi, rbp, rsp, r12, r13, r14, r15	Нет	rax, ecx, edx	al	2/4	0	Да
coremark	232	4	rax, rbx, rsi, rdi, rbp, rsp, r8b, r12, r13, r14, r15, xmm0	bh	ecx, edx, r9d	eax	2/6	0	Да

В заключении приводятся основные результаты и выводы, полученные автором в ходе работы.

В приложениях приведены список обработанных приложений, статистический анализ частоты использования инструкций в составе гаджетов эксплойтов, детальные результаты экспериментов, свидетельства о регистрации программ для ЭВМ, акты внедрения.

Основные выводы и результаты работы

1. Проведены поиск, анализ и систематизация публично доступных *RoP*-уязвимостей, методов противодействия *RoP*-уязвимостям и их программных реализаций. В результате были выявлены проблемы, связанные с отсутствием подходов к снижению числа уязвимых участков при отсутствии возможности перекомпиляции защищаемого приложения. Анализ методов оценки защищенности, применяемых в указанных средствах, показал отсутствие объективных критериев оценки возможно-

сти проведения атак. В свою очередь, исследование реальных уязвимостей позволило сформулировать критерии участков, подлежащих защите.

2. Разработанная модель вычисления выходных данных программ позволила формально определить условия неразличимости программ, а разработанный алгоритм резервирования мест для встраивания кода средств защиты позволяет предотвращать искажение логики их работы в ходе данной операции.

3. Разработанный алгоритм синонимической замены элементов программы, содержащих опасные значения, которые могут быть использованы в составе гаджетов, и методика контроля целостности ГПУ, обеспечивающая защиту штатных инструкций возврата из состава эпилогов подпрограмм, позволяют устранять опасные участки без искажения алгоритма защищаемого приложения.

4. Проведенная экспериментальная оценка результата применения предлагаемого метода защиты показала падение производительности на 12,9 %, что находится на уровне ближайших аналогов, которые обеспечивают меньшую защищенность.

5. Проведённая экспериментальная оценка подверженности приложений рассматриваемым атакам на основании разработанных метрик защищенности показала объективное устранение возможности проведения атак за счет минимизации числа и разнообразия гаджетов.

6. Разработанное средство защиты на базе предложенных алгоритмов позволило проводить автоматизированное внедрение кода системы защиты и осуществлять оценку корректности выполненных преобразований. Результаты экспериментальной оценки подтвердили, что разработанный программный комплекс позволяет повышать защищенность приложений, написанных на языках Си и Си++, к *RoP*-атакам без требования доступности их исходных кодов.

Публикации

Статьи, опубликованные в журналах, входящих в перечень рецензируемых научных журналов и изданий, рекомендованных ВАК:

1. Лубкин И.А. Метрики защищенности приложений при использовании средств противодействия уязвимостям, основанным на возвратно-ориентированном программировании / Лубкин И.А. // Доклады ТУСУР. – 2021. – Т. 24. – № 4. – С. 46–51.

2. Лубкин И.А. Комплексная система защиты от уязвимостей, основанных на возвратно-ориентированном программировании / И.А. Лубкин, В.В. Золотарев // Информатика и автоматизация. – 2022. – № 2(21). – С. 275–310.

3. Шудрак М.О. Методика динамического анализа уязвимостей в бинарном коде / Шудрак М.О., Золотарев В.В., Лубкин И.А // Вестник Сибирского государственного аэрокосмического университета им. М.Ф. Решетнева. Красноярск. 2013. – № 4(50) – стр. 84–87.

4. Шудрак М.О. Методика и программное средство защиты кода от несанкционированного анализа / Шудрак М.О., Лубкин И.А. // Программные продукты и системы. – Тверь, 2012. – № 4 – стр. 176–180.

5. Шудрак М.О. Статический анализ бинарного кода в сфере информационной безопасности / Шудрак М.О., Лубкин И.А., Золотарев В.В. // Известия ЮФУ. Технические науки. – Таганрог, 2012. – № 12 – стр. 54–60.

6. Шудрак М.О. Методика декомпиляции бинарного кода и её применение в сфере информационной безопасности / Шудрак М.О., Лубкин И.А., Золотарев В.В. // Безопасность информационных технологий НИЯУ МИФИ. – М., 2012. – № 3 – стр.75–80.

7. Лубкин И.А. Методика защиты программного кода от несанкционированной модификации и исследования посредством его хеширования / Кукарцев А.М., Лубкин И.А. // Вестник сибирского государственного аэрокосмического университета им. академика М. Ф. Решетнева. – Красноярск, 2008. – № 1 (18) – стр. 56–60.

Статьи, индексируемые в международной базе Scopus:

8. Lubkin I.A. Automatic equivalency restoration after software patching / Lubkin I. A., Zolotarev V. V. // Proceedings of the 2021 IEEE International Conference «Quality Management, Transport and Information Security, Information Technologies» (IT&QM&IS). Yaroslavl, 2021. – стр. 217–222.

9. Lubkin I.A. Methodology of software code decomposition analysis / Lubkin I. A., Bazhenov I. O. // Dynamics of systems, mechanisms and machines. Omsk, 2018. – стр. 1–5.

10. Lubkin I.A. Technique of verified program module modification with algorithm preservation / Subboton N.A., Lubkin I.A. // 11th International IEEE scientific and technical conference "Dynamics of systems, mechanisms and machines". Omsk, 2017. – стр. 1–5.

Прочие публикации:

11. Лубкин И.А. Исследование генераторов псевдослучайных чисел, используемых для защиты от атак переполнения буфера / И.А. Лубкин // Материалы XXV Международной научной конференции «Решетневские чтения». – Красноярск, 2021.

12. Шудрак М.О. Методика декомпиляции бинарного кода и её применение в сфере информационной безопасности / Шудрак М.О., Лубкин И.А. // Материалы II Всероссийской молодежной конференции «Перспектива – 2012». Таганрог, 2012. – стр. 197–202.

13. Шудрак М.О. Методика декомпиляции бинарного кода и её применения в сфере информационной безопасности / Шудрак М.О., Лубкин И.А. // Материалы Всероссийской научно-технической конференции студентов, аспирантов и молодых ученых ТУСУР. Томск, 2012. – стр. 245–250.

14. Шудрак М.О. Защита программного обеспечения методом полиморфной генерации кода / Шудрак М.О., Лубкин И.А. // Материалы XIV Международной научной конференции «Решетневские чтения». Красноярск, 2010. – стр. 199–200.